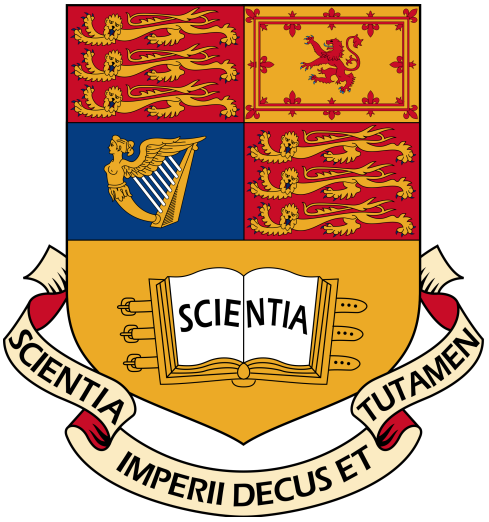


Imperial College London

C362 - Software Engineering Group Project

Deblurring License Plates

Authors
Ramon Fernández i Mir
Chris Hawkes
Robert Holland
Ruiao Hu
Rakhilya Lee Mekhtieva
Aris Papadopoulos

Supervisor
Prof. William Knottenbelt

Autumn 2017

Contents

List of Figures

List of Tables

1 Executive Summary

With the advances in the field of Machine Learning in recent years, many image processing problems can now be solved using new techniques such as neural networks. A good example of one of these problems, in the area of security, is being able to read the license plates in pictures of vehicles taken by surveillance cameras. These cameras often record at a very low frame rate and in low resolution, making it quite challenging to automatically read license plates.

To remedy this, CCTV cameras could be replaced with high resolution ones but this would be, of course, a very costly and time consuming operation. For this reason, an alternative approach based on software seems at the moment way more promising. Companies providing public safety solutions for authorities around the world, like Huawei does in China, have achieved the best results reading license plates to date using Machine Learning techniques instead of traditional image processing ones.

A common approach is to first deblur the picture, so that an OCR system can read the characters. Huawei's R&D team that focuses on media processing proposed the problem of deblurring license plates as a challenge open to all Imperial College students.

The method we propose was conceived after exhaustive research on both the classical and state-of-the-art methods used for image deblurring. Multiple experiments were carried out to find the best match to perform this very specific task. Our algorithm, Moussaka, named after a many-layered Greek dish, is based on Deep Learning and has an accuracy of over 75% with a sample of 100 unseen license plates, as evaluated by Huawei's Machine Learning experts. They praised Moussaka for the clarity of the output images and, in particular, its performance with Chinese characters. We were awarded the runner-up prize in the competition.

Furthermore, as a proof of concept and for demoing purposes we have developed an Android application that wraps the core functionality in a usable product. The app not only allows us to quickly show what our algorithm does in a real world scenario but also demonstrates its efficiency by running it offline on a mobile phone.

2 Motivations and Achievements

In this fragment of the report we explain what each of the parties involved in this project wanted to achieve with it and what was actually achieved in the end.

2.1 Huawei's Goal

Public safety is of vital importance in our society and it is especially difficult to ensure in congested areas such as city centres. In order for authorities to offer the citizens an efficient emergency response, crime and hazard prevention and more, video surveillance is always needed to some degree. The software used to manage this equipment and analyse the large amounts of data that are associated with constant surveillance is very complex and there are many factors to take into account.

Huawei's vision of the future is a completely interconnected world where all the devices communicate with one another to offer a service to the community. One of the areas they want to develop in that direction is precisely public safety. They offer a package to governments that they call 'Safe City Solution'. According to their website, some of the of the challenges they address is 'Surveillance blind spots, unclear images, difficult video retrieval, and data damage or loss' [12]. This is precisely the area of their business this project is related to. The team responsible for this product is led by Dr Jia Cai and located in the Hangzhou Research Centre. It was this team that proposed the topic of the competition. They also deal with other image processing and understanding issues.

With this competition they particularly wanted to challenge students to use original techniques to tackle a very specific problem: deblurring license plates. As explained in the executive summary, this is an important step to read the characters in a license plate, which is the ultimate objective of that fraction of their software.

2.2 Our Goal

Before the list of project proposals was out, we had already discussed that we wanted to do a project involving Computer Vision. We were all also very keen on doing a Machine Learning project, since it was an area we were interested in but of which we had a very shallow understanding. When we saw this proposal, we thought that it combined both in a nice way and that it was the perfect opportunity to strengthen our knowledge in these areas. Another important factor that influenced our decision was the research component in the project, since all the members of our team are to some extent interested in research. Finally, the fact that it was in the form of a competition was a great motivation for us.

2.3 Competition Results and Learning Outcomes

The competition was open to all Imperial College students from undergraduate to post-doc level. From all the teams that submitted their deblurring algorithms to the competition we are proud to say that we were awarded the runner-up prize.

Figure 2.1: The team in the awards ceremony.

The prize included a Huawei P10 Plus phone for each of us as well as a week-long all-expenses-paid trip to China to see their headquarters in Shenzhen and meet their Image Deblurring experts in Hangzhou. We presented our algorithm to them and they presented theirs to us. This exchange and the discussion that came with it was a very nourishing experience.

Figure 2.2: Meeting the experts in the Hangzhou Research Centre.

We achieved our goal of having a better understanding of Machine Learning applied to a real world problem not only through the development process but also through the meetings with the Huawei experts in our trip to China. We are grateful to Huawei for offering us such an incredible opportunity and to our supervisor, Prof. William Knottenbelt, for making this possible.

3 Project Management

In this section we explain all the organisational aspects of the project. Taking good care of these was crucial to successfully deliver a working product in time. We used Scrum as our agile development framework and worked in two-week-long sprints.

One of our main challenges was having a working version for the competition we had signed up to, with deadline on the 20^{th} of November. The other problems we had to overcome were the scheduling issues caused by the disparate choices of modules between team members, coursework and interviews. These last were expected but could not be dealt with directly during our initial planning but rather on a day-to-day basis.

3.1 Project Plan

In our initial planning session we allocated the first sprint to research, the second and third sprints to developing the core algorithm and the fourth sprint to wrapping it in a usable product. We followed this general scheme, however some of the specifics were changed and updated in each checkpoint adapting to how the project evolved. The table below shows the main objectives accomplished in each sprint:

Sprint	**Objectives**
09/10 - 20/10	· Meet with the Huawei competition organisers and obtain the data set. · Research existing technology for image deblurring. · Experiment with some basic image processing techniques. · Ascertain which resources are required.
23/10 - 03/11	· Introduce our own deep learning network. · Build an end-to-end testing pipeline with score metrics.
06/11 - 17/11	· Write a comparative report of the different approaches. · Make final changes before submitting to the competition. · Train the final version on Azure.
20/11 - 01/12	· Investigate on recolouring the output. · Wrap the algorithm in an Android app using Tensorflow Mobile.

Table 3.1: Main objectives of each sprint.

Regarding the changes from the original plan, we did not consider Dev Ops as an important part of the project at first. Our supervisor suggested dedicating some time at the beginning of the second sprint to build an automated testing pipeline. All this work paid off on the third sprint when we were fine-tuning the learning parameters before submitting a trained model to the competition so it was very convenient to run various experiments and gather the results. We discuss Dev Ops in more detail in Section 3.3.

The other main diversion from what we initially decided has to do with the usable product. At the beginning, we were going to do a simple website but then we read about Tensorflow Mobile and decided to shift to an app since it seemed more challenging and appealing. Before officially notifying this change to our supervisor, we de-risked it by running one of the models on the phone which was the part we were more unsure about.

3.2 Team Structure

Following the Scrum development framework, one of the first things that we did was to choose a Product Owner and a Scrum Master. Because of the nature of our project, the responsibilities that normally come with these roles in real world software development were reduced and they mainly worked as developers. The Product Owner was essentially in charge of the communication with our stakeholders (the competition organisers from Huawei) and the Scrum Master organised the meetings and kept the backlog up to date.

In the following table, we show the main three tasks that each of the team members was responsible for:

Ramon Fernández i Mir Product Owner	· Took care of Dev Ops. · Worked on the final tweaks before submission. · Developed the user interface of the app.
Chris Hawkes Developer	· Added dropouts and pretraining. · Wrote several scripts to augment the data set. · Made Tensorflow Mobile work in Android.
Robert Holland Developer	· Wrote the skeleton of our core network. · Designed the data generation scheme. · Was responsible of data visualisation.
Ruiao Hu Developer	· Wrote half of the 'blurrer'. · Carried out many of the initial experiments. · Read papers on CNNs.
Rakhilya Lee Mekhtieva Developer	· Wrote unit tests for the CNN. · Experimented with a DCGAN and image processing. · Read papers on CNNs and adversarial networks.
Aris Papadopoulos Scrum Master	· Wrote the other half of the 'blurrer'. · Worked on the final tweaks before submission. · Contributed to the data set augmentation.

Table 3.2: Contributions of each team member.

3.3 Development Cycle

In the next subsections we explain in chronological order the steps of our development process from the conception of a feature to the realisation of it. We used Trello as our backlog, Slack as our communication tool, TravisCI as our testing server and GitHub as our version control hosting service.

We found it quite challenging to adapt and take advantage of the software engineering practices we had learnt in previous years and summer internships. The reason why is that this project was mostly a series of experiments rather than building a product. Nevertheless, that did not mean that these practices were not compatible, it meant that we needed to put some more thought into our development process to make sure we were being as efficient as possible. This was actually our main topic of discussion during our first meeting with Dr. Robert Chatley.

3.3.1 Sprint Planning

At the beginning of each sprint we always held a long meeting where we decided the user stories, the tickets in each of them and who was in charge of each ticket. We took into account our deadlines from other modules as well as other exceptional circumstances. These meetings also served as sprint reviews of the previous sprint.

