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

Doctoral Dissertation
Doctoral Program in Electronical, Electronics and Communications Engineering
($30^{th}$cycle)

# License Plate Recognition using Convolutional Neural Networks Trained on Synthetic Images

By

**Tomas Björklund**

******

**Supervisor(s):**
Prof. E., Magli

**Doctoral Examination Committee:**
Prof. L. Marcenaro, Università degli Studi di Genova
Prof. S. Tubaro, Politecnico di Milano
Prof. M. Grangetto, Università degli Studi di Torino
Prof. T. Bianchi, Politecnico di Torino
Prof. S. Fosson, Politecnico di Torino

Politecnico di Torino
2018

# 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.

Tomas Björklund
2018

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

*For Axel*

# Acknowledgements

This thesis work has been supported by TIM, Telecom Italia S.p.A.

I would like to thank my supervisor Prof. Enrico Magli for his support and valuable advice, as well as Gianluca Francini and Attilio Fiandrotti who have also helped out in this work.

Finally, thanks for all the support I have received from all my friends, including all my colleagues, and my family.

# Abstract

In this thesis, we propose a license plate recognition system and study the feasibility of using synthetic training samples to train convolutional neural networks for a practical application.

First we develop a modular framework for synthetic license plate generation; to generate different license plate types (or other objects) only the first module needs to be adapted. The other modules apply variations to the training samples such as background, occlusions, camera perspective projection, object noise and camera acquisition noise, with the aim to achieve enough variation of the object that the trained networks will also recognize real objects of the same class.

Then we design two convolutional neural networks of low-complexity for license plate detection and character recognition. Both are designed for simultaneous classification and localization by branching the networks into a classification and a regression branch and are trained end-to-end simultaneously over both branches, on only our synthetic training samples.

To recognize real license plates, we design a pipeline for scale invariant license plate detection with a scale pyramid and a fully convolutional application of the license plate detection network in order to detect any number of license plates and of any scale in an image. Before character classification is applied, potential plate regions are un-skewed based on the detected plate location in order to achieve an as optimal representation of the characters as possible. The character classification is also performed with a fully convolutional sweep to simultaneously find all characters at once.

Both the plate and the character stages apply a refinement classification where initial classifications are first centered and rescaled. We show that this simple, yet effective trick greatly improves the accuracy of our classifications, and at a small increase of complexity. To our knowledge, this trick has not been exploited before.

To show the effectiveness of our system we first apply it on a dataset of photos of Italian license plates to evaluate the different stages of our system and which effect the classification thresholds have on the accuracy. We also find robust training parameters and thresholds that are reliable for classification without any need for calibration on a validation set of real annotated samples (which may not always be available) and achieve a balanced precision and recall on the set of Italian license plates, both in excess of 98%.

Finally, to show that our system generalizes to new plate types, we compare our system to two reference system on a dataset of Taiwanese license plates. For this, we only modify the first module of the synthetic plate generation algorithm to produce Taiwanese license plates and adjust parameters regarding plate dimensions, then we train our networks and apply the classification pipeline, using the robust parameters, on the Taiwanese reference dataset. We achieve state-of-the-art performance on plate detection (99.86% precision and 99.1% recall), single character detection (99.6%) and full license reading (98.7%).

# Contents

# List of Figures

# List of Tables

# Nomenclature

### Roman Symbols

| | |
|---|---|
| $(\cdot)^+$ | Moore-Penrose pseudoinverse |
| $\lfloor \cdot \rfloor$ | Floor operator, i.e. rounding down |
| $C$ | number of character classification labels |
| $f$ | focal length |
| $i$ | integer valued coordinate on the horizontal axis |
| $j$ | integer valued coordinate on the vertical axis |
| $K$ | camera internal parameter matrix |
| $x$ | real valued coordinate on the horizontal axis |
| $x_0$ | central x-coordinate offset |
| $y$ | real valued coordinate on the vertical axis |
| $y_0$ | central y-coordinate offset |

### Greek Symbols

| | |
|---|---|
| $\phi$ | rotation around the horizontal (pitch) |
| $\psi$ | rotation around the optical axis (roll) |
| $\sigma_x$ | internal camera parameter for horizontal scaling |
| $\sigma_y$ | internal camera parameter for vertical scaling |

$\theta$ rotation around the vertical axis (yaw)

**Acronyms / Abbreviations**

ANN Artificial Neural Network

AOLP Application Oriented License Plate dataset. a test set of Taiwanese license plates

AUROC Area Under Receiver Operating Characteristic curve

CCA Connected Component Analysis

CDVS Compact Descriptors for Visual Search

CNN Convolutional Neural Network

FN False Negative classification

FP False Positive classification

FPR False Positive Rate

GAN Generative Adversarial Network

HMM Hidden Markov Model

HSI Hue-Saturation-Intensity color space

HSL Hue-Saturation-Lightness color space

IoU Intersection over Union

LPD License Plate Detection classification task

LPDR License Plate Detection and Recognition classification task

LPR License Plate Recognition

LR License Recognition classification task

MSE Mean Square Error

PSF Point Spread Function

RBM Restricted Boltzmann Machine

ReLU Rectified Linear Unit, non-linear activation function

RGB Red-Green-Blue image, the color space of a common digital image

ROC curve Receiver Operating Characteristic curve

SGD Stochastic Gradient Decent

SVM Support Vector Machine

TanH Hyperbolic Tangent, non-linear activation function

TN True Negative classification

TP True Positive classification

TPR True Positive Rate

# Chapter 1

# Introduction

This thesis describes the research I carried out between November 2014 until October 2017 at the TIM Joint Open Lab and Politecnico di Torino, Department of Electronics and Telecommunications. This chapter starts with the motivation for this work, followed by a literature review on License Plate Recognition (LPR). Chapter 2 gives an introduction to Convolutional Neural Networks (CNNs). In chapter 3 the synthetic sample generation method used is described, which is used to generate the samples to train the CNNs described in chapter 4. In chapter 5 the classification pipeline using the trained networks is explained and finally, chapter 6 presents the experimental results obtained.

## 1.1 Motivation

In License Plate Recognition systems [1], vehicular traffic images or video flows are captured, sent to a remote center and processed to detect the presence of plates (plate detection) and to read the characters in each plate (character detection). LPR is expected to be a key enabling technology to the smart cities of tomorrow, where a network of cameras deployed at every crossroad identifies the vehicles to track them as they move through the city.

Existing LPR systems are largely based on ad-hoc imaging systems such as infrared illuminators and cameras installed on dedicated infrastructures [2]. However, a large-scale vehicle tracking infrastructure must necessarily rely on low-cost cameras, such as visible light IP surveillance cameras, mounted on already existing

infrastructures such as road poles. In such conditions, LPR becomes very challenging due to large variations in plate scale, perspective, and illumination conditions.

Deep learning and Convolutional Neural Networks (CNNs, described in chapter 2) in particular have emerged as the cornerstone to efficiently solve a number of image classification and regression problems such as character recognition [3–5] object categorization or localization [6, 7]. CNNs are feed-forward, multi-layer neural networks, with some layers connecting their weights in a convolutional manner[8] such that the same pattern of weights is shifted over the data, while allows them to search for learnt features over the whole image, while traditional networks would need to learn the features individually for each location.

With CNNs, LPR on non-dedicated hardware is now becoming feasible [9–11]. Such a solution would not only enable LPR on traffic surveillance cameras but in virtually any camera, down to dash cams and mobile phones which would also lighten work tasks in other areas, such as city police patrols and parking controls.

However, a limitation of deep learning algorithms is that they require a large amount of labelled training data. Acquiring samples alone may be problematic due to local privacy regulations. But even if enough samples can be acquired, labelling training samples is labour intensive and can become totally infeasible depending on the number of different types of license plates the LPR system should be trained to recognize.

Therefore, we are interested in investigating whether synthetic sample generation is feasible for LPR (not only for data augmentation, but as the only source of training data) and whether such a system can easily be modified to generate and classify other plate types, and whether the performance is comparable to that of systems trained on real samples. We also design our system to be practical for real-world, real-time application, therefore we also want to find a design that allows for scale-invariant, high classification accuracy, with a low complexity. Specifically, we develop a synthetic sample generation algorithm for Italian license plates, two low complexity CNNs for license plate and character recognition trained purely on the generated samples, and an LPR system for scale independent license plate detection, based on the two CNNs. We show that, with minor adaptations, we can generate Taiwanese plates to train our system and test it on the Taiwanese Application Oriented License Plate dataset (AOLP) proposed by [12] and compare our performance with [9], a CNN based LPR system trained on real character and plate samples.

## 1.2 Related works

The first automatic license plate recognition system was invented by the UK Police Scientific Development Branch in 1976. But this system still required the license number to be typed by hand from the automatically acquired images for a database search.

Until today, installed systems still use dedicated hardware which makes the automatic recognition significantly easier (see Fig. 1.1), while LPR on RGB images acquired by normal cameras is an active and open research topic [12–71]. The cost of installing dedicated hardware solutions may be difficult to motivate in many cases, e.g. when the purpose does not directly generate an income related to passing cars (e.g. by highway fees or zone fines), such as traffic flow monitoring or mobile applications.


Fig. 1.1 Example of image acquired with dedicated hardware for license plate acquisition, including IR illuminator and IR pass filter on the camera combined with task customized shutter speed and aperture settings. Downloaded from: http://elitevirtualsecurity.com/cctv-systems

Before CNNs and deep learning revolutionized the image recognition scene, LPR algorithms typically performed three image processing tasks [72] sequentially:

1. **License plate localization**
2. **Character segmentation**
3. **Character recognition**

### 1.2.1 License plate localization

License plate localization can be performed on binarized images by edge detection with a mathematical evaluation of the morphology of the edges and filter out unwanted edges [13–17] or segmentation by Connected Component Analysis (CCA) followed by spatial measurement such as area, orientation and aspect ratio of the segmented objects [18, 19].

On gray-scale images, high or maximum contrast has been used (for plates with reflective white background and black text) [20, 21] or vertical edge detection to locate high contrast text [12, 22–27].

Color image processing has also been proposed, but mainly due to country specific color characteristics in certain license plates, where specific color combinations are unique to license plates and no other objects [28, 29] and can be combined with edge detection to detect color specific edges [30]. These methods typically convert the image into a more natural color representation than RGB, such as Hue-Saturation-Intensity/Lightness (HSI/HSL).

Finally, classifiers have also been employed, using fixed features such as Haar-like features in [31–34] and classifiers such as cascaded perceptrons [32–34], Support Vector Machines (SVMs) [35, 36], Artificial Neural Networks (ANNs) [37–43] or genetic algorithms [44–47].

Reported success rates by license plate detection algorithms vary from the low 80% up to 100% but unfortunately there is little consensus of a common test set, therefore a fair comparison cannot be made, [72]. The authors contribute with a dataset of Greek license plates, but unfortunately the are not labeled. While the authors of [12] also contribute with a dataset of 2049 Taiwanese license plates for three different LPR subtasks with different requirements due to camera acquisition angle and distance, these images are also labeled with plate location and character classifications. Reported performance may also vary depending of how costly false positives are considered, some methods may prioritize recall over precision to ensure no candidates are dropped before the character segmentation and classification stages while others use a more balanced metric.

### 1.2.2 Character segmentation

Next, the identified license plate regions are analyzed to segment any present characters. A common approach is to use vertical and horizontal projection of the pixels [18, 22, 24, 26, 29, 37, 42, 48–55], which will produce strong spikes in any space between characters on the horizontal projection as well as above and under the characters on the vertical projection. CCA is also commonly used [13, 14, 38, 43, 54, 56–60] in combination with other measurements of e.g. height and width or even projections to filter out objects of undesired shapes.

Despite the high contrast between character and background, global thresholding techniques are not accurate enough [21], instead local or adaptive thresholding techniques are needed [30, 59, 61–63], e.g. Otsu Thresholding [73].

The authors of [64] recognize that simple segmentation algorithms such as projection or CCA often fail in complicated cases or outside the dataset they were developed for, and propose a two stage segmentation algorithm with a template matching a using harrow-shaped filter bank and minimum response followed by segmentations between connected or overlapping characters are adjusted by a path-finding algorithm.

### 1.2.3 Character classification

To recognize the segmented characters, classifiers are typically used but also simple pattern matching, since the font of license plates is typically known a priori. Classifiers include Hidden Markov Models [74] (HMMs) implemented by [53, 59], SVM [49], ANN [25, 38, 40, 42, 51, 56, 61, 65–70] or combinations of classifiers [75], while pattern matching has been attempted by [18, 22, 55, 71]. This typically includes computing distance metric such as root mean square error for every character pattern, shifted over the character segmentation area.

Since multiple characters need to be classified within each plate for a correct detection, the character segmentation and classification accuracy is critical for a successful reading. Good character recognition results have been reported with neural networks variants (up to 99.5% [69]) but such algorithms require a huge amount of training samples, therefore many algorithms employ some stock OCR solution instead. Most algorithm require a license plate height about 20-25 pixels

in the source image for the characters to be sufficiently resolved such that character classification is possible [72].

### 1.2.4 Available LPR Software products

While most commercial LPR software does not reveal much detail of how their algorithms work, OpenALPR Technology develops a pure software solution for LPR and keeps their software as open source [76] while providing services for their users, such as classification via a cloud service [77]. Their software use a pipeline with the following stages:

1. *Detection*, using local binary patterns to identify possible license plate regions.
2. *Binarization*, where multiple binarized crops are extracted at different binarization thresholds.
3. *Character Analysis*, searching for character sized connected blobs, regions without potential character blobs are discarded.
4. *Plate edges*, refines the plate location using hough lines.
5. *Deskew*, transforms the plate to (ideally) remove rotation and skew.
6. *Character segmentation*, uses vertical histogram to find gaps and removes the plate edges and disconnected speckles to clean up the image.
7. *OCR*, analyses all characters independently and provides confidence ratings for each character class.
8. *Post Processing*, determines the best guess of plate letters combination based on the specified plate type.

This software supports multiple plate types, although the input type need to be specified by hand before classification. The cloud API tool is very accurate but sometimes has problems with small plates and when the viewing angle is extremely high, and the reported processing time is just above 0.5 seconds for a 1 Mpixel image with a single plate.

### 1.2.5 CNN and other deep learning algorithms applied to LPR

In later years, designs incorporating deep learning techniques such as CNNs and recurrent neural networks have been proposed.

J. Li et al. [78] propose a plate detection system based on multi-scale CNNs, where the fully connected layers are fed with features extracted at different convolutional layers to achieve robustness against scale variations. The experiments show good plate detection performance under reasonable variations of the plate size in the input image. However, it is not clear to which extent such approach is suitable to cope with wide changes in plate size, as in the case of different practical LPR applications.

Menotti et al. [11] explore randomly generated CNNs for the character detection problem. First, a number of convolutional feed-forward networks where the topology parameters (e.g. number and size of the filters) are initialized at random and trained in a supervised manner. Then, an SVM is trained to classify the features extracted by the best performing network, reporting single character recognition rates of over 98% for digits and 96% for letters. Not unsurprisingly; the best performing network topologies are reported for networks where the number of filters per convolutional layer is higher, whereas the associated computational complexity is not discussed.

H. Li et al. [9] employ a combination of different networks to tackle the LPR problem from a slightly different perspective. First they use a CNN trained on the datasets created by [79] and [80] to possibly detect characters in the input image. The candidate image areas are classified as plate or non-plate via a second CNN trained on the AOLP dataset [12] via cross-validation to discard false positives. Finally, a long short-term memory recurrent network trained on the same character set is used to label the characters as a textual sequence, avoiding the otherwise critical character segmentation step. Despite the use of three different networks, this approach shows improved results over the same dataset used in [12]

Gou et al. [81] propose a LPR algorithm based on Extremal Regions and Restricted Boltzmann Machines [82] (RBMs). First a coarse detection of license plates is performed using edge detection and image filtering. Characters regions are then extracted using Extremal Regions which in turn are used to refine the plate region. Finally the characters are recognized using hybrid discriminative RBMs trained on character samples extracted from real photos augmented by rotation and noise.

However, plate localization deals only with plates of specific size and aspect ratio and is not demonstrated to be robust to changes in perspectives.

Bulan et al. [10] propose to use a weak and sparse classifier followed by a strong CNN to sort out readable license plates. For character recognition, they avoid the segmentation step using a sweeping SVM classifier and a hidden Markov model to infer their locations. The character classifier is trained either with real samples labeled by an already working classifier or over synthetic data. However, the experiments show a performance loss when the network is trained over their synthetic data.

### Training samples for deep learning techniques

Since deep learning requires a large amount of training samples, this is another challenge to overcome. Menotti et al. [11] and H. Li et al. [9] circumvented this by training multiple models using k-fold cross validation on the test set itself while Bulan et al. [10] use classical edge detection to locate the plates and propose to use synthetic character samples or an already operational OCR algorithm for the character classification, without going into much detail about how synthetic samples would be generated.

J. Li et al. [78] and Gou et al. [81] however, take on the laboursome task of labelling real samples. Gou et al. collect 12,366 labeled Chinese characters and 10,408 labeled letters for the character classification while J. Li et al. manually annotated about 5,613 license plates and carefully cropped them for character classification.

Synthetic Samples can also be generated with Generative Adversarial Networks [83] (GANs), but like any other neural network, GANs are data driven and still require a large amount of labelled data to be trained. And the problem of LPR is not access to training data, but labelling the data. Furthermore, the quality of synthetic samples generated from GANs may be impressive considering they are spawned from random noise, but the resolution is low and the quality does not fool a human eye other than at a first glance. They may however be useful to augment an already existing dataset, by first training the GAN on it, then use the generative network to dilute the original dataset with additional synthetic samples.

### 1.2.6 Challenges in LPR

While LPR systems have matured over time, there is still room for improvement. Systems having a high detection rate typically do so by sacrificing precision when avoiding to reject positive classifications. In this case, even if all plates are automatically recognized, a lot of manual labor remains to discard the remaining false recognitions.

Since CNNs have shown groundbreaking performance in a number of image processing tasks [3, 84, 85], we base our system around those to achieve high accuracy and balanced in terms of precision and recall.

However, the use of CNNs also poses other challenges, as they require a large amount of training samples. This may be achievable by hand when training classifiers to detect a single plate type, but to train classifiers for multiple plate types this quickly becomes an overwhelming task. Therefore we will train our classifiers on automatically annotated and generated synthetic training samples.

Another challenge that arises with the use of complex CNNs is the processing time, since many LPR applications are constantly active, it is necessary to classify images at the same rate as they are captured (or at the very minimum, one classification within the time frame it takes for a car to pass the camera). Therefore, rather than using a very deep CNN architecture such as [86, 87], we will try to keep the networks shallow and with as few parameters as possible without losing performance.

Scale dependency is also a limitation of CNNs; although some CNN architectures exists that provide some scale robustness [88, 89], images can come in very different resolutions. This means the resolution of the actual plate will also vary greatly and in high resolution images, plates within the same image may even vary by several factors in scale e.g. if the image is taken alongside a road of passing cars. Therefore we will instead take a scale independent approach by employing a scale pyramid; in combination with generated training samples we can control the inherent scale robustness of the CNN itself to match with the pyramid scale step such that the increased complexity of employing it is optimized and limited well below that of the initial classification of the full resolution image.

# Chapter 2

# Background on Convolutional Neural Networks

CNNs are feed-forward deep neural networks with some layers in the form of convolutions patterns; typically the convolution layers act as a *feature extraction stage*, followed by an *inference stage* of fully connected layers [8]. Convolution kernels in the feature extraction stage act as filters, activating upon the detection of one specific feature in the input. The output of the feature extraction stage is processed by one or more fully-connected layers in the inference stage, the actual number of layers and learnable parameters in each layer depends on the specific application. Finally, the last layer of the network provides the desired output such as an object class label (classification problems) or the object position in the image (regression problems).

These networks are extremely powerful models for image recognition, where the relationship between neighbouring channels are more relevant than individual channel values since, unlike in classical classification tasks, features are not tied to specific channels (pixels) in images.

They were first introduced back in 1980 by K. Fukushima under a different name, "neocognitron" [90], which at the time was performing too slow to be practically useful. The power of CNNs was not quite realized until Y. LeCunn et al. re-introduced the concept under its current name in 1999 to recognize hand written digits [3].

As the computational power matured, CNNs have emerged as the cornerstone to efficiently solve a number of image classification and regression problems such as character recognition [3–5], object categorization and localization [6, 7].

## 2.1 Definition of fully connected layer

Before defining the convolutional layer of the CNN, let us begin with the mathematical model of a single neuron and fully connected layer, to introduce the notation. After all, these are necessary parts in a CNN as well.

The artificial neuron is inspired by the biological neuron with multiple inputs (representing dendrites) and a single output (axon), providing a non-linear response based on the input signal:

$$y = f(\sum_i I(i)W(i) + b), \tag{2.1}$$

where $y$ is the output value of the neuron, $f$ is a non-linear activation function (see section 2.4), $I(i)$ is an element of the input vector, $I$ (or vectorized image), $W(i)$ is an element in a vector, $W$, of learnable weights connected to every input (hence fully connected) and $b$ is the bias term. Fig. 2.1 (left) illustrates a neuron as described in (2.1).

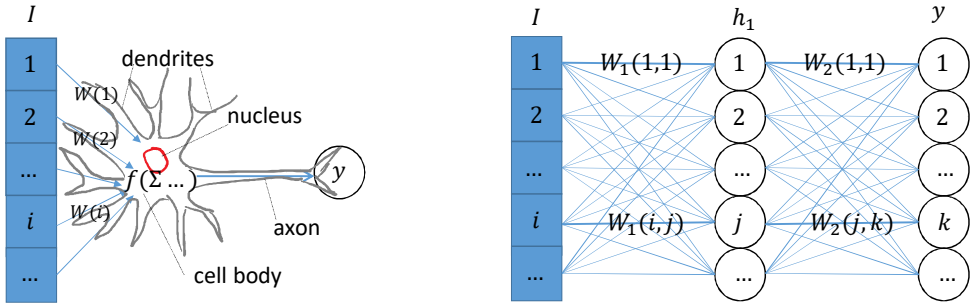



Fig. 2.1 Illustrations of (left) a biological neuron modeled by (2.1), input signals are received via the dendrites and an output signal is sent via the axon, and (right) the two-layer fully connected network in (2.3-2.4). Each input element is connected through weights to all neurons in the hidden layer, the output of the hidden layer acts as input to the output layer.

This can be expanded to a layer of fully connected neurons

$$y(j) = f(\sum_i I(i)W(i,j) + b(j)), \tag{2.2}$$

and it can be further extended to a multilayer network by adding another layer, such that the output of the first layer is the input of the second layer:

$$h_1(j) = f(\sum_i I(i)W_1(i,j) + b_1(j)) \tag{2.3}$$

$$y(k) = f(\sum_j h_1(j)W_2(j,k) + b_2(k)), \tag{2.4}$$

where $h_1$ is a *hidden layer*, this is also illustrated in Fig. 2.1 (right). In principle, any number of layers can be added in this manner, a network is said to be deep if it has more than two layers. The more layers, the more complex the decision boundary can become (and the higher the risk of overfitting if the sample quantity is not sufficient).

## 2.2 Definition of convolutional layer

While the fully connected layer uses individual learnable weights between each connection of an input and an output, a convolutional layer uses patterns of shared weights applied in a convolutional manner:

$$y(i,j) = f(\sum_m \sum_n I(i+m-1, j+n-1)W(m,n) + b), \tag{2.5}$$

where $y(i,j)$ is an output neuron, $I$ the input image and $W$ a matrix of convolution weights, this is also illustrated in Fig. 2.2. Note that this actually is not the discrete version of a convolution, but rather the discrete cross-correlation. Many machine learning libraries actually implement the cross-correlation instead of the convolution [91]. Although, in practice, the only difference between the two operations is that $W$ is flipped horizontally and vertically. Of the two operations cross-correlation is the more intuitive since the pattern is flipped the same way as the image, simplifying visual comparisons. So, convolution layers are measuring the correlation between the input image and a feature pattern, however, we will continue to call this convolution, as is conventional.

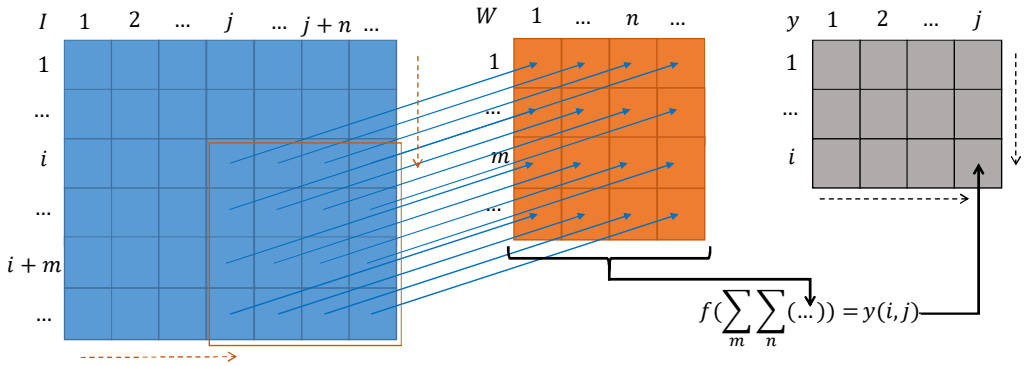



Fig. 2.2 Illustrations of a convolution layer. The weight matrix (orange) is shifted over the input image (blue) and element-wise multiplied, producing a matrix of outputs (gray).

Typically a convolutional layer consists of multiple convolution patterns, or *kernels*, each producing an output layer. The input layer is typically also multidimensional, such as a color image, or a hidden layer from a previous convolution layer:

$$y(i,j,l) = f(\sum_m \sum_n \sum_k I(i+m, j+n, k) W^l(m,n,k) + b^l). \tag{2.6}$$

where $k$ is the spectral dimension index of the input, $I$, and $l$ denotes individual convolution patterns, each producing a "spectral" dimension in the output, $y$. Note that the convolution does not act in the spectral dimension. Instead the weights of the convolution patterns are extended to match the spectral dimension size, thus measuring the local correlation in all input layers individually. Each convolution pattern, $W^l$, results in a separate "spectral" dimension of the output, henceforth we will call the outputs of convolution layers *feature maps* with *feature layers* along a *feature dimension* rather than images with *spectral bands* in a *spectral dimension* (such as the Red, Green and Blue layers in an RGB image).

In this thesis, a convolutional layer as defined in (2.6) will be described as a layer of $k$ $M \times N$ convolutions, where $M$ and $N$ represent the size of the first and second dimension of $W$, while the third dimension is typically omitted since it is already given by the previous layer.

This means each convolution pattern yields a feature map with the spatial relations of the original image conserved, but with the spectral dimensions replaced by

correlation measurements related to that spatial region. And multiple convolution patterns replaces the spatial dimensions with feature dimensions. Compared to a fully connected layer, the number of learnable weights are greatly reduced, for example, a $3 \times 3$ pattern operating on a large RGB image has only 28 learnable parameters (27 + 1 for the bias).

As with fully connected layers, convolutional layers can naturally be stacked to produce deeper networks:

$$h_1(i,j,l) = f(\sum_m \sum_n \sum_k I(i+m, j+n, k) W_1^l(m,n,k) + b_1^l) \tag{2.7}$$

$$y(i,j,l) = f(\sum_m \sum_n \sum_k h_1(i+m, j+n, k) W_2^l(m,n,k) + b_2^l). \tag{2.8}$$

This enables the feature extraction stage to form hierarchical features, where the lower layers form features of relevant texture details, and each layer gradually forms more complex features based on the features of the previous layer, with a larger and larger *field of view*, i.e. the size of the area in the input image which a kernel has connections to, through previous layers. This allows the feature extraction layer to recognize complex features with a relatively low number of learnable weights, reducing the risk of overfitting.

## 2.3 Max-pooling layer

Typically, a max-pooling [92] layer is placed between convolutional layers, this further increases the field of view of the next kernels without increasing its size (and thereby computational complexity). It also decreases the importance of spatial relationships between previous features, which can also help achieving a more general model that accepts smaller shape variations of objects without necessarily being trained on all possible variants. It is defined as follows:

$$y(i,j,k) = \max_{m=(i-1)s_x+1}^{(i-1)s_x+s_m} \max_{n=(j-1)s_y+1}^{(j-1)s_y+s_n} h(m,n,k), \tag{2.9}$$

where $s_x$ and $s_y$ are downscaling factors, typically chosen equal to the pooling size ($s_m \times s_n$) dimensions such that each feature value is considered exactly once.

As suggested before, mixing convolutional layers with max-pooling layers allows the network to learn small texture patterns as features while becoming less and less spatially specific when forming more and more complex features in deeper layers. If no spatial information is desired in the final classification, a final max-pooling filter of the same dimensions as the final convolution layer can even be employed to achieve full translation invariance.

## 2.4 Activation functions

We have so far left out the activation function, other than specifying it must be non-linear, which is necessary since stacked linear layers can only produce a linear decision boundary such as a hyperplane. The following lists some common choices of activation functions, the first three are also illustrated in Fig. 2.3:

- **Sigmoid**

$$y(x) = \frac{1}{1+e^{-x}} \tag{2.10}$$

  Historically the typical activation function to use was the sigmoid function, which squeezes the output into a range [0,1] and has well defined and simple derivatives, which makes learning with backpropagation (see section 2.6) easy. A shortcoming of the sigmoid is that it only asymptotically approaches 0 and 1 which are typically the values used as classification targets.

- **Hyperbolic Tangent**

$$y(x) = A\tanh(Sx) \tag{2.11}$$

  The Hyperbolic Tangent (TanH) is similar in shape to the sigmoid but symmetric [3]. The constants $A$ and $S$ also allows for customization of the range to $[-A,A]$ and steepness of the curve, this is particularly useful if TanH is used as the final activation unit before the outputs, to customize the range according to the output target values and avoid the limitation of the sigmoid by allowing it to overshoot the target values.

- **Rectified Linear Unit**

$$y(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

The Rectified Linear Unit [93] (ReLU) is perhaps the most similar activation function to that of an organic neuron, since it only propagates positive signals. This helps the classifier to focus on positive features and ignoring background features rather than learning them as negative features. This has become the typical choice of activation function for neural networks applied to classification problems, while TanH is typically used for regression problems.

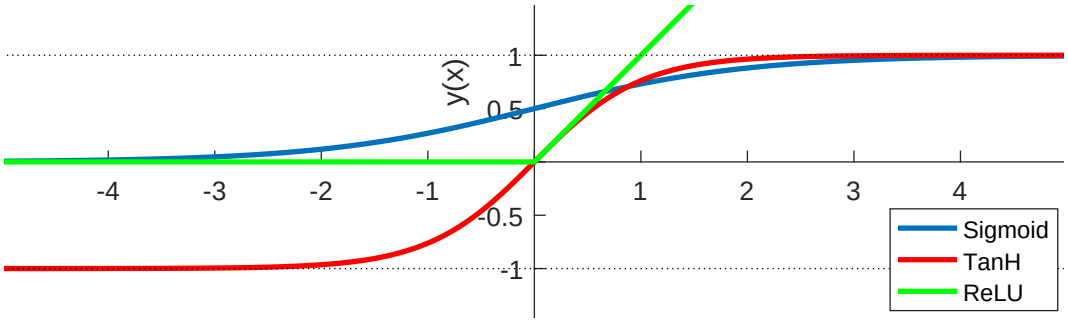



Fig. 2.3 Plot of the non-linear activation functions Sigmoid (blue), TanH (red) and ReLU (green).

