ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Electronics and Telecommunications Engineering (31^{st} cycle)

Deep Learning for Image Analysis in Satellite and Traffic Applications

By

Sina Ghassemi

******

Supervisor(s):
Prof. Enrico Magli, Supervisor

Doctoral Examination Committee:
Prof. Gabriele Moser, Referee, Università degli studi di Genova
Prof. Farid Melgani, Referee, Università degli studi di Trento
Prof. Marco Grangetto, Università degli studi di Torino
Prof. Sophie Fosson, Politecnico di Torino
Prof. Tiziano Bianchi, Politecnico di Torino

Politecnico di Torino
2019

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Sina Ghassemi
2019

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

I would like to dedicate this thesis to my loving parents

Acknowledgements

And I would like to acknowledge that the research presented in this thesis has been supported by a fellowship from TIM and has been done at the Joint Open Lab Cognitive Computing.

Besides, I would like to thank my supervisor Prof. Enrico Magli for his support and encouragement during the projects. I would also like to thank my colleagues at the TIM joint open lab Attilio Fiandrotti, Gianluca Francini, Pedro Porto Buarque and Skjalg Lepsøy for all the help they provided me.

Abstract

In this thesis, two fundamental problems in computer vision have been addressed by proposing novel approaches which are based on deep learning. First, we address fine-grained object recognition over vehicular images regarding vehicle makes and models. Secondly, we address semantic segmentation over remotely sensed images on a global geographical scale.

To address vehicle make and model recognition (VMMR), a classification architecture based on Convolutioanl Neural Network (CNN) and multi-scale attention windows is developed. The proposed architecture consists of a localizer and a classifier module. First, the localizer module predicts a number of attention windows over each image to capture most representative parts of a vehicle. Then, the classifier module extracts and aggregates visual representations over the predicted attention windows to perform classification over vehicle make and model. We show that VMMR can benefit substantially by capturing most distinctive parts of a vehicle over attention windows with non-identical sizes which provide discriminative visual patterns over multiple scales. Moreover, the proposed architecture leverages spatial transform module to spatially manipulate the input image and to backpropagate the error from the localizer to the classifier. Thus, unlike many other competitive part-based approaches, the proposed localizer is trained to minimize the classification error without requiring expensive part annotations over training samples.

Additionally, a multi-scale patch training methodology is proposed which enables predicting attention windows with desired scales. Moreover, the classifier module is proposed with multiple outputs to allow joint prediction on vehicle make and model. Hence, we formulate a loss function accounting for classification errors over both vehicle make and model. In the end, we evaluate the proposed methodology over two publicly available datasets: Stanford car dataset [51]; Compcar dataset [105]. Our proposed architecture surpasses all prior state-of-the-art methods in both datasets.

In the second part of the thesis, semantic segmentation on satellite images is addressed proposing a CNN encoder-decoder architecture. In contrary to most of the recent work where the segmentation is studied over samples with similar distributions (i.e. samples are extracted from one geographical area), we develop a scheme which is deployable over a broad range of aerial images with different statistics with different geographical locations. Satellite images captured in different locations by different sensors or even in different time intervals experience variations in their distribution, such variations over a segmentation model input known as covariate shift in machine learning. Accordingly, in this work, we study the proposed architecture capability to reduce the performance degradation associated with covariate shift. We show that a class of CNN namely residual network that enables very deep networks (up to 200 layers), if employed as encoder module in the proposed architecture, allows learning visual representations of high semantic level which are more robust to covariate shift. Training such deep encoder over a large amount of satellite images captured at different locations enables learning features of high semantic level which are not specific to a particular image.

Additionally, we propose two domain adaptation techniques to further enhance the segmentation over each specific image. In the first method, performance is improved over each image by fine-tuning the network over a small subset of annotated samples. In the second approach, batch normalization statistics are fine-tuned over each image improving the segmentation without requiring annotations. We evaluate the proposed architectures and domain adaptation methodologies over a homegrown dataset and also two publicly available datasets of satellite images namely ISPRS Vaihingen 2D semantic segmentation contest [15] and INRIA aerial images benchmark [63]. We show that while our network benefits from less complex structure it advances state-of-the-art results on binary segmentation and competes closely with far more complex methods on multi-class segmentation task.

Finally, we propose a similar encoder-decoder CNN to address cloud screening over satellite images such that it can be implemented on the satellite platform. Thus, we investigate experimentally several solutions to make CNN more efficient in terms of resource consumption while preserving its cloud screening accuracy. We show that the proposed architecture can be implemented on the satellite platform while performing with reasonably high accuracy compared with the state-of-the-art approaches.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Image classification plays a critical role in many image analysis applications and refers to the problem of labeling an image according to a number of predefined classes. The similar problem of semantic segmentation may also be considered as an image classification task where instead of assigning a class to the whole image, each image pixel is given a label. Both image classification and semantic segmentation are essential for applications in many fields such as medical image analysis, autonomous driving, remote sensing, security systems, intelligent transportation systems and so on. In the recent decade, thanks to the rapid advancement in the area of deep learning, availability of computational resources, and large scale datasets, deep learning has achieved state-of-the-art performance in many fields including image analysis. Particularly, Convolutional Neural Networks (CNNs) have shown to be effective tools for extracting high-level visual representations from raw data. CNNs have markedly outperformed their competitive methods in many image analysis tasks including image classification and segmentation.

In the first part of the thesis, we address fine-grained image classification over vehicle makes and models by developing a novel approach based on CNN and multi-scale attention windows. In the second part, a CNN architecture along with domain adaptation techniques are proposed to address semantic segmentation problem over satellite images such that it can be applied and adapted over samples which are not present in the training set. Finally, in the third part of the thesis, we address the problem of optimizing memory consumption of a CNN such that it can be applied as

an onboard cloud screening unit on satellite platform with the purpose of detecting the images which are contaminated by clouds.

1.1 Vehicle Make and Model Recognition

Vehicle Make and Model Recognition (VMMR) refers to the problem of recognizing a vehicle according to its manufacturer company, model and sometimes manufacturing year. Compared to other object classification tasks, VMMR presents several unique challenges. First of all, vehicle is regarded as a rich object class considering the large number of vehicle makes and models. While some classes may easily be separable, many others are difficult to tell apart due to subtle differences in appearance. For example, distinguishing two vehicles such as an SUV from a hatchback may be straightforward, while recognizing two models from a same make but different models like Peugeot 207 and Peugeot 208 requires more attention. Furthermore, vehicles captured from different view points illustrate almost different parts which introduces intra-class visual variations. Such variations along with inter-class similarity demand a robust classification scheme to perform VMMR. Therefore, many classification methodologies which are proposed for coarse-grained object recognition problems may not be able to address VMMR if the aforementioned challenges related to vehicles are not addressed properly.

Fig. 1.1 Surveillance cameras in intelligent city.

1.1.1 Motivation

In recent decades, road transport in many developed countries has experienced an increasing growth associated with growing mobility requirements. However, rising energy consumption and destruction of environment are some side effects presented by such volume of transportation traffic. In order to manage these challenges, many

countries has been developing Intelligent Transport Systems (ITS). According to the European parliament directive [1], "ITS are advanced applications which without embodying intelligence as such aim to provide innovative services relating to different modes of transport and traffic management and enable various users to be better informed and make safer, more coordinated and 'smarter' use of transport networks". Therefore, ITS includes a broad range of technologies from basic management systems such as traffic signal control systems, car navigation or speed cameras to monitor applications such as security CCTV systems or even more advanced applications that combine live data and feedback from several other sources such as parking guidance systems. Nevertheless, in many ITS related applications, an important component is to identify a vehicle according to its make and model.

The traditional systems address VMMR by relying mostly on human observation which makes their deployment impractical for real-time applications and over large scale transportation networks. Moreover, the increasing deployment of traffic surveillance cameras and advancement in computer vision techniques have made a growing demand for automated visual vehicle recognition systems. A reliable automated VMMR technology can significantly enhance the performance of many applications in the field of ITS.

For instance, electronic toll collections over highways are implemented to reduce the delay and traffic by obviating the need of cash payment and consequently preventing car stop in the highways. However, most toll collections booths charge different rates for different vehicle types, hence require distinguishing different type of passing vehicles. Yet many existing methods rely on complex system that employs several sensors such as inductive sensors in the road surface and shape detective light-curtain lasers in order to classify the vehicle. Therefore, a reliable VMMR which is based on visual analysis can significantly facilitate the vehicle classification with little cost.

VMMR systems can also be used in conjunction with technologies such as Automated Number Plate Recognition (ANPR) in order to improve the system reliability. For instance, many vehicle monitoring and security systems rely on ANPR such as road-rule enforcement cameras. However, ANPR systems alone are not sufficient to ensure security and reliability of vehicle identification. Thus, an effective way to improve the reliability of ANPR systems is to integrate them with VMMR technologies [84]. Moreover, since most of proposed ANPR systems based on visual analysis, a parallel

VMMR system can be applied with no extra cost to boost the overall identification reliability.

1.1.2 Objectives

A number of successful approaches to VMMR are based on part-based image models [39, 35, 7, 10, 59, 85]. In these models, first input image is subdivided into a set of different vehicle parts (e.g. headlights, wheels) and then the classification is performed over each part and corresponding results are aggregated to identify the vehicle class. However, these methods rely on the knowledge of vehicle part locations, hence either such information is given in prior or a separate part detector is developed parallel to classification scheme. Those approaches that rely on prior knowledge of part positions, however require all vehicle images to be captured from a certain view point (e.g. front view) to prevent potential variations in part positions. Other part-based approaches that require a part detection scheme, although can perform on wider range of view points, however demand costly part annotations over vehicular images.

To overcome the mentioned shortcomings and challenges of commonly used VMMR approaches, we proposed a novel classification architecture to address the VMMR based on multi-scale attention windows. To provide an intuitive explanation of the proposed classification scheme, let us consider the case where we are supposed to recognize a vehicle according to its make and model and based on its appearance. At first glance, we look at the overall shape of the vehicle, then we will concentrate on parts of the vehicle which are more deterministic. Depending on vehicle model, these parts may include a large portion of the vehicle or small details such as the shape of headlights. Therefore, the visual features of different scales help us to make a decision on vehicle make and model. Based on this intuition and also our experimental results, we show that indeed if visual representations over the most deterministic part of a vehicle and over different scales are combined, such representations provide the classifier more distinctive features and hence improves the classification accuracy.

In order to obtain the mentioned objective, we propose an architecture including CNN-based localizer and classifier modules connected with spatial transform module. In the proposed scheme, localizer module predicts multi-scale attention windows

and classifier performs VMMR over these attention windows. Spatial transform module allows spatial manipulation of data between localizer and classifier module in forward pass, and error back-propagation from classifier to localizer in backward pass. As a result, localizer module trained by minimizing classification error without providing part annotations. Thus, attention windows are predicted to capture vehicle parts on different scales which minimizes the overall classification error.

Furthermore, we proposed a training methodology which allows localizer to predict attention windows of desired scales. In the proposed training methodology, first, scale-specific convolutional layers are pretrained over patches extracted with desired scales from training image. Next, the localizer is trained to predict the attention windows with corresponding scales minimizing the classification error.

Moreover, we minimized a joint loss function compromises classification error both on vehicle make and model. This allows joint classification on vehicle make and model instead of utilizing a classifier for each task. We have experimentally evaluated our proposed classification scheme over two publicly available challenging vehicle datasets [105] ,[52], advancing previous state-of-the-art by considerable margin.

1.2 Semantic Segmentation of Satellite Images

Semantic segmentation (also semantic labeling or pixel-based classification) is the task of assigning each pixel of a given image a class from a number of predefined classes. These classes in the case of satellite images are usually defined as land-use classes such as vegetation, road, building, water bodies and so on. Segmentation of satellite images is regarded as a fundamental component in many technologies employed for automated aerial image interpretations.

Despite the intense research in this field, many proposed architectures focus only on a particular case where a single area such as a city is subject of the study, however, in practice this is not necessarily the case. For instance, to perform an emergency mapping, the segmentation technique is required to perform in any area with almost constant and reliable accuracy. Nevertheless, this is not straightforward due to the statistical variations present between satellite images when they are captured by different sensors or in different time intervals and most importantly over different locations worldwide.

To provide a better insight, let consider a segmentation class such as building. For instance, buildings which are constructed in a city with arid climate can include visual features that vary a lot from the same class but located in tropical regions. Most of these variations stem from the difference in the material used for constructing building in different locations and some others may also be resulted from different type of sensors used to capture images. Hence, if the segmentation scheme is optimized over a particular region, it is most probably not applicable over another area. Therefore, this necessitates a study on segmentation architecture which is more robust to variations in satellite images.

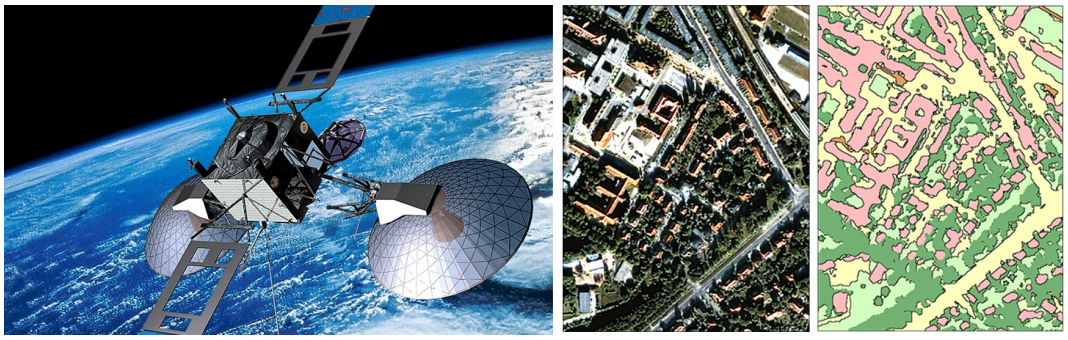


Fig. 1.2 Semantic segmentation of satellite images.

1.2.1 Motivation

In recent years, huge and still growing volume of high resolution aerial images has underlined the significance of aerial image interpretation. Aerial image interpretation has found a broad range of applications such as urban planning, agriculture, land-use analysis, climate modeling, post-disaster damage assessment, and many others. However, analyzing such enormous amount of aerial images by hand is very extensive and hence prohibitive which requiring an automated aerial image interpretation tools.

Recent advances in deep learning as well as the availability of GPU-accelerated frameworks have led to a breakthrough in many fields including computer vision tasks. Various deep learning architectures has been emerging and outperforming their competitive traditional methods by considerable margins in many fields [57] ,[70], [13], [6], [54]. These deep learning tools operate on a large amount of data allowing training and estimating the probability distribution over the desired domain.

Deep learning architectures have also found applications to satellite image segmentation following the availability of large amount of annotated training images [50, 99, 3, 64, 106, 75]. Despite the successes of deep learning architectures in satellite image segmentation, almost in all studies the training and evaluation are limited to a small geographical area such as a city. Therefore, a part of a city is used to train a proposed scheme then the other part of the same city is considered in the evaluation process of the trained architecture. Thus, despite their achievements, most of the proposed approaches pay less attention to the performance consistency over a wider range of aerial images.

Nevertheless, as already mentioned, in many applications the trained scheme must be deployable over an image which is not necessarily used in the training stage. In other words, in practical scenarios, there are always statistical variations between images used in training and the images to be segmented. In the literature, this variation between training (source domain) and test (target domain) images is defined as covariate shift [97]. Domain adaptation techniques provide solutions and aim to adapt an architecture which is trained on source domain to be deployed over a target domain. Therefore, domain adaptation techniques which improve the segmentation performance over each target domain, are in great demand.

1.2.2 Objectives

In this thesis, our main objective is to improve the performance of the proposed deep learning method over a much larger geographical area. Moreover, we investigate domain adaptation techniques to enable high precision segmentation over each target area by considering the time constraint and reducing the manual inference in the segmentation process.

We have proposed a CNN based architecture comprised of encoder and decoder modules processing the input image in top-down and bottom-up manner respectively. The proposed encoder module extracts visual representations from the input patch while the decoder takes as input these visual features and predicts segmentation maps. Our objective is to learn visual features from a considerable large amount of various aerial images such that the learned representations be robust to variations between different images. We argue that a certain type of CNN namely residual network if employed as the encoder in the proposed architecture, enables learning

visual representations of increasing semantic level thanks to a deeper encoder. We show that these representations are less sensitive to image statistics variations and hence provide a more generalized solution for satellite image segmentation.

In addition, in order to avoid performance degradation in the segmentation of the proposed scheme associated with the covariate shift, we have proposed two domain adaptation techniques. First, we observe that updating the batch normalization layers statistics over the target image improves the network performance without human intervention. Second, we show that refining a trained network over a few samples of the image boosts the network performance with minimal human intervention.

We evaluate the proposed architecture over three datasets of satellite images: a building dataset constructed over nine areas of interest captured by high resolution Earth observation satellites, Inria aerial image labeling dataset [63] and ISPRS Vaihingen semantic labeling dataset [15]. The proposed method shows state-of-the-art performance in binary segmentation of previously unseen images and competitive performance with respect to more complex techniques in a multiclass segmentation task.

1.3 Cloud Screening

In recent years, the rapid advance of remote sensing technology has allowed acquiring high-resolution images over large geographical scale which can be employed in a broad range of applications such as environmental monitoring, agriculture, land-use analysis and so on. Nevertheless, clouds are estimated to cover about 66% of the Earth surface [86], potentially contaminating a large portion of the captured images. Such contamination masks objects on the Earth surface making the affected images useless for analysis.

1.3.1 Motivation

Onboard cloud screening can in principle be applied on the satellite platform as a pre-processing step before image compression and transmission, selecting images with a low cloud cover percentage and discarding the others, thereby avoiding to

process and transmit the images which are covered by clouds. Therefore, a cloud screening unit on the satellite platform enables efficient use of satellite resources.

1.3.2 Objectives

In this study, our objective is to address cloud screening by a CNN architecture which can be implemented on low-power accelerators with memory constraints, as are expected to be available in the near future for onboard processing of satellite images. Nevertheless, CNNs usually includes millions of parameters to provide high accuracy segmentation. Hence, to optimize CNN memory usage while preserving its accuracy, we empirically investigate various solutions such as performing on half precision floating point, reducing the number of input spectral bands or spatial size, utilizing a smaller number of network filters and also making use of shallower networks. Finally, we show that the proposed CNN can perform close to state-of-the-art approaches over a publicly available dataset of SPARCS [44] while occupying much less memory during inference.

The rest of the thesis is organized as follows: in chapter 2, we briefly describe the most important elements of CNNs, well-known architectures and also the training process of neural networks; in chapter 3 the related work, proposed methodology and experimental results regarding VMMR are detailed; chapter 4 describes the related work, proposed methodology and experimental results addressing the segmentation of satellite images; chapter 5 provides the background, proposed network architecture as well as experimental results regarding onboard cloud screening task; in the end, we draw our conclusions in chapter 6. In addition, appendix A proposes a methodology to address VMMR by conditional pooling layer which is ought to be further developed in the near future.

1.4 Publications

The methodologies and experimental results presented in this study have been published in the following articles:

- Ghassemi, S.; Fiandrotti, A.; Francini, G.; Magli, E. "Learning and adapting robust features for satellite image segmentation on heterogeneous datasets". *IEEE Transactions on Geoscience and Remote Sensing*, **2019**, *accepted*.

- Ghassemi, S., Fiandrotti, A., Caimotti, E., Francini, G., Magli, E. "Vehicle joint make and model recognition with multiscale attention windows". *Signal Processing: Image Communication*, **2019**, 72, 69-79.

- Ghassemi, S., Sandu, C., Fiandrotti, A., Tonolo, F. G., Boccardo, P., Francini, G., Magli, E. "Satellite image segmentation with deep residual architectures for time-critical applications". In 2018 26th European Signal Processing Conference (EUSIPCO) (pp. 2235-2239). IEEE.

- Ghassemi, S., Fiandrotti, A., Magli, E., Francini, G. "Fine-grained vehicle classification using deep residual networks with multiscale attention windows". In 2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP) (pp. 1-6). IEEE.

- Ghassemi, S.; Magli, E. "Convolutional neural networks for on-board cloud screening". *Remote Sensing:Special Issue "Real-Time Processing of Remotely-Sensed Imaging Data"*, **2019**, *under revision*.

Chapter 2

Convolutional Neural Network

Convolutional Neural Network (CNN) is a class of neural network that has gained outstanding performance in many fields particularly in computer vision tasks such as semantic segmentation, image classification, and object localization. CNN typically comprises two main parts: feature extractor and classifier. Figure 2.1 illustrates a particular CNN, namely LeNet [58], proposed for classifying hand-written digits. As can be seen, high-level visual representations from the input image are first extracted using multiple convolutional filters which are stacked together in a sequential manner. Then, these features are vectorized and utilized by a classifier implemented as multiple fully connected layers to perform digit recognition. The convolutional filters sometimes referred to as kernels have learnable weights and biases which are optimized during the training. By defining an objective function that we wish to minimize and also by adopting an optimization approach, the network parameters can be learned. In the following we describe each part of a CNN in detail and also we discuss the corresponding procedure related to training of neural networks.

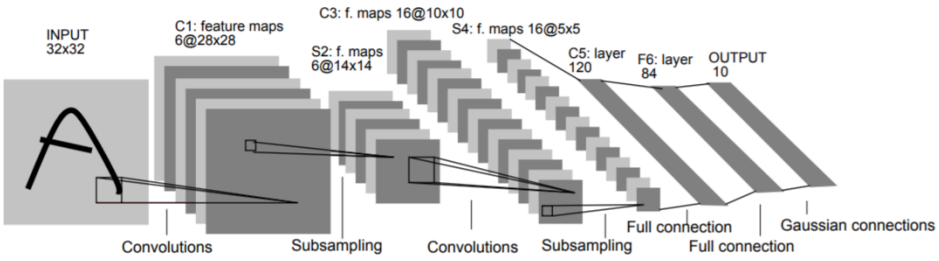


Fig. 2.1 LeNet [58].

2.1 Convolutional Layers

As it is mentioned earlier, CNNs are made of several convolutional layers stacked together in a sequential way. Each convolutional layer includes several filters which apply convolution operations over input data using filter weights and biases and produces a number of outputs equals to the number of the filters as it is shown in Figure 2.2. The resulting outputs called *feature maps* are then input to subsequent convolutional layers to generate even higher level feature maps.

A 3-D convolution operation can be expressed as followings:

$$y_{i,j,k} = b_k + \sum_{l=1}^{f_d} \sum_{s=1}^{f_w} \sum_{t=1}^{f_h} w_{k,s,t,l} \cdot x_{m,n,l} \tag{2.1}$$

where $y_{i,j,k}$ is the output corresponds to the k-th filter in the spatial location determined by i and j, b_k and $w_{k,s,t,l}$ are the k-th filter bias and weight respectively determined by indexes of s, t and l. f_w and f_h indicates filter width and height respectively while f_d is the filter depth. m and n are the input x indexes and are computed as follows:

$$m = d_w \cdot (i-1) + s \tag{2.2}$$

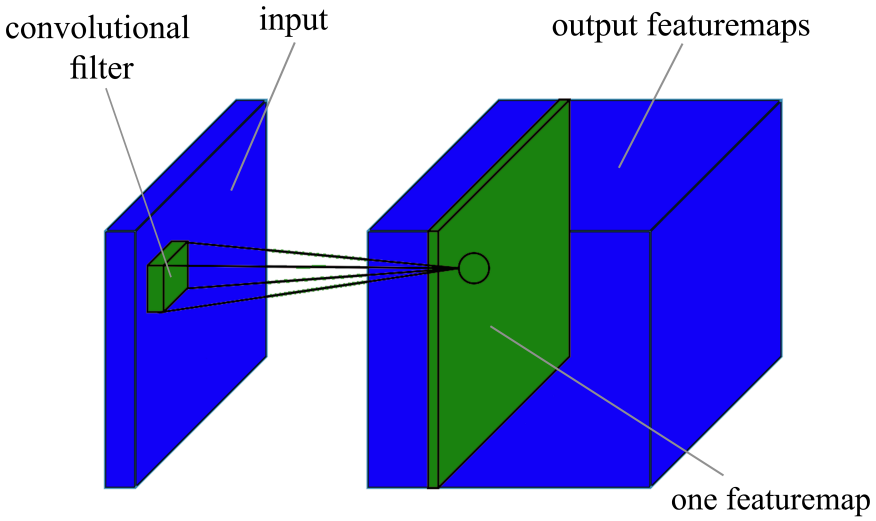


Fig. 2.2 Convolution layer.

$$n = d_h \cdot (j-1) + t \tag{2.3}$$

d_w and d_h are the convolution stride for height and width.

Therefore, the spatial size of output feature maps can be computed as follows:

$$width_{output} = \lfloor (width_{input} + 2 \cdot pad_w - f_w)/d_w + 1 \rfloor \tag{2.4}$$

$$height_{output} = \lfloor (height_{input} + 2 \cdot pad_h - f_h)/d_h + 1 \rfloor \tag{2.5}$$

To preserve the spatial resolution during convolution operation usually the input is padded with zeros as it is shown in Equation 2.4 and 2.5 as pad_w and pad_h. For instance, for convolution layer with the size of 3 by 3 ($f_w = f_h = 3$) and with stride of 1 ($d_w = d_h = 1$), to preserve the spatial size during convolution operation, the height and width of input array are padded with one pixel of zeros on each side ($pad_h = pad_w = 1$).

2.2 Fully Connected Layers

Fully connected layer, as its name implies, connects every neuron of input data to every neuron of output. As it is shown in Figure 2.7, each link connects an input neuron to an output neuron with an assigned weight. Accordingly, input values are multiplied by the fully connected layer weights and summed up to compute an output value in the output layer. Finally, a bias value is added to each output neuron.

Therefore, the j-th output of a fully connected layer can be computed as follows:

$$y_j = b_j + \sum_{i=1}^{N} w_{ij}.x_i \tag{2.6}$$

Where w_{ij} is the weight connecting input x_i to output y_j, b_j is bias corresponds to j-th output neuron. Typically one or more fully connected layers are employed at CNN last layers to perform classification or regression operation.

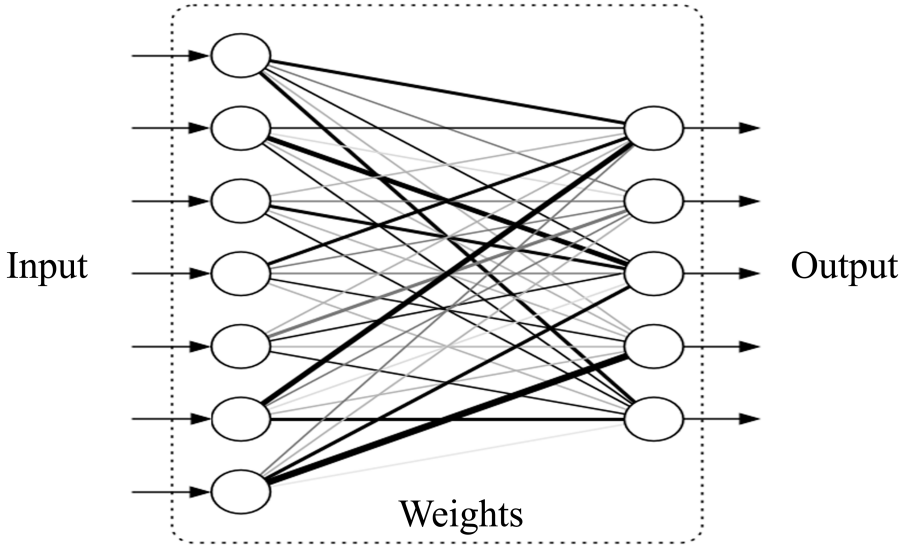


Fig. 2.3 Fully connected layer.

2.3 Pooling Layers

The pooling layers are interleaved in CNNs to reduce spatial dimensions of feature maps, hence their presence is crucial to reduce the computation complexity and memory consumption especially in very deep networks. The pooling layer performs a sub-sampling operation over a fixed size window and generates output by striding such window over input feature maps. Figure 2.4 shows a max pooling layer with the kernel size of 2 by 2 and the stride of 2 operating over an input with the size of 4 $\times$ 4 generating an output with size of 2 $\times$ 2 . In addition to spatial down-sampling, pooling layers contribute to translation invariant of CNNs and also prevent over-fitting by reducing the number of network parameters. However in some task such as

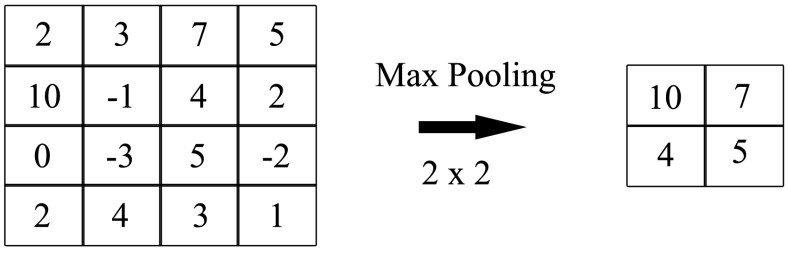


Fig. 2.4 Pooling layer.

image segmentation or localization, pooling layers may lead to loss of useful spatial information.

2.4 Deconvolutional Layers

Deconvolution also called “transposed convolution” or “reverse convolution” is proposed by [108] and has found many applications in CNN based schemes [73], [82] addressing the loss of mid-level cues related to downsampling layers such as pooling layers.

Deconvolution layers reverse downsampling process and scale up the input using a two stages process. First, the input pixels are interleaved with zeros to generate a scaled sparse output (i.e. unpooling). Next, through a convolution-wise operation, the sparse data becomes dense using a set of learnable filters. Therefore, deconvolutional layers, same as convolutional layers, include filters which are learned during the training stage, hence one could consider deconvolutional layers as learnable upsampling filters. Many recent designs, in particular fully convolutional networks, use deconvolutional layers to recover spatial resolution of feature maps. Recovering the spatial resolution of feature maps plays a crucial rule in image segmentation scheme where the predicted segmentation maps are required to match the input resolution.

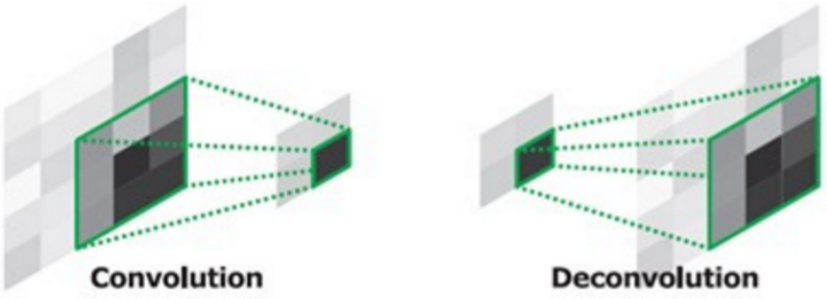


Fig. 2.5 Convolutional (left) and deconvolutional (right) layers.

2.5 Activation Functions

The activation function is the core element of neural networks and each convolutional and also fully connected layer are followed by an activation function. These activation functions mostly are non-linear functions which give neural networks the non-linearity property enabling solving complex problems. Without non-linear activation functions, neural networks are not able to solve sophisticated problems which are mostly non-linear. Figure 2.6 shows a set of most popular activation functions. Among these functions, ReLU (Rectified Linear Unit) is a vastly used activation function in recently proposed CNNs as well as our proposed architecture in this text. It achieves better performance by overcoming the issue of vanishing gradient which is caused by the saturating region of other activation functions such as Sigmoid. Moreover, by outputting zero for negative inputs, it encourages sparsity within the neural network which contributes to better generalization in the network.

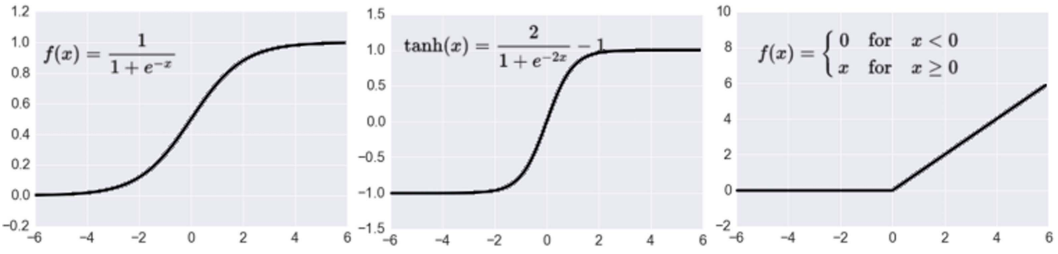


Fig. 2.6 Activation functions: Sigmoid (left), TanH (middle), ReLU (right).

2.6 Softmax Layers

Softmax layer is mostly implemented as the last layer of CNNs with applications in classification problems. Softmax layer assigns a probability to each input so that all probabilities sum up to one. Softmax function of a input can be expressed as follows:

$$Softmax(y_i) = \frac{e^{y_i}}{\sum_{i=1}^{N} e^{y_i}} \tag{2.7}$$

2.7 Batch Normalization

Batch normalization is essential in most recently designed CNNs and it is interleaved before each activation function throughout the network. Batch normalization is proposed in [46] to address *internal covariate shift* phenomenon which refers to changes in distribution of activations (i.e. activation function inputs) within a deep neural network. In the case of saturating activation, batch normalization prevents inputs to be trapped in the saturating region and hence speeds up the training. Moreover, batch normalization makes error gradients less correlated to network parameters scale or initial value, therefore eliminates the need of careful initializations and allows higher learning rates.

Turning now to normalization process, since training of neural network is carried out over batches of samples (mini-batches), batch normalization layer computes mean and variance of activations over each mini-batch and accordingly performs normalization. Considering $x = (x^{(1)}, ..., x^{(d)})$ as input vector of activation functions, normalization is performed as follows:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} \tag{2.8}$$

where $E[x^{(k)}]$ and $Var[x^{(k)}]$ are expected value and variance of $x^{(k)}$ respectively, and they are estimated over each mini-batch during training. Next, activations are scaled and shifted by γ and β which are learned independently for each activation:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)} \tag{2.9}$$

Such transformation preserves activations representation power from potential damage associated with normalization. Moreover, in each operation, normalization statistics (mean and variance) are tracked in the memory of batch normalization layer. So that during evaluation, these tracked statistics are used to perform normalization over validation mini-batches assuming training and validation samples follow similar distributions.