After a sprint planning meeting, our Trello board looked like this:

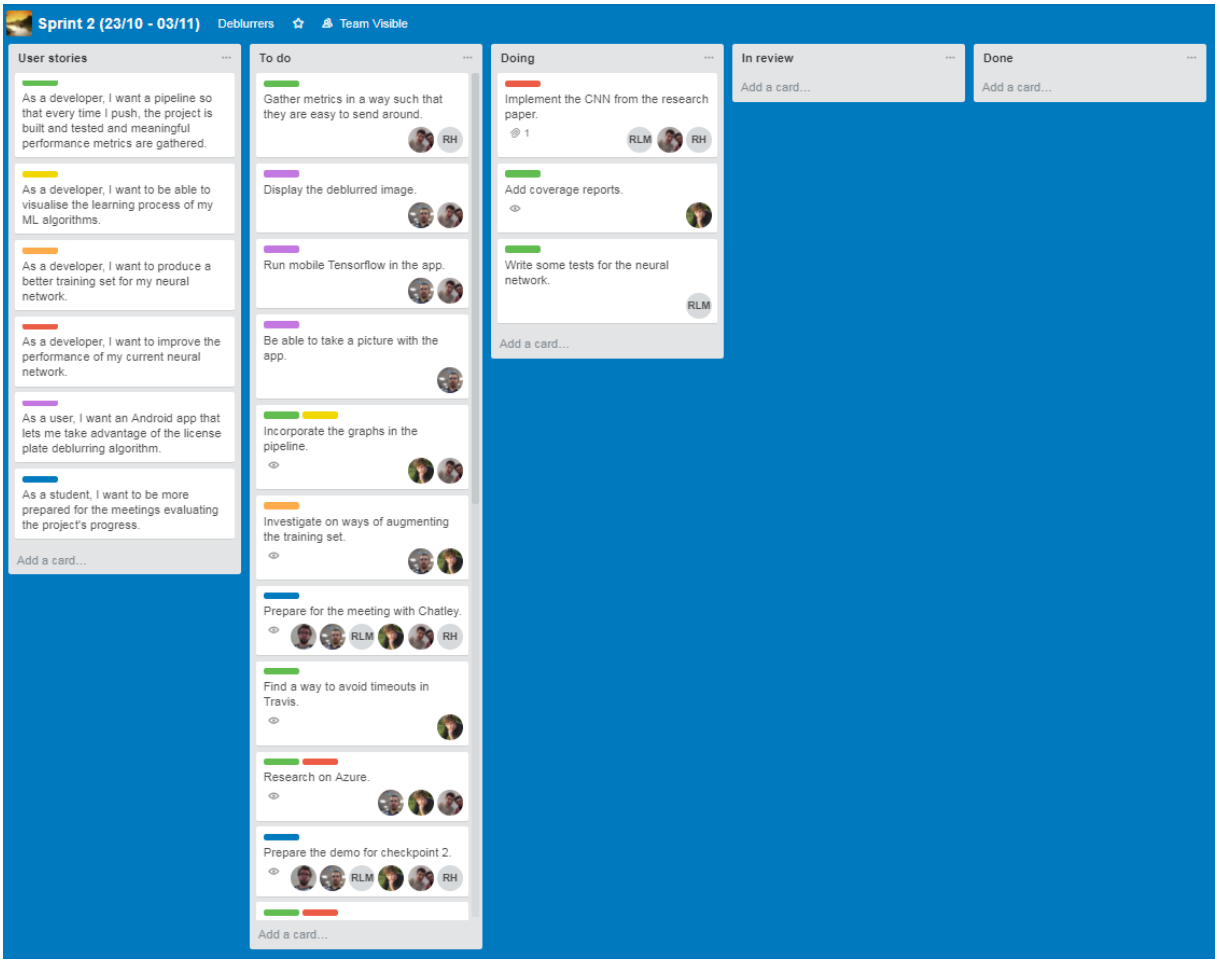


Figure 3.1: State of our Trello board after the second sprint planning.

The colour labels associated each ticket to a user story and gave us an idea of where we were putting more effort. This was useful to distribute the tasks better in the following sprints.

3.3.2 Stand-up Meetings

After the sprint planning, we started working on the tasks assigned and, in order to know about the progress, we arranged daily stand-up meetings. Since it was almost impossible to find a time to meet every day, we decided to have virtual stand-up meetings and created a Slack channel for this purpose. For instance, this was our stand-up meeting after the second sprint planning:

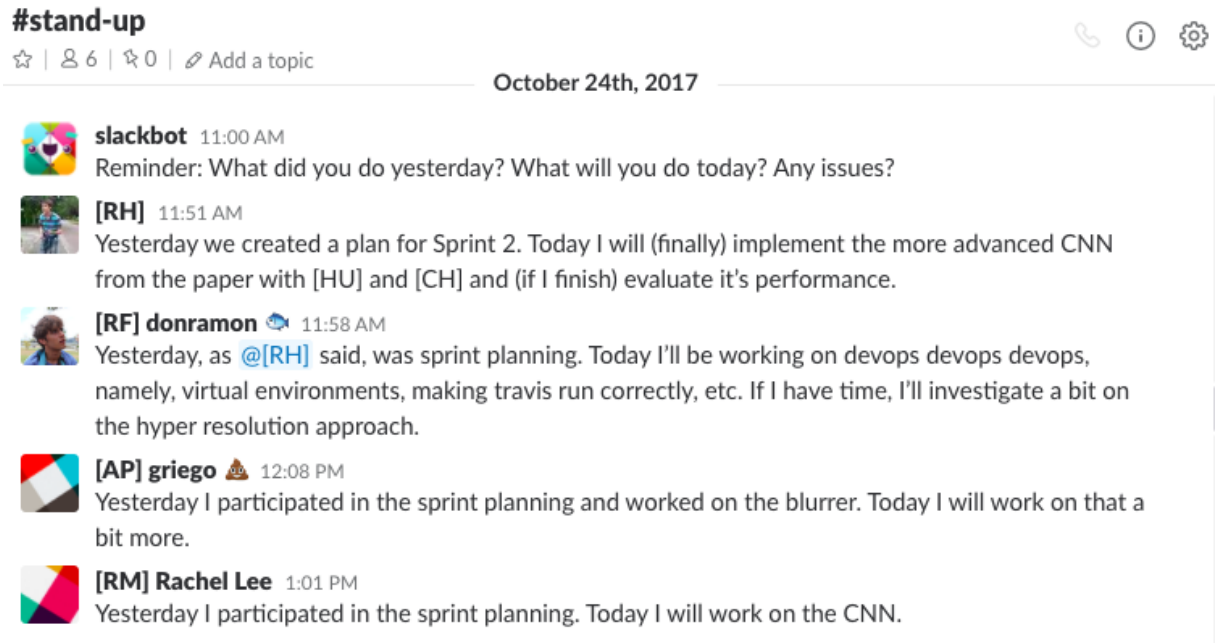


Figure 3.2: Virtual stand-up meeting on Slack.

We also tried to meet in person as often as possible and, in fact, most of the work was done in groups of two or three, we rarely worked individually. Apart from the **#stand-up** channel, we also had the following channels:

- **#general**, where we talked in more depth about the updates and the problems we encountered.
- **#github-alerts**, where we received all the notifications about a pull request being created, reviewed, approved or rejected.
- **#trello-alerts**, where we were notified when a ticket was created or moved.
- **#performance-metrics**, where we received the results of every TravisCI run.

After each stand-up meeting we would also move the corresponding Trello tickets.

3.3.3 Pull Requests

After moving a few tickets to the 'In Review' column, we would hopefully be in a good position to make a pull request. We tried to have one feature or bug fix per pull request. That was not always the case since we were often trying out completely different approaches in different branches. The master branch of the main repository was blocked and we worked in either forks or other branches.

In our commit messages, we always included the initials of the team members that had participated in the changes.

Figure 3.3: Example commit message.

3.3.4 Testing and Approval

Before approving a pull request, we would first of all look into our TravisCI builds and make sure that the build was successful. This meant that all the unit tests passed and that it was able to run the model on Azure and retrieve some results. We were not strict with coverage or style checks.

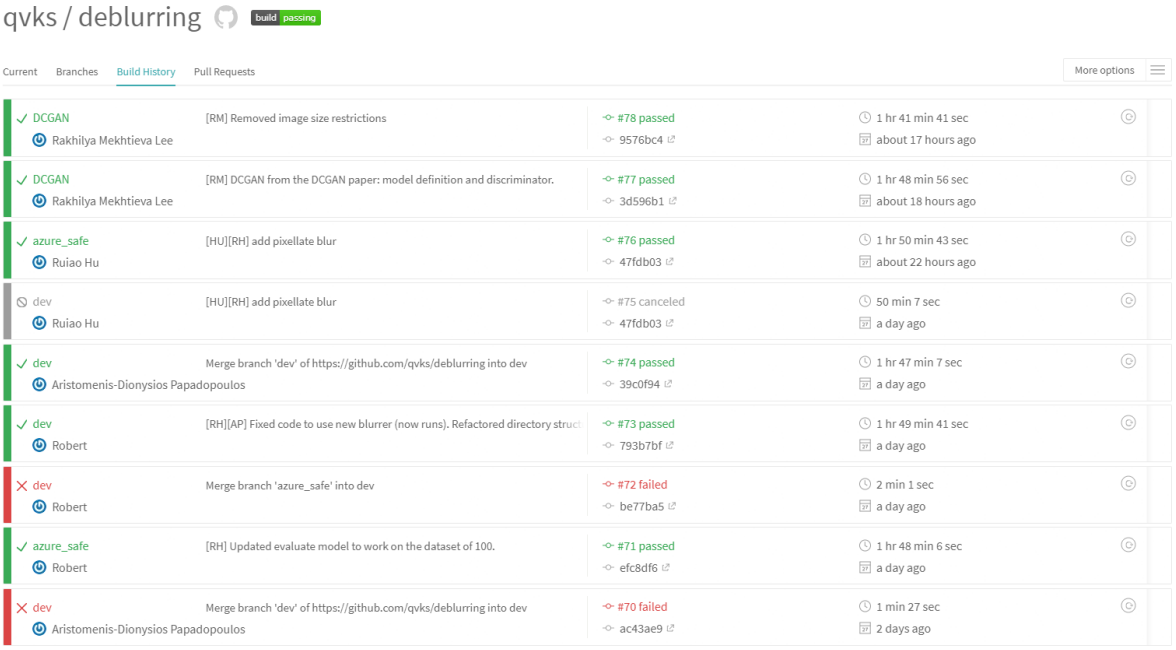


Figure 3.4: List of TravisCI builds.