- **SoftMax**

$$y(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.13}$$

The SoftMax function [94] is useful for a the final classification layer because it normalizes the output units, $y(x)_i$, such that $\sum_i y(x)_i = 1$, like a probability distribution. Note however, that it does not necessarily mean the output class is correct with that probability, or that it is even a good approximation thereof. Therefore we will call such outputs *confidence ratings* rather than probabilities.

## 2.5 CNN architecture

As mentioned before, a typical CNN consists of a feature extraction stage followed by a inference stage. The feature stage contains a series of convolution layers with

small kernels ($3 \times 3$) and ReLU activation functions, and $2 \times 2$ max-pooling layers in between them to decrease the resolution of the hidden layers. This scheme allows for an increased amount of convolution kernels per layer, with an increased field of view (with respect to the input layer), without increasing the computational cost of the layer compared to previous layers. As illustrated by the following example:

**Ex 1.** Consider a feature map of size $I_h \times I_v$ with $F$ feature layers, and a convolution layer with $2F$ kernels of size $C_h \times C_h$. Assuming the convolution is applied with padding for simplicity (if $I_h \times I_v >> C_h \times C_h$ this is still a good approximation), then the convolution is shifted $I_h \times I_v$ times and each of the $2F$ convolution kernels has $C_h \times C_h \times F$ weights. That is, the number of connections between the feature map and the convolution kernels (and thereby multiplications needed to compute the next feature map) is $(I_h I_v)2F(C_h C_h F) = 2F^2 I_h I_v C_h C_h$.

**Ex 2.** Now consider the next layer, following this scheme, after the max-pooling will be of size $\frac{I_h}{2} \times \frac{I_v}{2}$ with $2F$ feature layers, and the next convolution layer with $4F$ kernels of size $C_h \times C_h$. Yielding $(\frac{I_h}{2}\frac{I_v}{2})4F(C_h C_h 2F) = 2F^2 I_h I_v C_h C_h$ connections again.

The inference stage typically consists of a number of fully connected layers with ReLU activation functions between them and a SoftMax activation function after the final layer; the more layers the more complex the decision boundary is allowed to be. The number of neurons also affects the complexity of the boundary, but can also be used to gradually scale down the number of connections towards the output. Fig. 2.4 shows an example of a simple CNN architecture.

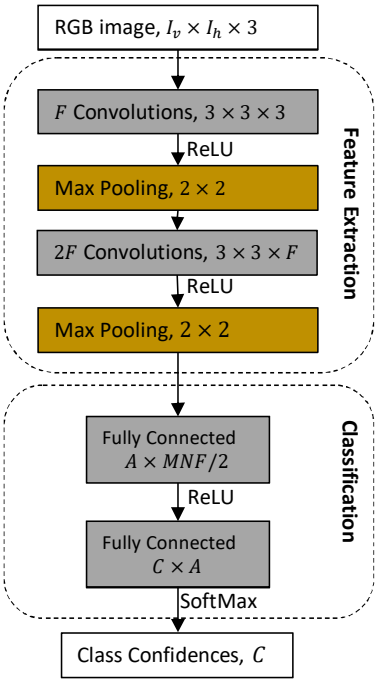



Fig. 2.4 Example of a simple CNN architecture.

## 2.6 Learning

To be able to classify anything meaningful, the network first needs to learn a set of good weights. This is done with the help of a *training set* of labeled examples. A smaller *validation set* can also be used to select a good model, which the network does not directly learn weights from, but it can be used to apply a stop condition for the learning process and to optimize hyper parameters and the network architecture. Indirectly the composition of the validation set may therefore still affect the network model, therefore a separate *test set* should be used for the final evaluation of the network model to more accurately measure how it will perform on new, uncorrelated data.

To learn good weights, the training samples are first classified, initially with random weights, and the classification error is used to update the weights using a *backpropagation* algorithm [95] which calculates the error derivatives of each weight with respect to the output target value, starting from the output layer, using chain

rule to propagate these error derivatives backwards in the network all the way to the input. These derivatives tell us how the output will change if we change the value of that weight, while keeping everything else fixed, and changing a weight in the opposite direction of the error derivative is guaranteed to decrease the error when the same sample is classified again.

In practice however, this is too slow, therefore all weights are changed by a small step in the direction towards the error derivative simultaneously. To average out the learning path, multiple images can be classified in parallel before all errors are backpropagated together; this is called a *mini-batch*. If the mini-batch is too small, the learning process becomes unnecessarily costly due to the number of backpropagations, and the learning may converge badly due to each sample pushing the weights in a different direction to improve the classification of that particular sample. On the other hand, if the mini-batch is too large, the learning is slow because many samples are classified on old weights that could have been improved by an earlier backpropagation. Classifying and learning from the full set of training samples is called a training *epoch*, typically multiple epochs are needed before the weights of all layers of the network have converged to a good solution with a low error.

This can be formulated as a minimization problem of a cost function defining an error metric, which is solved by Stochastic Gradient Decent (SGD) [95]. The classic cost function is the Mean Square Error (MSE), defined as:

$$E(W) = \frac{1}{C}\sum_{i=1}^{C}(y_i^o - t_i)^2, \tag{2.14}$$

where $E(W)$ is the error as a function of the set of weights, $W$, to be minimized (it is a function of $W$ through $y_i^o$), $C$ is the number of output units or classes, $y_i^o$ is the output of neuron $i$ and $t_i$ is the corresponding ground truth value. This cost function is typically used for regression problems, while the cross-entropy cost function is usually found to work better on classification problems, defined as:

$$E(W) = -\sum_{i=1}^{C} t_i \log y_i^o. \tag{2.15}$$

The SGD update rule is defined as follows:

$$W := W - \eta \nabla_W E(W), \tag{2.16}$$

that is, each weight in $W$ is iteratively updated in the direction of the negative gradient $-\nabla_W E(W)$ scaled by a *learning rate* factor $\eta$. Typically the learning rate is decreased during training to allow for larger update steps in the beginning to accelerate learning and smaller updates towards the end as a too large step might jump past the minimum. Alternatively an adaptive gradient algorithm can be used that adapts the learning rate, such as AdaGrad [96].

## 2.7 Fully convolutional neural networks

A limitation of the architecture described above is that it is limited to training and classification of images of a specific size, this is because the fully connected layers have fixed connections, which also requires the convolutional layers to be applied to a specific input size in order to match the input dimension of the fully connected layer. However, by replacing the fully connected layers with convolutional layers, a network can be applied to images of varying size. The output then becomes a matrix of classifications, each of a shifted window of the input image. This is essentially the same as a sliding window classifier, but this architecture allows for a much more efficient implementation compared to a normal sliding window classifier because it avoids re-processing of areas where multiple sliding windows are overlapping.

A fully convolutional network can be constructed to be equivalent to a CNN with fully connected layers by replacing the fully connected layers by convolutions (without padding) that are large enough to connect to all inputs without any space to shift. That is, a network with a separate feature extraction stage followed by a classification stage, such as in Fig. 2.4, needs to replace the first fully connected layer by a convolution of the size of the final feature map produced by the feature extraction stage. If the fully connected layer connects to $A$ neurons, the convolutional equivalent requires $A$ kernels. The convolution has no space to shift, so each kernel will only produce 1 output neuron, i.e. the output feature map will be $1 \times 1 \times A$. This is illustrated in Fig. 2.5 left and center.

Each fully connected layer after the first are then replaced by $1 \times 1$ convolutions, with one kernel per output neuron of the corresponding fully connected layer. Apply-

ing this network to a larger input image will allow the convolutions space to shift and produce multiple outputs, corresponding to classifications of a sliding window classifier, as illustrated in Fig. 2.5 right. Fig. 2.6 shows an example of the network in Fig. 2.4 converted to a fully convolutional network.

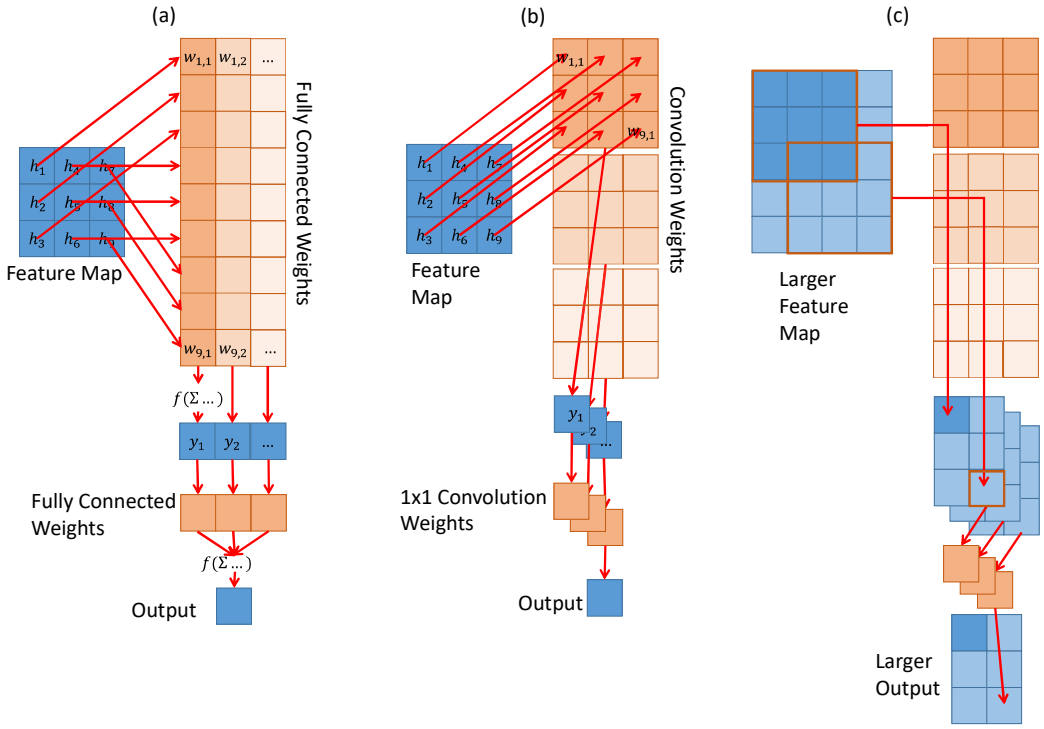



Fig. 2.5 Comparison of (a) fully connected vs. (b) equivalent fully convolutional architecture and (c) an illustration of how the fully convolutional version produces a matrix of outputs as a result of a larger input image.

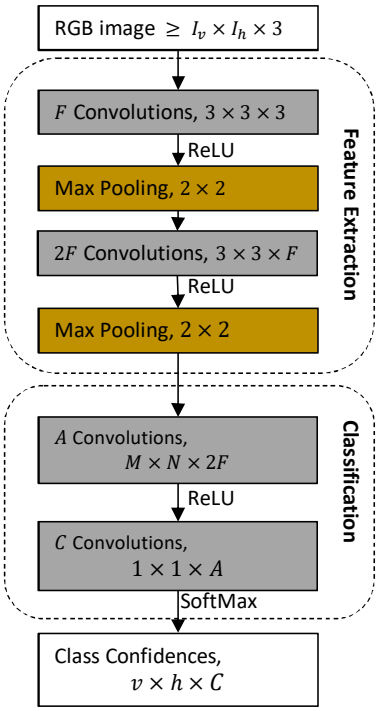



Fig. 2.6 Example of the CNN example in Fig. 2.4 converted to a Fully Convolutional CNN.

# Chapter 3

# Synthetic Sample Generation

We decided to investigate how feasible it would be to exclusively use synthetic samples to train the neural networks for LPR, because of how time consuming it would be to manually label enough training samples, and how badly that scales when expanding to recognition of other plate types. Synthetic generation scales much better as it allows us to generate plates of different types, e.g. from other countries with very small changes to the code. Thanks to this, we can use a Taiwanese dataset from [12] as the final test set to evaluate our algorithm on, see section 6.5.

## 3.1 A word on generative adversarial networks

GANs [83] are an obvious potential candidate for synthetic sample generation that deserves to be mentioned. GANs use two networks in an adversarial fashion during training, one generative network and one discriminative network. The task of the generative network is to produce samples that are able to fool the discriminative network that they are real samples, with only noise as input. The discriminative network on the other hand receives a real sample or one generated by the generative network and has to predict whether it is a real or a generated sample. Backpropagation is used to train the networks as a single system. If the discriminator is able to discriminate between real and fake, the generative network will learn to produce samples more similar to the real samples and if the generator manages to fool the discriminator, it will learn to look at finer details to tell the difference.

However, there are several reasons why we chose not to implement GANs for our sample generation:

- Labelled training samples are still needed to train the GANs.
- There was never a point where we considered it possible to produce more accurate samples with GANs compared to the samples generated by our own algorithm at that current time. GANs are not (yet) good enough to generate samples of the same distribution as the training samples [97].
- It is unclear how to implement location labelling and if it would be accurate.

## 3.2 Synthetic sample generation algorithm

The algorithm described in this section is implemented in MATLAB and produces a set of 50% positive and 50% negative images, intended to be used for training of both plate and character classifiers, with some minor post processing (cropping and scaling) left intentionally for the training stage (so that the training set may be more useful to others, and so that we would not need to generate a new training set in case of network design changes). The positive image contains a synthetic license plate over a background image. For each positive image, there is a corresponding negative image using the same background image without a license plate (but sometimes with a random text superimposed instead). This to ensure that the classifier does not learn background features nor to rely solely on text to identify plates. Then the process is simply repeated to generate as many samples as needed.

The process of generating synthetic images is divided in eight modules, of which only the first needs to be changed to generate samples for another plate type or country. A rectangular plate template is first generated (in section 3.2.1) and a background image loaded (in section 3.2.2). Occlusions are added to the plate (in section 3.2.3), followed by perspective distortion (in section 3.2.4) and addition of noise (in section 3.2.5). Then the plate is added to the background image (in section 3.2.6) and global noise is added to the entire image (in section 3.2.7). Finally, a negative sample with other text or only a background image is also generated (in section 3.2.8). The process to generate a pair (a positive and negative sample) is illustrated in Fig. 3.1 and described in detail in the following sections.

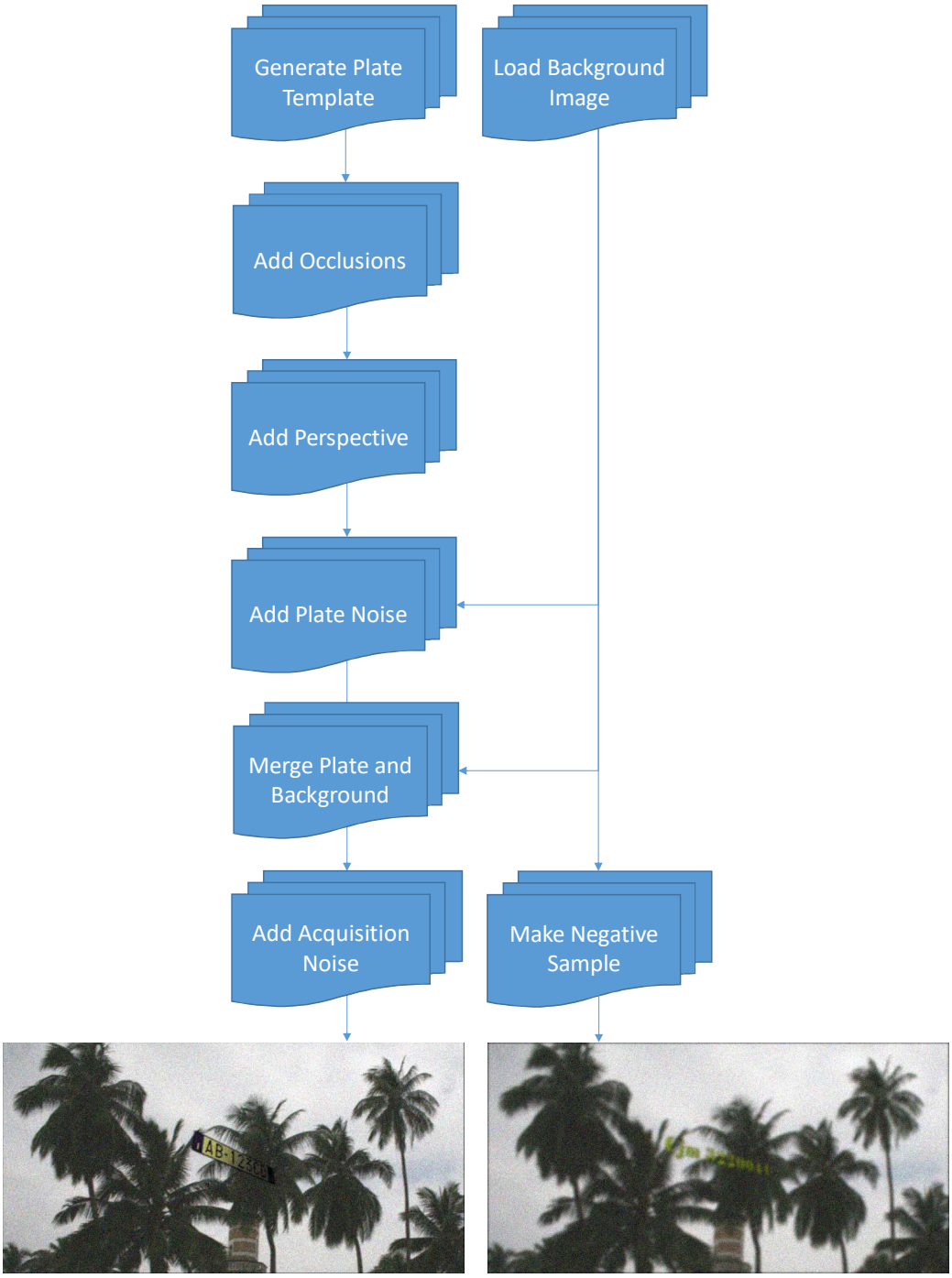



Fig. 3.1 Flow Chart of the sample generation algorithm. Each block is detailed in their corresponding sections.

The algorithm evolved iteratively in parallel with the network training and an initially very crude version of the classification algorithm. Occasionally the algorithm was tested on the Italian test set (described in section 6.2.1) to see if we were on the right track, and the errors were evaluated to see where improvements could be made. Since the classification algorithm was improved in parallel, the results of these experiment are of little use to compare the impact of different features in the sample generation algorithm. To demonstrate the relevance of certain non-mandatory features, an experiment with such features (active versus inactive) was performed on the final version of the classification algorithm and is presented in section 6.4.2.

Some of these error corrections may have indirectly biased the algorithm towards the distribution of samples in the Italian test set. In order to check that our LPR system can generalize properly, the final version of the sample generation algorithm was also tested on a test set of Taiwanese license plates, see section 6.5.

Note also that the the samples are not intended to all be completely realistic, but are rather designed to ensure the variance of real samples is at least covered (often with some margin of unrealistic samples). The decision boundary does not need to be tight around real plates as long as it does not include other real objects. Neither are the plates intended to represent the true distribution of real plates, therefore plate details and noise features that are added are always uniformly distributed to give the classifier an equal opportunity to learn all variations rather than bias it towards the more likely features. In other words, rather than allowing the classifier to learn and take advantage of the a priori probability distribution when performing predictions, we aim to make the classifier learn to recognize all required features, even if that may require more samples. This design choice is more stable if the underlying distribution would change in the future. An example of this is that the first characters in the alphabet are severely overrepresented as the first character of Italian license plate. If we adopted this bias in the distribution our classifier may perform better on the current test set, but it would become weaker in the future as more plates are produced and the distribution changes.

### 3.2.1 Generate plate template

This module generates image patches of the different details of the license plate and stitches them together into a plate with random properties within the desired plate

format. Along with the plate image, it also outputs the classification labels needed for training, i.e. license registration number and initial coordinates of the plate and character corners (later modules will update the coordinates accordingly as the plate is transformed and placed into a background image).

This module is the only one that is plate type dependent, in the following description will focus on EU-format Italian back plates for regular cars as an example, but it can be interchanged with relatively small changes, as shown in section 6.5.1. Although we have not tried it, it should also work to use objects other than license plates, e.g. street signs.

### Italian rear license plate format

The basic appearance of an Italian back license plate is shown in Fig 3.2. The license format is: **LL DDDLL**, where **L** indicates a letter from A to Z with I, O, Q, U excluded (due to being too similar to other letters or numbers) and **D** indicates a digit from 0 to 9. Fig 3.2 shows all mandatory details, some plates also include an EU symbol, a two digit number indicating the year of registration is sometimes indicated inside the circle and/or two letters in the lower right corner indicating the province of registration.


Fig. 3.2 Italian rear plate with measurements specified.

The license plate generation algorithm is illustrated in Fig. 3.3 and described below.

### Load graphics

First, gray-scale versions of the required graphics are loaded, the dimensions of the patches are chosen such that the backgrounds fill up the entire plate once fitted

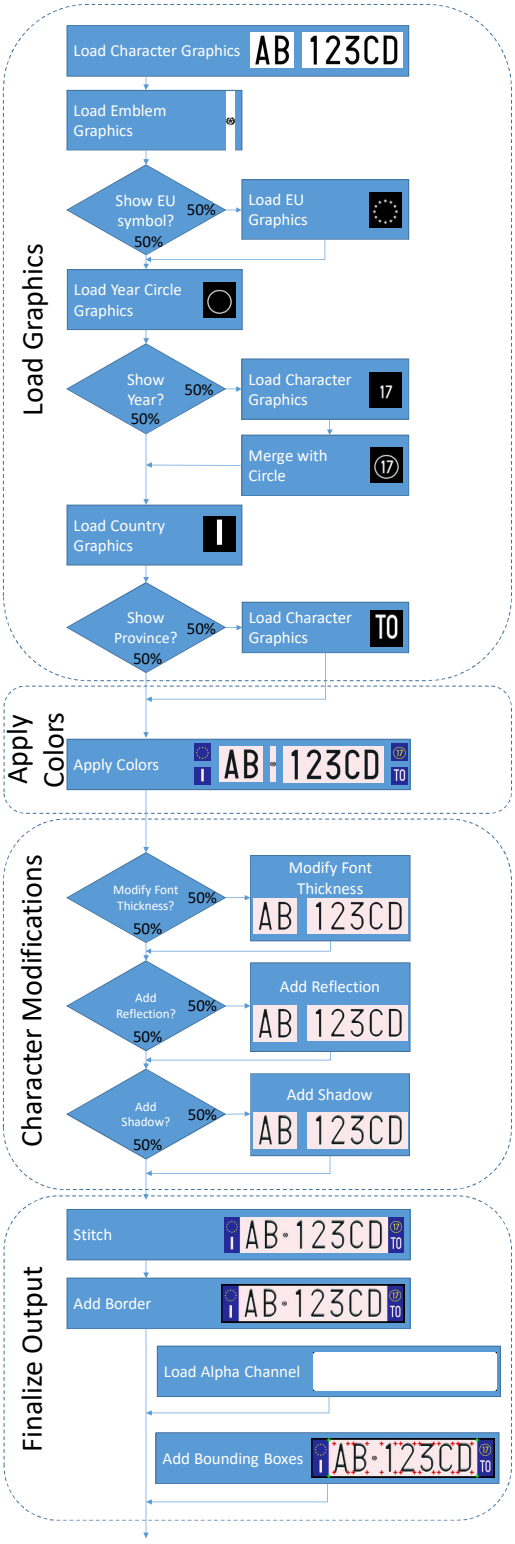



Fig. 3.3 Flow chart of the plate generation algorithm.

Table 3.1 Graphical elements of the Italian plate template.

| Graphical element | Active | Inactive | Size [pixels] |
|---|---|---|---|
| Characters |  | | $122 \times 100$, $284 \times 100$ |
| Emblem |  | | $14 \times 100$ |
| EU symbol |  |  | $45 \times 50$ |
| Registration year |  |  | $45 \times 50$ |
| Country letter |  | | $45 \times 50$ |
| Province letters |  |  | $45 \times 50$ |

together. This includes the below listed image patches. The graphical elements, including alternative branches, are also listed in Table 3.1.

- Two image patches with the groups of 2 and 5 characters (according to the provided input license string) loaded, of $122 \times 100$ and $284 \times 100$ pixels respectively.
- A patch of $14 \times 100$ pixels with the emblem located between the two groups of characters.
- With 50% probability, a patch of $45 \times 50$ pixels of the EU symbol otherwise an empty patch of the same dimensions.
- A patch of $45 \times 50$ pixels with a circle, and with 50% probability, 2 random digits are loaded and inserted inside the circle.
- A patch of $45 \times 50$ pixels with an I for Italy.
- With 50% probability, two random letters are inserted in a patch of $45 \times 50$ pixels, otherwise an empty patch of the same dimensions.

Table 3.2 Colors of graphical elements.

| Graphical element | Foreground color | Background color | Example |
|---|---|---|---|
| Character | black | white |  |
| Emblem | black | white |  |
| EU symbol | yellow | blue |  |
| Registration year | yellow | blue |  |
| Country letter | white | blue |  |
| Province letters | white | blue |  |

### Apply colors

To account for color variations during production or ageing, the four colors used, black, white, blue and yellow, are given some random variation before being applied as background and foreground colors to the loaded graphics, within ranges such that each is still perceived as the intended color. At this stage the colors are homogeneous within a plate, variations e.g. due to illumination and noise are applied at a later stage. The colors applied to each graphic element are shown in Table 3.2, each object use only two colors, a background and a foreground color, initially labelled by black and white (shown in Table 3.1). However, gray-scale is used when the boundaries are not exactly along the pixel boundaries. In such cases the applied colors are mixed according to the percentage of black and white encoded in the gray.

The ranges within which the colors are randomized were selected by comparing a set of real plate images under different light conditions and cutting color volumes in the RGB color space until the observed colors were contained within the volumes (with some margin). The boundaries are defined below and illustrated as volumes in the RGB color space in Fig. 3.4.

$$Black = (K_r, K_g, K_b)$$

where $K_r, K_g, K_b \in \mathbb{N}_0[0,60]$

$$White = (L+W_r, L+W_g, L+W_b)$$

where $L \in \mathbb{N}_0[155,225]$ and $W_r, W_g, W_b \in \mathbb{N}_0[0,30]$.

$$Blue = (B_r, B_g, B_b)$$

where $B_r \in \mathbb{N}_0[0,100]$, $B_g \in \mathbb{N}_0[0,155]$ and $B_b \in \mathbb{N}_0[max(B_r+100, B_g, 100), 255]$.

$$Yellow = (Y_r, Y_g, Y_b)$$

where $Y_r, Y_g \in \mathbb{N}_0[L,255]$ and $Y_b \in \mathbb{N}_0[0,60]$ ($L$ is the same value as for $White$).

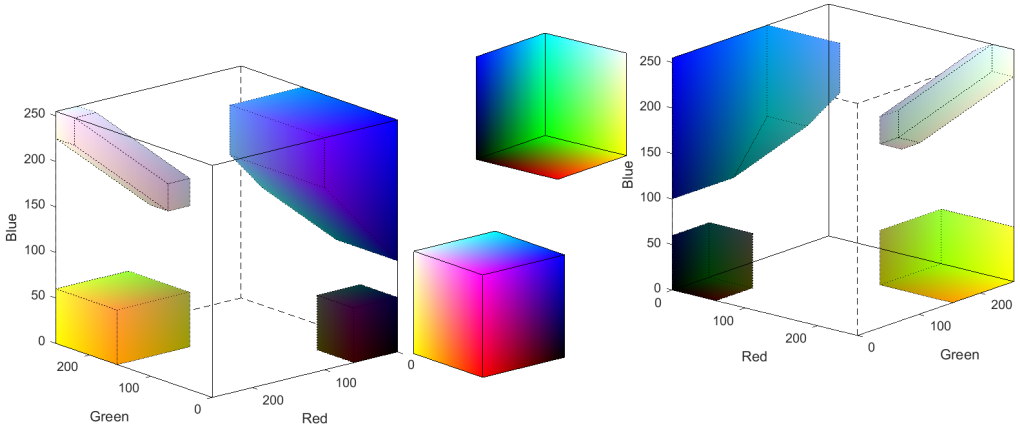



Fig. 3.4 Black, white, blue and yellow color volumes from which the colors are randomly selected. Left: View from above, centered over magenta. Center bottom: Reference of the full RGB color volume from the same view. Center top: Reference of the reverse view. Right: Reversed view, from below, centered over green.

### Character modifications

To reflect the appearance of real plates some modifications are made to the font of the characters. The text may appear thicker or thinner on images, e.g. because blurry edges may appear as either white or black due to pixel saturation. Therefore, with 50% probability, the font boldness is varied to be up to 35% thicker or thinner then originally.

The characters are also often embossed, with can produce distinct reflections and shadows on the background. With 50% probability reflection is therefore added in the form of a "shadow" of saturated white (i.e. [255 255 255]), extending up to 50% of the original character thickness at a random angle.

Similarly, a shadow of gray can also be added at the same manner; however, when both features are active, the angle of the shadow is always shifted 180° with respect to the reflection angle. A few examples with different modifications activated are shown in Fig. 3.5.

Fig. 3.5 Character modifications. Top-left: Input (and output in case no modifications activated.) Top-center: Thickened font. Top-right: Shadow actived (on thickened font). Bottom-left: thinned font. Bottom-canter: Reflection actived (on thinned font). Bottom-right: Shadow and reflection actived (on thinned font).

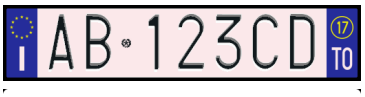
### Finalize output

Finally the graphical elements are stitched together (the sizes of the patches are already selected to cover the entire background of the plate) and a black border is added. The rounded corners of the plate are created by adding an alpha channel to the image, where 0 (displayed as black in figures) denote full transparency and 255 (displayed as white) denote 100% opacity.

Coordinates of the corners for bounding boxes of the 7 characters and the plate itself are also stored in separate image layers, this to simplify keeping track of their final coordinates in the samples after the plate is transformed and placed into the background image.

To sum up, the template generation module produce the output listed with examples in Table 3.3.

Table 3.3 Template generation output.

| Graphical element | Example |
|---|---|
| License plate image |  |
| Alpha channel | |
| Plate and character coordinates |  |

### 3.2.2 Load background image

The background image is the only part of the sample generation that is not synthetic, since there is no labelling related to the background image there is no advantage in trying to make it synthetic. It does however, not include a car (other than occasionally) but only completely random natural images.

This module loads random background images from the CDVS distractor set [98] (see Fig. 3.6 for a few examples), this set contains 1 million images of varied resolution and content, collected from FLICKR. The images are cropped down to a resolution of $768 \times 384$, if the original resolution of either dimension is insufficient the image is first mirrored and duplicated along that dimension until it is large enough.

The resolution of $768 \times 384$ pixels is chosen because the final plate samples require a patch with size a third of those dimensions, i.e. $256 \times 128$ pixels. With the plate placed within the central $256 \times 128$ pixels, it still allows to fully shift the crop region in any direction so any amount of the plate is visible in the sample (from 100% to 0%, not necessarily to be used as positive plate training samples, as will be described later), $768 \times 384$ is the minimum size that allows maximum flexibility. How the final samples are cropped before training of plates and characters is detailed in sections 4.1.4 and 4.2.4.

### 3.2.3 Add occlusions

Due to protrusions of the car, the plate can sometimes be occluded or shaded, either from above or a side. To simulate such conditions, an occlusion module was added.

Fig. 3.6 Some examples of images from the CDVS distractor set, as can be seen, dimensions may vary.

This module may add an opaque occlusion, a shadow and/or a transparency to the plate image (each with a 50% probability). This process is illustrated in Fig. 3.7 and described in more detail below, the characteristics are also summarized in Table 3.4 at the end of this section.