Another benefit of batch normalization is its regularization effects related to variations in computed mean and variance across different mini-batches which make each layer to be more robust to such variations.

2.8 Training

In this section, we briefly describe the most important steps of training a neural network.

2.8.1 Dataset

Besides the development of neural network architectures, another crucial factor in deep learning is the availability of large-scale datasets. Without large datasets, it is not feasible to train deep neural networks which include millions of parameters. Such datasets contain thousands of samples which are provided with proper annotations to allow supervised training of neural networks.

To begin with training, the common approach is to first subdivide available samples into training, validation and test sets (Figure 2.7). The training set is used to train and optimize the network according to a loss function and in order to perform a specific task (e.g. image classification). Another portion of samples is used to track the network performance during training course defined as the validation set. A number of evaluation metrics are computed over mini-batches of validation samples after each training epoch. Such validation allows keeping track of network performance over a distinct set from the training set in order to detect the over/under fitting problem and also in order to fine-tune hyperparameters such as learning rate in the course of training. At the end, when training has been completed, the test set is

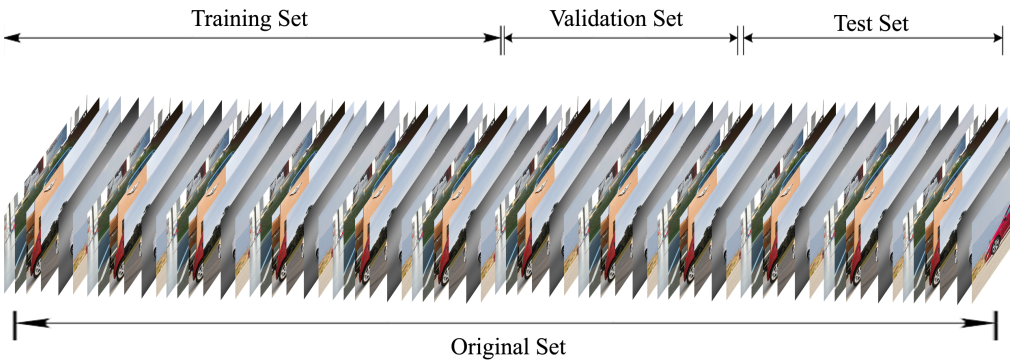


Fig. 2.7 Training, validation and test sets.

utilized to evaluate the trained network. It is necessary to evaluate the trained network over a portion of the dataset which is used neither in training nor in validation.

2.8.2 Data Augmentation

Data augmentation is applied over training samples to enlarge the dataset and to ease the network generalization. It is well-known that some label preserving transformations when applied over training samples reduce the risk of over-fitting in deep neural networks [89], [90], [54]. Considering the case of images which is the subject of this study, data augmentation can include a number of image transformations applied with random parameters. Cropping patches at random positions, random flipping, adding deformation by changing the image height or width are some examples of image transformations applied as data augmentation. Moreover, other techniques include adding random noise or alerting brightness and contrast which help the network learn visual presentations which are robust to these variations. Therefore, data augmentation facilitates generalization of the network by increasing the size of the dataset and enabling learning visual representations that are invariant to the transformations. A number of data augmentation techniques are shown in figure 2.8.

Fig. 2.8 Data augmentation.

2.8.3 Loss Function

The loss function is computed over network outputs using the input labels and enables to measure the network error at predicting the accurate output such as image class in the case of image classification. Depending on the problem we wish to solve, the loss is computed based on different functions.

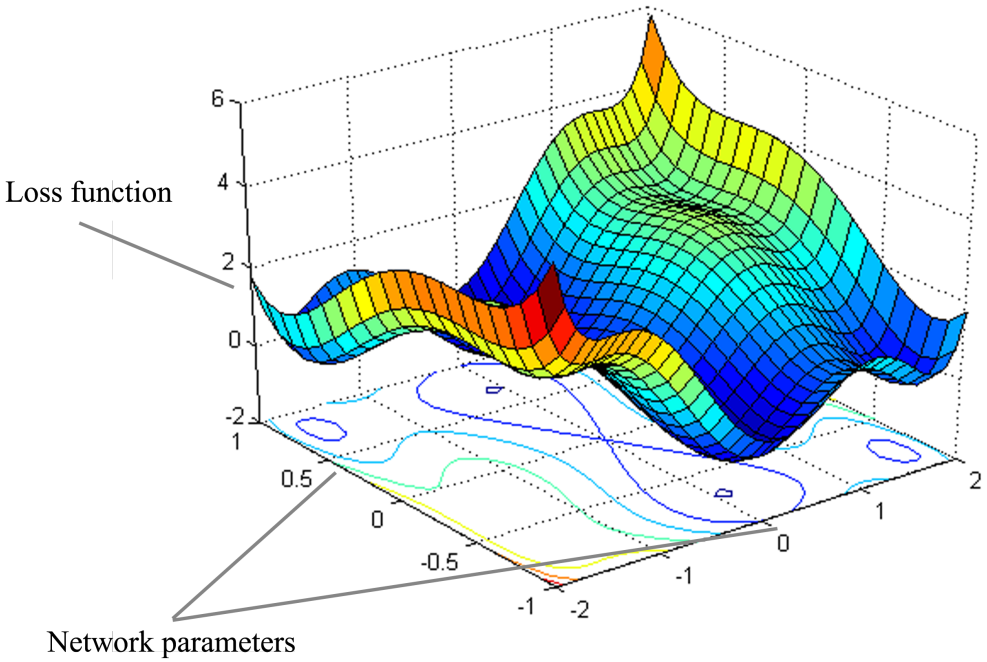


Fig. 2.9 Loss function.

For instance, let us consider the case of image classification where outputs of the network are scores over a number of classes. One of the most common function employed as loss in classification problems is cross entropy loss which is computed as follows:

$$L(\theta, y, t) = -\sum_{i=1}^{N} t_i \log (y_i). \tag{2.10}$$

where y is the output of Softmax layer, t is the one-hot vector representing the ground truth and N in the number of classes while θ represents network parameters. A simple illustration of a loss function with respect to two parameters is provided in

figure 2.9. As it is shown the loss is a non-convex function of network parameters which can include multiple local minima.

2.8.4 Optimization

After defining the loss function, an optimization methodology has to be adopted to optimize network parameters by minimizing the loss. As it is discussed, optimizing a network with millions of parameters is a sophisticated task which requires to be addressed carefully. To this end, backpropagation is introduced as an effective solution to this problem. Such that, first gradients of the loss function with respect to network parameters are derived. Then, according to the optimization policy, network parameters are updated. Since training is carried out over mini-batches of samples, optimization is also performed over a number of mini-batches.

One of the most commonly used optimization algorithms is Stochastic Gradient Descent (*SGD*). Considering SGD as optimization method, network parameters at $n+1$-th mini-batch (θ_{n+1}) are updated according to their value at n-th mini-batch and loss value of $J(\theta; y^{(i:i+l)}; t^{(i:i+l)})$, as followings:

$$\theta_{n+1} = \theta_n - \eta \cdot \nabla_\theta J(\theta; y^{(i:i+l)}; t^{(i:i+l)}) \tag{2.11}$$

where $y^{(i:i+l)}$ and $t^{(i:i+l)}$ are network outputs and their corresponding labels over mini-batch samples with length l.

Nevertheless, due to oscillations around local minima where the surface of loss function is steeper in one dimension than in another, the optimization performance degrades [94]. To address this issue, SGD with momentum [79] is employed as follows:

$$v_{n+1} = \gamma v_n + \eta \nabla_\theta J(\theta; y^{(i:i+l)}; t^{(i:i+l)}) \tag{2.12}$$

$$\theta_{n+1} = \theta_n - v_{n+1} \tag{2.13}$$

Using momentum with SGD enables faster optimization by adding a fraction γ of update vector in previous iteration v_n to the current update vector v_{n+1}. Figure 2.10

provides a simple illustration of SGD optimization with and without momentum (right and left graphs respectively) around a local minima.

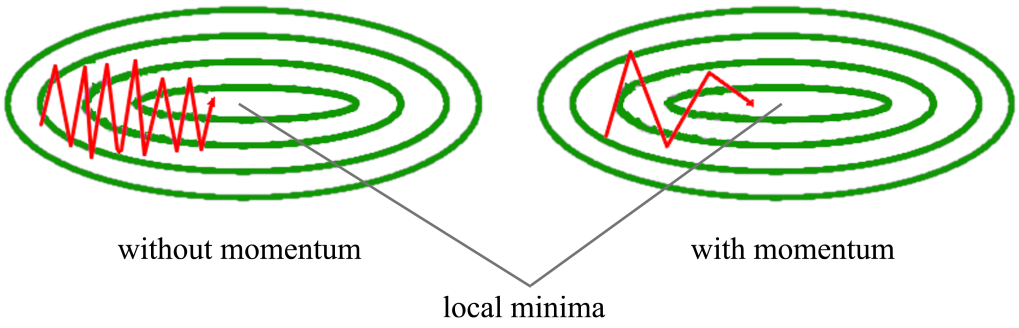


Fig. 2.10 SGD with and without momentum.

2.8.5 Regularization

Regularization techniques are proposed to tackle the problem of overfitting, particularly in very deep neural networks. Overfitting refers to a condition where a network performs much better over training set than over validation or test set. In other words, the network is not capable of generalization, instead, it memorizes irrelevant information (i.e. noise) from training samples. Regularization techniques provide a better generalization of neural networks by limiting the number of network parameters as well as their values. The most common types of regularization are *L1* and *L2* regularization where an additional term is added to overall loss function (cost function):

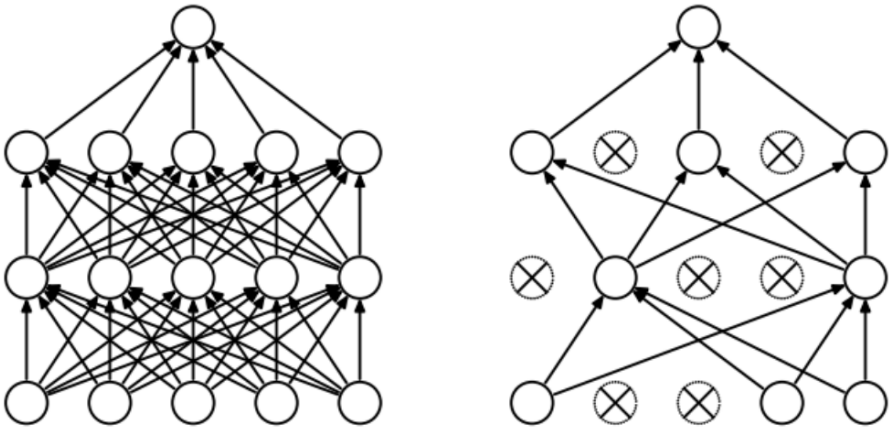
$$Cost = Loss + \frac{\lambda}{2m} \sum_{i=1}^{m} \|\theta_i\|_1 \tag{2.14}$$

$$Cost = Loss + \frac{\lambda}{2m} \sum_{i=1}^{m} \|\theta_i\|_2^2 \tag{2.15}$$

where Eq. 2.14 and 2.15 represent L1 and L2 regularization respectively, λ is the regularization parameter and m is the number of learn-able parameters.

In addition to L1 and L2, dropout [92] is another effective regularization method which avoids overfitting by randomly disabling a portion of neurons of a layer. Figure

2.11 provides a visual representation of implementing dropout over a fully connected layer.



(a) Standard Neural Net (b) After applying dropout.

Fig. 2.11 Dropout [92].

2.9 Residual Networks

Residual networks (*ResNets*) have been proposed by He *et al.* [36] to address the degradation problem in very deep CNNs. While using deeper and larger CNN is assumed to increase the network performance, it is shown in practice that these networks suffer from a degradation in performance. This degradation indicates that deep CNNs are not straightforward to optimize and problems such as vanishing/exploding gradients prevent them to perform in their full capacity. ResNets addressed the aforementioned issue by introducing a residual learning framework.

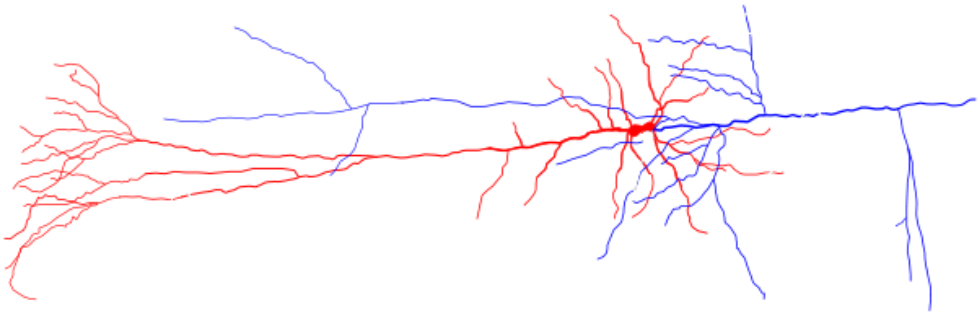


Fig. 2.12 Brain cells having similar structure to residual networks.

Instead of learning unreferenced functions, functions with reference to input are learned within ResNets. This phenomenon is achieved by providing skip connections and as a result identity mapping between layers within the network. The authors demonstrate that residual functions are easier to be learned and ResNets can gain improvement due to increased depth.

In addition to artificial residual networks, it has been shown that biological neural networks follow a similar structure and can be seen as residual architectures as shown in Fig 2.12.

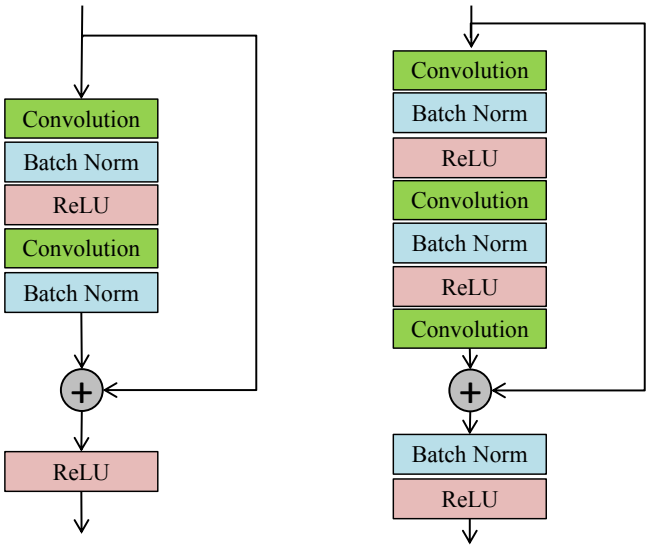


Fig. 2.13 Residual block for ResNet with depth ≤ 34 in left and for ResNet with depth ≥ 34 in right.

ResNets proposed in [36] consist of residual building units as it is shown in Fig. 2.14 for the ResNet of 18 layers. In ResNets with the depth of 34 layers and less, building unit (Fig. 2.13) includes two convolutional layers with a kernel size of 3 by 3 followed by ReLU as activation functions. While for ResNets with the depth of 50 layers and more, to manage the memory consumption, building unit includes two 1 by 1 convolution layers reducing and increasing the number of feature maps and in the middle a 3 by 3 convolution layer. The output of the residual unit is then added to input through short cuts. However, when dimensions do not match, a convolution layer is utilized to increase the input dimension to match the output.

In contrast to typical CNNs where pooling layers are interleaved within the network, in ResNets (except for the second block where a 2 by 2 max pooling is employed),

convolutional layer with a stride of two is interleaved instead. Therefore, the size of feature maps and memory consumption is managed by using a stride 2 convolution at the first layer in each block which scales down the input by a factor of two, whereas the number of feature maps increases from one block to subsequent block. In the end, a 7 by 7 average pooling is used and then features are vectorized and input to a fully connected layer to finally perform the classification.

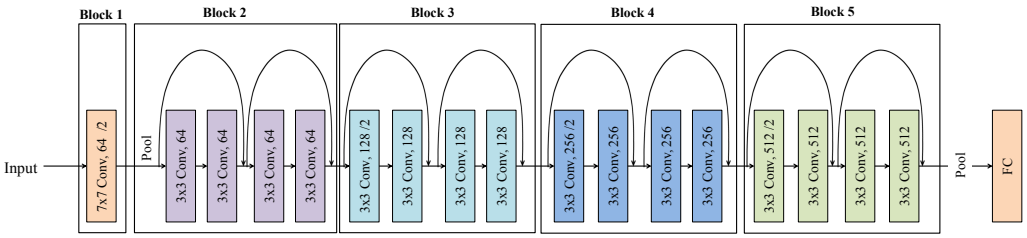


Fig. 2.14 Residual network of depth 18.

2.10 Fully Convolutional Networks

Fully convolutional network (*FCN*) [62] is introduced to address the pixel-level prediction on images such as semantic segmentation. FCNs can process input of arbitrary size and generate the output with the desired resolution. In FCN, fully connected layers are replaced with convolutional layers with the kernel size of 1 by 1. Therefore, the decisions are made according to local visual representations rather than global features which is performed in typical CNNs in classification tasks. For instance, considering the task of assigning a class to image pixels, each pixel is classified according to its neighboring pixels within a certain range. While for image classification, the image is labeled based on the visual representations of the

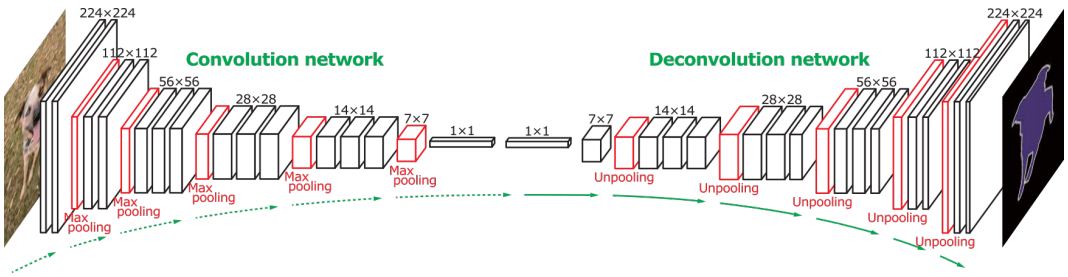


Fig. 2.15 Fully convolutional network [73].

whole image. Hence, many recent CNNs designed for semantic segmentation are fully convolutional.

As is shown in Fig. 2.15, FCN encompasses two parts. One part (Convolutional network in Fig. 2.15) extracts high-level feature maps however with coarser resolution compared to input. The resolution degradation is mainly caused by pooling layers or strided convolutional layers in the network. The other part of the network (Deconvolution network in Fig. 2.15) recovers feature maps resolution through upsampling layers. The upsampling layers scale up feature maps using operations such as bilinear upsampling or deconvolutional layers. Deconvolutional layers can reverse the convolutional layer process, upsampling input through a set of learnable filters

Chapter 3

Vehicle Make and Model Recognition

This chapter describes the developed classification scheme for VMMR as follows. Sec. 3.1 briefly overviews the recent literature in the field. Sec. 3.2 describes our developed architecture to tackle VMMR, while Sec. 3.3 details the related procedure to generate samples for training. In Sec. 3.4 the devised training methodology is provided. Finally, in Sec. 3.5 we experimentally assess the performance of our proposed architecture over two distinct vehicular datasets.

3.1 Related Work

Considering recent approaches proposed to address VMMR and related fine-grained object recognition problems, we can divide them in three groups.

The first group includes schemes which mostly rely on part-based models. Approaches relying on part-based models leverage a priori knowledge of the geometry of the object (and, in some cases, of the scene) for improved performance.

Hsieh *et al.* [39] address VMMR relying on Speeded-Up Robust Features (SURF) and Histogram Of Gradient (HOG). In the proposed methodology, first, the vehicle is located by detecting symmetric matching pairs in the input image using symmetrical SURF descriptors. Then, the vehicle front region is subdivided into several grids each indicating a different part of the vehicle. HOG and SURF descriptors are extracted from these parts which then employed by several weak SVM classifiers. Finally, with a Bayesian averaging technique, these classifiers are integrated to perform VMMR. However, one major shortcoming is that the proposed symmetric part detector works on vehicles that captured from the front view with viewing angle not exceeding more than 20 degrees.

Region based CNN is proposed by Girshick *et al.* [30] in order to address general object detection. R-CNN extracts object proposals from the input image exploiting an external region proposal algorithm (selective search), then the warped proposals are fed to a CNN for feature extraction and eventually an SVM performs object classification. Authors in [29] improve the proposed technique extracting object proposal straight from the features maps resulting in more efficient R-CNN implementation. Later, in [81], Ren presents an evolution of R-CNN that introduces a Region Proposal Network (RPN) to replace the external proposer hence enabling cost-free region proposals. Many recent approaches make use of R-CNN in fine-grained recognition tasks.

Zhang *et al.* [112] address the different yet related problem of fine-grained birds classification relying on R-CNN for learning both the whole object and parts detectors as shown in Fig. 3.1. Object semantic parts are realized and scored by part detectors and a geometric constraint are applied over these regions. Part-specific features are then learned from these object parts which allows training parts-specific SVM classifiers. While model-based objects achieve good performance especially in fine-

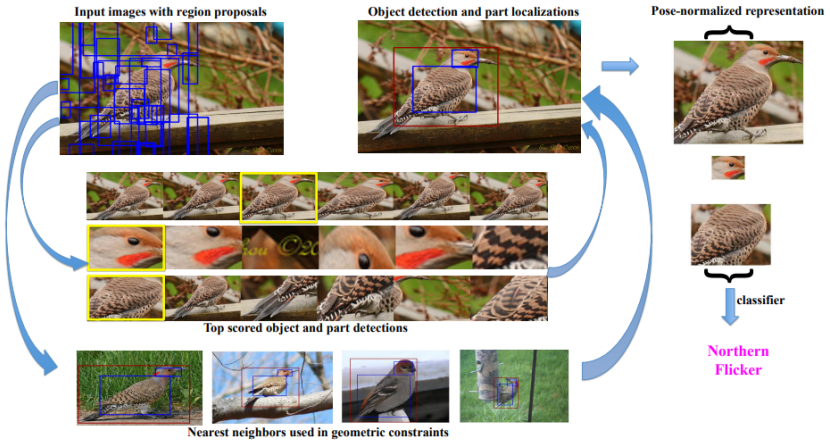


Fig. 3.1 Proposed scheme in [112] addressing related fine-grained birds classification using R-CNN.

grained object classification tasks, this comes at the price of increased architectural complexity and additional effort in annotating training images.

In many other approaches, VMMR is addressed based on models which require the vehicle to be seen from a certain viewpoint so that the model does not deal with the intra-class variations related to changes in viewpoint.

Llorca *et al.* [61] tackle the problem of sub-model vehicle recognition (motorization, trim-level, etc.) by focusing on variations in car emblems size and location. Namely, they extract HOG features from images captured from rear-view, then a committee of class-specific SVM classifiers is used to classify the vehicle. Although the devised classification scheme marked high accuracy over more than 1000 vehicular images, there is a major downside, the proposed scheme is only deployable over images that are captured from rear-view limiting its application.

In [35], He *et al.* propose a method to jointly detect car make and model from surveillance camera images. First, a parts-based model tailored to the front view of a car detects components such as lamps and license plate. Then, centers of these detected parts provide anchor points to normalize feature region with respect to viewpoint and illumination. Next, specialized CNNs classify each part of the vehicle, yielding a global car classification. While this method achieves good performance over a homegrown test dataset, its design tailored specifically to front views narrows somewhat its scope.

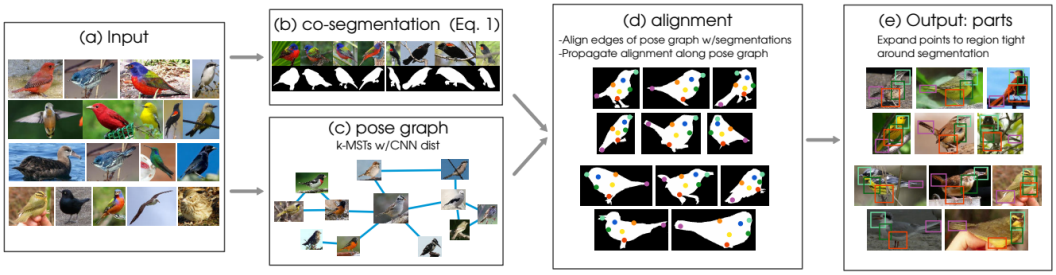


Fig. 3.2 Procedure proposed to tackle fine-grained birds classification in [53].

Hu *et al.* [40] propose a spatially weighted pooling layer replacing the standard pooling layers in conventional deep CNNs. Such pooling layer includes a number of weighted masks whose parameters are learned in the course of training. Hence the last fully connected layer is provided with robust feature representations by magnifying features corresponding to discriminative parts of the image. Despite advancing state-of-the-art over publicly available car datasets, the proposed scheme introduces two drawbacks. First, since all learned masks are fixed and not conditioned on the input image, it challenges the ability of the network to handle images where vehicle position and scale do not remain constant. Moreover, replacing CNN last average pooling layer with a number of parameterized pooling layers (equal or more than the number of features) comes with the price of an increasing number of network parameters.

Biglari *et al.* [7] employ Support Vector Machines (SVMs) to discover the most relevant parts of each vehicle class. Then, for each vehicle category, a part-based model is trained. Next, a cascading scheme is applied over class-specific classifiers performed on inputs sequentially. Despite the good performance over two vehicle datasets, the drawback of their designed system is that all images are required to have a similar view limiting their application scope.

In addition to part-base and view-based models described above, some other work has been proposed to tackle related fine-grained object recognition problem however leveraging more complex architecture to obviate the need of costly part annotations.

Krause *et al.* [53] address the related problem of fine-grained classification of bird species. The proposed scheme consists of several preprocessing steps enabling part detection without the need of annotations as shown in Fig. 3.2. First,

a co-segmentation is performed which allows the separation of foreground and background. In addition, a foreground refinement step is applied to further improve the segmentation. Then, in the next step, a graph is constructed in which based on co-segmented images, images with similar poses are connected together. This pose graph facilitates aligning images that share similar poses. Next, a part generator component is employed to produce parts for images with similar poses. A set of discriminative parts are then realized through a max-margin template selection policy. In the end, the classification is performed over these parts. While their approach shows good performance on a birds dataset, since it requires a number of intermediate steps such as pose alignment and co-segmentation, it further complicates the overall system pipeline design.

A bilinear CNN model to tackle the fine-grained classification is proposed in [60]. Two CNNs are employed to process input image then the generated outputs are multiplied using an outer product. Next, the result of the outer product is downsampled using pooling layer and fed into fully connected layers for classification. Therefore, the proposed architecture enables capturing the interaction between an object visual features and the part which is particularly useful for fine-grained classification. The bilinear model achieves excellent performance over fine-grained recognition task, however, one drawback of the proposed architecture is the large number of network parameters related to employing two distinct CNNs. Moreover, since the function of each CNN branch in bilinear model are not fully understood, considering a different classification task, the architecture yielding the best results should be obtained through extensive experimental results exploring different combinations of CNN in each branch.

The scheme proposed in this work can be categorized into the latter group, where discriminative parts of the image are detected without prior knowledge of object parts.

3.2 Proposed Architecture

This section describes the architecture of our proposed VMMR system, as it is illustrated in Fig. 3.3 . The architecture is composed of one *localizer* module and one *classifier* module connected via a *transform* module. In the following, we detail the architecture of each module, whereas the description of the relative training procedure is deferred to the following sections.

3.2.1 Localizer Module

The localizer module, as illustrated in the dashed box in Fig. 3.5, processes the input image and predicts parameters of W (constrained) affine 2D transforms indicating the scale and position of each attention window. In our design, the localizer module is composed by one convolutional *trunk* designated for feature extraction which is followed by two branches responsible for predicting W affine transforms parameters. In the localizer design, we consider several aspects to improve network performance which is described in details as follows.

Concerning the convolutional trunk, unlike Jaderberg *et al.* [47] where a *GoogLeNet* architecture is originally considered, we employ the ResNet18 architecture illustrated in Fig. 3.4 in reason of the better localization accuracy enabled by ResNet architecture [24].

As it is illustrated in Fig. 3.4, ResNet18 consists of five convolutional blocks (i.e. blocks A, B, C, D, E in the figure) followed by a pooling layer and fully connected layer. Considering an input image sized 224×224, block E outputs 512 feature maps each sized 7×7. Then, the penultimate pooling layer vectorizes the corresponding feature maps which are then input to the fully connected layer. However, as it will be explored further in Sec. 3.5, since ResNet18 is originally proposed for image classification task, we have made modifications to network architecture optimizing it for localization task.

First, we have observed that there is a trade-off between the number of output feature maps and their spatial resolution over localization accuracy. Such that, due to the presence of strided convolutional layer in each block, the resolution of output feature maps decrease by a factor of 2 over each block which in fact can reduce localization accuracy due to spatial information loss. However, by proceeding to deeper blocks,

the number of feature maps increases allowing to capture interactions between a larger number of features which eventually can improve localization accuracy. As a result, to find the optimal number of convolutional blocks, we run an experiment which is provided in Sec 3.5 where we realized that the best choice for our localizer is to truncate ResNet at block D including four out of five convolutional blocks. Therefore, the localizer convolutional trunk takes as input image of 224×224 and outputs 256 feature maps sized 14×14 (block D output).

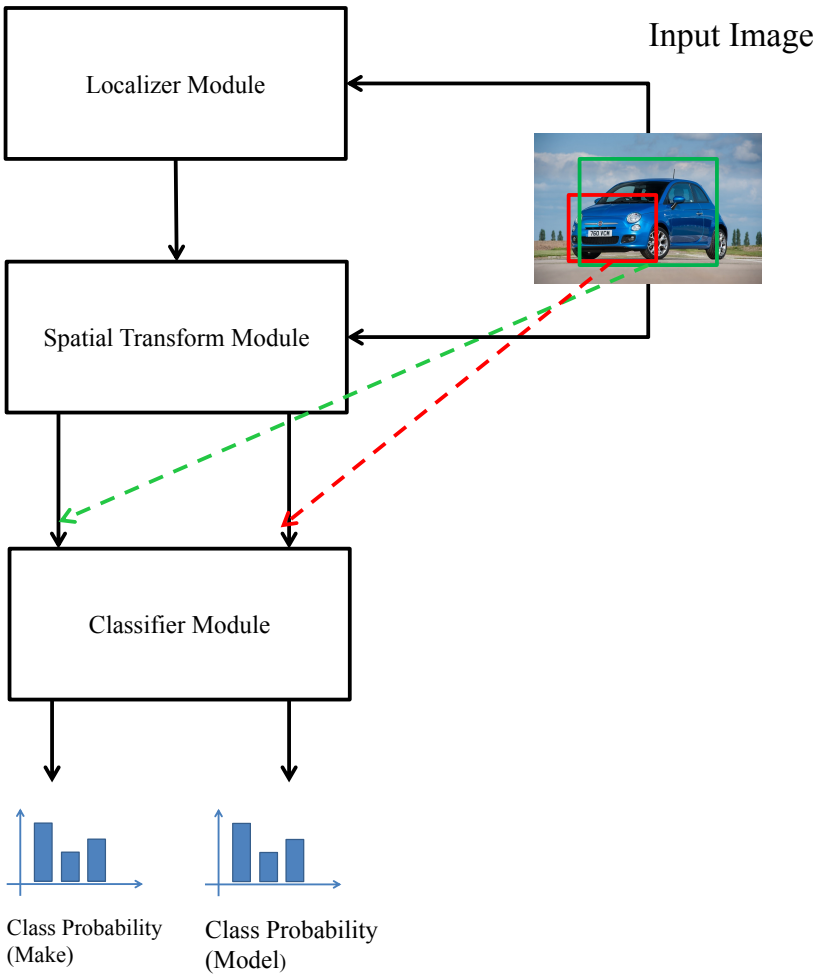


Fig. 3.3 An example of the proposed VMMR architecture with two attention windows ($W = 2$). A vehicle sample with predicted two attention windows ($W = 2$) is depicted in the figure.

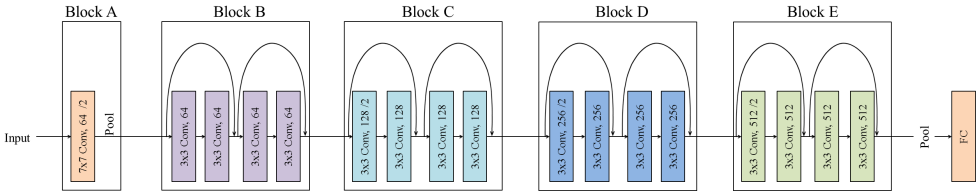


Fig. 3.4 The ResNet18 architecture subdivided in 5 convolutional blocks with different depth and number of feature maps and feature map size.

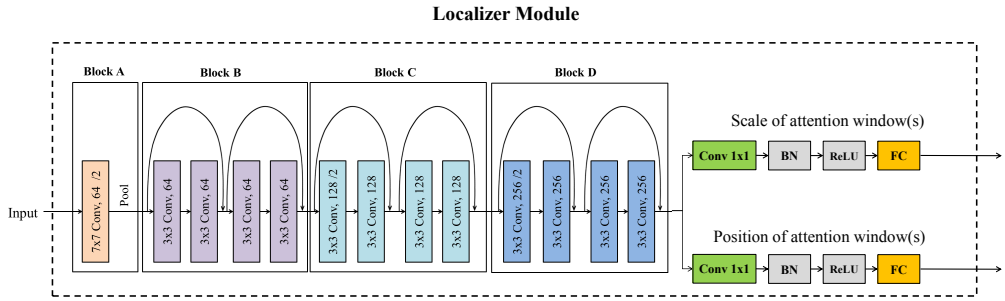


Fig. 3.5 The localizer module architecture.

Secondly, we have removed the pooling layer which is employed in standard ResNet design. Pooling is utilized in classification task which allows extracting global visual context, however in localization, since we are more interested in local features and also to prevent spatial information loss, the pooling layer is omitted in our design.