Some of the results from the Azure run were sent to Slack, in particular, the learning rate (see Section 6). If this seemed promising we manually looked into the logs and judged the results subjectively by looking at the performance with seen license plates. Finally, we would test it with unseen license plates and make the final decision. This completes the development cycle, the reasoning behind which will become apparent after reading Section 5. This is shown graphically below:

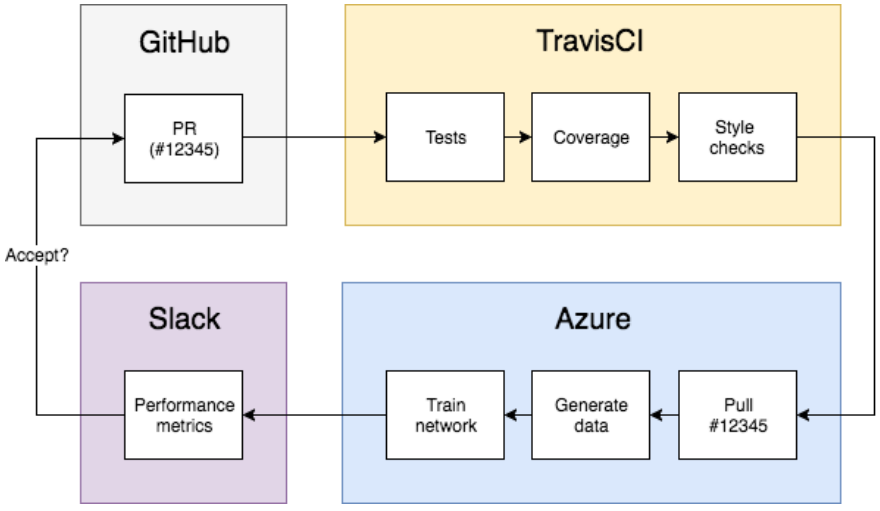


Figure 3.5: Development pipeline.

4 Initial Research

In this section we will explain in short some of the background theory, to which later sections assume the reader is accustomed. We further list a plethora of influences which affected our final product and describe some of the state-of-the-art techniques in the area of Machine Learning that we considered.

4.1 The Image Deblurring Problem

Almost all images are more or less blurry and hence we find that the image deblurring problem is fundamental to all of Computer Vision. Our aim in image deblurring is to recover the sharp, true image through the use of the blurred image alone. This is what is known as *blind deconvolution* and, unfortunately, blind deconvolution is not solvable without making assumptions on the input. Some information is hidden and, although it is present in the blurred image, we may only hope to partially recover it. Moreover, some information might be completely lost, therefore a good result can only be achieved from the contextual information, e.g. knowing that the image is a license plate. [16]

The image deblurring problem is the central topic of our project, we have evaluated possible techniques during the research stage based on how well they perform on our specific case. In our endeavour we have explored both using knowledge of the blurring process and of the original image as methods to reconstruct the true image.

4.2 Classical Image Processing

Classical image processing techniques are methods of manipulating images by treating them as matrices, where each entry corresponds to the colour value of a pixel. This allows us to apply matrix operations to them in order to deduce information and to alter them in desirable ways.

Very early into the project, we decided that approaching the deblurring problem from a purely classical image processing standpoint would not yield satisfactory results because of the nature of the images which we had been asked to deblur. These images are very small (around 50×120 pixels) and of fairy low resolution.

Figure 4.1: Examples of blurry license plates.

Therefore, as one can see in figure 4.1, they do not give enough information on the latent sharp image which we would want to reconstruct. Instead, we decided to focus on using Machine Learning techniques to tackle the problem as is explained in more detail in later sections. Classical image processing techniques were nonetheless used in this approach as is explained in detail in Section 5.1.

4.2.1 Discrete Convolution

This section discusses some basics of image processing using discrete 2-D convolution. Convolution by definition is the point-wise multiplication of two functions. When applied to discrete 2-D domains, the definition is as follows:

$$g(i,j) = \sum_{(m,n)\in\Omega_{ij}} \sum f(m,n)h(i-m, j-n).$$

Here Ω is the *convolution domain*, $f(n,m)$ is the *convolution kernel* and $h(n,m)$ is the function (here representing the image) on which the convolution is applied. We can consider convolution as the operation to replace each pixel with the weighted sum of neighbouring pixels, where the weights are specified by the kernel. From now on, 2-D discrete convolution will simply be referred to as *convolution*.

Below is an example of a convolution as found in [4]. The blue matrix corresponds to the input feature map and the green matrix corresponds to the subsampled output. The kernel applied to this matrix is:

$$\begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}.$$

The below collection of diagrams demonstrates how each entry in the output is generated from the input. The centre of the shaded region in the blue matrix is the point to be replaced by the weighted sum of the neighbouring entries enclosed by the region. The resulting pixel is the shaded green square. This operation is performed on all the entries in the feature map excluding the borders.

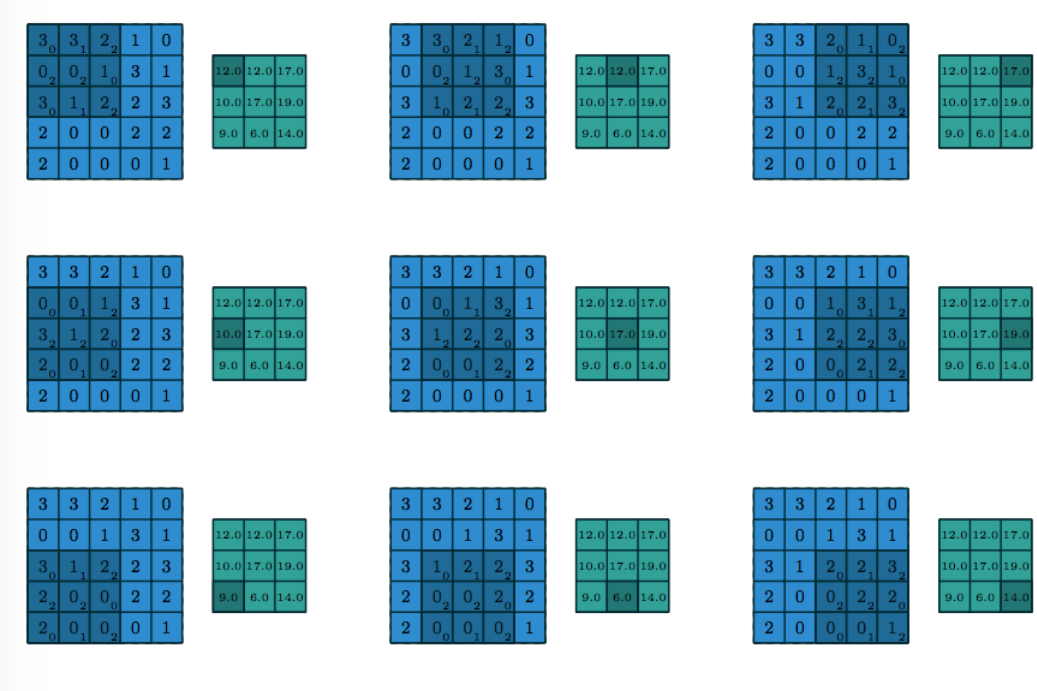


Figure 4.2: All steps convolution steps [20].

When the output of convolution is smaller than the input, i.e. the kernel does not overlay any region outside of the image, we refer to it as *classical convolution*. There are certain variations of convolution that behave differently on boundaries of the matrix. For example, to preserve the size of matrix, one heuristic is to pad the image with enough layers of zeros so that when we apply the convolution, all the entries can be multiplied by the kernel including the borders. This can be seen in the image below:

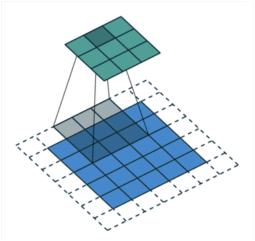


Figure 4.3: Padded convolution [4].

The effect of a convolution is completely determined by the convolution kernel. For example, matrices like the one below can be used for edge detection:

$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

However, for the purpose of image blurring, we want to consider the convolution kernel as the matrix representation of a *point spread function* (PSF) [19]. Considering images as matrices, a PSF describes how information of the central entry in the matrix spreads across the surrounding region. Hence it describes the distribution of pixel properties in neighbouring regions. For this intuition to make sense, we are only considering square kernels of odd sizes as the concept of central entry only exists in this set of matrices.

