DEEP LEARNING FRAMEWORK FOR CHARACTER RECOGNITION IN LOW QUALITY LICENSE PLATE IMAGES

by
Christina Dorothy Paolicelli

A thesis submitted to Johns Hopkins University in conformity with the requirements for the degree of Master of Science

Baltimore, Maryland
August, 2020

Abstract

Automatic License Plate Recognition (ALPR) systems commercially available are limited in their capability to provide character recognition on low-quality images [20]. The ability to perform character recognition on low-quality license plate images would allow for increased efficiency in determining license plate characters. This would be beneficial for tasks, such as digital forensics, that currently require human involvement in order to read low-quality license plate characters. In recent years, advances in Deep Learning have shown that for object detection tasks these networks can achieve as good as, or better than, human levels of performance [2,6]. The intent of this thesis is to introduce a foundational Deep Learning framework for analysis of character recognition performance on license plate images that have undergone various types and levels of image quality degradation.

A framework is defined in this thesis to provide end-to-end tooling for analyzing this problem space. The framework allows for the creation of synthetically generated datasets on which image degradation can be applied. Image augmentation techniques analyzed in this thesis include: JPEG Compression, motion blur, Gaussian noise, and affine transforms. TensorFlow was used as the Machine Learning framework for the development of Machine Learning networks. Network training was performed on the Maryland Advanced Computing Center's GPU based system. A pipeline for obtaining results and metrics on the trained networks was developed.

Network performance was evaluated for several Machine Learning architectures and models, with the Faster R-CNN ResNet 50 network having optimal performance for the specialized object detection task of license plate images. Network performance across image sizes was evaluated, and an image size of 32 x 16 pixels was determined to be the lower threshold for

adequate recognition. The network was also trained and tested on image datasets with data augmentation applied. The results show that when trained on augmented synthetic data the license plate character recognition system is robust to most image degradation techniques.

Primary Reader and Advisor: Nicholas Beser, Ph.D.

Secondary Reader: John Samsundar, Ph.D.

Secondary Reader: Cleon Davis, Ph.D.

Table of Contents

List of Tables

List of Figures

Chapter 1

1 Introduction

1.1 Problem Space

1.1.1 Forensics Problem

Automatic license plate recognition (ALPR) is a common computer vision task used across many domains including, but not limited to, automated tolling, access control, and law enforcement. Current commercial systems require controlled lighting conditions, specific angles of capture, an image of sufficient size, and specific camera settings to accurately read license plate characters [20]. These constraints limit the applications common commercial ALPR systems can be applied in. Potential applications of ALPR, such as digital forensics, have a need to identify license plates from noisy images obtained on non-dedicated camera equipment under poor lighting conditions with unknown camera specifications and locations. For example, consider an attempt to use an ALPR system to identify the license plate of a car at a crime scene with footage from a closed-circuit television (CCTV) camera. The inability of commercial ALPR systems to perform reliably on these low-quality images means that for the purposes of forensics, license plates must be identified manually. Identifying license plates manually is a labor-intensive process requiring specially trained individuals. Development of an ALPR system that is robust to conditions contributing to low-quality images, therefore, has the potential to provide benefits where the identification of low-quality plates is needed or where the identification is not currently possible but would improve situational information.

1.1.2 Implications of Machine Learning

Machine Learning is “programming computers to optimize a performance criterion using example data or past experience” [1]. This process has allowed computers to solve complex

problems, such as low-quality character recognition [26]. In recent years the capabilities of Machine Learning have been extended due to the implementation of Deep Learning. Deep Learning, “learn[s] feature levels of increasing abstraction with minimum human contribution.”... “in most applications, we do not know what structure there is in the input and any sort of dependencies that” ... “should be automatically discovered during training. It is this extraction of dependencies, or patterns, or regularities, that allows abstraction and learning general descriptions” [1]. Simply put, Machine Learning and Deep Learning allow for computers to solve problems of logical abstraction by training on large datasets.

With the advances of Deep Learning, Machine Learning has begun to show that it can achieve results that are as good as, or even better than, humans in terms of object detection and identification. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [2] is a benchmark challenge in object detection and classification that is run annually. The annual nature of the challenge allows for observation of the improvements in network performance over time, and, as the challenge has determined a threshold of human performance on the dataset, comparison to the human performance threshold. As shown in Figure 1, object classification networks began to exceed the human performance threshold on this dataset as early as 2015 with the Inception-v3 network [16].

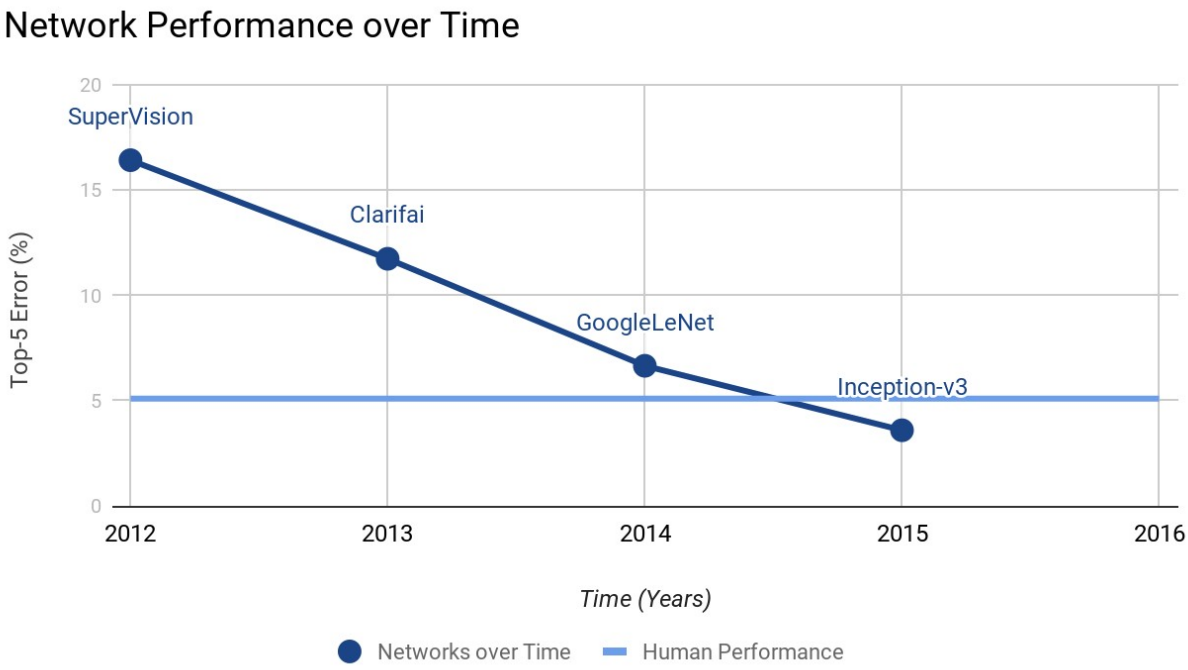

Figure 1: ILSVR Challange scores over time vs. human performance, data from [2, 6].

The intent of this project is not to reach human or better levels, or even to evaluate against humans or trained forensics analysts on the identification of License Plate characters, but to develop foundation work in License Plate recognition on low-quality images with the understanding that work in this field has the promise to be better than humans and in the long term reduce the workload of digital forensics analysts.

1.2 Past Work

1.2.1 Limitations without Machine Learning

There is a large body of work [3, 5, 7, 8, 21] that attempts to solve the automatic license plate recognition problem, and not all approaches implement Machine Learning. A common, non-Machine Learning-based ALPR solution, is to use optical character recognition (OCR). OCR is

the conversion of images of text to a string of characters such that a computer can understand it. This is frequently coupled with Machine Learning approaches, where Machine Learning is used to extract the license plate from a general scene image and segment the individual plate characters that are then fed to an OCR system for recognition. M. Sarfraz *et al.* [3] show the implementation of an OCR driven license plate recognition system that achieves 95% accuracy for Arabic characters from high-resolution digital images. OCR, however, faces challenges as picture quality degrades. When trying to classify a character, an OCR system tries to match the pixels of the image to known templates for each character. As an image degrades these pixel maps become less ideal, and OCR systems can struggle to match the appropriate template.

1.2.2 Multilayer Perceptron (MLP)

Since there is a desire to perform license plate character recognition on images with lower quality than is optimal for OCR performance, alternate methods such as Machine Learning have been explored. The simplest implementation of Machine Learning is the Multilayer Perceptron (MLP). A perceptron is a basic processing element that takes inputs, either from the external environment or from other perceptron's, associates each of these inputs with a weight, and then provides an output that is a function of the inputs and the weights. A perceptron can be used to distinguish between two classes by checking the sign of the output, where a negative answer indicates one class and a positive answer the other. A single perceptron with only a single layer of weights can only approximate linear functions of the input and therefore cannot solve nonlinear problems. The breakthrough of the multilayer perceptron, as shown in Figure 2, is that adding hidden layers between the input and the output allows the network to solve nonlinear problems. Nonlinear data is any dataset that when plotted (in any dimension space) cannot be separated into classes by a linear function [1].

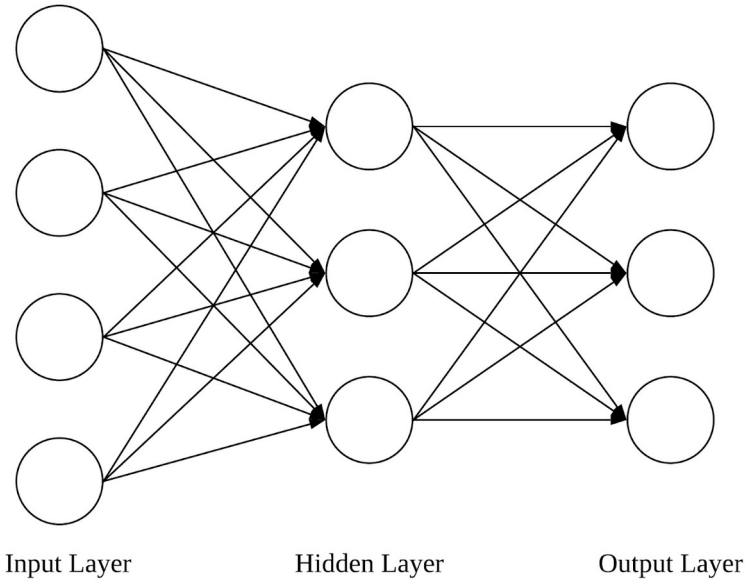

Figure 2: Basic MLP Architecture

Mello, *et al.* [5] uses two MLPs for character recognition in their ALPR system. In this approach, the recognition of letters and numbers is divided between the two different MLPs. This can be done because for most license plate sequences the order of letters and numbers is fixed, i.e. the first three characters are letters and the last four characters are numbers. In this approach, recognition rates of 85.94% and 95.40% were achieved for letters and numbers, respectively. The lower recognition rate for letters is explained by the low number of sample images, in some cases as low as two training samples for a given character.

1.2.3 Convolutional Neural Networks (CNNs)

In object detection problems, Convolution Neural Networks (CNNs) are one of the most commonly used types of Machine Learning, and the problem of ALPR is no exception. Basic neural nets like the Multilayer Perceptron's are intended for one-dimensional data and do not scale well into image problems. CNNs were designed to solve this. CNNs process data that

comes in the form of multi-dimensional arrays (like images) and can successfully capture the spatial and temporal dependencies found in images. The basic CNN architecture is shown below in Figure 3.

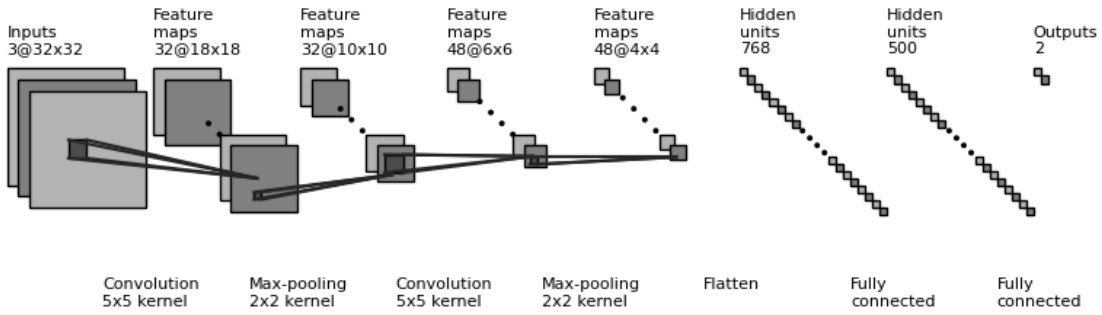

Figure 3: Basic CNN Architecture

Four key ideas drive the CNN structure: local connections, shared weights, pooling, and the use of a number of layers. In a CNN, local connections are employed by having each neuron (perceptron's) connected to only a subset of the input image. This is in contrast to MLP where the layers are fully connected. In images, local groups of values are often highly correlated with local patterns that are easily detected by using local connections. Furthermore, all neurons (perceptron's) in a particular feature map share weights. These shared weights along with local connections help reduce the number of parameters in the system which make the computations more efficient. Pooling allows the CNN to merge similar features and to progressively reduce the number of parameters – and therefore the amount of computation in the network. Pooling also has the added benefit of suppressing noise. In a CNN, the first few stages are convolutional and pooling layers. This takes advantage of the improvements in computational efficiency and noise suppression that convolutional and pooling layers provide. The result of these stages is then flattened into a column vector via a fully connected layer and fed into a

feed-forward network such as the MLP above to perform classification. CNNs were neglected in the Machine Learning field for many years, but the advent of more powerful computing resources and GPUs allowed CNNs to go deep, adding many layers which dramatically improved their capability for object detection and made CNNs the basis for competitive networks [6].

Laroca, *et al.* [7] uses three CNNs for ALPR. One CNN for license plate character segmentation, one for recognizing digits, and one for recognizing letters. For single-frame images, these networks have a recognition rate of 64.89%. This was further improved in this approach by the incorporation of temporal data from video capture.

1.2.4 Recurrent Neural Networks (RNNs)

The previously discussed neural networks are memory-less. The introduction of recurrance in Recurrent Neural Networks allows for the addition of short-term memory, which provides improved performance for sequential information. Simple RNNs are similar to Multilayer Perceptron's except that in addition to "feedforward connections, units have self-connections or connections to units in previous layers. This recurrence acts as short-term memory and lets the network remember what happened in the past" [1]. Figure 4 shows a simple RNN with the recurrence in red.

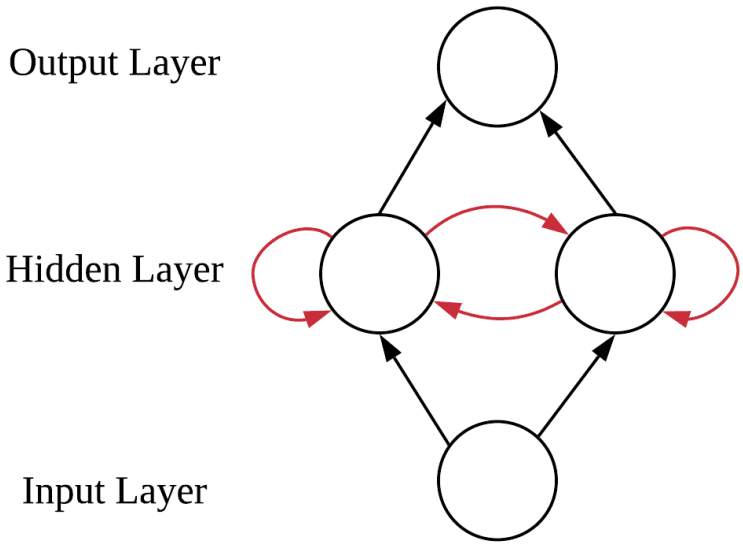

Figure 4: Simple RNN. Recurrence is shown in red

The capabilities of RNNs have been extended to allow for long time gaps in the sequence by the use of long short-term memory (LSTM) cells, shown in Figure 5. This memory is gated, meaning that the unit decides whether or not to store or delete the data. Deciding if the information is important happens through the assignment of weights [22, 23]. Weights are assigned through the network training process. This process uses the network to make guesses for training examples, computes the error between the guess and truth, and from this error adjusts the weights. Weights are adjusted through training until error, or loss, is below a desired threshold.

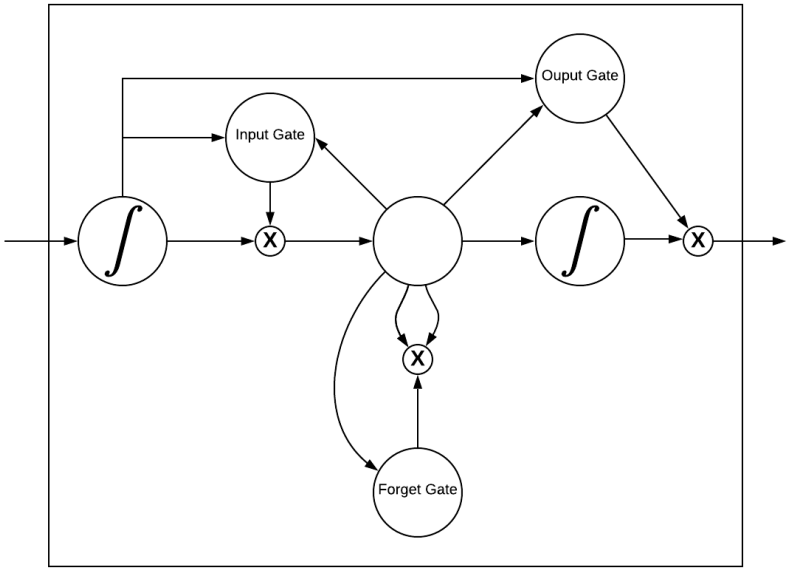

Figure 5: LSTM Cell

Li, *et al.* [8] uses RNNs with LSTMs to perform character recognition on license plates. The benefit of this method is that the system is able to process the whole string of license plate characters without the need to be segmented before feeding to the recognition system like the prior approaches that use CNNs or MLPs.

1.2.5 Poor Quality License Plates

These previous works have described systems trained on idealized datasets without deliberate applications of noise or other features that may impede performance. This thesis focuses on ALPR for degraded imagery. Previous work in identifying license plate characters under deliberate application of degradation is discussed in this section.

1.2.5.1 Motion Blur

Motion blur is an image degradation that is caused by the relative movement of objects in the field of view of the camera while the image is being captured. Motion blur can be reduced by shortening of exposure time which has the trade-off of higher noise, imposing higher

requirements on image sensor quality, or by providing stronger illumination. These methods of reducing motion blur are not always possible or cost-effective, and therefore, deblurring in post-processing is considered a viable alternative. Svoboda, *et al.* [9] develop a CNN to perform character recognition on license plates with motion blurring artifacts. The CNN was a 15 layer network CNN-L15 that consisted of only convolutional and ReLU layers. The work showed that Machine Learning had superior performance to OCR on motion-blurred images.

1.2.5.2 Severely Degraded Plates

Agarwal, *et al.* [18] performed work looking at extremely low-quality license plate images with resolution on the order of 20 pixels. They developed a CNN with eight convolutional layers and three fully connected layers followed by three separate softmax functions that provide a probability distribution of the 36 possible alphanumeric characters. The CNN was able to recognize characters in these degraded images that (untrained) humans were unable to do.

Chapter 2

2 Methodology

2.1 Tools and Infrastructure

2.1.1 Computational Resources

Machine Learning, and in particular Deep Learning, requires extensive computational resources in order to train the network on a dataset. These resources include memory, CPU/GPU access, and time. GPU technology, in particular, has been one of the pivotal advances that has allowed Machine Learning to solve problems with increasing complexity, especially with respect to the ability of networks to get “deeper”. This research, therefore, required a GPU based system capable of supporting multiple Machine Learning models. Johns Hopkins University is affiliated with the Maryland Advanced Research Computing Center (MARCC), a high-performance computing (HPC) facility. This research project was conducted using computational resources at MARCC, which supplied access to a collection of multi-core/GPU based Linux systems.

2.1.1.1 Simple Linux Universal Resource Manager (SLURM)

Resource allocation and scheduling are managed on MARCC using Simple Linux Resource Manager (SLURM) [24], a resource management system commonly used by HPC centers. SLURM uses partitions to allocate resources to jobs. The partition a job is requested on defines the maximum amount of resources that can be allocated. Resources that vary by partition include: the number of CPU cores, number of GPUs, and the maximum time a job can be run. Memory is tied in fixed amounts to a CPU core, and therefore, the amount of memory is based on the number CPU cores allocated. The different partitions available on MARCC, and their available resources are shown in Table 1.

Table 1: MARCC Partition Resources [24]

Partition	Default/Max Time (hours)	Default/Max Cores per Node	Default/Max Mem per Node	Serial / Parallel	Limits
shared	1/72	1/24	4.9 GB / 117 GB	Serial (multithreaded)	1 node per job
unlimited	1/unlimited	1/24	4.9 GB / 117 GB	Serial, Parallel	
parallel	1/72	24/24-28	4.9 GB / 117 GB	Parallel	
gpuk80	1/48	1/24	4.9 GB / 117 GB CPU 20 GB GPU	Serial, Parallel	
gpup100	1/12	1/24	4.9 GB / 117 GB 24 GB GPU	Serial	1 node (2 GPUs per user)
lrgmem	1/72	1/48	21 GB 1008 GB	Serial, Parallel	
scavenger	6	1/24-28	5 GB / 128 GB	Serial, Parallel	5 nodes per job
express	1/12	1/6	3.5 GB / 86 GB	Serial (multithreaded)	1 node per job
skylake	1/72	1/24	3.5 GB / 86 GB	Serial (multithreaded)	1 node per job

This project was developed with a single GPU and six CPU cores for training. As will be discussed later, this project was designed such that if a job was to terminate in the middle of training, the state of the trained model is saved and training can resume on the same model during a later job. This means that time limits, while a nuisance, were not a limiting factor.

2.1.2 Language & Library Selection

2.1.2.1 MATLAB Constraints on MARCC

While MARCC provides access to resources it is not without limitations, particularly as management of the system is not controlled by the project. At the beginning of this project's development (January 2020), the latest version of CUDA (GPU drivers) installed on the MARCC system was CUDA 9.0. The most recent version of MATLAB (at that time was 2019b) required CUDA 10.0. Using an older version of MATLAB significantly reduced the options available for Machine Learning development and not using a GPU would significantly increase the time it would take to train a Machine Learning network. This version issue (resolved March 2020, too late for this project) effectively precluded the usage of MATLAB as the Machine Learning toolset for the project.