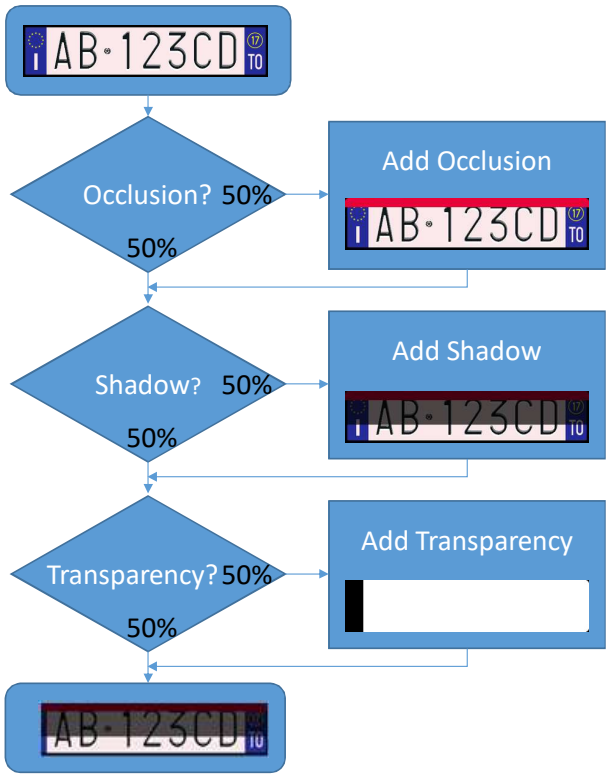



Fig. 3.7 Flow chart describing the process of adding occlusions on the plate.

### Normal occlusion

To simulate an occlusion, a horizontal or vertical occlusion of a "car color" may be added over a small part of plate image, with 50% probability. The occlusion is limited to 1 - 15% or the plate height (for horizontal) or width (for vertical) because the license number is required to be legible (by eye) on a positive sample. The color or the occlusion is limited to clearer colors because when randomizing over the full color spectrum, most colors tend to become gray-brownish. Therefore the color are randomly selected from $R, G, B \in \mathbb{N}_0[0,55] \cup [200,255]$, these colors are illustrated in Fig. 3.8.

The occlusion may appear horizontally from the top with 50% probability, or from either the left or right, with 25% probability each, Fig. 3.9 show a few examples of possible occlusions.

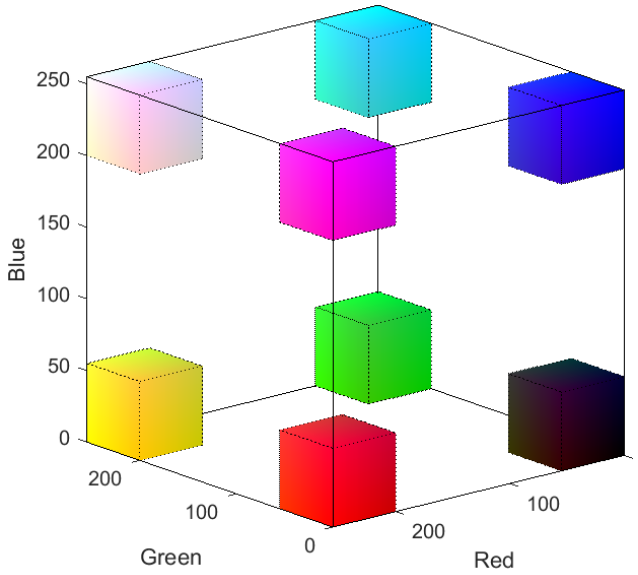



Fig. 3.8 Possible colors of the occlusions added over the plates.

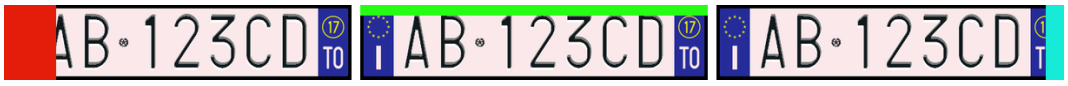


Fig. 3.9 Three examples of possible occlusions added to the plates.

### Shadow

From better angles than in the occlusion case, the sun (or other light) may still appear from a narrow angle, causing a protruding car part to cast a shadow on the plate. Therefore, a horizontal or vertical shadow is added over the plate image with 50% probability (independent from the chance to add an occlusion). Because the text can be read through the shadow, it may cover a larger part of the plate compared to the occlusion, 10 - 70% of the plate is darkened, from either the top or either side, the direction is determined with the same probabilities as in the occlusion case, but independently, i.e. 50% from the top, 25% for the sides. The pixels are darkened by 10-70%, i.e. each color intensity is multiplied by a factor of 0.1 - 0.7). Fig. 3.10 show a few examples of possible shadows.

Fig. 3.10 Three examples of added shadow, of varying degree, from lighest to darkest.

Table 3.4 Occlusion parameters

| Type | Width [%] | | Chance to appear on edge | | | | Color |
|---|---|---|---|---|---|---|---|
| | Min | Max | Top | Bottom | Left | Right | Modification |
| Occlusion | 1 | 15 | 50% | 0% | 25% | 25% | Random |
| Shadow | 10 | 70 | 50% | 0% | 25% | 25% | 10-70% darker |
| Transparency | 1 | 15 | 25% | 25% | 25% | 25% | Background |

**Transparency**

To simulate other kinds of less defined partial occlusions, the possibility of transparency is also added. Not because it is expected that something behind the plate may be visible through it, but rather because something else may appear in front of the plate which is not of a homogeneous color (as is added by the normal occlusion step above).

A transparency is added with 50% probability, horizontally or vertically, independently from previous occlusions. But unlike the other occlusions, it may also be added from below, with equal probability to appear on any side of the plate. As for the occlusions, it is limited to 1-15% of the plate height or width, to ensure that the license number is still legible. The transparency is added by modifying the alpha channel or the image, Fig. 3.11 shows a few examples along with the plate images with the alpha channel applied.

Fig. 3.11 Three examples of added transparencies, below each plate image is the corresponding alpha channel.

To sum up, Table 3.4 shows a summary of the parameters for the three different occlusion types added in this module.

### 3.2.4 Add perspective

Now that the plate appearance differences have been represented, this module represents plates at a varied perspective, just like a real plate can be photographed from different angles and distances. This process is illustrated in Fig. 3.12 and described below.

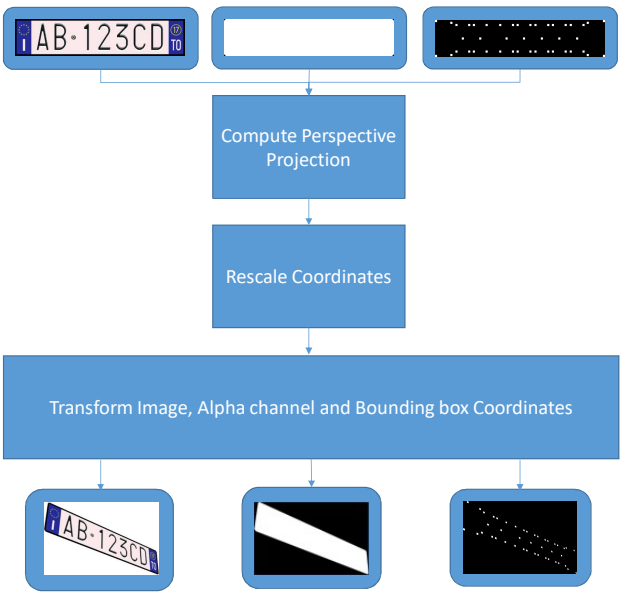



Fig. 3.12 Flow chart of the process of adding perspective to the plate.

#### Compute Perspective Projection

The perspective projection is computed as explained by [99] and summarized in appendix A, using (A.14) repeated below.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \lambda' KR(\phi, \theta, \psi) \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3.1}$$

Where in our case, $(x', y')$ is the projection of a plate corner point $(X, Y, Z)$ while $\phi$, $\theta$ and $\psi$ describe how the plate is rotated with respect to the camera, see 3.13.

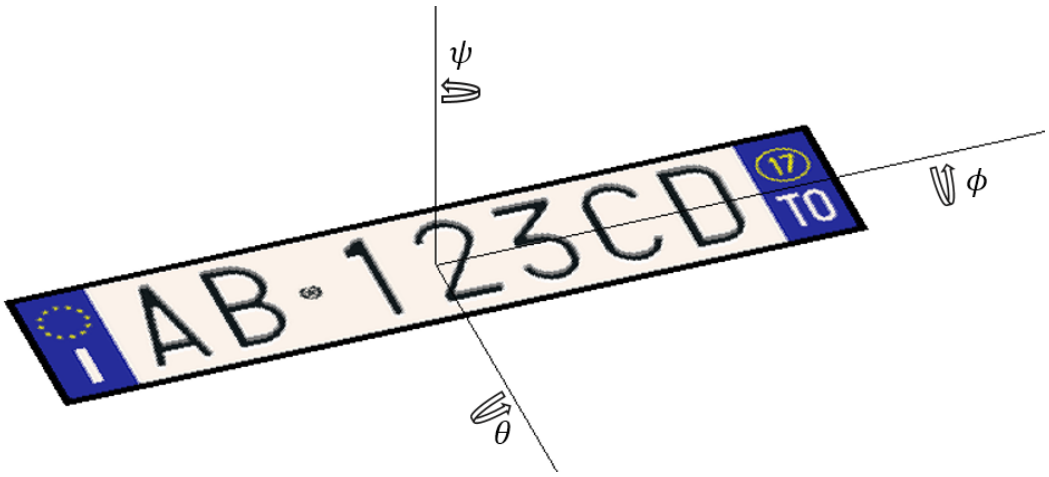


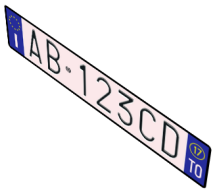

Fig. 3.13 Plate rotation axes.

For our purpose, we restrict these angles such that, 1) the license is still legible and 2) the angle should be possible based on correctly mounted license plate on a car, and a correctly mounted/held camera. These restrictions are summarized and illustrated in Table 3.5 with examples and explained in the following.

Because of 1) the horizontal rotation, $\phi$, is limited to $\pm 70^\circ$ and the vertical rotation, $\theta$, to $\pm 80^\circ$ and the total angle $|\phi| + |\theta| < 90$ (these limits were decided by applying high angle rotations and observing by eye if the plate was clearly recognizable or not, because the plate extends significantly more horizontally, such rotations were easier to recognize. Furthermore, such extreme camera angles are more commonly observed in real photos while extreme vertical angles are very rare). And because of 2) the axial rotation, $\psi$, is limited to $\pm 5^\circ$; essentially, this means we require the plate and the camera to be horizontally mounted/held. This is of course limiting the generalization of the algorithm, but if needed it can easily be increased, e.g. for motor bike plates, since the bike may be tilted when turning or parked. However, because a combination of horizontal and vertical rotation gives a similar appearance, we observed that plates with axial rotation are still recognized.

We want to rotate the plate rather than the camera, since camera rotation leads to unnecessary translation of the plate in the image plane. Therefore we include the camera distance to the position of the plate *after* applying the rotation, yielding:

Table 3.5 Restrictions of projection angles.

| Rotation | Min angle | Max Angle | Example (Max Angle) |
|---|---|---|---|
| $\phi$ | $-70°$ | $70°$ |  |
| $\theta$ | $-80°$ | $80°$ |  |
| $\psi$ | $-5°$ | $5°$ |  |
| $\lvert\phi\rvert + \lvert\theta\rvert$ | | $90°$ |  |

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \lambda' K \left( R(\phi,\theta,\psi) \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ Z_p \end{pmatrix} \right) \tag{3.2}$$

where $Z_p$ is the distance from camera to the plate center (the coordinate system of $(X,Y,Z)$ has the plate center as origin). The translation is limited to the Z axis in order to keep the plate centered in the projection, because shifting the plate sideways is essentially the same as changing the viewing angle - which might push it outside the desired bounds. Before discussing the range of $Z_p$ the internal camera parameters in $K$ should be defined.

$$K = \begin{bmatrix} \sigma_x f & \gamma & x_0 \\ 0 & \sigma_y f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

For our purpose, we can keep things simple and keep the origin of our camera sensor in the center, i.e. $x_0 = y_0 = 0$, assume the horizontal and vertical axes are orthogonal, i.e. the skew parameter $\gamma = 0$, and scaled equally i.e. $\sigma_x = \sigma_y = \sigma$. This means $K$ is diagonal and essentially serves as a scale factor of $Z$. This means by varying $Z$ we also cover variations of $\sigma f$, only at a different scale, which is anyway something we need to cover later in terms of software scale variations. That is, the

factor $\sigma f$ only has the function to scale $Z$ and $Z_p$ to reasonable units with respect to the $X$ and $Y$ axes. We will use $f = 0.017$ (i.e. simulating a lens with 17mm focal length), and $\sigma = 10$, i.e. a scale factor of 10 between the sensor coordinate system and the outside world.

With the internal camera parameters reasonably set, the depth distortion depending on different object distances $Z_p$ becomes reasonable. The range of $Z_p$ from which to generate samples is chosen such that the minimum gives very notable depth distortion without seeming unnatural while the maximum is far enough away to approximate parallel projection, i.e. so that almost no depth distortion is noticed. The distances used in the final algorithm are 0.5 - 2.5m. Table 3.6 show a few examples of the effect the viewing distance has on the projected image of the plate and Fig. 3.14 shows a sampling of all possible viewpoints.

Before calculating the coordinates, the scale factor for homogeneous coordinates, $\lambda'$, remains to be fixed.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \lambda' \begin{bmatrix} \sigma f & 0 & 0 \\ 0 & \sigma f & 0 \\ 0 & 0 & 1 \end{bmatrix} \left( R(\phi, \theta, \psi) \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ Z_p \end{pmatrix} \right) \tag{3.4}$$

Resolving the homogeneous condition in the third row of the equation system, we get $\lambda'$:

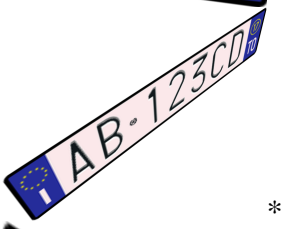
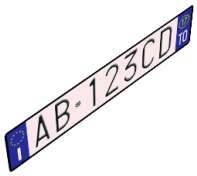
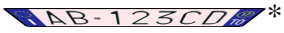
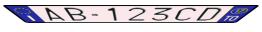
$$\lambda' = \frac{1}{R_{3,1}X + R_{3,2}Y + R_{3,3}Z + Z_p}. \tag{3.5}$$

where $R_{i,j}$ are elements in $R(\phi, \theta, \psi)$ according to (A.9). This finally allows us to calculate the projection $(x', y')$ of any point $(X, Y, Z)$, using (3.4). However, at this stage, we only compute the projection of the corners, i.e. $(X, Y, Z) = (W/2, H, 2, 0)$ , $(-W/2, H, 2, 0)$, $(W/2, -H, 2, 0)$, $(-W/2, -H, 2, 0)$, where $W$ and $H$ are the plate width and height in meters.

### Rescale Coordinates

Varying the viewing distance does not only change the depth perspective of the plate, it also changes the size it appears at (as can be seen in Table 3.6). This is

Table 3.6 Projection at different viewing distances.

| | $Z_p = 0.5m$ | $Z_p = 1.5m$ | $Z_p = 2.5m$ |
|---|---|---|---|
| $\phi = 0°$ $\theta = 50°$ * |  * |  |  |
| $\phi = 45°$ $\theta = -45°$ * |  * |  |  |
| $\phi = 0°$ $\theta = -80°$ * |  |  |  |
| $\phi = 70°$ $\theta = 0°$ * |  * |  |  |

* * * * indicate colors of viewpoints illustrated in Fig 3.14

* indicate that the plate is rescaled to fit page

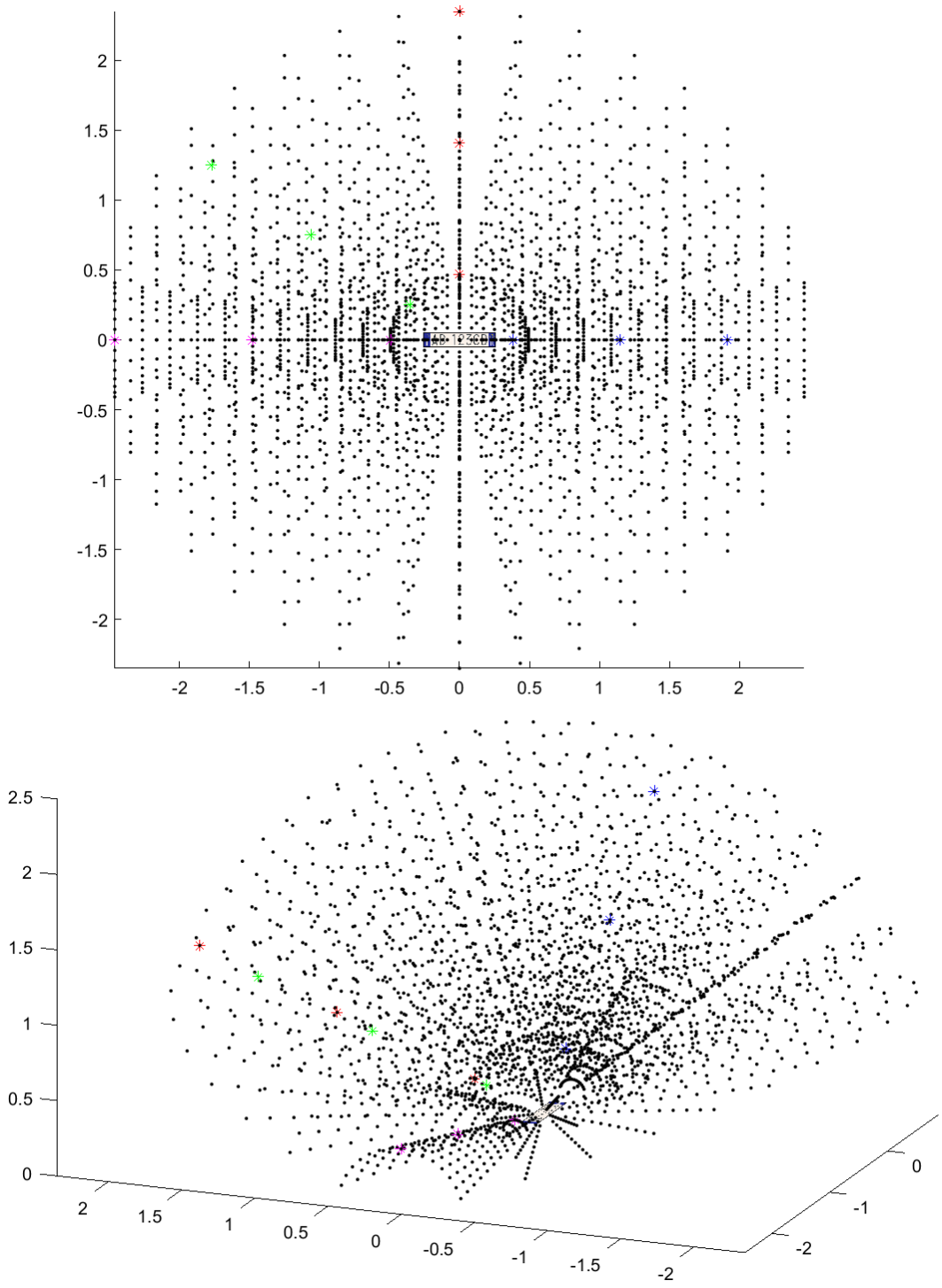



Fig. 3.14 Illustration of some viewpoints from which the plate is projected. Colored * indicate viewpoints from which the plates in Table 3.6 are projected. Left: Top view. Right: Angled view. Note that this figure is using parallel projection rather than perspective projection.

undesirable since the classifiers should recognize plates of a well controlled range of scales (as will be described later in section 4.1.4). Furthermore, it is preferable that the distribution of plate sizes is as uniform as possible, to avoid biasing the classifier towards certain sizes during training (the distribution of the plate sizes after perspective projection is rather complex).

Therefore the coordinates are rescaled to a desired output width in pixels. This range is chosen with two things in mind, i.e. the smallest plates need to have high enough resolution that the characters are still legible, the largest plates should be at least twice as large as the smallest, such that a scale pyramid with a factor of 2 can be used during classification of objects of any scale (this will be described later). Based on this, after comparing samples of various sizes, a minimum pixel width of 75 and a maximum of 200 is used. A few examples of the smallest and largest samples in the database are shown in Fig. 3.15.


Fig. 3.15 Examples of smallest and largest training samples. Note that the plate corners we use for classification are the corners of the white area.

### Transform Image

Finally, as described in section A.3, by determining an affine transformation between the original image corner coordinates and the rescaled projected corners, the plate image, the alpha channel and all the bounding box coordinates are transformed to achieve perspective projection. An example is shown in Fig. 3.16.

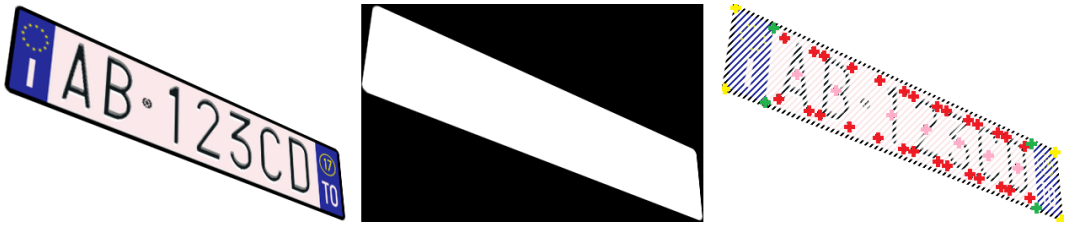



Fig. 3.16 Example after perspective projective is applied. Left: Plate image. Center: Alpha channel. Left: Annotated coordinates of plate and characters.

### 3.2.5 Add plate noise

This module varies the plate appearance, in other ways than the perspective, such as light conditions and dirt. This process is illustrated in Fig. 3.17 and described below.

#### Light Conditions

While the plate colors were already varied in 3.2.1, this module focus on the appearance rather than the actual production colors. The plate is first converted to the HSL color space [[100]] (see Fig. 3.18), which is a more natural representation of colors where the axes represent more independent quantities compared to the RGB color space (in which, changing one value affects both color and brightness at the same time).

The reason HSL was preferred over the HSI color space is because the lightness is equal for all saturated colors while the Intensity of HSI is higher for cyan, magenta and yellow than it is for red, green and blue and because it is spanning the full volume of the color cylinder, so any HSL value has an RGB representation, i.e. it is guaranteed to be invertible after modifications of the color, in particular, hue does not become undefined (hence the white and black slices in Fig. 3.18) when the lightness reaches 0 or 1, this allows the previous color to be remembered if it becomes relevant again after further processing.

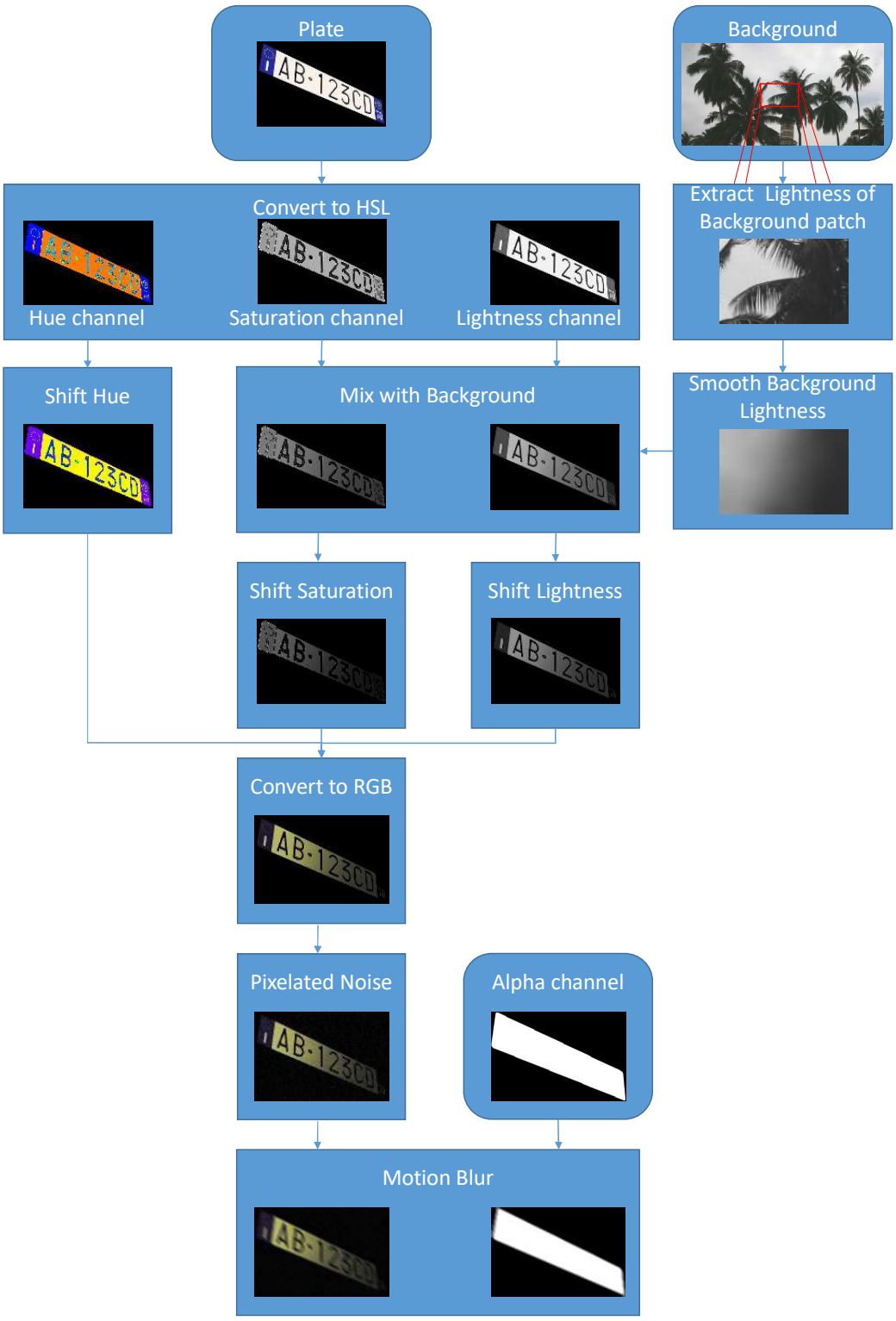



Fig. 3.17 Flow chart of the process of adding plate noise.

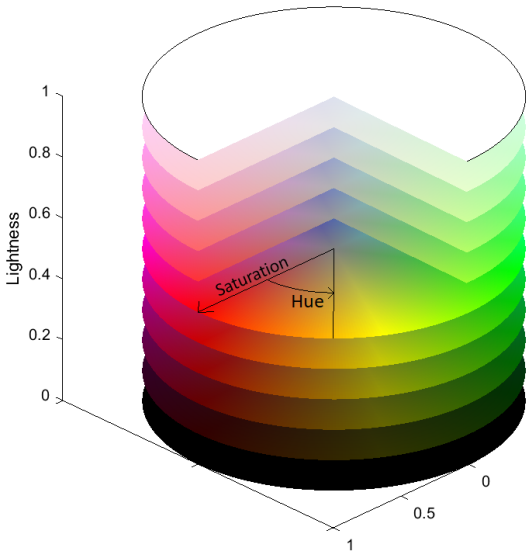



Fig. 3.18 Illustration of the HSL color cylinder with the axes indicated.

To achieve a more naturally varying light conditions over the plate, the lightness of a random section of the background image is analyzed, this patch has a 50% chance to be the area directly behind where the plate will be inserted such that the lightness of the plate will roughly match the background, and 50% chance the be randomly selected from anywhere in the background image. After extraction this patch is first smoothed heavily, using a circular mean filter with a radius up to half of the smallest dimension of the patch. The lightness and saturation channels of the plate image are dampened by the smoothed background according to:

$$L^p_{i,j} = (1-F)L^o_{i,j} + FL^o_{i,j} max(L^b_{i,j}, 0.1),$$

$$S^p_{i,j} = (1-F)S^o_{i,j} + FS^o_{i,j} max(L^b_{i,j}, 0.1),$$

where $L^p_{i,j}$ is the new lightness and $S^p_{i,j}$ the new saturation of pixel $(i, j)$, while $L^o_{i,j}$ and $S^o_{i,j}$ are the original values and $L^b_{i,j}$ is the lightness of the corresponding blurred background pixel, and $F$ is a random weight factor between 0 and 0.8, to ensure at least 20% of the original contrast is maintained so the plate remains legible even if the background happens to be completely black.

Before the image is converted back to RGB colors, all three channels are manipulated slightly to account for perceived color variations due to illumination (e.g. sunlight or street lights). Hue is globally shifted (i.e. the same amount on all pixels) by up to $\pm 30^{\circ}$ and saturation and lightness each by $\pm 15$ percent points.

#### Plate Noise

Finally a pixelated noise filter and a motion blur filter are applied to simulate dirt and motion of the plate during exposure. The pixelated filter adds white noise with a mean of 0 and standard deviation up to 10% (randomized individually for each plate). The motion blur is applied at a random angle and "smears" the pixels up to 6 pixels, using bilinear interpolation. To make the motion blur appear correctly on the edges once the plate is added to the background image, the same motion blur filter is applied on the alpha channel. However, since the filter is symmetric and has the center of mass in the center, the layers holding the bounding box coordinates do not need to be filtered. The minimum limits of both filters are set such that they leave the image unchanged, i.e. the filters vary from not noticeable to quite prominent, because the final images will be subjected to further noise, the maxima were chosen well below the limits where the characters were barely legible.

### 3.2.6 Merge plate and background

In this module the plate image and the background are merged into a training sample. As was already explained in 3.2.2, the plate is inserted in the center of the background image to allow maximum flexibility during training. The process is illustrated in Fig. 3.19, and described below.