Next, we have employed two separate branches, each one specializes in their own task, namely one predicts scales of attention window and the other one predicts the positions. Our preliminary analysis showed that the position and scale distributions substantially differ, justifying learning separate prediction functions via distinct output branches. In addition, having distinct output branches for position and scale allows imposing different learning rates for each branch, facilitating training the module from scratch, as discussed in the following sections. However, these branches share most convolutional layers (i.e. blocks A, B, C, D), resulting in more efficient architecture, and separate in the last convolutional and fully connected layers.

Now describing the out branch architecture, each one consists of one convolutional layer with 1×1 filters with ReLU activation functions followed by one fully con-

nected layer with hyperbolic tangent activations. The 256 14×14 feature maps output by convolutional trunk are input to each of these branches where 1×1 convolutional is employed to reduce the dimensionality generating 8 14×14 feature maps. We should note that the dimensionality reduction is crucial to prevent overfitting and it has to be applied before feature maps are input to last fully connected layers. Moreover, we have included the convolutional layer, which is responsible for dimensionality reduction, within output branches in place of prior dimensionality reduction in convolutional trunk preventing potential information loss. In the next step, the feature maps are vectorized forming a $8 \times 14 \times 14$ feature vector input to fully connected layer. Each fully connected layer has output neurons equal to $2 \times W$ presenting scale or position across image width and height.

For clarity of exposition, let us consider the case of two attention windows ($W = 2$), the localizer module output can be represented by the vector

$$\left[\sigma_x^1 \ \sigma_y^1 \ \tau_x^1 \ \tau_y^1 \ \sigma_x^2 \ \sigma_y^2 \ \tau_x^2 \ \tau_y^2 \ \right]. \tag{3.1}$$

For the i-th attention window, the localizer output can be interpreted as the position and scale coordinates of a bounding box cast over the input image. Each i-th box is defined by the relative horizontal and vertical position of the box ($\tau_x^i \ \tau_y^i$) and horizontal and vertical scale of the box with respect to the input image ($\sigma_x^i \ \sigma_y^i$).

3.2.2 Spatial Transform Module

The spatial transform module (STM) is employed between localizer and classifier modules and serves two purposes concerning the architecture shown in Fig. 3.3.

In the forward pass, STM takes as input the predicted affine transform parameters (i.e. attention windows scale and position) and also the input image, then through a sampling process, it samples input at corresponding attention windows. Thus STM enables spatial manipulation of the input image within the proposed architecture and between localizer and classifier.

In the backward pass, STM enables backpropagating the classification error from classifier to localizer module using the chain rule. To be clear, let us consider the case of two attention windows (W=2) where there are two convolutional trunks in

the classifier, each processing the sampled input over each attention window. In this case, localizer outputs 8 parameters representing scales and positions of two attention windows. Hence, to backpropagate the error gradients from the classifier to the localizer, the chain rule is used as follows: first, the error gradients are backpropagated from the classifier output layer to each classifier convolutional trunk. From there, the gradients with respect to each of convolutional trunks parameters and also their inputs are computed. Next, since the convolutional trunks inputs are in fact the sampled input image at attention windows, and because the image sampling is differentiable, the computed error can be backpropagated through sampler from trunk inputs to its corresponding output neurons in the localizer output layers where the scale and position of each attention window have been predicted. As the error with respect to each localizer output neurons is computed, the gradients with respect to localizer parameters can be computed as well using chain rule up to the first localizer layer. Finally, as the error gradient with respect to all network parameters is calculated, based on optimization policy parameters can be updated. In this way, the computed error gradients in the first layer of the classifier is reused to compute the error gradients with respect to affine transform parameters. Therefore, as the error gradients are calculated with respect to predicted affine parameters by the localizer, from there the error gradients with respect to all localizer parameters can be computed easily.

Notice that [47] deals with classifying objects parts with bounded scale variance and a prior known scale, so the attention windows size is assumed constant and square $\sigma_x^w = \sigma_y^w \forall w \in [1 \dots W]$. Therefore, [47] deals only with the problem of locating the attention window positions, not their scales. Conversely, in our vehicle classification problem, distinctive car parts may have a widely different size not known a priori, thus attention window scales must be predicted for each image together with its position.

As it is mentioned, the spatial transform module performs an affine transformation over the *source* (input) image producing a *target* (transformed) image. Such transformation takes place on the source image using a bilinear filter and through a grid characterized according to the parameters predicted by the localizer module. Considering a grid on the target image to be a regular grid (*i.e.* the grid in which points are equally distributed over target image height and width), the image sampler samples the source image over the grid defined by affine transform parameters. Thus, this transformation can be divided into two stages: first, a grid is defined over source

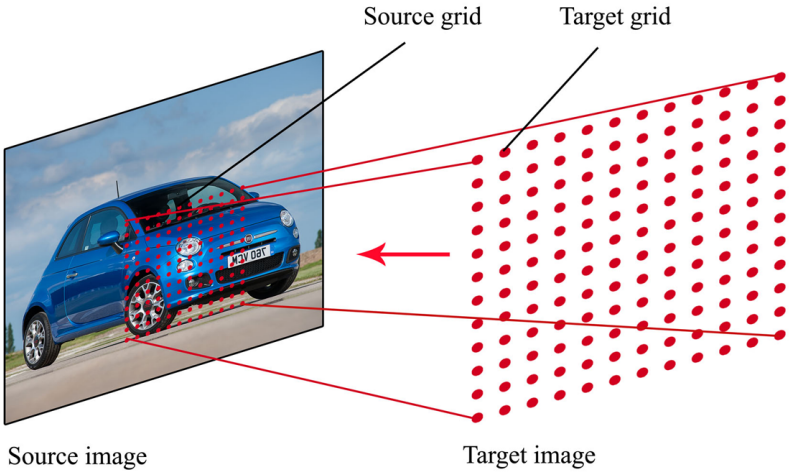


Fig. 3.6 First step of image sampling by spatial transform module: defining grids over source and target image.

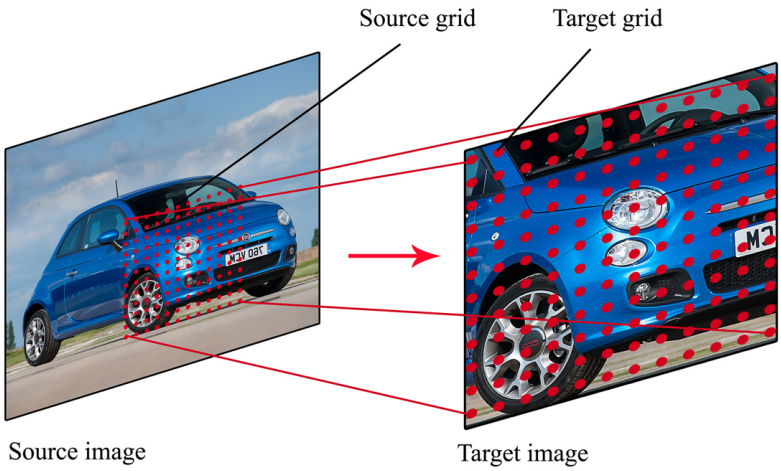


Fig. 3.7 Second step of image sampling by spatial transform module: sampling source image through bilinear sampler.

image as shown in Fig. 3.6, then based on the generated grid the image is sampled to generate target image as shown in Fig. 3.7.

For clarity of exposition, let $[x^s \; y^s]$ be the horizontal and vertical coordinates of one point of the grid in the source image, and let $[x^t \; y^t \; 1]^T$ be the corresponding coordinates of a point of the regular grid in the target image. The transform module performs the point-wise transformation

$$[x^s \; y^s]^T = \mathscr{T} \left[x^t \; y^t \; 1\right]^T, \tag{3.2}$$

where $\mathscr{T}$ is a 2×3 *transform matrix* that allows any 2D affine transformation such as translation, rotation, scaling and shearing. $\mathscr{T}$ defines the shape of the grid in the source image based on which the sampler samples the input image. For the purpose of this work, we constrain the set of affine transformations to translation and scaling. Therefore, in our architecture the transform matrix $\mathscr{T}$ is defined as

$$\mathscr{T} = \begin{bmatrix} \sigma_x & 0 & \tau_x \\ 0 & \sigma_y & \tau_y \end{bmatrix}, \tag{3.3}$$

where the non-zero elements of $\mathscr{T}$ are the transform parameters predicted by the localizer module as described above.

Thus far, we have described the process of attention windows prediction by localizer as well as image sampling process by STM, next, we detail the classifier task in processing input image over attention windows to predict vehicle classes.

3.2.3 Classifier Module

The classifier module, as illustrated in the dashed box in Fig. 3.8 for $W = 2$, processes the attention window(s) generated by spatial transform module and outputs two distinct score maps over vehicle make and model classes respectively.

The classifier module architecture we present in this work includes a number of major improvements over [47]. In our design, the classifier module is composed of W convolutional trunks each one operating on a specific scale, and also two output branches which are specialized in make and model prediction respectively. In following we detail each novel aspect of classifier architecture.

Concerning W convolutional trunks, each trunk processes sampled input image over an attention window hence over a particular scale. To this end, first during training as described in Sec. 3.4, each convolutional trunk is trained over patches extracted at a specific scale, hence each trunk learns to recognize visual features over that scale. As a result, each convolutional trunk can be seen as a sequence of scale-specific convolutional layers. Later, when the localizer module is trained by backpropagating classification error from such scale-specific convolutional trunks, the attention windows are generated with scales equal to patch scales used during training classifier convolutional trunks.

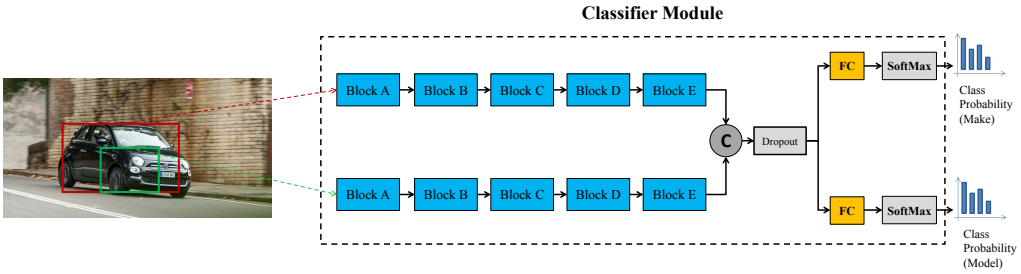


Fig. 3.8 The classifier module architecture.

Regarding the convolutional trunks architecture, we show experimentally that residual networks enable better classification accuracy than the original GoogLeNet architecture used in [47]. ResNets enable much deeper networks while their optimization remains quite feasible compared to plain CNNs. As it is discussed in Sec. 2.9, ResNets overcome challenges in training very deep CNNs by introducing shortcuts between convolutional layers. These shortcuts enable the network to learn a residual function with reference to the input image in place of non-referenced functions. Thus, it allows tackling the problem of vanishing or exploding gradients which is commonly occurred in very deep plain CNN.

Therefore, we employed ResNets, particularly a variant called "wide ResNets" [107] in the convolutional trunk. Wide ResNets differ from ResNets in the number of convolutional filters employed in each convolutional blocks. The increased number of filters enables achieving wider networks hence gaining higher accuracy with a shallower depth. We experimentally show in Sec 3.5 that wide ResNet with the depth of 50 achieves the best performance among other depths, thus it is used as the basis for convolutional trunk architecture.

Each convolutional trunk includes five blocks of wide ResNet-50. The last fully connected layer is omitted in order to obtain the feature vector as the output. The sampled input image at each attention window is processed by each trunk generating a feature vector with 2048 elements. For each attention window and hence for each scale, a feature vector is extracted which can be seen as an image descriptor over a particular scale. These feature vectors then are concatenated to form a vector sized $W \times 2048$ and fed to the two fully connected layers.

Moreover, because of a large number of input features to fully connected layer, we found dropout [38] to be useful in preventing the fully connected layers from overfitting, especially when $W > 1$. In particular, we want the expected number of active outputs of the classifier convolutional trunks to remain constant for any W. Thus, the dropout layer shown in the figure drops outputs with probability $p = \frac{W-1}{W}$ ($p = 0$ for $W = 1$).

Concerning the two output branches, this work aims at jointly predicting a vehicle make and model, thus the classifier module includes two distinct branches, one per attribute. Both branches take as input the vector of concatenated visual features extracted from attention windows at different scales. Sharing the same distinct-scales visual features for both make and model prediction is essential to train the overall architectures to learn features useful for both tasks. Also, this results in a leaner architecture in terms of number of learnable parameters with respect to the case where two distinct classifiers trained separately for each task. Each of the output branches contains a number of units that depends on the considered number of vehicle makes and models, respectively.

Finally, multinomial logistic regression (also known as a SoftMax layer) is used to yield the sought score maps over both vehicle make and model classes as follows.

$$Softmax(y_i) = \frac{e^{y_i}}{\sum_{i=1}^{N} e^{y_i}} \tag{3.4}$$

where y_i is the i-th output of the fully connected layer.

3.3 Generating Samples

In this section, we will provide the steps required previously to train the proposed architecture detailed in Sec. 3.2. To begin with, we describe the procedure used to extract patches from training samples over different scales. Next, we detail the data augmentation techniques employed on generated samples.

3.3.1 Extracting Patches with Different Scales

One prominent component of the proposed architecture is the scale-specific convolutional trunks in the classifier module. Although classifier convolutional trunks in terms of architecture are identical, the training procedure makes such parts scale-specific. Therefore (as it is detailed in Sec. 3.4) each trunk is first trained over patches which are generated from training samples at different scales. Hence the convolutional layers at each trunk are trained to extract visual features corresponds to a particular scale.

The procedure used to extract patches at the desired scale over vehicular images can be described as follows which is summarized in Fig. 3.9. First, each image is scaled (up or down depending on the original image size) using a bilinear sampling algorithm, so that the smallest side of the image be equal to particular value. This transformation is performed over image isotropically meaning that aspect ratio between image height and width is preserved avoiding deformation. Next, a fixed size patch of 224×224 equal to network input size is extracted randomly out of the scaled image. Therefore, since the size of the extracted patch is fixed to network

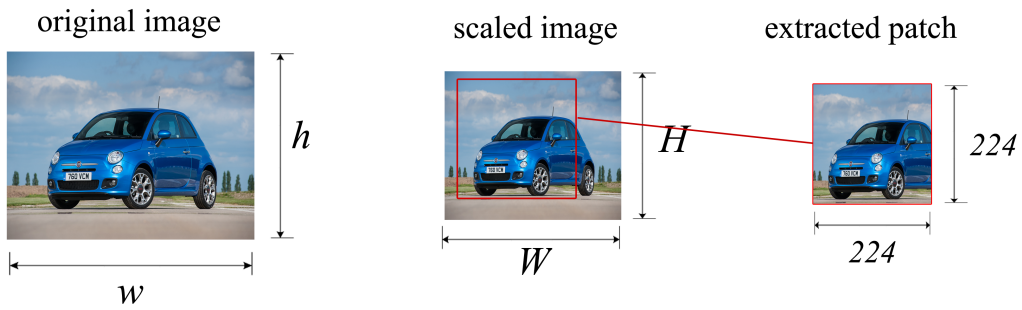


Fig. 3.9 The patch extraction procedure divided in three steps.

input size, the value used to scale training sample defines the patch scale with respect to the training sample. Scaling the training sample by large values allow the patch to capture small visual context while scaling training sample by smaller values enables the extracted patch to cover a larger portion of the sample.

To be clear, let h and w be the height and width of such training sample respectively. Let us assume that the smallest side of the sample is the height and it is scaled to be equal to a desired H. Also let us consider α as the aspect ratio of the training sample $\alpha = h/w$. To generate the training patch, the sample is scaled such that its height will be equal to H and its width to W where W is computed such that $W = \alpha^{-1}H\beta$ and β is drawn at random in the interval $[0.96, 1.04]$ during training in order to add slight deformation to extracted patch. Next, the patch with the size of 224 by 224 is extracted at random position from the scaled sample. As a result, depending on a desired H, the extracted patch will include only a random detail of the vehicle in the training image.

3.3.2 Data Augmentation

As it is explained earlier in Sec. 2.8.2, Data augmentation is applied over samples mostly during the course of training in order to increase the dataset size as much as possible. It is well-known that data augmentation helps generalization and prevents the network to overfit on training data [89], [90], [54]. Data augmentation involves label preserving transformations that are implemented over training samples. Therefore, the augmentation entails a number of image transformation performed with random parameters.

Considering patch extraction detailed in the previous section, we have already implemented some of data augmentation techniques in the procedure used for generating training patches. We showed that the training image first is scaled in a way that a deformation factor in the interval $[0.96, 1.04]$ is applied over the scaled image, hence allowing the network to be more robust to the deformation that might be occurred when the vehicle is capture from a different angle. Moreover, we extract the patch from a random position over the scaled image hence adding variations to the position of the patch in the image. These variations in patch position additionally increase the probability of learning discriminative portion of the vehicle at a specific scale.

In addition to augmentation in the patch extracting stage, during the training, a number of label-preserving transforms are applied as well. The patch is horizontally flipped with the probability of 0.5 during training. Note that the augmentation transforms should simulate the variations which actually happen over different samples of vehicle images. Therefore we prevent vertical flip since it is not the case in vehicle images in a real scenario.

Furthermore, color jittering which adds variations to RGB channels of the image is also utilized during training. Our intuition is that the network also should be invariant to changes in color as it should learn visual patterns rather than colors. Varying brightness and contrast are included during training as well.

3.4 Training

This section describes the training methodology adopted to train architecture described in Sec 3.2. First, we define the loss function which accounts for both make and model classification in Sec. 3.4.1. Then we describe the procedure necessary to train convolutional trunks over patches with different scales in Sec. 3.4.2. The initialization method used for the proposed architecture is detailed in Sec. 3.4.3. Finally training the overall network and related optimization configuration is addressed in Sec. 3.4.4.

3.4.1 Cost Function Formulation

To start with, we define a cost function suitable to minimize a joint make and model classification error of the classifier module. Let us indicate the i-th training sample (e.g., the i-th training image) as x_i. The response of the network to x_i is the output of the classifier module branches y^c and y^f, representing the coarse grained (make) and fine grained (model) vehicle classification respectively. Let us indicate the corresponding expected (target) outputs as t^c and t^f, respectively. We further define the network response (output) to x_i as $y_i = \{y_i^c, y_i^f\}$, and the target output $t_i = \{t_i^c, t_i^f\}$. Next, the cost function for the i^{th} sample is (we omit the i subscript for the sake of readability)

$$J(\theta, y, t) = -\sum_{k=1}^{C_c} t_k^c \log\ (y_k^c) - \sum_{j=1}^{C_f} t_j^f \log\ (y_j^f) + \lambda R(\theta), \tag{3.5}$$

where θ represents the learnable parameters (weights and biases) of the network, λ is regularization term, C_c and C_f are the number of coarse and fine-grained classes. Finally, the term $R(\theta)$ represents an optional regularization term that helps preventing overfitting to the training samples and is defined as the squared L2-norm of all the weights in the network [55] as follows:

$$R(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \|\theta_i\|_2^2 \tag{3.6}$$

Where m is the number of learnable parameters. The defined loss function can be interpreted as the sum of the network classification errors over two recognition tasks. Note that we consider here recognitions over make and model of the vehicle, however it can account for other recognition tasks as well. In Sec. 3.5, we experiment with a case in which recognition over vehicle model and type is explored instead of make. Moreover, the defined loss function has the potential to be modified to include many more recognition tasks over other vehicle attributes such as color, speed, number of doors, etc.

3.4.2 Training the Classifier Convolutional Trunks

As the first step, we separately train W classifier modules over patches that are extracted from the training images. The patches used to train each classifier are extracted at a specific scale with respect to the training image using the procedure described in Sec. 3.3. We recall that each training image is scaled such that its smallest side is equal to the desired value denoted by $H \geq 224$. Then a fixed sized 224×224 patch (equal to network input size) is extracted randomly from the scaled training image. Thus, depending on the value of H, the extracted patch can encompass a small portion of the vehicle capturing detailed visual information (for a large value of H), or it can cover a much larger portion of the vehicle capturing global visual context (for a small value of H).

After the training samples are generated, we can proceed to train a set of classifier modules each over different scales. Each classifier module is trained over a set of

patches extracted from training samples with a particular scale. The architecture of the classifier module is identical to the architecture defined in Sec. 3.2.3 and for the particular case of having one attention window ($W = 1$). Therefore, the classifier contains one convolutional trunk in its architecture operating over a particular scale. The training is carried out using stochastic gradient descent optimization with a base learning rate of 10^{-3} and weight decay of 5×10^{-3}. The learning rate is divided by a factor of 10 after 50 epochs and the training stops when the loss function stops decreasing over validation sets.

That is, we train scale-specific classifier modules so that the convolutional layers in each classifier learn to extract visual representations over a particular scale. For the sake of clarity, let us assume the classifier module that is trained over patches with a scale of 0.5 with respect to training image. The convolutional layers in this classifier hence are trained to capture visual features on this scale such as texture over vehicle wheels or the shape of the headlight. In the contrary, let us consider a classifier module which is trained over a relatively larger scale of 0.9 with respect to training image. Accordingly, the convolutional layers in this classifier are capable of extracting visual features over larger portion hence more global features of the vehicle such as a combination of wheels and headlights altogether.

As we describe in Sec. 3.4.3, convolutional trunks of these classifier modules are then used as pre-trained convolutional trunks in the final step of training. Therefore, the trained convolutional trunks are frozen (*i.e.* parameters in these parts are fixed and are not updated during training) while the other parts of the proposed network are trained (*i.e.*the localizer and last fully connected layers).

3.4.3 Initializing the Network

As the second step and after the classifier convolutional trunks each trained over the desired scale, we can proceed to initialize the parameters (weights, biases) of the network.

Concerning the localizer module, the weights of the convolutional trunk are initialized according to the *Xavier* scheme [31]. For each layer, the value of each weight is independently drawn from a normal distribution with zero mean and standard deviation equal to:

$$\sigma = (2/N_i + N_o)^{0.5} \tag{3.7}$$

where N_i and N_o are the number of inputs and outputs of the layer, multiplied by the width and height of the layer filters respectively.

The weights of the two output branches of the localizer are instead initialized with a different scheme, as follows. Since the initial values of these layers determine the initial scale and position of the attention windows, let us first discuss the optimal setting of these attention windows. Since in most of the images used in this study, the vehicle is located roughly at the center, the best value of the initial position of attention windows would be the center of the image. Considering the initial scale of attention windows, their optimal values would be equal to the patch scales used to pretrain W convolutional trunks in the classifier module as described in Sec. 3.4.2. Now let us define the i-th output of the output layer as follows:

$$y_i = \sum_{j=1}^{N} w_{i,j}.x_{i,j} + b_i, \tag{3.8}$$

where b_i is the bias of the i-th neuron, $w_{i,j}$ is the weight connecting the j-th input to the i-th neuron in the output layer, and N is the number of input features. The weights are set to zero and the biases are set to the optimal initial values of position and scale as discussed above.

Concerning the classifier module, each of the W convolutional trunks are initialized with the parameters learned while pretraining the W classifier modules as detailed in Sec. 3.4.2. To recall, the convolutional trunks are trained over a set of desired scales so that the predict attention windows be able to capture the most representative vehicle parts on the same desired scales. It should be noted that the output branches parameters of pretrained classifiers are not used in the classifier module initialization since they refer to single-trunk architecture. However, since in the proposed architecture the output features by the convolutional trunks in the classifier are concatenated, hence the number of input features to each fully connected layers depends on the number of attention windows(W). Therefore, the output branches of the classifier module are initialized randomly and based on *Xavier* scheme same as localizer fully connected layers.

3.4.4 Training and Optimization

As the final step, when all parts of the proposed network are initialized according to a desired setting, we proceed to train the network end-to-end using the following procedure. Stochastic gradient descent [8] with the momentum of 0.9 and regularization factor $\lambda = 5 \times 10^{-3}$ is used. The cost function defined in Eq. (3.6) is minimized and training is carried out for a total of 200 epochs. All learnable layers in the network, except the convolutional trunks in classifier module, have learning rates of $\eta = 10^{-2}$ divided by 10 every 50 epochs. However, in the classifier module, W convolutional trunks are frozen by setting their learning rate to $\eta = 0$. Therefore, these layers are not updated during training, since they have already been trained using the procedure defined in Sec. 3.4.2. Nevertheless, error gradients are allowed to backpropagate from the classifier to the localizer via STN.

The error backpropagation through the STN module as the rest of the network is carried out using the chain rule. As detailed in Sec. 3.2.2, the STN module takes as input the affine transform parameters predicted by the localizer module and samples the input image at the attention windows using a bilinear filter. The image sampling is a point-wise and differentiable process [47] which allows to backpropagate the error gradients from the classifier to the localizer. First, the error gradients are computed with respect to all classifier parameters and also with respect to each classifier convolutional trunk input (*i.e.* the sampled image at each attention window). Thus, since STN module has performed differentiable sampling, then the gradients with respect to predicted affine transform parameters $\mathscr{T}$ and for each attention window can be computed. Finally, the computed gradients are backpropagated to the localizer fully connected layers and then gradients with respect to all localizer parameters can be calculated accordingly.

The proposed strategy trains the localizer module to predict non-identical attention windows whose scales are aligned with the patch scales used in sec. 3.4.2 to pre-train convolutional trunks in the classifier module. Finally, the classifier output layers predict score maps over two classes of vehicle's make and model. Moreover, we have utilized a number of augmentation techniques including random crop, horizontal flip and color jittering during the training in order to ease the network generalization as detailed in Sec. 3.3.2.

The training procedure described in this section significantly differs from the approach of [47]. In [47], authors first initialized the parameters of both the localizer and classifier modules with ImageNet pre-trained networks. Then, the entire network is trained end-to-end with a moderate (10^{-4}) learning rate in order to guarantee convergence. Such procedure suffices in the context of [47] because its goal is predicting attention windows of identical size (e.g., bounding boxes over characters). However, we experimentally verified that such procedure is not suitable to train the localizer to predict attention windows at multiple scales, which is instead the goal of the present work. The training methodology we presented here guarantees both convergence of the training process and distinct scales of attention windows.

To conclude, the training steps are detailed in Algorithm 1. N_{scales} , N_{images} ,N_{epochs} and $N_{minibatches}$ are total numbers of scales , images, epochs and mini-batches respectively; h_n, w_n are training sample original hight and width; H_s is the desired value used to scaling training samples for scale of s; P_s is set of patches extracted with scale of s; $\theta^{conv}_{classifier_s}, \theta^{fc}_{classifier_s}, \theta^{conv}_{localizer}, \theta^{fc}_{localizer}$ are the parameters of classifier convolutional trunk and fully connected layers, localizer convolutional trunk and fully connected layers respectively; S_0, Pos_0 indicate attention windows initial scale and position; η is the learning rate and m in the length of mini-batch; x is the training image (not cropped at bounding box); and finally, θ presents the union of all network parameters.

3.5 Results

In this section, we report the results of our VMMR experiments over two publicity available datasets of vehicular images. Preliminarily, we experiment with the hyperparameters of each module of our architecture in isolation to maximize the performance of each module. Then, we train our proposed architecture and compare its performance with respect to a number of competing architectures. For our experiments, we consider two challenging datasets of vehicular images collected in different lighting and pose conditions.

Algorithm 1 Training procedure with multi-scale attention windows

Generating patches with desired scales Sec. 3.3
for $s = 1...N_{scales}$ **do**
  for $n = 1...N_{images}$ **do**
    $\alpha \leftarrow \frac{h_n}{w_n}$
    $min(h_n, w_n) \leftarrow H_s$
    $max(h_n, w_n) \leftarrow \alpha^{-1} H\beta$
  end for
end for
Training classifiers over patches of desired scales Sec. 3.4.2
for $s = 1...N_{scales}$ **do**
  $\{Y_s\} \leftarrow classifier_s(\{P_s\})$
  Optimize $\{\theta^{conv}_{classifier_s} \cup \theta^{fc}_{classifier_s}\}$ based on $J(\theta, Y_s, T)$ 3.4.1
end for
Initializing the proposed network Sec. 3.4.3
$\theta^{conv}_{localizer} \leftarrow \mathcal{N}(\mu, \sigma^2(N_i, N_O))$ ▷ Xavier initialization
$\theta^{fc}_{localizer} \leftarrow f(\{S_0\}, \{Pos_0\})$
for $s = 1...N_{scales}$ **do** ▷ Initializing classifier convolutional trunks
  $\theta^{conv}_{classifier_s} \leftarrow \theta^{conv,optimized}_{classifier_s}$
end for
$\theta^{fc}_{classifier} \leftarrow \mathcal{N}(\mu, \sigma^2(N_i, N_O))$ ▷ Xavier initialization
Training the proposed network Sec. 3.4.4
for $s = 1...N_{scales}$ **do** ▷ Freezing classifier convolutional trunks
  $\eta_{\theta^{conv}_{classifier_s}} \leftarrow 0$
end for
for $epoch = 1...N_{epochs}$ **do**
  for $b = 1...N_{minibatches}$ **do** ▷ Performing Refinement
    $\{y_{b...m+b}\} \leftarrow Network(\{x_{b...m+b}\})$
    $J(\theta, y_{\beta_b}, t_{\beta_b}) \leftarrow \{y_{b...m+b}\}, \{t_{b...m+b}\}$
    $\theta_b = \theta_{b-1} - \eta \cdot \nabla_\theta J(\theta, y_{\beta_b}, t_{\beta_b})$ ▷ Fine-tuning parameters
  end for
end for

3.5.1 Stanford Car Dataset

The *Stanford* dataset [52] contains 16,185 vehicular images subdivided into 8,144 training and 8,041 test images. The images are classified according to 196 different vehicle models, where two models may refer to the same vehicle model but differ in the model years. The images are also coarsely classified into 9 different types of vehicles including sedans, SUVs, vans, cabs, coupes, convertibles, pickups,

hatchbacks, and station wagons. In addition to class labels, the images are annotated with the vehicle position in the form of a tight bounding box drawn around the vehicle. In Fig. 3.10 a few samples of car images on this dataset are provided.

3.5.2 CompCar Dataset

The *Comprehensive Cars (CompCars)* dataset [105] contains a large number of vehicular images sourced from the web and surveillance cameras. In our experiments, we rely on the same subset of images used in [105] and [40], containing 16016 training images and 14939 test images. Such images are classified into 431 car models and 75 car makes. For each vehicle model, different model-years are considered as a single class of model. In addition to class labels, the images are annotated with the vehicle position in the form of a tight bounding box drawn around the vehicle. In Fig. 3.11 a few samples of car images on this dataset are provided.

Fig. 3.10 Stanford car dataset.

3.5.3 Optimizing the Localizer Module Architecture

As the first preliminary experiment, we experimentally find the ResNet18 architecture depth that maximizes the performance of the localizer module defined in Sec. 3.2.1. As we explain in Sec. 3.2.1, there is a trade-off between the number of feature maps and their spatial resolution inside the localizer architecture, on localization accuracy. As detailed in Table 3.1, for each additional residual block, the number of feature maps produced as output doubles whereas the area of each feature map drops by a factor of 4.

For this preliminary experiment, we train the localizer to predict one single attention window (W=1) encompassing the entire vehicle. Only for this experiment, we train the localizer to minimize the Mean Square Error (MSE) between the predicted and ground truth attention window size and location rather than minimizing the loss function in Eq. (3.6).

Fig. 3.11 CompCar dataset.

Table 3.1 shows the localizer accuracy as the Intersection over Union between predicted and ground truth vehicle bounding boxes as a function of the localizer depth. The four depth values B, C, D, E in the table indicate the ResNet18 architecture as it is shown in Fig. 3.12 obtained truncating the localizer convolutional trunk after the blocks B, C, D, and E. The localizer accuracy increases from depth B to D but it decreases again for depth E. Our understanding is that an increasing number of feature maps improves the localization accuracy up to a certain point. Beyond such point, the loss in feature map resolution sets off the gain yield by an increased number of feature maps. Best localization accuracy is obtained when the localizer convolutional trunk yields 256 feature maps of size 14x14 (depth D). Therefore, in the remainder of this section, we will consider a localizer module based on the ResNet18 architecture configured with depth D.

Table 3.1 Localization performance over Stanford car dataset.

Localizer Depth	Size of Feature maps	Number of Feature maps	Localizer Accuracy (IoU)
Block B	56×56	64	0.82
Block C	28×28	128	0.84
Block D	14×14	256	**0.89**
Block E	7×7	512	0.86

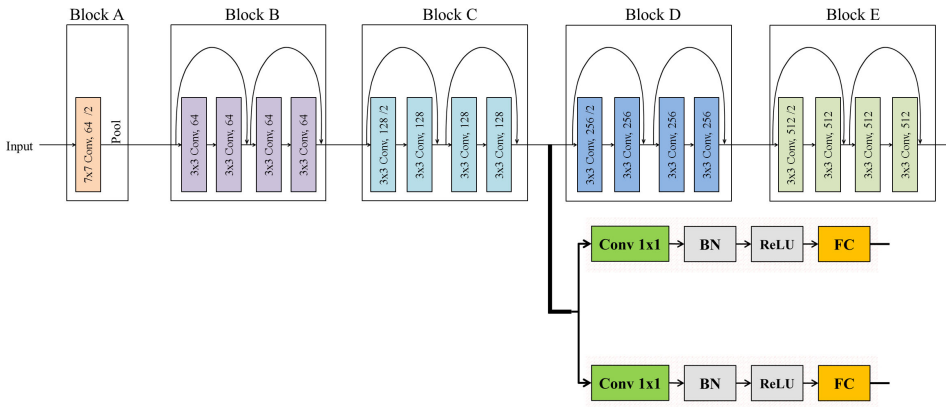


Fig. 3.12 The localizer architecture for different depth, particularly in this figure localizer uses trunked ResNet18 at block C.

3.5.4 Optimizing the Classifier Module Architecture

As a second preliminary experiment, we investigate the optimal architecture for the baseline classifier module described in Sec. 3.2.3. In this section, in addition to investigating the classifier depth effects, we compare standard ResNet [36] and wide ResNet [107] as well as Inception [95] architecture originally used in [47]. For this experiment, we train the classifier to minimize the loss function in Eq. 3.6). However, rather than training the classifier over the attention windows generated by the localizer, in this case we train the classifier over the ground truth boxes encompassing the entire vehicle, since, for now, we are only interested in classification accuracy of different architecture.