To understand how pixel information spreads to neighbouring regions, we can view the kernel as a probability distribution. This distribution is only defined in the region enclosed by the kernel and the magnitude of the entries in the kernel defines the likelihood of information of central pixel being present. Hence the kernel used for convolution are all normalised, i.e. the entries of the kernel add up to 1. An example of the 3×3 averaging blur kernel is given below.

$$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Knowing the PSF of a given problem means we can easily revert its effects by inverse convolution using a 2-D discrete Fourier transform. Nevertheless, this is usually not possible as an estimation of a PSF is an equally ill-posed problem as image deblurring, hence this technique is not explored in more depth.

4.2.2 Wiener Deconvolution

Given a PSF, the Weiner filter minimises the mean squared error between the estimated deblurred signal and the desired deblurred signal. The method works well without the

advent of noise but its performance decreases drastically as noise increases in the blurred image. An example of the optimal Wiener deconvolution effects are shown below:

Figure 4.4: Wiener Deconvolution [21]

In general, classical image processing techniques, including Wiener filter deconvolution suffer from the inability to adapt to added noise, which is present in most images obtained by low resolution cameras. Using this method was one of our initial experiments but due to the nature of the images it was not successful and it was one of the main reasons why we decided to use Machine Learning instead of classical image processing. Using this technique would further require approximating the PSF which has been attempted, for example in [8], however this paper describes an iterative way of approximating it which would not be suitable for our application in a practical sense, because of its computational cost.

4.3 Convolutional Neural Networks

Convolutional neural networks (CNN's) are a class of Neural Networks that has been effectively applied to tasks such as image classification or, in our case, image deblurring.

4.3.1 Principles of CNN's

Neural Networks are generally based on affine transformations where the input (a vector) is multiplied with a matrix to produce the output. Any input can be flattened into a vector before the transformation including image, sound and video information. [20]

A convolution that produces the convolution layers in a CNN is a linear transformation that preserves ordering. Convolutions are directly useful for our purpose as we are seeking to preserve the implicit structure of licence plate images. In convolutions, the same weights are applied to multiple data points in the input and only a few input data points contribute to an output data point.

A variation of convolution important to our further research for this project is *fractionally-strided convolution* [4]. The importance of this method lies in its image upsampling property, i.e. the output is larger than the input, which forms a basis for the image generation model discussed in the next section. Upsampling is achieved by expanding the input space by surrounding every pixel with zeroes, which makes the output space larger than the original space.

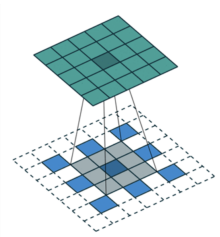


Figure 4.5: Padded fractionally-strided (or transpose) convolution [4].

4.3.2 Current Models

We looked at breakthrough changes in models of the previous winners of the annual ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). Even though our problem is not image classification, these models were capable of extracting information out of images and thus held interesting information for us.

4.3.2.1 ResNet

This was the winner for the ILSVRC 2015 classification task. Its importance came through as it was able to train a 152 layer net and achieve a 3.57% classification error rate. The depth of this model was impressive and they were able to overcome the critical issue of overly saturated accuracy as the depth of the model increased. Their experiments led them to verify that naively increasing the depth causes a higher test error so it is not as easy as stacking more layers to make a better network. They proposed a residual block architecture to combat the *degradation problem* (accuracy getting saturated as depth increases).

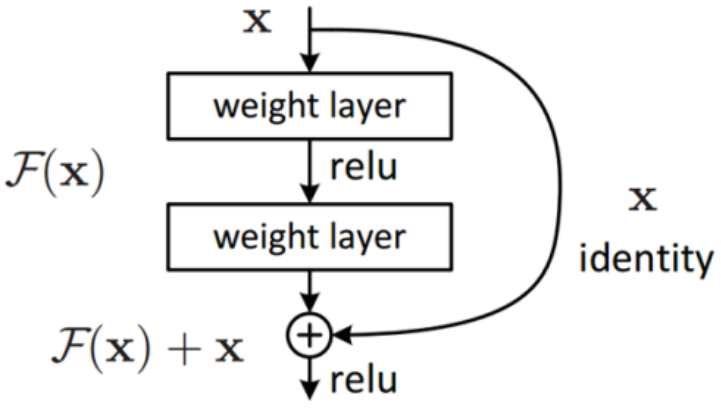


Figure 4.6: A residual learning building block [14]

By stacking many residual blocks they were able to obtain a better solution by explicitly learning the residual mapping. Their results back up their claim that this new way of adding layers does not suffer the degradation problem found in plain networks as depth increases. [14] However, due to concerns on the training time we could afford previous to the competition deadline, we did not decide to incorporate this into our final design.

4.3.2.2 VGG

This model performed exceptionally well in the ILSVRC 2014 classification task. Unlike the previous model, it achieved a 7.3% error rate with only 16 layers. Their proposal was to use small 3×3 filters so that computation would be more manageable than previously used larger filter sizes. Incorporated in their reasoning was that many stacked small filters would still have the same receptive field as one large filter. For example two 3x3 filters would have an effective receptive field of one 5x5 filter, with fewer parameters. [18] This model advocated that simplicity and depth was a winning combination. Additionally we used this model as an incentive to scale down our filter sizes as much as we were capable of whilst keeping them big enough to shrink the inputs down to a code layer in a handful of layers (see Section 5.2).

4.3.2.3 Inception

Inception set the new state-of-the-art standard for classification and detection in the 2014 ILSVRC. It attempted to increase the depth and width of the model whilst also keeping the computational budget low. This was achieved by the ubiquitous use of dimensionality reduction prior to expensive convolutions with larger patch sizes [9]. Dimensionality reduction was achieved through simple 1×1 filters. Although it helps remove computational bottlenecks, the lower dimensional information is in a dense, compressed form and compressed information is harder to process in a CNN. In Section 5.2 we explain the idea of dimensionality reduction but does so spatially by reducing the height and width of the image contrast to reducing the size of the feature space.

4.4 Generative Adversarial Network

Convolutional neural networks can be used not only for image classification, but also for image generation. GAN (Generative Adversarial Network) is a recent approach which, given a dataset of images, attempts to expand it by creating new images like those in the dataset.

4.4.1 The Generator and Discriminator

The goal is achieved by training two networks: the Discriminator and the Generator, simultaneously.

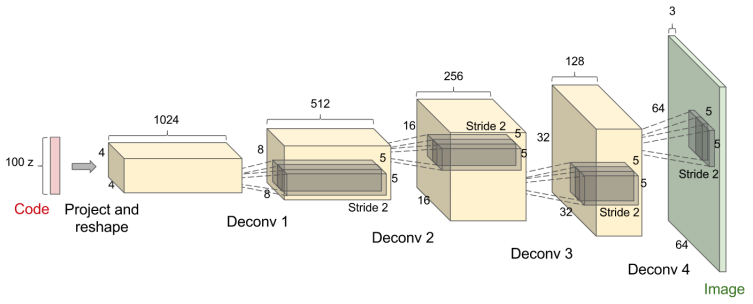


Figure 4.7: The generator network [2].

The Generator (G) is trained to create new realistic data. It takes a vector that describes an image and outputs an image. One advantage of having a vector as input is that we can perform vector arithmetic on it in order to 'subtract' data with irrelevant features and 'add' data with desired features, producing realistic images not present in the original dataset. The original model described in [1] uses fractionally-strided convolutions, as explained in the previous section, to upsample images, therefore, as illustrated in the above diagram, every output layer is larger than the previous layer.

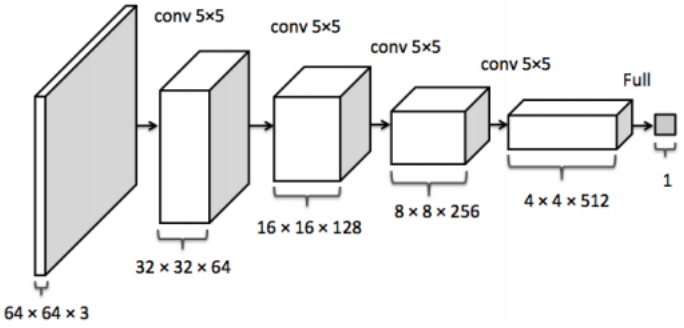


Figure 4.8: The discriminator network [1].

The Discriminator (D) is trained to tell the real data apart from the generated data, thus challenging the Generator (G) to improve the quality of the newly generated images. It takes an image as input and returns the probability p of it being real where $0 < p < 1$, denote this as $D(image) = p$.

4.4.2 The Minimax Game

The Generator has to produce images that fool the discriminator into believing they are real. Therefore, the Generator aims to maximise $D(G(z))$ while the Discriminator aims to maximise $D(x)$ for x in the original data and minimise it otherwise. Therefore, training adversarial networks is done with the following minimax game as defined in [2], where D and G are trained by taking the gradients of this expression with respect to their parameters:

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}} log D(x) + \mathbb{E}_{x \sim p_z}[log(1 - D(G(z)))]$$

4.4.3 Convergence Limitations

In image generation that was required in our project, the main problem to avoid is D converging much faster than G, in which case the images produced are not of satisfactory quality. In [15], updating G twice for each D network update is proposed, however, for the purpose of our project this was still not enough, as by doing that we generated the following output:

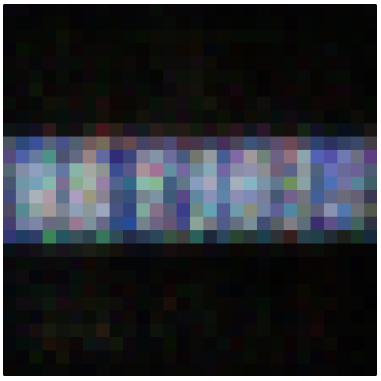


Figure 4.9: A 'licence plate' generated by G.