2.1.2.2 TensorFlow

Given the barriers to using MATLAB, other Machine Learning frameworks were investigated and TensorFlow was ultimately selected. TensorFlow is a commonly used end-to-end open-source Machine Learning platform written in Python. TensorFlow provides a pre-built object detection API which "makes it easy to construct, train, and deploy object detection models" [10]. There are two major versions of Tensorfow, the newer version (anything > 2.0) has the same problem as MATLAB and requires a minimum of CUDA 10.0. However, the Object Detection API only supports the older version anyways which works with CUDA 9.0 thus resolving the MARCC CUDA support issue.

2.1.3 User Interface

MARCC has a strict requirement that job execution shall not be executed on system login but rather must be run by submitting a job request to one of the partitions. The simplest way a job

can be passed to a partition is by providing a bash script. In this project, a bash script could be defined which would run Python scripts to interact with the TensorFlow library. Running purely through bash scripts, however, does not allow for user interaction in real-time. MARCC also offers the option of, rather than passing an explicit bash script, requesting a job on a partition that launches Jupyter Notebook or Lab instance which in turn allows the user to run underlying scripts. For this project there was a strong desire to be able to interact with the training in real-time, therefore jobs were requested on MARCC to launch Jupyter Lab.

2.1.3.1 Jupyter Lab

Jupyter Lab is a visualization tool developed to support interactive scientific computing. Jupyter Lab allows the developer to see progress in real-time, allows different parts of the project framework to be run individually, and allows for the modification of the configuration on the fly, all while in the same SLURM job. A screenshot of the Jupyter Lab file for this project is shown in Figure 6.

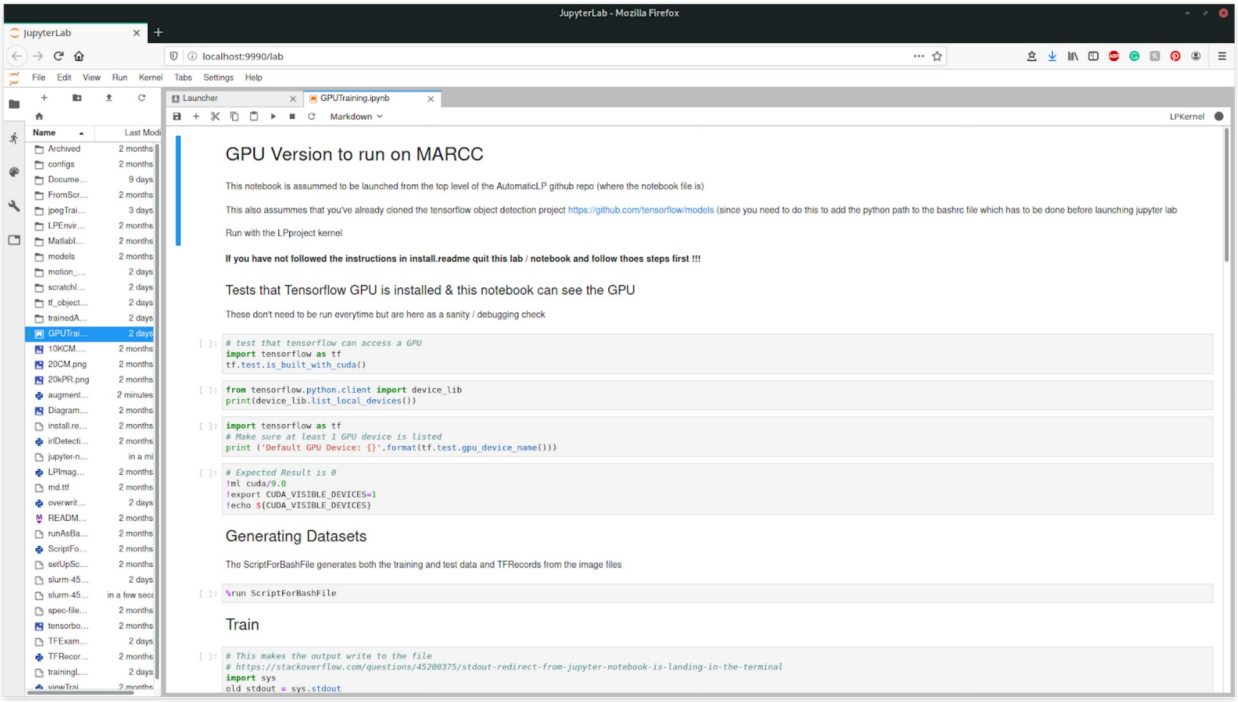

Figure 6: Screenshot of Jupyter Lab instance running on MARCC

2.2 Framework

With access to computing resources through MARCC, and a Machine Learning toolset decided upon, a framework to train and test neural networks for the ALPR problem was developed as shown in Figure 7. The framework has three major components: Data generation and network configuration (yellow), training (green), and evaluation (blue). These three components are covered in brief here and discussed in detail in the following sections.

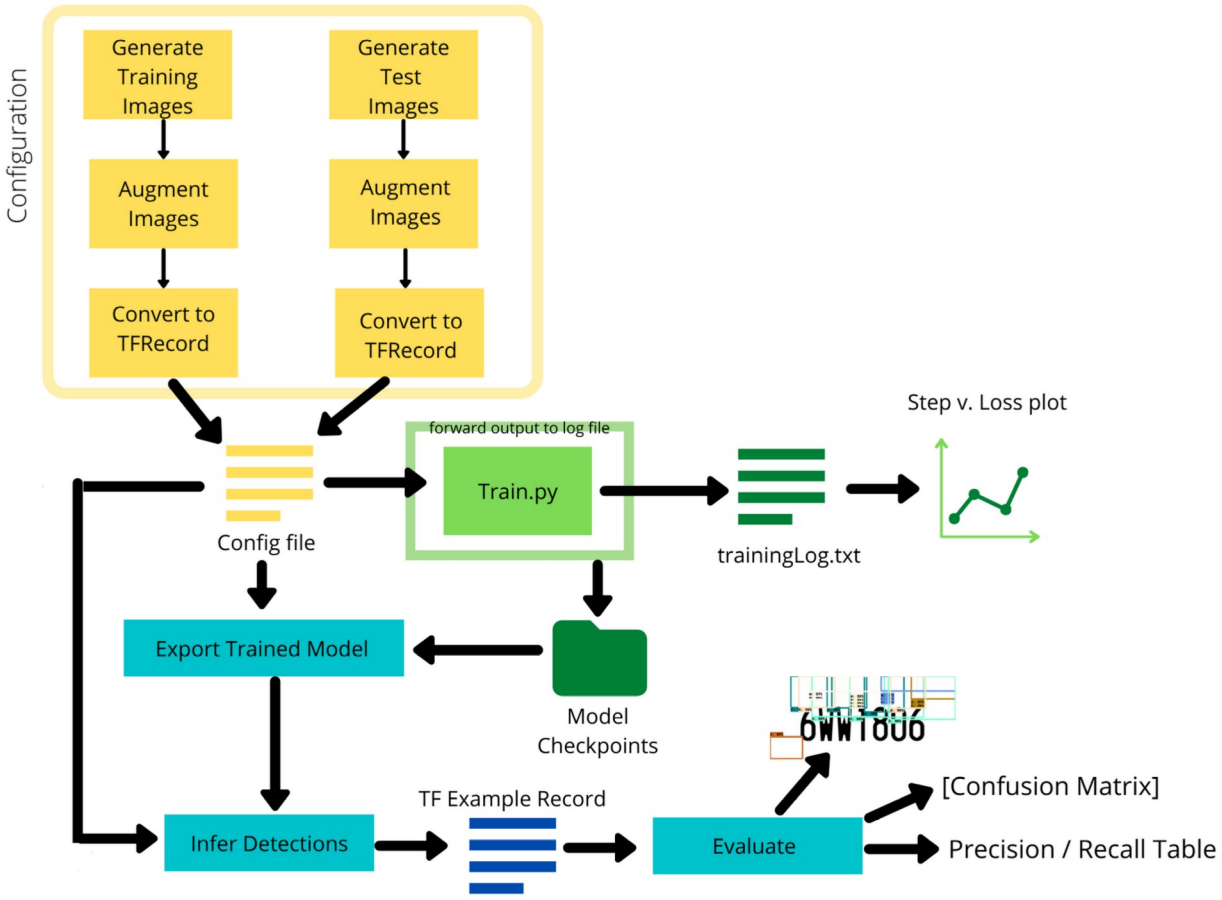

Figure 7: Highlevel Framework Flow

Machine Learning relies on training a network with large amounts of data. There are two options for obtaining this data, go out in the real world and collect it, or generate it synthetically. Since the goal of this project is to understand the impacts of system performance with controlled

image degradation, images were synthetically generated and broken into training data and test data. A dataset can be expanded past the initially generated images by using data augmentation. Data augmentation is a technique that increases the amount of available data by transforming the original dataset by applying various algorithms. Data augmentation also adds variation to an “ideal” data set making object recognition robust to variations. When applied to the test dataset augmentation allows for evaluation of performance under the applied augmentation. Images and ground truth are converted into a format called a TFRecord which can then provide the TensorFlow Object Detection API defined network with labeled data. Defining a network for the TensorFlow Object Detection API happens through a configuration file, which in addition to specifying the type of network architecture, controls training variables such as hyperparameters including batch size and learning rate. Batch size is the number of samples, in this case training images, to process before updating the model parameters. Learning rate controls the amount the network weights change with respect to error when updated.

Once data is generated and a network is defined the training data is fed into the TensorFlow Object Detection API executable. As shown in Figure 7 this produces two outputs (when run through the Jupyter Lab Configuration), a log file which is then converted into plots of training loss and “Model Checkpoints” which are essentially save files of training. These “Model Checkpoints” allow training to be restarted if processing is interrupted due to MARCC processing queue timeout.

In this framework, Evaluation is broken into three steps. First, the model checkpoints are converted into a “frozen graph” which exports the trained model to a format that can be used to

make detections on images. Second the test images that were generated are run against the trained model to infer detections, these inferred detections are saved in a TF Example Record. The last step is to convert the TF Example Record to various object detection and classification performance metrics.

2.2.1 Dataset Generation

The images generated are license plate strings that are compliant with the rules regarding the most recent Maryland passenger plates on a blank background to develop baseline performance. The most recent passenger vehicle Maryland plates follow the alpha-numeric sequence of 1AB2345 [11], all permutations of which give a sample size of over 10 million plates. Note that Maryland excludes easily confused letters such as “I” and “O”, meaning that there were 32 overall characters to evaluate [12]. Sample generated images are shown in Figure 8.

Figure 8: Sample Generated "Ideal" License Plate Images

2.2.1.1 Data Augmentation

Data augmentation is applied to generated training images in order to increase the training dataset size and train on non-idealized plates. Commonly used types of data augmentation

include, but are not limited to: JPEG compression, motion Blur, affine transforms, and Gaussian noise. Samples of some data augmentation techniques are shown in Figure 9.

Figure 9: Commonly used data augmentation techniques

A Python library called Imgaug was used to support the application of data augmentation to the generated training dataset. Augmentation techniques used on a dataset need to be used in the context of the problem being solved. Even if a data augmentation technique can be applied, the question of whether or not it provides a benefit to the problem must be addressed. Table 2 covers a selection of some of the data augmentation techniques available in the Imgaug library. The table indicates if the technique is applicable to the problem space, and if it was evaluated in this project. As with any project, this one is limited in scope and focused on a selection of data augmentation techniques.

Table 2: Overview of some data augmentation techniques

Available Data Augmentation Technique	**Applicable to ALPR (as configured here)**	**Rational**	**Used**
JPEG Compression	Yes	License Plate Images are stored and compressed	Yes
Motion Blur	Yes	License Plate Images are taken of moving objects	Yes
Affine Transforms	Yes	Images may be from a number of angles and sizes	Yes
Gaussian Noise	Yes	Noise added from the environment, camera, etc.	Yes*
Hue / Saturation	No	This project focuses on BE images	No
Flip (Horizontal & Vertical)	No	License Plate images are unlikely to experience this	No
Snow	Yes	License Plate image may be taken in many environmental conditions	No
Fog	Yes	License Plate image may be taken in many environmental conditions	No**
Brightness	Yes	Lighting may be unknown when image is taken	No**
Contrast	Yes	Lighting may be unknown when image is taken	No**

* Data Augmentation was implemented via config spec not Imgaug

** While these are applicable to the problem space they were not evaluated due to project scope restrictions

2.2.1.2 Implementation

The license plate generation script that can be run through this project's Jupyter lab interface produces two directories of "ideal" license plate images and a csv of ground truth data for each directory. One directory is comprised of 60% of the samples and is used for training and the other is comprised of the remaining 40% and is used for evaluation. In order to apply data augmentation transforms to the data, an augmenter was written with the Python Imgaug library. The augmenter provides the user augmentation customization capability by offering a command line interface through Jupyter lab to choose which data augmentation techniques to apply and the severity of the applied technique. The augmenter not only augments the images with the technique, but modifies the ground truth data if needed as well (image rotation, for example, requires ground truth modification but Gaussian noise does not). The overall implementation process for data generation is show in Figure 10.

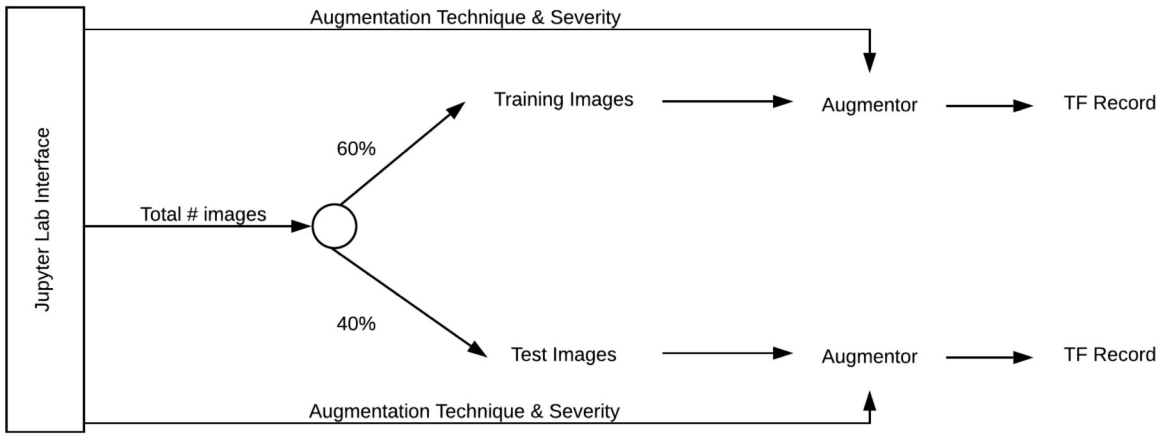

Figure 10: Data Generation process flow

2.2.1.2.1 TF Records

The TensorFlow Object detection API requires input data to be in a format called a TFRecord. The TFRecord is a format for storing a sequence of binary records based on protocol buffers which are a cross-platform, cross-language library that allows for efficient serialization of data.

The TFRecord converter script takes in generated images (training or testing) as well as the csv defining the ground truth and generates a TF Record. These records are used in training so that the network can compare its “guess” for classification to the correct class and ground truth. During training this allows the network to determine the error and adjust the model weights. During testing the knowledge of the ground truth allows the detection of the trained network to be compared against truth and performance metrics for the network to be determined.

2.2.2 Network Definition & Setup

Once a synthetic dataset is generated, a network needs to be developed in order to have something to train and test the data with. For the TensorFlow Object Detection API, networks are defined through a config spec. The config spec is what makes this API highly flexible. For example, the same training and testing framework can be easily modified to be run on different Machine Learning models and architectures by simply changing a section of the config spec. The config spec can also define variables associated with training like the hyperparameters. Driving network definition through this file, therefore, allowed for rapid testing.

2.2.2.1 TensorFlow Object Detection Model Zoo

The TensorFlow Object Detection API provides a collection of pre-trained models on common computer vision datasets and the configuration files associated with them in what is called the TensorFlow Object Detection Model Zoo. These configuration files and pre-trained models for object classification tasks provide a baseline for this project to perform transfer learning. That is, taking a model trained on one task and re-purposed to perform on a second task. In this case, the re-purposing is comprised of training the network against the dataset of license plate images for license plate character recognition. Since this is an object detection API all the networks included have shown high performance on object detection and classification tasks, however, every network has its trade-offs which may make it more or less suitable for a more specific task

such as the license plate recognition problem posed in this thesis. This Zoo allows for the rapid development of these different models for analysis of their capability for performance in this secondary problem space by using the baseline configuration files as a starting place.

2.2.2.2 Image Resizer

The configuration file can control how image sizing is treated on input to the network. This is important as in Machine Learning every pixel of an image is treated as a discreet piece of information. Therefore, if the number of pixels representing an image, or in this case a character, are decreased by reducing the size of the image the network has less information on which to make a decision. For this reason, size can be a limiting factor in the ability of a Machine Learning network to perform on a problem set. For network configuration, size is important as it can drive layer size and the number of layers used in order to make a localization and class determination on the input image. The size of later layers in a network is dependent on the size of the input layer. Therefore, all images must be the same size when passed into the network. When raw input images vary in size, like would be the case for a license plate image cropped out of a larger scene, the raw image must be adjusted to fit the expected size for input into the network.

The Object Detection API can resize an image in one of two ways when it's fed into the network; either as a "fixed shape resizer" or as a "keep aspect ratio resizer". The "fixed shape resizer" stretches the input image to the height and width that is specified in the config spec. The "keep aspect ratio resizer" resizes the image, keeping the aspect ratio to satisfy the minimum and maximum size constraints. If the "keep aspect ratio resizer" option is specified there is a sub-option of padding – setting "pad to maximum dimensions" to true pads the resized image is padded with zeros to the bottom and right. For this project, a "fixed shape resizer" was used.

The network was configured to expect images of a single size of which only images of that size were provided to the network. If the image was augmented with the Imgaug library in a way that resizes the image, such as the application of an affine transform, the augmented images are already padded to the default size before processing to the TFRecord.

2.2.2.3 Data Augmentation

The Imgaug library was used for most of the augmentation as it has the capability to provide basically any augmentation method that could be desired. Using an external library, however, does require additional piping. Therefore, although the options for data augmentation in the config spec are limited they are easy to apply with no extra effort. The implementation of Gaussian noise, for example, evaluated in this project was implemented through the config spec rather than an external augmenter.

2.2.2.4 Hyperparameters

The first hyperparameter controlled by the config spec is the batch size which is the number of samples evaluated between updates of network weights. Networks are trained as optimization problems where weights are updated based on error estimates. The more training examples (larger batch size) used in the estimate the more accurate the estimate will be; however, the batch size is normally kept quite small. Even though a larger batch size, in theory, has better results, a smaller batch size is generally better at generalization and can be used to combat overfitting. Additionally, a smaller batch size makes it easier to fit a batch worth of training data in memory, especially as datasets get larger and larger.

The other important hyperparameter that can be controlled via the config spec is the learning rate. This controls the amount the model changes with respect to the error when model weights are updated, that is it controls the speed at which the model learns. Machine Learning is an

optimization problem in which the computer is searching over a space for the optimal solution. In choosing a learning rate there is a trade-off between convergence and overshooting the solution. A learning rate that is too small can result in longer training times and get stuck in a local minimum. A learning rate that is too high, on the other hand, can jump over minima missing the optional solution. In order to achieve fast convergence, reduce oscillations, and avoid local minima while not missing the global minima the learning rate is often set to be adaptive. There are several types of learning rates that can be selected with the config spec: exponential decay learning rate, cosine decay learning rate, and manual step rate. The first two allow the learning rate to change over the course of training based on those mathematical patterns, these give the benefits of an adaptive learning rate. Manual step rate allows the learning rate to be explicitly specified based on what step of training is occurring, this allows custom scheduling of learning rates as the learning rate can be modified at as many training steps as desired.

2.2.2.5 Label Map

The TensorFlow Object Detection API requires a label map that maps each of the used labels to an id integer value. TensorFlow does not allow zero to be used as an id. Based on this 1-9 are mapped to their respective numerical values and used letters are mapped to 10-31 (remember MD does not use all letters in its plates). Zero is mapped to 32.

2.2.3 Training

In Machine Learning, training is the process of determining the values of weights and biases from the training dataset. This is a computationally intensive process that iterates over the training dataset, makes predictions, computes error, and updates the model parameters based on the error. The process of adjusting weights is repeated until the system loss is reduced to below the desired threshold. Once the loss is below the desired threshold the model can be “exported” with those weights and evaluated against test data. Because the evaluation process

is not iteratively updating weights it is significantly less computationally intensive. Training and test data are kept separate such that after training has occurred there is a second distinct dataset that can be evaluated against in order to understand performance on data the network did not see while training. An overview of the training process is shown in Figure 11 and described in detail in the following sections.

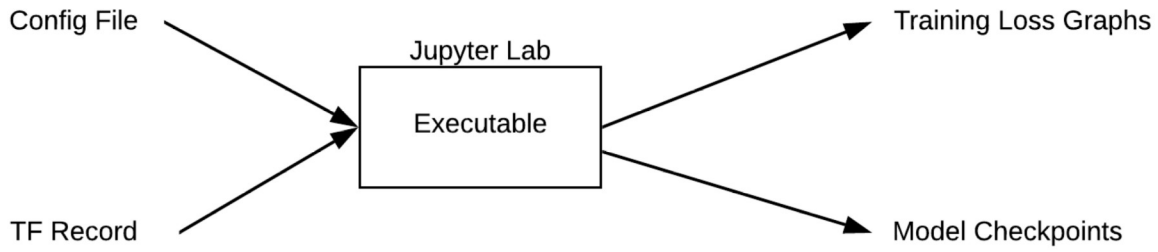

Figure 11: Flowchart of training process

2.2.3.1 Executable

To perform training the TensorFlow Object Detection API top-level training executable is run. It is recommended that the network is trained until the total loss reaches at least 2 (1%). Total loss is the sum of the localization and classification loss which are both percentages. The total loss is printed every global step, that is every batch that is being processed. The exact method of how loss is determined is dependent on the model chosen and is defined in the config spec. The executable saves what is called “model checkpoints” every 10 minutes, the most recent model checkpoint is what is used for evaluation or to resume training if, for example, a job is terminated.

2.2.4 Visualization

In the Jupyter lab instance, the output of the training executable script is logged to a text file, including the total loss numbers. Once training is complete a script is run that provides two graphs to show how training is going.