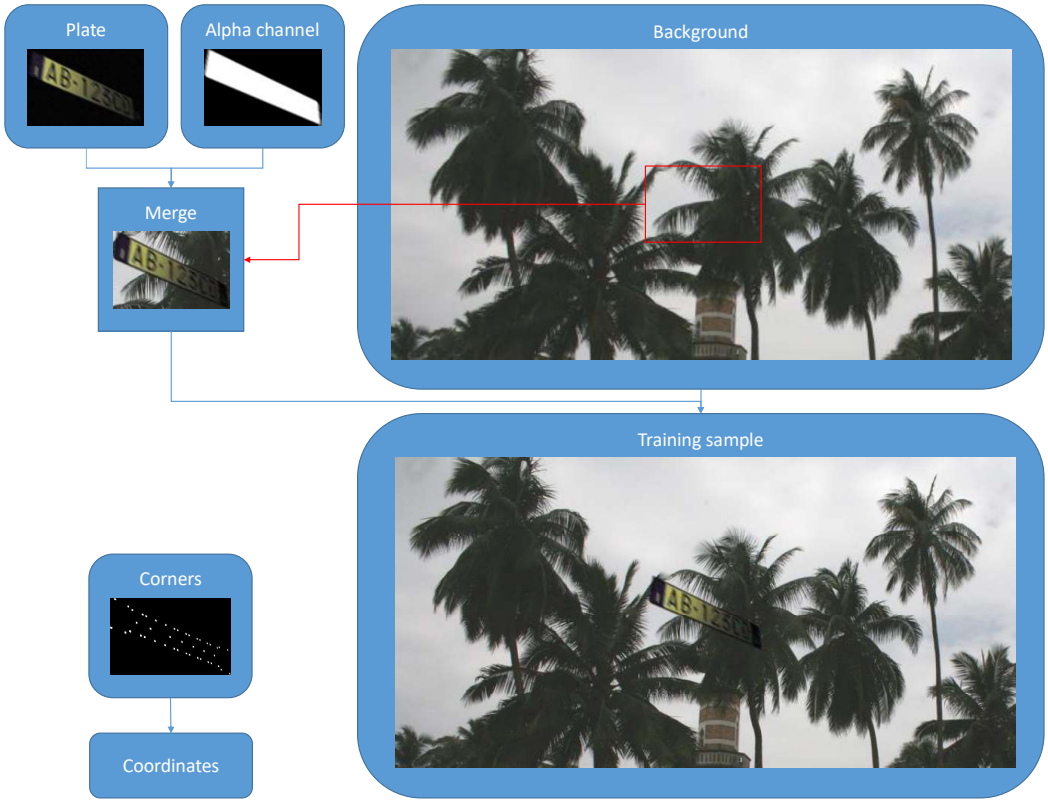



Fig. 3.19 Flow chart of the process of merging the plate image with the background image.

$$T_{k,l} = \begin{cases} P_{i,j}A_{i,j} + B_{k,l}(1 - A_{i,j}) & \text{if } \lfloor (K-I)/2 \rfloor < k < \lfloor (K-I)/2 \rfloor \\ & \text{and } \lfloor (L-J)/2 \rfloor < l < \lfloor (L-J)/2 \rfloor , \\ B_{k,l} & \text{otherwise} \end{cases} \tag{3.6}$$

where $T_{k,l}$ and $B_{k,l}$ are the pixels at coordinate $(k,l)$ of the training sample and background image respectively, both of size $K \times L$ pixels. $P_{i,j}$ and $A_{i,j}$ are the pixels at coordinate $(i,j)$ of the plate image and it's alpha channel, of size $I \times J$ and $\lfloor \cdot \rfloor$ is the floor operator. The offset between the plate and the background coordinate system is given by the upper left corner of the plate image when centered inside the background image $(\lfloor (K-I)/2 \rfloor, \lfloor (L-J)/2 \rfloor)$, i.e:

$$\begin{aligned} i &= k - \lfloor (K-I)/2 \rfloor \\ j &= l - \lfloor (L-J)/2 \rfloor \end{aligned} \tag{3.7}$$

#### Extract Coordinates

Since the more complex image transformations have been performed at this point, and only translations remain, the coordinates of the bounding boxes are extracted from the additional image layers into values, by (discrete) integration of the center of mass and shifting by the offset of the plate image with respect to the background image, given by (3.7):

$$\begin{aligned} x_l &= \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} C_{i,j}^{l}\, i}{\sum_{i=1}^{I}\sum_{j=1}^{J} C_{i,j}^{l}} + \lfloor (K-I)/2 \rfloor \\ y_l &= \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} C_{i,j}^{l}\, j}{\sum_{i=1}^{I}\sum_{j=1}^{J} C_{i,j}^{l}} + \lfloor (L-J)/2 \rfloor \end{aligned}, \tag{3.8}$$

where $x_l$ and $y_l$ are the real values of coordinate $l$ in the training sample, previously encoded as intensity values, $C_{i,j}^{l}$, in additional layers of the plate image.

### 3.2.7 Add acquisition noise

Now that the full scene is assembled, more global noise factors can be considered, typically caused by the camera, which is added by this module. The process is illustrated in Fig. 3.20.

#### Camera movement

To simulate that the camera may move during acquisition, motion blur may be added to the training sample with 50% probability. Note that the motion blur previously added in section 3.2.5 was applied to simulate motion of the object alone, while this simulates movement of the entire scene. The filter is applied in the same manner as previously, but the angle and size is independent between the two application.

#### Bad focus

The camera may not be perfectly focused on the plate itself, therefore, with a 50% probability a Gaussian convolution filter may be applied applied with a standard deviation of up to 2 pixels. This may seem low, but at a higher level plates run a too high risk of becoming illegible in combination with other noise. Fig. 3.21 (left) shows a central cross section of the widest filter ($\sigma = 2$) that may be applied.

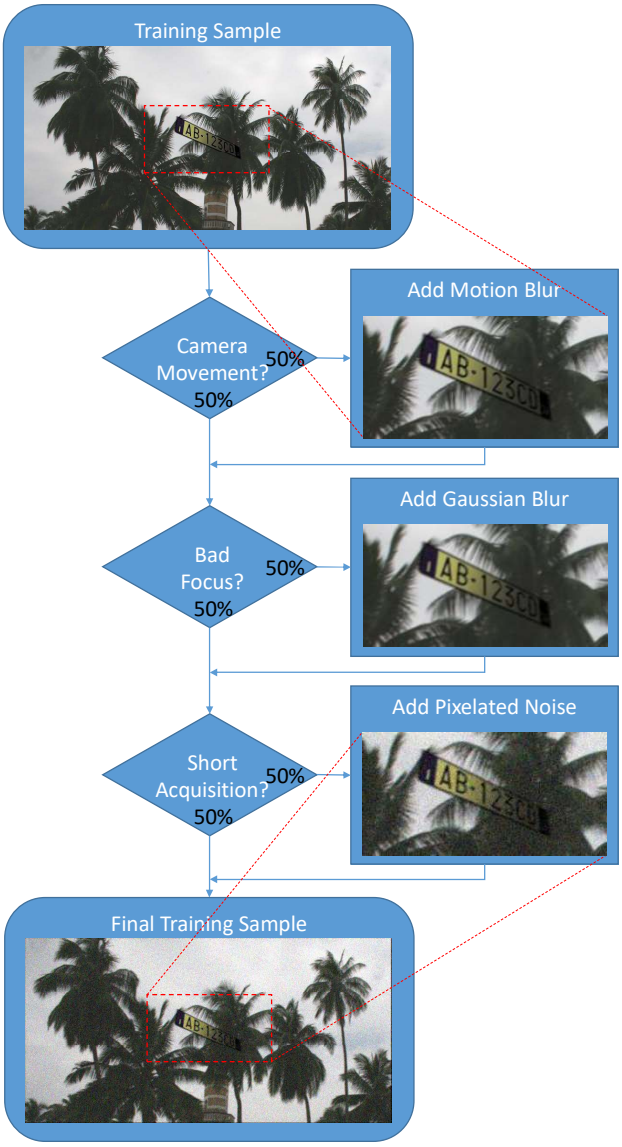



Fig. 3.20 Flow chart of the process of adding acquisition noise to the training sample.

Although the Gaussian bell shape is not strictly the correct shape of the Point Spread Function (PSF) of incoherent light focused by a good lens (such as a normal camera), it is close enough for our purpose, see Fig. 3.21 (right) for a comparison of a Gaussian and Bessel function based PSF [101].

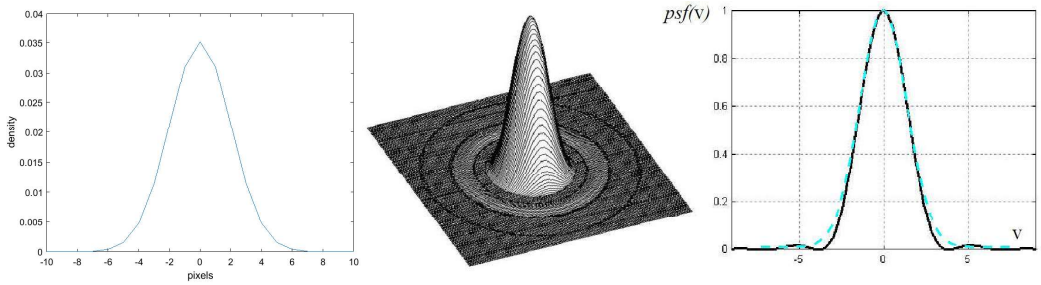



Fig. 3.21 Left: Cross section of the widest Gaussian filter that may be applied. Center: physics based 2D PSF function based on the Bessel $J_1$ function. Right: Cross section comparison with a Gaussian bell curve overlayed in cyan. Source: Center and right figures (except the cyan curve) are copied from [101] with the author's permission.

.

### Low light conditions

Finally, if light conditions are low, or too high ISO number is used (i.e. sensor sensitivity) to reduce acquisition time, pictures may appear grainy. This noise has its source on the individual sensor pixels, partly from quantum noise (random variations in the number of photons hitting with respect to the average expected light flux), partly electronic noise (inaccuracy in measuring the actual incoming light). Both these noise sources are more prominent at low light conditions. Photon quantum variance increases proportionally to the square root of the expected incoming photons, so although the variance increases with a stronger signal, the signal to noise ratio decreases, meaning the noise is less noticeable. Electronic noise on the other hand is more constant, which again means a higher signal to noise ratio if the signal is low.

When mixing multiple random distributions, Gaussian noise is usually a good model, therefore, with 50% probability pixelated Gaussian noise is applied to the full image. The added noise has a mean of 0 and a standard deviation of up to 7%.

Since this noise has the source after the lens, as the light is binned on the sensor pixels, also while simulating it, it needs to be applied last in the process.

With this, a positive training sample is complete and stored to the database along with the labels, i.e. license string and plate and character corner positions. Before repeating the procedure for another positive sample, a negative sample is produced with the same background image, as described in the next section.

### 3.2.8 Negative sample generation

This module produce a negative sample for each positive sample created by the previous 7 modules. To ensure the plate classifier learns to base its classifications on the plate features and ignore other features, the negative samples are generated as similarly as possible to the positive samples. Therefore, the same background image is used for the negative sample and the modules producing positive samples are reused with the same settings, all except the *generate plate template* module, which is instead replaced by random text in this module to ensure that there is enough text in the negative samples that the classifier does not learn to classify any text, such as street signs, as license plates.

Random text is added with 50% probability, the other 50% contain only the background. The full process is summarized in Fig. 3.22 and the details of adding random text is described below, while the reused modules are already described in previous sections above.

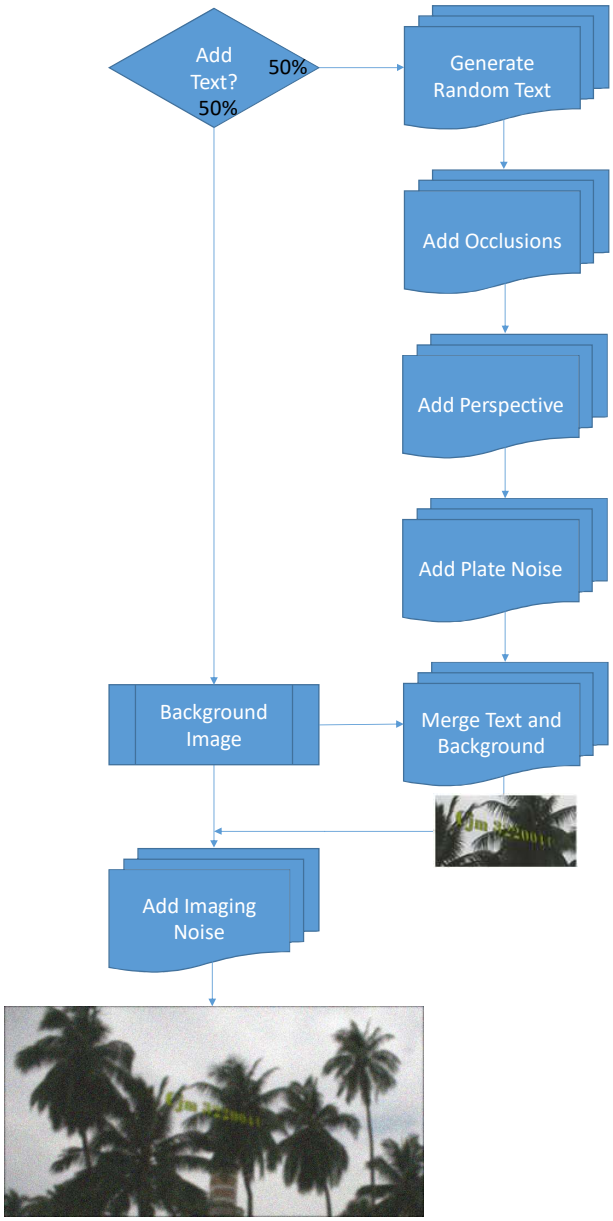



Fig. 3.22 Flow chart of the process of generating a negative training sample.

### Random Text

The random text module adds an image of a random text string up to 20 characters long, of a random color and font in plate of the plate image, and is treated by the following modules as if it were a plate image. Fig. 3.23 shows an example of random text added on the background image.

Fig. 3.23 Example of random text merged on the background image

### 3.2.9 Dataset generation

This concludes the final part for generating a pair of a positive and negative synthetic training image. To generate a data set for training or validation, this process is simply repeated as many times as desired. The number of samples needed to train our classification pipeline is evaluated in section 6.4.1.

Note that before being used for plate or character training, the generated samples are cropped and scaled to suit the desired purpose; the database itself was designed to be flexible and allow for further design decisions on the training process to be made at a later stage without the need to re-generate the entire database from scratch, and also to allow other algorithms to use the database for training, without being too limited by our design decisions. The details on how training samples for the plate and character classifiers are extracted from these generated samples are described in sections 4.1.4 and 4.2.4.

# Chapter 4

# Convolutional Neural Networks: Design and Training

This chapter discusses the design of the CNNs we decided to use to solve different parts of the LPR problem. For an introduction to CCNs and the notation used in this section to describe the networks, see chapter 2.

Since we focus on exploring the feasibility of using synthetic training data, we decided to use a more classic convolutional neural network design, such as AlexNet [84] for classification and Overfeat [7] for localization, rather than a potentially more powerful but more specific design, such as ResNet [86] or GoogLeNet [102].

Early works on LPR divide the task into four subtasks, segmentation of plate shaped objects, classification of the license plates, segmentation of characters and finally character classification. However, since CNNs can generally be trained to recognize object even in cluttered scenes with background noise, segmentation should not be necessary. Instead we try to localize the object in parallel with the classification, as in [7]. We also demonstrate that it works well to train the classification and localization parts together, as suggested by [7] (although not actually performed). However, we still divide the task in two subtasks, *plate detection* and *license number recognition* and design one network for each task, we will refer to them as the *plate network* and the *character network*.

The plate network was designed to take a $128 \times 64$ RGB image as input, this image size is enough to recognize a license plate in, with room for shifting it around at varying scales, since it cannot be expected to be central nor of known scale during

inference. The network should have 2 classification outputs, for positive (plate) and negative (no plate) classifications, and 8 localization outputs, i.e. $x$ and $y$-coordinates for the four corners. Detection is often performed with only 4 units, but since the plate orientation is useful to facilitate the character classification, as will be shown in section 5.5. The details of the final design of this network is illustrated in Fig. 4.1 described in section 4.1, including the training process.

The character network on the other hand was designed for smaller input images, of size $24 \times 40$, which is enough resolution to read a character, while two characters side by side are almost too small to read, the ratio approximately reflects the ratio of the characters to be classified. This network uses $C$ classification output units, where $C-1$ is the number of legal characters for the desired plate (i.e. 32 for Italian plates) and the last unit represents no character present, and 4 location units - top and bottom $y$-coordinates and leftmost and rightmost $x$-coordinates. Unlike the plate network, the exact orientation is not necessary. The final configuration for this network and its training process are described in section 4.2 and the network is illustrated in Fig. 4.4.

Chapter 5 describes how these two networks are applied and combined to achieve the full LPR system capable of multi-plates classification at any scale.

## 4.1 License plate network

The CNN for license plate detection (illustrated in Fig. 4.1) takes an $128 \times 64$ RGB image as input and begins with 4 convolution layers for image feature extraction, followed by a branching into separate stages for classification and localization, each with 3 fully connected layers with no interconnections between the branches after feature extraction part. The output consists of 2 classification neurons (true/false) and 8 localization outputs.

### 4.1.1 Feature extraction

The feature extraction stage begins with a layer of 16 $5 \times 5$ convolutions with a ReLU activation function followed by a $2 \times 2$ max-pooling layer for downsampling, the following 3 convolution layers use $3 \times 3$ convolutions with 32, 64 and 128 convolutions. Like the first layer, each has a ReLU activation function and is

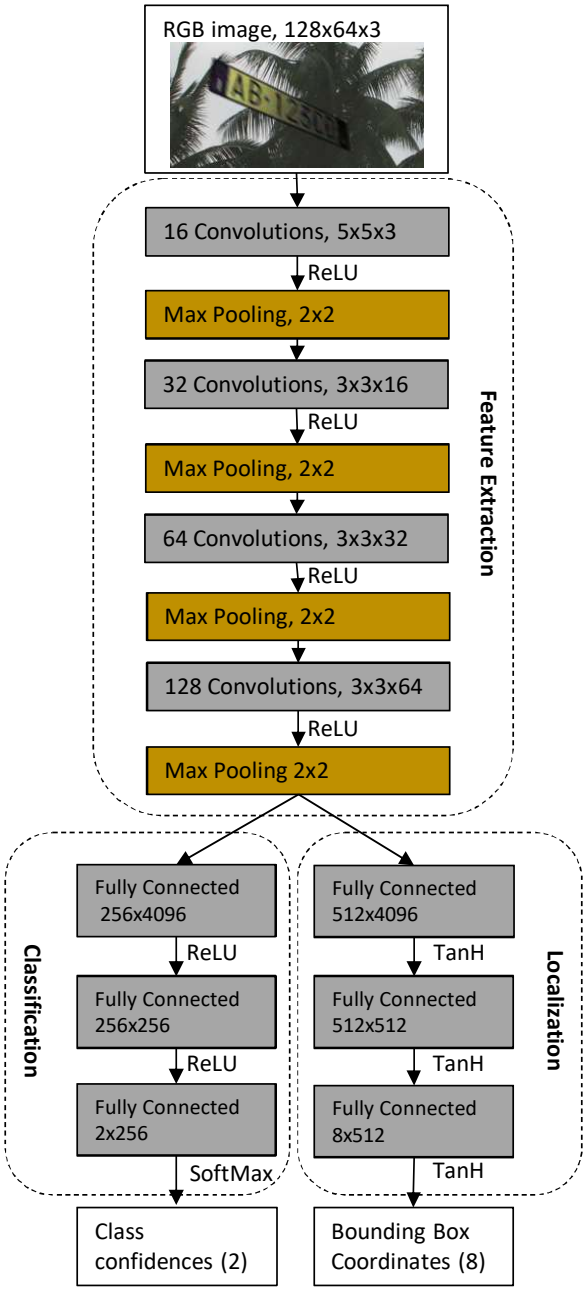



Fig. 4.1 CNN architecture for plate classification and localization.

followed by a $2 \times 2$ max-pooling layer. All convolution layers use padding to retain the spatial input resolution. This means the feature extraction stage ends with a hidden layer of $8 \times 4 \times 128 = 4,096$ neurons. The details for each hidden layer are shown in Table 4.1. As can be seen from the number of connections between the layers, doubling the number of convolutions after each max-pooling keeps the number of connections (and thereby computations) constant. Since the input has only 3 spectral dimensions we can afford to use a larger convolution compared to the following layers without causing a bottleneck.

Table 4.1 Feature extraction details, plate network.

| Layer | Description | Size | Output Dimensions | Parameters | Connections |
|---|---|---|---|---|---|
| Input | Normalized RGB image | | $128 \times 64 \times 3$ | | |
| 1 | 16 Convolutions | $5 \times 5 \times 3$ | $128 \times 64 \times 16$ | 1216 | 9.83M |
| | ReLU | | $128 \times 64 \times 16$ | - | - |
| | Max-pooling | $2 \times 2$ | $64 \times 32 \times 16$ | - | - |
| 2 | 32 Convolutions | $3 \times 3 \times 16$ | $64 \times 32 \times 32$ | 4640 | 9.44M |
| | ReLU | | $64 \times 32 \times 32$ | - | - |
| | Max-pooling | $2 \times 2$ | $32 \times 16 \times 32$ | - | - |
| 3 | 64 Convolutions | $3 \times 3 \times 32$ | $32 \times 16 \times 64$ | 18496 | 9.44M |
| | ReLU | | $32 \times 16 \times 64$ | - | - |
| | Max-pooling | $2 \times 2$ | $16 \times 8 \times 64$ | - | - |
| 4 | 128 Convolutions | $3 \times 3 \times 64$ | $16 \times 8 \times 128$ | 73856 | 9.44M |
| | ReLU | | $16 \times 8 \times 128$ | - | - |
| | Max-pooling | $2 \times 2$ | $8 \times 4 \times 128$ | - | - |
| total | | | | 98208 | 38.2M |

Table 4.2 Classification branch details, plate network.

| Layer | Description | Size | Output Dim. | Params. | Connections |
|---|---|---|---|---|---|
| Input | F.E. stage output | | $8 \times 4 \times 128$ | | |
| 5C | 256 Fully Conn. | $256 \times 4096$ | 256 | 1.049M | 1.049M |
| | ReLU | | 256 | | |
| 6C | 256 Fully Conn. | $256 \times 256$ | 256 | 65792 | 65536 |
| | ReLU | | 256 | | |
| 7C | 2 Fully Conn. | $2 \times 256$ | 2 | 514 | 512 |
| | Softmax | | 2 | | |
| total | | | | 1.115M | 1.115M |

### 4.1.2 Classification

The classification stage consists of 3 fully connected layers gradually decreasing the number of connections towards the 2 output units. The first layer consists of $256 \times 4096$ weights (i.e. 256 hidden units all connected to each of the 4096 (hidden) output neurons of the feature extraction layer), the second $256 \times 256$ weights and the third $2 \times 256$ weights, connecting to the two output units. The first two use ReLU as activation function while the final layer use SoftMax to normalize the output. The details for each hidden layer and a summary of the number of parameters are shown in Table 4.2.

### 4.1.3 Localization

The localization stage also consists of 3 fully connected layers. But since the task is more challenging, with 8 outputs and real valued target values to learn (regression), each layer uses twice as many hidden units compared to the classification stage. Which means the weight matrices become $512 \times 4096$ in the first layer, $512 \times 512$ in the second and $8 \times 512$ in the third, connecting to the 8 output units. Each layer is followed by a TanH activation function. The details for each hidden layer and a summary of the number of parameters are shown in Table 4.3.

Table 4.3 Localization branch details, plate network.

| Layer | Description | Size | Output Dim. | Params. | Connections |
|---|---|---|---|---|---|
| Input | F.E. stage output | | $8 \times 4 \times 128$ | | |
| 5L | 512 Fully Conn. | $512 \times 4096$ | 512 | 2.098M | 2.097M |
| | TanH | | 512 | | |
| 6L | 512 Fully Conn. | $512 \times 512$ | 512 | 262.7K | 262.1K |
| | TanH | | 512 | | |
| 7L | 8 Fully Conn. | $8 \times 512$ | 8 | 4104 | 4096 |
| | TanH | | 8 | | |
| total | | | | 2.364M | 2.363M |

### 4.1.4 Sample selection

The plate training samples are selected by extracting $256 \times 128$ crops from the generated training samples and downscaling them to $128 \times 64$. This means the plates the classifier is represented with will be from 37.5 to 100 pixels wide, and the smallest plates will be too small for the license to be legible, therefore it is not useful to recognizing smaller plates. Plates larger than 100 pixels are recognized by employing a scale pyramid of downscaled input images at classification, as is explained in section 5.1.1.

#### Positive sample extraction

For each epoch, one positive plate training samples is extracted from each positive training image by randomly selecting a shifted $256 \times 128$ crop from the image such that the entire plate is contained within the image as illustrated in Fig. 4.2, that is, a crop with an offset $(x_o, y_o)$ randomized within the ranges $[x_{min}, x_{max}]$ and $[y_{min}, y_{max}]$, where:

$$\begin{aligned} x_{min} &= \max_{l \in platecorner}(x_l) - 256 \\ x_{max} &= \min_{l \in platecorner}(x_l) \\ y_{min} &= \max_{l \in platecorner}(y_l) - 128 \\ y_{max} &= \min_{l \in platecorner}(y_l). \end{aligned} \tag{4.1}$$

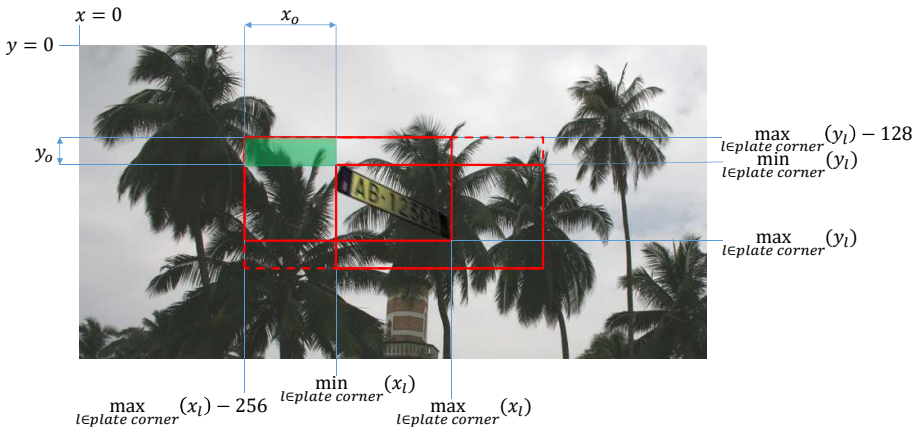



Fig. 4.2 Illustration of the area within which the positive plate training samples is selected. The green area indicate the region of allowed offsets, i.e. of the upper left corner of the crop.

While adjusting the corner locations according the crop offset, $(x_o, y_o)$, they are also normalized to the range $[-1, 1]$ before being used as localization output targets, $(t_x^l, t_y^l)$:

$$\begin{aligned} t_x^l &= \frac{x_l - x_o - 128}{128} \\ t_y^l &= \frac{y_l - y_o - 64}{64} \end{aligned} \tag{4.2}$$

### Negative sample extraction

Negative samples are with 50% probability cropped from the same location as the positive plate training sample, but from the negative image sample. Since the classifier will be presented with images partly including plates, they will otherwise be cropped from the positive samples such that they include up to 50% of a plate. That is, the combination of ranges $x_o \in [128, 384]$ and $y_o \in [64, 192]$ is not allowed, as illustrated in Fig. 4.3.

This leaves some margin of crops that cannot be selected, neither as positive nor negative samples (the area between the red and the green area in Fig. 4.3). This to avoid "confusing" the classifier with too similar samples with different labels during training, while during inference, both classes are acceptable choices (although a positive classification will probably lead to an inaccurate localization since not all corners of the plate are visible).

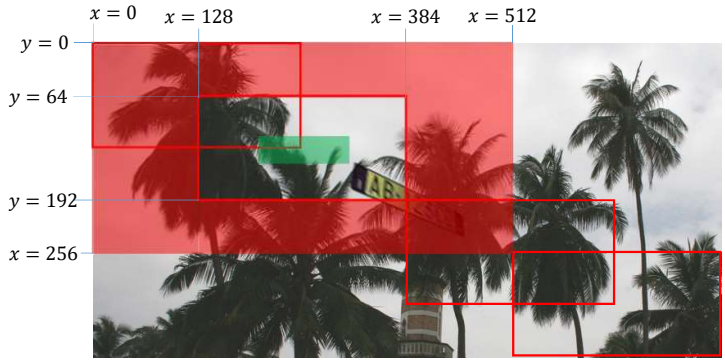



Fig. 4.3 Illustration of the area within which the negative plate training samples is selected. The red area indicate the region of allowed offsets, i.e. the upper left corner of the crop. For comparison the green area representing the allowed region for a positive sample is also shown, from Fig. 4.2. (An offset to the right or below the red area would result in the bottom right corner of the crop region being outside the image.)

### 4.1.5 Training procedures

In order to train the plate detector CNN, we first generate at least 20,000 synthetic images (50% positive and 50% negative samples) using the procedure described in chapter 3. Our experiments show that above 20k samples the gains are begin to stagnate. For more details on how the performance depends on the training set size, see the experiment in section 6.4.1, which shows an evaluation of networks trained with 10k-80k samples. We also generate 5,000 additional images for a validation set.

We feed batches of 128 samples (64 positives, 64 negatives) to the network and train with backpropagation of the cost function defined in (4.3) below, updating the weights after each batch using the AdaGrad [96] algorithm for learning rate adaptation. Our experiments suggested that an initial learning rate of $10^{-4}$ is a good compromise between convergence speed without risking to fail to converge towards a solution. After each epoch, we test the trained network over the validation set of 5,000 synthetic images: the experiments showed that the average validation error derivative becomes zero after about 500 epochs (for a training set of 80k samples, for other training set sizes the number of epochs should be adjusted such that the total number of shown samples during training remains constant at $500 \times 80\text{k} = 40\text{M}$). Concerning the cost function parameters, we experimentally found that $a = 0.1$ and

$\lambda = 10^{-6}$ best minimize the overall validation error when training the plate detector CNN.

Finally, we found dropout [103] with a dropout factor of 0.5 to be useful in the last convolutional layer, and using batch normalization after each convolutional layer speeds up the convergence rate of the training process.

#### Target cost function definition

Since different cost functions are typically preferred for classification and regression, while our neural network combines both in separate branches trained in parallel, we define our own combined cost function as follows.

$$E(W, y^o, t) = aE^c(W, y^c, y^l) + (1-a)E^l(W, y^l, t^l) + \lambda R(W), \tag{4.3}$$

where $W$ represents the learnable parameters (weights and biases) of the network, $y^o$ the network output units divided into classification stage outputs, $y^c$, and localization stage outputs $y^l$ with their corresponding target values $t$, likewise divided into $t^c$ and $t^l$. $E^c$ is the cross-entropy defined in (2.15) of the classification branch output units and $E^l$ is the MSE defined in (2.14) of the localization branch outputs. The factor $a$ drives the balance between the contributions of the localization and the classification errors to the overall cost function. Finally, the component $R(W)$ represents an optional regularization term that helps preventing overfitting to the training samples and is defined as the squared $L^2$-norm of all the weights in the network.

## 4.2 Character network

The CNN for character detection (illustrated in Fig. 4.4) takes a $24 \times 40$ RGB image as input and begins with 3 convolution layers for image feature extraction, like the plate network it is followed by a branching into separate classification and localization parts, each with 3 fully connected layers. The output consists of $C = C_t + 1$ classification neurons, where $C_t$ is the number of legal characters for the plate type ($C_t = 32$ for Italian plates), and 4 localization outputs.

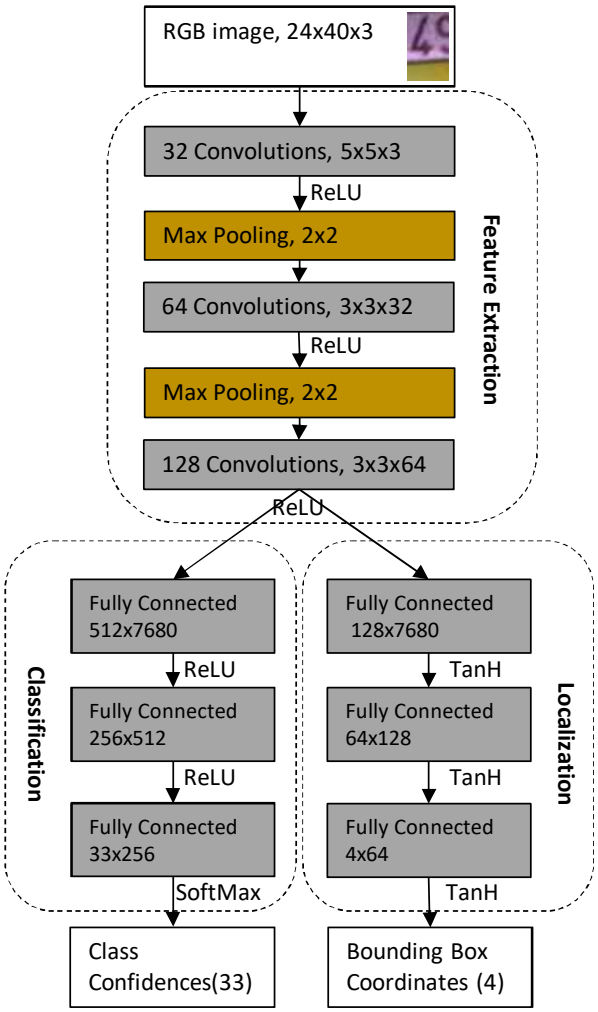



Fig. 4.4 CNN architecture for character classification and localization network.