Table 3.2 shows the Top1 classification error for vehicle make and model over the Stanford and CompCar datasets. Considering the Inception architecture, due to presence of parallel convolutional layers, depth of the network can not be considered as reliable parameters to analysis Inception architecture. It can be seen from results, the ResNet variants outperform Inception by a considerable margin.

Moreover, considering the standard ResNet architecture, classification error decreases as the network depth increases, except for a slight increase when the depth increase from 101 to 152 layers. Considering wide ResNet architecture, the error decreases to its minimum when the depth increases from 34 to 50 layers. For both datasets, the 50-layers wide ResNet architecture achieves the lowest error on both recognition tasks over both datasets. Such results are in line with the findings of [107] for an image classification task over the ImageNet dataset, indicating that the wide ResNet of 50 layers outperforms the standard ResNet for much deeper topologies.

Notice that in [107] no classification results are provided for wide ResNet with more than 50 layers due to *computational reasons* (we assume, memory limitations). Nevertheless, here we train wide ResNet of 101 and 152 layers with the same widening factor of two as reported in the table. However, due to memory constraints, we had to use smaller mini-batches of 4 images at training time due to memory constraint.

Table 3.2 shows that the classification error increases when the wide ResNet depth increases above 50 layers. We provide two non-exclusive explanations to such evidence. First, our experiments showed that reasonably large minibatches are needed

to effectively train a ResNet, and in order to train 101 and 152 layers wide ResNets we had to decrease the mini-batch size. We conjecture that if memory constraints could be addressed, larger minibatches may have further improved the deeper wide ResNet architectures. Second, the increased number of network parameters is not matched by an increase in the training samples, thus over-fitting may have occurred and more effective regularization techniques are required. Since the wide ResNet-50 architecture achieves the minimum classification error, we choose such network architecture for the classifier module of the proposed architecture.

Table 3.2 Top-1 [%] classification error for different classifier module architectures , depths and batch size trained over patches with scale of 0.95 w.r.t. training samples.

		Batch	Stanford		CompCar	
Network	Depth	Size	Model	Type	Model	Make
Inception	N/A	32	11.34	6.03	10.59	6.17
ResNet	34	32	10.44	5.12	9.18	5.44
	50	32	8.58	3.49	6.35	2.76
	101	32	8.25	3.08	6.08	2.57
	152	32	8.28	3.71	6.52	2.85
Wide	34	32	10.16	3.89	8.64	3.94
ResNet	50	32	**7.45**	**3.07**	**5.19**	**1.58**
	101	4	8.55	3.19	8.06	3.61
	152	4	9.29	3.67	8.22	3.36

3.5.5 Training a Baseline Classifier

As a third preliminary experiment, we investigate the optimal attention window scale for training a baseline classifier module to be used as a reference later on. The wide ResNet-50 architecture which achieved minimal classification error in the previous experiments is trained over various patch scales extracted from the Stanford and CompCar dataset.

Table 3.3 shows the Top-1 classification error for vehicle make and model over patches with different scales with respect to the original input image. The scale value indicates the ratio of patch width and height to those of the image. Considering the scale values, we choose 0.95 as upper bound to retain the ability to perform data augmentation at training time by cropping image patches at random position.

Table 3.3 Top-1 [%] baseline classifier error for different patch scales.

Scale (σ)	Stanford		CompCar	
w.r.t. image	Model	Type	Model	Make
1	7.85	3.52	6.24	1.61
0.95	**7.45**	**3.07**	**5.19**	1.58
0.85	7.52	3.44	5.38	1.54
0.75	8.12	3.66	6.35	**1.45**
0.65	9.25	4.54	7.89	1.62
0.55	10.45	5.26	10.03	1.52

Moreover, since patches with a scale lower than 0.50 do not provide meaningful visual context, the minimum value is set to 0.50. In the end, the values are chosen in this interval and with the step of 0.1.

As the results reveal, the classification error on Stanford dataset and over model and type and also on Compcar dataset over model is minimized for the scale of 0.95 and increases as the scale decreases. However this trend is not observed on Compcar dataset and over make where the scale of 0.75 yields the best result. Hence, as the results imply, the scale on which captured visual context is more representative depends on the classification task.

Furthermore, our observation from results showed that classifiers trained over different scales correctly classify different subsets of each dataset (or, equivalently, perform different mistakes). For instance, the classifier trained over a scale of 0.65 may correctly classify a set of images that are incorrectly labeled by the same classifier trained on a scale of 0.95. This indicates that each classifier predicts the image according to the extent of visual details which it is trained on. However, combining visual features from different scales trough multiple attention windows reduces the classification error as we will show later on.

3.5.6 Training the Proposed Architecture

In this section, we experiment with our proposed architecture over a different number of attention windows W and different combination of scales. Namely, we experimentally find the scale values that minimize the classification error and for $W \in \{1,2,3\}$ attention windows for the same scale values considered in the previous experiment.

Table 3.4 Classification error over vehicle model and on Stanford car dataset using proposed systems of one attention window ($W = 1$), two attention windows ($W = 2$) and three attention windows ($W = 3$). In each system the scale of i-th attention window varies in the set of $\sigma_i \in \{0.95, 0.85, 0.75, 0.65, 0.55\}$.

Scale (σ_i) w.r.t. image	$W = 1$ scale [σ_1]	$W = 2$ scale [**0.95**,σ_2]	$W = 3$ scale [**0.95**,**0.75**,σ_3]
0.95	**7.06** %	7.03 %	5.42 %
0.85	7.08 %	6.27 %	5.40 %
0.75	7.29 %	**5.24** %	5.39 %
0.65	8.18 %	5.80 %	**5.36** %
0.55	8.99 %	5.97 %	5.41 %

Fig. 3.13 Predicted attention windows over a sample from Stanford dataset (left) and a sample from Compcar datasets (right) and at five scales of 0.95, 0.85, 0.75, 0.65, 0.55.

Fig. 3.14 Predicted attention windows for $W = 1$ (top row), $W = 2$ (middle row), $W = 3$ (bottom row) on Stanford dataset.

In this experiment, we follow the complete training procedure as described in Sec3.4.4, minimizing however only the error on model classification. Due to the

computational complexity associated with exploring multiple combinations of multiple scales, we rely on an iterative approach restricted to the Stanford dataset as follows. Therefore, if S is the number of scale values intended to be explored, the proposed experiment has a complexity that grows with $S \times W$, which is far lower than the complexity of exploring the complete combination of scale values, which amounts to S^W.

First, we experiment with $W = 1$, searching for the σ_1 that minimizes the classification error. The second column of Table 3.4 shows that $\sigma_1 = 0.95$ yields the lowest error. A comparison with Table 3.3 related to the baseline classifier trained on entire images rather than attention windows, we see that the attention windows-based architecture outperforms its corresponding baseline architecture for the same W by a considerable margin. Such improvement is achieved by providing the classifier the most discriminative part of the image at a specific scale as found by the attention window.

Fig 3.13 shows the predicted attention windows of different scale on a sample from Stanford as well as a sample from CompCar dataset. While the first row of Fig 3.14 and Fig 3.15 visualizes the predicted attention window by proposed system with $W = 1$ and over a same scale of $\sigma_1 = 0.95$ on different samples for Stanford and Compcar datasets respectively.

Next, we experiment with $W = 2$ and $\sigma_1 = 0.95$, searching for the σ_2 that minimizes the classification error. The third column of Table 3.4 shows that $\sigma_2 = 0.75$ yields the lowest error. Most importantly, the results indicate that aggregating visual representations of two distinct scales via two attention windows significantly reduces the classification error (-1.82 %). Such evidence can be seen in the second row of Fig. 3.14 and Fig 3.15, where sample attention windows for $\sigma_1 = 0.95$ and $\sigma_2 = 0.75$ are shown respectively on Stanford and Compcar datasets.

Finally, we experiment with $W = 3$ and $\sigma_1 = 0.95$ and $\sigma_2 = 0.75$, searching for the σ_3 that minimizes the classification error. The fourth column of Table 3.4 shows that σ_3 minimizes the classification error for $W = 3$. However, when comparing the case $W = 3$ with the case $W = 2$, we see that adding a third attention window doesn't improve the results. That is, when the visual features over different scales represent the same context, accuracy does not increase any further. The predicted attention windows for $W = 3$ are presented in the third row of Fig. 3.14 and Fig

Fig. 3.15 Predicted attention windows for $W = 1$ (top row), $W = 2$ (middle row), $W = 3$ (bottom row) on Compcar dataset.

3.15 for $\sigma_1 = 0.95$, $\sigma_2 = 0.75$ and $\sigma_3 = 0.55$ over Stanford and Compcar datasets respectively.

Table 3.5 Top-1 [%] classification error of our proposed system for different combinations of attention window scales ([0.95], [0.95,0.75], [0.95,0.75,0.65]).

	Stanford		CompCar	
	Model	Type	Model	Make
Proposed ($W = 1$)	7.06 %	2.82 %	4.00 %	1.31 %
Proposed ($W = 2$)	**5.24 %**	**1.67 %**	**2.25** %	0.55 %
Proposed ($W = 3$)	5.36 %	1.81 %	2.31%	**0.39** %

We now consider an architecture with W = 1, 2 and 3 with fixed scales of 0.95, $\{0.95, 0.75\}$ and $\{0.95, 0.75, 0.65\}$ respectively and measure the performance on joint classification. Table 3.5 reports the related results over Stanford and Compcar datasets individually. As seen in the previous table, the best performance for both classification tasks over Stanford car dataset is obtained for $W = 2$. Only for the Compcar dataset, the architecture with $W = 3$ outperforms the architecture with $W = 2$ and on vehicle makes only.

As the number of attention windows W increases, the number of input features to the classifier fully connected layers increases as well. This large number of input

features can significantly increase the possibility of overfitting in the network. If the visual representations captured by the additional attention windows do not provide representative features to the classifier for a specific classification task, this degrades the accuracy as it is observed for model and type on Stanford and model on Compcar dataset. Whereas for the make classification on Compcar dataset, the third attention window contributes to lower classification error.

3.5.7 Comparison with State-of-the-art

In this section, we finally compare our proposed VMMR architecture tuned as described in the previous sections with several competing techniques over the Stanford and CompCar datasets.

Table 3.6 Classification accuracy on vehicle model over Stanford dataset.

Architecture	Accuracy on Model	Accuracy on Type
Chai *et al.* [11]	78.0 %	-
FV-CNN [33]	82.7 %	-
Bilinear-CNN [60]	91.3 %	-
Faster R-CNN	92.4 %	96.6 %
Krause *et al.* [52]	92.8 %	-
SWP-CNN [40]	93.1 %	-
Proposed - Baseline	92.5 %	96.9 %
Proposed ($W = 1$)	92.9 %	97.2 %
Proposed ($W = 2$)	**94.8** %	**98.3 %**
Proposed ($W = 3$)	94.6 %	97.7 %

Table 3.6 shows the make and model classification accuracy over the Stanford dataset for our proposed architecture for a different number of attention windows W. A number of different reference architectures are considered for comparison purposes. The baseline classifier refers to the classifier module trained as in Sec. 3.5.4. The figures for the references [11, 33, 60, 40] are extracted from the respective articles: notice that for such references only model accuracy was provided. In addition, we implemented the faster R-CNN [81] architecture and retrained it for the VMMR task. The classifier network used within the faster R-CNN is the same baseline classifier we previously trained. The faster R-CNN Region Proposal Network (*RPN*) is the standard VGG-16 pretrained on ImageNet, pruned at layer conv5_3 that we refined

for 60 epochs using SGD. The RPN was trained using 2 anchors corresponding to two different object scales with a landscape aspect ratio. Similar results are obtained using ResNet101.

Our proposed system accuracy tops 94.8% for W=2 attention windows, in line with the results in Table 3.5. The proposed system exhibits a 2.3% improvement in classification accuracy over our baseline classifier. Such result is due to the distinct attention windows at different scales, whereas the baseline classifier operates on the entire image.

Our proposed system outperforms by 1.7% the closest competitor, i.e. the convolutional architecture with spatially weighted polling *SWP-CNN* [40]. In our approach, the attention windows are in fact predicted as a function of the particular input image. Conversely, *SWP-CNN* relies on a large number of learned pooling masks that are the same for every image.

Finally, our proposed approach outperforms faster R-CNN by 2.4% on model and 1.7% on make: such gain we hypothesize is due to multi-scale classification schemes of our proposed network.

Table 3.7 Classification accuracy on vehicle model over CompCar dataset.

Architecture	Accuracy on Model	Accuracy on Make
Yang *et al.* [105]	76.7 %	82.9 %
BoxCars [91]	84.8 %	-
SWP-CNN [40]	97.6 %	99.3 %
Proposed - Baseline	94.8 %	98.4 %
Proposed ($W = 1$)	96.0 %	98.7 %
Proposed ($W = 2$)	**97.8** %	99.4 %
Proposed ($W = 3$)	97.7 %	**99.6** %

Next, Table 3.7 shows the corresponding results for the CompCar dataset. Our proposed system achieves top performance for W=2 concerning model classification and for W=3 for make classification, in line with the results in Table 3.5. Considering the baseline classifier, our proposed system improves the accuracy by 3% and 1% on vehicle model and make respectively due to the use of attention windows. Considering the closest competitor, the spatially weighted pooling architecture [40], the proposed system yields a 0.2% gain on model classification and a 0.3% increase for make classification.

3.5.8 Single Attention Window as Localizer

Fig. 3.16 Single attention window as a localizer over samples of Stanford dataset.

Thus far, we assumed that training images are annotated with the vehicle position, i.e. that a vehicle bounding box is provided for each image. In this section, we experiment with a more challenging scenario where no bounding boxes are available, i.e. each image is annotated with the class label only. Our experiments with the baseline classifier confirmed that when the classifier module is trained over whole images, classification accuracy worsens due to the irrelevant background shown to the classifier. Therefore, we now experiment retraining our proposed architecture in Fig. 3.3 over whole training images rather than on bounding boxes. Since lacking bounding boxes the vehicle scale is not known, we are unable to train multiple attention windows at different scales and so we consider a single attention window only (W=1). Accordingly, the attention window functions as a vehicle localizer in the image.

Since there is no need to train classifier modules at different scales, we train our architecture end-to-end in one single step rather than following the procedure described in Sec. 3.4 without providing pre-trained classifiers.

However, our preliminary experiments show that if the system is trained in this way it will not be able to accurately localize the car inside the image and most likely training will overfit on the training set. This results from learning irrelevant information such as background during training and since localizer is trained by minimizing the classification error, it would not be able to distinguish the car from the background.

Fig. 3.17 Adding random background to penalize more inaccurate prediction of attention window.

To address such issue, we deploy an augmentation technique in which car images are put into random backgrounds during training in order to add much larger variation to car position and scale with respect to the image as shown in Fig 3.17. Thus, the classification accuracy will be penalized more if the localizer does not perform accurately.

Moreover, to make training converge, we choose a learning rate of 10^{-3} for the convolutional trunks of localizer and classifier modules, 10^{-4} for localizer scale branch, 10^{-5} for localizer position branch and 10^{-2} for classifier output branches. The network is trained for 200 epochs and every 50 epochs the learning rate is divided by a factor of 10.

Table 3.8 Top-1 classification error over the Stanford dataset without bounding boxes annotations.

Architecture	Model	Make
Proposed - Baseline	17.32%	9.63 %
Proposed - 1 A.W.	**7.70** %	**4.12** %

Table 3.8 shows the make and model Top-1 classification error over Stanford datasets for our baseline classifier and for our proposed system trained as detailed above. Our proposed system shows a 9.6% improvement on model classification and 5.5% improvement on make classification accuracy over the baseline classifier. Fig. 3.16 shows the predicted attention window for a few Stanford dataset images. Despite the network was trained without the aid of bounding boxes, the localizer module succeeds in localizing the vehicle from the background, explaining the gains over the baseline classifier.

Chapter 4

Satellite Image Segmentation on Heterogeneous Datasets

In this chapter, we address the semantic segmentation problem of satellite images on heterogeneous datasets. We should note that semantic image segmentation also referred to as supervised image classification or pixel-based image classification in remote sensing literature, however, for brevity, we referred to this problem as image segmentation in the following. In this chapter, first, in Section 4.1, we discuss the recent literature in the field and the major shortcomings of existing approaches. Section 4.2 describes our proposed architecture, while Section 4.3 detailes the procedure used to prepare training samples. Next, Section 4.4 describes the related training procedure. Section 4.5 presents two domain adaptation methods to improve the performance of a trained network over specific images. Finally, in Sec. 4.6 we experimentally assess the performance of our proposed architecture over three distinct datasets of satellite images.

4.1 Related Work

Automatic segmentation of hyperspectral satellite images has been the subject of extensive studies over the past decade. Mainstream approaches rely on manually designing class-specific features extractors where the extracted features are further classified for image segmentation. Morphological index and Pixel Shape Index (PSI) [4, 43, 111] are among the best known families of hand-crafted features proposed in the literature. Concerning feature classification, discriminative learning [48, 110, 109] aims at discovering the informative subspace within the feature domain to improve the classifier performance. Concerning feature classification, mainly the support vector machine [67] is deployed to process the generated features and perform final decision over pixels of the image.

As already discussed in previous chapters, CNNs have pushed the frontier in many computer vision tasks advancing state-of-the-art by a considerable margin. In the wake of this success and fostered by the availability of large sets of annotated images and leveraging the computational capabilities of modern GPUs, a number of approaches based on CNNs have been proposed recently for image segmentation in general as well as satellite image segmentation in particular. To begin with, first, we describe prominent state-of-the-art approaches employed in the general image segmentation task.

Farabet *et al.* [20] proposed a multi-scale CNN to address scene labeling. Their proposed architecture consists of three convolutional branches each operating on a specific scale of input image generated by a Laplacian pyramid. The output feature maps of these branches are scaled up to recover the original size of the input and then concatenated over the three scales. Finally, the segmentation results of convolutional branches are used by a super-pixel algorithm or conditional random field in order to predict the final segmentation and to enforce spatial consistency.
Long *et al.* [62] introduced a fully CNN for image segmentation which can process an input of arbitrary size. In the proposed scheme, coarse feature maps from deeper layers are combined with those in early layers and with finer resolution which contributes to more precise segmentation.
In particular the specific yet related domain of medical imaging, Ronneberger *et al.* [82] devised an architecture called *U-Net* composed of a contracting branch consisting of convolutional layers and a symmetric expanding branch including deconvolution layers. The contracting path processes the input image through

convolution and pooling layers producing coarse feature maps. In the expanding part, these feature maps are scaled up using deconvolution operations to match the input size and produce the score maps over segmentation classes. Skip connections are used to help the flow of information between these two parts contributing to precise fine segmentation.
Ghiasi *et al.* [28] deployed a similar multi-resolution reconstruction architecture built upon Laplacian pyramid. The coarse feature maps are refined through the reconstruction branch by fusing feature maps with the information of early layers in the network.
Chen *et al.* [12] introduced *Deeplab*, a CNN in which atrous convolution is employed to address image segmentation. Atrous convolution helps to enlarge the field of view of feature maps while keeping their resolution the same. Moreover, atrous spatial pyramid pooling is employed to add multi-scale content to feature maps. In the end, the authors implemented fully connected conditional random fields to refine CNN outputs and perform the segmentation.

Pushed by the success scored with general images segmentation, a number of methods have been studied particularly to address satellite image segmentation which; in following we describe some of the most common approaches.

Authors in [50] addressed semantic labeling over Vaihingen city using patch-based CNN and also fully convolutional architecture. To overcome the imbalanced classes, authors introduce a cross-entropy loss function weighted by median frequency balancing which results in better performance in less frequent classes like cars.
In [99] a downsample-then-upsample architecture similar to [82] is devised utilizing deconvolutional layers in order to tackle semantic labeling of Vaihingen and Potsdam cities. Good performance is achieved over the validation set.
A multimodal architecture operating on the infrared, red, green and digital surface model with multi-scale encoders is proposed in [3] which advances the state-of-the-art over Vaihingen dataset by fusing segmentation results obtained on three scales.
[64] presents an edge-detection network to combine the prediction of class boundaries with the segmentation score maps to increase the segmentation precision. They evaluated their approach over Vaihingen and Potsdam datasets and improved the baseline performance.
In [75] hand-crafted features are fused with features predicted by a CNN architecture,

and a conditional random field inference is employed to make final predictions.

Although most of the studies yield sufficiently accurate segmentation maps, few of them address the problem of deploying the trained network over images with different statistics from those used for training. The difference between training samples and test images is very common in satellite image segmentation scenario and are mainly associated with changes in acquisition condition and most prominently it is present when training and test images come from different geographical locations. Nonetheless, most proposed CNN models are required to be trained on the annotated samples that come from the image under study. However, this is not applicable due to two reasons: first, the annotation of every image under study is costly particularly considering images which cover large areas, secondly, in most applications, the segmentation has to be performed in short time interval preventing time-consuming annotation and training a CNN from scratch.
Therefore, recently a number of domain adaptation techniques have been proposed to address the aforementioned issue of dataset shift between training and test images. Some approaches rely on selecting a subset of features which are invariant to the shift in domain [9, 78]. Other approaches focus on data distributions of target and source domain and they aim to make these domains statistically similar to keep the classifier unchanged [45, 72, 66, 104]. Other methods rely on adaptation of classifier rather than data distributions across domains [80, 65]. In addition, inspired by the generative adversarial networks (GAN) [32], some recent studies have been carried out in order to learn representations which are invariant to domain shift with the aid of an additional adversarial term to the total cost function.[23, 17]. Whereas domain adaptation techniques have been proven successful in image classification, they have received less attention for satellite image segmentation, leaving the covariate shift issue largely open.

In this work, we tackle the segmentation problem of satellite images designing a network capable of learning high-level semantic features that are more robust to image variation and hence can generalize over a wider range of satellite images. Moreover, we devise methods to further improve its performance with respect to each image to be segmented.

4.2 Proposed Architecture

Our proposed architecture is composed of one encoder network paired with a decoder network which processes the input image in top-down bottom-up manner as shown in Fig. 4.1. The encoder takes as input the image to be segmented and generates a number of feature maps on different semantic level and with different resolution. While the decoder takes as input such feature maps and outputs a *segmentation map*, where each pixel is labeled according to one of the possible land usage classes. In the following, we detail the architecture of each network part and the operations of the encoder and the decoder in detail.

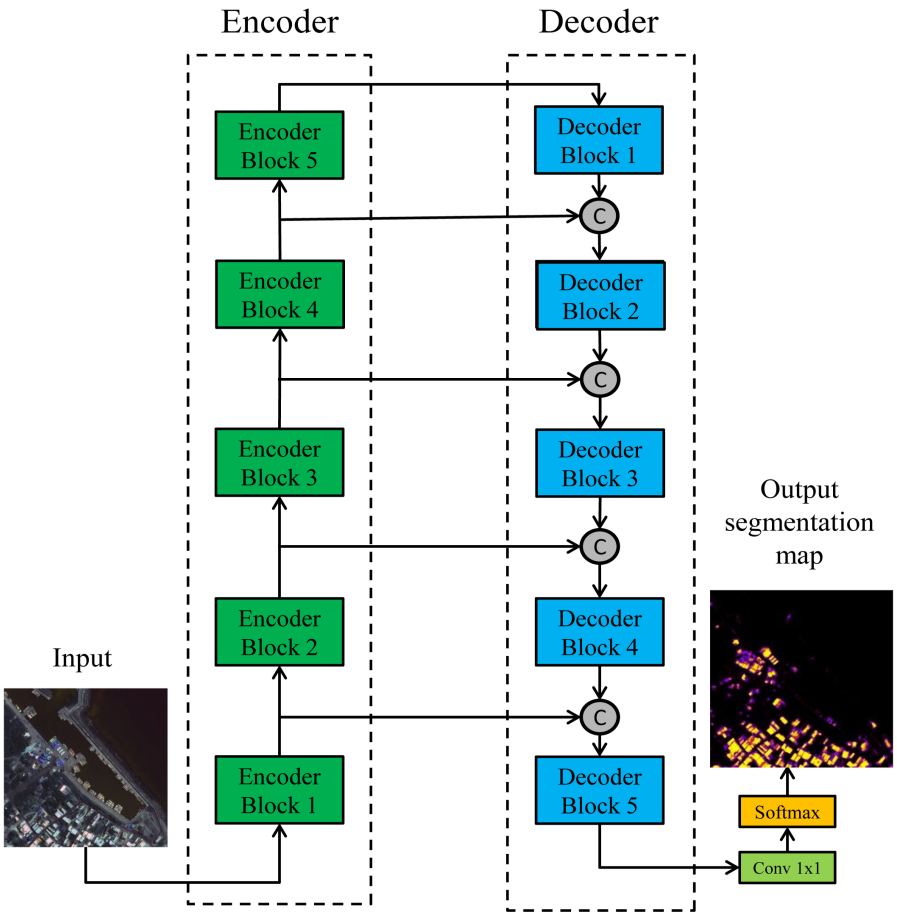


Fig. 4.1 Proposed encoder-decoder convolutional architecture for satellite image segmentation.

4.2.1 Encoder

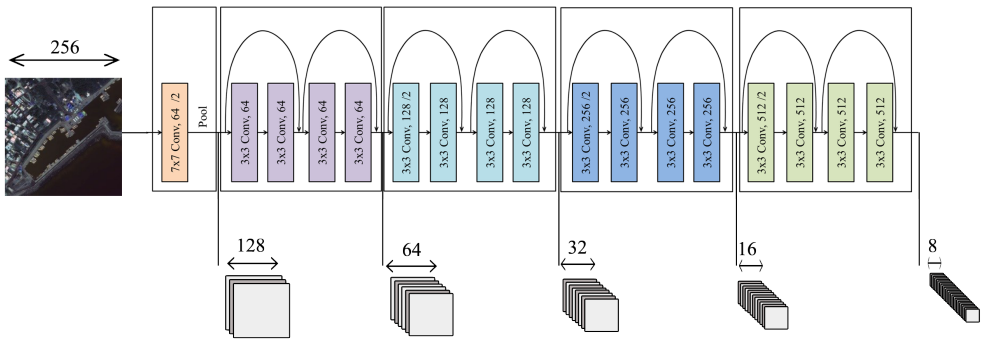


Fig. 4.2 Encoder architecture with depth of 18 and input size of 256 by 256 is visualized.

As it is provided in Fig. 4.2, the encoder is composed of five residual blocks. As it has been discussed in section 2.9, we recall that the first convolutional layer of each block of a ResNet has a stride of two pixels. Therefore, the resolution of output feature maps at each block is halved with respect to input feature maps. On the contrary, as the feature maps processed by each block, the number of generated feature maps in output increases with respect to the number of input feature maps.

For clarity of exposition, let us consider the case where an encoder with a depth of 18 convolutional layers is employed. Let the input image be a $256 \times 256 \times D$ image, where D is the number of spectral or color channels. With such setting, the first block in encoder takes as input the $256 \times 256 \times D$ image, and in output, 64 feature maps of resolution 128×128 are generated. Next, the second encoder block takes as input such feature maps and produces accordingly 64 feature maps sized 64×64 in outputs. Following the same rule, 128 feature maps sized 32×32 and 256 feature maps sized 16×16 are output in third and fourth encoder block respectively. In the end, the fifth and last block of encoder produces 512 feature maps each sized 8×8. Table 4.1 details the number of feature maps output by each block and the number of convolutional layers in the block for encoders with different depths.

Therefore, by proceeding to deeper layers in the encoder, the number of output feature maps increases hence enabling capturing higher-level visual representations. In addition, the field of view of feature maps increases at each block as well, allowing learning visual representations across wider spatial ranges. On the contrary, as we proceed to deeper encoder layers, the resolution of feature maps decreases thus producing more coarse feature maps in deeper encoder blocks.

Table 4.1 Number of convolutional layers, output feature maps and their spatial size for each encoder block and for different depths.

Depth		Block 1	Block 2	Block 3	Block 4	Block 5
18	conv.	1	4	4	4	4
	feat.	64	64	128	256	512
	size	128×128	64×64	32×32	16×16	8×8
34	conv.	1	6	8	12	6
	feat.	64	64	128	256	512
	size	128×128	64×64	32×32	16×16	8×8
50	conv.	1	9	12	18	9
	feat.	64	256	512	1024	2048
	size	128×128	64×64	32×32	16×16	8×8
101	conv.	1	9	12	69	9
	feat.	64	256	512	1024	2048
	size	128×128	64×64	32×32	16×16	8×8
152	conv.	1	9	24	108	9
	feat.	64	256	512	1024	2048
	size	128×128	64×64	32×32	16×16	8×8

Relying only on coarse feature maps generated at the last encoder block can result in blurred segmentation, hence output feature maps in all encoder blocks are utilized in the decoder in order to leverage all information available and combine the feature maps of various resolutions and semantic levels and also with different fields of view. Therefore, a number of shortcut connections are implemented between encoder and decoder networks which will be discussed in the next section.

While the encoder design pattern follows that of a ResNet, the proposed encoder differentiates from standard ResNet in a number of noticeable aspects. First, the pooling layer found in standard ResNets after the 5-th block is omitted to avoid an unneeded loss of spatial information. Second, we obtain a fully convolutional architecture by dropping the fully connected layer found in standard ResNets. In principle, this allows the network to efficiently process input images of arbitrary size without the need for shifting and stitching. Third and most important, the output of each block is not just provided as input to the following block, but is also provided as input to a specific block of the decoder unit. Providing feature maps extracted at multiple scales helps the decoder to refine its output as detailed in the following.

Our choice to design the encoder around a residual architecture rather than a plain convolutional one is meant to improve the network ability to generalize over

a wider range of images. Our conjecture is that the residual encoder enables the depth required to learn visual representations of high-level which are more robust to statistics variations and as a result are more robust to domain shift between training and test images. Hence, such learned visual representations are the key contributor to improving network performance on novel images. In Sec. 4.6, the connection between residual encoder depth and the network performance will be verified experimentally.

4.2.2 Decoder

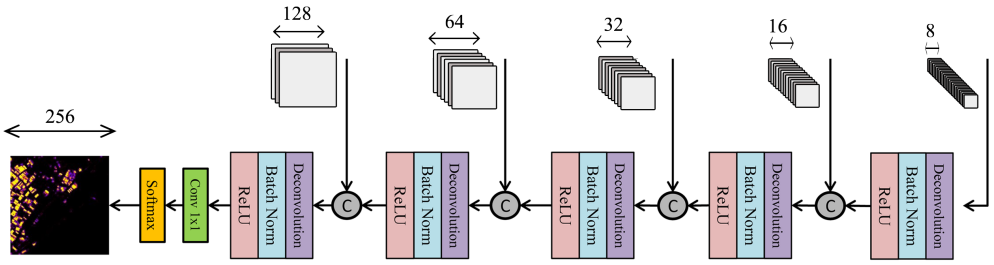


Fig. 4.3 Decoder architecture corresponds to encoder with depth of 18 and input size of 256 by 256.

Table 4.2 Number of deconvolutional layers, output feature maps and their spatial size for each decoder block and for different depths.

Depth		Block 1	Block 2	Block 3	Block 4	Block 5
18,34	deconv.	1	1	1	1	1
	feat.	256	128	64	64	64
	size	16×16	32×32	64×64	128×128	256×256
>34	deconv.	1	1	1	1	1
	feat.	1024	512	256	64	64
	size	16×16	32×32	64×64	128×128	256×256

Fig. 4.3 illustrates the architecture of the decoder, which is composed of five *deconvolutional blocks* symmetric to the five residual blocks of the encoder. Each decoder block includes one deconvolutional layer, one batch normalization [46] layer and ReLU activations, as illustrated in the dotted box in Fig. 4.3.

Deconvolutional layers (*backward convolution*) were originally proposed to address the loss of mid-level cues [108] caused by pooling operators used in convolutional

networks. Each deconvolutional layer contains one or more deconvolutional filters, where each filter can be interpreted as a learnable upsampling function. Deconvolution works in two steps: first, a sparse feature map is generated by interleaving zeros within the pixels, thereby upsampling the input feature maps by a specific factor (*unpooling*). Next, a dense feature map is generated by applying a convolution filter to the sparse feature map. Thus, each decoder block reverses the subsampling operation performed by the first convolutional layer of each encoder block.

Skip connections contribute to more precise and finer predictions as the spatial information of early layers in the encoder is used as well. Thus, output feature maps by each decoder block are concatenated with the feature maps produced by the corresponding encoder block. Notice that the number of feature maps coming from the previous block and those from skip connections remains identical. We experimentally verified that this prevents one group of feature maps from dominating the other when they are concatenated and provided as input to the next deconvolutional block.

For the sake of clarity, we exemplify the operations of a decoder module with reference to an 18-layers encoder. Following the example given in the previous section for the encoder, let the input image be $256 \times 256 \times D$ image, where D is the number of spectral or color channels. Hence, as it is observed, the encoder last block would output 512 feature maps sized 8×8. The 1-st decoder block takes as input such 512 8×8 feature maps then using deconvolutional layer these feature maps are scaled up by a factor of two, reaching a 16×16 resolution. In addition to scaling up the spatial size, deconvolutional layer also performs dimensionality reduction over feature maps. Therefore, regarding this example, 1-st decoder block takes as input 512 8×8 feature maps and outputs 256 16×16 feature maps. Such 256 feature maps are then concatenated with the identically sized 256 feature maps generated by the 4-th encoder block. The rule is same for all decoder blocks and for any depth: the spatial size and number of output feature maps should be identical to output feature maps of corresponding block in the encoder which intended to be concatenated and fed to the subsequent block in the decoder. The 512 concatenated 16×16 feature maps are then provided as input to the 2-nd decoder block, and so forth. The decoder output finally consists of 64 feature maps with a size of 256×256.