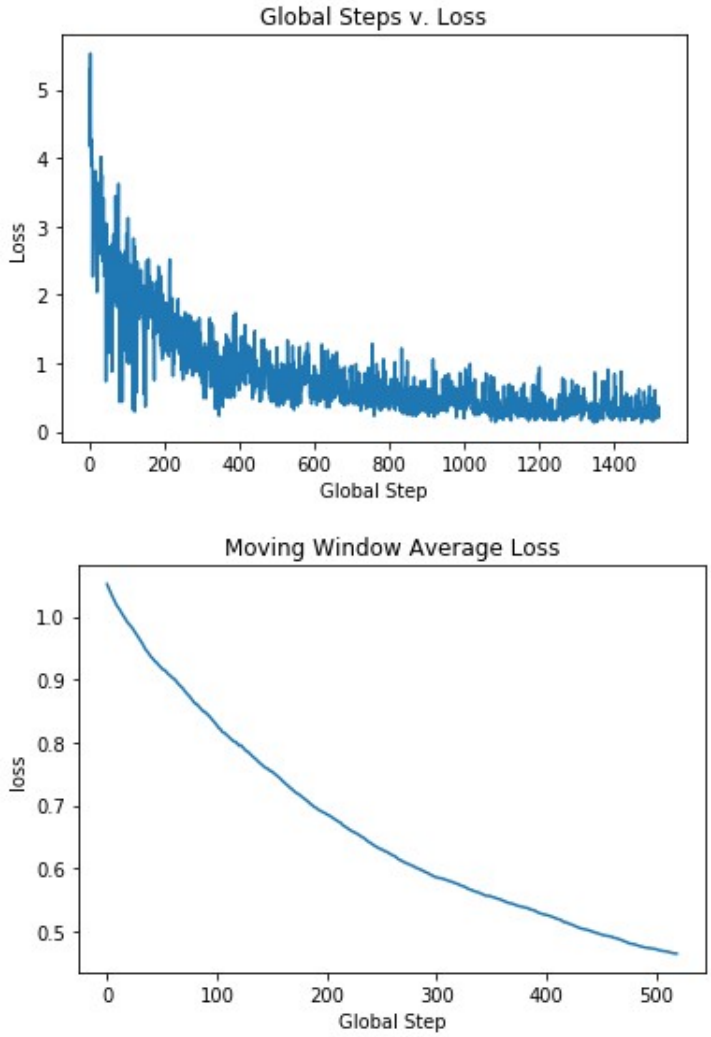

Figure 12: Sample plots of training loss. Top is per-step, bottom is moving average.

Sample plots of what is produced in the Jupyter Lab instance are shown in Figure 12. The top plot shows loss per training step. This plot is frequently very noisy and, therefore, difficult to read especially with longer training runs which increase the density of the data. For this reason, the bottom plot is also used which shows a moving average of the loss over the training steps. The moving average allows the user to clearly see the trends in the training loss. These graphs provide a visual representation of how training is going which can be easier for human comprehension than the logged text output.

2.2.5 Evaluation

2.2.5.1 Getting Results

In order to get results from a trained network to a metric that can be evaluated the trained network needs to be exported to what is called a “frozen graph”. The “frozen graph” allows for the images that are saved in the test dataset (via the testing TFRecord) to be evaluated against the trained model. The test images, their ground truth data, and the detections (both detected bounding boxes and class guesses) are then saved in a TF Example Record. A flow chart of the evaluation process is shown in Figure 13.

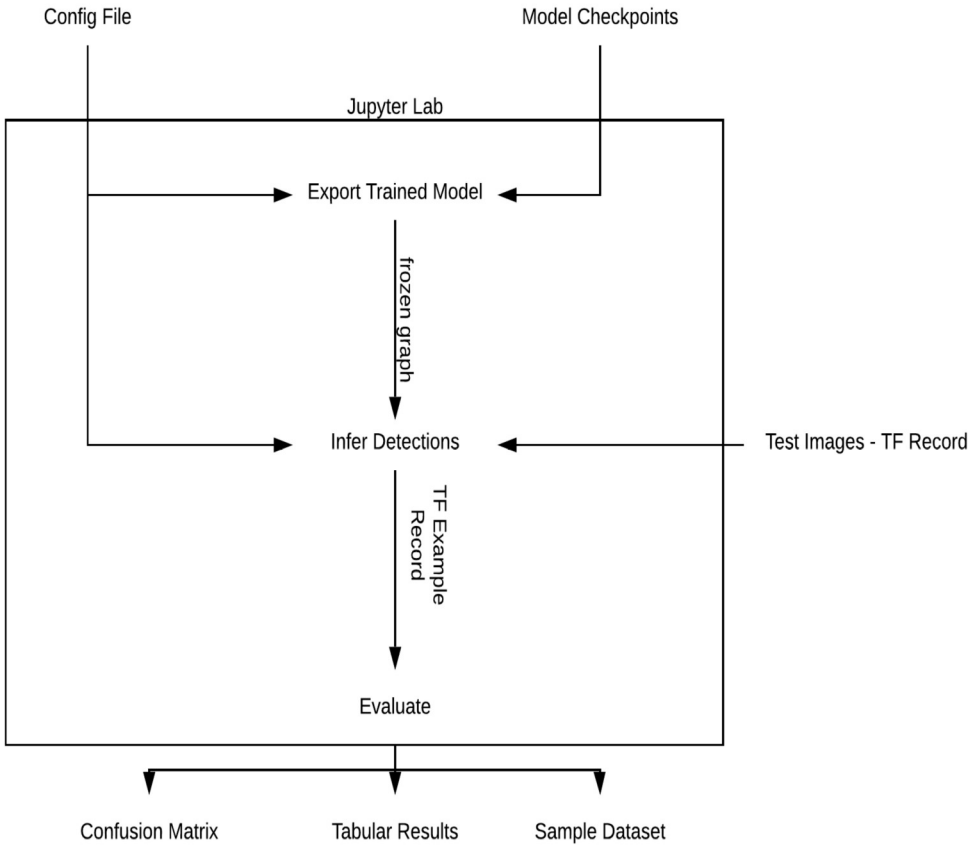

Figure 13: Flow of Evaluation

2.2.5.2 Object Detection Performance Tools & Metrics

In order to understand the results of the test images, a number of performance metrics need to be discussed.

2.2.5.2.1 Intersection Over Union (IOU)

Intersection Over Union (IOU) is a metric that quantifies the similarity between the ground truth bounding box and the predicted bounding box on a 0-1 scale with the closer the prediction is to the truth the higher the value. IOU is defined mathematically as, $IOU = \frac{truth \cap prediction}{truth \cup prediction}$.

2.2.5.2.2 Predictions

IOU provides a threshold for converting the scores to positive/negative classifications where IOUs above a threshold (generally 0.5) are positives and IOUs below the threshold are negatives. This allows for the determination of true positives, false positives, and false

negatives. A True Positive (TP) is an outcome when the model correctly predicts the correct class. A False Positive (FP) is an outcome where the model incorrectly predicts the correct class. A False Negative (FN) is when the model fails to predict a class when one exists.

2.2.5.2.3 Precision and Recall

The true positive, false positive, and false negative numbers can be combined into more generic metrics of precision and recall. Precision is the probability of the predicted bounding box to match the actual ground truth box and is defined as, $Precision = \frac{TP}{TP + FP}$. Recall, or the true positive rate, is the probability of ground truth objects being correctly identified and is defined as, $Recall = \frac{TP}{TP + FN}$. A result with high recall but low precision means that most objects are being detected, but most detections are incorrect. Conversely, a low recall rate with high precision means that objects that are detected are classified correctly, but most objects are not being detected. The ideal outcome is a high recall and high precision which indicates that the objects are detected and classified correctly.

2.2.5.2.4 Confusion Matrix

A confusion matrix is a graphical representation of performance that shows actual classification vs. predicted classifications. The y-axis shows the truth for each class and the x-axis shows the predictions for each class. This means that true predictions are on diagonal of the plot – the darker the cell is the more samples were predicted correctly. A sample confusion matrix is show in Figure 14, note that the last column shows false negatives and the bottom row shows false positives.

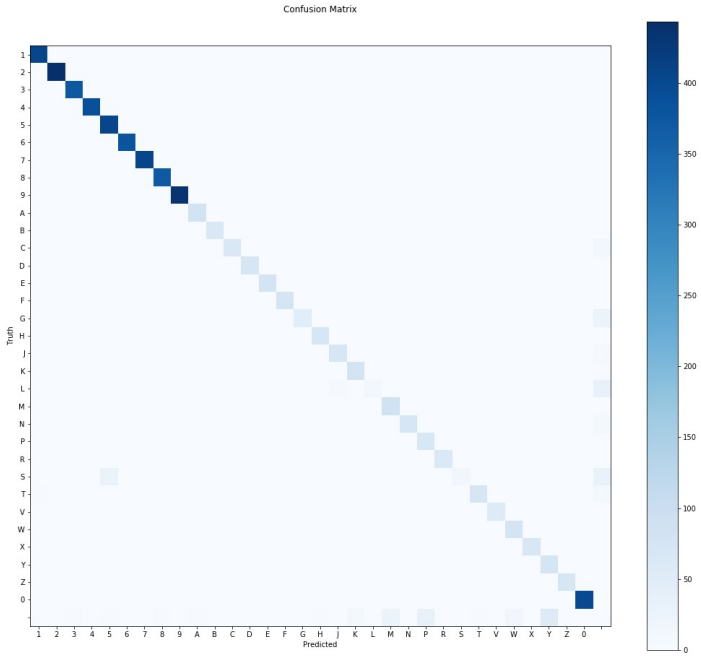

Figure 14: Sample Confusion Matrix. Note that for the license plate problem letters occur significantly less frequently than numbers which accounts for the relative shading.

2.2.5.3 Framework Outputs

When the evaluation is run through the Jupyter lab instance three outputs are generated. A directory is created with a 100-image sample subset of the larger test dataset with both the ground truth boxes and detected boxes overlaid – this allows the user to visually see what is happening and to uncover potential problems. Second, a Confusion matrix is generated. Lastly, a table of precision, recall, TP, FN, and FP for each character is printed.

2.3 Models & Architectures

A Machine Learning architecture is the structure of the neural network and defines the type and number of layers. This is in contrast to the model which is the actual neural network and defines the kernel size, activation function, and the number of nodes. An architecture can be combined with a number of different models. While all of these models and architectures have been used to solve object detection and classification problems they each were developed to address different problems and have different pros and cons. Therefore, it is useful to evaluate the performance of these architectures and models on the license plate problem posed in this thesis to understand the trade-offs choosing to use any one of these may have on the solution.

2.3.1 Architectures

2.3.1.1 Mobilenet

Mobilenet is a network that was designed to run on mobile devices, specifically for vision problems and for use with TensorFlow. It is able to maximize accuracy while dealing with restricted resources that come with embedded applications. The Mobilenet architecture is based on depthwise separable convolutions, shown in Figure 16, which are a form of factorized convolutions that factorize a standard convolution and a 1x1 convolution (also known as a pointwise convolution). This has the effect of reducing computation and model size. The architecture consists of normal convolutional layers, depthwise convolutional layers, and ends with an average pooling layer, fully connected layer, and a softmax layer. Each convolutional layer is followed by batch normalization and a ReLU nonlinearity [13]. The full Mobilenet architecture is shown in Figure 15.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 15: Full Architecture of Mobilenet [13]

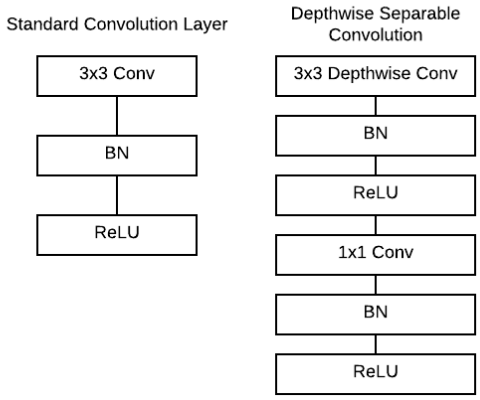

Figure 16: Comparison between standard convolution layer and depthwise separable convolution that is used in Mobilenet

Although not the point of this particular project, there is much ongoing work in the space of real-time license plate recognition on resource-restricted systems such as mobile phones or embedded platforms. Mobilenet, given its resource-limited conscious nature, may be a good choice for ALPR systems that wish to offer capabilities on these in the field platforms.

2.3.1.2 Inception v2

Inception v2 is an update of the original Inception architecture with incremental improvements in accuracy and reduced computational complexity. The Inception architecture was designed to address two problems: that choosing a kernel size can be difficult, and increasing depth of networks results in high computation cost and overfitting. The part of an image that is of interest (in this case the characters of a license plate) can be one of many sizes in various locations in an image meaning that choosing a kernel size in order to reliably locate the object of interest can be difficult. CNN based architectures prior to Inception (and Inception v2) stacked

convolution layers deeper and deeper to achieve performance, this has two downsides; first, it is computationally expensive, and second very deep networks are prone to overfitting.

The original Inception architecture addresses these problems by proposing filters with multiple sizes that operate on the same level. Essentially the network gets wider rather than deeper, reducing the computational complexity and resolving the overfitting issue. Inception v2 improves on this. It is noted that an issue with the original Inception architecture is that reducing the dimensionality too much can cause loss of information in what is known as a “representational bottleneck”. In order to work around this, the 5x5 convolutions of the Inception architecture are factorized into two 3x3 convolution operations in Inception v2. This improves computational speed, as a 5x5 convolution is 2.78 times more expensive than a 3x3 convolution. This idea of breaking down convolutions for speed gain is further extended in Inception v2 by recognizing that a convolution of size nxn can be factorized to a combination of 1xn and nx1 convolutions [14]. Figure 17 to Figure 20 show the full architecture of Inception v2.

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 18	35×35×288
5×Inception	As in figure 19	17×17×768
2×Inception	As in figure 20	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figure 17: Full Inception v2 architecture [14]. Referenced Figures follow.

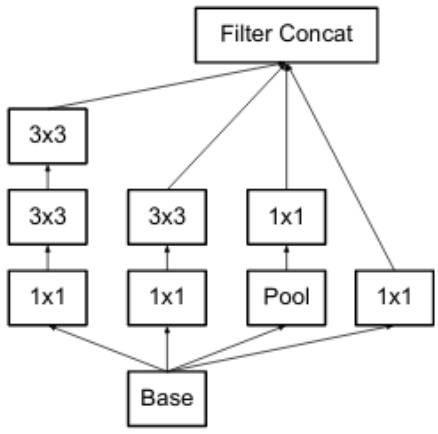

Figure 18: Inception module where 5x5 convolution is replaced by two 3x3 convolutions [14]

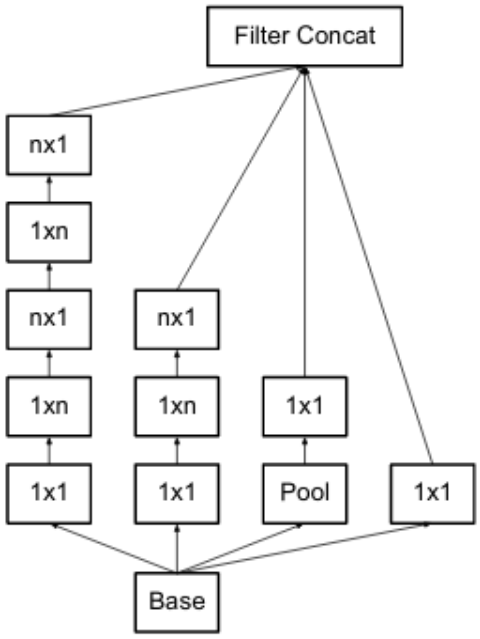

Figure 19: Inception modules after the factorization of the nxn convolutions [14]

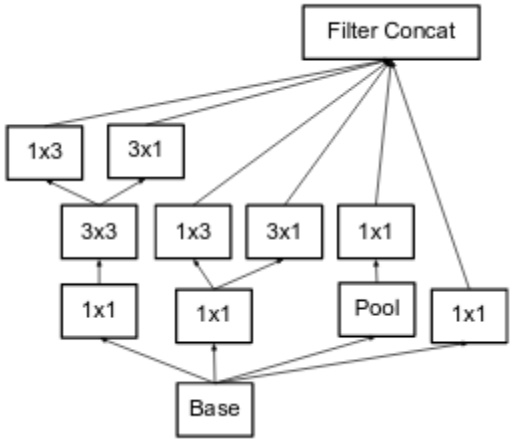

Figure 20: Inception module with an expanded filter bank [14]

Reduction of computational complexity and overfitting is beneficial for any neural network – Inception v2, however, may be a good choice for this problem as the multiple kernel sizes can help find license plate characters which may be rotated, on an angle or in a non-deterministic part of the image.

2.3.1.3 Residual Network (ResNet)

Residual networks, or ResNet, arose from the problem that the deeper a neural network is, the harder it is to train. To work around this rather than trying to learn an underlying mapping as traditional networks do, ResNet learns the differences between the input and output also known as the “residual”. ResNet architectures consist of building blocks that are composed of a few

stacked layers and a shortcut connection that allows for adding the input to the output and computing the residuals [15]. Sample ResNet architectures are shown in Figure 21.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 21: Sample ResNet Architecture for a variety of layers [15]

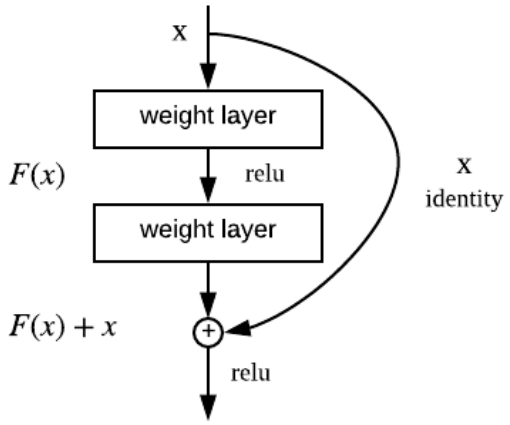

Figure 22: ResNet building block [15]

ResNet networks are classified by the number of layers that they are composed of. Two architectures analyzed in this project are ResNet 101 and ResNet 50. ResNet 101 is a 101-layer network that consists of 3-layer blocks, see Figure 22, and ResNet 50 is a 50-layer architecture.

ResNet has been shown to have good baseline performance for object detection tasks which makes it a good candidate for ALPR problems.

2.3.1.4 Inception-ResNet

Inception architectures tend to be very deep. Residual connections are argued to be instrumental in effectively training very deep networks. Given this, combining the Inception and ResNet architectures should lead to an architecture that can be deep but easier to train. In order to achieve the combination of the architectures, filter concatenations in the Inception architecture are replaced by residual connections which allows an Inception-style architecture to reap the benefits of the residual approach while retaining its benefits in computational efficiency [16]. The Inception-ResNet architecture is shown in Figure 23 to Figure 28.

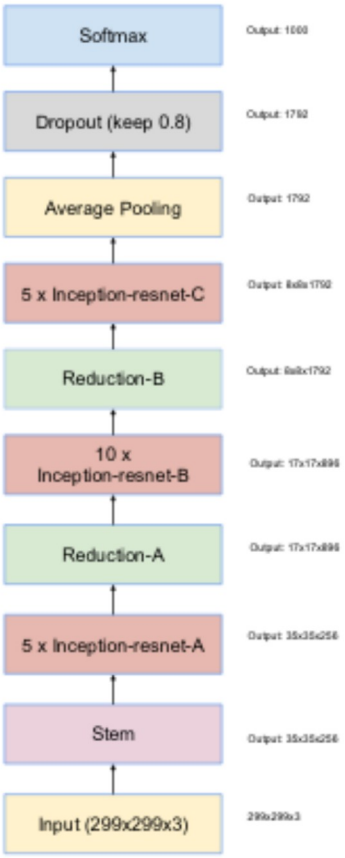

Figure 23: Schema for Inception-ResNet network [16]

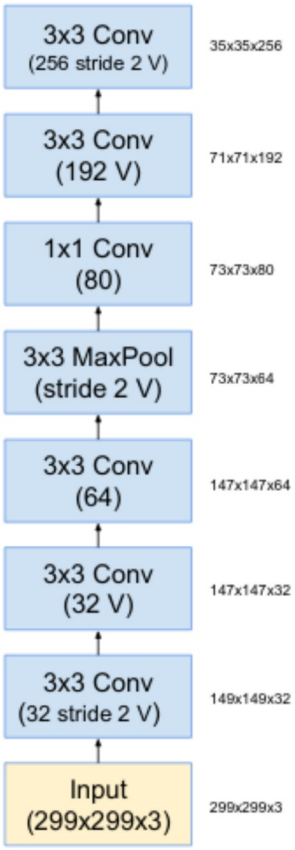

Figure 24: Stem for the Inception-ResNet-v1 network [16]

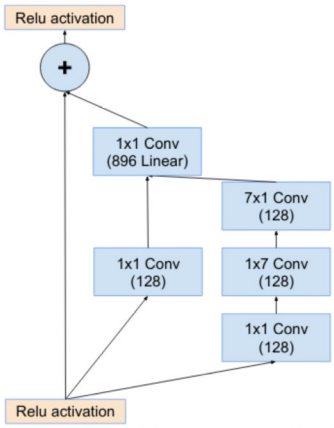

Figure 25: Schema for the 17 x 17 (Inception-ResNet-B) module of the Inception-ResNet-v1 network [16]

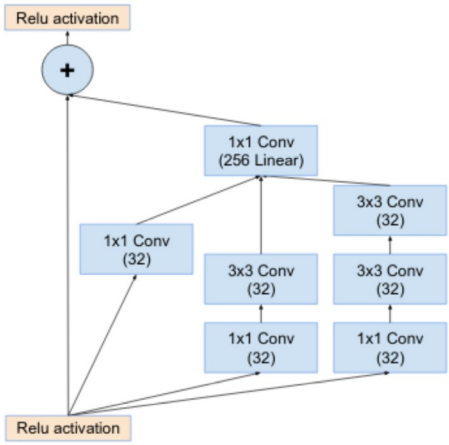

Figure 26: Schema for 35 x 35 grid (Inception-ResNet-A) module of the Inception-ResNet-v1 network

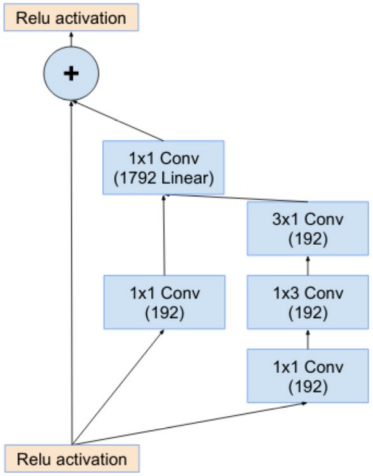

Figure 27: Schema for 8 x8 grid (Inception-ResNet-C) module of Inception-ResNet-v1 network [16]

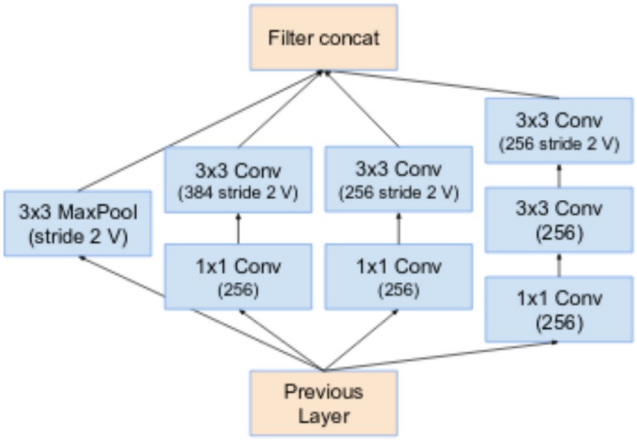

Figure 28: "Reduction-B" 17x17 to 8x8 grid-reduction module [16]

Like its components, Inception and ResNet, Inception-ResNet as an architecture has been shown to have good baseline performance on object detection tasks. Furthermore, it theoretically should have better performance than either Inception or ResNet architectures by themselves making it a promising architecture to try for ALPR tasks.