Table 4.4 Feature extraction details, character network.

| Layer | Description | Size | Output dim. | Params. | Connections |
|---|---|---|---|---|---|
| Input | | | $24 \times 40 \times 3$ | | |
| 1 | 32 Convolutions | $5 \times 5 \times 3$ | $24 \times 40 \times 32$ | 2,432 | 2.30M |
| | ReLU | | $24 \times 40 \times 32$ | - | - |
| | Max-pooling | $2 \times 2$ | $12 \times 20 \times 32$ | - | - |
| 2 | 64 Convolutions | $3 \times 3 \times 32$ | $12 \times 20 \times 64$ | 18,496 | 4.42M |
| | ReLU | | $12 \times 20 \times 64$ | - | - |
| | Max-pooling | $2 \times 2$ | $6 \times 10 \times 64$ | - | - |
| 3 | 128 Conv. | $3 \times 3 \times 64$ | $6 \times 10 \times 128$ | 73,856 | 4.42M |
| | ReLU | | $6 \times 10 \times 128$ | - | - |
| total | | | | 94,784 | 11.15M |

### 4.2.1 Feature extraction

Since the dimensions of the character network input is considerably smaller than the plate network input dimensions, the feature extraction stage employs only 3 layers. The first layer has 32 $5 \times 5$ convolutions with a ReLU activation function followed by a $2 \times 2$ max-pooling layer for downsampling. The second and third convolution layers use $3 \times 3$ convolutions with 64 and 128 convolutions respectively, and the ReLU activation function. But only the second layer is followed by a $2 \times 2$ max-pooling layer, as the third max-pooling was found to have a negative impact on the accuracy.

Like the plate classifier, all convolution layers use padding, yielding a feature extraction stage ending with a hidden layer of $5 \times 10 \times 128 = 7,680$ neurons. This is more than the plate classifier, however, the character network ultimately also has more outputs than the plate network. The details for each layer are shown in Table 4.4.

### 4.2.2 Classification

Like the plate network, the classification stage consists of 3 fully connected layers. The first layer consists of $512 \times 7680$ weights, the second of $256 \times 512$ weights and the third of $2 \times 256$ weights, connecting to the $C$ output units. The first two use ReLU as activation function while the final layer use SoftMax to normalize the output. The

Table 4.5 Classification branch details, character network.

| Layer | Description | Size | Output Dim. | Params. | Connections |
|---|---|---|---|---|---|
| Input | F.E. stage output | | $6 \times 10 \times 128$ | | |
| 5C | 512 FC neurons | $512 \times 7680$ | 512 | 3.93M | 3.93M |
| | ReLU | | 512 | | |
| 6C | 256 FC neurons | $256 \times 512$ | 256 | 131K | 131K |
| | ReLU | | 256 | | |
| 7C | $C$ FC neurons | $C \times 256$ | $C \approx 33$ | 8481 | 8448 |
| | SoftMax | | $C \approx 33$ | | |
| total | | | | 4.07M | 4.07M |

details for each hidden layer and a summary of the number of parameters are shown in Table 4.5.

As can be seen, comparing table 4.5 to 4.2 the character branch requires significantly more parameters than the plate counterpart. In part this is explained by the increase in outputs. But the accuracy requirements are also higher on this network because each correct plate classification requires multiple character classifications, with no errors. So each classification is extremely critical at this stage.

### 4.2.3 Localization

This branch also consists of 3 fully connected layers. But with only 4 localization outputs, this is the lightest of the four branches. The weight matrices are $128 \times 7680$ in the first layer, $64 \times 128$ in the second and $4 \times 64$ in the final layer connecting to the 4 output units. And as with the plate localization stage, each layer is followed by a TanH activation function. The details for each hidden layer and a summary of the number of parameters are shown in Table 4.6.

### 4.2.4 Sample selection

The character training sample selection is not as straightforward as for plate training, because of the way localized plates are rectified before character classification in the pipeline, as described in section 5.5. Since plates are expected to appear

Table 4.6 Localization branch details, character network.

| Layer | Description | Size | Output Dim. | Params. | Connections |
|---|---|---|---|---|---|
| Input | F.E. stage output | | $6 \times 10 \times 128$ | | |
| 5L | 128 FC neurons | $128 \times 7680$ | 128 | 983k | 983k |
| | TanH | | 128 | | |
| 6L | 64 FC neurons | $64 \times 128$ | 64 | 8,256 | 8,192 |
| | TanH | | 64 | | |
| 7L | 4 FC neurons | $4 \times 64$ | 4 | 260 | 256 |
| | TanH | | 4 | | |
| Total | | | | 992k | 991k |

approximately horizontally when shown to the character classifier, the training samples are "un-rotated" to a degree before sample extraction.

### Un-rotation

Based on observations of how accurate the plate localizer is, the plates are not rectified to undo the perspective projection from a random angle (which this is exactly what we aim to do in 5.5 after plate localization). Instead we only un-rotate the plate, based on the angle of the horizontal central line of the plate, up to a 5° error. This is based on measurements of the angular error of the plate localizer applied on a validation set of synthetic images, with doubled margin to account for higher error on real images. Furthermore, the images are only rotated and not rectified, because we observed that plates often remained skewed to some degree after the rectification. This occurs when the plate localizer successfully predicts the angular rotation of the plate by accurately representing the y-coordinates of the plate, but the fine difference between the x-coordinates of the pairs of leftmost and rightmost corners are inaccurate.

Therefore, to keep the characters as skewed as they may be shown, but on a relatively horizontal line (as also shown to the classifier during inference), the samples images are rotated such that the horizontal central line of the plate is less than $\pm 5^\circ$ of angle relative to the actual horizontal line, as illustrated in Fig. 4.5.

Fig. 4.5 Illustration of un-rotation of a sample before character training sample extraction. Up to $\pm 5^\circ$ of rotation is intentionally left.

### Positive sample extraction

From each un-rotated positive sample image, each character in the plate is cropped once as a positive character sample in the training set. A positive character sample requires the character to be entirely contained in the sample and such that the center of the crop is overlapping the character (such that, in the case of another character also being visible within the crop, it is well defined which character the sample represents and which is background noise). The patches are cropped and rescaled so that the characters in the patch are 16-40 pixels high and 10-24 pixels wide and randomly positioned within the crop without any margin.

A few positive character training samples are shown in Fig. 4.6.

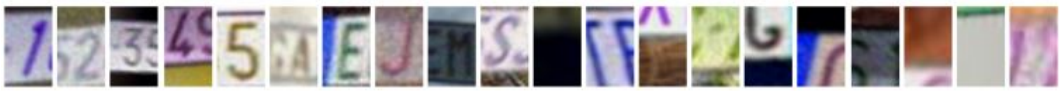


Fig. 4.6 Examples of positive (10 leftmost) and negative (10 rightmost) samples for training the character detector CNN.

### Negative sample extraction

Negative character samples are also cropped from the positive image. In this context, a character sample is negative if it contains no character at all or just a partial view of the character, which the character detector will frequently encounter when deployed on the field. To ensure a character is not randomly included we verify that the Intersection over Union (IoU, Jaccard Index [104]) of the potential crop area and the

area within the 4 corner labels of each character does not exceed 50%. The IoU is defined as

$$J(A,B) = \frac{\mu(A \cap B)}{\mu(A \cup B)}, \tag{4.4}$$

where $A$ and $B$ are two sets of points and $\mu(\bullet)$ measure the area of such a set. Some negative character training samples are shown in Fig. 4.6.

### 4.2.5 Training procedures

To train the character network we use all character samples available in all the 40k positive training images already used for plate training as well as an equal amount of negative samples, with the procedure described in sections 4.2.4 and 4.2.4. Unlike during plate training, the character samples are not read and shifted every epoch; although this feature has a positive impact on the training it was initially implemented due to memory restrictions for the plate detector only. But because there is more processing related to the character extraction it would slow down the training time significantly if this strategy was applied here.

To equalize the number of training samples of each character, we restart the extraction process after all characters have been extracted once, but reject samples of any character not already containing the maximum amount collected of any character. This is repeated until all character classes are equally represented, for each accepted sample a negative sample is added as well the balance of 50% negative samples. Overall, this amounts to 960k training samples.

The CNN responsible for character detection is trained according to the same procedure described for the plate detector network in section 4.1.5. However, in this case we use batches of 512 samples each and we experimentally found that the error function derivative becomes negligible after about 200 epochs. Concerning the cost function parameters, we set $a = 0.6$ and $\lambda = 10^{-6}$ when training the character detector CNN.

## 4.3 Fully convolutional conversion

The trained networks require a fully convolutional form in their practical deployment in chapter 5. Namely, units in the first fully connected layer in the classification and localization branches are rearranged into a convolutional layer with filter size equal to the dimension of the output features from the last feature extraction layer and the following fully connected layers are converted to $1 \times 1$ convolutions. This procedure is described in more detail in section 2.7.

However, such design significantly increased the training time, therefore the networks are trained with fully connected layers. If the training samples were larger than the input window however, to allow for multiple shifts during training, the fully convolutional training approach may be favourable. However, since that would only include multiple shifted copies of the same training samples, it is doubtful if such an approach would have any positive impact on the training with respect to our current training approach.

The structure of the plate network after rearranging the trained weights into the fully convolutional form is detailed in Table 4.7, the last column specifying the output size given an input image of arbitrary size, corresponding fully connected network is described in Tables 4.1 - 4.3.

Likewise the character network is rearranged into the fully convolutional network detailed in Table 4.8, for reference the original fully connected network was detailed in Tables 4.4 - 4.6.

As can be seen comparing the output size of arbitrary sized images from the plate and character networks (see the last column of Tables 4.7 and 4.8) the plate network is downscaled by a factor of 16 while the character network is downscaled by a factor of 4, this is determined by the Max-pooling layers and affects the distance between classification shifts. While 16 pixels leaves plenty of overlap between plate classifications to avoid risking to miss a plate, it would be too much to on the character classifier. In fact, compared to the plate network it has 1 convolution layer less but two max-pooling layers less - one extra layer was removed at the end of the feature extraction stage in order to produce a denser classification map.

Table 4.7 Fully convolutional network details, plate network.

| Layer | Description | Size | Training Dimensions | Inference Dimensions |
|---|---|---|---|---|
| Input | Normalized image | | $128 \times 64 \times 3$ | $(128+M) \times (64+N) \times 3$ |
| 1 | 16 Convolutions | $5 \times 5 \times 3$ | $128 \times 64 \times 16$ | $(128+M) \times (64+N) \times 3$ |
| | ReLU | | $128 \times 64 \times 16$ | |
| | Max-pooling | $2 \times 2$ | $64 \times 32 \times 16$ | $(64+M/2) \times (32+N/2) \times 16$ |
| 2 | 32 Convolutions | $3 \times 3 \times 16$ | $64 \times 32 \times 32$ | $(64+M/2) \times (32+N/2) \times 32$ |
| | ReLU | | $64 \times 32 \times 32$ | |
| | Max-pooling | $2 \times 2$ | $32 \times 16 \times 32$ | $(32+M/4) \times (16+N/4) \times 32$ |
| 3 | 64 Convolutions | $3 \times 3 \times 32$ | $32 \times 16 \times 64$ | $(32+M/4) \times (16+N/4) \times 64$ |
| | ReLU | | $32 \times 16 \times 64$ | |
| | Max-pooling | $2 \times 2$ | $16 \times 8 \times 64$ | $(16+M/8) \times (8+N/8) \times 64$ |
| 4 | 128 Convolutions | $3 \times 3 \times 64$ | $16 \times 8 \times 128$ | $(16+M/8) \times (8+N/8) \times 128$ |
| | ReLU | | $16 \times 8 \times 128$ | |
| | Max-pooling | $2 \times 2$ | $8 \times 4 \times 128$ | $(8+M/16) \times (4+N/16) \times 128$ |
| 5C | 256 Convolutions | $8 \times 4 \times 128$ | $1 \times 1 \times 256$ | $(1+M/16) \times (1+N/16) \times 256$ |
| | ReLU | | $1 \times 1 \times 256$ | |
| 6C | 256 Convolutions | $1 \times 1 \times 256$ | $1 \times 1 \times 256$ | $(1+M/16) \times (1+N/16) \times 256$ |
| | ReLU | | $1 \times 1 \times 256$ | |
| 7C | 2 Convolutions | $1 \times 1 \times 256$ | $1 \times 1 \times 2$ | $(1+M/16) \times (1+N/16) \times 2$ |
| | Softmax | | $1 \times 1 \times 2$ | |
| 5L | 512 Convolutions | $8 \times 4 \times 128$ | $1 \times 1 \times 512$ | $1+M/16 \times (1+N/16) \times 512$ |
| | TanH | | $1 \times 1 \times 512$ | |
| 6L | 512 Convolutions | $1 \times 1 \times 512$ | $1 \times 1 \times 512$ | $1+M/16 \times (1+N/16) \times 512$ |
| | TanH | | $1 \times 1 \times 512$ | |
| 7L | 8 Convolutions | $1 \times 1 \times 512$ | $1 \times 1 \times 8$ | $1+M/16 \times (1+N/16) \times 8$ |
| | TanH | | $1 \times 1 \times 8$ | |
| | | Total output: | $1 \times 1 \times (2+8)$ | $(1+M/16) \times (1+N/16) \times (2+8)$ |

Table 4.8 Fully convolutional network details, character network.

| Layer | Description | Size | Train. Dim. | Inference Dim. |
|---|---|---|---|---|
| Input | Normalized image | | $24 \times 40 \times 3$ | $(24+M) \times (40+N) \times 3$ |
| 1 | 32 Convolutions | $5 \times 5 \times 3$ | $24 \times 40 \times 32$ | $(24+M) \times (40+N) \times 3$ |
| | ReLU | | $24 \times 40 \times 32$ | |
| | Max-pooling | $2 \times 2$ | $12 \times 20 \times 32$ | $(12+M/2) \times (20+N/2) \times 32$ |
| 2 | 64 Convolutions | $3 \times 3 \times 32$ | $12 \times 20 \times 64$ | $(12+M/2) \times (20+N/2) \times 64$ |
| | ReLU | | $12 \times 20 \times 64$ | |
| | Max-pooling | $2 \times 2$ | $6 \times 10 \times 64$ | $(6+M/4) \times (10+N/4) \times 64$ |
| 3 | 128 Convolutions | $3 \times 3 \times 64$ | $6 \times 10 \times 128$ | $(6+M/4) \times (10+N/4) \times 128$ |
| | ReLU | | $6 \times 10 \times 128$ | |
| 5C | 512 Convolutions | $6 \times 10 \times 128$ | $1 \times 1 \times 512$ | $(1+M/4) \times (1+N/4) \times 512$ |
| | ReLU | | $1 \times 1 \times 512$ | |
| 6C | 256 Convolutions | $1 \times 1 \times 512$ | $1 \times 1 \times 256$ | $(1+M/4) \times (1+N/4) \times 256$ |
| | ReLU | | $1 \times 1 \times 256$ | |
| 7C | $C$ Convolutions | $1 \times 1 \times 256$ | $1 \times 1 \times C$ | $(1+M/4) \times (1+N/4) \times C$ |
| | Softmax | | $1 \times 1 \times C$ | |
| 5L | 128 Convolutions | $6 \times 10 \times 128$ | $1 \times 1 \times 128$ | $1+M/4 \times (1+N/4) \times 128$ |
| | TanH | | $1 \times 1 \times 128$ | |
| 6L | 64 Convolutions | $1 \times 1 \times 128$ | $1 \times 1 \times 64$ | $1+M/4 \times (1+N/4) \times 64$ |
| | TanH | | $1 \times 1 \times 64$ | |
| 7L | 4 Convolutions | $1 \times 1 \times 64$ | $1 \times 1 \times 4$ | $1+M/4 \times (1+N/4) \times 4$ |
| | TanH | | $1 \times 1 \times 4$ | |
| | | Total output: | $1 \times 1 \times (C+4)$ | $(1+M/4) \times (1+N/4) \times (C+4)$ |

### 4.3.1 Fully convolutional networks and convolution padding

In theory, convolution padding should not be used during training to make the convolutional network observations identical to those of a sliding window classifier. During fully convolutional application, the padding is only applied on the outside of the image, while the sliding windows inside will have real image data rather than padding as during training.

Because of this, much effort was put into designing networks without padding and adapting the sample extraction to compensate for this. However, these networks never performed as well as the simpler training technique with padding, hence the result and the described approach in this thesis are based on that.

A reason why the padding does not impact our performance negatively is because of the reclassification employed (see sections 5.4 and 5.8), which extracts a window of the size of the training samples and thus applies the padding identically in the same manner as during the training phase. It could be argued that the reason the reclassification is needed is because the padding is employed incorrectly, it may indeed be less needed with networks without padding, however the main reason to apply the reclassification is to present the network with an image of as close to optimal scale and position as possible, and this reason remains regardless of the padding.

Since the final classification is still employed correctly, the advantage of using padding seems to more than compensate for the disadvantage of using it in the fully convolutional classification. We also attempted to train unpadded networks to use only for the fully convolutional classification and use the padded network for the re-classification, however, this did not improve the performance but only complicated the design.

# Chapter 5

# Automatic License Plate Recognition System

This chapter describes the pipeline surrounding the two core classifiers described in the chapter 4. Fig. 5.1 shows an overview of the pipeline, detailed in the following sections.

## 5.1 Preprocessing

In the following, we assume that one RGB image is provided as input to the system. Each pixel in the image is normalized according to the pixel mean and standard deviation values computed over the training images to accelerate learning. In the following, we refer to such image as the *query* image.

### 5.1.1 Scale invariant representation

In order to achieve scale invariance (typical surveillance cameras resolution may range from at least 1920×1080 full HD to 640×480 VGA), we rely on a scale-pyramid approach. We repeatedly downsample the query image such that the height and width of the $(n+1)^{th}$ layer is half that of the $n^{th}$ layer. The query image is repeatedly downsampled until it becomes equal to or smaller than the plate detector input window (with padding applied to the final layer if needed).

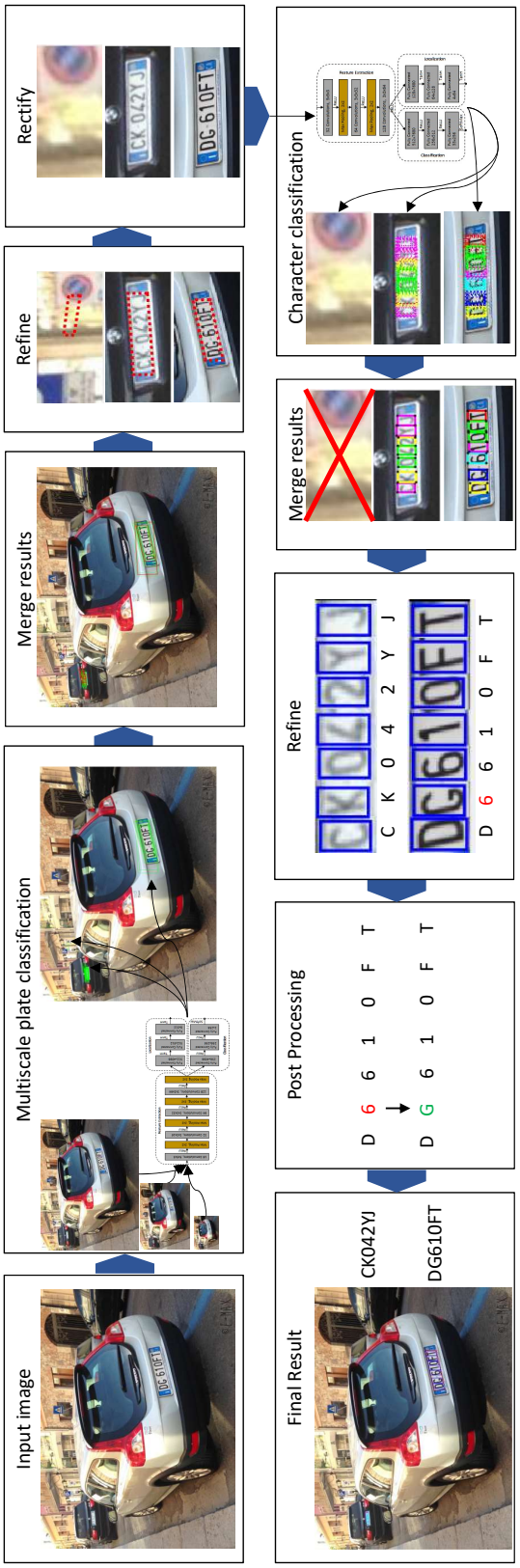



Fig. 5.1 Summary of the classification pipeline architecture.

Since the plate detector was trained on plates with a scale difference larger than a factor of two, plates in the $n^{th}$ layer not fitting the plate detector input window will fit at least in one of the following layers, enabling plate detection at any scale (down to the limit at which the plate in the source image is no longer readable, as explained in section 4.1.4).

Note that detecting plates in the downscaled pyramid layers does not increase the complexity significantly, as the total additional area of the $n$ downsampled layers does not exceed 33% of the original image area ($\Sigma_{n=1}^{\infty} \frac{1}{4^n} = \frac{1}{3}$). Therefore we preferred implementing a classical scale pyramid rather than a scale robust CNN such as [89, 105], which provides *some* robustness to scale variations with ROI pooling. However, the scale ranges at which they can reliably detect objects is still bounded and depends on the network topology and on the training samples. Conversely, our design is scale-agnostic both concerning the network topology and the training samples. Furthermore, such architectures are designed for producing loose, category agnostic, bounding boxes, while our approach yields plate-tight bounding boxes enabling rectification, described in section 4.1.

## 5.2 Fully convolutional plate detection

Plates in the query image are detected by a fully convolutional implementation of the plate detector network described in section 4.1. Each layer of the query pyramid is independently fed to the plate detector CNN. We recall that the network includes 4 pooling layers in its convolutional stage, so adjacent input windows are shifted by $2^4 = 16$ pixels, vertically and/or horizontally. The network returns one response for each of the possible $128 \times 64$ window shifts spanning over the pyramid layer. For each window, the network predicts if it contains a plate as well as the coordinates of its vertices, Fig. 5.2 shows an example of the output of the fully convolutional classifier (the locations of negative classifications are omitted). In the following, we define *platebox* as the output of the network for a given input window. A platebox defines i) the confidence that the window (entirely) contains a plate and ii) the predicted coordinates of the plate corners. Adjacent platebox views overlap completely but by 16 pixels vertically or horizontally.


Fig. 5.2 Example of classification output of the fully convolutional plate classifier. Only localizations where the plate classification confidence is larger than 0.5 are shown.

## 5.3 Plate merging

Since the input windows overlap, a plate may appear in part or entirely within multiple adjacent windows in the same scale layer, as well as in different layers because the scale range of trained plates is slightly larger than the scale factor between the scale layers. Therefore, partial platebox views need to be rejected and redundant plateboxes need to be merged.

Since the coordinate predictions are normalized within their viewing window, they first need to be converted to absolute positions before they can be compared for merging (and related to the image graphics as in Fig. 5.2). This is done by:

$$\begin{aligned} x &= ((x_{norm}+1)W/2+S)2^{n-1} \\ y &= ((y_{norm}+1)H/2+S)2^{n-1}, \end{aligned} \tag{5.1}$$

where $(x, y)$ are real valued absolute coordinates in pixel-units of the original image scale, $(x_{norm}, y_{norm})$ the coordinates predicted by the network, normalized to [-1,1] within their window view. $W = 128$, $H = 64$ (window width and height at the original image scale), $S = 16$ (window shift step size) and $n$ is the layer index in the scale pyramid of the classified image, as defined in section 5.1.1).

First, plateboxes that have a confidence lower than a threshold $\Theta_1^p$ are discarded, as they are expected to contain no plate or a partial view of a plate. During training and validation, accuracy is typically reported at $\Theta_1^p = 0.5$. However, by evaluating the performance on a validation this can be adjusted if desired, a decreasing the threshold will improve recall while increasing it will improve precision of the classification step. How this and other thresholds introduced later in this chapter affect the end performance, is evaluated in section 6.3.

Next, for all the remaining plateboxes, the overlap is computed between each pair of plateboxes, where the overlap is measured in terms of the IoU, defined in (4.4). The pair of plateboxes with highest overlap is merged into a new platebox, such that its vertex coordinates are computed as the average of plateboxes to be merged. When already merged plate boxes are subject to more merging, the average is weighted by the number of original plate boxes each originated from. This procedure is iterated until no more pairs of plateboxes remain such that the IoU is higher than a threshold $\omega^p$.

Note that this process is similar to the method proposed in [7] for merging generic object bounding boxes. However, in [7] the overlap between two boxes is defined as the average distance between the center of each box and the center of the box resulting from the overlap of the two boxes. While the two distance metrics perform similarly in our application, the IoU metric is scale invariant, unitless and is bounded in the range $[0, 1]$, which simplifies finding a suitable threshold for merging, regardless of application and scale of the objects. In fact, we find that using a threshold $\omega = 0.5$ works well for both plates and characters, although the former are typically far apart while the latter are grouped close together.

## 5.4 Refine plate classification

Initial experiments revealed that networks such as that in Fig. 4.1 are more precise at localizing objects well centered in the input window and whose size is comparable to the average size of the objects in the training images (see Fig. 6.10). Therefore, we crop a rectangular patch around each platebox from the query image and resize them to $128 \times 64$ to fit the classifier network for a single classification. The patches are tailored such that the platebox is centered and the size of the platebox corresponds to the average size of plates in the training samples, to provide as good conditions as possible to make a final classification. The platebox position and confidence values are refined according to the network output. If the updated platebox prediction is below a threshold $\Theta_2^p$, the platebox is discarded.

Plate localization refinement brings an almost negligible complexity increase compared to the already applied and necessary fully convolutional search over the query image. Even so, ideally this step should not be needed with a perfect network and to our knowledge, this simple trick to increase the accuracy has not been used before. Nevertheless, the impact this re-classification has on the overall accuracy is significant (10% IoU increase and over 3% precision and recall increase of final classifications), as the experiment in section 6.4.4 shows.

## 5.5 Plate rectification

Since detected plates may appear under any arbitrary perspective, which complicates character recognition later on, we rectify the plate. i.e. we apply a geometric transformation which maps each platebox to a rectangular area of predefined size and perspective to improve the performance of the following character detection step, as described in section A.3, and already used in section 3.2.4, although the purpose this time is reversed.

Fig. 5.3 Example of the rectification of a plate based on the refined corners localizations from section 5.4. As can be seen, if the horizontal difference between the of the corners on each side is not bad, the horizontal orientation of the plate will be corrected, but it will appear skewed.

Before performing such transformation, we estimate the corresponding transformation parameters. Borrowing in part the notation from [106], let us indicate the $(x,y)$ coordinates of the four corners of the platebox detected by the network in the query image (the source platebox) as

$$X^s = \begin{bmatrix} x_1^s & x_2^s & x_3^s & x_4^s \\ y_1^s & y_2^s & y_3^s & y_4^s \end{bmatrix},$$

where the subscripts indicate the corner indices and the coordinates are normalized with respect to the query image size. Now, let us indicate the normalized coordinates of the corners of a corresponding target platebox in homogeneous coordinates as

$$X^t = \begin{bmatrix} x_1^t & x_2^t & x_3^t & x_4^t \\ y_1^t & y_2^t & y_3^t & y_4^t \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

where $X^t$ is such that i) the target platebox corners are located at predefined, known a priori, positions ii) the target platebox edges are parallel to the canonical axes (i.e. matching the coordinates of Fig. 3.2). Then, we find the transformation matrix

$$A_t = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{32} \end{bmatrix},$$

such that $X^s = A_t X^t$. We recall that the source platebox coordinates, $X^s$ are known (predicted) and the target platebox coordinates $X^t$ are fixed in advance. Hence,

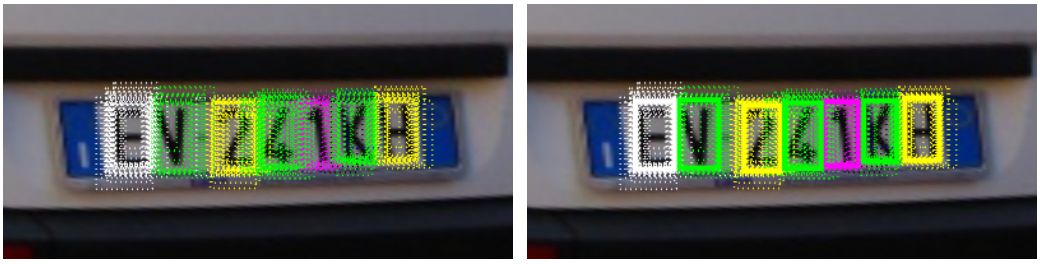
$X^s = A_t X^t$ represents an overdetermined system of 8 equations and 6 unknown variables (the elements of $A_t$). Therefore, we find the least squares solution given by $A_t = X^s (X^t)^+$.

Once the parameters of the geometric transformation are known, the RBG values of each pixel in the target platebox can be computed from corresponding pixels from the source image as follows. Let us indicate the position of the i-th pixel in the source platebox $[x_i^s \; y_i^s]^T$ and the position of the corresponding pixel in the target platebox as $[x_i^t \; y_i^t \; 1]^T$. For each pixel within the target platebox, we find the corresponding coordinates in the source platebox as $[x_i^s \; y_i^s]^T = A_t [x_i^t \; y_i^t \; 1]^T$, linearly interpolating nearby pixels. Taking into account that the corners $X^s$ of a plate may have been predicted by the plate detection network with some error, we extract a patch of $200 \times 100$ pixels centered on the target platebox such that the platebox is 30 pixels high and the width is such that the aspect ratio of the plate format is achieved.

Notice that in [106] the authors solve for the unknown $X^s$ given a target $X^t$ and the parameters of $A_t$ as output by a localizer network as no knowledge of the actual $X^s$ is available. Conversely, we train our plate localizer over synthetic images for which $X^s$ is known, so we solve for the unknown $A_t$ for a given target $X^t$ for better localization of the plate corners.

## 5.6 Fully convolutional character classification

Each rectified plate patch is independently fed to a fully convolutional implementation of the character network described in section 4.2. The network input window is $24 \times 40$ pixels, so characters sized about $20 \times 30$ pixels, as produced during plate rectification, fit within the input window, including some tolerance margin. The output of the network is a classification response of $24 \times 40$ input windows, spaced by 4 pixels, spanning over the patch (adjacent input windows stride is equal to 4 pixels vertically and horizontally due to the network topology). We define as *charbox* the network response at a given input window position. A charbox defines i) the confidence that the window (entirely) contains each of the considered $C$ character classes and ii) the predicted bounding box (i.e. the top, bottom, right and left position). Thus, adjacent charboxes overlap completely but by 4 pixels, vertically and horizontally. First, we discard charboxes for which the class with the highest confidence rating correspond to $C_{null}$ (the background class) or is below a threshold $\Theta_1^c$.