Next, the decoder output is processed by a convolutional layer with C filters (where C is the number of land classes) of size 1×1. The output of such layer consists

in C feature maps of size 256×256 pixel: the i-th pixel in the k-th feature map $o_{i,k}$ ($k \in [1, C]$) represents the relative confidence that the i-th pixel in the input image belongs to the k-th class. We are interested in estimating, for each i-th pixel, a class probability distribution over the k classes $y_{i,k}$. For each i-th pixel, the spatial SoftMax produces a normalized *score map* for each k-th class as follows:

$$y_{i,k} = e^{o_{i,k}} / \sum_{k=1}^{C} e^{o_{i,j}} \tag{4.1}$$

such that:

$$\sum_{k=1}^{C} y_{i,k} = 1 \tag{4.2}$$

Finally, each i-th pixel is labeled according to the k-th class that maximizes the pixel score y_i: the 256×256 map of labels is referred to as *segmentation map* in the following.

4.3 Constructing Dataset

Given a dataset of annotated satellite images, it should be first subdivided into three sets as follows.

First, the *training set* which refers to images (or parts thereof) used for training the network is constructed and it usually includes 60 to 70 percent of available images. Then, the *validation set* referring to images (or parts thereof) used to validate the training procedures is prepared to contain 30 to 40 percent of the available dataset. We recall that training and validation images are required to be annotated with proper ground truth of segmentation maps where each pixel is assigned a label according to a set of predefined classes. Moreover, it is necessary that training and validation sets follow similar statistics and probability distributions. Such condition enables accurate evaluation of the network performance during training enabling fine-tuning hyperparameters such as learning rate and weight decay.

The third set of images is *test set* which refers to images representative of those over which the trained network is to be deployed. As such, their statistics may differ even

by a large margin from the statistics of training and validation images. It is common that in case of publicly available datasets, benchmark organizer will not publish the ground truth corresponds to images in the test set.

As it has been discussed earlier, in this study we are investigating CNN based architectures with the purpose of increasing the network robustness to the potential changes which are commonly present between training and test images. To be precise, generally, such condition is defined as dataset shift when the joint distribution between input and out of a model differs in training and test stages [93]. Particularly in this study, we are dealing with a more common case of dataset shift namely covariate shift, where only input distribution varies between training and test sets. It is well-known that the variation in acquisition conditions or mapping locations leads to covariate shift in satellite images [97]. Many datasets that have been studied for satellite image segmentation includes training and test images which come from the same location, hence the covariate shift is small. Nevertheless, few datasets have recently been made publicly available which contain images captured over different locations introducing considerably large covariate shift between training and test set [63].

Commonly the dataset is provided as large images in a scale of thousands pixels over each side. Additionally, some other information such as near infrared band and sometimes digital surface models (*DSM*) is provided as well. Considering the high resolution of current satellite missions, the size of such large images can exceed tens of Gigabytes. Hence, first training, validation and test images required to be subdivided into much smaller tiles. In the following, we provide the detailed procedure employed for extracting training, validation and test tiles.

Concerning the training and by taking to account the available GPU memory, the network size and reasonable mini-batch size of 8, we choose the network input patch size to be 256×256 in training, although it allows any other sizes since the architecture is fully convolutional. Additionally, as discussed in Section 2.8.2, a number of affine transformation with random parameters can be utilized as data augmentation techniques helping the network generalization. To this end, we also implement a number of data augmentations such as cropping at random position, horizontal and vertical flips and rotation with a random angle. However, the augmentation is applied during training and over tiles instead of applying such augmentation over original large images which require more computation and memory resources. Therefore,

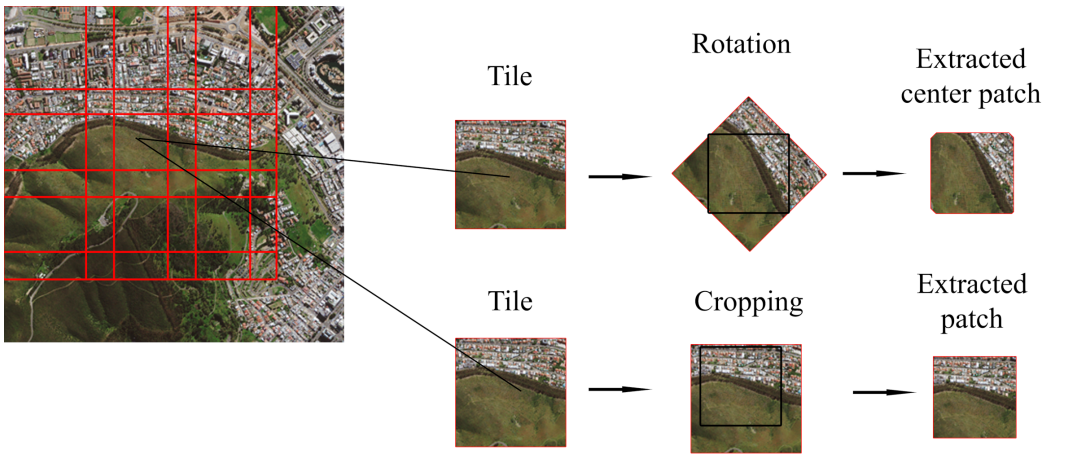


Fig. 4.4 Extracting training tiles (left), rotation (upper right) and cropping (bottom right) are illustrated.

to enable transformations such as rotation with random angle, we consider training tile size to be larger than input patch size. As it is shown in Fig. 4.4, in this way, first the tile is rotated then the input patch is extracted from the center of rotated tile limiting the input patch within tile boundary. The training tile are extracted with size of 364×364 allowing rotation with a random angle θ where $\theta \in (0, 2\pi)$ while keeping the center patch 256×256 within tile boundary in worst case scenario of $\theta = \frac{\pi}{4}$. To conclude, first each training image is subdivided into 364×364 tiles, then, a 256×256 sample is randomly extracted from each tile as follows. Note that these tiles are extracted using a translation of 108 $(364 - 256)$ to ensure all possible patches are used during training. Next, with 50% probability, a 256×256 patch is cropped at a random position from the tile. Otherwise, a 256×256 patch is cropped from the center of the image which has undergone a bi-linear rotation with a random angle θ drawn from a uniform distribution in the interval $\theta \in (0, 2\pi)$. In addition to crop and rotation, horizontal and vertical flips each with the probability of 50% are applied over each tile independently.

Concerning validation images, each image is simply subdivided into 256×256 non-overlapping tiles and no further random alterations are applied to the sample. Note that since augmentation such as random rotation is not applied during validation, validation tiles have the same size as the input patch size of the network.

Concerning test images, we extract 512×512 partially overlapping samples from each test image. Whereas our network design is fully convolutional and allows in

principle to operate over images of arbitrary size, 512 $\times$ 512 was the maximum image size allowed by our memory setup. It should be noted that during training 256 $\times$ 256 is considered as patch size to enable reasonable sufficient samples in each mini-batch improving the training optimization. Whereas in the test stage, we are more interested in spatial consistency across predicted segmentation maps rather than mini-batch size, hence we choose the maximum patch size allowing by memory constraint. Concerning overlapping, we found it to be necessary in order to cope with potential artifacts at the boundaries of the network output.

4.4 Training Methodology

After the training samples are generated according to the procedure defined in the previous section, we proceed to train the proposed architecture. In this section, we first define the loss function to minimize at training time then we detail the related optimization procedures.

4.4.1 Cost Function

The network is trained end-to-end in a fully supervised manner providing the proper segmentation map ground truth.
For clarity of exposition, let θ be the parameters representing the weights and the biases of the network, let x be the sample provided as input to the network, let y be the segmentation map predicted by the network and let t be the expected (*target*) map (i.e., the ground truth). In detail, let $y_{i,k}$ and $t_{i,k}$ indicate the predicted and expected output for the i-th pixel x_i and for the k-th class among C different possible classes. Let t_i take the form of a one-hot vector, i.e. only the element corresponding to the correct class is equal to one, whereas all the other $C-1$ elements are equal to zero. The network is trained by minimizing the loss function:

$$L(\theta, y, t) = -\sum_{i=1}^{H \times W} \sum_{k=1}^{C} t_{i,k} \log\ (y_{i,k}). \tag{4.3}$$

we recall that $y_{i,k}$ is computed using spatial Softmax layer after network output layer:

$$y_{i,k} = e^{o_{i,k}} / \sum_{k=1}^{C} e^{o_{i,j}} \tag{4.4}$$

where $o_{i,k}$ is the i-th pixel in the k-th output feature map.

Such loss function, known also as spatial cross-entropy, represents the network inaccuracy in predicting the segmentation map of the sample x across the C classes. Additionally, to prevent the network from overfitting to training samples, a regularization term $R(\theta)$ is added to loss function, obtaining final cost function

$$J(\theta, y, t) = \eta L(\theta, y, t) + R(\theta) \tag{4.5}$$

where η is the learning rate, i.e. the size of the parameters update step. $R(\theta)$ is the squared L2 norm of all the weights in the network:

$$R(\theta) = \frac{\lambda}{2m} \sum_{i=1}^{m} \|\theta_i\|_2^2 \tag{4.6}$$

where λ is the corresponding regularization factor and m is the number of learn-able parameters.

4.4.2 Training and Optimization

After generating the samples and defining the cost function in Eq. (4.5), we proceed training the network via stochastic gradient descent with an additional momentum of 0.9. Concerning the learning rate adaptation strategy, we chose a base learning rate of 10^{-2} that is divided by factor of 10 every 50 epochs. A factor of 5×10^{-3} is applied to the regularization term in Eq. (4.6). Given the size of the training samples which is equal to 256×256, we train the network with 8 samples in each mini-batch which is the maximum allowed by memory constraint. In our experimental setup, the training ends when the validation error stops decreasing or after 300 epochs.

The training process is summarized in Algorithm 2, where the proposed network is denoted by NN, and x^s, y^s and t^s denote input samples, network outputs and target maps over the training set (source domain) respectively. $\nabla_w J$ denotes network parameters gradients with respect to the cost function, γ is the momentum used in

Algorithm 2 Training process

Training *NN* over training set (source domain)

for $e = 1...n_{train}$ **do** ▷ training over n_{train} epochs
  $y^s \leftarrow NN(x^s)$ ▷ forward pass
  $L(w, y^s, t^s) \leftarrow (y^s, t^s)$ ▷ computing loss
  $J(w, y^s, t^s) = \eta L(w, y^s, t^s) + \lambda R(w)$ ▷ computi. cost
  $\nabla_w J(w, y^s, t^s) \leftarrow J(w, y^s, t^s)$ ▷ backward pass
  $v_e = \gamma v_{e-1} + \nabla_w J(w, y^s, t^s)$ ▷ momentum
  $w_e = w_{e-1} - v_e$ ▷ parameters optimization
end for

optimization and n_{train} is the number of training epochs. Notice that in practice, the training is carried out over mini-batches of samples, however, in order to avoid unnecessary complexity, the mini-batches are not shown in Algorithm 2.

4.5 Domain Adaptation Strategies

In this section, we propose two domain adaptation strategies to improve the performance of a trained network when applied to a specific image to be segmented. The two strategies differ mainly in the required inputs, the first proposed strategy requiring no human intervention, while the second one needs manual image interpretation over a small subset of samples.

4.5.1 Batch Normalization Statistics Refinement

Algorithm 3 Batch Normalization Statistics Refinement

Note: *NN* is first trained according to Alg. 2

for $e = 1...n_{refine}$ **do** ▷ refining over n_{refine} epochs
  for $n = 1...n_{mb}$ **do**
    $\{u_{n...m+n}\} \leftarrow NN(\{x^t_{n...m+n}\})$ ▷ forward pass
    $\mu_{\delta_n} = \frac{1}{m}\sum_{i=1}^{m} u_{n+i}$ ▷ n-th mini-batch mean
    $\sigma^2_{\delta_n} = \frac{1}{m}\sum_{i=1}^{m}(u_{n+i} - \mu_{\delta_n})^2$ ▷ n-th mini-b. var.
    $z_{\delta_n} \leftarrow \{\mu_{\delta_n} \cup \sigma^2_{\delta_n}\}$ ▷ n-th mini-batch statistics
    $z_n = \alpha\, z_{n-1} + (1-\alpha)\, z_{\delta_n}$ ▷ refining statistics
  end for
end for

The first domain adaptation strategy we propose consists of refining the BN statistics learned during training over each image to be segmented. As introduced in Sec. 2.7, BN speeds up the training by normalizing the inputs to each layer activation function throughout a network. Borrowing the notation from [46], let the vector $u = (u^{(1)}, ..., u^{(d)})$ represent the inputs of a layer activation function. The normalized inputs are computed as:

$$\hat{u}^{(k)} = \frac{u^{(k)} - E[u^{(k)}]}{\sqrt{\text{Var}[u^{(k)}]}} \tag{4.7}$$

where $E[u^{(k)}]$ and $\text{Var}[u^{(k)}]$ are computed over each mini-batch of train data. Since the procedure is the same for every activation function (any k), for brevity in the following we omit k e.g. replacing $u^{(k)}$ with u. Next, to prevent the activation functions operating exclusively in their saturated region, the normalized inputs are shifted and scaled as

$$v = \gamma\hat{u} + \beta \tag{4.8}$$

where γ and β are learned independently for each layer.

During training, BN keeps track of computed statistics (i.e. mean and variance), then such stored statistics are used to normalize the activations inputs during evaluation. Let $z_{(n-1)}$ be the statistics tracked at the end of the $n-1$-th mini-batch; at the n-th mini-batch they are updated as

$$z_{(n)} = \alpha z_{(n-1)} + (1-\alpha) z_{\delta_n}, \tag{4.9}$$

where z_{δ_n} are the statistics computed during the n-th mini-batch and α is the momentum.

Under some assumption, the statistics computed at training time can be used to normalize the activation function inputs at deployment time. However, when the network is deployed over data whose statistics do not match those of the training samples, the statistics computed at training time may be useless towards normalizing the activation function inputs. Therefore, we propose an improved BN strategy where after the network is trained, the computed statistics z are preliminarily refined over (a subset of) the image to be segmented. Algorithm 3 details the proposed BN statistics refinement. First, the proposed network is required to be trained over

the source domain according to Algorithm 2. Next, for every image on the target domain (x^t), the trained network is refined without requiring any annotations. Such refinement is carried out for n_{refine} epochs over patches extracted from the image on the target domain. After patches are split into n_{mb} mini-batches with length m, in each iteration a mini-batch of patches ($\{x^t_{n...m+n}\}$) is input to the trained network so that the activation functions inputs can be obtained ($\{u_{n...m+n}\}$) throughout the network. Next, the mean and variance of the activations are computed over the mini-batch (μ_{δ_n}, $\sigma^2_{\delta_n}$) in order to update and refine BN layer statistics according to (4.9) where Z_{δ_n} is the new observed statistics over n-th mini-batch, and Z_{n-1} is the previously updated statistics. As a result, after n_{refine} epochs, BN layers statistics are refined according to statistics of patches over the target domain. In our experiments, we found that the network performance is maximum when the BN statistics are updated for about 10 epochs ($n_{refine} = 10$) with momentum $\alpha = 0.9$, independently over each test image. Further refining the BN statistics has the effect of overfitting to the image area used for updating the statistics, jeopardizing the network performance over the rest of the image.

We would like to point out that this strategy does not require additional image labeling over the target domain, since BN statistics refinement is carried out without computing loss function and without performing back-propagation as shown in Algorithm 3.

4.5.2 Active Learning

The second domain adaptation strategy we propose relies on active learning [80, 49, 65]. In this strategy, a number of patches from each image on the test set (target domain) are first hand-selected and annotated by the user. Then, the annotated patches are used to refine a network previously trained on training images (source domain). The strategy is divided into three steps and is illustrated in Fig. 4.5 and detailed in Algorithm 4.

The first step a) deals with the selection of suitable regions over the test image. Since a satellite image usually covers a large geographical area (e.g. a city and the rural surroundings), land usage classes are not distributed evenly across each image. For example, over an image, some areas may contain just buildings, whereas other areas may contain just vegetation. Furthermore, each land usage class statistics may

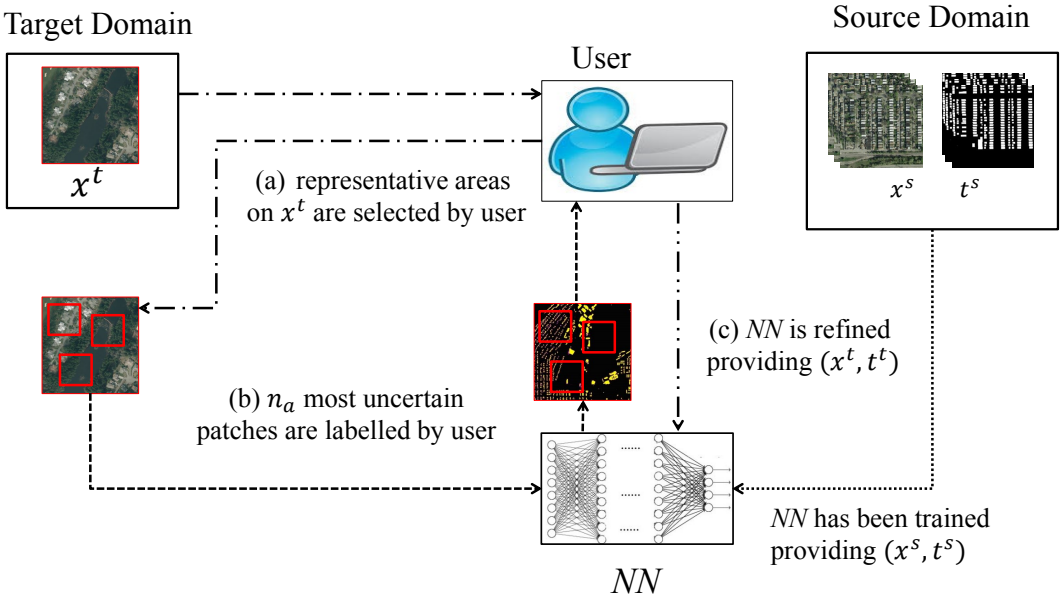


Fig. 4.5 Proposed active learning method depicted over three steps. *NN* stands for the proposed neural network, x^s, t^s, x^t and t^t denote images and target maps over source and target domains respectively.

Algorithm 4 Active Learning

Note: *NN* is first trained according to Alg. 2
Step(a): Selecting representative areas over x_t and extracting patches
$\{x^t_{1...n}\} \leftarrow USER(x^t)$
Step(b): Selecting most uncertain patches
$\{y^t_{1...n}\} \leftarrow NN(\{x^t_{1...n}\})$ ▷ forward pass
$\{uc_{1...n}\} \leftarrow \{y^t_{1...n}\}$ ▷ computing uncertainty
$\{x^t_{1...n_a}\} \leftarrow sort(\{uc_{1...n}\})$ ▷ n_a uncertain patches
Step(c): Labelling n_a patches and refining *NN*
$\{t^t\} \leftarrow USER(\{x^t_{1...n_a}\})$ ▷ labelling patches
for $e = 1...n_{refine}$ **do** ▷ refining over n_{refine} epochs
  $y^t \leftarrow NN(x^t)$ ▷ forward pass
  $L(w, y^t, t^t) \leftarrow (y^t, t^t)$ ▷ computing loss
  $J(w, y^t, t^t) = \eta L(w, y^t, t^t) + \lambda R(w)$ ▷ computi. cost
  $\nabla_w J(w, y^t, t^t) \leftarrow J(w, y^t, t^t)$ ▷ backward pass
  $v_e = \gamma v_{e-1} + \nabla_w J(w, y^t, t^t)$ ▷ momentum
  $w_e = w_{e-1} - v_e$ ▷ parameters optimization
end for

be affected by some internal variance (e.g. buildings in some areas of a city may not look like other buildings in other areas of the same city). Therefore, during step a) the user manually locates image areas that are both representative of the different land use classes and account for at least some of each class internal variance.

During step b), at first, an uncertainty metric is computed for each patch which is extracted over representative areas during the previous step. For example, considering a binary segmentation problem of building-background, the uncertainty of the k-th patch is computed as $uc_k = \sum_i 1 - |y_{k,i,1} - y_{k,i,2}|$, where $y_{k,i,j}$ is the pretrained network score for the j-th class of the i-th pixle of the k-th patch. Metric uc_k indicates how much the network is uncertain about the classification of the k-th patch. Patches over which the network is more uncertain about the pixel classes are in fact more useful for active learning [97]. Thus, the top-n_a patches with higher uncertainty are selected to be hand-annotated by the user, for example using a graphical tool for image segmentation.

Finally, during step c), the network is refined over the n_a patches labeled by the user. The refinement consists in further optimizing the network parameters over the hand-labeled patches (x^t, t^t) according to the training procedure defined in Sec. 4.4.1. Therefore, the network can be optimized over a set of informative samples on the test image itself, finally closing the semantic gap between training and test samples.

In our experiments, the refinement employs a more conservative learning rate of 10^{-4} and only for 30 epochs. Due to the relatively small number of annotated patches, high learning rates or long training may lead to overfitting to the selected patches, jeopardizing the ability of the refined network to generalize well over the rest of the test image. Note that in the first domain adaptation method, only BN statistics are refined which does not require annotations over the test image, however in the second approach, network parameters along with BN statistics are both optimized providing labels over test image.

4.6 Experiments and Results

In this section, we evaluate our proposed architecture over two public and one homegrown dataset of high-resolution satellite images. Whenever possible, we compare our results with state-of-the-art references.
The experimental setup used for all the experiments below consists of a 12-cores Intel server with 128 GB of CPU memory and four NVIDIA GeForce GTX 1080 Ti GPUs with 11 GB of memory each.
The interested readers will find more information about the implementation in the GitHib repository [1] where all codes necessary to generate the presented results have been made publicly available.

4.6.1 Evaluation Metrics

The predicted segmentation maps are evaluated using a set of metrics which will be defined in the following.

Precision

Precision is defined as the fraction of predicted positive pixels which is detected correctly.

$$Precision = \frac{tp}{tp + fp} \tag{4.10}$$

where tp , fp are correctly predicted and inaccurately predicted pixels respectively regarding a segmentation class. Considering binary image segmentation, precision is computed for one class while for multi-class segmentation precision is computed for every segmentation class.

[1] https://github.com/sinaghassemi/semanticSegmentation

Recall

Recall is defined as the fraction of labeled positive pixels which is detected accurately.

$$Recall = \frac{tp}{tp + fn} \tag{4.11}$$

where tp , fn are correctly predicted pixels and inaccurately missed pixels respectively regarding a segmentation class. Considering binary image segmentation, same as precision, recall is computed for one class while for multi-class segmentation recall is computed for every segmentation class.

F1-Score

F1-Score is defined as the harmonic mean of precision and recall.

$$F1 = 2 \cdot (precision \cdot recall)/(precision + recall) \tag{4.12}$$

same as recall and precision, F1-score is computed for each segmentation class.

Intersection over Union

Intersection over Union (IoU) is the ratio of correctly predicted area to the union of predicted pixels and the ground truth for each class as shown in Fig. 4.6.

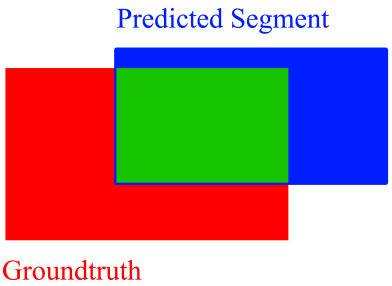


Fig. 4.6 Intersection (green area) over union (the whole colored area).

Overall Accuracy

Overall accuracy (Acc) is the fraction of all pixel population which are correctly predicted for all classes.

$$Acc = \frac{\sum_{i=1}^{N_{class}} tp_i}{N_{pixels}} \tag{4.13}$$

where, N_{class} and N_{pixels} are number of classes and pixels respectively and tp_i denotes true positive regarding class i.

4.6.2 Buildings Dataset

The first dataset we consider for our experiments is composed of nine high resolution images acquired by three different Earth observation satellite sensors, WorldView-2 , Pléiades-1A and Pléiades-1B and over nine different urban areas worldwide. The nominal spatial resolution of the images is 50 centimeters and each image includes 4 bands: red, green, blue and near infrared. Moreover, the images are not acquired from a perfect nadiral angles. Areas (cities) B1, B2, B3, B4, B5, and B6 have been chosen for training and validation , whereas areas (cities) B7, B8 and B9 are reserved for testing. That is, the network is tested over cities that are different from the cities used for training, which introduces a particularly challenging covariate shift scenario. Table 4.3 shows a detailed description over each image while Fig. 4.7 illustrated some samples from building dataset.

Table 4.3 Image description on buildings dataset.

Image	Size (pixels)	Resolution (meter)	Spectral bands	Satellite sensor	Off-nadir angle	Set
B1	15000 ×14000	0.5	R,G,B,IR	Pleiades-1A	26.5°	train & val
B2	14000 ×11000	0.5	R,G,B,IR	Pleiades-1A	23°	train & val
B3	14000 ×16000	0.5	R,G,B,IR	Pleiades-1B	24.7°	train & val
B4	23000 ×15000	0.5	R,G,B,IR	Pleiades-1B	18°	train & val
B5	25000 ×17000	0.5	R,G,B,IR	WorldView-2	17.4°	train & val
B6	13000 ×11000	0.5	R,G,B,IR	Pleiades-1B	17°	train & val
B7	13000 ×11000	0.5	R,G,B,IR	Pleiades-1A	17°	test
B8	10000 ×9000	0.5	R,G,B,IR	WorldView-2	23°	test
B9	15000 ×10000	0.5	R,G,B,IR	Pleiades-1B	17.2°	test

Fig. 4.7 Six samples from building dataset.

That is, the network is deployed over cities that are different from those used for training. Training and validation samples are generated as described in Section 4.3, reserving 70% of areas B1, B2, B3, B4, B5 and B6 for training and the rest for validation. For each area, the ground truth includes the two classes of building and background. Thus, the problem of segmenting such dataset is a binary pixel classification problem.

Network Depth

In the first experiment, we evaluate the performance of our proposed architecture as a function of encoder depth. Table 4.4 (middle) shows that performance increases with the residual encoder depth. The network with the 34-layers encoder outperforms the 18-layers counterpart by 1.93% and 0.39% respectively. As the inner architecture of the residual units is identical (see Tab. 4.1), we attribute such gain to the 16 extra residual layers. The proposed network with 152-layers encoder outperforms the 18-layers counterpart by about 9% and 1% respectively. As the number of filters in each residual unit increases in blocks 2 to 4, we attribute such gain both to the four-fold increase in the number of filters per layer and to the 134 extra layers. This result is in line with those of [36], where a ResNet performance in an image classification task was found to increase with its depth.

Table 4.4 F1-score and accuracy over the buildings dataset test areas. Top: The proposed network and U-net with different encoder depths and Deeplab V3+ with two different backbone, Bottom: Adapted networks using BN statistics refinement (Norm), active learning over 10% (AL-10%) and 30% (AL-30%) of each test area.

Method	F1-score [%] B7	B8	B9	Average F1- score [%]	Oveall Acc. [%]
Prop-18	65.81	71.92	76.07	71.26	94.78
Prop-34	69.26	74.79	75.54	73.19	95.17
Prop-50	72.34	74.08	76.07	74.16	95.23
Prop-101	75.82	78.87	79.47	76.33	95.47
Prop-152	**79.27**	**79.16**	**83.51**	**80.65**	**95.76**
U-net-19 [82]	66.31	69.96	73.22	69.83	94.13
U-net-35 [82]	67.47	68.14	75.60	70.40	94.48
DeepLab V3+* [12]	71.88	75.02	78.98	75.29	95.14
DeepLab V3+** [12]	72.15	73.82	70.00	71.99	94.82
Prop-152 (Norm)	83.23	81.06	86.23	83.51	96.10
Prop-152 (AL-10%)	81.98	85.06	86.97	84.67	96.10
Prop-152 (AL-30%)	**86.87**	**85.99**	**87.80**	**86.88**	**96.71**
U-net-35 (Norm)	69.12	70.95	75.90	71.99	94.75
U-net-35 (AL-10%)	70.37	71.58	76.17	72.70	94.89
U-net-35 (AL-30%)	73.03	75.12	77.85	75.33	95.11
Deeplab V3+* (Norm)	74.15	79.99	79.11	77.75	95.32
Deeplab V3+* (AL-10%)	75.56	80.80	79.25	78.53	95.39
Deeplab V3+* (AL-30%)	81.57	81.36	83.10	82.01	95.82
Deng *et al.* [16]	52.33	58.38	58.90	56.53	92.85

* with ResNet101 backbone
** with Xception backbone

As a reference, we implemented an encoder according to the plain convolutional U-Net architecture [82] with depths of 19 and 35 layers (i.e., the decoder is untouched). The two resulting networks were trained from scratch according to the procedure described in [82]. Table 4.4 (top) shows that the 18 and 34 layers residual encoder outperforms the plain convolutional encoder for similar depths of 19 and 35 layers by 1.4 % and 2.8 % respectively in terms of average F1-score. Moreover, the 18 layers

residual encoder outperforms the 35 layers U-Net plain convolutional counterpart on the average. Similar result was found in [36], where an 18 layers residual network outperformed a plain 34 layers CNN in an image classification task. Such results suggest that a shallower residual encoder offers better generalization ability than a deeper plain convolutional architecture (Fig. 4.8). Our experience with deep residual networks also suggests they are easier to train than plain convolutional networks, so we hypothesize that careful fine-tuning of the U-Net optimization algorithms parameters may reduce at least in part such gap.

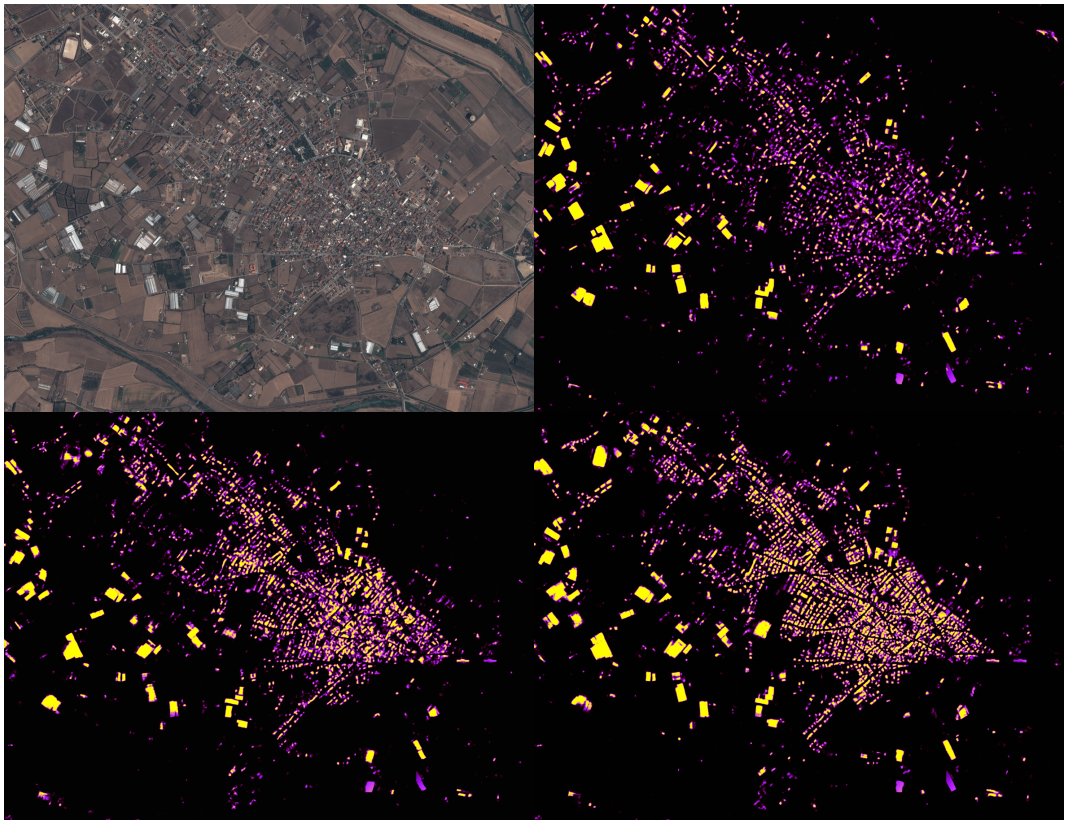


Fig. 4.8 Score maps over area B7 (top left) using proposed network with encoder depth of 18 (top right), 50 (bottom left), 152 (bottom).

We conclude that a deep residual encoder has better generalization ability than a shallower residual counterpart (and to some extent of a deeper plain convolutional encoder). We hypothesize that such advantage comes from the larger number of filters a deep encoder can learn at training time that allows it to learn visual representations of high semantic level, contributing to network performance over a wider range of satellite images.

In addition to U-Net, we compare with Deeplab V3+ [12], which achieved state-of-the-art performance over PASCAL VOC 2012 [19] and Cityscapes [14] datasets. Deeplab V3+ has an encoder-decoder architecture which makes use of a *backbone* network in the encoder for extracting feature maps. Deeplab V3+ with ResNet-101 and Xception backbones obtained the best performance in [18], therefore, we also train Deeplab V3+ once with ResNet-101 and another time with Xception backbone over the building dataset. As Table 4.4 (top) shows, Deeplab V3+ with ResNet-101 backbone achieves better performance compared with Xception backbone. These results are in line with our previous findings and suggest that ResNet has better generalization ability. Nevertheless, our proposed architecture with the same depth (Prop-101) outperforms Deeplab V3+ with ResNet-101 backbone. We conjecture such gain is first related to the larger number of skip connections employed in our architecture which results in finer segmentation maps, and secondly to the deconvolutional layers used in our architecture which can be optimized during training comparing with the bilinear upsampling layers used in Deeplab V3+ decoder which have fixed filters parameters.

Domain Adaptation

In the second experiment, we assess the domain adaptation techniques proposed in Sec. 4.5. As a baseline, we refer to the architecture with the 152-layers encoder that achieved top performance in the previous experiment.

Considering the scheme in Sec. 4.5.1, we update the batch normalization statistics over each test image independently. In detail, we applied the procedure provided in Algorithm 3 over the trained network of Prop-152 for 10 epochs ($n_{refine} = 10$) and momentum of 0.9 ($\alpha = 0.9$). Table 4.4 (bottom) shows that the proposed strategy improves the network performance by about 3% over the baseline in terms of average F1-score and without the need for human input. Considering the scheme in Sec. 4.5, we independently refine the trained network over 10% (AL-10%) or 30% (AL-30%) of each test image.