We have therefore concluded that more research is needed to effectively use this technique in our project, as explained in Section 8. We attempted a more crude method of generating our own license plates at 5.2.

5 Design and Implementation

In this section we describe our final approach that we developed during our research period, and refined over the implementation.

5.1 Data Generation

Having abandoned the prospect of using classical image processing techniques to solve this extremely ill-posed problem early on we adapted, as described above, to a CNN approach. To this end, we focused our attention on generating artificially blurred images that resemble those in the testing data set to use as training data. This involved a lot of trial and error, as it is crucial that the artificial blur is realistic enough; otherwise the network performs poorly on real data. Subsequent sections will refer to the following license plate image to demonstrate the effects of blurring algorithms:

Figure 5.1: Example license plate image

Note that the image above is from the 4000 clean images data set, to which we will henceforth refer as the *training set*. On the other hand the images in Figure 4.1 are all from the 100 blurry images data set, to which we will henceforth refer as the *testing set*.

5.1.1 The 'corrupter'

The *'corrupter'* is the image processing core of our project and it uses a combination of standard methods, based on a classical image processing approach as introduced in Section 4.2. The main goal of the corrupter is to, given an image from the training set, produce an image that resembles as closely as possible the images in the testing set. To replicate blurred images seen in the testing data we were given, we settled with two types of blurs and a series of other transformations under the assumption that every natural corruption can be constructed from a specific composition of these. The complete list of methods used is:

- Gaussian Blur (5.1.1.1)
- Motion Blur (5.1.1.2)
- Pixelation (5.1.1.3)
- Reshaping (5.1.1.4)
- Histogram Equalisation (5.1.1.5)
- Brightness Variation (5.1.1.6)

Each of these 'corruptions' are explained in detail below.

The output image of the corrupter depends on parameters given by random draws from suitable probability distributions for each transformation. When we train our model, these random parameters clearly will differ from image to image. The reasoning behind this was that each corruption of the entire data set (corresponding to the data given in

each epoch of the training) would be essentially unique and the network would not be able to overfit on the data.

As far as the order of application of these transformations in the corrupter, note that the convolution operation is *commutative* and *associative*. Hence the ordering of the blurs using convolution in the composition of transformations does not affect the final result. Although *histogram equalisation* and *perspective transformations* do not commute with the other transformations, we apply them in a consistent order, which maximises their effect, by changing the brightness first and then changing the contrast. The effect of applying all these transformations is shown in the figure below:

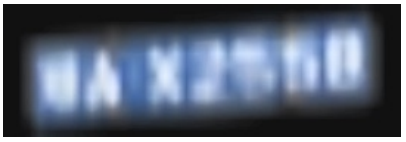


Figure 5.2: Corrupted license plate by the main corruption type.

We quickly came to the conclusion that this blur was 'too general' and it would not be sufficient to realistically approach the blur present in some of the images of the testing set. It would therefore be impossible to deblur all the images in the testing data set, as even though most images resembled some combinations achievable by our corrupter, there were specific images which showed extreme blurs. For example:

Figure 5.3: Unachievable blur by the main corruption type.

We thus had to adjust our corrupter to produce specific blurs a percentage of the time. In the end our blurrer had 5 different blur procedures, which were applied in a manner consistent with what appeared in the testing data. The following table shows the distributions of the various types of blurs applied to the training set:

Corruption type	Usage %	Example
Main Blur	88%	
Extreme Brightness Variation	3%	
Extreme Pixelation	2%	
Extreme Motion Blur	5%	
Mild Blur	2%	

Table 5.1: Different types of corruption.

The reasoning behind most blurs mentioned above seems intuitively clear, with the main exception potentially being the *mild* blur. We decided that producing some images which were close to the original ones was necessary to avoid overfitting on blurred images. The problem of overfitting is discussed in more detail in Section 5.2.3.

The percentages of the time that each blur is applied as well as the ranges of the parameters for each of the blur procedures were developed subjectively, mainly by examining the distribution of blurs in the testing set. We noticed much better performance in the extreme cases when using this specific blurs approach without sacrificing the performance in the general case.

In the following subsections we delve into the details of each specific transformation.

5.1.1.1 Gaussian Blur

Gaussian blur is the simplest of the two blurs we used to approximate natural blurs. Essentially the values in the kernel matrix are given by a Gaussian distribution. The values in the kernel are produced, informally, in the following way:

1. The value in the centre is the probability of the mean, so the highest probability in the distribution.
2. Values in the other cells are given by considering their distance from the centre

and taking the probability of a value that many standard deviations away from the mean.

To parametrize a Gaussian kernel, we need to model a Gaussian distribution as well as determine the magnitude of the blur, this gives a total of three parameters, two for the Gaussian distribution and one for the magnitude. However we only need 2 parameters, namely the standard deviation and magnitude. This is because we assume that the mean of distribution is centred at the origin so that in matrix representation, the mean is central entry of the matrix. Standard deviation is reflected in the entries of the kernel while the magnitude is dependent to the size of the kernel. An example Gaussian kernel is shown below.

$$\frac{1}{273}\begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix}$$

3×3 Gaussian kernel with standard deviation $\sigma = 1.0$

And the effects of applying the above kernel to 5.1 is

Figure 5.4: Gaussian blur on the example license plate.

The result of the Gaussian Blur is, as one can see an overall 'smoothing' of the image, reducing detail. For this reason we tried to keep Gaussian blur to a relative low, using it not so much as an effect that reduces the image quality (as for this reason we used pixelation) but to help the algorithm learn to distinguish 'hard' edges from smoother ones.

5.1.1.2 Motion Blur

Motion blur is slightly more complicated. In general there are two parameters which need to be considered when dealing with such a blur. Intuitively, these are the magnitude and the direction of the blur. To replicate motion blur using a blur kernel we start by constructing a matrix of zeroes. We then replace the entries of a specific row with 1's (usually the middle row), to imitate the effect of the blur. Finally we rotate the matrix by a specified angle and normalise it. [13]

The kernel of such a blur represents the point spread function which describes how the information of central pixel spreads the local, neighbouring, regions. Non-zero matrix entries represents the valid 'local direction' of the blur, while the global direction of the sequence of non zero entries represents the global motion blur direction. An augmentation of the established motion blur kernel is to increase the number of adjacent rows that are

filled with 1's before rotation and normalisation.

The angle of rotation of the kernel (measured from the positive x-axis anticlockwise) gives the direction of the blur. The magnitude of the blur is determined by the size of kernel. This follows from the intuition that by increasing the size of the kernel we increase the number of non-zero entries in kernel matrix, thus increasing the spread of the central pixel. Hence larger kernels with more non-zero entries represent a blur of greater magnitude. For example, consider the following matrices:

$$\begin{pmatrix} 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0.04 & 0.07 & 0 \\ 0 & 0.04 & 0.07 & 0.07 & 0.07 \\ 0.04 & 0.07 & 0.07 & 0.07 & 0.04 \\ 0.07 & 0.07 & 0.07 & 0.04 & 0 \\ 0 & 0.07 & 0.04 & 0 & 0 \end{pmatrix}$$

Standard kernel with $\frac{\pi}{2}$ rotation

'Wide' kernel with $\frac{\pi}{4}$ rotation

The result of applying larger versions of the above kernels to test image 5.1 is shown below:

Figure 5.5: Motion blur with kernel of size 9 and rotation $\frac{\pi}{2}$.

Figure 5.6: Motion blur with kernel of size 11 and rotation $\frac{\pi}{4}$.

The result of motion blur is a bilateral perturbation of pixel information which creates lasting artefacts in the region where perturbation 'transports' the pixel. This blur is fundamental to our blurring process as it is dependent on multiple real world parameters extrinsic to the video system.

5.1.1.3 Pixelation

One additional technique we used was *image pixelation* to achieve the low resolution of surveillance cameras. The effect of pixelation can be seen below:

Figure 5.7: Pixelation transformation reducing the size by 4.

This effect is achieved by linearly reducing the size of the image and then restoring the smaller image to its original dimension. The effect of this is that information is lost, when

the image is minimised and this information has to be extrapolated in the restoration procedure, which effectively ends up pixelating the image. The parameters governing pixelation process are the image resizing factor and the extrapolation process.

Intuitively, the magnitude of the resizing factor is the magnitude of loss of information from the original image. As far as the type of extrapolation is concerned, there are 4 different types readily available for us to use as part of the `OpenCV` library which are *nearest point extrapolation*, *linear extrapolation*, *quadratic extrapolation* and *cubic extrapolation*. Essentially, these are nothing more than polynomial interpolations in a 2-D discrete domain setting. Hence the quality of extrapolation is lowest using nearest point extrapolation and highest using cubic extrapolation. For our blurring process, we chose the linear one as it gives the most accurate approximation of the images in the testing data set.

To achieve more aggressive pixelations we also tried removing random pixels from both the reduced and final image. Further we considered adding noise to reduced images, but this intuitively means that regions which are extrapolated from the noise are affected with a linear order of degrading, due to the choice of 'linear' extrapolation method. Combining this with removing random pixels after the extrapolation allows us to have further control of region dependent pixelation. However the result was not realistic enough hence this approach was ultimately discarded.

5.1.1.4 Reshaping

Additionally we considered *perspective transformations* which involved changing the relative sizes of parts of the images while maintaining straight lines. Such transformations were used to recreate the effect of a licence plate not being centred in the image. This was crucial, as the testing data consists almost entirely of license plates obtained by surveillance cameras are rarely in perfect focus, since the cars are not aligned with the camera. Similarly, we also resized the license plate in order to allow pictures taken from different points of view. The following are examples of reshaped license plates:

Figure 5.8: Pixelation transformation reducing the size by 4.