(a) (b)

Fig. 5.4 Character fully convolutional classification (a) and resulting merged classifications (b).

An example of fully convolutional classification outputs is shown in Fig. 5.4a.

## 5.7 Character merging

Next, redundant boxes are discarded by means of the same iterative merging process as for plateboxes (see section 5.3), until no more charboxes exist with an IoU $> \omega^c$. When boxes to be merged belong to different classes, the class with the highest merging weight (i.e. originating from the most original boxes) is chosen. If the weight is equal the class of the highest confidence is used.

An example of merged fully convolutional character classifications is shown in Fig. 5.4b.

## 5.8 Refine character classification

Once merged, the precision of each charbox classification is refined via a second application of the character detector network. Due to the tolerance margin and window stride, detected characters may be significantly offset from the input window center.

For each charbox, we crop a patch from the rectified plate patch, centered on the charbox central point. The character patch is cropped to include 20% margin relative to the height and width of the charbox and is then resized to 24×40 pixels. A batch of extracted patches, one for each charbox, is fed as input to the character detector network and charboxes for which the highest confidence rating is below a threshold

$\Theta_2^c$ are discarded along with inputs classified as $C_{null}$. As for the refinement of detected plateboxes, this step increases the overall system complexity only marginally whereas it consistently improves the overall performance, as we experimentally show later on.

Table 6.2 compares the character classification with this module active compared to deactivate, as can be seen, this model is critical to an accurate license reading.

## 5.9 Post processing

For each detected plate, the character detector may return a sequence of a variable number of characters. Each character bears a class confidence score and the position within the corresponding rectified plate patch. By assuming this confidence score is a good approximation of the probability if a correct classification we can take a statistical approach to find the most likely license number with the correct amount of characters and with the correct syntax.

Let us assume that the character detector has detected $M$ characters and the expected number of characters is equal to $N$. If $M > N$, up to $M - N$ spurious characters may have been detected in the plate (for example, background clutter may have been interpreted as a character). Namely, we have $M$ *choose* $N$ possible sequences of $N$ characters each, called words in the following. Let us indicate as $w_i$ the $i^{th}$ word of $N$ characters, where $w_i = \{c_{i,1}, \ldots, c_{i,j}, \ldots, c_{i,N}\}$ and $c_{i,j}$ indicates the $j^{th}$ character in the $i^{th}$ word, where the characters are sorted in left-to-right order. Let $P\{c_{i,j}\}$ indicate the maximum probability that $c_{i,j}$ is a valid character, as predicted by the plate detector. Furthermore, let $l_{i,j}$ be 1 if $c_{i,j}$ character is allowed to appear at the $j^{th}$ position in the $i^{th}$ word according to the considered syntax scheme, otherwise 0. The probability that the $i^{th}$ word is correct can be defined as

$$P\{w_i\} = \prod_{j \in 1,N} c_{i,j} l_{i,j}.$$

That is, for each of the $M$ *choose* $N$ words of $N$ characters, we compute the probability that the word is admissible. Next, for each plate we return the most likely word with the relative correctness probability as computed above. Note that for $M = N$, we just have one possible reading. Conversely, if $M < N$, the character detector has

not detected enough characters in the plate and the classification is discarded as incomplete (e.g. due to occlusions in the query image).

Finally, before returning the classifications, any license plate detection without a license reading of the correct syntax is discarded. A final check is also performed to verify that there are not multiple classifications of the same plate; if two classifications of the same license number have been identified, they will be merged if they have any overlap. Note that this should not occur if the plate overlap threshold, $\omega^p$, is properly set.

## 5.10 Final classification output

The final output of the classification pipeline is a set of license numbers with attached place corner coordinates and character coordinates (if any license plates with license numbers of the correct syntax were found). Fig. 5.5 shows an example of classification outputs from an image with multiple plates.


Fig. 5.5 Example of output classification of an image with multiple cars, license plate readings are placed above the image for readability.

# Chapter 6

# Experiments and Results

In this section, we experimentally evaluate a Torch [107] implementation of our LPR system, using two CNN classifiers (described in sections 4.1.5 and 4.2.5) trained on synthetic images and tested in this chapter on real vehicle images.

First we define the classification tasks and their accuracy measurements in section 6.1 and describe the test sets in section 6.2. Then we show how we reliably obtain robust classification thresholds and trained models that work out of the box with the LPR pipeline without further calibration in section 6.3. Then we test the performance on the Italian data set we created in section 6.4 and evaluate the impact of variations of our system, such as training set size and feature composition, the classification thresholds and the non-mandatory parts of the classification pipeline. We also compare our final algorithm to the OpenALPR algorithm [77]. To show robustness in the whole system and that it generalizes to other plate types, in section 6.5 we generate Taiwanese license plates, train new networks and classify a data set of Taiwanese license plates and compare our results to other LPR systems. Finally, we evaluate the complexity and classification time of our system in section 6.6.

## 6.1 Definition of LPR subtasks and corresponding accuracy measurements

We evaluate our system considering three different tasks, defined below along with their performance metrics:

1 The *License Plate Detection* (LPD) task aims at assessing the LPR system performance at localizing plates in the input image. Concerning our proposed pipeline in Fig. 5.1, we consider the plate in an image as correctly detected (i.e., a true positive detection) if it returns a platebox overlapping with the ground truth box by at least 50%. Each detected bounding box overlapping the ground truth by $< 50\%$ is counted as a false positive. If no matching bounding box is returned ($IoU \geq 50\%$), the plate is counted as a false negative. We indicate the number of true positive (TP), false positive (FP) and false negative (FN) detections as $\#TP$, $\#FP$ and $\#FN$ respectively. The LPD performance is measured in terms of $Precision = \#TP/(\#TP+\#FP)$ and $Recall = \#TP/(\#TP+\#FN)$. Note that neither precision nor recall requires a true negative count, but for the sake of completeness: a true negative (TN) sample is every area of the image that is neither classified as a plate nor contain a real plate.

2 The *License Recognition* (LR) task aims at assessing the LPR system performance at character recognition and is further subdivided in two sub-tasks. The *Character Recognition* subtask consists in recognizing each single character in each plate. Considering the Character Recognition subtask, let us indicate the number of true positive character recognitions as $\#TP$ and the number of incorrect classifications (wrong character, missed character or background as a character) as $\#F$. Then, we define the character recognition accuracy as $Accuracy = \#TP/(\#TP+\#F)$. The *License Recognition* subtask consists in recognizing all the characters in each plate. Let us define a true positive if all and only the actual plate characters are correctly recognized, otherwise the recognition is false (F). The overall LR accuracy over a set is defined as $Accuracy = \#TP/(\#TP+\#F)$.

3 The *License Plate Detection and Recognition* (LPDR) task aims at assessing the overall, end-to-end, LPR system performance. For this task, we define a true positive plate detection and recognition as i) the plate has been correctly localized within the image with IoU $> 50\%$ and ii) all and only the characters actually in the plate have been correctly recognized. Each positive classification (i.e. a positive plate classification with a recognized character string of the correct format) such that IoU $< 50\%$ or incorrect license classification is counted as a false positive. For each labelled sample not found (e.g. no LPD detection with sufficient overlap, of due to an incorrect LR detection, i.e. an

FP detection is often coupled with an FN detection), we count it as a false negative classification. LPD is evaluated in terms of Precision and Recall with the same definition as the LPD task (with the TP, FP and FN definitions of this task).

Notice that the LPD and LR task definitions and performance metrics are borrowed from [12, 9], such that the results are comparable, while neither of them report any accuracy of the complete system.

## 6.2 LPR evaluation data sets

We evaluate our system on two data sets, a set of Italian plates described in section 6.2.1 and a set of Taiwanese plates proposed in [12] and described in section 6.2.2. The Italian set was used to evaluate the earlier stages of LPR system and help us understand where improvements were needed. Although we tried to avoid biasing the algorithm towards this set, the critical reader may consider this a validation set rather than a test set. The Taiwanese data set however, was introduced after the Italian version of the system was completed and the system was only modified to accommodate for differences between the plate types, such as plate dimensions and number of characters present on the plates.

### 6.2.1 PlatesMania dataset of real Italian license plates

This database includes images of different types of vehicles captured with smartphones and digital cameras that users have uploaded to *platesMania.com*, a website collecting vehicle license plates from all over the world. For our experiments, we downloaded 1153 images of vehicles captured under a wide range of conditions of light and perspective, a few such examples are shown in Fig. 6.1. The images are already annotated with the license number by the uploaders but we performed the license plate corner annotations ourselves, the character corners are not annotated and thus the character localization accuracy is not measured (although it indirectly affects the character classification accuracy through more accurate merging and re-classification, see sections 5.8-5.7).

Fig. 6.1 A few samples from the *PlatesMania.com* dataset. Original resolution is 0.3 - 3 Mpixels.

### 6.2.2 AOLP dataset of real Taiwanese license plates

This dataset was introduced by Hsu et al. in [12]; the database contains 2049 Taiwanese vehicle license plate images of which 1891 are annotated with license strings and plate positions of one or more cars (we use the annotated part of the dataset). The database is subdivided into three application-oriented subsets; Access control (AC), Traffic Law Enforcement (LE) and Road Patrol (RP); Fig. 6.2 show one example of each. The AC subset contains small images of front plates, acquired from slightly above but with little horizontal angle (e.g. by a camera mounted above an access gate). The LE subset contains larger images of vehicles in traffic, acquired with a large horizontal angle (e.g. by a roadside camera). Finally the RP subset contains images acquired from a patrolling vehicle (e.g. a mounted or hand held camera).

Fig. 6.2 *Application Oriented License Plate* (AOLP) database samples: (left) Access Control subset, (center) Traffic Law Enforcement subset, (right) Road Patrol subset. Original Resolution: 0.08 - 0.3 Mpixels.

## 6.3 Classification thresholds and training time

Throughout the development process we initially had difficulties in consistently finding a good set of parameters for a trained model. With empirical testing, sometimes models at only 75 epochs (of 80k training samples) worked well, and other times the thresholds would need to be adapted based on observed errors on the test set (which would rather make it a validation set), or a longer training time was needed.

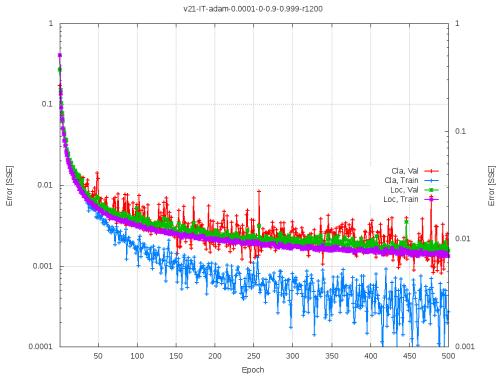
Given a validation set of real, labelled samples, calibrating the thresholds and testing multiple versions at different epochs can certainly improve the performance. However, since validation sets of real samples may not be available (e.g. because collecting and labelling samples for multiple plate types would be very laborsome), we needed to find a robust method to obtain our trained models as well as robust classification parameters that works out of the box with newly trained models without calibration.

Our approach to find this method was more empirical in practice, the following analysis of a training process of a plate network is rather meant to show why it works, and uses a set of crops extracted from the Italian *PlatesMania.com* data set in the same manner as the training samples were extracted (see section 4.1.4) with 50% positive samples and 50% negative samples with a chance to include up to 50% of a plate.

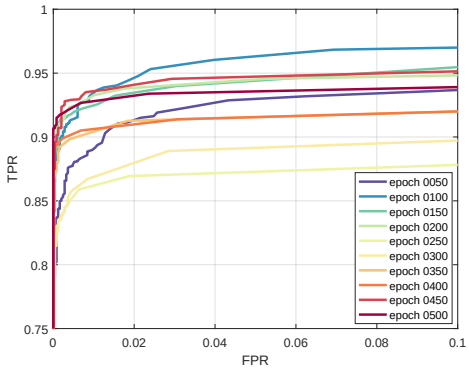
### 6.3.1 Selecting plate training parameters

The training and validation prediction errors (on a synthetic validation set) during the training process over 500 epochs are shown in Fig. 6.3a (this process takes about 1 day on a TITAN X Pascal GPU). As can be seen the error is still slowly decreasing at the end of the training process, without any clear signs of overfitting. The training process saves the parameters of every $50^{\text{th}}$ training epoch, which are further analysed with the data set just described above. This is done with more classification near performance measurements, namely Receiver Operating Characteristic curves (ROC curve) showing for the classification and IoU for the localization.

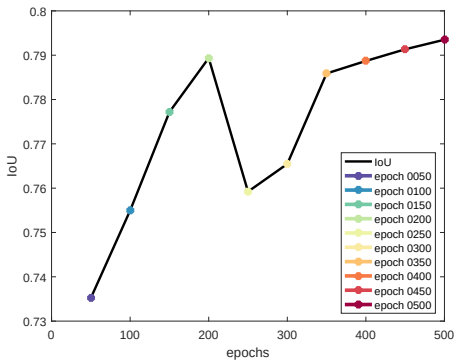
The ROC curve is useful to illustrate the trade-off between precision and recall by plotting True Positive Rate (TPR) versus the False Positive Rate (FPR), while the



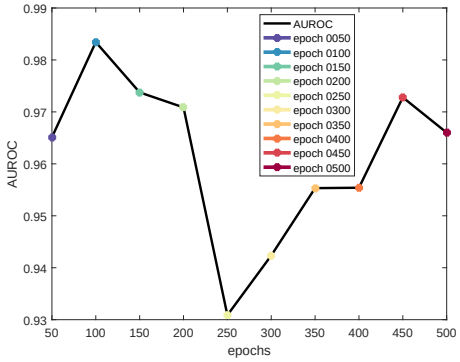

(a) Classification and localization prediction errors for each training epoch.




(b) ROC curves




(c) Intersection over union




(d) Area under ROC curve

Fig. 6.3 Classification and localization prediction error.

area under this curve is a measurement of the overall performance before choosing a threshold and thereby the trade-off point.

Fig. 6.3b show the ROC curves for every 50$^{th}$ epoch and Fig. 6.3d shows the Area Under ROC (AUROC) with the data points color coded as the corresponding ROC curves, and Fig. 6.3c shows the location accuracy in terms of IoU for the same epochs. As can be seen in Fig. 6.3d and 6.3c, the classification performance on the dataset of real samples is initially good before taking a dip around 250 epochs and recovering towards the end. However, the localization accuracy is not stable until epochs 350-500.

These observations explain why some models trained a shorter time than the 500 epochs can perform well (e.g. epoch 200 in this example) but the results of earlier epochs are unstable. A validation set of real samples can be used to stop early, as well as to calibrate the classifier threshold to achieve desired balance between

precision and recall from the ROC curve. Another effect of letting the training process go on longer is that the model better learns the classification threshold since the classification confidences are being pushed closer and closer towards a binary classification the longer the training process goes on. Fig. 6.4 illustrate this effect by showing the location of some fixed threshold on the ROC curves of epoch 50 vs 500.

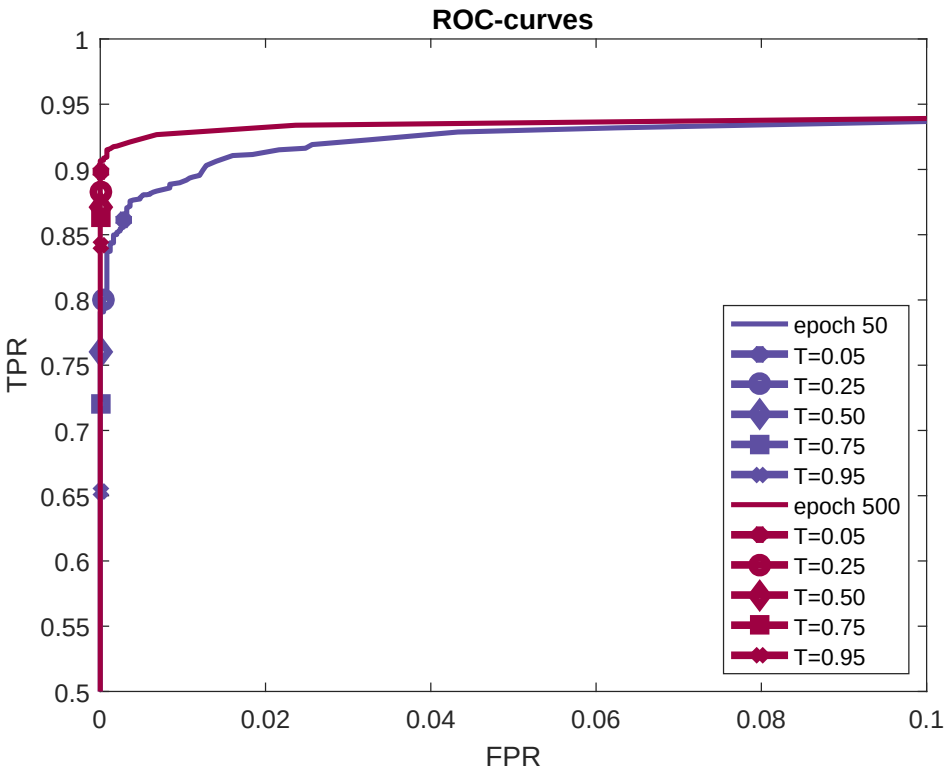



Fig. 6.4 ROC curves of epoch 50 and 500 illustrating the concentration difference of the thresholds 0.05, 0.25, 0.50, 0.75 and 0.95 on the two curves. An early epoch requires the threshold to be carefully chosen to perform optimally, while a later epoch is more robust to the threshold choice, although it is not optimal.

As can be seen in Fig. 6.4, earlier epochs are more flexible because they have not yet learnt the classification threshold, allowing (and requiring) the user to choose that after training. However, as explained at the beginning of this section, we want to avoid such calibration since a validation set of real samples is not always available, therefore we prefer to train the network for longer so that it becomes almost binary and the classification more robust.

Hence, by training the network for as long as 500 epochs, although early stopping *may* achieve a similarly performing model, we noticed that we can reliably train

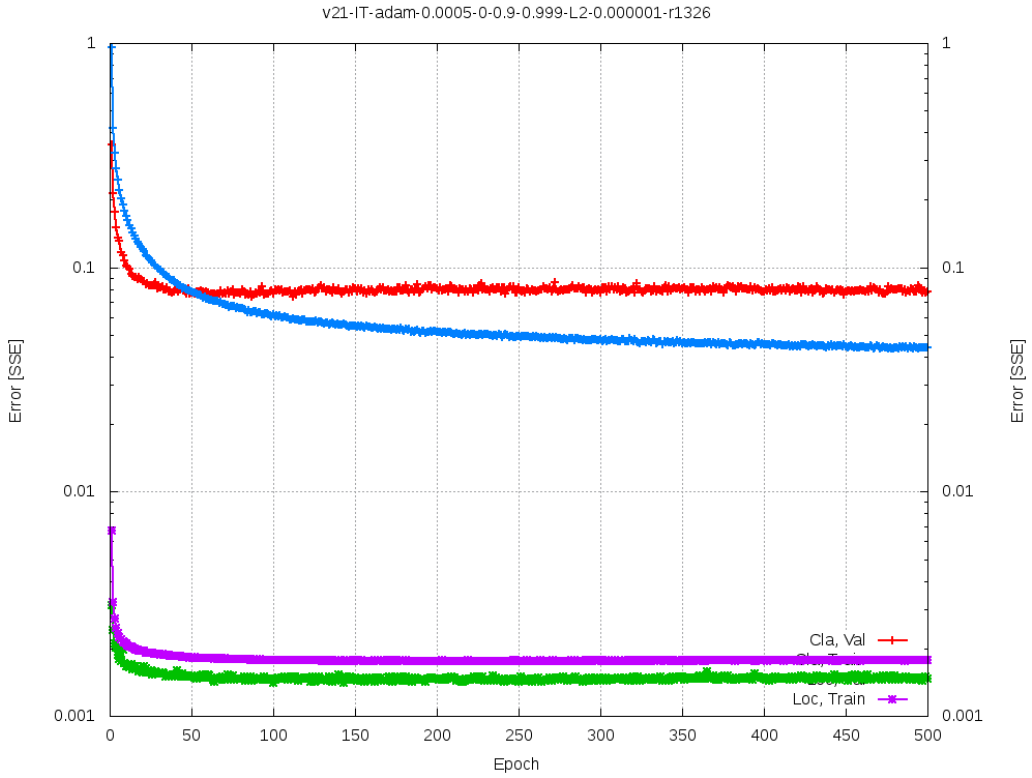



Fig. 6.5 Character classification and localization prediction errors for each training epoch.

multiple models that perform well, without any need of calibration and simply using the default classification threshold of 0.5 for both fully convolutional classification and the refinement classification, i.e. $\Theta_1^p = \Theta_2^p = 0.5$.

### 6.3.2 Selecting character training parameters

Similarly for the character classifier, by training it for a longer time than necessary we found that we reliable achieve a robust classifier that works well with $\Theta_1^c = \Theta_2^c = 0.5$. Although, because there are significantly more character samples per epoch, it converges faster than the plate classifier and 200 epochs are enough. Fig. 6.5) shows the training and validation prediction errors of the character classifier. Unfortunately we can not show the same analysis of real samples for the character classifier, because we do not have any annotated corner location of the real character samples which are required to accurately extract the samples (and evaluate the localization performance).

### 6.3.3 Selecting overlap thresholds for merging

To further avoid biasing our method towards a calibration set, and to show robustness of the algorithm, we will also use 0.5 as merge thresholds to merge fully convolutional plate classifications (see section 5.3) and character classifications (see section 5.7), i.e. $\omega^p = \omega^c = 0.5$. We consider as 0.5 a default threshold because an overlap of 50% is commonly considered sufficient for a localization prediction agreeing with the ground truth [108].

However, we will also evaluate the effect of changing these thresholds (as well as the plate and character classification thresholds) in section 6.4.3.

## 6.4 Experiments on the Italian PlatesMania data set

The robust parameters from section 6.3 allow us to train multiple models to evaluate the impact of variations of the system by observing how the classification performance on the Italian data set is affected. Here we investigate the impact of different design choices of the synthetic sample generation, the training process and classification pipeline on the classification output. Specifically, in section 6.4.1 we evaluate the effect of varying the training set size and in section 6.4.2 we evaluate the effect of some non-mandatory feature variations added to synthetic samples. In section 6.4.3 we vary the classification and merging thresholds of the classification pipeline to see how they impact the classification output and in section 6.4.4 we disable the non-mandatory stages of the pipeline to reveal how much their presence improves the classification performance.

### 6.4.1 Dataset size evaluation

To evaluate the number of synthetic training samples needed to train our classifiers, we generated 80k training samples (during the development process we mainly used 40k samples) and created subsets of 10k, 20k, 40k, 60k and all 80k unique samples to train plate and character networks on. The number of epochs was adjusted to compensate for the amount of samples per epoch, such that each network would receive an equal amount of backpropagations during the full training process. Fig. 6.6 shows the end-to-end LPDR performance of the classification pipeline in terms

of precision and recall when using plate and character classifiers trained on different amounts of unique training samples.

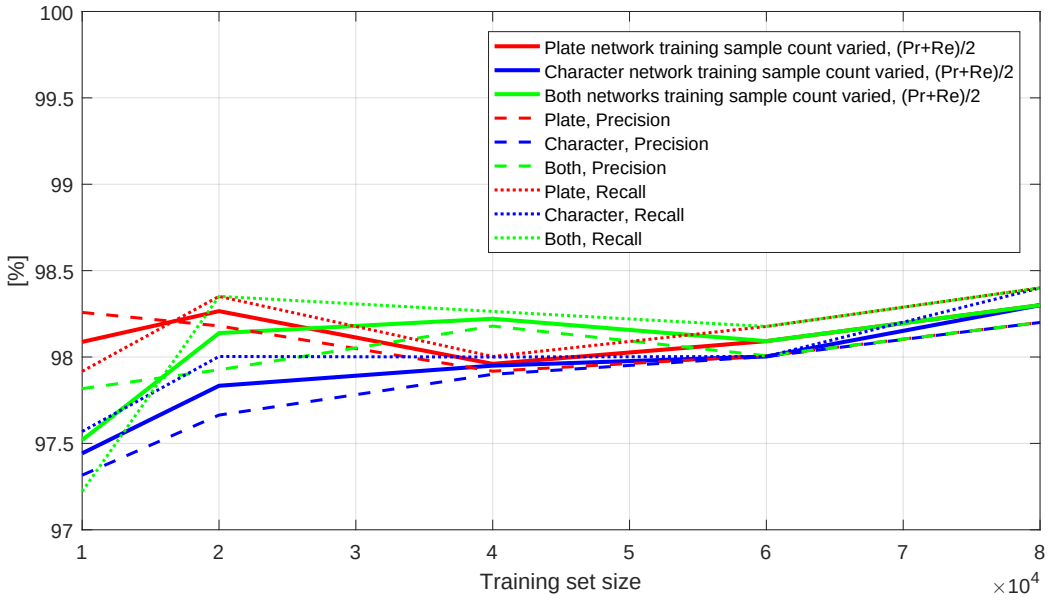



Fig. 6.6 License plate detection and recognition precision and recall as a function of the number of training samples in the training set in the plate (red), character (blue) and both (green) classifiers. Full lines indicate the average of precision and recall, dashed lines precision and dotted lines recall.

Two trends are rather clear; i) the performance when increasing the number of samples for the character network (blue) is steadily increasing, and ii) the difference between 10k and 20k samples is significant for all cases. However, the results are not very conclusive, e.g. why is the performance with fewer samples for both classifiers (green) better than when one of the classifiers use all samples (blue and red)? Likely there is too much variance when only one trained network is used for each data point. However, because of the amount of computation time already required to train multiple networks for this plot, we decided that effort was better spent elsewhere.

Based on these results we decided to continue using 80k samples, as that data point shows the best results. Increasing the data set size more would perhaps be beneficial to, but as the storage size and generation time of 80k samples was already cumbersome, we decided to settle for 80k samples also for the Taiwanese training set generated in section 6.5.

### 6.4.2 Synthetic database evaluation

Since 40k samples is sufficient too, we can divide the dataset of 80k training samples into subsets where some non-mandatory features that activate with 50% probability are consistently either all activated within the same subset or all deactivated, to evaluate the usefulness of these features.

Note that this is an evaluation of the final synthetic sample generation algorithm and not the process used to select which features to include. The design decisions regarding which features to use in the modules was a continuous process in parallel with the development of the classification process, after reviewing the errors made, which sometime required improvement of the sample generation and sometimes improvements of the classification process, rendering previous comparisons outdated. However, because the random activation of features added later are all only activated 50% of the time (to ensure that the variance of features in the previous was maintained, while adding more variance), the database can be divided into subsets within which such a feature is either always active or always inactive, to compare whether it was useful or not.

Therefore, as a final evaluation to verify that the features were still useful on the final classification and training algorithms, multiple plate and character networks were trained on these subsets and compared to the final version. The results are shown in Table 6.1, where the number of training epochs was doubled for the subsets compared to the reference set to compensate for the number of samples being halved.

As can be seen, the one feature standing out is *text thickness* (see section 3.2.1), which improves the plate localization when active while being almost on par with the reference on the other scores. This indicates that it certainly was a good addition, in fact so good that it might be better to keep it on all the time instead of only 50%. The other features generally show slight improvements on the sets where they are active compared with inactive, however, the combination of both is still better (i.e. the reference).

Overall these results gave no strong indications that the database could be improved by limiting the full database to the features of one of these subsets. Therefore the algorithm was not changed before being used to generate a Taiwanese database to train classifiers to be used on the final evaluation of our algorithm on the test set of Taiwanese license plates, also used in [12] and [9].

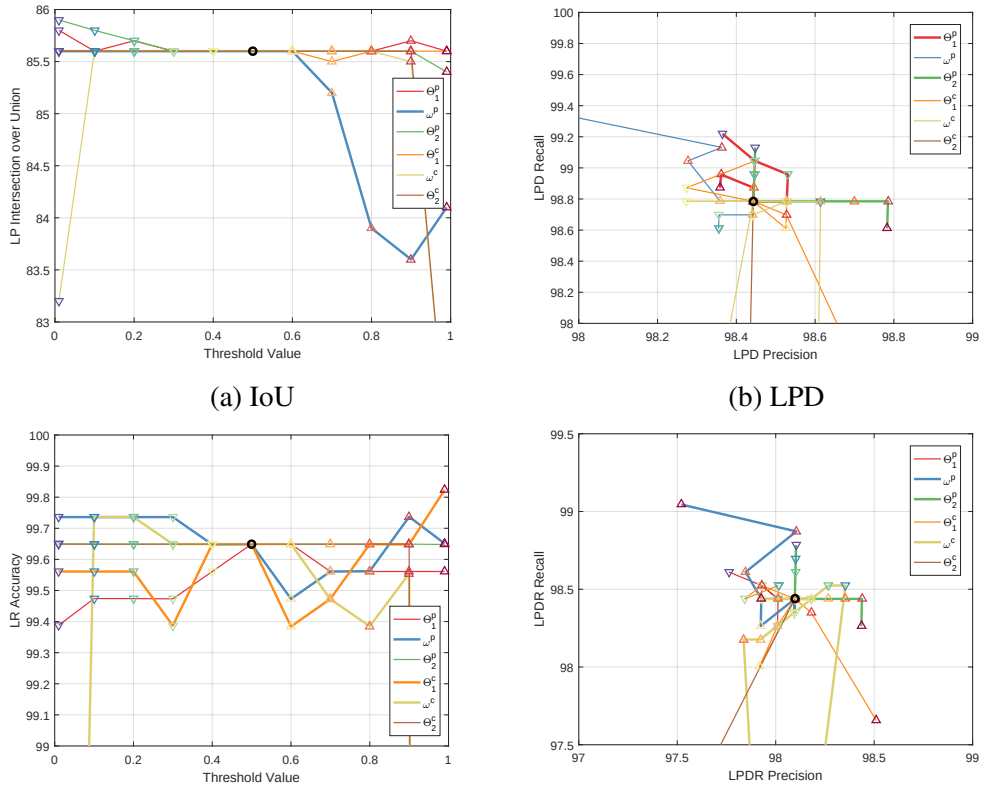
Table 6.1 Classification and localization accuracy with training sets feature-divided in subsets.

| | Plate | | | Character |
|---|---|---|---|---|
| Subset | Prec. [%] | Recall [%] | IoU [%] | Accuracy [%] |
| Text Shadow 3.2.1 | 99,0 | 98,0 | 85,5 | 99,5 |
| Text Shadow inactive | 98,9 | 98,2 | 85,5 | 99,5 |
| Text Reflection 3.2.1 | 98,9 | 98,0 | 85,4 | 99,3 |
| Text Reflection inactive | 98,6 | 97,9 | 85,8 | 99,4 |
| Text Thickness 3.2.1 | **99,0** | **98,4** | **86,4** | 99,4 |
| Text Thickness inactive | 98,4 | 98,1 | 85,4 | 99,5 |
| Plate Occlusion 3.2.3 | 98,7 | 98,3 | 84,4 | 99,4 |
| Plate Occlusion inactive | 98,5 | 98,3 | 83,8 | 99,3 |
| Plate Shadow 3.2.3 | 98,6 | 98,3 | 85,2 | 99,5 |
| Plate Shadow inactive | 98,3 | 98,2 | 84,2 | 99,5 |
| Bad Focus 3.2.7 | 98,6 | 97,9 | 84,8 | 99,5 |
| Bad Focus inactive | 98,9 | 98,1 | 85,5 | 99,5 |
| Low Light 3.2.7 | 98,9 | 98,2 | 85,6 | **99,6** |
| Low Light inactive | 98,5 | 98,1 | 86,0 | 99,4 |
| Full Database | **99,0** | **98,4** | 85,0 | **99,6** |

### 6.4.3 Evaluation of pipeline classification thresholds

To evaluate the effects of the classification thresholds, $\Theta_1^p$, $\omega^p$, $\Theta_2^p$, $\Theta_1^c$, $\omega^c$, $\Theta_1^c$ (in order of application) they were varied from 0.01 to 0.99 one by one while classifying the *PlatesMania.com* data set and observing the performance metrics defined in section 6.1.