Moreover, we also apply our proposed adaptation methods with the same parameters to the 35-layers U-Net (U-Net-35) and Deeplab V3+ with ResNet-101 backbone. As seen in Table 4.4 (bottom), U-Net-35 average F1-score improves by 1.6%, 2.3% and 4.9%, whereas Deeplab V3 average F1-score increases by 2.4%, 3.2% and 6.7% using BN statistics refinement (Norm) and active learning (AL-10%, AL-

30%) respectively. These results imply that the proposed adaptation techniques are not specific to our architecture and other deep learning schemes can benefit from such adaptations. Nevertheless, since our proposed architecture exhibits better performance, it outperforms the other schemes.

Fig. 4.9 shows results with and without batch normalization update on area B8.

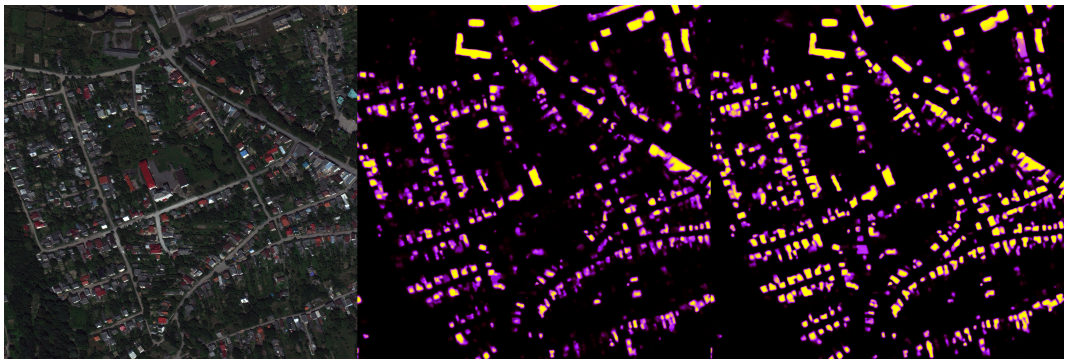


Fig. 4.9 Score maps over area B8 (left) using proposed network with encoder depth of 152 without domain adaptation (middle) and with domain adaptation using batch normalization update (Norm) on B8 area (right).

Additionally, we have implemented the active transfer learning network proposed in [16] for hyperspectral images, and applied it to our 4-bands building dataset. Authors in [16] addressed the related problem of domain adaptation over satellite images by proposing a spectral-spatial feature learning network. The network includes three sparse stacked autoencoders (SSAE): one operating on extended morphological attribute profiles (spatial SSAE), another one operating on the spectrum (spectral SSAE) and the last one is used to fuse the features learned using spatial and spectral SSAEs. SSAEs are trained first unsupervisedly over training samples, then based on a query criterion, a set of samples along with the labels are used to iteratively train the last softmax layer and also to fine-tune the SSAEs. As Table 4.4 shows the results of our implementation of [16] over building datasets, it can be seen that our approach outperforms it by a great margin. However it should be noted that the method in [16] is originally proposed for hypersepctral images which cover very small geographical areas; conversely, our datasets include vast geographical areas and contains only 4 spectral bands.

This experiment confirms that active learning can improve the performance of a previously trained network in a satellite image segmentation context. However, about

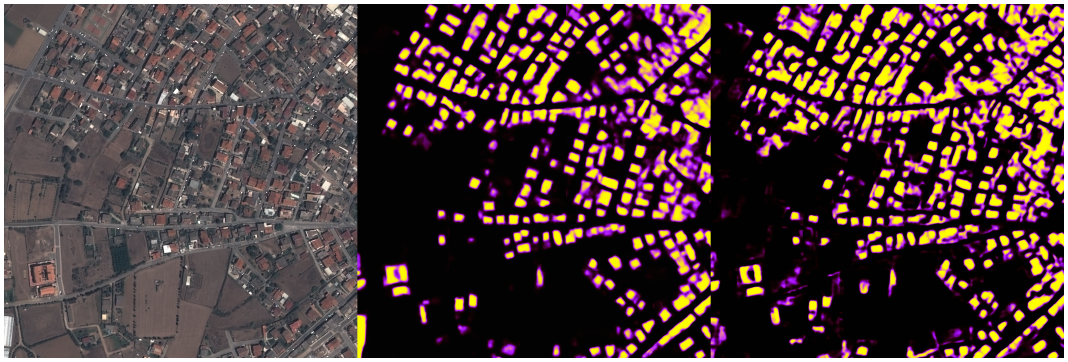


Fig. 4.10 Score maps over area B7 (left) using proposed network with encoder depth of 152 without domain adaptation (middle) and with domain adaptation using network refinement over 30% (FT-30) on annotated B7 area(right).

50% of such improvements are actually due to updating the batch normalization statistics alone, which has the advantage of requiring no additional annotations.

Finally, we compare the complexity of training the proposed network (152-layers encoder) from scratch with that of adapting a previously trained network. Training network from scratch for 300 epochs required 37 hours in our experimental setup. Adapting the trained network with the strategy in Sec. 4.5.2 required 5 minutes, without accounting for the time required to annotate the area of test image used for network refinement. Otherwise, updating the batch normalization statistics in Sec. 4.5.1 required about 200 seconds (no extra annotations required). Concluding, adapting a previously trained network is significantly less complex than retraining a network from scratch, offering a remarkable edge in time-critical applications such as emergency mapping.

4.6.3 INRIA Aerial Image Labeling Dataset

The second dataset we consider for our experiments is the INRIA Aerial Image Labeling Dataset [63]. Such dataset covers dissimilar urban settlements, ranging from urban areas (e.g., San Francisco's financial district) to alpine towns with a nominal resolution of 0.3 meters. As provided in Table. 4.5, the training set consists of 180 tiles of 5000 $\times$ 5000 pixels from the cities of Austin, Chicago, Kitsap County, Western Tyrol, and Vienna. The test set includes the same number of identically sized tiles covering the cities of Bellingham, Bloomington, Innsbruck, San Francisco, and Eastern Tyrol. Fig. 4.11 shows three tiles of Inria training images.

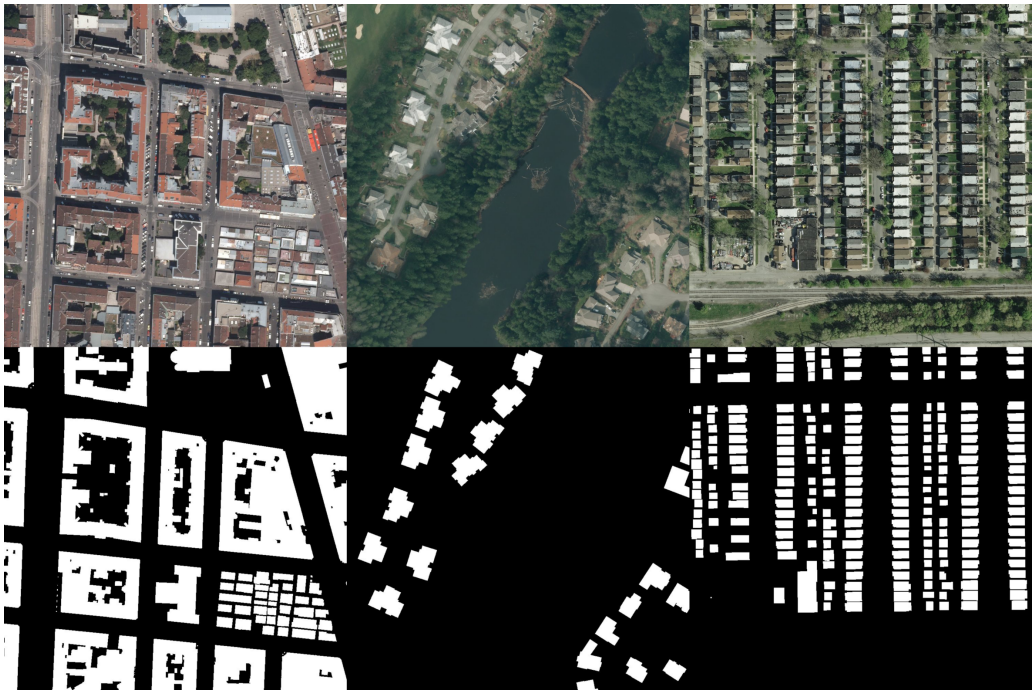


Fig. 4.11 Three tiles from Inria dataset (top) and their corresponding ground truth segmentation maps (bottom).

As in the previous experiment, the network is tested over cities different from those used for training indicating covariate shift in the dataset. The training set is annotated labeling each pixel as building or background; conversely, test images annotations are retained by the benchmark provider. We subdivide the annotated images into training and validation sets according to the benchmark organizer suggestions , i.e. for each city, the first five tiles are reserved for validation and the rest are used for training.

As a first experiment, we evaluate the performance of our proposed architecture as a function of the encoder depth. Since for this dataset, no ground truth is provided for the test set, the performance is first evaluated on the validation set using a larger set of 5 different encoder depths (18, 34, 50, 101, 152).

Table 4.6 shows that as the encoder depth increases, the segmentation quality improves. Such results are aligned with our previous findings in Table 4.4 with the buildings dataset. While validation and training sets are drawn from the same cities, we argue that the network shall be able to learn features relative to multiple cities, thus the network shall still be able to generalize across different areas of the same city.

Table 4.5 Image description on Inria dataset.

Image	Size (meter)	Resolution (meter)	Spectral bands	Set
Austin	1500 ×1500	0.3	3 bands	train & val
Chicago	1500 ×1500	0.3	3 bands	train & val
Kitsap County	1500 ×1500	0.3	3 bands	train & val
Western Tyrol	1500 ×1500	0.3	3 bands	train & val
Vienna	1500 ×1500	0.3	3 bands	train & val
Bellingham	1500 ×1500	0.3	3 bands	test
Bloomington	1500 ×1500	0.3	3 bands	test
Innsbruck	1500 ×1500	0.3	3 bands	test
San Francisco	1500 ×1500	0.3	3 bands	test
Eastern Tyrol	1500 ×1500	0.3	3 bands	test

Table 4.6 F1-Score and Accuracy of the proposed architecture over INRIA validation areas as a function of the encoder depth.

Encoder	F1-score [%]						Overall
Depth	Austin	Chicago	K. County	W. Tyrol	Vienna	Avg.	Acc. [%]
18	93.58	88.77	83.41	92.00	91.82	89.91	94.88
34	93.66	88.95	84.93	92.82	92.01	90.47	95.02
50	93.82	89.59	85.23	93.04	91.85	90.70	95.12
101	93.92	89.92	85.27	94.68	92.61	91.28	95.37
152	94.61	89.82	87.36	94.75	93.39	91.98	95.62

Hence, the results demonstrate that the encoder depth plays a key role in learning more robust visual representations with respect to covariate shift.

In the second experiment, we investigate how the encoder depth and the proposed batch normalization statistics refinement affect the network performance over test images. For this experiment, we used the previously trained networks, however, with a smaller set of encoder depths including 18, 50 and 152 layers encoders to segment the 5 test images. Then, only for the 152-layers network, we applied the adaptation strategy in Section 4.5 to segment the 5 test images (due to the lack of the annotations required for refinement, we could not evaluate the domain adaptation strategy in Section 4.5). The BN statistics refinement is carried out following the procedure detailed in Algorithm 3 and for 10 epochs ($n_{epochs} = 10$) over each image with momentum of 0.9 ($\alpha = 0.9$). Then, the resulting segmentation maps were provided to the benchmark organizer that computed and returned us the relative

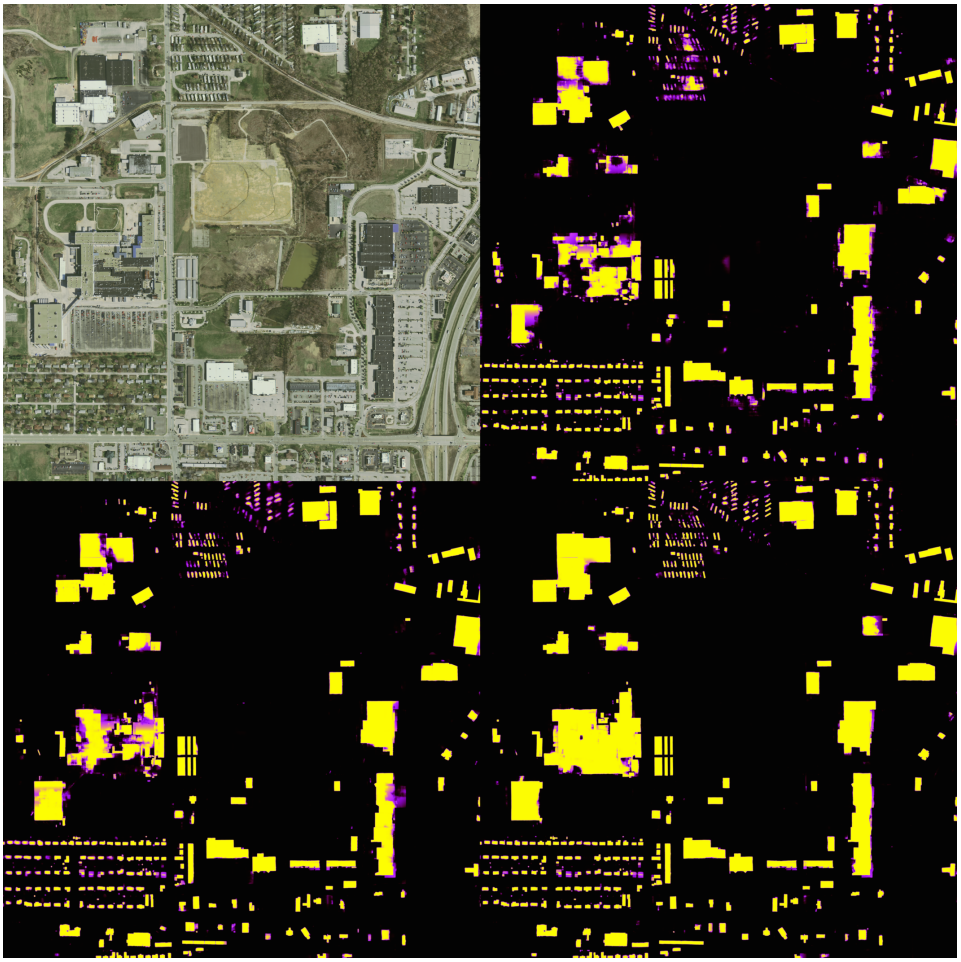


Fig. 4.12 Results over test area 6 in Bloomington city of INRIA dataset. The RGB input image is on the top left while score maps (Decoder SoftMax outputs) for the proposed network with 50, 101 and 152 layers provided in the top right, bottom left and bottom right respectively. As the encoder depth increases, the quality of the score maps improves.

segmentation accuracy in terms of Intersection over Union (IoU) as shown in Table 4.7 together with the top-5 performing references reported in [41].

Consistently with our previous experiments over the buildings dataset, the results show that a deeper encoder improves the network performance over most test images. Namely, the 152-layers encoder network achieves a 2.26 % gain over the 18-layers encoder network in terms of average IoU and a 0.14 % gain in overall accuracy. This gain supports our finding that a deep residual encoder is able to learn visual representations that are more robust to covariate shifts, results in better performance over unseen images. Fig. 4.12 shows how the output score map over an area of Bloomington city from test set improves as encoder depth increases.

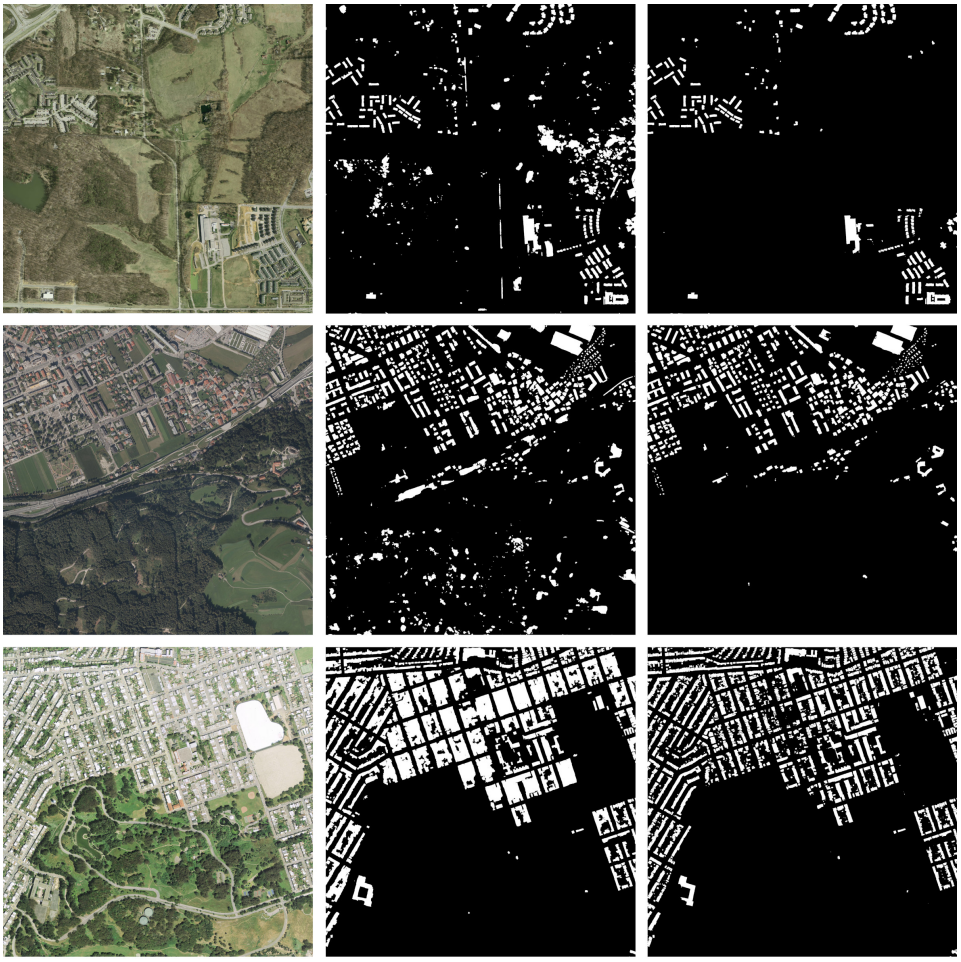


Fig. 4.13 In the left column, RGB images from INRIA test set are provided. Each of these images shows an area in the cities of Bloomington (top), Innsbruck (middle) and San Francisco (bottom). The central column shows the segmentation maps predicted by the proposed network (152 layers encoder). The right column shows the segmentation maps predicted by the adapted network using normalization statistics refinement over each test image.

Finally, BN statistics refinement considerably improves by 2.87% in terms of average IoU over our baseline, outperforming the other references in 4 out of 6 cities and all other references both in terms of mean IoU and overall accuracy. Fig. 4.13 illustrates the improvement due to batch normalization statistics refinement over three images of INRIA test set.

Table 4.7 Segmentation performance as IoU and Accuracy over INRIA test images (numbers provided by the benchmark organizer).

Method	Bellingham	Bloomington	Innsbruck	San Francisco	East Tyrol	Mean IoU	Overall Acc.
AMLL [41]	67.14 %	65.43 %	72.27 %	**75.72** %	74.67 %	72.55 %	95.91 %
NUS [41]	**70.74** %	66.06 %	73.17 %	73.57 %	76.06 %	72.45 %	95.90 %
ONERA [41]	68.92 %	68.12 %	71.87 %	71.17 %	74.75 %	71.02 %	95.63 %
Raisa [41]	68.73 %	60.83 %	70.07 %	70.64 %	74.76 %	69.57 %	95.30 %
INRIA [63]	56.11 %	50.40 %	61.03 %	61.38 %	62.51 %	59.31 %	93.93 %
Proposed-18	69.70 %	66.70 %	72.16 %	65.85 %	73.91 %	68.50 %	95.40 %
Proposed-50	68.17 %	67.97 %	73.07 %	66.78 %	75.42 %	69.20 %	95.52 %
Proposed-152	69.13 %	70.30 %	72.51 %	69.64 %	75.31 %	70.76 %	95.54 %
Prop-152 (Norm)	69.47 %	**75.17** %	**75.90** %	72.76 %	**76.89** %	**73.63** %	**96.10** %

4.6.4 Vaihingen ISPRS 2D Semantic Labeling Dataset

The third and last dataset we consider for our experiments is the ISPRS 2D Semantic Labeling Dataset [15], which includes 33 areas extracted from the city of Vaihingen, Germany as provided in Fig. 4.14. Each area consists of a true orthophoto (TOP) image (near-infrared, red and green bands) and relative Digital Surface Model (DSM); the ground sampling distance is 9 cm. A total of 16 areas out of 33 are meant for training and validation and are annotated with ground truth. The remaining 17 areas are meant for testing and so the related ground truth is not made available by the benchmark organizer.

While in the two previous datasets training and test images are captured from different satellites across multiple cities, with this dataset all images account for the same city as captured by the same satellite. Therefore, the covariate shift between training and test sets is small compared with the other datasets. However, while two previous datasets address a binary building-background segmentation problem, this dataset classifies each pixel into six classes: impervious surfaces, building, low vegetation, tree, car, and clutter (background). Thus, whereas this dataset is less suitable to stress a network robustness to covariate shift, the presence of similar classes such as low vegetation and trees and the difficulty in distinguishing small objects such as cars from background clutter makes it a challenging test for our architecture. Moreover, this dataset includes DSM which our network is not designed to handle.

The training and validation samples are generated subdividing the 16 annotated areas into validation and training subsets. Following the approach of [50] and [99], we

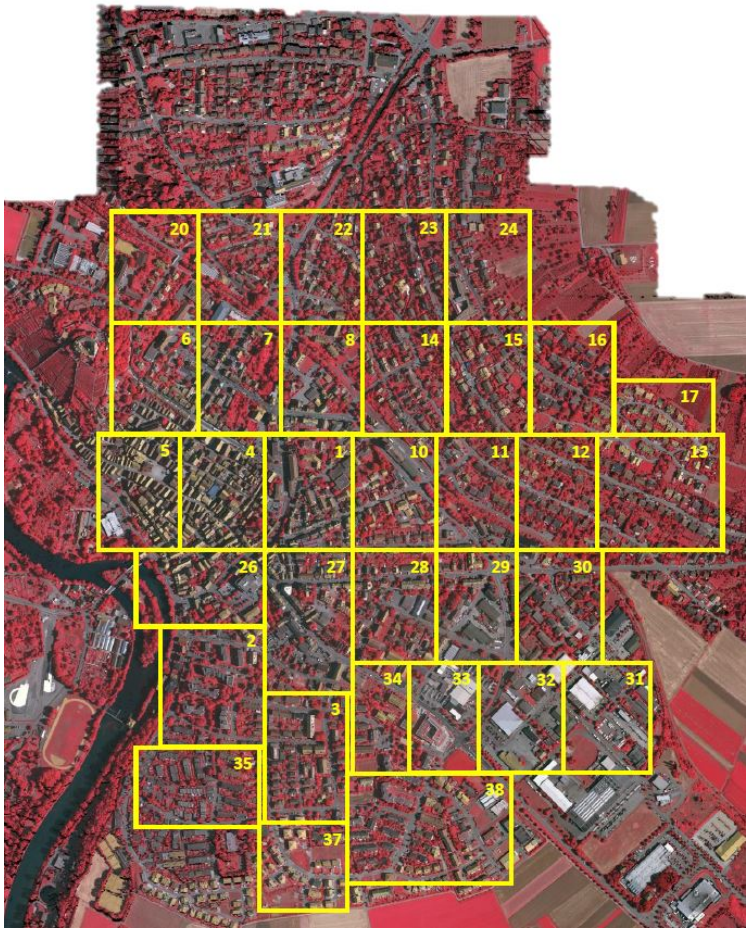


Fig. 4.14 Vaihingen city subdivided into 33 tiles.

reserve areas (11, 15, 28, 30, 34) for validation, whereas the rest is used for training. Most of the previous studies carried out on this dataset process the DSM separately and then the results are fused with those of TOP files in reason of the different nature of DSM data. However, since our goal is not to devise a scheme specialized for DSM images, we consider the DSM data as an additional color band for a total of four input bands. As for the other datasets, all networks are retrained from scratch following the same procedure.

As the first experiment, we study the effect of the encoder depth on the network performance over the validation areas as the ground truth of the test areas is not available to us. Table 4.8 shows that the segmentation quality improves with the encoder depth, coherently with our previous results (the clutter class was excluded from the table following the example of the dataset provider as it is of limited

Table 4.8 F1-Score and accuracy of the proposed architecture over Vaihingen validation images as a function of the encoder depth.

	F1-score [%]						Overall
Enc. depth	Imp. Sur.	Building	Low Veg.	Tree	Car	Avg.	Acc.[%]
18	85.96	91.15	72.63	83.99	68.01	80.34	83.83
34	86.01	91.80	73.27	84.30	67.98	80.67	84.15
50	86.70	92.30	74.76	84.86	71.36	81.99	84.99
101	87.24	93.65	74.31	84.71	82.82	84.57	85.56
152	89.17	93.78	77.08	85.54	83.84	85.88	86.77

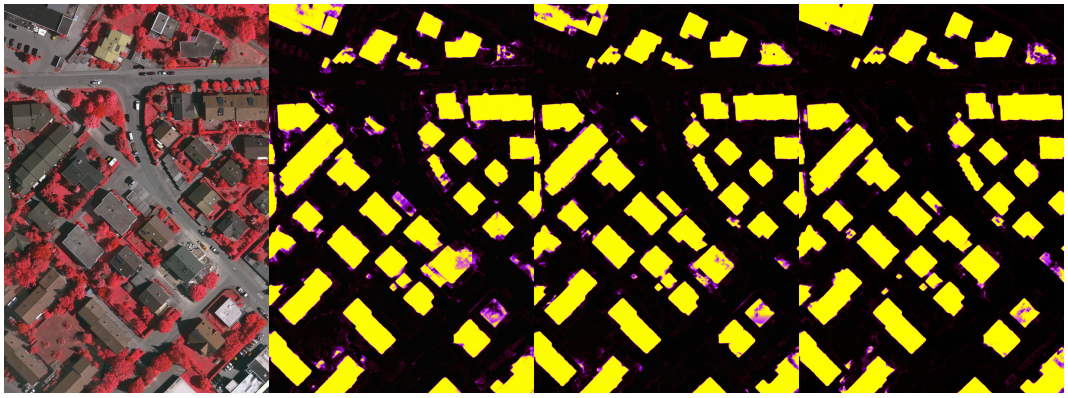


Fig. 4.15 Results over a validation area in Vaihingen city . The RGB input image is on the left while score maps (Decoder SoftMax outputs) for the proposed network with 50, 101 and 152 follows.

interest). We observe that besides the low vegetation that can be possibly mistaken with trees, cars represent the most difficult objects to recognize, we hypothesize due to the small scale of vehicles.

Fig. 4.15 and 4.16 shows an example from validation set for different encoder depth.

As a second experiment, we use the 152-layers encoder network to segment the 17 test areas. The segmentation maps were provided to the benchmark organizer who computed the performance against the retained ground truth. Table 4.9 contains the confusion matrix (top half) and per-class precision, recall and F1-Score averaged over the 11 test areas (bottom half). The matrix supports our hypothesis that low vegetation can be easily mistaken with trees and shows that cars are often mistaken by impervious surfaces, which are characterized by similar small scale. Yet, our

Table 4.9 Confusion matrix (top half) and segmentation performance (bottom half) for our proposed architecture with 152-layers encoder over the Vaihingen test images (numbers provided by the benchmark organizer).

	[%]					
	Imp. Sur.	Building	Low Veg.	Tree	Car	Clutter
Imp. Sur	90.6	3.6	4.6	0.8	0.3	0.1
Building	2.5	95.6	1.5	0.3	0	0.1
Low Veg.	5.7	1.7	81.2	11.3	0	0.1
Tree	1.1	0.3	9.4	89.2	0	0.0
Car	11.8	7.4	0.8	0.4	79.2	0.4
Clutter	24.7	30.0	4.6	4.0	0.8	36.0
Precision	91.1	93.6	81.8	88.1	87.2	77.2
Recall	90.6	95.6	81.2	89.2	79.2	36.0
F1-score	90.8	94.6	81.5	88.7	83.0	49.1

architecture is capable of correctly identifying buildings with an F1-score close to 95%.

Table 4.10 Segmentation accuracy over the 17 Vaihingen dataset test images (numbers provided by the benchmark organizer).

	F1-Score [%]					
	Imp. Sur.	Building	Low Veg.	Tree	Car	Overall Acc.[%]
Pa. *et al.* [74]	89.5	93.2	82.3	88.2	63.3	88.0
Ka. *et al.* [50]	92.1	95.3	83.9	91.0	83.6	89.2
Au. *et al.* [3]	91.0	94.5	84.4	89.9	77.8	89.8
GSN [101]	91.8	95.0	83.7	89.7	81.9	90.1
DLR-9 [64]	92.4	95.2	83.9	89.9	81.2	90.3
Proposed-152	90.8	94.6	81.5	88.7	83.0	89.0

Table 4.10 compares our proposed architecture with the top-5 best performing references made available on the benchmark organizer website. The DLR-9 [64] scheme achieves top performance via a network operating on three different scales. Moreover, two distinct networks are employed, one for detecting class boundaries and the other one to predict score maps. Then, the boundaries and segmentation results are fused to generate the final segment map. Audeber *et al.* [3] proposes a similar strategy, deploying a multi-scale and multi-modal architecture to address the pixel-

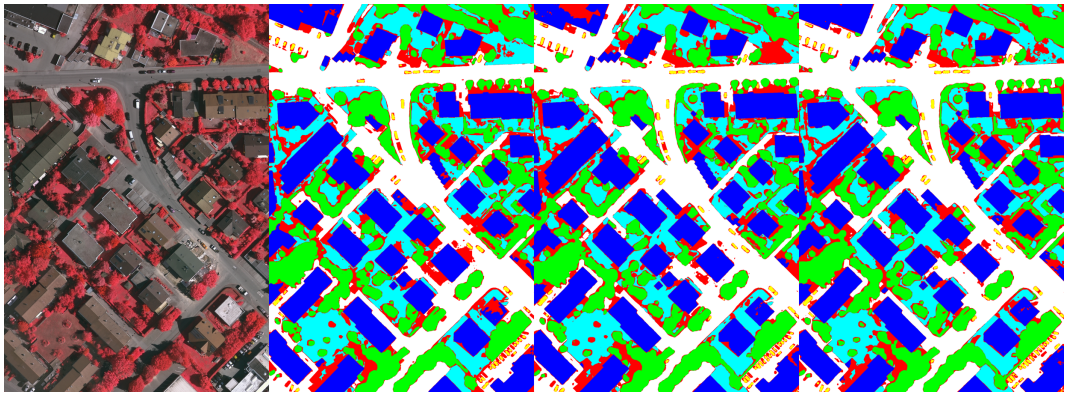


Fig. 4.16 Results over a validation area in Vaihingen city . The RGB input image is on the left while segmentation results for the proposed network with 50, 101 and 152 follows. Red colors indicates false predictions while the rest are true predictions.

based classification. Kampffmeyer *et al.* [50] also devised multi-modal strategy using patch-based and fully convolutional networks and also median frequency balancing is implemented on loss function to overcome the issue of unbalanced classes in the dataset. By comparison, our architecture is considerably less complex since it relies on a single scale network and is not designed to deal with DSM data. Despite our architecture being simpler and meant to improve performance against covariate shift, still, it performs almost as well as specialized and more complex ones. For this dataset, we do not report any result concerning the adaptation strategies in Sec. 4.5 since train and test samples are captured from the same city by the same satellite, thus covariate shift is minimal.

Chapter 5

Onboard Cloud Screening for Satellite Images

A cloud screening unit on a satellite platform for Earth observation can play an important role in optimizing communication resources by selecting images with interesting content while skipping those that are highly contaminated by clouds. In this chapter, we address the cloud screening problem by investigating an encoder-decoder CNN similar to architecture proposed in the previous chapter in Sec. 4.2. Nevertheless, as we have seen, CNNs usually employ millions of parameters to provide high accuracy; on the other hand, the satellite platform imposes hardware constraints on the processing unit. Hence, to allow an onboard implementation, in this chapter we investigate experimentally several solutions to reduce the resource consumption by the CNN while preserving its classification accuracy. We experimentally explore approaches such as halving the computation precision, using fewer spectral bands, reducing the input size, decreasing the number of network filters and also making use of shallower networks, with the constraint that the resulting CNN must have sufficiently small memory footprint to fit the memory of a low-power accelerator for embedded systems. The trade-off between the network performance and resource consumption has been studied over the publicly available SPARCS dataset [44], [98]. Finally, we show that the proposed network can be implemented on the satellite board while performing with reasonably high accuracy compared with the state-of-the-art.

5.1 Related Work

A very well-known cloud detection algorithm is F1-mask [116] which is employed on Landsat imagery and employs top-of-atmosphere reflectance and brightness temperature for all Landsat bands, detecting cloudy pixels through a series of spectral tests. Similarly, authors in [34] report that the EO-1 spacecraft employs calculation of top-of-atmosphere reflectance followed by a few threshold tests in order to perform onboard cloud screening. While this is the first demonstration of cloud screening, screening a 1024x256 image requires about 30 minutes, which is far too much for real-time processing. In addition, as many traditional remote sensing approaches make use of handcrafted features such as pixel shape index [111] or morphological functions [5, 4, 42, 43], such features are also found to be an effective tool to detect clouds as described by authors in [21] over images acquired by SPOT Earth observation satellites. More recently, cloud detection has been addressed by employing machine learning techniques including Bayesian statistical techniques [68, 96], decision-tree classifier [88], support vector machine [83] and also random forest [100]. Finally, authors in [71] study a combination of machine learning approaches such as linear, quadratic and nonparametric discriminant analysis, principal component and independent component discriminant analysis to handle the cloud detection over MODIS images.