2.3.1.5 Neural Architecture Search (NAS)

Neural Architecture Search (NAS) isn't an architecture per-say but rather the process of automating architecture engineering. That is, it is a process that automatically finds the best architecture for the given problem. The process behind NAS finds the optimal architecture from all possible architectures by following a search strategy that will maximize performance. In the TensorFlow Object Detection API, NAS is implemented in the same way architecture is selected (through the config spec) and is included here for that reason. This option is appealing, if the optimal architecture for any given problem can always be found with this approach, in theory this should be the default option to solve any problem including that of the ALPR task.

2.3.2 Models

2.3.2.1 Single Shot Multibox Detector (SSD)

Single Shot Multibox Detector (SSD) is a model that reduces the computational intensity by eliminating bounding box proposals and subsequent pixel or feature resampling. Compared to previous models with similar goals, SSD achieves high accuracy using relatively low-resolution input. SSD takes all possible bounding boxes and breaks them up into a set of default boxes that span different aspect ratios for each feature map location. At prediction time, the network will generate scores for the presence of each object category in each default box and adjustments will be made to the boxes in order to better match object shape. Predictions from multiple feature maps that have different resolutions are combined in order to handle objects of different sizes in the images [17]. The full schema of the SSD model is shown in Figure 29.

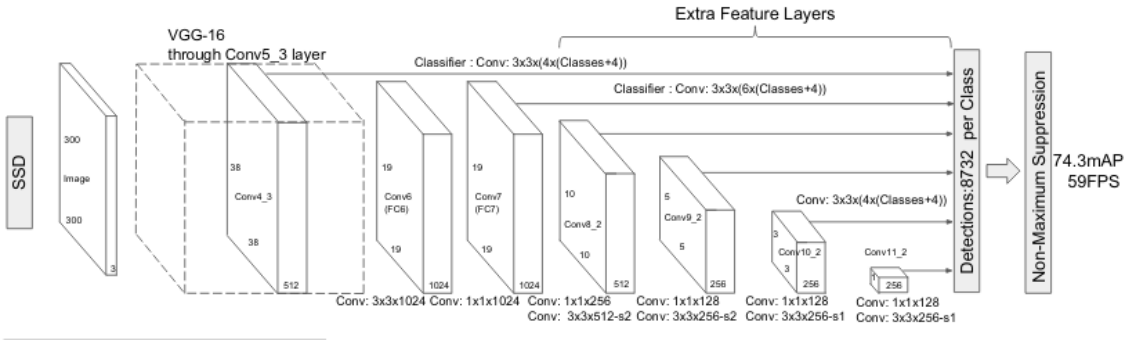

Figure 29: SSD schema[17]

2.3.2.2 Faster R-CNN

In order to discuss the Faster R-CNN an understanding of the networks it is based on, R-CNN and Fast R-CNN, must be developed. R-CNN was developed in order to take an image and both detect and localize the object within it. In order to minimize the number of potential regions to evaluate the object being in, a selective search is used to extract region proposals (or regions of interest ROIs) from the image. These generated region proposals are then fed through a trained CNN. A State Vector Machine (SVM) is then used in order to determine the presence of an object within the region.

R-CNN requires a forward pass of the CNN for every single region proposal of every image which is a time-intensive process. Furthermore, three separate networks must be trained: a CNN to generate features, a classifier, and a regression model to tighten the bounding boxes. In order to address these challenges, Fast R-CNN was introduced. Instead of feeding region proposals to the CNN, the input image is fed directly to the CNN to develop a feature map from which the region proposals can be determined. The feature map is passed to an ROI pooling layer that reshapes it to a fixed size such that it can be passed to a fully connected layer and then to a softmax layer to predict the class and bounding boxed of the proposed region. This is

a “Fast” R-CNN because the work can be done in a single pass of the image through the CNN rather than each region proposal needing to be passed through separately. Additionally, the CNN, classifier, and bounding box regressor can be trained jointly in a single model reducing training overhead.

Faster R-CNN fixes the remaining bottleneck of the process – determining the region proposals. For Fast R-CNN, region proposals are determined through a selective search, this process is slow and has been shown to be the limiting factor in network speed. In Faster R-CNN, similar to Fast R-CNN the image is provided to a CNN which in turn produces a convolutional feature map. Rather than apply the selective search to determine the region proposals, however, a separate network known as a region proposal network is used. The region proposal network functions by passing a sliding window over the CNN feature map and at each point outputting k potential bounding boxes and scores based on how good each of the bounding boxes is expected to be. Finally, ROI pooling is done to format the data in order to pass to a classifier [19] [27]. The overall network process for Faster R-CNN is shown in Figure 30.

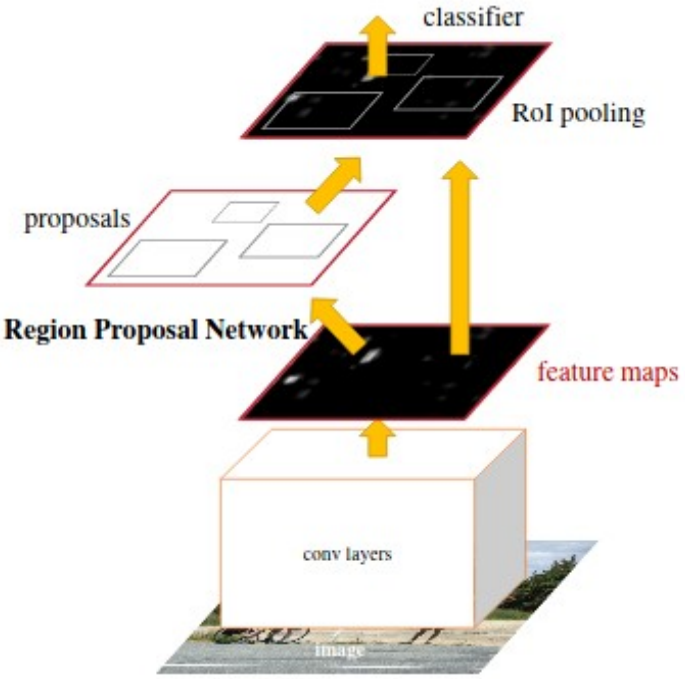

Figure 30: Faster R-CNN network for object detection [19]

2.3.2.3 Region-based Fully Convolutional Network (R-FCN)

Region-based Fully Convolutional Network (R-FCN) begins with the fact that prior models are region-based detectors that apply, per-region, a subnetwork hundreds of times. This reapplication of the network on each region is a computationally costly endeavor. R-FCN is a shared, fully convolutional architecture that uses score maps and banks of specialized convolutional layers as the fully convolutional output followed by a pooling layer that collects the information [25]. The schema for the R-FCN model is shown in Figure 31.

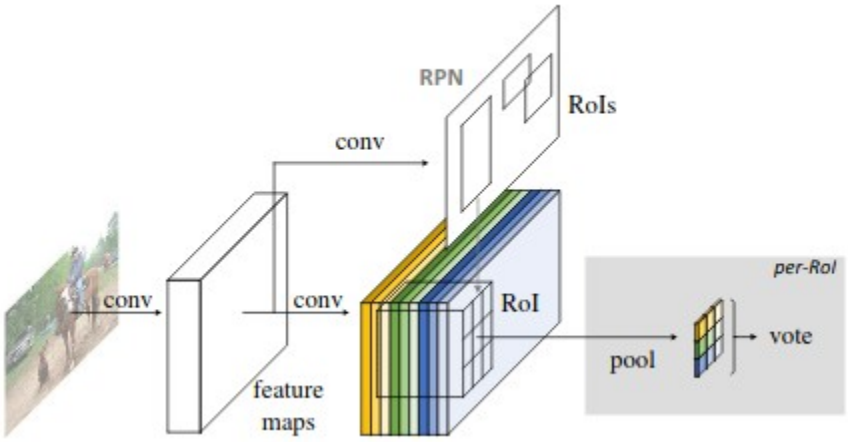

Figure 31: Overall schema of R-FCN [25]

Chapter 3

3 Results

3.1 Model & Architecture Results

In order to determine the optimal network for identifying license plate characters in the idealized dataset several architecture / model combinations were trained against the baseline dataset and evaluated. A brief summary of the tested models and architectures is shown in Table 3 and the performance of each of these is covered in depth in the following sub-sections.

Table 3: Summary of tested models and architectures

Model	Architecture	Speed	Precision	Recall
SSD	Inception v2	Fast	High	Low
SSD	Mobilenet	Fast	Medium	Low
Faster R-CNN	Inception v2	Fast	High	High
Faster R-CNN	ResNet 50	Fast	High	High
Faster R-CNN	0	Fast	High	High
Faster R-CNN	Inception-ResNet	Slow	-	-
Faster R-CNN	NAS	Slow	-	-
R-FCN	0	Fast	High	High

3.1.1 SSD Inception v2

SSD was tested with the Inception v2 architecture. For the characters that were detected precision was quite high, but recall was very poor and for a number of characters not a single instance was detected. This poor performance is shown in the confusion matrix, Figure 32, and Table 4. Note that the last column of the confusion matrix indicates false negatives which were very high for this model, ideally all detections would be on the diagonal, indicating that the prediction matches truth.

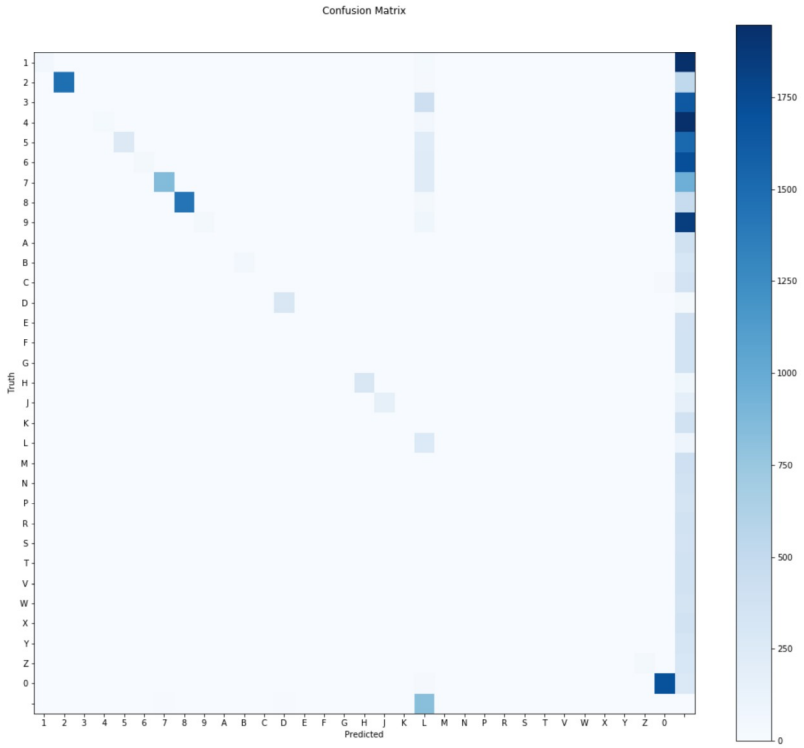

Figure 32: Confusion Matrix for SSD Inception v2

Table 4: SSD Inception v2 Tabular results

Class	Precision (%) @ 0.5 IOU	Recall (%) @ 0.5 IOU	TP	FP	FN
0	98.7	85.2	1694	23	295
1	100	2.4	48	0	1955
2	100	73.4	1472	0	533
3	-	0	0	0	2063
4	100	1.2	25	0	1999
5	100	12.3	245	0	1746
6	100	2	39	0	1949
7	95.8	41.8	856	37	1192
8	100	74.3	1436	0	497
9	100	1.8	36	0	1920
A	-	0	0	0	386
B	94.3	13.3	50	0	325
C	100	0.8	3	0	371
D	89.9	88	295	33	40
E	100	0.8	3	0	356
F	100	1	4	0	365
G	-	0	0	0	367
H	99	79.3	292	0	76
J	95.4	47.3	167	0	186
K	-	0	0	0	367
L	11	72.4	265	2165	76
M	-	0	0	0	186
N	-	0	0	0	376
P	-	0	0	0	341
R	-	0	0	0	367
S	-	0	0	0	357
T	100	0.8	3	0	371
V	-	0	0	0	375
W	-	0	0	0	349
X	-	0	0	0	367
Y	-	0	0	0	329
Z	100	12	41	0	300

3.1.2 SSD Mobilenet

SSD was tested with the Mobilenet architecture. Precision was mostly in the 80 and 90 percent range, which while good, is far from perfect. Recall was even worse than SSD Inception v2. The results are shown in the confusion matrix, Figure 33, and in more detail in Table 5.

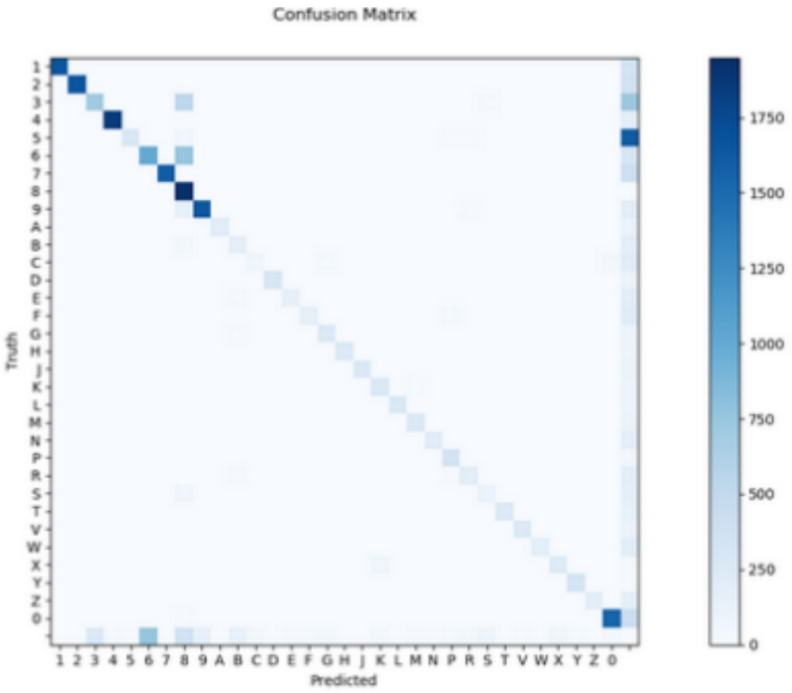

Figure 33: Confusion Matrix for SSD Mobilenet

Table 5: SSD Mobilenet Tabular results

Class	Precision (%) @ 0.5 IOU	Recall (%) @ 0.5 IOU	TP	FP	FN
0	95.8	78.4	1552	68	427
1	99.7	83.3	1659	5	332
2	100	82.4	1663	0	354
3	73.6	36	714	256	1269
4	98	91.2	1861	37	179
5	95.6	14.2	282	13	1698
6	56.6	48.4	1000	766	1066
7	99.6	79.8	1601	6	406
8	49.8	100	1949	1965	0
9	91.7	83	1651	150	337
A	97.1	61.8	201	6	124
B	49.4	44.9	171	175	210
C	66.7	21	75	37	282
D	98.3	75.6	295	5	95
E	90.5	43.1	153	16	202
F	87	40.9	161	24	233
G	80	71.5	261	65	104
H	97	75.7	256	8	82
J	98.9	71.5	271	3	108
K	72.6	71.8	273	103	107
L	100	75.1	272	0	90
M	86	71.6	252	41	100
N	90.6	53.2	202	21	178
P	81.4	91	342	78	34
R	74.4	47.4	186	64	206
S	51.1	33.9	120	115	234
T	94.8	63.2	237	13	138
V	95.4	72.2	249	12	96
W	98.8	47.6	171	2	188
X	78.6	72.3	243	66	93
Y	93.2	97	329	24	10
Z	98	53.8	197	4	169

3.1.3 Faster R-CNN Inception v2

Faster R-CNN was tested with the Inception v2 architecture. On the surface the network had both high precision and high recall, however, there were significant false positives and false negatives. The worst performing characters were “L” with 10 false positives and “N” with four false positives and 75 false negatives as shown in the confusion matrix in Figure 34 and the tabular results in Table 6.

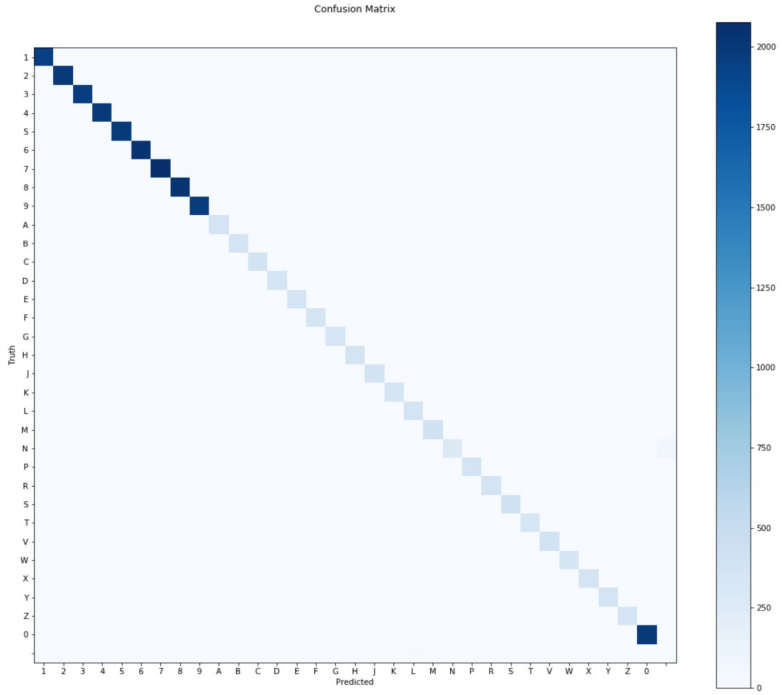

Figure 34: Confusion Matrix for Faster R-CNN Inception v2

Table 6: Faster R-CNN Inception v2 tabular results

Class	Precision (%) @ 0.5 IOU	Recall (%) @ 0.5 IOU	TP	FP	FN
0	100	100	1998	0	0
1	100	100	1945	0	0
2	100	100	1999	0	0
3	100	100	1960	0	0
4	100	100	1993	0	0
5	100	100	1983	0	0
6	100	100	2044	0	0
7	100	100	2077	0	0
8	100	100	2029	0	0
9	100	100	1972	0	0
A	100	100	369	0	0
B	100	100	368	0	0
C	100	100	361	0	0
D	100	100	356	0	0
E	100	100	350	0	0
F	100	100	356	0	0
G	100	100	326	0	0
H	100	100	367	0	0
J	98.6	99.7	369	5	1
K	100	100	349	0	0
L	97.4	100	368	10	0
M	98.2	100	392	7	0
N	98.6	78.5	274	4	75
P	100	100	368	0	0
R	100	100	392	0	0
S	100	100	339	0	0
T	100	100	384	0	0
V	100	100	356	0	0
W	99.7	99.7	372	1	1
X	100	100	372	0	0
Y	100	100	365	0	0

3.1.4 Faster R-CNN ResNet 101

Faster R-CNN was tested with the ResNet 101 architecture. The network had both high precision and high recall, but did have false positives for “1”, “P”, “T”, and “Y” as well as a false negative for “Y” as shown in the confusion matrix, Figure 35, and more clearly in the tabular results in Table 7.

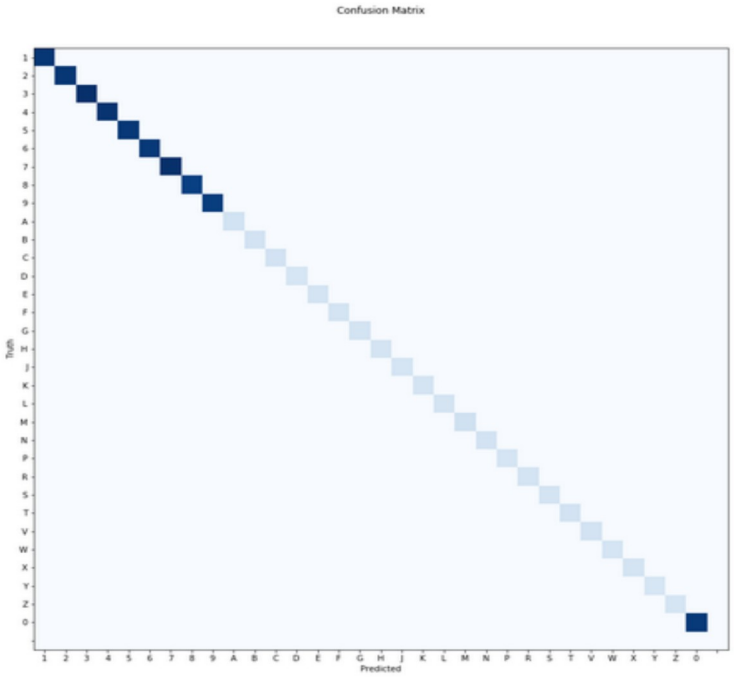

Figure 35: Confusion Matrix for Faster R-CNN ResNet 101

Table 7: Faster R-CNN ResNet 101 tabular results

Class	Precision (%) @ 0.5 IOU	Recall (%) @ 0.5 IOU	TP	FP	FN
0	100	100	1989	0	0
1	99.9	100	2003	1	0
2	100	100	2005	0	0
3	100	100	2063	0	0
4	100	100	2024	0	0
5	100	100	1991	0	0
6	100	100	1988	0	0
7	100	100	2048	0	0
8	100	100	1933	0	0
9	100	100	1956	0	0
A	100	100	386	0	0
B	100	100	375	0	0
C	100	100	374	0	0
D	100	100	335	0	0
E	100	100	359	0	0
F	100	100	369	0	0
G	100	100	367	0	0
H	100	100	368	0	0
J	100	100	353	0	0
K	100	100	367	0	0
L	100	100	366	0	0
M	100	100	405	0	0
N	100	100	376	0	0
P	99.7	100	341	1	0
R	100	100	367	0	0
S	100	100	357	0	0
T	99.7	99.7	373	1	1
V	100	100	375	0	0
W	100	100	349	0	0
X	100	100	367	0	0
Y	99.6	100	329	1	0
Z	100	100	341	0	0

3.1.5 Faster R-CNN ResNet 50

Faster R-CNN was tested with the ResNet 50 architecture. The network had both high precision and high recall with only a single false positive on “L” as shown in the confusion matrix, Figure 36, and tabular results in Table 8, making this network have a slightly better performance than Faster R-CNN ResNet 101.