We begin analyzing the metrics one by one, comparing all thresholds together and observing which threshold values are outstanding for each metric (after this we will also analyze the thresholds one by one to identify which metrics they affect). Fig. 6.7 shows all the metrics of all the thresholds, with the point of all default thresholds marked as a black circle, while the other threshold points are marked with triangles pointing upwards for higher thresholds and downwards for lower thresholds and rainbow-color coded according to their values. The thresholds standing out which are discussed below are indicated with thicker lines in the relevant plots.




(a) IoU (b) LPD

(c) LR (d) LPDR

Fig. 6.7 Accuracy of the license plate recognition tasks with varied thresholds $\Theta_1^p$, $\omega^p$, $\Theta_2^p$, $\Theta_1^c$, $\omega^c$ and $\Theta_1^c$. The black circle indicate all thresholds set to the default value of 0.5, upwards pointing triangles indicate the threshold is increased from the default value of 0.5 while downwards pointy triangles indicate decreased value, while color coding of the triangles indicate to which degree.

### License plate detection

For the LPD task (see Fig. 6.7b) the default thresholds are not optimal. Decreasing the plate classification, $\Theta_1^p$ (red line), can improve both precision and recall, while increasing the plate reclassification threshold, $\Theta_2^p$ (green line), increases the precision without a loss of recall while decreasing it increases the recall without a loss of precision.

The plate localization accuracy in terms of IoU (see Fig. 6.7a) is mostly unaffected by all threshold changes except for the merge overlap threshold, $\omega^p$ (blue line, which is indeed the most expected parameter to affect the plate localization), choosing $\omega^p < 0.6$ appears to have a negative impact. From observations of classifications, this can sometimes cause the fully convolutional classifications not to merge fully, resulting in two clusters merged to opposite ends of the plate, so this result is expected.

### License plate reading

For the LPR task (see Fig. 6.7c) the default thresholds are close to optimal. It appears that bringing the character classification threshold, $\Theta_1^c$ (orange line) up as high as 0.99 has a positive impact, given that the character accuracy is in the range of 99% it makes sense that the prediction scores of accurate classifications are driven that high and such a high threshold may reject more negative samples than it fails to detect correct classifications. However, note that the LPR accuracy is already so high that the difference is only on two samples in the entire data set.

Both overlap thresholds for merging, $\omega^p$ and $\omega^c$ (blue and yellow lines), also show improvements at some values, but on a single sample compared to the default thresholds.

### License plate detection and reading

For the combined task, LPDR (see Fig. 6.7d), the default thresholds are average, the improvements from LPD of the $\Theta_2^p$ (green line) carries well over to the LPDR task, as does the improvement from a low $\omega^c$ from the LPR task. Notable is also the increased recall from $\omega^p \geq 0.9$, despite the low IoU mentioned previously. This configuration causes the plate classification stage to not merge detections effectively,

forwarding multiple localization guesses to the character classifier, resulting in multiple character classifications for each license plate. This configuration is not recommended because it has a very negative impact on both classification time and memory requirements.

### Plate classification thresholds

Fig. 6.8 shows the plate classifier thresholds from Fig. 6.7 isolated for a less cluttered view to easier analyze the effects of changing these thresholds.

$\Theta_1^p$ (red line) has no clear tendencies; while the LPR precision and recall appears to both improve for carefully selected lower values, this does not appear to carry over to the overall results because this also causes the LR accuracy to decrease, furthermore the changes are small and may be random variations.

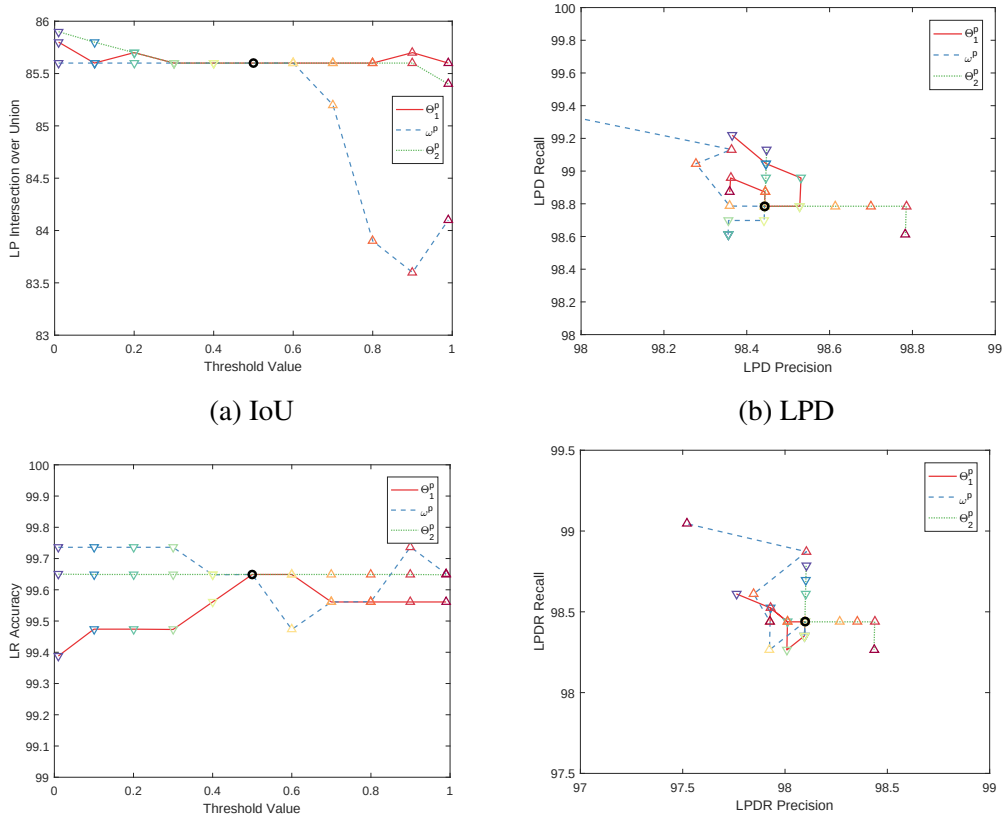
As explained above, increasing $\omega^p$ (blue dashed line) causes less accurate plate localization because the merging is not complete; this has a positive effect on recall because merging is more accurate after the character classification has been performed compared to before. However, it causes multiple character classifications of the same plate and unnecessary repeated classifications of the same object. If only recall is relevant, this configuration is a good option, as it increases LR accuracy as well as LPD and LPDR recall, at the cost of precision and processing time.

$\Theta_2^p$ (green dotted line) offers a trade-off between LPD and LPDR precision and recall, which should come as no surprise since a high threshold is more likely to reject false plate detections (higher precision) while a lower threshold is more likely to accept plate detections (higher recall).

### Character classification thresholds

Fig. 6.9 shows the character classifier thresholds from Fig. 6.7 isolated to analyze the effects of these thresholds.

$\Theta_1^c$ (orange line) naturally mainly has an impacts on the license reading, but due to the rejection of plate detections when the predicted license does not match the required syntax, it also affects the precision-recall trade-off for plates as well as the complete LPDR prediction. A low threshold value increases the acceptance rate of characters, which can lead to license predictions that are incorrect but of the right




(a) IoU (b) LPD

(c) LR (d) LPDR

Fig. 6.8 Accuracy of the license plate recognition tasks with the plate classification thresholds, $\Theta_1^p$, $\omega^p$ and $\Theta_1^p$ varied. The black circle indicate all thresholds set to the default value of 0.5, upwards pointing triangles indicate the threshold is increased from the default value of 0.5 while downwards pointy triangles indicate decreased value, while color coding of the triangles indicate to which degree.

syntax, thereby accepting the correct LPD while erring on the LPDR task (compare the recall gain in Fig. 6.9b and Fig. 6.9d). A high threshold value on the other hand will prefer to reject characters over making a wrong prediction, causing an incorrect syntax and a rejection of the plate detection as well.

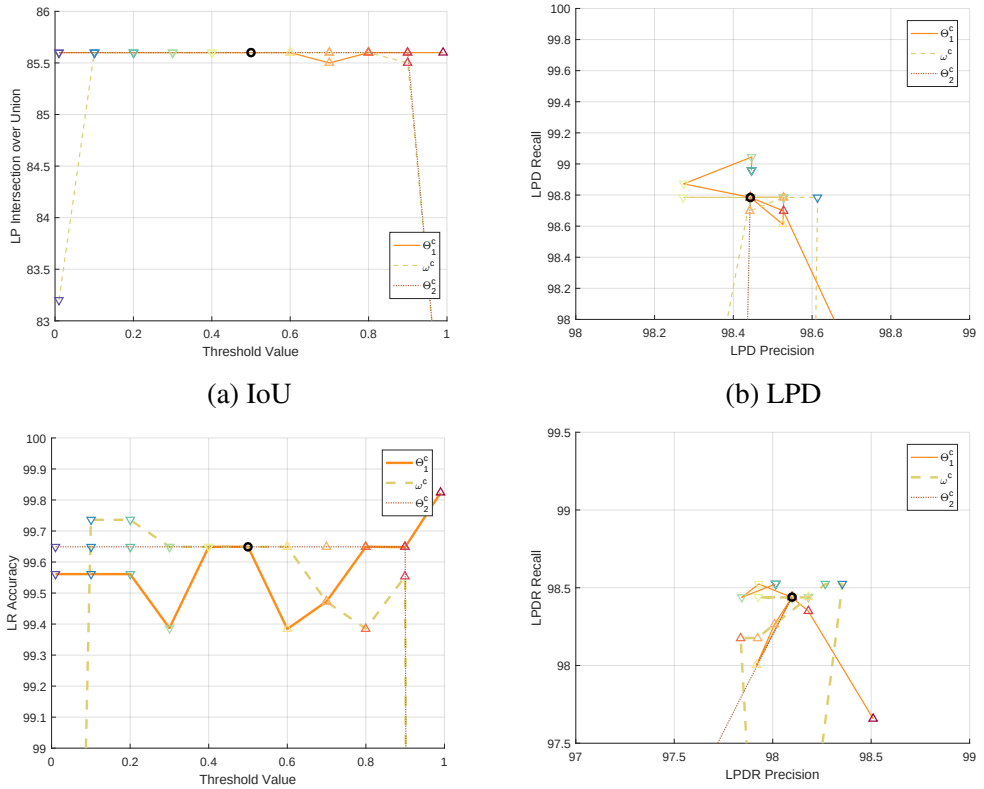
$\omega^c$ (yellow dashed line) has no clear trade-offs; it appears that lower values work better than the default by a slight increase in LR accuracy which directly leads to increased precision of the LPDR precision but also the LPD precision since correct predictions naturally have the correct syntax as well. However, when the threshold is pushed to either extreme (0.01 or 0.99) the character prediction completely collapses.

$\Theta_2^c$ (brown dotted line) has little to no impact unless pushed to the upper extreme at $\Theta_2^c = 0.99$, where it causes the character classification to completely collapse; at low thresholds it still works because when multiple predictions are above the threshold, only the highest prediction is chosen (and essentially rising the actual threshold for the other predictions). In fact, the results from all the other variations of $\Theta_2^c$ are hidden behind the black circle of the default threshold because they all result in identical predictions, from $\Theta_2^c = 0.01$ to $\Theta_2^c = 0.9$.

### 6.4.4 Evaluation of classification pipeline components

In this experiment we are interested in assessing how the non-mandatory components of our LPR system in chapter 5 contribute to the system overall performance. Namely, we are interested in assessing how plate localization refinement, plate rectification, character classification refinement and the error correction components (see Fig. 5.1) affect the overall system performance. Table 6.2 shows our LPR system performance for the three experimental tasks defined in section 6.1.

The *Full system* reference scheme corresponds to our full-fledged LPR system as described in chapter 5. The four following schemes represent the cases where plate classification and localization refinement, plate rectification, character classification refinement and error correction 5.9 have been excluded respectively. Note that we have also disabled LPR feedback on the LPD task in this case to better single out the classification results from the individual classifiers, normally plates classifications with a license reading not matching the correct syntax are eliminated. This feedback is re-enabled for the final experiments in section 6.4.5.




(a) IoU (b) LPD

(c) LR (d) LPDR

Fig. 6.9 Accuracy of the license plate recognition tasks with the character classification thresholds, $\Theta_1^c$, $\omega^c$ and $\Theta_1^c$ varied. The black circle indicate all thresholds set to the default value of 0.5, upwards pointing triangles indicate the threshold is increased from the default value of 0.5 while downwards pointy triangles indicate decreased value, their color coding is matched between the sub-figures such that the vertical axis on (a) or (c) can be used to indicate the threshold values in (b) and (d).

Table 6.2 LPR accuracy for Italian plates with pipeline modules disabled [%].

| Task | LPD | | | LR | | LPDR | |
|---|---|---|---|---|---|---|---|
| Scheme | Prec. | Recall | IoU | Char. | Lic. | Prec. | Recall |
| Full system | 93.76* | 99.22* | 85.56 | **99.81** | **99.48** | 98.87 | **98.70** |
| No plate refinement | 69.61* (-24.1) | 96.03* (-3.19) | 75.37 (-10.2) | 99.11 (-0.70) | 98.81 (-0.67) | 93.97 (-4.90) | 94.80 (-3.90) |
| No plate rectification | 93.76* | 99.22* | 85.56 | 97.44 (-2.37) | 92.36 (-7.12) | 96.69 (-2.18) | 91.23 (-7.47) |
| No character refinement | 93.76* | 99.22* | 85.56 | 85.67 (-14.1) | 58.45 (-41.0) | 54.28 (-44.6) | 52.26 (-46.4) |
| No error-correction | 93.76* | 99.22* | 85.56 | 99.80 (-0.01) | 99.30 (-0.18) | **99.82 (+0.95)** | 98.52 (-0.18) |

*includes classifications rejected by post-processing

Note that this is not how the the algorithm is intended to run; the plate network is not intended to be used alone for plate detection, it is designed to have high recall at almost any cost of precision. Even if the characters do not need to be correctly interpreted for the license plate detection task, verifying that a potential plate contains characters is a crucial part. A high recall is necessary throughout the pipeline to avoid that errors accumulate; a false positive classification can be corrected by rejecting a candidate later on, but a missed detection can never be recovered.

When plate rectification is disabled, detected plates are still rescaled, this part is considered mandatory since the character network is not designed to be scale independent. The character network is also re-trained without applying the un-rotation described in section 4.2.4, to account for the fact that the input from the pipeline will also remain rotated.

As can be seen in Table 6.2 both the plate and character refinement components are critical to their corresponding task. Plate rectification also improves the character recognition significantly by providing more consistent input to it. However, disabling the error correction step shows a significant improvement on LPDR precision (+0.95pp), for a relatively small cost in LPDR recall (-0.18pp). For this reason we choose to leave the error correction off for all further experiments.

Error correction is not working as well as expected because the dataset contains un-annotated plates, often of other formats from different countries that are still

detected by the plate detector. When error correction is deactivated, such plate detections are correctly discarded because the syntax of the detected license number does not match. However, with error correction enabled, it tries to find the most likely Italian license string instead, resulting in an FP classification. If classifications of plates that are not annotated were ignored instead of counted as FP when they are incorrect, the full system with error correction enabled indeed performs the best, with an LPDR precision and recall at 99.74% and 98.70% respectively.

### Why refinement classifications improve the prediction accuracy

The reason why we chose to implement the refinement classification was an observation that off-centered localizations were often less accurate than well centered ones. Further experiments show this is also true for the classification accuracy as well as the scale of the objects. Fig. 6.10 illustrates how the classification detection rate and localization error depend on the object location within the classification window (left) and the scale of the plate (right).

The fully convolutional classifier will typically cover the object with at least 4 shifted windows, often more. Since the plate will not be optimally located in most of those, neither the average classification nor localization will be optimal by averaging the positive classifications.

As long as the object is fully inside the classification window this effect is small. Therefore an alternative refinement method to the re-classification could be to estimate which of the contributing predictions did not view the full object, based on the merged plate location, and remove them from the averaging. However, this may be inaccurate because such predictions will estimate that the object are inside their classification window, which may cause the average location prediction to erroneously be within the window and thus not eliminating the faulty classification.

Table 6.3 LPR accuracy for Italian plates [%].

| Task | LPD | | LR | | LPDR | |
|---|---|---|---|---|---|---|
| Scheme | Prec. | Recall | Char. | Lic. | Prec. | Recall |
| Proposed | 100 | 99.21 | 99.80 | 99.30 | 99.82 | 98.52 |
| Full system | 98.97 | 99.22 | 99.81 | 99.48 | 98.87 | 98.70 |
| Calibrated | 100 | 99.21 | 99.86 | 99.39 | 99.82 | 98.61 |
| OpenALPR | 100 | 99.13 | 99.88 | 99.21 | 99.64 | 98.35 |

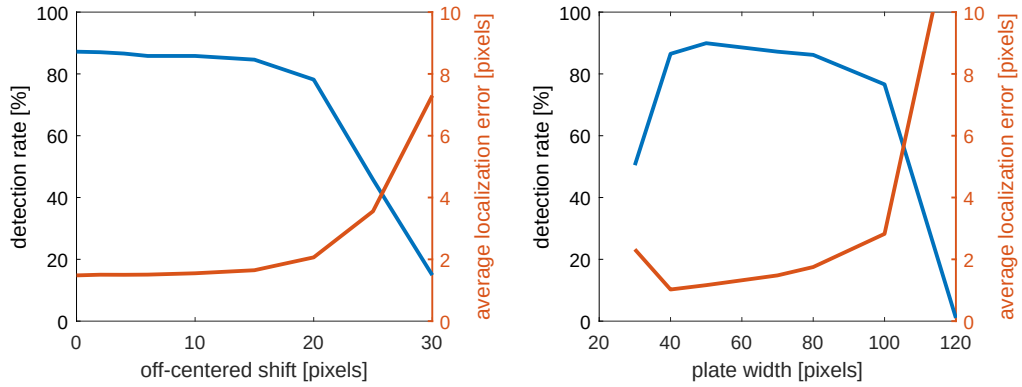



Fig. 6.10 (a) Prediction accuracy of off-centered classifications. Detection rate (blue) and average localization error (orange) as the plate is shifted off the center of the classification view. (b) Prediction accuracy of scaled classifications. Detection rate (blue) and average localization error (orange) as the plate width is increased.

### 6.4.5 Evaluation of the full pipeline

Finally we evaluate the proposed pipeline in terms of LPD, LR and LPDR, and compare it to the full system, the calibrated system and the OpenALPR software [77], the results are shown in Table 6.3. "Proposed" uses the robust classification thresholds for the pipeline (i.e. $\Theta_1^p = \Theta_2^p = \Theta_1^c = \Theta_2^c = \omega^p = \omega^c = 0.5$) with error correction disabled. "Full system" also uses the robust thresholds but with error correction enabled. "Calibrated" uses the calibrated thresholds ( $\Theta_1^p = \Theta_2^c = \Theta_1^c = 0.5$, $\Theta_2^p = \omega^p = 0.9$ and $\omega^c = 0.1$) from section 6.3 with error correction disabled. "OpenALPR" uses the OpenALPR cloud service set to recognize EU-plates.

As can be seen in Table 6.3 the proposed system performs overall (LPDR) better than the full system (as already discussed in section 6.4.4 and slightly better than OpenALPR in both precision and recall. The calibrated system has slightly higher

recall but is calibrated on the test set itself and we can therefore not guarantee that the improvement generalizes to another set. The reason OpenALPR shows a higher character accuracy, yet a lower license reading accuracy may be related to a stricter plate rejection policy than our algorithm; both algorithms reject potential plate detections with the help of character classification, but their method may be stricter, causing a few character misclassifications to count as false negative plate predictions instead of incorrect LR classifications and a different trade-off between LPD recall and character accuracy than our algorithm and may be the reason for the slightly lower LPD recall of the OpenALPR system. This rejection is also the reason why both algorithms can achieve 100% LPD precision, i.e. 0 false positive plate detections, it is highly unlikely that other objects both get classified as potential plates by the plate detector and the character detector finds enough characters within the region.

## 6.5 Experiments on Taiwanese Plates

To classify Taiwanese plates, we generated a new synthetic training set following the same procedure described in chapter 3 for generating the Italian synthetic plate images, with the template generation module from section 3.2.1 modified to produce Taiwanese plates, as described below in section 6.5.1.

Hence, this experiment does not only evaluate the performance of the classification pipeline on Taiwanese vehicle plates from the *Application Oriented License Plate* data set (introduced in section 6.2.2). It tests the whole system, including sample generation and training, also putting the parameter settings to test (which were decided by the experiments in sections 6.3 and 6.4.1).

### 6.5.1 Generation of the synthetic Taiwanese LPR training set

The basic appearance of an Taiwanese license plates of the AOLP data set is shown in Fig 6.11. The license format is: **XX-DDDD** or **DDDD-XX**, where **D** indicate a digit from 0 to 9 and the **X**:es indicate letters or digits, at least one of which is always a letter. The letters go from A to Z with I and O excluded. Three Chinese characters may be located at the top but are non-mandatory features, the four black

bars towards the corners are holes for mounting, which we represent in the alpha channel for transparency.


Fig. 6.11 Taiwanese license plate.

With these differences applied to the plate template module from section 3.2.1 we leave the synthetic sample generation algorithm otherwise unchanged and generate a training set of 80k samples. An example is shown in Fig. 6.12.

Fig. 6.12 Taiwanese synthetic training image (left) with corresponding plate training crops (positive: top center, negative: top right) and character training crops (positive: bottom center, negative: bottom right).

### 6.5.2 Training the Taiwanese classifiers

The training process of the plate classifier is unchanged with respect to the Italian plate classifier. The character classifier is modified to have $C = 35$ output because the Taiwanese plates allow two more characters than the Italian plate format (U and Q). However, due to the disproportionately small fraction of letters on the plates, the data set becomes significantly larger after the character counts have been equalized (as described in section 4.2.5). Fig. 6.12 shows some examples of extracted plate and character samples from a synthetic image.

Table 6.4 License plate detection precision (PR) and recall (RE) on the AOLP data set. A classification is considered positive if IoU$>$ 50% with respect to the ground truth.

| Subset | AC | | LE | | RP | |
|---|---|---|---|---|---|---|
| Method | PR [%] | RE [%] | PR [%] | RE [%] | PR [%] | RE [%] |
| Hsu et al. [12] | 91 | 96 | 91 | 95 | 91 | 94 |
| Li et al. [9] | 98.5 | 98.4 | 97.8 | 97.6 | 95.3 | 95.6 |
| Proposed | **100** | **99.85** | **99.84** | **99.33** | **99.84** | **100** |

### 6.5.3 Classification of the AOLP data set

To classify the AOLP dataset, the plate network model after 500 epochs of training and the character network after 200 epochs are used, and the classification and merging thresholds are all set to 0.5 (i.e. $\Theta_1^p = \Theta_2^p = \Theta_1^c = \Theta_2^c = \omega^p = \omega^c = 0.5$), as suggested after the experiments in section 6.3. The classifications are compared to the ground truth labels and evaluated according to the three subtasks defined in section 6.1 and discussed in detail below.

#### License plate detection task

Table 6.4 shows the performance of our system on the AOLP dataset in the LPD task and compares it with [12] and [9]. Our proposed approach outperforms the approach of [12], where the plate location is determined via as single CNN, for every AOLP subset. Interestingly, our approach also outperforms that of [9], which relies on a double plate detection approach where the characters are first detected and then the plate is localized based on the character detector output. The high precision values reported in the table are confirmed by the actual measured IoU, which is equal to 85% on average over the three AOLP subsets.

#### License recognition task

Table 6.5 shows the performance of our proposed system over the AOLP dataset in the LR task and its two character recognition (Char) and license recognition (Lic) subtasks. The results of [12] and [9] are extracted from Table 8 of [9]. Our character detector CNN outperforms both the character detector of [9] based on a CNN and the character detector of [9] based on a LSTM recurrent network. The

reason for the superior performance of our system is the improved ability to correctly classify characters by refinement of already classified characters, as our experiments in section 6.5 revealed also for Italian plates. We recall that character classification refinement in our proposed system yields a negligible complexity increase as it does not entail a fully convolutional character search in the query image.

Table 6.5 Comparison of single character recognition accuracy (CH) and full license recognition accuracy (LIC) on correctly localized plates from the three subsets of the AOLP dataset.

| Subset | AC | | LE | | RP | |
|---|---|---|---|---|---|---|
| Method | CH [%] | LIC [%] | CH [%] | LIC [%] | CH [%] | LIC [%] |
| Hsu [12] | 96 | - | 94 | - | 95 | - |
| Li - CNN [9] | 98.2 | 94.0 | 98.4 | 92.9 | 95.6 | 87.7 |
| Li - LSTM [9] | - | 94.9 | - | 94.2 | - | 88.4 |
| Proposed | **99.43** | **97.05** | **99.78** | **98.83** | **99.51** | **97.71** |

### License plate detection and recognition task

The performance of our system over the AOLP dataset in the LPDR task is shown in Table 6.6. A comparison with the corresponding results over the *PlatesMania.com* database of Italian plates in Table 6.2 shows a comparable performance. Although the Taiwanese plates have one character less to classify we consider the LR task more difficult because more similar digits are allowed in the same positions (in fact, most of the errors observed could never occur on Italian license plates, because they regard a banned character, a letter classified as a number or vice versa). Therefore, this experiment shows that our LPR system is capable of accurate plate detection across widely different types of vehicle plates. Fig. 6.13 shows a few examples of final classifications of the LPDR task.

Table 6.6 Complete classification precision (PR) and recall (RE) for our proposed system on the AOLP dataset.

| Subset | AC | | LE | | RP | |
|---|---|---|---|---|---|---|
| Method | PR [%] | RE [%] | PR [%] | RE [%] | PR [%] | RE [%] |
| Proposed | 97.63 | 96.91 | 99.33 | 98.50 | 98.84 | 98.03 |


Fig. 6.13 Examples of AOLP images processed by our LPR system. Blue boxes correspond to detected plates, red boxes to detected characters (textual reading is superimposed).

## 6.6 GPU computational complexity

As a final experiment, we measure the computational complexity on a GPU of our LPR architecture in two different scenarios. In the first scenario, a smart surveillance camera with an *NVIDIA Jetson TX1* embedded board (256 CUDA cores) captures mid-resolution images and performs LPR aboard the camera. In the second scenario, a high-resolution camera streams the captured images to a remote LPR server equipped with a *GeForce GTX 1080* GPU with 2560 CUDA cores. We measure the time required by the GPU to apply the described CNNs in the pipeline. Table 6.7 shows that, as expected, the plate processing time is proportional to number of pixels. Most important, most of the complexity lies in the processing of the first, full-resolution, layer of the pyramid of query images. The additional complexity of processing the remaining downscaled layers is only a fraction thereof. Therefore, the relative penalty of our pyramid-based approach to scale invariance with respect to architectures such as [89, 105] is only marginal, whereas the low-complexity CNN design allows for about 1-2 fps processing in the embedded scenario, and about 13-40 fps in the server scenario, while coping with a potentially unlimited number of scales. Finally, we benchmark the VGG16-based *Faster R-CNN* [89] architecture with test images of the same size (one detectable object per image). The complexity of the R-CNN architecture is higher than that of our entire LPR system, showing the benefits of our ad-hoc, parameter-savvy, CNN design.

Table 6.7 GPU Processing Time [ms].

| Platform | Jetson TX1 | | GeForce GTX 1080 | |
|---|---|---|---|---|
| Resolution | $320 \times 240$ | $640 \times 480$ | $640 \times 480$ | $1280 \times 960$ |
| Plate classif. (base layer) | 101 | 457 | 12.5 | 47.5 |
| Plate classif. (other layers) | 26.4 | 127 | 5.5 | 18.0 |
| Plate refinement | 23.9 | 23.9 | 2.2 | 2.2 |
| Char classific. | 191 | 191 | 3.2 | 3.2 |
| Char refinement | 46.2 | 46.2 | 2.2 | 2.2 |
| Total Proposed | 389 | 845 | 25.5 | 72.7 |
| Faster R-CNN | - | - | 82 | 96 |

# Chapter 7

# Conclusion and Future Work

In this thesis we studied the feasibility of using synthetic samples to train convolutional neural networks for license plate detection as character recognition and design of a classification pipeline to make use of such networks.

By using automatically annotated synthetic training samples we avoid the often unfeasible tasks of collecting and manually annotating real samples and we show that it is possible to achieve good classification results with only synthetic samples. Furthermore, we show that by training these networks for sufficiently long, the classifications are robust enough to generalize on real samples without need of a validation set of real, annotated images for calibration. However, should such a set be available, it is possible to further calibrate the thresholds and improve the accuracy and/or balance the precision-recall trade-off as desired.

Furthermore, by generating Taiwanese plates and testing our system on the AOLP dataset; we show that the whole system, from including sample generation, training and classification pipeline with robust thresholds, generalizes to other new license plate types, by achieving state-of-the-art performance on the dataset.

We also show that our added stages in the pipeline to solve the issue of scale dependency in CNNs (by employing scale pyramid), and the refinement classifications stages to improved the accuracy, both increase the computational complexity by a relatively small amount compared to the required initial classification stage.

A limitation of this study is that although it can be applied to different plate types, it only considers one type at a time. In a real world scenario there are often multiple

plate types available, and if would not be sufficient with a classifier discriminating between a single plate type and background.

For the synthetic plate generation this poses no real challenges, other than a bigger data to account for each plate type, it rather highlights the advantage of synthetic sample generation when considering the increased amount of work that would be required to manually collect and annotate real samples for each type, of which some plate types may be rarer than other.

The plate classifier would either need to be trained for a multiple class problem, or be trained on a mixed set if the plate type is not required as long as the license number is correctly read. For our pipeline design the former is probably preferred, for two reasons. First, because the rectification stage should ideally rectify each plate type to its native dimensions to simplify the task of the character classifier as much as possible. While in the latter case, the rectification would not work well because the specific plate dimensions to transform the plate onto would not be known, likely it would work better to replace the rectification stage with one that only rotates the image to aligns the plate horizontally, which would still be useful before the character classification. And second, because different plate types may use different fonts, so individual character classifiers may be preferred for some or all plate types.

The error correction stage would also need some modifications to handle correction of plate type to consider if the plate was possibly of another type than first classified, if the syntax of the found characters match another plate type. This would not be a complicated change since the current method already considers the multi-class problem of multiple characters, so it would be similar to just adding another character.

Another limitation is that we only consider single frame classification, while many of today's cameras can acquire multiple frames per second.

A simple solution would be to first treat frames individually with the current pipeline, and merge classifications afterwards, and if two classifications between frames are similar but not identical, use a modified version of the error correction stage to determine which is most likely to be correct.

Another approach would be to consider the images as a time series in the classification stage and use either recurrent convolutional neural networks or a using 3D convolutional neural networks which allow the convolution to shift in the time

dimension as well. This would require some significant changes to the synthetic sample generation algorithm to output multiple frames with movement of the plate between them. However, since the algorithm is already considering the plate as a 3D object being projected to a 2D camera, realistic movements would be relatively easy to simulate by updating the 3D coordinates of the plate position according to realistic velocity, acceleration and rotation before projecting the frames as 2D images.