The recent decade has witnessed rapid development in the area of deep learning leading to an outstanding performance in many fields including computer vision. Well-known convolutional neural networks (CNNs) have advanced the state-of-the-art in many computer vision tasks such as image classification and semantic segmentation [54, 95, 36, 12, 37, 27]. Not an exception to such trend, remote sensing has been also enjoying the benefits of deep learning algorithms in achieving state-of-the-art performance in many tasks such as land-use classification and segmentation [103, 2, 76, 114, 26, 25]. CNNs usually consist of thousands of filters with millions of learnable parameters which are trained to detect semantic representations useful for a particular task and over a large amount of annotated images. Regarding cloud detection, recent studies have been carried out to make use of CNNs also in performing such detection task [87, 56, 102, 113]. For instance, in [87], to perform cloud detection, satellite images first undergo a simple linear iterative clustering process in which homogeneous pixels are clustered into superpixels, then a four layers CNN employed to extract features and finally, two fully connected

layers predict the superpixels class. In [56], authors show that a similar CNN architecture with 6 convolutional layers operating on 32×32 patches when combined with superpixel clustering can perform cloud detection with reasonably high accuracy on SPOT 6 images. A very well-known CNN architecture which is widely used in many segmentation tasks is U-net [82] which is originally proposed to address biomedical image segmentation. U-net has a simple yet effective architecture which consists of an encoder and a decoder network which are connected together with skip connections. The use of skip connections contributes to finer segmentation results without the need of post-processing step as the spatial information of early layers in the encoder is used in the decoder. Authors in [69] address the cloud detection over Landsat 8 images proposing an architecture similar to U-net. To train such network, first a snow/ice removal framework which is based on gradient-based identification is applied over Quality Assessment (QA) band of Landsat images, then this layer is used as a groundtruth in the course of training which results in high accuracy cloud detection.

A lightweight convolutional encoder-decoder network is proposed in [113] to address cloud screening, whose input features are not the pixel values, but their wavelet coefficients. Authors construct experiments over SPARCS dataset using 4 out of 11 bands and the proposed architecture outperforms common machine learning techniques such as Adaboost, random forest and SVM by a considerable margin. The same architecture is used in [22] for high-resolution videos.

Cloud screening differs from cloud detection in that the outcome of the detection process is not used for scientific applications, but just to decide whether an image or part thereof has to be discarded. However, satellite board imposes hardware constraints on the cloud screening unit in terms of memory and power consumption. Additionally, in order to be useful, a cloud screening algorithm should be able to process an image in a near real-time manner with high precision.

In this study, we consider the cloud screening problem as a binary pixel-based segmentation problem where the image pixels are divided into cloud and non-cloud classes. We focus on multispectral images as such images do not contain finely-grained spectral information about the wavelengths at which clouds can be detected. Conversely, in hyperspectral images, one can pick a few wavelengths and efficiently detect the presence of clouds using very simple methods, see e.g. [96].

To tackle such problem, we also employ an encoder-decoder CNN inspired by U-net in which the encoder extracts visual representations (i.e. feature maps) over the input image, then the decoder takes as input such representations and generates segmentation maps. However, unlike previous work and original U-net architecture, we take into account the hardware limitations imposed by satellite platform, by designing and testing several variants of this encoder-decoder network having different representation power, classification accuracy, memory footprint, and complexity. Since most state-of-the-art CNNs include millions of parameters, such limitations introduce unique challenges which require to be addressed carefully. In particular, we study the trade-off between the resource consumption and the network performance in terms of classification accuracy by investigating several approaches such as limiting the number of network filters, decreasing the network depth, reducing the input size both in spatial and spectral domains and also operating on half-precision floating points. We provide our experimental results over the SPARCS (Spatial Procedures for Automated Removal of Cloud and Shadow) publicly available dataset [44], [98], and we show that the proposed network can perform close to the state-of-the-art CNNs, while consuming fewer resources. In terms of resources, we consider that a low-power accelerator for embedded systems such as the Intel Myriad family typically provides 500 MB of memory to accommodate the neural network, temporary data, and input data; so we target the design of neural networks whose memory footprint is around this value or lower.

In the following, first, we detail the architecture of the proposed network which we use as a baseline in our experiments. Then, the procedure used to generate training samples is described in detail. Next, the cost function used to optimize the network parameters and the related training process are defined. then the experimental results over SPARCS dataset are provided.

5.1.1 Network Architecture

The proposed network as depicted in Fig. 5.1 includes encoder and decoder networks. The encoder utilizes a sequence of convolutional layers to extract feature maps which can be seen as semantic representations of the input image. Then, the decoder takes as input such feature maps and through deconvolutional layers, it generates the segmentation map which labels the pixels into cloud and non-cloud classes. In the following, we describe separately the architecture of encoder and decoder networks in detail.

5.1.2 Encoder

The encoder network includes five convolutional layers illustrated as gray blocks in Fig. 5.1. Each convolutional layer is followed by a batch normalization layer and employs rectified linear unit (ReLU) as activation function, except for the last convolutional layer before the output which employs sigmoid activations and performs the final classification as explained in the following section. Note that in Fig. 5.1, for simplicity, batch normalization layers and activation functions are omitted.

Each convolutional layer in the encoder extracts feature maps from the input image using a number of filters. In Fig. 5.1, the number of input and output feature maps (i.e. channels), the filter size, stride and padding size are provided for each layer accordingly. An important aspect of CNNs that should be addressed in a segmentation task is the field of view of the network. Each generated feature map in the network has a specific field of view which is defined as the number of input pixels which are used to compute a pixel in that feature map. Therefore, the field of view of a specific feature map defines the window size on the input image over which each feature is computed. The field of view in the networks can expand by proceeding to the deeper layers or using large filter size or even using larger stride size. In our proposed network, all encoder layers, except the first layer, have 3×3 convolutional filters. Nevertheless, in the first encoder layer, the filter size is chosen to be 7×7 to increase the field of view in the first layer. Moreover, to better handle the memory consumption and also in order to further expand the field of view of encoder layers, all convolutions have a stride of two. Therefore, each layer outputs feature maps whose resolution is halved with respect to the feature maps taken as

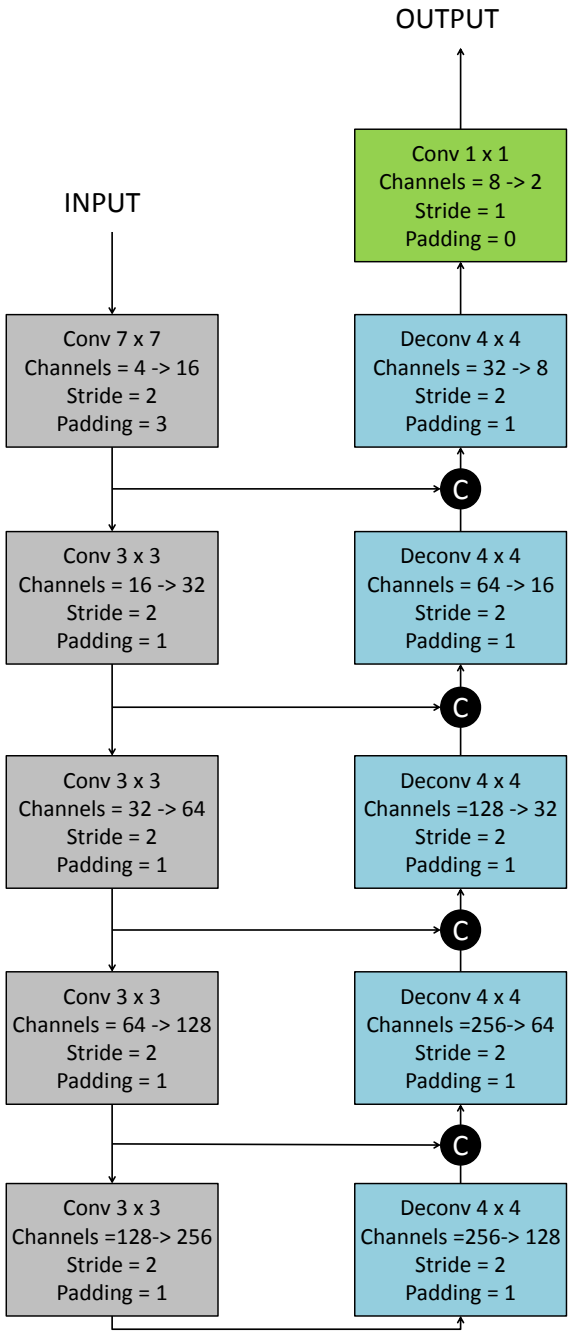


Fig. 5.1 Network architecture: the encoder (bottom) and the decoder (top) are illustrated in dashed boxes. Number of input and output channels (i.e. feature maps) as well as the size of filter, stride and padding are provided for each layer.

input. On the other hand, as we proceed to the deeper layers in the encoder, the number of output feature maps is increased by a factor of two starting with 16 feature maps in the first layer and ending with 256 feature maps at the last encoder layer.

To exemplify, let us consider the input image with the size of 256×256 with 4 spectral channels, the first encoder layer takes as input such 256 x 256 image and outputs 16 feature maps with the size of 128×128. Then, the second layer takes as input 16 feature maps with the size of 128×128 and outputs 32 feature maps with the size of 64×64. Therefore, by proceeding into deeper layers in the encoder, the generated feature maps resolution decreases as their number increases. At the end, the last encoder layer output 256 feature maps with the resolution of 8×8 pixels.

5.1.3 Decoder

The decoder network includes five layers paired to the five encoder layers as shown in Figure 5.1 by blue blocks. Each decoder layer consists of one deconvolutional layer followed by batch normalization layer and ReLU activation function. Deconvolutional layer (backward convolution) was originally proposed to address the loss of mid-level cues caused by pooling operators used in convolutional networks [62]. In our proposed decoder, we also make use of deconvolutional layers to upsample the feature maps generated by the encoder, in order to be able to recover the spatial resolution of the input image. A deconvolutional layer operates in two stages: first, the pixels over the input image (or input feature map) are interleaved with zeros, thus the input is upsampled and a sparse output is generated. Then, by applying a convolution filter to such sparse image, a dense output is finally produced. As a result, a deconvolutional layer can be seen as an upsampling layer which consists of learnable filters that attempt to reverse the sub-sampling operation performed by convolutional layers in the encoder.

Skip connections are also employed between the encoder and decoder layers to generate more precise and finer predictions as the spatial information of early layers in the encoder is used also in the decoder. Therefore, the output feature maps by each encoder layer are concatenated with the feature maps which are output by the corresponding layer in the decoder. In addition, in our design, we chose the number of filters in each decoder layer so that the number of feature maps coming from skip connections matches the number of feature maps generated by previous decoder

layer. We experimentally verified that such condition is necessary to prevent one group of feature maps from dominating the other when they are concatenated and forwarded to the next decoder layer.

For the sake of clarity, we exemplify the operations of the decoder network using the same example as provided in the previous section for the encoder. Thus, let us consider the input image with the size of 256×256, the 1-st decoder layer takes as input the 256 feature maps sized 8×8 generated by the 5-th encoder layer. The feature maps are then scaled up by a factor of two by the 128 deconvolutional filters, reaching a 16×16 resolution. Such 128 feature maps are then concatenated with the identically sized 128 feature maps generated by the 4-th encoder layer. The resulting 256 concatenated 16×16 feature maps are provided as input to the 2-nd decoder layer, and so forth. The 5-th decoder layer finally outputs 8 feature maps with the size of 256×256 matching the input size. Next, the decoder output is processed by a convolutional layer with 1×1 filters generating 2 feature maps with size 256×256 pixels: the i-th pixel in the k-th feature map $o^{i,k}$ represents the relative confidence that such pixel in the input image belongs to the k-th class, where in our case of binary segmentation $k = 2$. At the end, a sigmoid activation is applied over resulting feature maps squeezing the numbers in the interval (0,1): $y_{i,k} = 1/(1 + e^{o_{i,j}})$.

In the end, to compare our proposed network with the original U-net, we would like to highlight some of the differences which result in more efficient implementation. In our architecture, we don't utilize the pair of stride one convolutional layers in each encoder and decoder layer as they are used in the original U-net architecture which significantly reduces the number of network parameters (see Table 5.1). Moreover, since in our design, each encoder layer includes a convolutional layer with the stride of two, the max pooling layers which are used in the original U-net architecture has been omitted. Another difference is the use of larger convolutional filters in the first encoder layer with the size of 7 x 7 instead of 3 x 3 in original U-net. Such design enlarges the field of view of the network at this layer as well as proceeding layers. Additionally, unlike the U-net architecture, the output feature maps are not cropped in our architecture, since the spatial domain of concatenated feature maps is identical. Finally, the number of convolutional filters and hence the output feature maps with respect to original U-net has been decreased by a factor of four which considerably reduces the memory usage.

5.2 Generating Training and Test Samples

Similar to what we have seen in Sec. 4.3, given a dataset of annotated satellite images, the dataset is first subdivided into training and test sets as follows. To recall, the training set refers to images used for optimizing the network parameters. The test set refers to images used to validate the training procedure by measuring the trained network performance over such images.

The remote sensing datasets are usually provided in the form of very large images where each image side contains thousands of pixels. However, due to memory constraint, to be able to train the network, we first subdivide the image into smaller tiles. Therefore, as shown in Fig. 2, first, each image in the training set is subdivided into tiles of size 364×364. Notice that we consider the network input size to be 256×256, however, we extract larger tiles to be able to apply a set of augmentation transformations in the course of training as follows. From each tile, with 50% probability, a 256×256 patch is cropped at a random position. Otherwise, a 256×256 patch is cropped from the center of the tile which has been rotated using a bilinear transformation with a random angle drawn from a uniform distribution in the interval $[0, 2\pi]$. Next, horizontal and vertical flips each with the probability of 50% are applied independently over the extracted patch. Such augmentation techniques are necessary to prevent the network from being overfitted on the training set.

Concerning test images, since no augmentation is planned during the evaluation, we extract tiles of size 256×256 with partially overlapping samples from each test image. Averaging the network outputs over overlapped areas and on the neighboring patches helps to avoid artifacts.

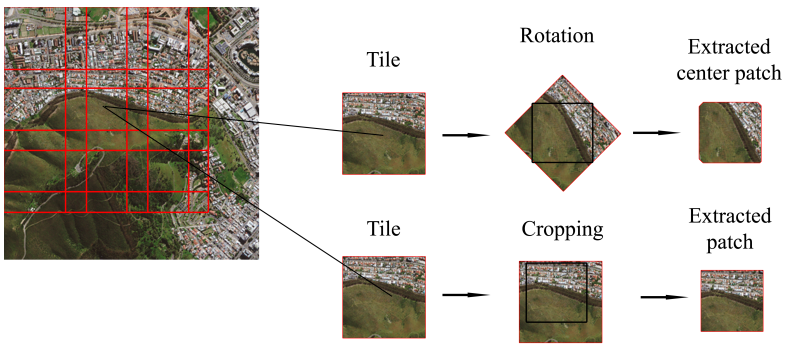


Fig. 5.2 Patch extraction and data augmentation during training similar to procedure detailed in Sec. 4.3.

5.3 Cost Function and Optimization

After the training and test samples are generated, the network is trained end-to-end in a fully supervised manner minimizing the binary cross-entropy loss function. To be precise, letting t_i, k be the one-hot target vector corresponding to class k, i.e. only the element corresponding to the correct class is equal to one, whereas all the other elements are equal to zero, then the binary cross-entropy loss function is computed as follows:

$$L(\theta, y, t) = -\sum_{i=1}^{H \times W} \sum_{k=1}^{2} (t_{i,k} \log (y_{i,k})) - ((1 - t_{i,k}) \log (1 - y_{i,k})). \tag{5.1}$$

where H and W are the input image width and height respectively and θ represents the network parameters. In addition, in order to prevent the network from overfitting on the training samples the final *cost* function we actually optimize at training time is:

$$J(\theta, y, t) = \eta L(\theta, y, t) + \lambda R(\theta), \tag{5.2}$$

where $R(\theta)$ is a regularization term defined as the squared L2 norm of all the weights in the network, and η and λ are the learning rate and regularization factor.

We train the network via stochastic gradient descent with a momentum of 0.9 and with the mini-batch size of 8. Concerning the learning rate adaptation strategy, we chose a base learning rate of $\eta = 0.05$ that is divided by a factor of 10 every 100 epochs and we trained the network for a total of 300 epochs. Moreover, L2 regularization with $\lambda = 0.001$ is applied during training.

5.4 Results

In this section, first we describe the dataset used in this study, then we define the evaluation metrics used to measure the performance; finally, the experiential results are provided.

Dataset

The dataset used in this study has originally been created by Hughes *et al.* [44] and is obtained manually from Landsat 8 Operational Land Imager (OLI) scenes. Its purpose was to validate cloud and cloud shadow masking derived from the Spatial Procedures for Automated Removal of Cloud and Shadow (SPARCS) algorithm. The dataset includes 10 spectral bands, however in most of our experiments we use only 4 bands corresponding to red, green, blue and infrared. The annotations are given for seven classes including shadow, shadow over water, water, snow, land, cloud, and flooded areas. In our experiments, we convert such annotations to the binary case of clouds and non-clouds pixels. 80 images with the size of 1000 by 1000 pixels are provided which we use 80% for training and the remaining 20% for testing. The training and test sets are chosen so that they cover different scenes over different geographical locations. Training set includes 64 images from which 1024 patches (16 patches over each image) of size 364×364 have been extracted, and test set includes 16 images from which 256 patches (16 patches over each image) of size 256×256 have been extracted.

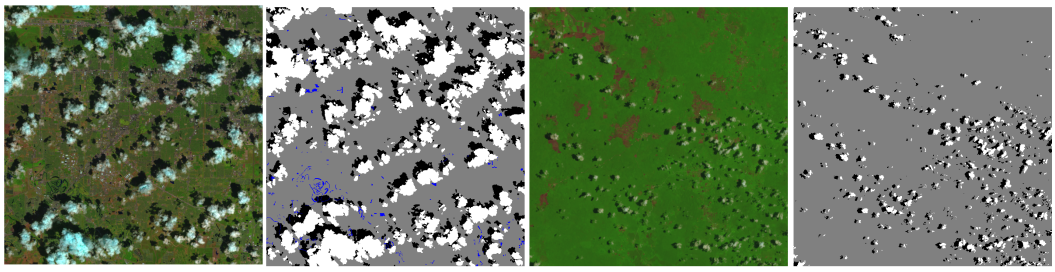


Fig. 5.3 Two images from SPARCS dataset with the corresponding masks.

5.4.1 Evaluation Metrics

In our experiments, we measure the network performance by measuring the following metrics.

- **F1-score** is defined as the harmonic mean of precision and recall: $F1 = 2 \cdot (precision \cdot recall)/(precision + recall)$, where *Precision* is defined as the ratio of correctly predicted pixels to all predicted pixels regarding a segmentation class: $Precision = tp/(tp + fp)$, and *Recall* is defined as the ratio of correctly predicted pixels to all pixels that belongs to a segmentation class: $Recall = tp/(tp + fn)$.Moreover, tp , fn and fp are true positive, false negative and false positive pixels respectively.

- **Overall accuracy** is the fraction of correctly labeled pixels for all classes, $Acc = \sum_{i=1}^{n_c} tp_i/n_p$ where n_c and n_p are the number of classes and the number of pixels respectively and tp_i denotes true positives for class i.

- **mIOU** is the ratio of correctly predicted area to the union of predicted pixels and the ground truth $IOU = \frac{tp}{tp+fp+fn}$ which is averaged over all classes.

- **Inference memory** is the amount of GPU memory which is occupied by the network during evaluation. The memory consumption is measured based on the maximum allocated GPU memory during inference using a Pytorch implementation of the proposed methodology.

- **Computation time** considers data loading time, the time interval in which the network processes the extracted patches over a 1000×1000 test image, the time needed to stitch patches to form segmentation maps and also the time required to compute the evaluation metrics.

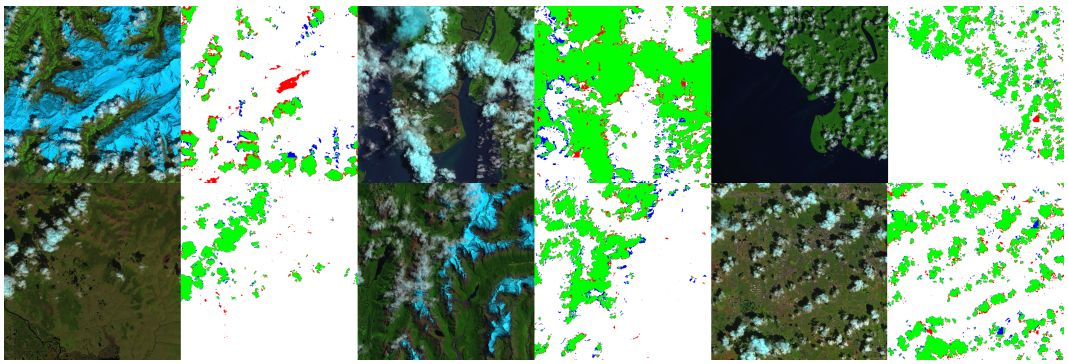


Fig. 5.4 The results of the proposed network (1-st row in Table 5.1) over 6 test images of SPARCS dataset. Green and white pixels represent true positive and true negative while blue and red pixels represent false negative and false positive outputs respectively.

5.4.2 Experiments

In this section, we provide the experimental results obtained by the network architecture defined in Sec. 5.1.1. In order to be able to measure the trade-off between the network resource consumption and its performance on the cloud screening task, we set up several experiments. In these experiments, we use different settings by considering a number of factors which can affect memory consumption as well as the network performance. The factors which are considered include the computation precision, the network encoder depth, the number of convolutional filters, the number of input spectral bands and also the input size. In all experiments, the network is trained from the samples generated over SPARCS dataset based on the procedure defined in Sec. 5.2 and employing the training methodology detailed in Sec. 5.3.

The experimental setup used for all the experiments below consists of running the methodology implemented in Pytorch framework [77] over NVIDIA GeForce GTX 1080 GPU with Pascal architecture and 8 GB memory.

Input bands: In the first experiment, we investigate how the number of spectral bands which are input to the network can affect both the network performance and its memory usage. Therefore, the proposed network defined in Sec. 5.1.1 is trained once over 4 bands of red (R), blue (B), green (G) and infrared (IR) bands, and another time over each one of these bands separately and also once over all 10 bands provided by SPARCS dataset. As is reported in Table 5.1, the best result in terms of F1-score, mIOU and accuracy is obtained when the network is trained over 4 bands of red, blue, green and infrared. Moreover, it can be seen that when the network trained over

Table 5.1 The proposed network performance is provided (top) with different encoder networks, encoder depths, computation precision, input spatial and spectral sizes. Moreover, the performance of state-of-the-art CNN, namely DeepLab V3+, is provided as well (bottom) with two different encoder.

Encoder type	**depth**	**Precision**	**Input bands**	**Input size** [pixels]	**Number of parameters**	**Inference memory** [MB]	**Overall accuracy** [%]	**F1-score** [%]	**mIOU** [%]
Plain	5	full	R,B,G,IR	256×256	1,269,018	15.52	95.24	90.36	83.53
Plain	5	full	R	256×256	1,266,666	14.72	94.51	87.99	80.37
Plain	5	full	B	256×256	1,266,666	14.72	94.36	88.27	80.55
Plain	5	full	G	256×256	1,266,666	14.72	94.06	87.25	79.30
Plain	5	full	IR	256×256	1,266,666	14.72	92.38	85.15	76.08
Plain	5	full	All	256×256	1,273,722	17.11	95.15	90.22	83.49
Plain	5	full	R,B,G,IR	128×128	1,269,018	10.20	95.46	90.06	83.55
Plain	5	full	R,B,G,IR	64×64	1,269,018	9.20	95.27	89.16	82.39
Plain	5	half	R,B,G,IR	256×256	1,269,018	8.05	85.06	75.45	55.49
Plain*	5	full	R,B,G,IR	256×256	318,478	7.03	95.28	90.08	83.09
Plain+	5	full	R,B,G,IR	256×256	80,232	3.87	94.79	88.90	81.37
ResNet	18	full	R,B,G,IR	256×256	16,550,722	132.56	96.24	**92.59**	**86.85**
ResNet	34	full	R,B,G,IR	256×256	26,658,882	267.23	**96.42**	92.39	86.45
ResNet	50	full	R,B,G,IR	256×256	103,629,954	889.29	96.23	91.77	85.62
U-net [82]	9	full	R,B,G,IR	256×256	39,402,946	315.36	96.08	91.01	84.89
FMask [115]	-	full	All	1000×1000	-	-	86.81	70.11	62.01
Deeplab V3+ [12]									
ResNet	101	full	R,B,G,IR	256×256	59,342,562	503.7	94.87	89.47	82.07
Xception	-	full	R,B,G,IR	256×256	54,700,722	481.69	89.85	83.58	73.51

* number of filters divided by two
+ number of filters divided by four

one spectral band, the more informative band is red in terms of overall accuracy and blue in terms of F1-score. We should note that the difference between F1-score and overall accuracy is that the overall accuracy takes into accounts also the true negative pixels, however, F1-score does not consider true negatives. Thus, this explains the variations between overall accuracy and F1-score also in the remaining experiments, where one network may outperform another network only in one metric.

Considering the inference, the network with one input band is faster by a small margin of 1 ms in the processing input image (inference time is shown in Table 5.2) and consume 2.4 MB less than the network with 10 input bands. The results imply that the number of input spectral bands has a small effect on network resource consumption since only the number of input channels of the first convolutional layer in the encoder (see Figure 5.1) changes and the rest of the network remains the same. This also shows that using more than 4 bands does not bring any improvement to the problem of cloud screening.

Table 5.2 The time interval which is required to: a) load the extracted patches (over 1000×1000 pixels test image) from hard drive into memory b) compute the network outputs over the input patches (i.e. patches extracted from a test image) c) stitch the network outputs (to produce 1000×1000 segmentation maps) d) compute the evaluation metrics, are provided.

Encoder type	**depth**	**Precision**	**Input bands**	**Input size** [pixels]	**Data loading** [ms]	**Inference** [ms]	**Stitch** [ms]	**Eval metrics** [ms]	**Total** [ms]
Plain	5	full	4	256×256	37	168	110	415	730
Plain	5	full	1	256×256	8	167	110	415	700
Plain	5	full	10	256×256	7240	170	110	415	7935
Plain	5	full	4	128×128	45	146	110	415	716
Plain	5	full	4	64×64	80	1136	110	415	1741
Plain	5	half	4	256×256	18	158	110	415	701
Plain*	5	full	4	256×256	37	153	110	415	715
Plain+	5	full	4	256×256	37	100	110	415	662
ResNet	18	full	4	256×256	37	407	110	415	969
ResNet	34	full	4	256×256	37	536	110	415	1098
ResNet	50	full	4	256×256	37	733	110	415	1295
U-net [82]	9	full	4	256×256	37	720	110	415	1282
FMask [115]	-	full	10	1000×1000	37	1470	-	415	1922
Deeplab V3+									
ResNet	101	full	4	256×256	37	422	110	415	984
Xception	-	full	4	256×256	37	441	110	415	1003

* number of filters divided by two
+ number of filters divided by four

Input size: In this experiment, we study the connection between network performance and the input size. Accordingly, we train the proposed network in Sec. 5.1.1 over samples which are generated based on the procedure described in Sec. 5.2, using different sizes for extracting patches i.e. 256×256, 128×128 and 64×64. Since the proposed network is fully convolutional, it can take as input an image with arbitrary size as long as its size is a multiple of 32. In all experiments in this part, the four spectral bands corresponding to red, green, blue and infrared are used. The 1-st, 7-th and 8-th rows in Table 5.1 show the results for input size of 256×256, 128×128 and 64×64 respectively; it can be seen that in terms of cloud screening performance, the network with input size of 128×128 has the best overall accuracy and mIOU while the network with input size of 256×256 performs with the best F1-score. In terms of memory consumption, decreasing the input size from 256×256 to 128×128 results in releasing about 5.3 MB of the memory, and in terms of inference time (Table 5.2) it processes the input data 22 ms faster. However, we note that further decreasing the input size contributes only marginally to memory consumption, although it can worsen the network performance and it leads to slower inference time due to the

larger number of patches that should be extracted to cover the test image. Such performance decline for smaller input size can be related to shrinking the network field of view. As is mentioned in Sec. 5.1.1, the larger the network field of view is, the more neighboring pixels are considered in predicting the score maps. To conclude, smaller input size can decrease the network memory consumption, but this comes at the expense of shrinking the network field of the view which may worsen the overall performance.

Precision: In this experiment, we investigate how the network performance can be influenced by the floating point precision employed for the representation of network parameters and the corresponding computations, hence we experiment with half precision computations. Pytorch framework by default uses full precision floating-point which occupies 32 bits, thus by halving the precision of the floating points, 16 bits will be occupied. As Table 5.1 shows, the network with half precision (9-th row) occupies 8.05 MB of memory. Comparing to the network with full precision (1-st row) which consumes 15.52 MB, half precision computations leads to releasing about 50% of the memory during inference. Moreover, the network with half precision process the input 10 ms faster. However, in terms of segmentation performance, halving the precision results in a considerable drop of 14% in terms of F1-score, 33% in mIOU and 10% in overall accuracy. To conclude, as it will be explored in the next sections, comparing with other approaches, halving the precision can be regarded as an expensive approach of managing memory in terms of network segmentation performance.

Convolutional filters: In this part, we consider the number of convolutional filters in the network which can have a significant impact on the network performance. We measure the cloud screening performance and the memory consumption for the network with the same architecture defined in Sec. 5.1.1 but with a number of convolutional filters which is divided first by a factor of two (Plain*) and then by a factor of four (Plain$^+$). As reported in Table 5.1, dividing the number of network filters by a factor of two (10-th row) and by a factor of four (11-th row) results in 75% and 90% reduction in the number of network parameters respectively. As a result, the first (Plain*) and second (Plain$^+$) networks consume about 8.5 MB and 11.6 MB less memory during inference compared with the original architecture (1-st row). While the performance of the first network (Plain*) drops by 0.28% in F1-score, 0.44 % in mIOU and remains approximately the same in terms of overall accuracy, the performance of second network (Plain$^+$) drops by larger margin of 1.4%6 in

F1-score, 2.16 % in mIOU and 0.45% in overall accuracy. Such results imply that, to some extent, reducing the number of convolutional filters within a specific network can reduce the network memory consumption while preserving its performance, but this must not be overdone.

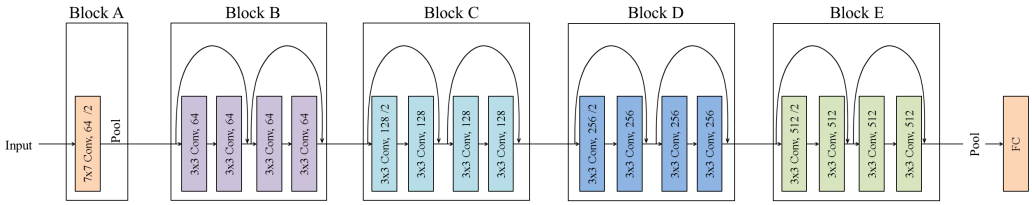


Fig. 5.5 ResNet with 18 layers depicted in five blocks.

Encoder depth: Finally, we measure how a deeper encoder can improve network performance. It is well known that deeper networks generalize better over large datasets[95]. Moreover, it has been shown that among deeper networks, residual networks (ResNets) have better generalization ability and can converge faster thanks to the employment of skip connection [26, 36]. Hence in this experiment, we deepen the encoder by using residual networks with 18, 34 and 50 layers in the encoding stage of the proposed network. As a result, the convolution layers in the encoder are replaced by residual blocks as shown in Figure 5.5 for ResNet-18, while the decoder architecture remains the same. As can be seen from the last three rows in Table 5.1, the residual encoder brings a considerable improvement in terms of both F1-score and overall accuracy. Using ResNet with 18, 34 and 50 layers as encoder leads to a increase of 2.23%, 2.03% and 1.41% in F1-score, 3.32 %, 2.92 % and 2.09 % in mIOU and 1%, 1.18% and 0.9% in overall accuracy respectively. However, the number of network parameters grows by a factor of 13, 21 and 81 for ResNet 18, 34 and 50. In terms of inference time for the forward pass, as can be seen from Table 5.2, using ResNets as encoder leads to an increase by a margin of 239, 368 and 565 ms for depth of 18,34 and 50 layers respectively. Therefore, using deeper residual networks can improve cloud detection performance significantly, but it comes at the expense of an increased cost in terms of memory consumption and inference time.

Comparison with state-of-the-art: Finally, we compare the performance of the proposed network with that of Deeplab V3+ [12] which achieved state-of-the-art performance over PASCAL VOC 2012 [19] and Cityscapes [14] datasets. Deeplab V3+ also makes use of encoder-decoder architecture to perform image segmentation. Authors showed in [18] that Deeplab V3+ with ResNet-101 and Xception encoders

obtained the best performance. Therefore, we also train Deeplab V3+ once with ResNet-101 and another time with Xception encoder over SPARCS dataset. As can be seen in the last two rows of Table 5.1, for the cloud screening task Deeplab V3+ with ResNet encoder outperforms its variant with Xception encoder by a considerable margin in overall accuracy and F1-score. In terms of memory consumption, comparing Deeplab with our proposed network and both with ResNet as the encoder, Deeplab has fewer parameters and occupies much less memory. This is due to the Deeplab architecture which does not include deconvolutional filters in its decoder and relies on upsampling operations which do not include parameters. Nevertheless, as can be seen from the results, our proposed network with the plain encoder (Plain* in Table 5.1) outperforms Deeplab in both overall accuracy and F1-score while consuming about 71 times less memory. We conjecture such decrease in Deeplab performance is related to the use of deeper encoder which makes it prone to overfitting. Analogously, we have noticed a similar decline in our proposed network performance when the encoder becomes larger. For instance, a drop in F1-score and mIOU can be seen in Table 5.1 by comparing the proposed network with ResNet 18 and 50 as the encoder. In addition to overfitting, we notice that the deconvolutional layers which are used in our network instead of bilinear upsampling layers in Deeplab, contributes to higher performance however increase the number of network parameters. Concerning the processing time, since the upsampling operation in Deeplab decoder is carried out on CPU in Pytorch, Deeplab takes more time compared with our proposed network. In addition to Deeplab V3+, we train the original U-net architecture over SPARCS dataset. Due to the differences which are highlighted in Sec.5.1.1, U-net consumes about 300 MB more memory during inference comparing with our proposed network with the plain encoder (1st row), however, it contributes to the performance by a margin of 0.65 and 1.36 in F1-score and mIOU respectively. Nevertheless, considering the proposed network with ResNet18 encoder, it not only outperforms original U-net in performance but also consumes less memory during inference. In the end, F-mask [115] algorithm is applied over the SPARCS test images using all available 10 bands. As Table 5.1 reports, CNN based approaches surpass F1-mask by a great margin.