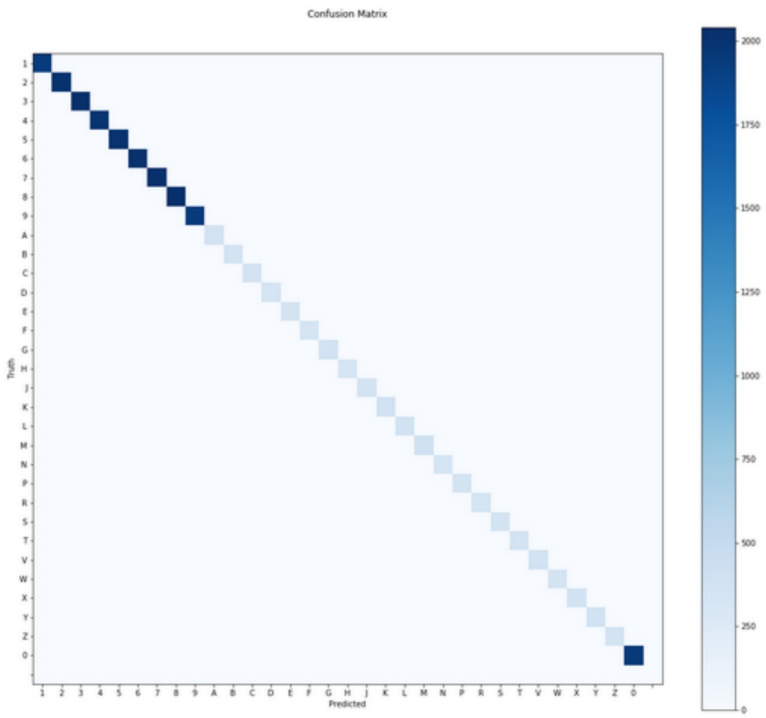

Figure 36: Confusion Matrix for Faster R-CNN ResNet 50

Table 8: Faster R-CNN ResNet 50 tabular results

Class	Precision (%) @ 0.5 IOU	Recall (%) @ 0.5 IOU	TP	FP	FN
0	100	100	1963	0	0
1	100	100	1940	0	0
2	100	100	2013	0	0
3	100	100	2040	0	0
4	100	100	2006	0	0
5	100	100	2012	0	0
6	100	100	2011	0	0
7	100	100	2030	0	0
8	100	100	2035	0	0
9	100	100	1950	0	0
A	100	100	369	0	0
B	100	100	366	0	0
C	100	100	368	0	0
D	100	100	342	0	0
E	100	100	351	0	0
F	100	100	340	0	0
G	100	100	373	0	0
H	100	100	349	0	0
J	100	100	362	0	0
K	100	100	385	0	0
L	99.7	100	382	1	0
M	100	100	403	0	0
N	100	100	348	0	0
P	100	100	359	0	0
R	100	100	345	0	0
S	100	100	353	0	0
T	100	100	366	0	0
V	100	100	373	0	0
W	100	100	352	0	0
X	100	100	374	0	0
Y	100	100	379	0	0
Z	100	100	361	0	0

3.1.6 Faster R-CNN Inception-ResNet

Faster R-CNN was trained with the Inception-ResNet architecture. Compared to training Faster R-CNN with ResNet 101, ResNet 50, and Inception v2 training with Inception-ResNet was extremely slow, see Figure 37. Not only was training slow but evaluating test images was as well. Given the poor speed response of the network and the desire for automatic license plate recognition to be run in real-time (not as an element of this project but in the problem space more broadly), this network was not further considered as a candidate for this project.

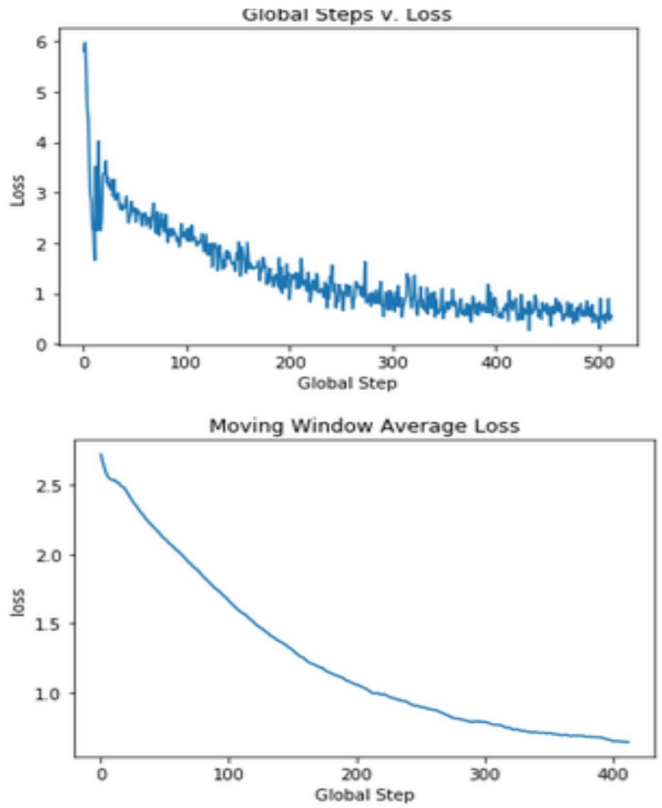

Figure 37: Training loss for Faster R-CNN Inception-ResNet

3.1.7 Faster R-CNN NAS

Faster R-CNN was trained with Network Architecture Search. Compared to training Faster R-CNN with ResNet 101, ResNet 50, Inception v2, and even Inception-ResNet training with Faster R-CNN NAS was extremely slow, see Figure 38. Not only was training slow but evaluating test images was as well. Given that Faster R-CNN Inception-ResNet was discounted due to its poor speed response and Faster R-CNN NAS is even worse this network was not included as a contender for further network comparison.

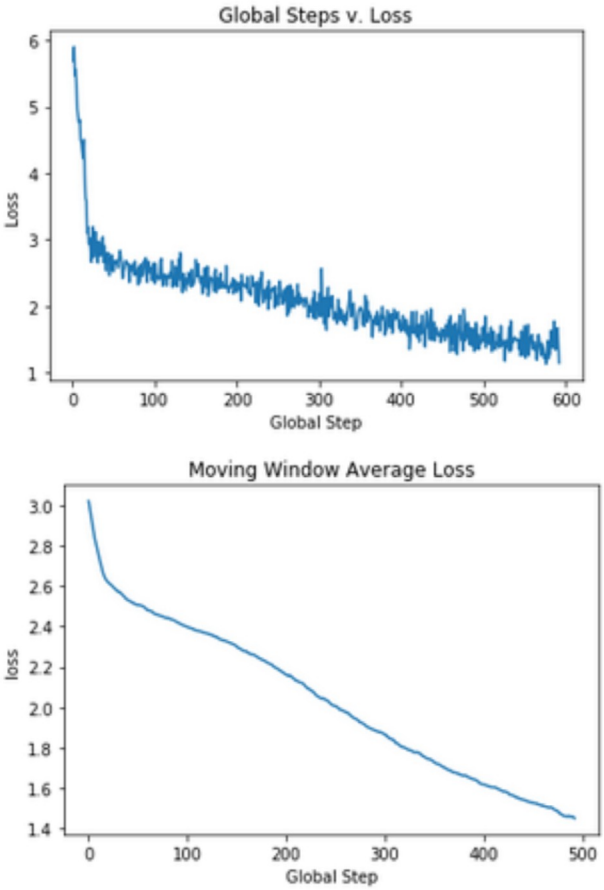

Figure 38: Training loss for Faster R-CNN NAS

3.1.8 R-FCN ResNet 101

R-FCN was tested with the ResNet 101 architecture. The network had high recall and precision comparable to the performance of ResNet 101 with the Faster R-CNN model. Note that as shown in the confusion matrix, Figure 39, and more clearly in the tabular results in Table 9 the detections were not perfect and there were false positives for “8”, “9”, and “V”.

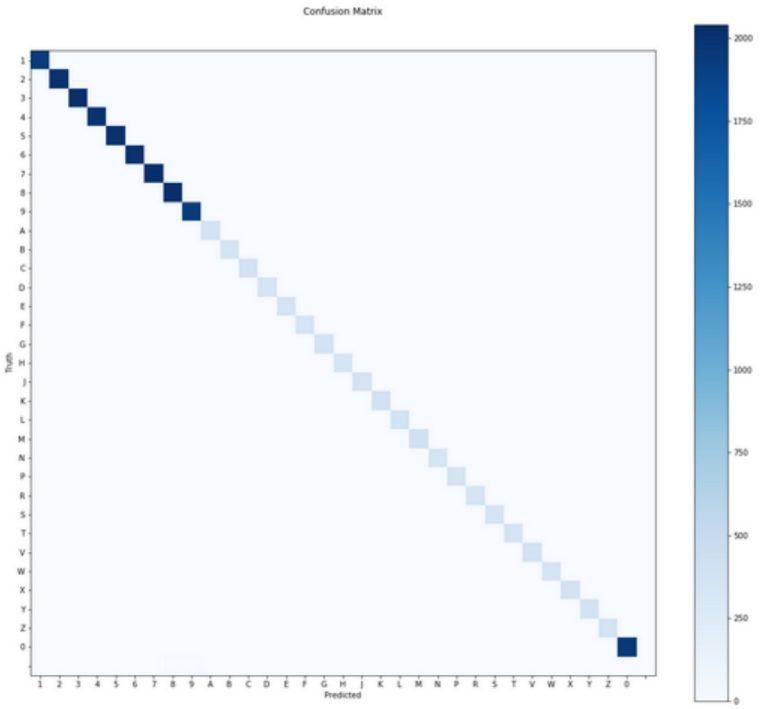

Figure 39: Confusion Matrix for R-FCN ResNet 101

Table 9: R-FCN ResNet 101 tabular results

Class	Precision (%) @ 0.5 IOU	Recall (%) @ 0.5 IOU	TP	FP	FN
0	100	100	1963	0	0
1	99.9	100	1940	1	0
2	100	100	2013	0	0
3	99.9	100	2040	1	0
4	100	100	2006	0	0
5	100	100	2012	0	0
6	100	100	2011	0	0
7	100	100	2030	0	0
8	99.6	100	2035	9	0
9	99.5	100	1950	9	0
A	100	100	369	0	0
B	100	100	366	0	0
C	100	100	368	0	0
D	100	100	342	0	0
E	100	100	351	0	0
F	100	100	340	0	0
G	100	100	373	0	0
H	100	100	349	0	0
J	100	100	362	0	0
K	100	100	385	0	0
L	100	100	382	0	0
M	100	100	403	0	0
N	100	100	348	0	0
P	100	100	359	0	0
R	100	100	345	0	0
S	100	100	353	0	0
T	100	100	366	0	0
V	99.7	100	373	1	0
W	100	100	352	0	0
X	100	100	374	0	0
Y	100	100	379	0	0
Z	100	100	361	0	0

3.1.9 Comparison

As discussed with each individual model several architecture/model combinations were trained against the baseline license plate dataset and resulting precision and recall were evaluated, averaged across all possible license plate characters for Maryland plates. The results of this comparison are shown in Figure 40.

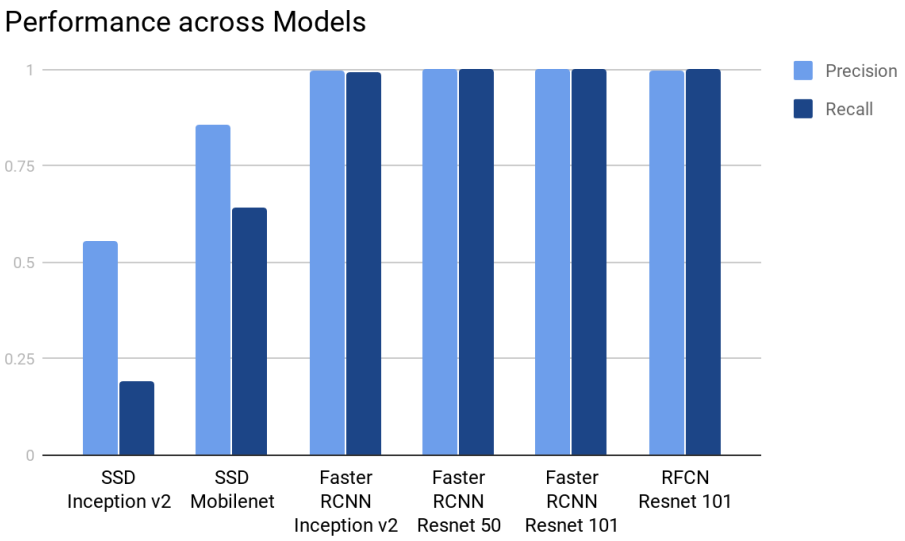

Figure 40: Performance across different architectures

Looking at the average precision and recall across the networks, the choice of the network is quickly reduced to a choice between Faster R-CNN and R-FCN. The ResNet architectures, both the 101-layer and 50-layer versions, performed marginally better than the Inception v2 architecture in both recall and precision. Of these three top-performing networks, Faster R-CNN ResNet 50 was chosen as the network to proceed with using for further evaluation of this problem.

3.2 Augmented Results

All experimental results were run on a Faster R-CNN ResNet 50 network based on a 0.5 IOU threshold. Confusion matrices and tabular data for augmentation runs, where available, are in Appendix B.

3.2.1 Image Size

In order to evaluate network performance as image size decreased, ideal license plate images of different sizes were generated and fed into the network. Significant degradation in performance was not seen prior to the smallest size tested, 32 x 16 pixels. Performance across different image sizes is shown in Figure 41.

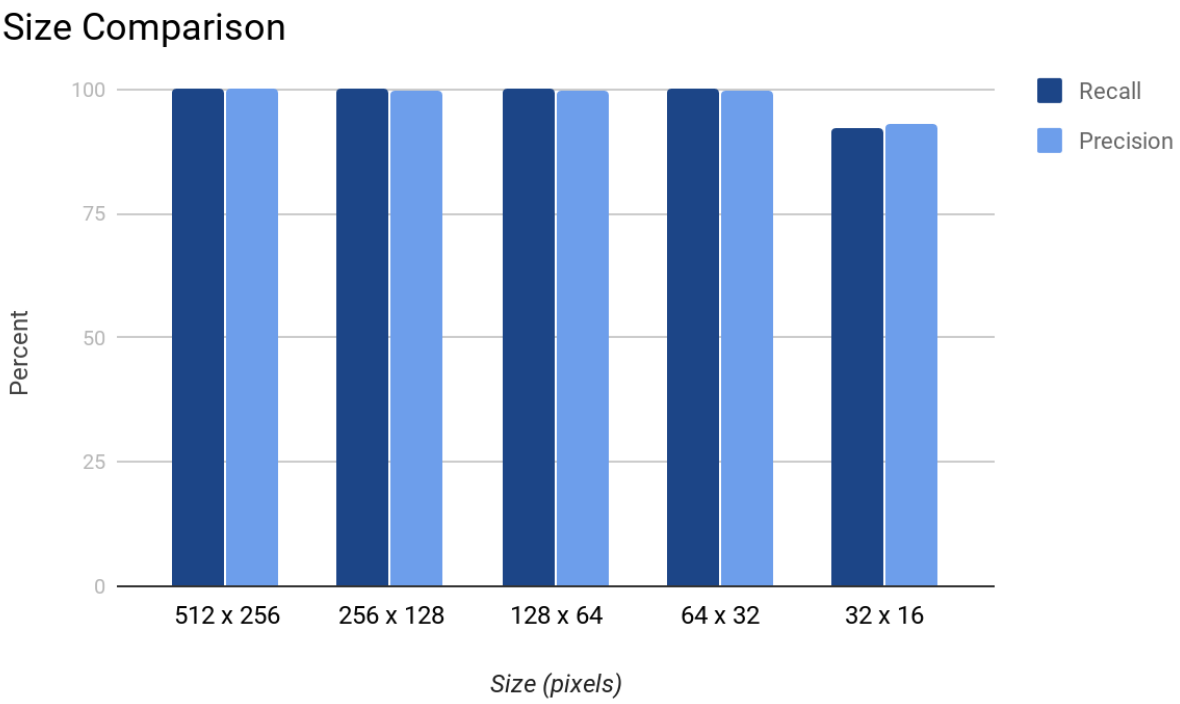

Figure 41: Performance across different image sizes

3.2.2 JPEG Compression

The Faster R-CNN ResNet 50 network was trained on a JPEG Compression augmented dataset and evaluated against a test image dataset at varying levels of JPEG Compression. Performance degrades slightly at the higher levels of JPEG Compression but does not noticeably impact performance. These results indicate that this network, when trained on JPEG

Compressed data, can effectively evaluate license plate images with JPEG Compression artifacts at a number of levels. JPEG compression is usually indicated by a quality metric ranging from 0 to 100 where 100 is perfect with no visible errors and 0 is no visible image. Observed degradation due to lowering the JPEG quality is data-dependent and reflects the amount of noise in the image. Images with high amounts of noise do not compress well. The severity levels 1-5 correspond to the quality levels: 25, 18, 15, 10, and 7 respectively. The synthetically generated images are relatively straight forward and compress well with the JPEG compression algorithm as shown in Figure 42. Performance across the different severity levels is shown in Figure 43.

Figure 42: Test Images at a variety of Imgaug JPEG Compression severity levels. Red boxes are ground truth, blue boxes are detections.

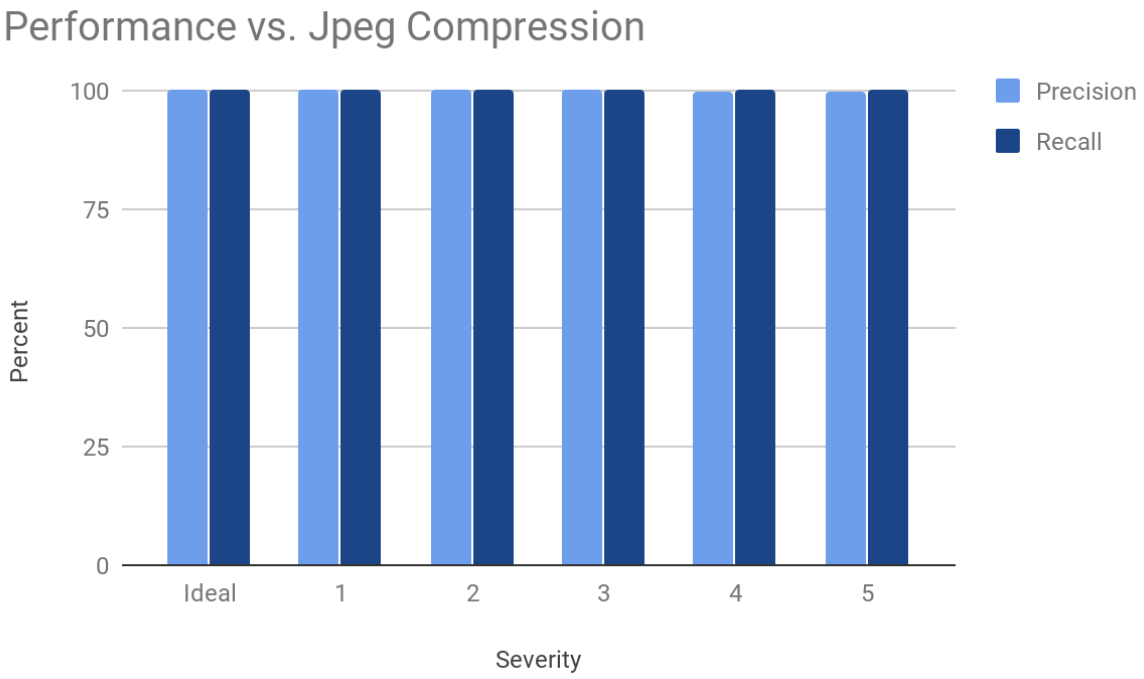

Figure 43: Plot showing performance at different JPEG Compression severity levels

3.2.3 Motion Blur

The Faster R-CNN ResNet 50 network was trained on a dataset augmented with motion blur of a kernel size between 4 and 25 (4 being the minimum kernel size allowed by the Imgaug library). Test data was generated with incremental kernel sizes to evaluate the trained network's performance. Performance remained relatively high across all tested value, although a drop off in performance was seen with the largest motion blur of 25 pixels. Sample images at the different motion blur kernel sizes are seen in Figure 45 and performance with different kernel sizes is shown in Figure 46.

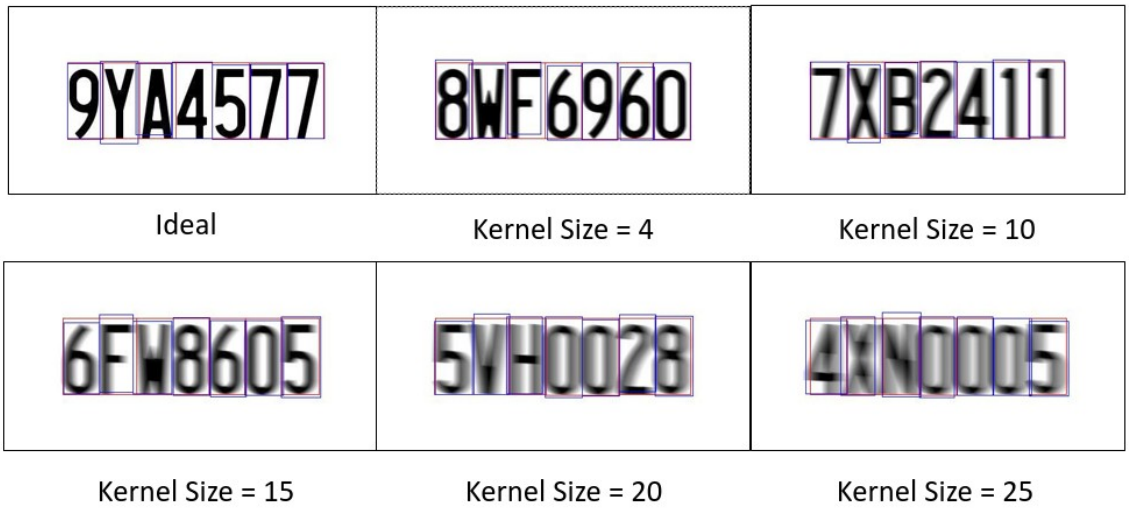

Figure 44: Sample images of license plates augmented with different kernel sizes. Red boxes are ground truth, blue boxes are detections.