It should be noted that when resizing part of the background would end up being empty and thus the part of the code responsible for reshaping also had to take care of choosing a random background colour.

Finally, it is important to mention that these transformations were not only present in the corrupted images but also in the goal images. Initially, we tried to run our algorithm with the goal images unrotated and in their original sizes, in order to learn the rotations and the resizing along the blur. It turned out that this was too hard too learn, so we decided to focus solely on the blur, this approach was justified by the fact that current OCR systems work fine with slightly rotated text.

5.1.1.5 Histogram Equalisation

Also, we applied variants of *histogram equalisation*, which are transformations used to increase contrast in our goal images. This mimics over-exposure in the camera taking the picture. [3]

Figure 5.9: Histogram Equalisation to intensify contrast.

5.1.1.6 Brightness Variation

We also employed some methods of *brightness variation* of the images to deal with low-light images. This is done simply by uniformly changing the intensity of all pixels in the image. This effect is shown in the figures below:

Figure 5.10: Brightness increase and decrease.

We do not consider non-uniform variations of intensity as this is rarely present in real world data, since such an effect would be specific to a camera and the local lighting. Essentially we considered this effect as part of the 'natural noise' present in the images of the training set.

5.1.2 Data set augmentation and refinement

The training, *clean*, dataset provided by Huawei had many very noisy images which we removed by eye. An attempt was made to remove noisy images automatically by objectively judging the degree of blur in them. This algorithm also utilises convolution with a specific class of kernel known as the Laplacian kernel [7]. The 3×3 Laplacian kernel is shown below:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

The 3×3 Laplacian kernel

The Laplacian kernel is an example of a 2^{nd} order derivative operator which is mostly used for edge detection. As this kernel has low response on regions of shape changing intensity, we wanted to consider the variance of this response, since it describes the variations in the sharpness of regions in the image. Low variance means the edges are sharp, which we can intuitively understand as a lack of blur. High variance on the other hand represents a

blurry image. This method of removing noisy image proved impractical in the data sets we were given because of the size and the resolution of the images. For such small images the variance of the Laplacian kernel was almost uniform for both the testing and training data.

Additionally, the training set also contained a small portion license plates with black text on yellow backgrounds that we deemed an unnecessary complication in the training phase. Overall 2622 images from the original 4000 were removed, keeping the least noisy and most desirable as goal images (cf. Figure 3.3).

However, it should be noted that some 'natural' noise being present in the images that are blurred is absolutely beneficial. We attempted to train the network on completely artificially generate number plates and it was incapable of recognising characters in real images since there was intrinsic noise from the camera that it had not experienced during training.

As mentioned we experimented extensively with using artificially generated images. These were of two sorts:

1. Images constructed from high resolution parts, found online.
2. Images constructed from the 1378 images in the 'clean' data set.

The process of image generation was fundamentally simple, as we evaluated the ratios of each character in the license plate and proceeded to split the license plates in these points. We then tried to construct an alphabet of all of the characters that we had obtained in this way and from this library build any possible license plate. Below are some examples of these *Frankenstein* license plates:

Table 5.2: *Frankenstein* Images

One of the initial reasons behind this approach was to be able to have uniformly distributed characters in the license plates, in order to prevent our network overfitting by constructing correlations between characters and their positions in license plates. This was a particularly important issue as the amount of license plates we had to train was not extremely large and the license plates repeating in each epoch would make this a valid possibility.

We soon discovered that by training on 100% generated images we would be unable to get satisfactory results as our network would be unable to handle the natural noise present in all of the license plates. So another approach we tried was reconstructing each license plate in the training data, blurring the real image and having as goal the artificial one. This also resulted in some overfitting so we abandoned this approach.

K-means clustering was used (though not in the final training of the model) to reduce the number of colours in the image of the license plate. This could remove small amounts

of noise from the image but exacerbated larger artefacts (such as the edge of the metal plate) and proved too unrealistic.

5.2 Our Neural Network: Moussaka

The inspiration behind Moussaka is to reduce the dimensionality of the license plate as it propagates through the network as was done in 4.3.2.3 model. It then attempts to recreate the ground truth (deblurred) license plate from this lower dimensional representation. 'Moussaka' is a traditional Greek dish formed by many layers, just like our neural network.

The dimensionality reduction restricts the amount of information the network can use to reconstruct the image. While this may sound counter-intuitive, it forces the network to be *picky* about what information it retains. More of the ground truth image can be reconstructed from an encoding in which characters are likely to be present than from information about noise and artefacts in the image. It is also brought to our attention that the images should be able to be represented by some subspace as they will all be restricted to the same letter and number characteristics. Thus, the network learns to discard noise, encode the the image in a dense representation and deblur the license plate.

5.2.1 Architecture Overview

A diagram of this architecture can be seen below in figure 5.11 which should be used for reference in this section. We chose to use kernels, or filters, of size 5x5 for every layer. This choice was inspired by theory from VGGs (see 4.3.2.2) that suggests that smaller kernel sizes give better results.

Firstly the image passes through two convolutional layers (*CL1*, *CL2*) that reduce the size of image to $\frac{1}{81}$ of its original area. This achieves the aforementioned dimensionality reduction. Additionally, each layer applies multiple filters to the propagating image resulting in multiple 'channels'. Consequently, the filters in the first can become specialised to look for certain features such as corners or edges in the image. The filters in the second layer can then use the edge and corner information to find characters.

At the centre of the network is the Code Layer. We observed that the image at the Code Layer is often a black background with some white pixels. Moreover, different number plates produce Code Layers with white pixels in different places. Thus, we concluded that this is the network's internal representation of the license plate. In training, if we see high variation in this layer between number plates we know that the network has successfully extracted high level information from the number plate. This is demonstrated below:

Number Plate			
Code Layer Representation			

Table 5.3: Demonstration of variance in Code Layer

We suspect that this layer contains information about the license plate's orientation and characters, but this is impractical to confirm since this representation only has meaning to the subsequent layers. We do know that at this point it has discarded noise, since none of the noise in the blurry input is present in the output deblurred image.

The image then undergoes transpose convolution in the layers *TCL1* and *TCL2* (for an explanation of how these work see 4.3.1). These also have stride 3, so increase the size of the images at the Code Layer to that of the original image. Here, low level features are created from the logical representation in the Code Layer. In essence, these layers can 'draw' a Chinese character or letter using the logical representation and (learned) contextual knowledge of the alphabet. The final layer adds colour and the deblurred image is output.

Below is a diagram detailing this process. The image propagates through the network from left to right.

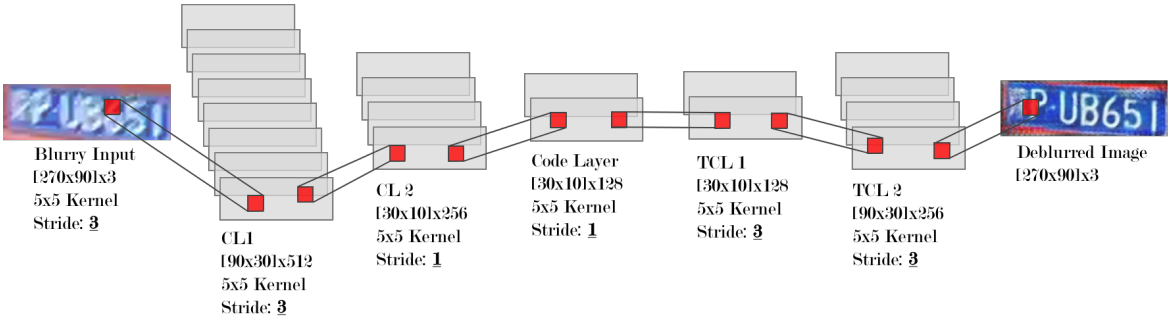


Figure 5.11: Moussaka CNN Diagram

5.2.2 Image Propagation

The step by step propagation of an image through the network is described in greater detail below:

Step	Description	Image
1	The blurry image is first resized to [270x90], though has three colour channels (RGB) so actually has dimensions [270x90]x3.	
2	The first convolutional layer, *CL1*, applies 512 filters with stride 3 resulting in 512 channels. Each filter application results in a [90x30] image. The first layer learns to extract low level features such as edges and corners.	
3	*CL2* applies filters resulting in 128 output channels. A stride of 3 reduces the size of the images again to just [30x10], or $\frac{1}{81}$ of the original area.	
4	The Code Layer represents the networks interpretation of what's in the license plate. This is the highest level of information, and is hard to interpret since it's only meaningful to the network. A single filter (essentially a piecewise mapping) is applied to each channel with a stride of 1, preserving the size.	
5	From just these images the network begins to reconstruct the deblurred license plate with Transpose Convolutional Layers (see 4.5). *TCL1* uses a stride of 3, increasing the image size to [90x30] with 256 channels.	
6	*TCL2* does the same as *TCL1* though this time it aggregates those 256 channels into just one channel. It also serves to recolour the image.	
7	This 'one' channel has dimensions [270x90]x3 and is the final, deblurred license plate.	

Table 5.4: Propagation of a blurry image through Moussaka

5.2.2.1 Activation Functions

The last layer employs a tanh activation function, meaning the final output is passed pixel-wise through tanh before being returned. We do this so that the output is constrained in the finite range $[-1, 1]$, meaning the final image will always have defined RGB values.

ReLU, or rectified linear unit, is our activation function for every other layer. This means that the output of each layer is passed pixel-wise through the ReLU function:

$$ReLU(x) = max(0, x).$$

Without ReLU the network does not converge to a low cost and performs poorly even on the training data. This is because a network can only learn non-linear functions (which we have observed the deblurring process to be) if there are non-linear functions present within the propagation.