# References


[1] Christos-Nikolaos E Anagnostopoulos. License plate recognition: A brief tutorial. *IEEE Intelligent transportation systems magazine*, 6(1):59–67, 2014.

[2] Rahim Panahi and Iman Gholampour. Accurate detection and recognition of dirty vehicle plate numbers for high-speed applications. *IEEE Transactions on Intelligent Transportation Systems*, 18(4):767–779, 2017.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

[5] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. *arXiv preprint arXiv:1406.2227*, 2014.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[7] Michael Mathieu, Yann LeCun, Rob Fergus, David Eigen, Pierre Sermanet, and Xiang Zhang. Overfeat: Integrated recognition, localization and detection using convolutional networks. 2013.

[8] Yoshua Bengio, Ian J Goodfellow, and Aaron Courville. Deep learning. *Nature*, 521:436–444, 2015.

[9] Hui Li and Chunhua Shen. Reading car license plates using deep convolutional neural networks and lstms. *arXiv preprint arXiv:1601.05610*, 2016.

[10] Orhan Bulan, Vladimir Kozitsky, Palghat Ramesh, and Matthew Shreve. Segmentation-and annotation-free license plate recognition with deep localization and failure identification. *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[11] David Menotti, Giovani Chiachia, Alexandre X Falcão, and VJ Oliveira Neto. Vehicle license plate recognition with random convolutional networks. In *Graphics, Patterns and Images (SIBGRAPI), 2014 27th SIBGRAPI Conference on*, pages 298–303. IEEE, 2014.

[12] Gee-Sern Hsu, Jiun-Chang Chen, and Yu-Zu Chung. Application-oriented license plate recognition. *IEEE transactions on vehicular technology*, 62(2):552–561, 2013.

[13] Fernando Martin, Maite Garcia, and José Luis Alba. New methods for automatic reading of vlp's (vehicle license plates). In *Proc. IASTED Int. Conf. SPPRA*, volume 26, pages 257–271, 2002.

[14] Shen-Zheng Wang and Hsi-Jian Lee. Detection and recognition of license plate characters with different appearances. In *Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE*, volume 2, pages 979–984. IEEE, 2003.

[15] Bai Hongliang and Liu Changping. A hybrid license plate extraction method based on edge statistics and morphology. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 831–834. IEEE, 2004.

[16] Danian Zheng, Yannan Zhao, and Jiaxin Wang. An efficient method of license plate location. *Pattern recognition letters*, 26(15):2431–2438, 2005.

[17] Abbas M Al-Ghaili, Syamsiah Mashohor, Abdul Rahman Ramli, and Alyani Ismail. Vertical-edge-based car-license-plate detection method. *IEEE transactions on vehicular technology*, 62(1):26–38, 2013.

[18] Nikolaos Bellas, Sek M Chai, Malcolm Dwyer, and Dan Linzmeier. Fpga implementation of a license plate recognition soc using automatically generated streaming accelerators. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8–pp. IEEE, 2006.

[19] Hsien-Huang P Wu, Hung-Hsiang Chen, Ruei-Jan Wu, and Day-Fann Shen. License plate extraction in low resolution video. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 1, pages 824–827. IEEE, 2006.

[20] Paolo Comelli, Paolo Ferragina, Mario Notturno Granieri, and Flavio Stabile. Optical recognition of motor vehicle license plates. *IEEE transactions on Vehicular Technology*, 44(4):790–799, 1995.

[21] Sorin Draghici. A neural network based artificial vision system for licence plate recognition. *International Journal of Neural Systems*, 8(01):113–126, 1997.

[22] J Barroso, EL Dagless, A Rafael, and J Bulas-Cruz. Number plate reading using computer vision. In *Industrial Electronics, 1997. ISIE'97., Proceedings of the IEEE International Symposium on*, pages 761–766. IEEE, 1997.

[23] Javier Cano and Juan-Carlos Pérez-Cortés. Vehicle license plate segmentation in natural images. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 142–149. Springer, 2003.

[24] Tsang-Hong Wang, Feng-Chou Ni, Keh-Tsong Li, and Yon-Ping Chen. Robust license plate recognition based on dynamic projection warping. In *Networking, Sensing and Control, 2004 IEEE International Conference on*, volume 2, pages 784–788. IEEE, 2004.

[25] A Broumandnia and M Fathy. Application of pattern recognition for farsi license plate recognition. In *ICGST Int. Conf. Graphics, Vision and Image Processing (GVIP)*, number V2, pages 25–31, 2005.

[26] Jun Kong, Xinyue Liu, Yinghua Lu, and Xiaofeng Zhou. A novel license plate localization method based on textural feature analysis. In *Signal Processing and Information Technology, 2005. Proceedings of the Fifth IEEE International Symposium on*, pages 275–279. IEEE, 2005.

[27] Hung Ngoc Do, Minh-Thanh Vo, Bao Quoc Vuong, Huy Thanh Pham, An Hoang Nguyen, and Huy Quoc Luong. Automatic license plate recognition using mobile device. In *Advanced Technologies for Communications (ATC), 2016 International Conference on*, pages 268–271. IEEE, 2016.

[28] Guangzhi Cao, Jianqian Chen, and Jingping Jiang. An adaptive approach to vehicle license plate localization. In *Industrial Electronics Society, 2003. IECON'03. The 29th Annual Conference of the IEEE*, volume 2, pages 1786–1791. IEEE, 2003.

[29] Xifan Shi, Weizhong Zhao, and Yonghang Shen. Automatic license plate recognition system based on color image processing. In *International Conference on Computational Science and Its Applications*, pages 1159–1168. Springer, 2005.

[30] Shyang-Lih Chang, Li-Shien Chen, Yun-Chung Chung, and Sei-Wan Chen. Automatic license plate recognition. *IEEE transactions on intelligent transportation systems*, 5(1):42–53, 2004.

[31] L Dlagnekov. License plate detection using adaboost, mar. 2004, san diego. *CA: Comput. Sci. Eng. Dept., Univ. California, San Diego.[Online]. Available: http://www. cse. ucsd. edu/classes/fa04/cse252c/projects/louka. pdf*.

[32] Qiang Wu, Huaifeng Zhang, Wenjing Jia, Xiangjian He, Jie Yang, and Tom Hintz. Car plate detection using cascaded tree-style learner based on hybrid object features. In *Video and Signal Based Surveillance, 2006. AVSS'06. IEEE International Conference on*, pages 15–15. IEEE, 2006.

[33] Huaifeng Zhang, Wenjing Jia, Xiangjian He, and Qiang Wu. Learning-based license plate detection using global and local features. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 2, pages 1102–1105. IEEE, 2006.

[34] Shen-Zheng Wang and Hsi-Jian Lee. A cascade framework for a real-time statistical plate recognition system. *IEEE Transactions on Information Forensics and Security*, 2(2):267–282, 2007.

[35] Kwang In Kim, Keechul Jung, and Jin Hyung Kim. Color texture-based object detection: an application to license plate localization. In *Pattern Recognition with Support Vector Machines*, pages 293–309. Springer, 2002.

[36] Yule Yuan, Wenbin Zou, Yong Zhao, Xinan Wang, Xuefeng Hu, and Nikos Komodakis. A robust and efficient approach to license plate detection. *IEEE Transactions on Image Processing*, 26(3):1102–1114, 2017.

[37] Eun Ryung Lee, Pyeoung Kee Kim, and Hang Joon Kim. Automatic recognition of a car license plate using color image processing. In *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, volume 2, pages 301–305. IEEE, 1994.

[38] MH Ter Brugge, JH Stevens, JAG Nijhuis, and L Spaanenburg. License plate recognition using dtcnns. In *Cellular Neural Networks and Their Applications Proceedings, 1998 Fifth IEEE International Workshop on*, pages 212–217. IEEE, 1998.

[39] Sung Han Park, Kwang In Kim, Keechul Jung, and Hyung Jin Kim. Locating car license plates using neural networks. *Electronics Letters*, 35(17):1475–1477, 1999.

[40] Wu Wei, Yuzhi Li, Mingjun Wang, and Zhongxiang Huang. Research on number-plate recognition based on neural networks. In *Neural Networks for Signal Processing XI, 2001. Proceedings of the 2001 IEEE Signal Processing Society Workshop*, pages 529–538. IEEE, 2001.

[41] Mario I Chacon and A Zimmerman. License plate location based on a dynamic pcnn scheme. In *Proc. Int. Joint Conf. Neural Netw*, volume 2, pages 1195–1200, 2003.

[42] Cemil Oz and Fikret Ercal. A practical license plate recognition system for real-time environments. In *International Work-Conference on Artificial Neural Networks*, pages 881–888. Springer, 2005.

[43] Gang Li, Ruili Zeng, and Ling Lin. Research on vehicle license plate location based on neural networks. In *Innovative Computing, Information and Control, 2006. ICICIC'06. First International Conference on*, volume 3, pages 174–177. IEEE, 2006.

[44] Sang Kyoon Kim, Dae Wook Kim, and Hang Joon Kim. A recognition of vehicle license plate using a genetic algorithm based segmentation. In *Image Processing, 1996. Proceedings., International Conference on*, volume 2, pages 661–664. IEEE, 1996.

[45] Giovanni Adorni, Stefano Cagnoni, and Monica Mordonini. Efficient low-level vision program design using sub-machine-code genetic programming. In *Proc. Workshop sulla Percezione e Visione nelle Macchine*, 2002.

[46] Seiki Yoshimori, Yasue Mitsukura, Minoru Fukumi, and Norio Akamatsu. License plate detection using hereditary threshold determine method. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 585–593. Springer, 2003.

[47] Xiong Jun, Du Sidan, Gao Duntang, and Shen Qinhong. Locating car license plate under various illumination conditions using genetic algorithm. In *Signal Processing, 2004. Proceedings. ICSP'04. 2004 7th International Conference on*, volume 3, pages 2502–2505. IEEE, 2004.

[48] Hans A Hegt, Ron J De La Haye, and Nadeem A Khan. A high performance license plate recognition system. In *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, volume 5, pages 4357–4362. IEEE, 1998.

[49] Kap Kee Kim, KI Kim, JB Kim, and Hang Joon Kim. Learning-based approach for license plate recognition. In *Neural Networks for Signal Processing X, 2000. Proceedings of the 2000 IEEE Signal Processing Society Workshop*, volume 2, pages 614–623. IEEE, 2000.

[50] Choudhury A Rahman, Wael Badawy, and Ahmad Radmanesh. A real time vehicle's license plate recognition system. In *Advanced Video and Signal Based Surveillance, 2003. Proceedings. IEEE Conference on*, pages 163–166. IEEE, 2003.

[51] A Taleb-Ahmed, Denis Hamad, and Gautier Tilmant. Vehicle license plate recognition in marketing application. In *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*, pages 90–94. IEEE, 2003.

[52] Yue Cheng, Jiaiuning Lu, and Takashi Yahagi. Car license plate recognition based on the combination of principal components analysis and radial basis function networks. In *Signal Processing, 2004. Proceedings. ICSP'04. 2004 7th International Conference on*, volume 2, pages 1455–1458. IEEE, 2004.

[53] Tran Duc Duan, TL Hong Du, Tran Vinh Phuoc, and Nguyen Viet Hoang. Building an automatic vehicle license plate recognition system. In *Proc. Int. Conf. Comput. Sci. RIVF*, pages 59–63, 2005.

[54] Shigueo Nomura, Keiji Yamanaka, Osamu Katai, Hiroshi Kawakami, and Takayuki Shiose. A novel adaptive morphological approach for degraded character image segmentation. *Pattern Recognition*, 38(11):1961–1975, 2005.

[55] Cheokman Wu, Lei Chan On, Chan Hon Weng, Tong Sio Kuan, and Kengchung Ng. A macao license plate recognition system. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 7, pages 4506–4510. IEEE, 2005.

[56] JAG Nijhuis, MH Ter Brugge, KA Helmholt, JPW Pluim, L Spaanenburg, RS Venema, and MA Westenberg. Car license plate recognition with neural networks and fuzzy logic. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 5, pages 2232–2236. IEEE, 1995.

[57] BL Lim, Wenzheng Yeo, KY Tan, and CY Teo. A novel dsp based real-time character classification and recognition algorithm for car plate detection and recognition. In *Signal Processing Proceedings, 1998. ICSP'98. 1998 Fourth International Conference on*, volume 2, pages 1269–1272. IEEE, 1998.

[58] Fatih Kahraman, Binnur Kurt, and Muhittin Gökmen. License plate character segmentation based on the gabor transform and vector quantization. In *International Symposium on Computer and Information Sciences*, pages 381–388. Springer, 2003.

[59] David Llorens, Andrés Marzal, Vicente Palazón, and Juan M Vilar. Car license plates extraction and recognition based on connected components analysis and hmm decoding. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 571–578. Springer, 2005.

[60] Hamid Mahini, Shohreh Kasaei, and Faezeh Dorri. An efficient features-based license plate localization method. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 2, pages 841–844. IEEE, 2006.

[61] Charl Coetzee, Charl Botha, and David Weber. Pc based number plate recognition system. In *Industrial Electronics, 1998. Proceedings. ISIE'98. IEEE International Symposium on*, volume 2, pages 605–610. IEEE, 1998.

[62] Takashi Naito, Toshihiko Tsukada, Keiichi Yamada, Kazuhiro Kozuka, and Shin Yamamoto. Robust license-plate recognition method for passing vehicles under outside environment. *IEEE Transactions on Vehicular Technology*, 49(6):2309–2319, 2000.

[63] Byeong Rae Lee, Kyungsoo Park, Hyunchul Kang, Haksoo Kim, and Chungkyue Kim. Adaptive local binarization method for recognition of vehicle license plates. In *International Workshop on Combinatorial Image Analysis*, pages 646–655. Springer, 2004.

[64] Jiangmin Tian, Ran Wang, Guoyou Wang, Jianguo Liu, and Yuanchun Xia. A two-stage character segmentation method for chinese license plate. *Computers & Electrical Engineering*, 46:539–553, 2015.

[65] Thanongsak Sirithinaphong and Kosin Chamnongthai. The recognition of car license plate for automatic parking system. In *Signal Processing and Its Applications, 1999. ISSPA'99. Proceedings of the Fifth International Symposium on*, volume 1, pages 455–457. IEEE, 1999.

[66] Thanongsak Sirithinaphong and Kosin Chamnongthai. Extraction of car license plate using motor vehicle regulation and character pattern recognition.

In *Circuits and Systems, 1998. IEEE APCCAS 1998. The 1998 IEEE Asia-Pacific Conference on*, pages 559–562. IEEE, 1998.

[67] C Anagnostopoulos, E Kayafas, and V Loumos. Digital image processing and neural networks for vehicle license plate identification. *Journal of Electrical Engineering*, 1(2):2–7, 2000.

[68] C Anagnostopoulos, I Anagnostopoulos, G Tsekouras, G Kouzas, V Loumos, and E Kayafas. Using sliding concentric windows for license plate segmentation and processing. In *Signal Processing Systems Design and Implementation, 2005. IEEE Workshop on*, pages 337–342. IEEE, 2005.

[69] Yafeng Hu, Feng Zhu, and Xianda Zhang. A novel approach for license plate recognition using subspace projection and probabilistic neural network. In *International Symposium on Neural Networks*, pages 216–221. Springer, 2005.

[70] Christos Nikolaos E Anagnostopoulos, Ioannis E Anagnostopoulos, Vassilis Loumos, and Eleftherios Kayafas. A license plate-recognition algorithm for intelligent transportation system applications. *IEEE Transactions on Intelligent transportation systems*, 7(3):377–392, 2006.

[71] Yo-Ping Huang, Shi-Yong Lai, and Wei-Po Chuang. A template-based model for license plate recognition. In *Networking, Sensing and Control, 2004 IEEE International Conference on*, volume 2, pages 737–742. IEEE, 2004.

[72] Christos-Nikolaos E Anagnostopoulos, Ioannis E Anagnostopoulos, Ioannis D Psoroulas, Vassili Loumos, and Eleftherios Kayafas. License plate recognition from still images and video sequences: A survey. *IEEE Transactions on intelligent transportation systems*, 9(3):377–391, 2008.

[73] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[74] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedingfs of the IEEE*, 77(2):257–286, 1989.

[75] Xiang Pan, Xiuzi Ye, and Sanyuan Zhang. A hybrid method for robust car plate character recognition. *Engineering Applications of Artificial Intelligence*, 18(8):963–972, 2005.

[76] Inc. OpenALPR Technology. Openalpr - automatic license plate recognition, 2018.

[77] Inc. OpenALPR Technology. Openalpr - automatic license plate recognition, 2018.

[78] Jia Li, Changyong Niu, and Ming Fan. Multi-scale convolutional neural networks for natural scene license plate detection. In *International Symposium on Neural Networks*, pages 110–119. Springer, 2012.

[79] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Deep features for text spotting. In *European conference on computer vision*, pages 512–528. Springer, 2014.

[80] Kai Wang, Boris Babenko, and Serge Belongie. End-to-end scene text recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1457–1464. IEEE, 2011.

[81] Chao Gou, Kunfeng Wang, Yanjie Yao, and Zhengxi Li. Vehicle license plate recognition based on extremal regions and restricted boltzmann machines. *IEEE Transactions on Intelligent Transportation Systems*, 17(4):1096–1107, 2016.

[82] Yoav Freund and David Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. In *Advances in neural information processing systems*, pages 912–919, 1992.

[83] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

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

[85] 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*, pages 3431–3440, 2015.

[86] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[87] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[88] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2016.

[89] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[90] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[91] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[92] YT Zhou and R Chellappa. Computation of optical flow using a neural network. In *IEEE International Conference on Neural Networks*, volume 27, pages 71–78, 1988.

[93] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[94] M Bishop Christopher. *PATTERN RECOGNITION AND MACHINE LEARNING*. Springer-Verlag New York, 2016.

[95] Stephen Marsland. Machine learning: an algorithmic perspective. *USA: Chapman &Hall/CRC*, 2009.

[96] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[97] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[98] Yuri Reznik, G Cordara, and M Bober. Evaluation framework for compact descriptors for visual search. *ISO/IEC JTC1/SC29/WG11 N*, 12202:2011, 2011.

[99] Stefan Carlsson. Geometric computing in image analysis and visualization. Lecture Notes, KTH Royal Insittute of Technology, Department of Numerical Analysis and Computing Science, 2007.

[100] George H Joblove and Donald Greenberg. Color spaces for computer graphics. In *ACM siggraph computer graphics*, volume 12, pages 20–25. ACM, 1978.

[101] Kjell Carlsson. Light microscopy. *KTH Physics, Stockholm*, 2007.

[102] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[103] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[104] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.

[105] Jianan Li, Xiaodan Liang, ShengMei Shen, Tingfa Xu, Jiashi Feng, and Shuicheng Yan. Scale-aware fast r-cnn for pedestrian detection. *IEEE Transactions on Multimedia*, 2017.

[106] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.

[107] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[108] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

# Appendix A

# Camera perspective projection

This section follows [99].

When a camera captures an image of the 3D world, the image is *perspectively projected* on the image sensor, as illustrated in Fig. A.1. In reality, the image is flipped and recorded behind a lens as illustrated by the green rectangle. However, to avoid confusion with signs, let us define the plane $(x, y, 1)$ the *image plane*, illustrated by the red rectangle in Fig. A.1.

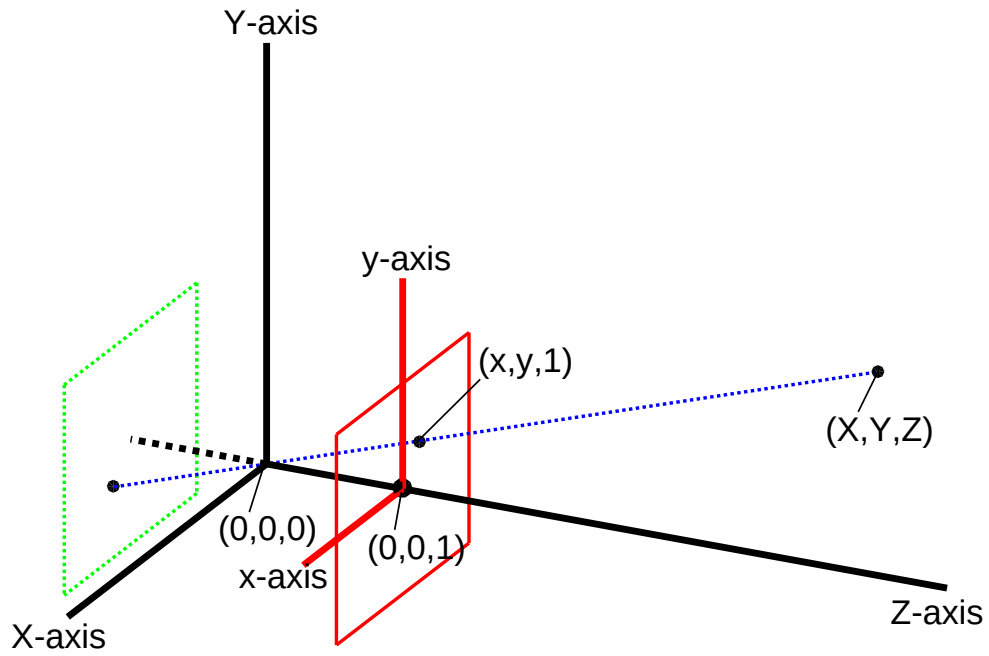



Fig. A.1 Perspective projection of 3D coordinates $(X, Y, Z)$ to image coordinates $(x, y)$

Through similar-triangle-relations we see that the projected coordinates in the image plate of a 3D point $P = (X,Y,Z)$ are given by:

$$\begin{aligned} x &= X/Z \\ y &= Y/Z \end{aligned} \tag{A.1}$$

Using *homogeneous coordinates*, this can be expressed as a linear equation system (note however that the projection is non-linear and therefore the scale factor $\lambda$ is introduced, which is typically different for each set of points):

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{A.2}$$

## A.1 Rotations

The above equations were particularly simple to derive because the coordinate systems of the image plane and the 3D world are aligned by sharing parallel X and Y axes. However, when different coordinates systems are used we can find the transformation between them, using translations and rotations, to keep using the simple equations.

A rotation around the Y-axis by an angle $\theta$, as in Fig. A.2, between the coordinate systems $(X,Y,Z)$ and $(X',Y',Z')$ has the following transformation:

$$\begin{aligned} X' &= cos(\theta)X + sin(\theta)Z \\ Y' &= \qquad\quad Y \\ Z' &= -sin(\theta)X + cos(\theta)Z \end{aligned} \tag{A.3}$$

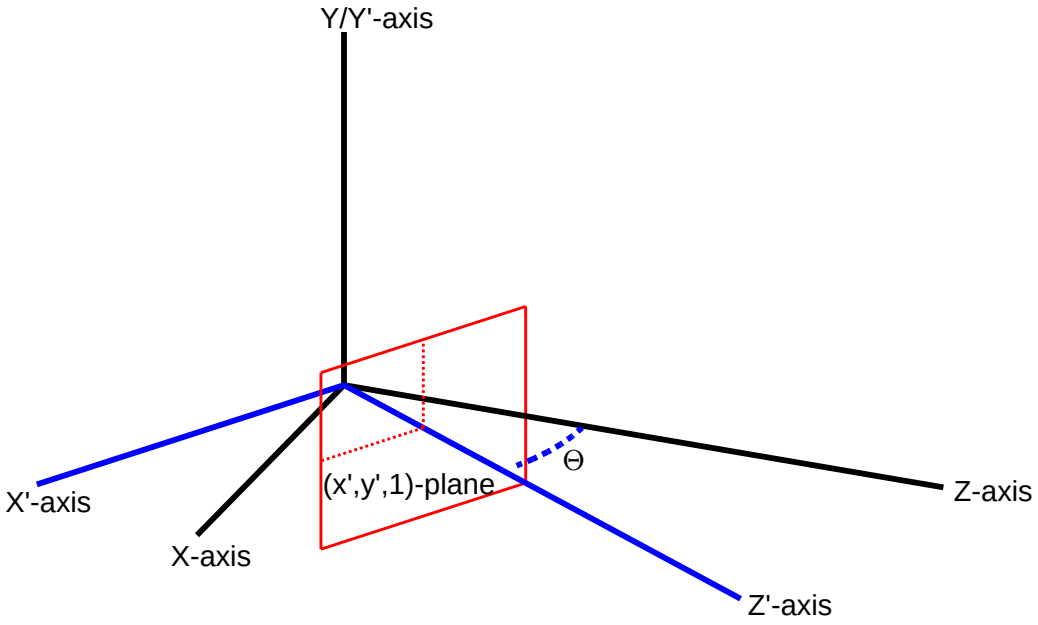



Fig. A.2 Rotation of camera angle $\Theta$ around the $Y-$ axis.

Using matrix notation we can isolate the transformation matrix, or *rotation matrix*:

$$\begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{A.4}$$

Applying this transformation we can obtain the coordinates in the rotated camera plane, of a point in the original system:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \lambda' \begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{A.5}$$

By defining rotation matrices for all the axes as follows,

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix} \tag{A.6}$$

$$R_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{A.7}$$

$$R_z(\psi) = \begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.8}$$

we can define a general rotation matrix,

$$R(\phi, \theta, \psi) = R_x(\phi)R_y(\theta)R_z(\psi), \tag{A.9}$$

and generalize (A.5) to a general rotation:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \lambda' R(\phi, \theta, \psi) \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{A.10}$$

Note: In reality the image plane is also on the negative $Z$-axis, and the image is flipped, i.e. all axes are revered in the real camera coordinate system, however we will continue to picture the image plate in a positively aligned coordinate system for simplicity.

## A.2 Internal camera parameters

The camera model in (A.2) is limited in that the image plane is at unit distance and the coordinates in the image are measured in the same unit as the 3D real world coordinates. In practice the image sensor is distanced by the *focal length* from the origin and the distances in the sensor are more conveniently measured in pixels, furthermore the sensor may not be perfectly centered along the optical axis. With the modified camera model in Fig. A.3 in mind, after rescaling the image coordinates to the image plane specific coordinate system, (A.1) becomes:

$$\begin{aligned} \frac{x-x_0}{\sigma_x f} &= X/Z \\ \frac{y-y_0}{\sigma_y f} &= Y/Z, \end{aligned} \tag{A.11}$$

where $x_0$ and $y_0$ are the image coordinates of the principal point (where the optical axis crosses the image plane), $\sigma_x$ and $\sigma_y$ are the scale factors for respective image axis and $f$ is the focal length of the lens.

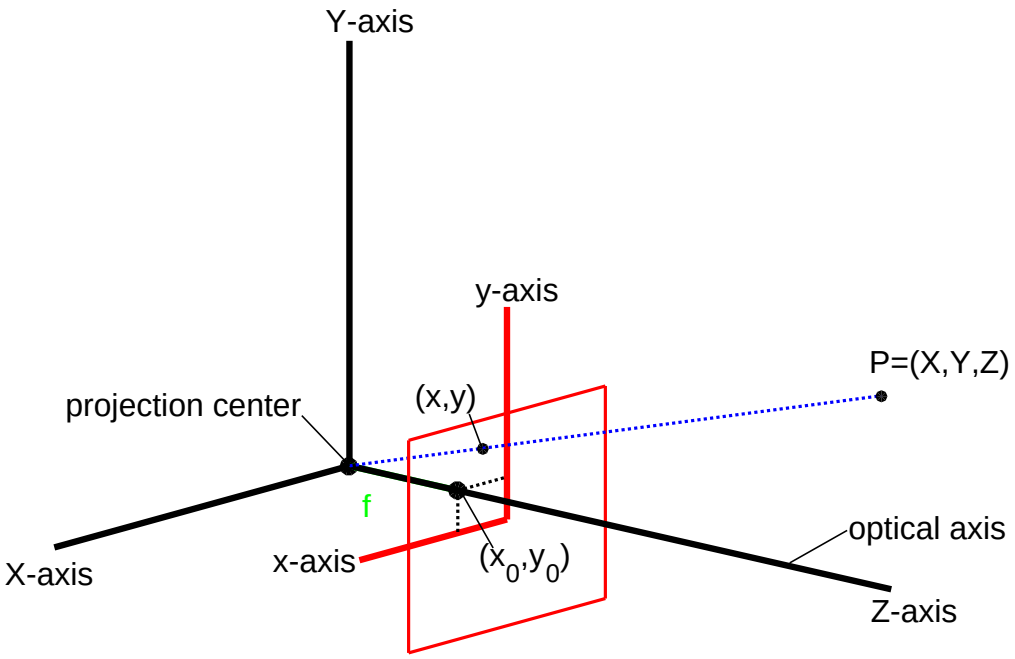



Fig. A.3 Perspective projection with internal camera parameters

Using (A.11) instead of (A.1), (A.2) can now be written as:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \lambda \begin{bmatrix} \sigma_x f & 0 & x_0 \\ 0 & \sigma_y f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{A.12}$$

Let us define $K$ as the matrix of internal camera parameters (for completeness a skew factor, $\gamma$, has been added, but for our purposes $\gamma = 0$):

$$K = \begin{bmatrix} \sigma_x f & \gamma & x_0 \\ 0 & \sigma_y f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.13}$$

Combining (A.12) with (A.10) we get the final equation from camera perspective projection:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \lambda' KR(\phi, \theta, \psi) \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{A.14}$$

## A.3 2D affine image transformation

Since the images in our case, although positioned in 3D, are actually 2D images, straight lines will be preserved throughout the perspective projection (but angles between lines will change). Therefore, an affine 2D to 2D transformation can be used once the corners are projected. This simplifies things significantly since $\lambda'$ would otherwise need to be calculated individually for each projected point.

Using the notation in [106]

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = A_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{1,1} & \theta_{1,2} & \theta_{1,3} \\ \theta_{2,1} & \theta_{2,2} & \theta_{2,3} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} \tag{A.15}$$

where $(x_i^s, y_i^s)$ is a source image point and $(x_i^t, y_i^t, 1)$ is its corresponding location in the transformed image, again using homogeneous coordinates. And $A_\theta$ is the $2x3$ affine transformation matrix. Using the 4 image corners as $(x_1^s, y_1^s)$ - $(x_4^s, y_4^s)$ their corresponding projections as $(x^t s_1, y_1^t)$ - $(x_4^t, y_4^t)$. This yields 8 equations, to determine the 6 transformation parameters in $A_\theta$, i.e. an overdetermined equation system:

$$\begin{bmatrix} x_1^s & x_2^s & x_3^s & x_4^s \\ y_1^s & y_2^s & y_3^s & y_4^s \end{bmatrix} = \begin{bmatrix} \theta_{1,1} & \theta_{1,2} & \theta_{1,3} \\ \theta_{2,1} & \theta_{2,2} & \theta_{2,3} \end{bmatrix} \begin{bmatrix} x_1^t & x_2^t & x_3^t & x_4^t \\ y_1^t & y_2^t & y_3^t & y_4^t \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{A.16}$$

or in short:

$$X^s = A_\theta X^t \tag{A.17}$$

Using the Moore-Penrose pseudoinverse (indicated with $(\cdot)^+$), the least squares solution of $A_\theta$ is given by:

$$A_\theta = X^s (X^t)^+ \tag{A.18}$$

Hence, with the projected corners and $A_\theta$ known, (A.15) provides an easier way to map the image pixels to the entire projected area instead of using (A.14 pixel by pixel, it allows for multiple pixels to be calculated as a matrix computation as in (A.16).