Chapter 6

Conclusions

In this thesis, first we addressed fine-grained object recognition over vehicle makes and models, secondly, we addressed the segmentation of satellite images in a global-scale and in the end, we studied CNN-based approach for cloud screening to be implemented on satellite platform.

In the first problem, we addressed the VMMR problem via a multi-scale attention windows CNN architecture. Our architecture enables locating most discriminative parts of a vehicle at different scales by minimizing an end-to-end joint make-model classification error. Additionally, the proposed network architecture can be trained over two types of ground truth enabling simultaneous prediction on two vehicle attributes thanks to the devised loss function and the classifier module with multiple outputs. The module demonstrated state-of-art performance over two publicly available car datasets. Future direction of this work can address broader range of fine-grained as well as coarse grained classification tasks.

In the second work, we designed an encoder-decoder CNN for satellite image segmentation deployable over images different from those used from training. Our experiments revealed that residual encoders offer better generalization abilities than a plain convolutional counterpart, and that generalization ability improves with the encoder depth. We hypothesize that deep residual architectures with a large number of filters spanning across a wider range of semantic levels allow the encoder to learn more robust features to covariate shift. To further improve the segmentation performance of the proposed network over a novel image, we proposed two domain adaptation techniques. First method entails adopting a previously trained network using smaller number of samples over the novel image and in an

active learning framework which we experimentally show that such approach can significantly improve the segmentation. Secondly, we devised a domain adaptation methodology based on fine-tuning batch normalization statistics over the novel image which empirically show considerable segmentation improvement without requiring annotations over novel image. Finally, the proposed architecture outperformed multiple references over two datasets characterized by large differences between train and test images.

Finally, we addressed the cloud screening problem by considering the hardware constraints imposed by the satellite platforms. We proposed an encoder-decoder CNN to perform pixel-based classification on satellite images, in order to discard images which are highly contaminated by clouds to preserve resources. To enable an onboard implementation, we limit the resource consumption by CNN while preserving its performance by optimizing the network architecture. The results show that the proposed network can outperform the state-of-the-art CNN over SPARCS dataset while consuming much less resources during inference. The memory consumption allows these networks to easily fit in the memory of low-power accelerators for embedded applications. The sustainable throughput depends on the specific accelerator and its integration with the CPU. Our results show that the time needed for the forward pass of the neural network is small on a powerful GPU, whereas a low-power accelerator would require more time. Moreover, a significant amount of time is devoted to data transfers between CPU and GPU; in an onboard implementation, these would be highly dependent on the specific architecture of the processing unit, highlighting the need of a careful design.

6.1 Future Work

As we have shown the effectiveness of multi-scale attention windows on VMMR, further research is required to extend the proposed methodology to other fine-grained classification tasks. We believe that the proposed multi-scale attention mechanism has the potential to capture discriminative features over other objects rather than vehicles, hence it can be advantageous for many other classification applications. Moreover, since in the proposed methodology the multi-scale constraint is imposed on the attention windows by pretraining the convolutional layers over patches with different scales, the training should be performed in two stages. Therefore, further

advancement of the proposed architecture can facilitate such pretraining stage if the multi-scale constraint can be imposed during the overall architecture training.

Regarding the proposed methodology for satellite image segmentation on heterogeneous datasets, further studies can be undertaken to explore the use of proposed domain adaptation methodologies in other transfer learning problems. In particular, the developed BN statistics refinement can be investigated in other domain adaptations scenarios. Due to the use of BN layers in almost all recent neural networks, such domain adaptation technique can be advantageous to many other problems and can be explored through future experimentation.

The proposed cloud screening CNN has shown to be applicable to low power satellite platforms while performing with high accuracy thanks to its memory efficient architecture. In the future, further experimental investigations should measure the actual throughput and memory consumption of the network after its implementation over low-memory accelerators.

References

[1] European Parliament (EP). *Directive 2010/40/EU of the European Parliament and of the Council of 7 July 2010 on the framework for the deployment of Intelligent Transport Systems in the field of road transport and for interfaces with other modes of transport*. 2010.

[2] Hasan Arief et al. "Land cover segmentation of airborne LiDAR data using stochastic atrous network". In: *Remote Sensing* 10.6 (2018), p. 973.

[3] Nicolas Audebert, Bertrand Le Saux, and Sébastien Lefèvre. "Semantic segmentation of earth observation data using multimodal and multi-scale deep networks". In: *Asian Conference on Computer Vision*. Springer. 2016, pp. 180–196.

[4] Jón Atli Benediktsson, Jón Aevar Palmason, and Johannes R Sveinsson. "Classification of hyperspectral data from urban areas based on extended morphological profiles". In: *IEEE Transactions on Geoscience and Remote Sensing* 43.3 (2005), pp. 480–491.

[5] Jon Atli Benediktsson, Martino Pesaresi, and Kolbeinn Amason. "Classification and feature extraction for remote sensing images from urban areas based on morphological transformations". In: *IEEE Transactions on Geoscience and Remote Sensing* 41.9 (2003), pp. 1940–1949.

[6] Yoshua Bengio et al. "Greedy layer-wise training of deep networks". In: *Advances in neural information processing systems*. 2007, pp. 153–160.

[7] Mohsen Biglari, Ali Soleimani, and Hamid Hassanpour. "A Cascaded Part-Based System for Fine-Grained Vehicle Classification". In: *IEEE Transactions on Intelligent Transportation Systems* 19.1 (2018), pp. 273–283.

[8] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[9] Lorenzo Bruzzone and Claudio Persello. "A novel approach to the selection of spatially invariant features for the classification of hyperspectral images with improved generalization capability". In: *IEEE Transactions on Geoscience and Remote Sensing* 47.9 (2009), pp. 3180–3191.

[10] Florian Chabot et al. "Deep MANTA: A Coarse-to-fine Many-Task Network for joint 2D and 3D vehicle analysis from monocular image". In: *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.(CVPR)*. 2017, pp. 2040–2049.

[11] Yuning Chai, Victor Lempitsky, and Andrew Zisserman. “Symbiotic segmentation and part localization for fine-grained categorization”. In: *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE. 2013, pp. 321–328.

[12] Liang-Chieh Chen et al. “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.4 (2018), pp. 834–848.

[13] Ronan Collobert and Jason Weston. “A unified architecture for natural language processing: Deep neural networks with multitask learning”. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 160–167.

[14] Marius Cordts et al. “The cityscapes dataset for semantic urban scene understanding”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 3213–3223.

[15] Michael Cramer. “The DGPF-test on digital airborne camera evaluation–overview and test design”. In: *Photogrammetrie-Fernerkundung-Geoinformation* 2010.2 (2010), pp. 73–82.

[16] Cheng Deng et al. “Active transfer learning network: a unified deep joint spectral-spatial feature learning model for hyperspectral image classification”. In: *IEEE Transactions on Geoscience and Remote Sensing* (2018).

[17] Ahmed Elshamli et al. “Domain adaptation using representation learning for the classification of remote sensing images”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 10.9 (2017), pp. 4198–4209.

[18] “Encoder-decoder with atrous separable convolution for semantic image segmentation”. In: *European Conference on Computer Vision*. 2018, pp. 833–851.

[19] Mark Everingham et al. “The pascal visual object classes challenge: A retrospective”. In: *International journal of computer vision* 111.1 (2015), pp. 98–136.

[20] Clement Farabet et al. “Learning hierarchical features for scene labeling”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1915–1929.

[21] Adrian Fisher. “Cloud and cloud-shadow detection in SPOT5 HRG imagery with automated morphological feature extraction”. In: *Remote Sensing* 6.1 (2014), pp. 776–800.

[22] A.M. Francis, P. Sidiropoulos, and E. Vazquez. “Real-Time Cloud Detection in High-Resolution Videos: Challenges and Solutions”. In: *Proc. of Onboard Payload Data Compression Workshop*. 2018.

[23] Yaroslav Ganin et al. “Domain-adversarial training of neural networks”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 2096–2030.

[24] Sina Ghassemi et al. “Fine-grained vehicle classificationusing deep residual networks with multiscale attention windows”. In: *Multimedia Signal Processing (MMSP), 2017 IEEE 19th International Workshop on*. IEEE. 2017, pp. 1–6.

[25] Sina Ghassemi et al. “Learning and Adapting Robust Features for Satellite Image Segmentation on Heterogeneous Datasets”. In: *IEEE Transactions on Geoscience and Remote Sensing, under review* ().

[26] Sina Ghassemi et al. “Satellite image segmentation with deep residual architectures for time-critical applications”. In: *26th European Signal Processing Conference*. IEEE. 2018, pp. 2235–2239.

[27] Sina Ghassemi et al. “Vehicle joint make and model recognition with multiscale attention windows”. In: *Signal Processing: Image Communication* 72 (2019), pp. 69–79.

[28] Golnaz Ghiasi and Charless C Fowlkes. “Laplacian pyramid reconstruction and refinement for semantic segmentation”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 519–534.

[29] Girshick. “Fast R-CNN”. In: *IEEE International Conference on Computer Vision*. IEEE. 2015, pp. 1440–1448.

[30] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[31] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[32] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 2672–2680.

[33] Philippe-Henri Gosselin et al. “Revisiting the fisher vector for fine-grained classification”. In: *Pattern recognition letters* 49 (2014), pp. 92–98.

[34] M Griggin et al. “Cloud cover detection algorithm for EO-1 Hyperion imagery”. In: *Geoscience and Remote Sensing Symposium, 2003. IGARSS’03. Proceedings. 2003 IEEE International*. Vol. 1. IEEE. 2003, pp. 86–89.

[35] Hongsheng He, Zhenzhou Shao, and Jindong Tan. “Recognition of car makes and models from a single traffic-camera image”. In: *IEEE Transactions on Intelligent Transportation Systems* 16.6 (2015), pp. 3182–3192.

[36] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[37] Kaiming He et al. “Mask r-cnn”. In: *IEEE International Conference on Computer Vision*. IEEE. 2017, pp. 2980–2988.

[38] Geoffrey E Hinton et al. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *arXiv preprint arXiv:1207.0580* (2012).

[39] Jun-Wei Hsieh, Li-Chih Chen, and Duan-Yu Chen. "Symmetrical surf and its applications to vehicle detection and vehicle make and model recognition". In: *IEEE Transactions on intelligent transportation systems* 15.1 (2014), pp. 6–20.

[40] Qichang Hu et al. "Deep CNNs With Spatially Weighted Pooling for Fine-Grained Car Recognition". In: *IEEE Transactions on Intelligent Transportation Systems* 18.11 (2017), pp. 3147–3156.

[41] Bohao Huang et al. "Large-scale semantic classification: outcome of the first year of Inria aerial image labeling benchmark". In: *IEEE International Geoscience and Remote Sensing Symposium–IGARSS 2018*. 2018.

[42] Xin Huang and Liangpei Zhang. "A multidirectional and multiscale morphological index for automatic building extraction from multispectral GeoEye-1 imagery". In: *Photogrammetric Engineering & Remote Sensing* 77.7 (2011), pp. 721–732.

[43] Xin Huang and Liangpei Zhang. "Morphological building/shadow index for building extraction from high-resolution imagery over urban areas". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 5.1 (2012), pp. 161–172.

[44] M Joseph Hughes and Daniel J Hayes. "Automated detection of cloud and cloud shadow in single-date Landsat imagery using neural networks and spatial post-processing". In: *Remote Sensing* 6.6 (2014), pp. 4907–4926.

[45] Shilpa Inamdar et al. "Multidimensional probability density function matching for preprocessing of multitemporal remote sensing images". In: *IEEE Transactions on Geoscience and Remote Sensing* 46.4 (2008), pp. 1243–1252.

[46] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[47] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. "Spatial transformer networks". In: *Advances in neural information processing systems*. 2015, pp. 2017–2025.

[48] Xiuping Jia, Bor-Chen Kuo, and Melba M Crawford. "Feature mining for hyperspectral image classification". In: *Proceedings of the IEEE* 101.3 (2013), pp. 676–697.

[49] Goo Jun and Joydeep Ghosh. "An efficient active learning algorithm with knowledge transfer for hyperspectral data analysis". In: *Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International*. Vol. 1. IEEE. 2008, pp. I–52.

[50] Michael Kampffmeyer, Arnt-Borre Salberg, and Robert Jenssen. "Semantic segmentation of small objects and modeling of uncertainty in urban remote sensing images using deep convolutional neural networks". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition workshops*. 2016, pp. 1–9.

[51] Jonathan Krause et al. “3D Object Representations for Fine-Grained Categorization”. In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013.

[52] Jonathan Krause et al. “3d object representations for fine-grained categorization”. In: *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*. IEEE. 2013, pp. 554–561.

[53] Jonathan Krause et al. “Fine-grained recognition without part annotations”. In: *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE. 2015, pp. 5546–5555.

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[55] Anders Krogh and John A Hertz. “A simple weight decay can improve generalization”. In: *Advances in neural information processing systems*. 1992, pp. 950–957.

[56] Matthieu Le Goff et al. “Deep learning for cloud detection”. In: (2017).

[57] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), p. 436.

[58] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[59] Liang Liao et al. “Exploiting effects of parts in fine-grained categorization of vehicles”. In: *Image Processing (ICIP), 2015 IEEE International Conference on*. IEEE. 2015, pp. 745–749.

[60] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. “Bilinear cnn models for fine-grained visual recognition”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1449–1457.

[61] David Fernández Llorca et al. “Vehicle model recognition using geometry and appearance of car emblems from rear view images”. In: *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*. IEEE. 2014, pp. 3094–3099.

[62] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3431–3440.

[63] Emmanuel Maggiori et al. “Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark”. In: *IEEE International Symposium on Geoscience and Remote Sensing (IGARSS)*. 2017.

[64] Dimitrios Marmanis et al. “Classification with an edge: Improving semantic image segmentation with boundary detection”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 135 (2018), pp. 158–172.

[65] Giona Matasci, Devis Tuia, and Mikhail Kanevski. “SVM-based boosting of active learning strategies for efficient domain adaptation”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 5.5 (2012), pp. 1335–1343.

[66] Giona Matasci et al. “Semisupervised transfer component analysis for domain adaptation in remote sensing image classification”. In: *IEEE Transactions on Geoscience and Remote Sensing* 53.7 (2015), pp. 3550–3564.

[67] Farid Melgani and Lorenzo Bruzzone. “Classification of hyperspectral remote sensing images with support vector machines”. In: *IEEE Transactions on Geoscience and Remote Sensing* 42.8 (2004), pp. 1778–1790.

[68] CJ Merchant et al. “Probabilistic physically based cloud screening of satellite infrared imagery for operational sea surface temperature retrieval”. In: *Quarterly Journal of the Royal Meteorological Society* 131.611 (2005), pp. 2735–2755.

[69] Sorour Mohajerani, Thomas A Krammer, and Parvaneh Saeedi. “A Cloud Detection Algorithm for Remote Sensing Images Using Fully Convolutional Neural Networks”. In: *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*. IEEE. 2018, pp. 1–5.

[70] Abdel-rahman Mohamed et al. “Deep belief networks using discriminative features for phone recognition”. In: *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE. 2011, pp. 5060–5063.

[71] Loredana Murino et al. “Cloud detection of MODIS multispectral images”. In: *Journal of Atmospheric and Oceanic Technology* 31.2 (2014), pp. 347–365.

[72] Allan A Nielsen and Morton J Canty. “Kernel principal component and maximum autocorrelation factor analyses for change detection”. In: *Image and Signal Processing for Remote Sensing XV*. Vol. 7477. International Society for Optics and Photonics. 2009, 74770T.

[73] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. “Learning deconvolution network for semantic segmentation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1520–1528.

[74] Sakrapee Paisitkriangkrai et al. “Effective semantic pixel labelling with convolutional networks and conditional random fields”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2015, pp. 36–43.

[75] Sakrapee Paisitkriangkrai et al. “Semantic labeling of aerial and satellite imagery”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 9.7 (2016), pp. 2868–2881.

[76] Teerapong Panboonyuen et al. “Semantic segmentation on remotely sensed images using an enhanced global convolutional network with channel attention and domain specific transfer learning”. In: *Remote Sensing* 11.1 (2019), p. 83.

[77] Adam Paszke et al. “Automatic differentiation in pytorch”. In: (2017).

[78] Claudio Persello and Lorenzo Bruzzone. “Kernel-based domain-invariant feature selection in hyperspectral images for transfer learning”. In: *IEEE Transactions on Geoscience and Remote Sensing* 54.5 (2016), pp. 2615–2626.

[79] Ning Qian. “On the momentum term in gradient descent learning algorithms”. In: *Neural networks* 12.1 (1999), pp. 145–151.

[80] Suju Rajan, Joydeep Ghosh, and Melba M Crawford. “An active learning approach to hyperspectral data classification”. In: *IEEE Transactions on Geoscience and Remote Sensing* 46.4 (2008), pp. 1231–1242.

[81] Girshick Ren He and Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Neural Information Processing Systems 28*. NIPS. 2015.

[82] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[83] Riccardo Rossi et al. “Techniques based on support vector machines for cloud detection on quickbird satellite imagery”. In: *Geoscience and Remote Sensing Symposium (IGARSS), 2011 IEEE International*. IEEE. 2011, pp. 515–518.

[84] Sara Saravi and Eran A Edirisinghe. “Vehicle make and model recognition in CCTV footage”. In: (2013).

[85] Ravi Kumar Satzoda and Mohan Manubhai Trivedi. “Multipart vehicle detection using symmetry-derived analysis and active learning”. In: *IEEE Transactions on Intelligent Transportation Systems* 17.4 (2016), pp. 926–937.

[86] R Wl Saunders and K Ts Kriebel. “An improved method for detecting clear sky and cloudy radiances from AVHRR data”. In: *International Journal of Remote Sensing* 9.1 (1988), pp. 123–150.

[87] Mengyun Shi et al. “Cloud detection of remote sensing images by deep learning”. In: *Geoscience and Remote Sensing Symposium (IGARSS), 2016 IEEE International*. IEEE. 2016, pp. 701–704.

[88] Smadar Shiffman. “Cloud detection from satellite imagery: A comparison of expert-generated and automatically-generated decision trees”. In: (2004).

[89] Patrice Y Simard, Dave Steinkraus, and John C Platt. “Best practices for convolutional neural networks applied to visual document analysis”. In: *null*. IEEE. 2003, p. 958.

[90] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[91] Jakub Sochor, Adam Herout, and Jiri Havel. “Boxcars: 3d boxes as cnn input for improved fine-grained vehicle recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 3006–3015.

[92] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[93] Masashi Sugiyama, Neil D Lawrence, Anton Schwaighofer, et al. *Dataset shift in machine learning*. The MIT Press, 2017.

[94] Richard Sutton. “Two problems with back propagation and other steepest descent learning procedures for networks”. In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*. 1986, pp. 823–832.

[95] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[96] David R Thompson et al. “Rapid spectral cloud screening onboard aircraft and spacecraft”. In: *IEEE Transactions on Geoscience and Remote Sensing* 52.11 (2014), pp. 6779–6792.

[97] Devis Tuia, Claudio Persello, and Lorenzo Bruzzone. “Domain adaptation for the classification of remote sensing data: An overview of recent advances”. In: *IEEE Geoscience and Remote Sensing Magazine* 4.2 (2016), pp. 41–57.

[98] “U.S. Geological Survey, L8 SPARCS Cloud Validation Masks. U.S. Geological Survey data release”. In: (2016).

[99] Michele Volpi and Devis Tuia. “Dense semantic labeling of subdecimeter resolution images with convolutional neural networks”. In: *IEEE Transactions on Geoscience and Remote Sensing* 55.2 (2017), pp. 881–893.

[100] Kiri L Wagstaff et al. “Cloud Filtering and Novelty Detection using Onboard Machine Learning for the EO-1 Spacecraft”. In: (2017).

[101] Hongzhen Wang et al. “Gated convolutional neural network for semantic segmentation in high-resolution images”. In: *Remote Sensing* 9.5 (2017), p. 446.

[102] Xi Wu and Zhenwei Shi. “Utilizing multilevel features for cloud detection on satellite imagery”. In: *Remote Sensing* 10.11 (2018), p. 1853.

[103] Yongyang Xu et al. “Road extraction from high-resolution remote sensing imagery using deep learning”. In: *Remote Sensing* 10.9 (2018), p. 1461.

[104] Hsiuhan Lexie Yang and Melba M Crawford. “Spectral and spatial proximity-based manifold alignment for multitemporal hyperspectral image classification”. In: *IEEE Transactions on Geoscience and Remote Sensing* 54.1 (2016), pp. 51–64.

[105] Linjie Yang et al. “A large-scale car dataset for fine-grained categorization and verification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3973–3981.

[106] Xiwen Yao et al. “Semantic annotation of high-resolution satellite images via weakly supervised learning”. In: *IEEE Transactions on Geoscience and Remote Sensing* 54.6 (2016), pp. 3660–3671.

[107] Sergey Zagoruyko and Nikos Komodakis. “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146* (2016).

[108] Matthew D Zeiler et al. “Deconvolutional networks”. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE. 2010, pp. 2528–2535.

[109] Lefei Zhang et al. “Ensemble manifold regularized sparse low-rank approximation for multiview feature embedding”. In: *Pattern Recognition* 48.10 (2015), pp. 3102–3112.

[110] Lefei Zhang et al. “On combining multiple features for hyperspectral remote sensing image classification”. In: *IEEE Transactions on Geoscience and Remote Sensing* 50.3 (2012), pp. 879–893.

[111] Liangpei Zhang et al. “A pixel shape index coupled with spectral information for classification of high spatial resolution remotely sensed imagery”. In: *IEEE Transactions on Geoscience and Remote Sensing* 44.10 (2006), pp. 2950–2961.

[112] Ning Zhang et al. “Part-based R-CNNs for fine-grained category detection”. In: *European conference on computer vision*. Springer. 2014, pp. 834–849.

[113] Zhang Zhaoxiang et al. “Small satellite cloud detection based on deep learning and image compression”. In: ("Preprint Feb. 2018, available at www.preprints.org").

[114] Kaiqiang Zhu et al. “Deep convolutional capsule network for hyperspectral image spectral and spectral-spatial classification”. In: *Remote Sensing* 11.3 (2019), p. 223.

[115] Zhe Zhu, Shixiong Wang, and Curtis E Woodcock. “Improvement and expansion of the Fmask algorithm: Cloud, cloud shadow, and snow detection for Landsats 4–7, 8, and Sentinel 2 images”. In: *Remote Sensing of Environment* 159 (2015), pp. 269–277.

[116] Zhe Zhu and Curtis E Woodcock. “Object-based cloud and cloud shadow detection in Landsat imagery”. In: *Remote sensing of environment* 118 (2012), pp. 83–94.

Appendix A

VMMR with Conditioned Spatial Pooling

A.1 Conditioned Spatial Pooling

In this part, we briefly describe a classification architecture similar to the architecture proposed in Sec. 3. However, the proposed architecture differs in several aspects with designed multi-scale classification in Sec. 3.

First of all, the proposed architecture as it is shown in Fig. A.1 aims to predict *attention masks* in place of attention windows and in the form of spatially weighted pooling masks. Secondly, rather than realizing and aggregating visual representations over multiple scales as performed in Sec. 3, here we are interested in defining most discriminative parts of a vehicle in pixel-level. Accordingly, we replace common average pooling layer in classifier with a spatially weighted pooling layer as shown in Fig. A.1 and similar to the proposed pooling layer in [40] however with a major difference.

Authors in [40] employ a spatially weighted pooling with learnable masks in place of penultimate commonly used global pooling before fully connected classifier. Therefore, for each generated feature map by the classifier, one or several masks with learnable parameters are optimized during training. Such masks are used in weighted pooling layer to magnifying those pixels of feature maps which belongs to discriminative parts of the image, as well as diminishing feature maps pixels which belong to part of the image of less significance. However, the learned masks in [40]

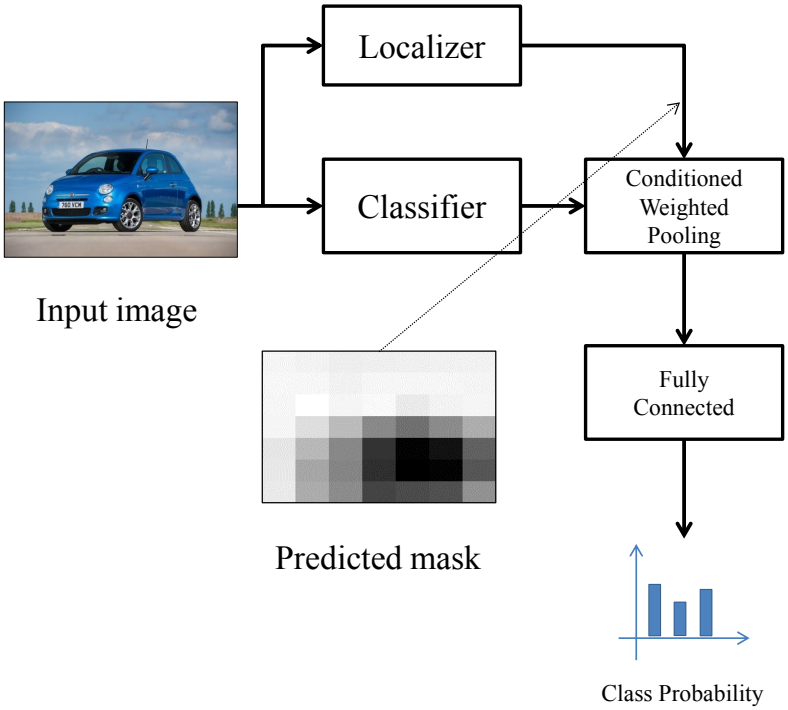


Fig. A.1 Proposed architecture.

are constant for every image which challenges its robustness for images where object scale and position varies.

In this work, we proposed a scheme with spatial learnable pooling layer replacing penultimate global pooling in CNN similar to [40], however such pooling uses masks that are conditioned on input image, unlike [40] where masks are fixed for every image. Similar to the architecture proposed in Sec. 3.2, a localizer module is employed. Nonetheless, the localizer module predicts spatially weighted masks which are then used by pooling layer to pool feature maps. Therefore, depending on the input image, localizer predicts a number of pooling masks where more deterministic pixels in feature maps are assign bigger weight than less discriminative pixels.

In the following, we provide details on network architecture and also preliminary results. Thus far, the results of predicted masks indicate that localizer can indeed determine discriminative feature maps pixels, however, we could not be able to

improve the classification accuracy using conditioned pooling comparing with global pooling. Hence, we additionally discuss the possible issues with the current proposed scheme which we could not explore further regarding thesis scope and time constraint, however, we conjecture that if they are addressed properly in future, classification would benefit from conditioned weighted pooling.

A.1.1 Architecture

The localizer module includes a ResNet without the penultimate pooling and the last fully connected layers. Additionally, a 1×1 convolutional layer is added which takes as input generated features sized $h \times w$ and outputs N_{mask} $h \times w$ feature maps which will be used as N_{mask} attention masks. These masks can be represented by matrix M:

$$M_{N_d \times N_m} = \begin{pmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,N_m} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,N_m} \\ \vdots & \vdots & \ddots & \vdots \\ f_{N_d,1} & f_{N_d,2} & \cdots & f_{N_d,N_m} \end{pmatrix} \tag{A.1}$$

where $N_d = hw$. Hence, every mask is vectorized over its height and width and represented as one row in M.

The classifier module also is ResNet without the penultimate pooling and last fully connected layers. Therefore, it generates N_f feature maps having hight and width of h and w respectively. As a results it can be seen as a matrix F:

$$F_{N_f \times N_d} = \begin{pmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,N_d} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,N_d} \\ \vdots & \vdots & \ddots & \vdots \\ f_{N_f,1} & f_{N_f,2} & \cdots & f_{N_f,N_d} \end{pmatrix} \tag{A.2}$$

where $N_d = hw$. Hence, every feature map is vectorized over its height and width and represented as one row in F.

Then, the conditioned spatial weighted pooling can be seen as matrix multiplier as follows:

$$V_{N_f \times N_m} = F_{N_f \times N_d} \times M_{N_d \times N_m} \tag{A.3}$$

where V is the resulting pooling feature maps using weighted masks with the following form:

$$V_{N_f \times N_m} = \begin{pmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,N_m} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,N_m} \\ \vdots & \vdots & \ddots & \vdots \\ f_{N_f,1} & f_{N_f,2} & \cdots & f_{N_f,N_m} \end{pmatrix} \tag{A.4}$$

Next, the matrix V is vectorized and fed into fully connected layers with a number of inputs of $N_f \times N_m$ and a number of outputs equal to the number of classes.

A.1.2 Results

We have trained the proposed architecture with ResNet18 in localizer and ResNet50 in classifier over Standford car datasets. Therefore $N_f = 2048$, $h, w = 7$ and $N_d = 49$ and we chose only 1 mask $N_m = 1$, hence the size of fully connected layer remains the same comparing with Standard ResNet.

The classifier base learning rate is 10^{-3} and localizer base learning rate is set to 10^{-4} and every 50 epochs it divided by a factor of 10. The training is carried out for 200 epochs using SGD optimization with weight decay of 5×10^{-3}.

Fig A.2 shows the predicted masks for a number of different samples of Stanford dataset. As it is evident, the predicted mask assigns more weights (i.e. black regions) to deterministic areas in the images Therefore the localizer is able to accurately realize significant pixels in terms of representation value for classification. However, numerical classification results imply that the fully connected layer could not benefit from such spatially weighted mask applied in pooling layer compare with regular average pooling.

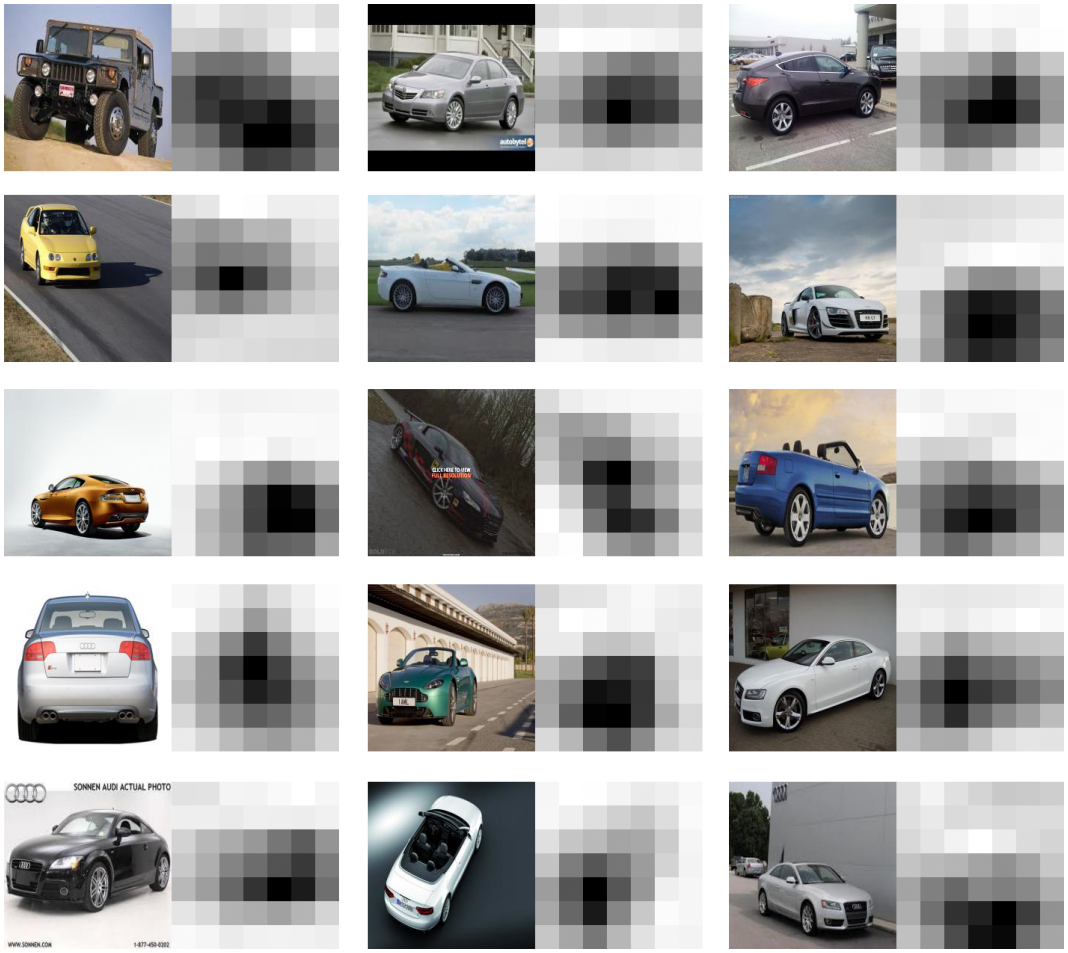


Fig. A.2 Samples of Stanford datasets with corresponding predicted 7×7 masks.

Nevertheless, as it has been proven in [40] that weighted pooling aims to better classification, and regarding that our proposed architecture enjoys conditioned weighted pooling unlike its unconditioned variant proposed in [40], we conjuncture that the classification can be improved by further exploring and improving the proposed architecture. Such as applying the conditioned weighted pooling layer in classifier intermediate layers where feature maps pixels are less correlated and therefore can benefit more from a weighted pooling, or introducing a mask for each generated feature map by taking to account that a particular feature map may have a deterministic region which differs from another feature map.