5.2.3 Optimisations

The final model architecture we settled on was a rather simple one, which is in resemblance to the VGG model described in 4.3.2.2, but it gave the best results in comparison to all other architectures we tried. Despite that, we considered the results unsatisfactory and employed a selection of optimisations to augment our algorithm. These are all explained below.

5.2.3.1 Dropout layers

During our initial experiments with the architecture we noticed the model had become dependent on the artificial blur techniques we used. Over-fitting was an issue made possible by the massive amount of parameters describing the network. To reduce the over-fitting and prevent the units from too much co-adaptation, we added dropout layers on every layer except the input layer, with a drop out rate of 50%.

With dropout, sections of the network cannot become totally reliant on other sections of the network. This enforces redundancy in the network making it more robust and reduces the effect of over-fitting.

5.2.3.2 Generative pre-training

The results published in [10] suggest that a deblurring convolutional neural network will perform better if it is initialised with weights given by a known blind deconvolution method such as the Weiner filter instead of a random initialiser like one proposed in [11]. We tried to improve our result in a similar fashion by reasoning that the goal image should be similar to the initial image. Additionally results shown in [5] further show that pre-training leads to lower test errors and takes greater effect as the size of the network grows.

Since we intended to make a deep network we built on this research and added a pre-training phase. This consisted of greedily training the network layer by layer by having the previous layer as the output goal for each new layer and feeding the network many blurred licence plates as inputs. The structure formed is known as a deep belief network [6]. The resulting generative model is then subject to fine tuning through normal training means after this initial set of unsupervised learning.

5.2.3.3 Decreasing learning rate

Initially we used a constant learning rate for gradient descent of 0.001. Though this is small, occasionally in the later stages of training the network would learn something wrong. Since the learning rate is constant, the effect of the bad batch on the network weights is the same as batches at the start of training. This effectively overwrites the progress of the network.

To avoid this, we use an exponentially decaying learning rate so that high level concepts can be learned at the start and refined with smaller details in the later stages.

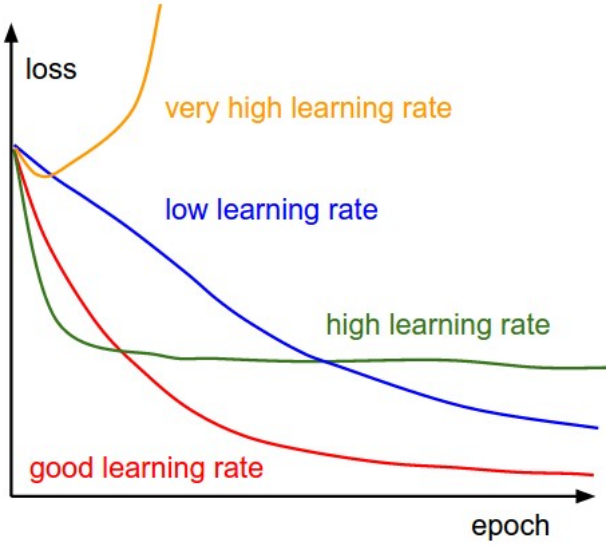


Figure 5.12: Learning rate choice effect on cost [17].

5.3 Training

All the models were trained on Azure. We were using machines specialised on Machine Learning and Data Science. More specifically, we used NC12's one of the virtual machines in Azure's NC-series. Its specifications are:

- 12 cores CPU (E5-2690v3),
- 2 × K80 GPU in one physical card,
- 112 GB of memory and
- 680 GB SSD of disk capacity.

Initially, we were using even more powerful virtual machines but we quickly realised that we were not taking advantage of them so decided to downgrade and be able to have more, as there is a 48 cores limit, in order to run several experiments at the same time.

Regarding the cost, we had 3000$ of Azure credit in total. We had 4 machines and each costed around 1.5$ per hour so we know that, as long as we turned off the machine when we were not using it, we would not run out of money.

The model submitted to the competition trained on an NC12 for 18 hours.

6 Results and Evaluation

6.1 Analysis of the Network Optimisations

In the figure below we can see a direct comparison of the moussaka model across different runs where each run may only be missing at most one optimisation.

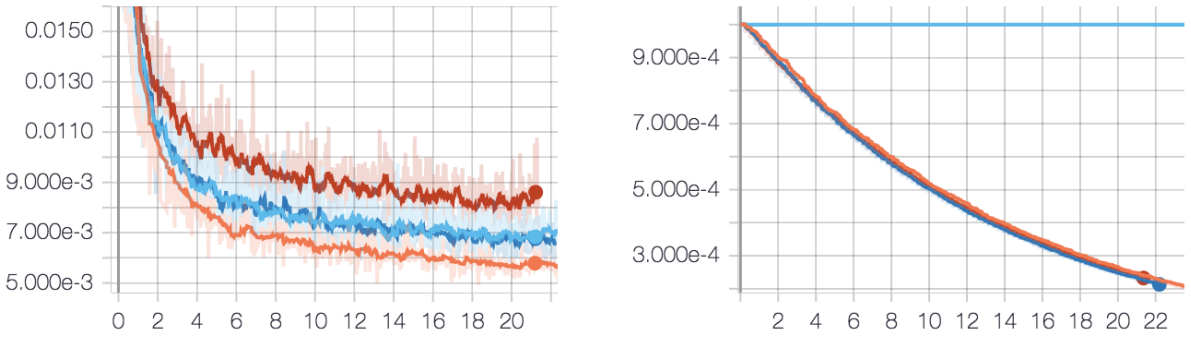


Figure 6.1: Direct comparison of the model with and without the optimisations.

The graph on the left of Figure 6.1 shows how the test error decreases over the relative time since each model has started. The test error is the pixel-wise L2 loss between the blurred input image and the ground truth image. The graph adjacent to it in Figure 6.1 shows how the learning rate decayed for each model over the same period of time. The graph pits three different optimisations against each other together with the run containing all of them. Of these the experiment labelled 'no selection' indicates the run that used all 4000 images provided by Huawei instead of the selected few we chose to use. This meant that each epoch on this run took longer as it ran over more images.

Colour	Experiment	Cost	Epoch	Time
●	All	5.5336e-3	3629	21 hr
●	No decay	6.6555e-3	3777	21 hr
●	No dropout	6.9685e-3	3760	21 hr
●	No selection	7.9423e-3	1252	21 hr

Table 6.1: Numerical results of the direct comparison.

Firstly it should be noted that the network converges quicker and to a smaller cost for the training images we hand picked, highlighting the importance of cleaning our data. Additionally, adding dropout slows the rate at which the network learns and causes it to converge at a higher test cost than our final run; however the great benefit of dropout becomes apparent on the unseen data below. Similarly, without decaying the learning rate we found that we had a slower convergence rate and higher test cost. This is best highlighted by the first row in Table 6.2 where the output is not so clear in the dropout column as it has not adapted well to the natural noise present in this image. Also in the second row of Table 6.2 the '0' becomes confused with an '8' if a decaying learning rate is not part of the experiment. We hypothesise this is due to the fine tuning provided by the lower learning rate as the solution converges. Below we present a visual comparison of network optimisations on three different unseen license plates.

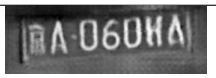
Blurred	No drop-outs	No selection	No decay	All
				
				
				

Table 6.2: Comparison of network optimisation on unseen data

6.2 Grey-scale vs. Colour

Usage of coloured input and output increases the complexity of our neural network and although extra information can be gleaned from the colour we felt that this was not necessary in creating a sharp reconstruction. Table 6.3 below validates our intuition as the quality of the reconstructions in each model is very similar. The much greater number of parameters in the colour model causes the training time to be much longer and the cost to converge much slower than the simple grey-scale model. It also adds challenges to our cost function which was previously the sum of pixel wise differences squared. This function is still valid in coloured images however colour models (RGB, HSV etc) need to be considered.

Comparing the de-blurring results using black and white images and coloured images, subjectively, we found that the grey-scale model produces clearer results. However since the OCR software used to automated the judging process is trained with coloured images, it performs better on coloured images. We note that our findings may only be a limitation of the time we had to train both of these models.

Blurred	Grey-scale	Colour
		

Table 6.3: Comparison of colour and grey-scale networks.

6.3 Overcoming the Issues with the Validation Tool

Huawei sent us their validation tool, which provides us with the accuracy of our algorithm on the testing data. This tool used their model trained to output text from an image of a license plate. Then this text was matched to the actual result, which they had obtained. Our accuracy was calculated by dividing the number of characters were read correctly from our deblurred images by the total number of characters. An important thing to note is that the tool did not take into account Chinese characters.

This tool would have been extremely useful for our development process since it would have given us clear objective results of the performance of each model. Unfortunately, due to formatting issues we were not able to use it. We contacted Huawei as well as

members from the other teams and everyone seemed to be having the same issue. Finally, the solution given by Huawei was that we could send them the deblurred testing images via e-mail and they would get back to us with the accuracy within one working day. They also included the text they read from our images, which was useful to understand where our algorithm was performing badly.

In order to mitigate the lack of validation software, we tried several alternatives. Firstly, we put considerable effort in trying to re-construct the file they had given us. Then, we tried generic OCR libraries, which could not read even our clean data. We also tried OCR libraries that were specialised on license plates. These, however, were trained on European and American license plates and did not recognise most of the characters in our set. Again, this shows how specific this problem was. Finally, we decided to rely on the cost function, a subjective evaluation of the results on seen and unseen data, and Huawei's response to the results we sent them.