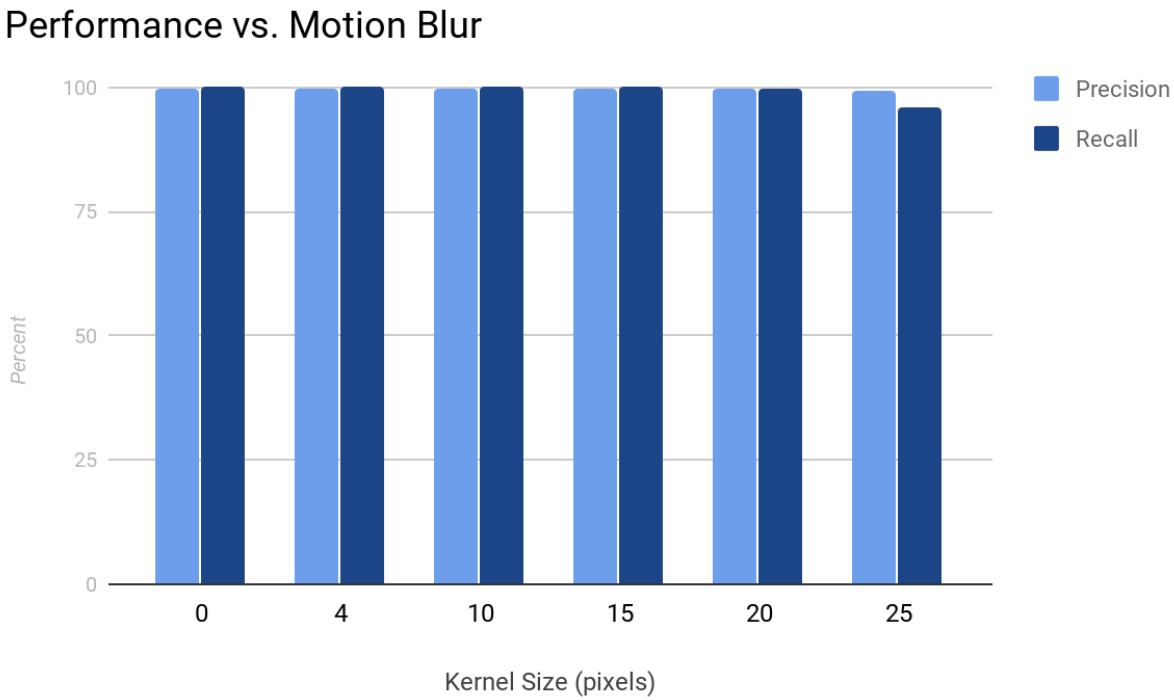

Figure 45: Plot of performance with test images of varying motion blur kernel sizes

3.2.4 Affine Transform

The Faster R-CNN ResNet 50 network was trained on a dataset augmented with affine transforms that scale the license plate image between 50-150% of original size and rotate in the positive and negative direction by +/- 5 degrees. This trained dataset was then tested on a set of test images at each of these transform limits. Samples of these transforms are shown in Figure 46 and plots of the performance are shown in Figure 47.

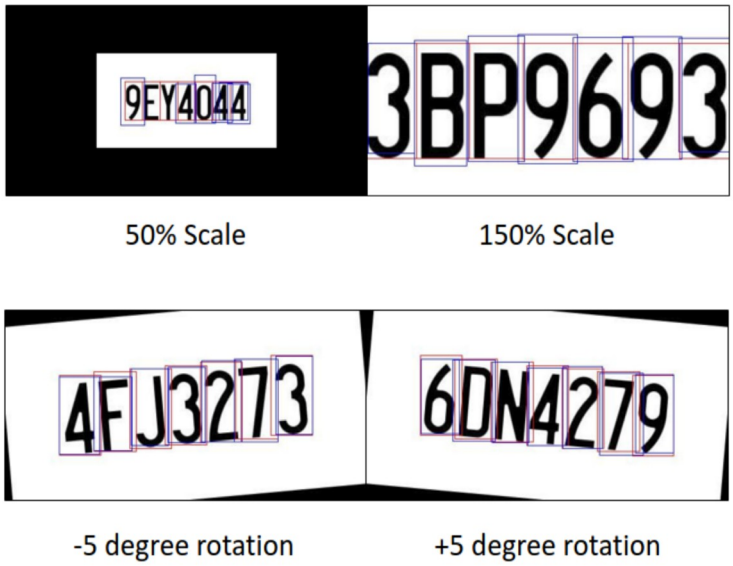

Figure 46: Sample affine transform augmented images. Red boxes are ground truth, blue boxes are detections.

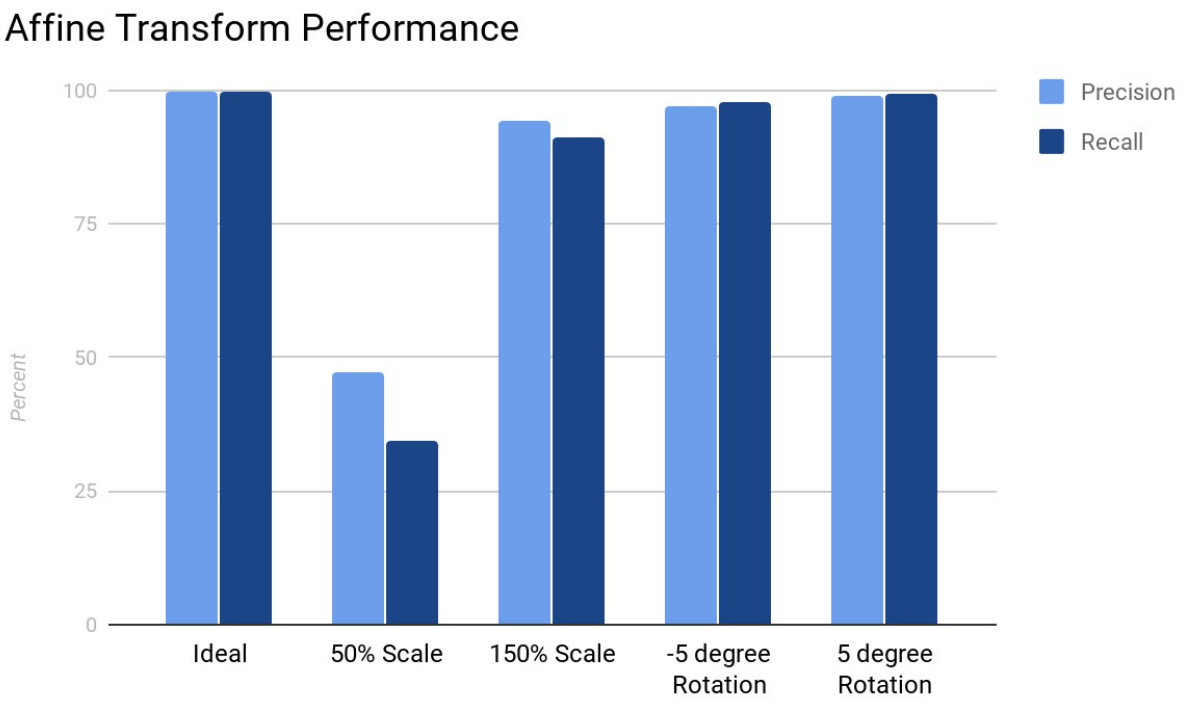

Figure 47: Plot of performance for different affine transform implemented on test images

Out of all data augmentations done thus far, this obviously has the worst performance, particularly the 50% scale images. This is somewhat surprising since when the images were made half size the network had no issue. The difference between the 50% scale and the half-size images is the black border surrounding the images in the 50% scaled images. Although various scaled images fit the problem space as a data augmentation technique this black border that is used when the Imgaug library generates the augmented images doesn't, and since the features of interest in the plate image are the same color as the border it is possible that this is causing part of the issue.

The affine transform further uncovers some limitations of this framework. The network was only trained and tested on 5-degree rotations. This is because, as shown in Figure 48, at higher

rotations the ground truth bounding boxes start to encompass multiple characters limiting their usefulness. Ground truth bounding boxes are defined by corner coordinates, the corner coordinates can also have the affine transform applied to them, but it doesn't account for the straight line that is still drawn between the points rather than on a diagonal as well which would produce better ground truth boxes.

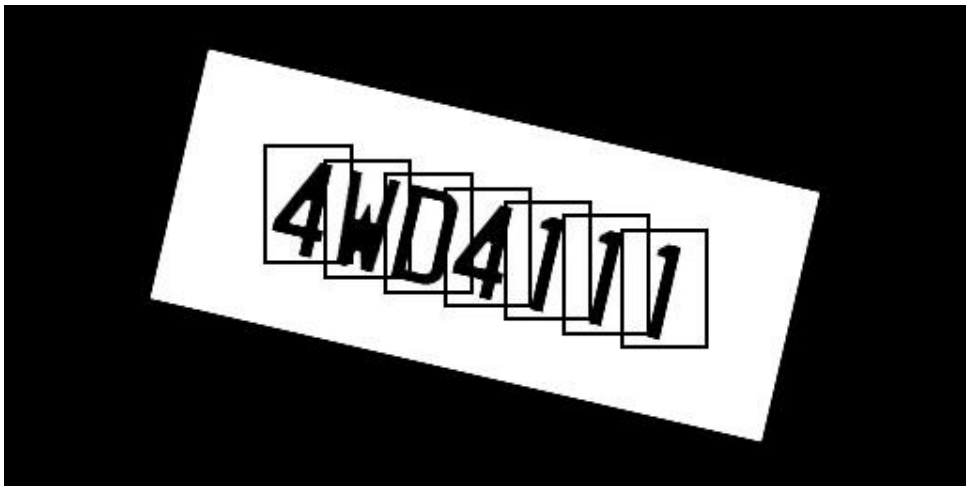

Figure 48: Bounding boxes at higher rotations

3.2.5 Gaussian Noise

The Faster R-CNN ResNet 50 network was trained on a Gaussian noise augmented dataset. Unlike the other data augmentation techniques which were implemented through Imgaug Gaussian noise data augmentation was implemented through the TensorFlow configuration file. Network performance remained high across the different Gaussian noise levels. Sample images at the different levels of Gaussian noise the network was tested against are shown in Figure 49 and performance is shown in Figure 50. Severity levels 1-5 correspond to random additive noise scaled by the magnitudes: 0.08, 0.12, 0.18, 0.26, and 0.38 respectively.

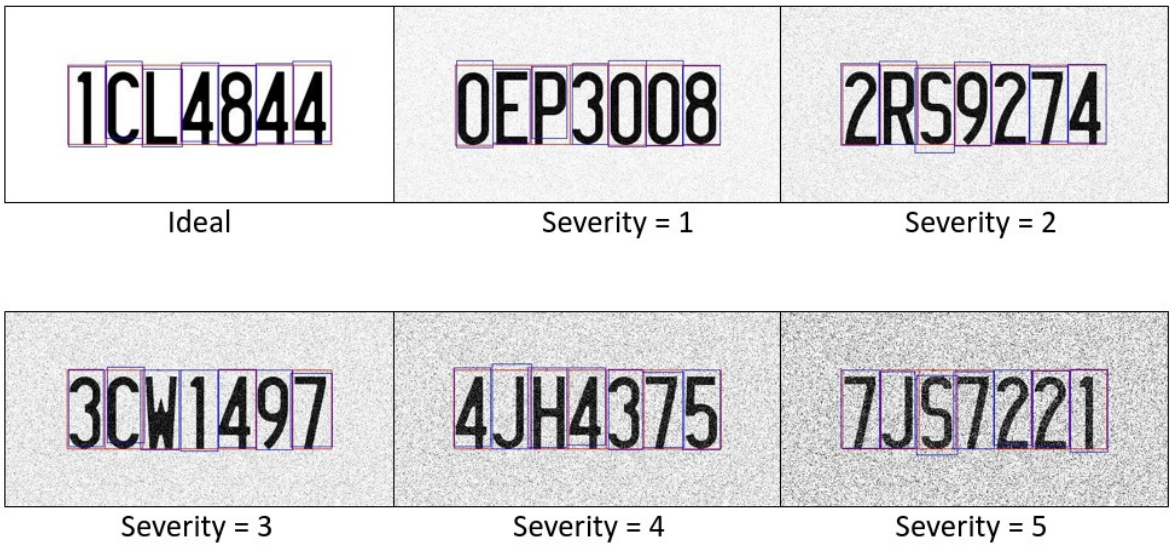

Figure 49: Sample Gaussian noise augmented images at various severity levels. Red boxes are ground truth, blue boxes are detections.

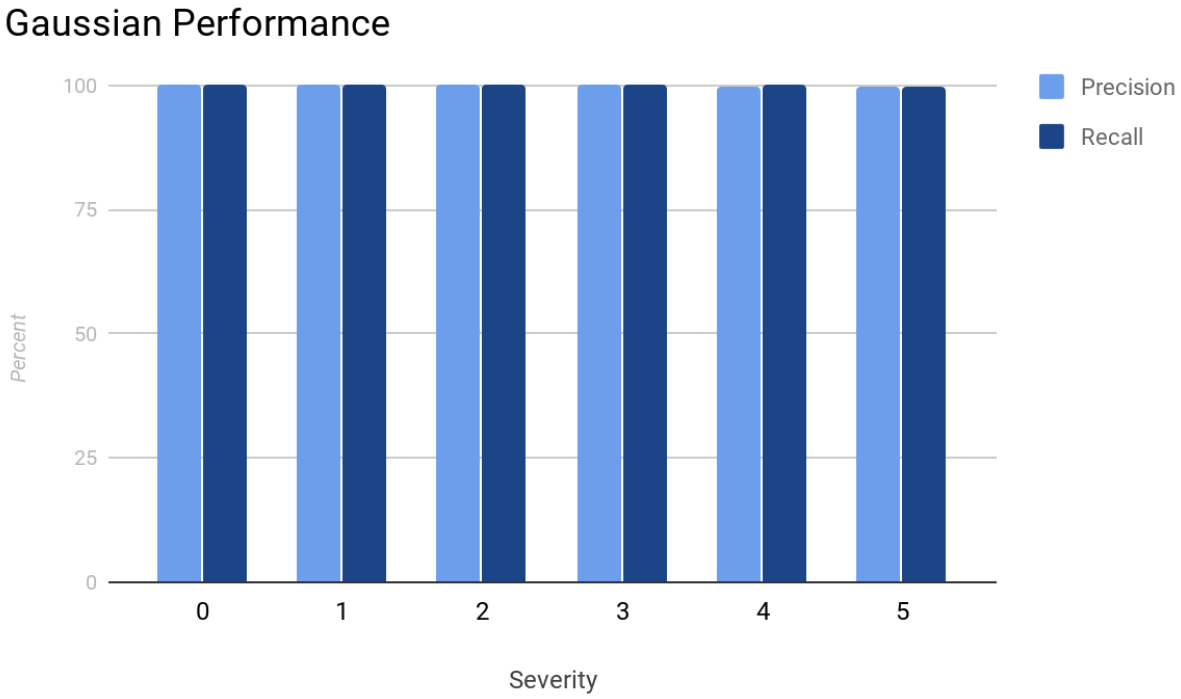

Figure 50: Plot of performance for test image at different levels of Gaussian noise

3.3 Conclusions

In this thesis, a Deep Learning framework has been developed on MARCC to perform character recognition on license plate images of varying qualities. Using this framework, performance was evaluated across network types and applied data transformations. Evaluating Machine Learning models and architectures, it was found that the Faster R-CNN and related R-FCN models along with the ResNet architecture at varying layer depths have the best performance for this specialized object classification task. The lower bound of ideal license plate image size before degradation in performance was found to be 32 x 16 pixels, this is significantly smaller than that of commercial systems [20]. Performance was evaluated for a network trained and tested on images with a number of data augmentation techniques applied. For datasets with Gaussian noise, JPEG compression, or motion blur a slight performance degradation was seen for the low end of the tested quality spectrum but overall the network was able to be trained to be robust to these forms of image degradation. The application of affine transforms, other the other hand, and in particular the 50% scaling, caused larger issues and revealed holes in the overall framework. The 50% scaling case had the worst performance of any data augmentation technique applied. Compared to the half-size images evaluated when the scaling data augmentation is applied the image itself doesn't become smaller rather it is padded – the network did not handle this padding well, potentially because the padding is the same pixel value (black) as the characters. A further issue uncovered with the affine transforms is the limit of the Imgaug library. The Imgaug library has a built-in ground truth bounding box transformer, however, it performs poorly for this dataset at high rotations (above 5 degrees). A solution to this would be to define a custom bounding box transform. This, however, was outside of the scope of this thesis but would be an improvement moving forwards.

This thesis provided a foundational Deep Learning framework for performing character recognition on license plate images on the MARCC system, however, due to scope limitations it was far from a comprehensive analysis and there is much future work that could be explored. The space of data augmentation on this dataset alone allows much to be done. In addition to improving the bounding box transform mentioned above, there are numerous data augmentation techniques that are relevant to this dataset that weren't applied and tested due to scope limitation, Table 2 lists some of these in brief. A fully robust system for performing character recognition on license plates would need to be robust to many more variations of degradation as well as the potential for multiple methods of distortion introduced in one image. This thesis also focused on using license plate images with a white background as a baseline for performance, moving forward analysis on license plates with the traditional image background would be useful and open the door for additional color-dependent data augmentation to be implemented. A final step would be to train and test on real-world images. This comes with a large collection and labeling overhead burden as compared to the synthetic generation work done here, but before deploying a similar system to real-world tasks would have to be performed. While there is much work between this system and one that could be deployed in digital forensics, this work shows that a Deep Learning-based approach has promise for character recognition on license plate images even when those images have undergone degradation.

Appendix A

Source Code

Source code for this project is available on GitHub at: https://github.com/Hriste/AutomaticLP

Appendix B

This appendix contains the available confusion matrices and tabular results for the data augmentation test runs.

B.1 Image Size

Image sizes were varied by powers of two from 512 x 256 pixels to 32 x 16 pixels and the Faster R-CNN ResNet 50 network was trained and tested on these datasets. Tabular results from these evaluation runs are shown below. These results show that network performance does not significantly degrade until the image size reaches the smallest tested; 32 x 16 pixels.

B.1.1 512 x 256

The following Figure 51 shows tabular results of the Faster R-CNN ResNet 50 network when trained and tested on ideal images of the size 512 x 256 pixels. The performance was perfect except for a single false positive on the letter “L”.

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.0	1963.0	0.0	0.0
1	1	1.000000	1.0	1940.0	0.0	0.0
2	2	1.000000	1.0	2013.0	0.0	0.0
3	3	1.000000	1.0	2040.0	0.0	0.0
4	4	1.000000	1.0	2006.0	0.0	0.0
5	5	1.000000	1.0	2012.0	0.0	0.0
6	6	1.000000	1.0	2011.0	0.0	0.0
7	7	1.000000	1.0	2030.0	0.0	0.0
8	8	1.000000	1.0	2035.0	0.0	0.0
9	9	1.000000	1.0	1950.0	0.0	0.0
10	A	1.000000	1.0	369.0	0.0	0.0
11	B	1.000000	1.0	366.0	0.0	0.0
12	C	1.000000	1.0	368.0	0.0	0.0
13	D	1.000000	1.0	342.0	0.0	0.0
14	E	1.000000	1.0	351.0	0.0	0.0
15	F	1.000000	1.0	340.0	0.0	0.0
16	G	1.000000	1.0	373.0	0.0	0.0
17	H	1.000000	1.0	349.0	0.0	0.0
18	J	1.000000	1.0	362.0	0.0	0.0
19	K	1.000000	1.0	385.0	0.0	0.0
20	L	0.997389	1.0	382.0	1.0	0.0
21	M	1.000000	1.0	403.0	0.0	0.0
22	N	1.000000	1.0	348.0	0.0	0.0
23	P	1.000000	1.0	359.0	0.0	0.0
24	R	1.000000	1.0	345.0	0.0	0.0
25	S	1.000000	1.0	353.0	0.0	0.0
26	T	1.000000	1.0	366.0	0.0	0.0
27	V	1.000000	1.0	373.0	0.0	0.0
28	W	1.000000	1.0	352.0	0.0	0.0
29	X	1.000000	1.0	374.0	0.0	0.0
30	Y	1.000000	1.0	379.0	0.0	0.0
31	Z	1.000000	1.0	361.0	0.0	0.0

Figure 51: Tabular results for Faster R-CNN ResNet 50 on images of size 512x256 pixels

B.1.2 256 x 128

The following Figure 52 shows tabular results of the Faster R-CNN ResNet 50 network when trained and tested on ideal images of the size 256 x 128 pixels. Recall was perfect, and precision was in the high 90%s to 100% with small amounts of false positives for “L”, “F”, and “Y”.

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.0	420.0	0.0	0.0
1	1	1.000000	1.0	399.0	0.0	0.0
2	2	1.000000	1.0	387.0	0.0	0.0
3	3	1.000000	1.0	394.0	0.0	0.0
4	4	1.000000	1.0	385.0	0.0	0.0
5	5	1.000000	1.0	390.0	0.0	0.0
6	6	1.000000	1.0	438.0	0.0	0.0
7	7	1.000000	1.0	400.0	0.0	0.0
8	8	1.000000	1.0	368.0	0.0	0.0
9	9	1.000000	1.0	419.0	0.0	0.0
10	A	1.000000	1.0	62.0	0.0	0.0
11	B	1.000000	1.0	61.0	0.0	0.0
12	C	1.000000	1.0	87.0	0.0	0.0
13	D	1.000000	1.0	63.0	0.0	0.0
14	E	1.000000	1.0	65.0	0.0	0.0
15	F	0.982759	1.0	57.0	1.0	0.0
16	G	1.000000	1.0	64.0	0.0	0.0
17	H	1.000000	1.0	84.0	0.0	0.0
18	J	1.000000	1.0	65.0	0.0	0.0
19	K	1.000000	1.0	74.0	0.0	0.0
20	L	0.967033	1.0	88.0	3.0	0.0
21	M	1.000000	1.0	73.0	0.0	0.0
22	N	1.000000	1.0	69.0	0.0	0.0
23	P	1.000000	1.0	73.0	0.0	0.0
24	R	1.000000	1.0	72.0	0.0	0.0
25	S	1.000000	1.0	73.0	0.0	0.0
26	T	1.000000	1.0	93.0	0.0	0.0
27	V	1.000000	1.0	69.0	0.0	0.0
28	W	1.000000	1.0	81.0	0.0	0.0
29	X	1.000000	1.0	74.0	0.0	0.0
30	Y	0.987500	1.0	79.0	1.0	0.0
31	Z	1.000000	1.0	74.0	0.0	0.0

Figure 52: Tabular results for Faster R-CNN ResNet 50 on images of size 256x128 pixels

B.1.3 128 x 64

The following Figure 53 shows tabular results of the Faster R-CNN ResNet 50 network when trained and tested on ideal images of the size 128 x 64 pixels. Recall was perfect, and precision was in the high 90%s to 100% with small amounts of false positives for “L”, “V”, and “W”.

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.0	404.0	0.0	0.0
1	1	1.000000	1.0	390.0	0.0	0.0
2	2	1.000000	1.0	376.0	0.0	0.0
3	3	1.000000	1.0	416.0	0.0	0.0
4	4	1.000000	1.0	385.0	0.0	0.0
5	5	1.000000	1.0	405.0	0.0	0.0
6	6	1.000000	1.0	425.0	0.0	0.0
7	7	1.000000	1.0	437.0	0.0	0.0
8	8	1.000000	1.0	375.0	0.0	0.0
9	9	1.000000	1.0	387.0	0.0	0.0
10	A	1.000000	1.0	78.0	0.0	0.0
11	B	1.000000	1.0	62.0	0.0	0.0
12	C	1.000000	1.0	55.0	0.0	0.0
13	D	1.000000	1.0	71.0	0.0	0.0
14	E	1.000000	1.0	83.0	0.0	0.0
15	F	1.000000	1.0	80.0	0.0	0.0
16	G	1.000000	1.0	86.0	0.0	0.0
17	H	1.000000	1.0	75.0	0.0	0.0
18	J	1.000000	1.0	73.0	0.0	0.0
19	K	1.000000	1.0	75.0	0.0	0.0
20	L	0.975610	1.0	80.0	2.0	0.0
21	M	1.000000	1.0	77.0	0.0	0.0
22	N	1.000000	1.0	75.0	0.0	0.0
23	P	1.000000	1.0	69.0	0.0	0.0
24	R	1.000000	1.0	71.0	0.0	0.0
25	S	1.000000	1.0	75.0	0.0	0.0
26	T	1.000000	1.0	71.0	0.0	0.0
27	V	0.936508	1.0	59.0	4.0	0.0
28	W	0.973333	1.0	73.0	2.0	0.0
29	X	1.000000	1.0	62.0	0.0	0.0
30	Y	1.000000	1.0	68.0	0.0	0.0
31	Z	1.000000	1.0	82.0	0.0	0.0

Figure 53: Tabular results for Faster R-CNN ResNet 50 on images of size 128x64 pixels

B.1.4 64 x 32

The following Figure 54 shows tabular results of the Faster R-CNN ResNet 50 network when trained and tested on ideal images of the size 64 x 32 pixels. The performance was perfect except for a single false positive on the letter “F”.

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.0	406.0	0.0	0.0
1	1	1.000000	1.0	399.0	0.0	0.0
2	2	1.000000	1.0	391.0	0.0	0.0
3	3	1.000000	1.0	398.0	0.0	0.0
4	4	1.000000	1.0	399.0	0.0	0.0
5	5	1.000000	1.0	382.0	0.0	0.0
6	6	1.000000	1.0	411.0	0.0	0.0
7	7	1.000000	1.0	403.0	0.0	0.0
8	8	1.000000	1.0	417.0	0.0	0.0
9	9	1.000000	1.0	394.0	0.0	0.0
10	A	1.000000	1.0	60.0	0.0	0.0
11	B	1.000000	1.0	90.0	0.0	0.0
12	C	1.000000	1.0	95.0	0.0	0.0
13	D	1.000000	1.0	64.0	0.0	0.0
14	E	1.000000	1.0	74.0	0.0	0.0
15	F	0.986301	1.0	72.0	1.0	0.0
16	G	1.000000	1.0	63.0	0.0	0.0
17	H	1.000000	1.0	78.0	0.0	0.0
18	J	1.000000	1.0	64.0	0.0	0.0
19	K	1.000000	1.0	69.0	0.0	0.0
20	L	1.000000	1.0	69.0	0.0	0.0
21	M	1.000000	1.0	66.0	0.0	0.0
22	N	1.000000	1.0	76.0	0.0	0.0
23	P	1.000000	1.0	65.0	0.0	0.0
24	R	1.000000	1.0	78.0	0.0	0.0
25	S	1.000000	1.0	79.0	0.0	0.0
26	T	1.000000	1.0	75.0	0.0	0.0
27	V	1.000000	1.0	64.0	0.0	0.0
28	W	1.000000	1.0	80.0	0.0	0.0
29	X	1.000000	1.0	74.0	0.0	0.0
30	Y	1.000000	1.0	63.0	0.0	0.0
31	Z	1.000000	1.0	82.0	0.0	0.0

Figure 54: Tabular results for Faster R-CNN ResNet 50 on images of size 64x32 pixels

B.1.5 32 x 16

The following Figure 55 shows tabular results of the Faster R-CNN ResNet 50 network when trained and tested on ideal images of the size 32 x 16 pixels. This is the image size where performance degradation begins to occur with a substantial number of false negatives and positives.

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	0.995012	1.000000	399.0	2.0	0.0
1	1	0.992629	1.000000	404.0	3.0	0.0
2	2	1.000000	1.000000	443.0	0.0	0.0
3	3	0.989418	1.000000	374.0	4.0	0.0
4	4	1.000000	1.000000	388.0	0.0	0.0
5	5	0.926606	1.000000	404.0	32.0	0.0
6	6	1.000000	1.000000	379.0	0.0	0.0
7	7	1.000000	1.000000	405.0	0.0	0.0
8	8	0.991979	1.000000	371.0	3.0	0.0
9	9	1.000000	1.000000	433.0	0.0	0.0
10	A	0.965116	1.000000	83.0	3.0	0.0
11	B	1.000000	1.000000	65.0	0.0	0.0
12	C	1.000000	0.822785	65.0	0.0	14.0
13	D	0.986111	0.986111	71.0	1.0	1.0
14	E	1.000000	1.000000	78.0	0.0	0.0
15	F	1.000000	1.000000	75.0	0.0	0.0
16	G	1.000000	0.640000	48.0	0.0	27.0
17	H	0.958904	1.000000	70.0	3.0	0.0
18	J	0.935065	0.947368	72.0	5.0	4.0
19	K	0.916667	0.974684	77.0	7.0	2.0
20	L	1.000000	0.220000	11.0	0.0	39.0
21	M	0.785047	1.000000	84.0	23.0	0.0
22	N	0.986111	0.887500	71.0	1.0	9.0
23	P	0.676471	1.000000	69.0	33.0	0.0
24	R	1.000000	1.000000	62.0	0.0	0.0
25	S	1.000000	0.202703	15.0	0.0	59.0
26	T	0.972222	0.864198	70.0	2.0	11.0
27	V	0.982143	0.982143	55.0	1.0	1.0
28	W	0.829787	1.000000	78.0	16.0	0.0
29	X	1.000000	1.000000	67.0	0.0	0.0
30	Y	0.576923	1.000000	75.0	55.0	0.0
31	Z	1.000000	1.000000	72.0	0.0	0.0

Figure 55: Tabular results for Faster R-CNN ResNet 50 on images of size 32x16 pixels

B.2 JPEG Compression

JPEG Compression data augmentation was applied while varying the severity, as specified in the Imgaug library from 1-5. The severity levels 1-5 correspond to the quality levels; 25, 18, 15, 10, and 7 respectively. The Faster R-CNN ResNet 50 network was trained and tested on these datasets. Performance degrades slightly at the higher levels of JPEG Compression but does not noticeably impact performance. These results indicate that this network, when trained on JPEG Compressed data, can effectively evaluate license plate images with JPEG Compression artifacts at a number of levels.

B.2.1 Ideal

Figure 56 and Figure 57 show results for an ideal dataset with no JPEG Compression applied.

Recall and precision both reach 100% for this dataset.

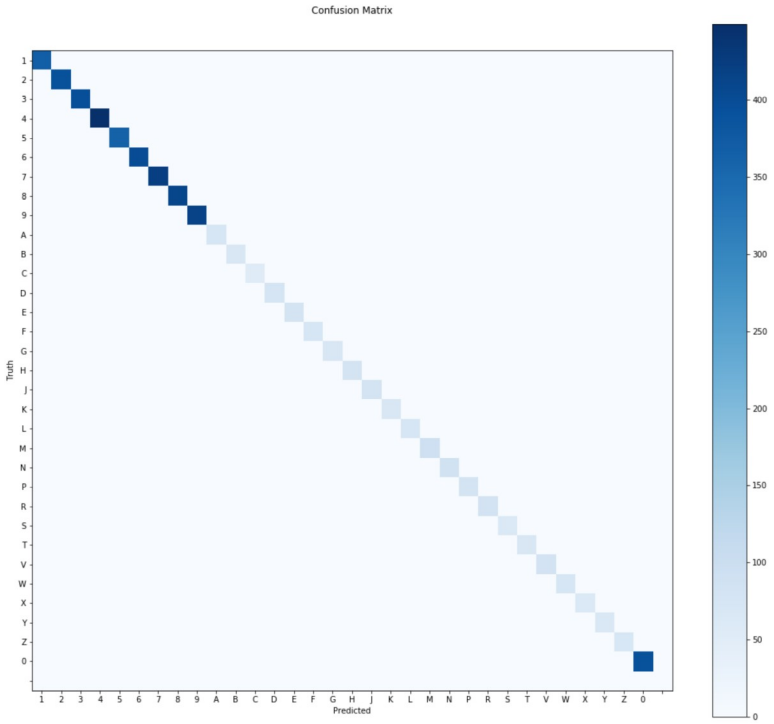

Figure 56: Confusion Matrix for Faster R-CNN ResNet 50 on images with no JPEG Compression applied

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.0	1.0	391.0	0.0	0.0
1	1	1.0	1.0	368.0	0.0	0.0
2	2	1.0	1.0	390.0	0.0	0.0
3	3	1.0	1.0	396.0	0.0	0.0
4	4	1.0	1.0	449.0	0.0	0.0
5	5	1.0	1.0	363.0	0.0	0.0
6	6	1.0	1.0	399.0	0.0	0.0
7	7	1.0	1.0	421.0	0.0	0.0
8	8	1.0	1.0	409.0	0.0	0.0
9	9	1.0	1.0	414.0	0.0	0.0
10	A	1.0	1.0	71.0	0.0	0.0
11	B	1.0	1.0	68.0	0.0	0.0
12	C	1.0	1.0	53.0	0.0	0.0
13	D	1.0	1.0	75.0	0.0	0.0
14	E	1.0	1.0	78.0	0.0	0.0
15	F	1.0	1.0	72.0	0.0	0.0
16	G	1.0	1.0	67.0	0.0	0.0
17	H	1.0	1.0	80.0	0.0	0.0
18	J	1.0	1.0	78.0	0.0	0.0
19	K	1.0	1.0	67.0	0.0	0.0
20	L	1.0	1.0	72.0	0.0	0.0
21	M	1.0	1.0	92.0	0.0	0.0
22	N	1.0	1.0	85.0	0.0	0.0
23	P	1.0	1.0	78.0	0.0	0.0
24	R	1.0	1.0	82.0	0.0	0.0
25	S	1.0	1.0	63.0	0.0	0.0
26	T	1.0	1.0	68.0	0.0	0.0
27	V	1.0	1.0	82.0	0.0	0.0
28	W	1.0	1.0	71.0	0.0	0.0
29	X	1.0	1.0	63.0	0.0	0.0
30	Y	1.0	1.0	65.0	0.0	0.0
31	Z	1.0	1.0	70.0	0.0	0.0

Figure 57: Tabular results for Faster R-CNN ResNet 50 on images with no JPEG Compression applied

B.2.2 Severity 1

Figure 58 and Figure 59 show results for a Faster R-CNN ResNet 50 network trained and tested on a dataset with JPEG Compression of severity 1. JPEG Severity 1 in the Imgaug library corresponds to a JPEG quality of 25. For this dataset recall and precision were both 100%.

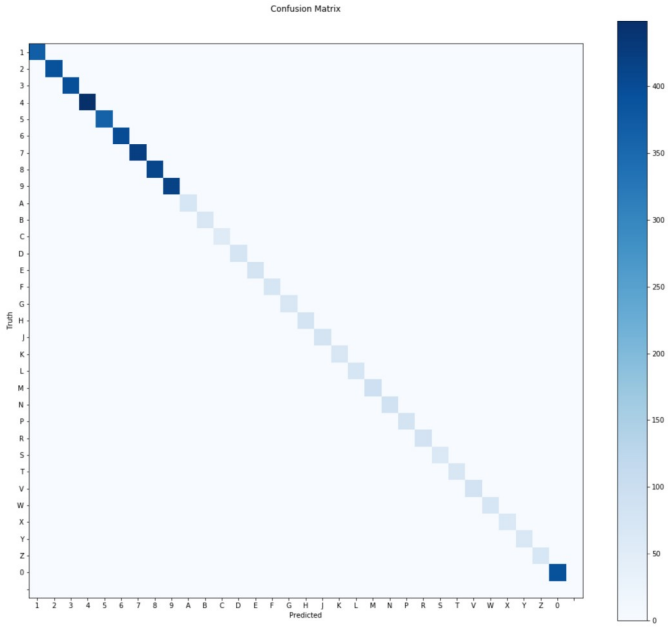

Figure 58: Confusion Matrix for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 1 (Quality 25)

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.0	1.0	391.0	0.0	0.0
1	1	1.0	1.0	368.0	0.0	0.0
2	2	1.0	1.0	390.0	0.0	0.0
3	3	1.0	1.0	396.0	0.0	0.0
4	4	1.0	1.0	449.0	0.0	0.0
5	5	1.0	1.0	363.0	0.0	0.0
6	6	1.0	1.0	399.0	0.0	0.0
7	7	1.0	1.0	421.0	0.0	0.0
8	8	1.0	1.0	409.0	0.0	0.0
9	9	1.0	1.0	414.0	0.0	0.0
10	A	1.0	1.0	71.0	0.0	0.0
11	B	1.0	1.0	68.0	0.0	0.0
12	C	1.0	1.0	53.0	0.0	0.0
13	D	1.0	1.0	75.0	0.0	0.0
14	E	1.0	1.0	78.0	0.0	0.0
15	F	1.0	1.0	72.0	0.0	0.0
16	G	1.0	1.0	67.0	0.0	0.0
17	H	1.0	1.0	80.0	0.0	0.0
18	J	1.0	1.0	78.0	0.0	0.0
19	K	1.0	1.0	67.0	0.0	0.0
20	L	1.0	1.0	72.0	0.0	0.0
21	M	1.0	1.0	92.0	0.0	0.0
22	N	1.0	1.0	85.0	0.0	0.0
23	P	1.0	1.0	78.0	0.0	0.0
24	R	1.0	1.0	82.0	0.0	0.0
25	S	1.0	1.0	63.0	0.0	0.0
26	T	1.0	1.0	68.0	0.0	0.0
27	V	1.0	1.0	82.0	0.0	0.0
28	W	1.0	1.0	71.0	0.0	0.0
29	X	1.0	1.0	63.0	0.0	0.0
30	Y	1.0	1.0	65.0	0.0	0.0
31	Z	1.0	1.0	70.0	0.0	0.0

Figure 59: Tabular results for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 1 (Quality 25)

B.2.3 Severity 2

Figure 60 and Figure 61 show results for a Faster R-CNN ResNet 50 network trained and tested on a dataset with JPEG Compression of severity 2. JPEG Severity 2 in the Imgaug library corresponds to a JPEG quality of 18. For this dataset recall and precision were both 100%.

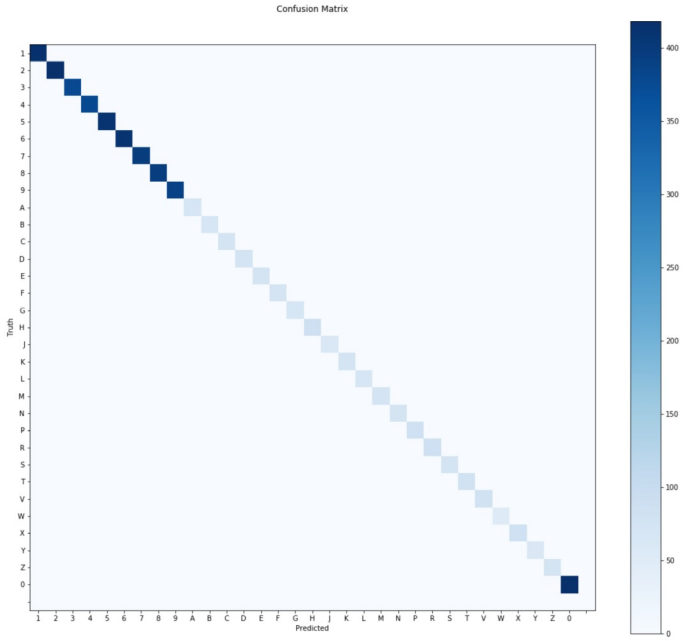

Figure 60: Confusion Matrix for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 2 (Quality 18)

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.0	1.0	418.0	0.0	0.0
1	1	1.0	1.0	414.0	0.0	0.0
2	2	1.0	1.0	417.0	0.0	0.0
3	3	1.0	1.0	377.0	0.0	0.0
4	4	1.0	1.0	375.0	0.0	0.0
5	5	1.0	1.0	410.0	0.0	0.0
6	6	1.0	1.0	410.0	0.0	0.0
7	7	1.0	1.0	396.0	0.0	0.0
8	8	1.0	1.0	395.0	0.0	0.0
9	9	1.0	1.0	388.0	0.0	0.0
10	A	1.0	1.0	67.0	0.0	0.0
11	B	1.0	1.0	68.0	0.0	0.0
12	C	1.0	1.0	72.0	0.0	0.0
13	D	1.0	1.0	73.0	0.0	0.0
14	E	1.0	1.0	72.0	0.0	0.0
15	F	1.0	1.0	75.0	0.0	0.0
16	G	1.0	1.0	67.0	0.0	0.0
17	H	1.0	1.0	85.0	0.0	0.0
18	J	1.0	1.0	60.0	0.0	0.0
19	K	1.0	1.0	74.0	0.0	0.0
20	L	1.0	1.0	67.0	0.0	0.0
21	M	1.0	1.0	73.0	0.0	0.0
22	N	1.0	1.0	73.0	0.0	0.0
23	P	1.0	1.0	84.0	0.0	0.0
24	R	1.0	1.0	88.0	0.0	0.0
25	S	1.0	1.0	75.0	0.0	0.0
26	T	1.0	1.0	79.0	0.0	0.0
27	V	1.0	1.0	79.0	0.0	0.0
28	W	1.0	1.0	52.0	0.0	0.0
29	X	1.0	1.0	84.0	0.0	0.0
30	Y	1.0	1.0	61.0	0.0	0.0
31	Z	1.0	1.0	72.0	0.0	0.0

Figure 61: Tabular results for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 2 (Quality 18)

B.2.4 Severity 3

Figure 62 and Figure 63 show results for a Faster R-CNN ResNet 50 network trained and tested on a dataset with JPEG Compression of severity 3. JPEG Severity 3 in the Imgaug library corresponds to a JPEG quality of 15. For this dataset recall and precision were both 100%.

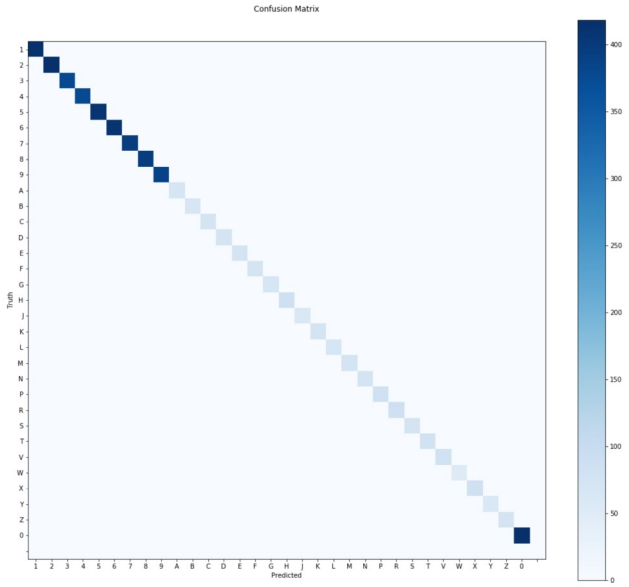

Figure 62: Confusion Matrix for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 3 (Quality 15)

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.0	1.0	418.0	0.0	0.0
1	1	1.0	1.0	414.0	0.0	0.0
2	2	1.0	1.0	417.0	0.0	0.0
3	3	1.0	1.0	377.0	0.0	0.0
4	4	1.0	1.0	375.0	0.0	0.0
5	5	1.0	1.0	410.0	0.0	0.0
6	6	1.0	1.0	410.0	0.0	0.0
7	7	1.0	1.0	396.0	0.0	0.0
8	8	1.0	1.0	395.0	0.0	0.0
9	9	1.0	1.0	388.0	0.0	0.0
10	A	1.0	1.0	67.0	0.0	0.0
11	B	1.0	1.0	68.0	0.0	0.0
12	C	1.0	1.0	72.0	0.0	0.0
13	D	1.0	1.0	73.0	0.0	0.0
14	E	1.0	1.0	72.0	0.0	0.0
15	F	1.0	1.0	75.0	0.0	0.0
16	G	1.0	1.0	67.0	0.0	0.0
17	H	1.0	1.0	85.0	0.0	0.0
18	J	1.0	1.0	60.0	0.0	0.0
19	K	1.0	1.0	74.0	0.0	0.0
20	L	1.0	1.0	67.0	0.0	0.0
21	M	1.0	1.0	73.0	0.0	0.0
22	N	1.0	1.0	73.0	0.0	0.0
23	P	1.0	1.0	84.0	0.0	0.0
24	R	1.0	1.0	88.0	0.0	0.0
25	S	1.0	1.0	75.0	0.0	0.0
26	T	1.0	1.0	79.0	0.0	0.0
27	V	1.0	1.0	79.0	0.0	0.0
28	W	1.0	1.0	52.0	0.0	0.0
29	X	1.0	1.0	84.0	0.0	0.0
30	Y	1.0	1.0	61.0	0.0	0.0
31	Z	1.0	1.0	72.0	0.0	0.0

Figure 63: Tabular results for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 3 (Quality 15)

B.2.5 Severity 4

Figure 64 and Figure 65 show results for a Faster R-CNN ResNet 50 network trained and tested on a dataset with JPEG Compression of severity 4. JPEG Severity 4 in the Imgaug library corresponds to a JPEG quality of 10. For this dataset recall was 100% but there was one false positive for character “G”.