We sent four batches of deblurred images to Huawei. The first one, three days before the deadline, which gave us some confidence in the model we had chosen. We did some minor modifications judging from the characters the algorithm got wrong and sent the other three batches on the final day before the deadline at different stages of the training to make sure it was improving and not overfitting. These are the results we received:

Time	**Accuracy**
17/11/2017 18:56 GMT	62.12%
20/11/2017 02:32 GMT	72.53%
20/11/2017 07:13 GMT	72.69%
20/11/2017 09:20 GMT	73.54%

Table 6.4: Accuracy results.

We left the last model training for some more hours (it trained for 18 hours in total). The final accuracy was not sent to us but we were told by the Huawei experts that it was over 75 %.

7 A Product Prototype

We wanted to wrap our model in a usable product so we built BCLPD (Blue Chinese License Plates Deblurrer), an Android application that allows us to showcase our algorithm in a real world situation. The main features are:

- Taking a picture or uploading it from the gallery.
- Cropping the picture to select the license plate.
- Deblurring the license plate offline.

We have kept the interface as simple and intuitive as possible.

The first thing the we see when we boot the app is a view with two big buttons to choose between taking a picture or uploading it from your gallery.

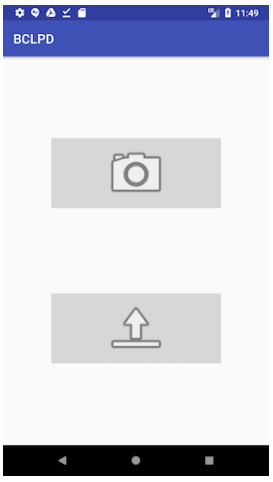


Figure 7.1: BCLPD main view.

Once a picture has been selected or captured, the crop view appears where the user can select the region containing the license plate in the picture. As the area selected changes the view zooms in or out to give the user more precision.

Figure 7.2: BCLPD crop view.

Finally, we move on to the final view where the cropped section is resized and run through our model. There is a circular progress bar animation and the image fades into black representing 'the black box'. Finally, the user can simply save the picture to the gallery.

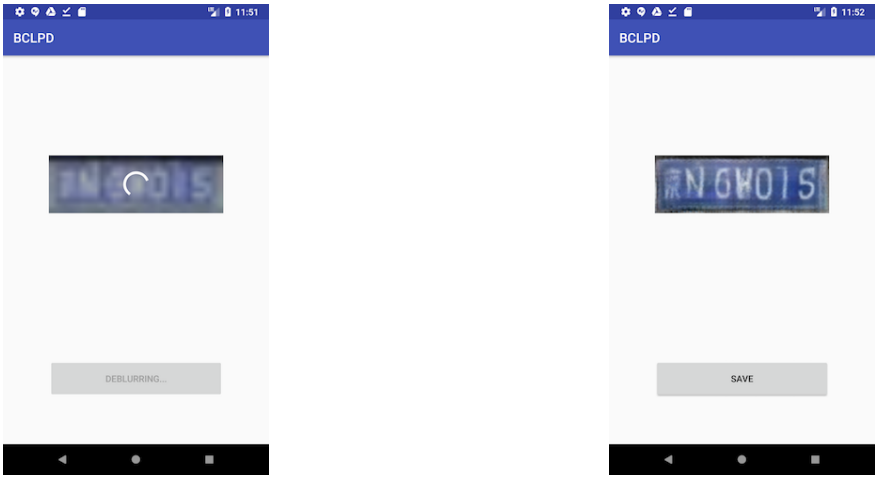


Figure 7.3: BCLPD result view.

In order to run the model offline, we use Tensorflow Mobile. We use Moussaka's trained model on coloured images. In order to boost its speed, we optimised the graph containing the weights. This is a common practice when putting highly resource-consuming Machine Learning programs in mobile or embedded systems.

It should be noted here that this application is in its current state meant to be demonstration of our algorithm and nothing more. As it is not yet meant to be used by the general public and was not the main deliverable of our project we have not had it tested by users and we have not published it for general use.

To sum up, BCLPD serves as a proof of concept of our algorithm. We believe that showing how our model performs using this applications is more appealing than running a command on a terminal and can inspire the audience by making them think of the possibilities of image processing with Machine Learning in a phone application.

8 Future Extensions and Conclusion

Overall we are satisfied with the quality of our algorithm, as it was able to not only compete with the one Huawei had developed, but also impress their Machine Learning experts with the results it had for Chinese characters, an aspect that was not taken into account by their validation tool, as it was deemed too hard.

However, with our current level of understanding, we are aware that there are plenty of improvable aspects and quite a few things which we would now do differently.

8.1 Future Extensions

In this section we discuss potential future extensions we would have worked on if we had more time. Two of them are related to Moussaka's design and the third one with BCLPD.

8.1.1 GAN

First of all, having heard both the presentation of the winners of the competition and that of the specialists in Huawei's Machine Learning Centre [1] we have certain ideas on improving our algorithm. We would change our data generation technique to incorporate a generator network, in a similar fashion as the one described in section 4.4. We are quite convinced that combining this aspect of a GAN with our CNN would yield significantly better results for this specific problem. This method has indeed been attempted, however the convergence limitations described in section 4.4.3 have not been entirely mitigated. For example, we have used the technique from [15] where the generator generator network is updated twice for each discriminator network update to avoid the fast convergence of the discriminator. However, this still did not lead to significantly better solutions. Therefore, a direct following step would be to incorporate more effective methods of preventing either of the two networks converging too fast.

We were unaware of the effectiveness of tools to generate realistic license plates, and as we were unable to develop our own we used only the training data we were given. However, during our visit to the Huawei Machine Learning centre we were presented with a 'game'. The aim was for the player to find the real license plate in a set of potentially generated ones. This proved an impossible task, as the difference was impossible to see with the blind eye. Such a tool would have given us a significant advantage when training the network.

8.1.2 Yellow License Plates

Further, as mentioned in passing in section 5.1.2, we focused our attention on blue license plates with white letters and completely ignored the yellow license plates with black characters, as such license plates made up no more than 5% of the testing data and we

[1] We are not able to explicitly state the techniques Huawei uses, as when they were presented to us it was mentioned that they were confidential.

deemed that considering them as well would overcomplicate the learning process. This is what currently happens when we try to deblur a yellow license plate:

Figure 8.1: Performance of Moussaka on a yellow license plate.

Given more time and a more efficient way of generating data we would adapt our network to work with both types of license plates. A potential architecture we considered was having to separate networks, one for each colour and a discriminator network (or a simple classical image processing based approach) to distinguish between the different plates and dispatch them to their corresponding network.

8.1.3 Application Usability

Apart from extending our algorithm to achieve better results, we would also make the small prototype application described in section 7 more user friendly. In general, potential target audiences for this application exist, as for example it could be used by police officers, allowing them to identify license plates of cars with a standard smart phone.

The main improvement we considered was to add a system in the app which automatically detects the license plate, in real time, saving time from the user and making it easier to use in time critical situations. Further, as our app would then have a potential user group, we would be able to have it tested by the users and solve any interface or general issues, based on their responses.

8.2 Conclusions

In conclusion, this project involved a large amount of research, as we were asked to solve an open problem. Through this research, we learned about various techniques in Machine Learning and applied them in a Software Engineering context, to create a satisfactory product.

We faced several difficulties along the way, from the fact that all of us had very different timetables and deadlines and we had to coordinate everything very meticulously in order to be able to deliver what we promised in each checkpoint, to the fact that when the project started we were very unsure about how to incorporate the Software Engineering practices he had learned in the context of a heavily research based project. Further, having an internal Huawei competition deadline almost two weeks before the end of the project meant that most of the work had to be done early and not in conjunction with the checkpoint deadlines.

Overall, we feel that we managed to overcome the challenges we faced and we are very pleased with the result of the competition, as most of our competitors were specialists in the field of Machine Learning or Image Processing.

References

[1] Soumith Chintala Alec Radford, Luke Metz. Unsupervised representation learning with deep convolutional generative adversarial networks. [Online; accessed 01-January-2018].

[2] Brandon Amos. Image completion with deep learning in tensorflow. [Online; accessed 01-January-2018].

[3] Wikipedia contributors. Histogram equalization — wikipedia, the free encyclopedia, 2017. [Online; accessed 7-January-2018].

[4] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning.

[5] Erhan et al. Why does unsupervised pre-training help deep learning?

[6] Hinton et al. A fast learning algorithm for deep belief nets, 2006.

[7] Pech Pacheco et al. Diatom autofocusing in brightfield microscopy: a comparative study, 2010.

[8] Shan et al. High-quality motion deblurring from a single image.

[9] Szegedy et al. Going deeper with convolutions.

[10] Xu et al. Deep convolutional neural network for image deconvolution.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks.

[12] Ltd. Huawei Technologies Co. Safe City Solution. `http://e.huawei.com/uk/solutions/industries/public-safety/safe-city/safe-city`. [Online; accessed 02-January-2018].

[13] Prateek Joshi. Opencv with python by example, 2015.

[14] Shaoqing Ren Kaiming He, Xiangyu Zhang and Jian Sun. Deep residual learning for image recognition.

[15] Taehoon Kim. Dcgan in tensorflow. [Online; accessed 01-January-2018].

[16] James G. Nagy Per Christian Hansen and Dianne P. O'Leary. Deblurring images: Matrices, spectra, and filtering.

[17] Faizan Shaikh. Introduction to gradient descent algorithm (along with variants) in machine learning. [Online; accessed 03-January-2018].

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition.

[19] Todd Veldhuizen. The point-spread function (psf) model of blurring. [Online; accessed 04-January-2018].

[20] Francesco Visin Vincent Dumoulin. A guide to convolution arithmetic for deep learning. [Online; accessed 03-January-2018].

[21] Andrew Zisserman. B14 image analysis, lecture 3, 2014. [Online; accessed 07-January 2017].