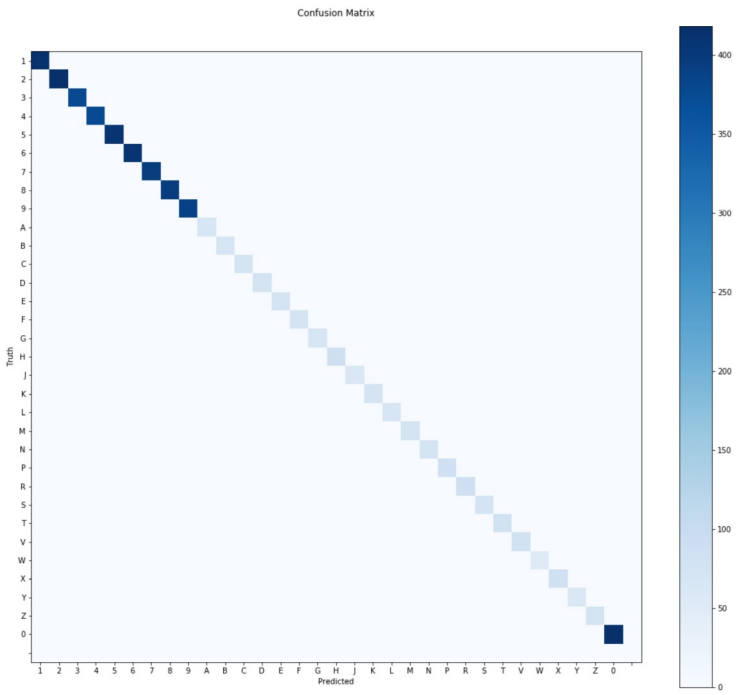

Figure 64: Confusion Matrix for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 4 (Quality 10)

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.0	418.0	0.0	0.0
1	1	1.000000	1.0	414.0	0.0	0.0
2	2	1.000000	1.0	417.0	0.0	0.0
3	3	1.000000	1.0	377.0	0.0	0.0
4	4	1.000000	1.0	375.0	0.0	0.0
5	5	1.000000	1.0	410.0	0.0	0.0
6	6	1.000000	1.0	410.0	0.0	0.0
7	7	1.000000	1.0	396.0	0.0	0.0
8	8	1.000000	1.0	395.0	0.0	0.0
9	9	1.000000	1.0	388.0	0.0	0.0
10	A	1.000000	1.0	67.0	0.0	0.0
11	B	1.000000	1.0	68.0	0.0	0.0
12	C	1.000000	1.0	72.0	0.0	0.0
13	D	1.000000	1.0	73.0	0.0	0.0
14	E	1.000000	1.0	72.0	0.0	0.0
15	F	1.000000	1.0	75.0	0.0	0.0
16	G	0.985294	1.0	67.0	1.0	0.0
17	H	1.000000	1.0	85.0	0.0	0.0
18	J	1.000000	1.0	60.0	0.0	0.0
19	K	1.000000	1.0	74.0	0.0	0.0
20	L	1.000000	1.0	67.0	0.0	0.0
21	M	1.000000	1.0	73.0	0.0	0.0
22	N	1.000000	1.0	73.0	0.0	0.0
23	P	1.000000	1.0	84.0	0.0	0.0
24	R	1.000000	1.0	88.0	0.0	0.0
25	S	1.000000	1.0	75.0	0.0	0.0
26	T	1.000000	1.0	79.0	0.0	0.0
27	V	1.000000	1.0	79.0	0.0	0.0
28	W	1.000000	1.0	52.0	0.0	0.0
29	X	1.000000	1.0	84.0	0.0	0.0
30	Y	1.000000	1.0	61.0	0.0	0.0
31	Z	1.000000	1.0	72.0	0.0	0.0

Figure 65: Tabular results for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 4 (Quality 10)

B.2.6 Severity 5

Figure 64 and Figure 65 show results for a Faster R-CNN ResNet 50 network trained and tested on a dataset with JPEG Compression of severity 5. JPEG Severity 5 in the Imgaug library corresponds to a JPEG quality of 7. For this dataset recall was 100% but there was one false positive for the character “T”.

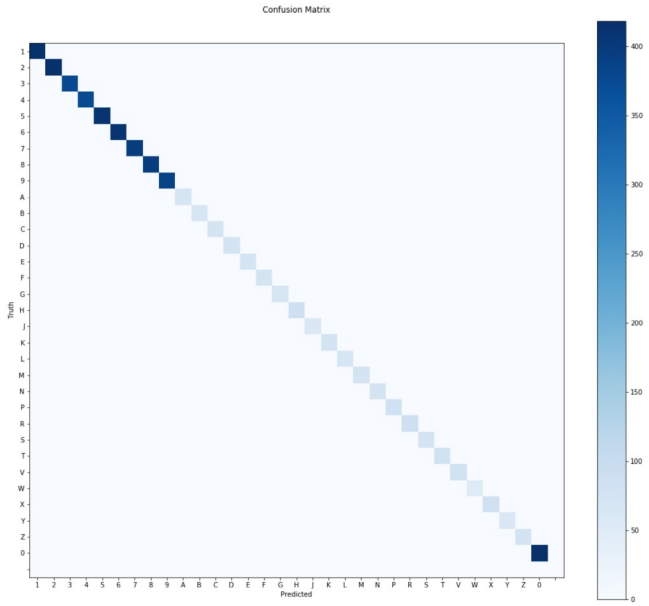

Figure 66: Confusion Matrix for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 5 (Quality 7)

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.0000	1.0	418.0	0.0	0.0
1	1	1.0000	1.0	414.0	0.0	0.0
2	2	1.0000	1.0	417.0	0.0	0.0
3	3	1.0000	1.0	377.0	0.0	0.0
4	4	1.0000	1.0	375.0	0.0	0.0
5	5	1.0000	1.0	410.0	0.0	0.0
6	6	1.0000	1.0	410.0	0.0	0.0
7	7	1.0000	1.0	396.0	0.0	0.0
8	8	1.0000	1.0	395.0	0.0	0.0
9	9	1.0000	1.0	388.0	0.0	0.0
10	A	1.0000	1.0	67.0	0.0	0.0
11	B	1.0000	1.0	68.0	0.0	0.0
12	C	1.0000	1.0	72.0	0.0	0.0
13	D	1.0000	1.0	73.0	0.0	0.0
14	E	1.0000	1.0	72.0	0.0	0.0
15	F	1.0000	1.0	75.0	0.0	0.0
16	G	1.0000	1.0	67.0	0.0	0.0
17	H	1.0000	1.0	85.0	0.0	0.0
18	J	1.0000	1.0	60.0	0.0	0.0
19	K	1.0000	1.0	74.0	0.0	0.0
20	L	1.0000	1.0	67.0	0.0	0.0
21	M	1.0000	1.0	73.0	0.0	0.0
22	N	1.0000	1.0	73.0	0.0	0.0
23	P	1.0000	1.0	84.0	0.0	0.0
24	R	1.0000	1.0	88.0	0.0	0.0
25	S	1.0000	1.0	75.0	0.0	0.0
26	T	0.9875	1.0	79.0	1.0	0.0
27	V	1.0000	1.0	79.0	0.0	0.0
28	W	1.0000	1.0	52.0	0.0	0.0
29	X	1.0000	1.0	84.0	0.0	0.0
30	Y	1.0000	1.0	61.0	0.0	0.0
31	Z	1.0000	1.0	72.0	0.0	0.0

Figure 67: Tabular results for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 5 (Quality 7)

B.3 Motion Blur

Motion blur data augmentation was applied while varying the kernel size from 4 pixels to 25 pixels. Motion blur is an image degradation that is caused by the relative movement of objects in the field of view of the camera while the image is being captured. Performance remained relatively high across all tested value, although a drop off in performance was seen with the largest motion blur of 25 pixels.

B.3.1 Ideal

Figure 68 and Figure 69 show results for an ideal dataset with no Motion blur applied. Recall was 100% for this dataset, but there were false positives for “E” and “J”.

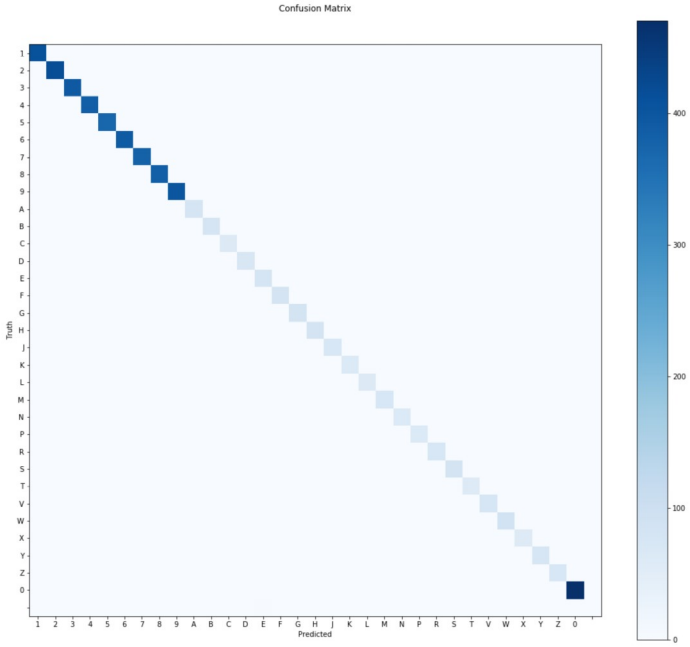

Figure 68: Confusion Matrix for Faster R-CNN ResNet 50 on images with no motion blur

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.0	470.0	0.0	0.0
1	1	1.000000	1.0	408.0	0.0	0.0
2	2	1.000000	1.0	416.0	0.0	0.0
3	3	1.000000	1.0	394.0	0.0	0.0
4	4	1.000000	1.0	384.0	0.0	0.0
5	5	1.000000	1.0	371.0	0.0	0.0
6	6	1.000000	1.0	391.0	0.0	0.0
7	7	1.000000	1.0	380.0	0.0	0.0
8	8	1.000000	1.0	384.0	0.0	0.0
9	9	1.000000	1.0	402.0	0.0	0.0
10	A	1.000000	1.0	80.0	0.0	0.0
11	B	1.000000	1.0	78.0	0.0	0.0
12	C	1.000000	1.0	62.0	0.0	0.0
13	D	1.000000	1.0	71.0	0.0	0.0
14	E	0.975309	1.0	79.0	2.0	0.0
15	F	1.000000	1.0	81.0	0.0	0.0
16	G	1.000000	1.0	84.0	0.0	0.0
17	H	1.000000	1.0	83.0	0.0	0.0
18	J	0.986486	1.0	73.0	1.0	0.0
19	K	1.000000	1.0	64.0	0.0	0.0
20	L	1.000000	1.0	61.0	0.0	0.0
21	M	1.000000	1.0	73.0	0.0	0.0
22	N	1.000000	1.0	64.0	0.0	0.0
23	P	1.000000	1.0	63.0	0.0	0.0
24	R	1.000000	1.0	72.0	0.0	0.0
25	S	1.000000	1.0	84.0	0.0	0.0
26	T	1.000000	1.0	58.0	0.0	0.0
27	V	1.000000	1.0	76.0	0.0	0.0
28	W	1.000000	1.0	88.0	0.0	0.0
29	X	1.000000	1.0	58.0	0.0	0.0
30	Y	1.000000	1.0	76.0	0.0	0.0
31	Z	1.000000	1.0	72.0	0.0	0.0

Figure 69: Tabular results for Faster R-CNN ResNet 50 on images with no motion blur

B.3.2 Kernel Size 4 pixels

Figure 70 and Figure 71 show results for a Faster R-CNN ResNet 50 network trained and tested on a dataset with a motion blur simulated with a kernel size of 4 pixels. For this dataset recall was 100% but there were false positives for “E”, “J”, and “S”.

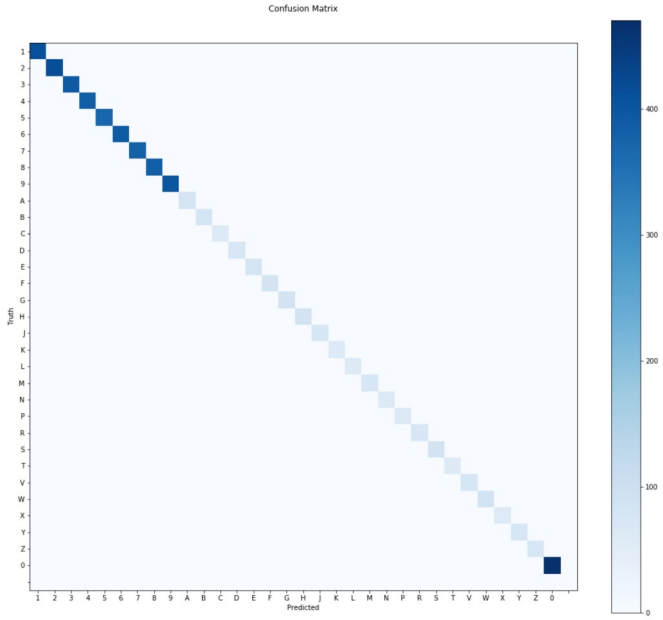

Figure 70: Confusion Matrix for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 4 pixels

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.0	470.0	0.0	0.0
1	1	1.000000	1.0	408.0	0.0	0.0
2	2	1.000000	1.0	416.0	0.0	0.0
3	3	1.000000	1.0	394.0	0.0	0.0
4	4	1.000000	1.0	384.0	0.0	0.0
5	5	1.000000	1.0	371.0	0.0	0.0
6	6	1.000000	1.0	391.0	0.0	0.0
7	7	1.000000	1.0	380.0	0.0	0.0
8	8	1.000000	1.0	384.0	0.0	0.0
9	9	1.000000	1.0	402.0	0.0	0.0
10	A	1.000000	1.0	80.0	0.0	0.0
11	B	1.000000	1.0	78.0	0.0	0.0
12	C	1.000000	1.0	62.0	0.0	0.0
13	D	1.000000	1.0	71.0	0.0	0.0
14	E	0.987500	1.0	79.0	1.0	0.0
15	F	1.000000	1.0	81.0	0.0	0.0
16	G	1.000000	1.0	84.0	0.0	0.0
17	H	1.000000	1.0	83.0	0.0	0.0
18	J	0.986486	1.0	73.0	1.0	0.0
19	K	1.000000	1.0	64.0	0.0	0.0
20	L	1.000000	1.0	61.0	0.0	0.0
21	M	1.000000	1.0	73.0	0.0	0.0
22	N	1.000000	1.0	64.0	0.0	0.0
23	P	1.000000	1.0	63.0	0.0	0.0
24	R	1.000000	1.0	72.0	0.0	0.0
25	S	0.988235	1.0	84.0	1.0	0.0
26	T	1.000000	1.0	58.0	0.0	0.0
27	V	1.000000	1.0	76.0	0.0	0.0
28	W	1.000000	1.0	88.0	0.0	0.0
29	X	1.000000	1.0	58.0	0.0	0.0
30	Y	1.000000	1.0	76.0	0.0	0.0
31	Z	1.000000	1.0	72.0	0.0	0.0

Figure 71: Tabular results for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 4 pixels

B.3.3 Kernel Size 10 pixels

Figure 72 and Figure 73 show results for a Faster R-CNN ResNet 50 network trained and tested on a dataset with a motion blur simulated with a kernel size of 10 pixels. For this dataset recall was 100% but there was a false positive for “E”.

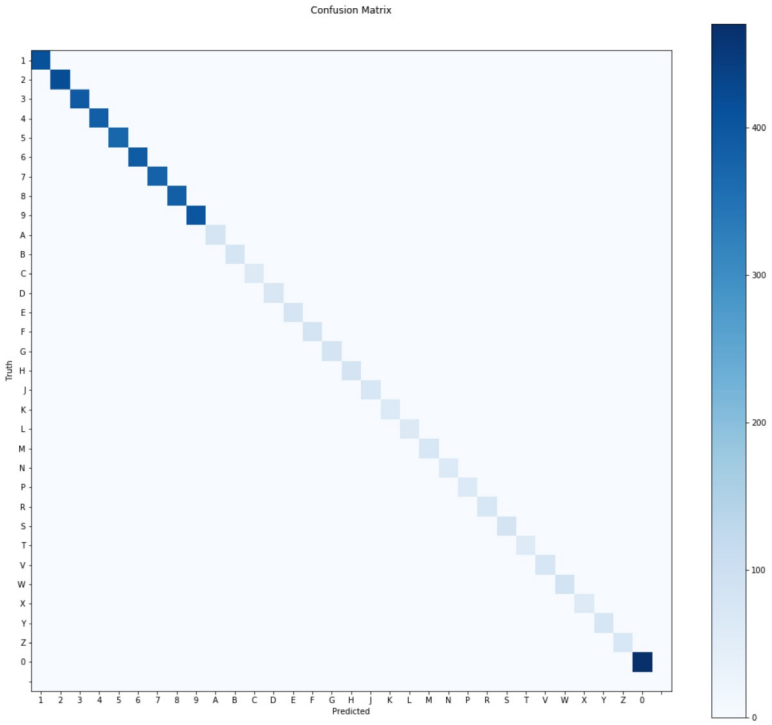

Figure 72: Confusion Matrix for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 10 pixels

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.0000	1.0	470.0	0.0	0.0
1	1	1.0000	1.0	408.0	0.0	0.0
2	2	1.0000	1.0	416.0	0.0	0.0
3	3	1.0000	1.0	394.0	0.0	0.0
4	4	1.0000	1.0	384.0	0.0	0.0
5	5	1.0000	1.0	371.0	0.0	0.0
6	6	1.0000	1.0	391.0	0.0	0.0
7	7	1.0000	1.0	380.0	0.0	0.0
8	8	1.0000	1.0	384.0	0.0	0.0
9	9	1.0000	1.0	402.0	0.0	0.0
10	A	1.0000	1.0	80.0	0.0	0.0
11	B	1.0000	1.0	78.0	0.0	0.0
12	C	1.0000	1.0	62.0	0.0	0.0
13	D	1.0000	1.0	71.0	0.0	0.0
14	E	0.9875	1.0	79.0	1.0	0.0
15	F	1.0000	1.0	81.0	0.0	0.0
16	G	1.0000	1.0	84.0	0.0	0.0
17	H	1.0000	1.0	83.0	0.0	0.0
18	J	1.0000	1.0	73.0	0.0	0.0
19	K	1.0000	1.0	64.0	0.0	0.0
20	L	1.0000	1.0	61.0	0.0	0.0
21	M	1.0000	1.0	73.0	0.0	0.0
22	N	1.0000	1.0	64.0	0.0	0.0
23	P	1.0000	1.0	63.0	0.0	0.0
24	R	1.0000	1.0	72.0	0.0	0.0
25	S	1.0000	1.0	84.0	0.0	0.0
26	T	1.0000	1.0	58.0	0.0	0.0
27	V	1.0000	1.0	76.0	0.0	0.0
28	W	1.0000	1.0	88.0	0.0	0.0
29	X	1.0000	1.0	58.0	0.0	0.0
30	Y	1.0000	1.0	76.0	0.0	0.0
31	Z	1.0000	1.0	72.0	0.0	0.0

Figure 73: Tabular results for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 10 pixels

B.3.4 Kernel Size 15 pixels

Figure 74 and Figure 75 show results for a Faster R-CNN ResNet 50 network trained and tested on a dataset with a motion blur simulated with a kernel size of 15 pixels. For this dataset recall was 100% but there were false positives for “E” and “J”.

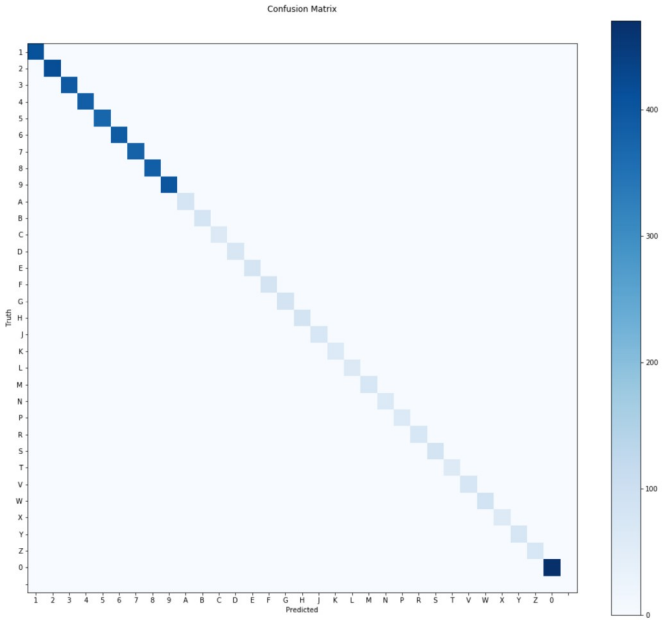

Figure 74: Confusion Matrix for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 15 pixels

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.0	470.0	0.0	0.0
1	1	1.000000	1.0	408.0	0.0	0.0
2	2	1.000000	1.0	416.0	0.0	0.0
3	3	1.000000	1.0	394.0	0.0	0.0
4	4	1.000000	1.0	384.0	0.0	0.0
5	5	1.000000	1.0	371.0	0.0	0.0
6	6	1.000000	1.0	391.0	0.0	0.0
7	7	1.000000	1.0	380.0	0.0	0.0
8	8	1.000000	1.0	384.0	0.0	0.0
9	9	1.000000	1.0	402.0	0.0	0.0
10	A	1.000000	1.0	80.0	0.0	0.0
11	B	1.000000	1.0	78.0	0.0	0.0
12	C	1.000000	1.0	62.0	0.0	0.0
13	D	1.000000	1.0	71.0	0.0	0.0
14	E	0.987500	1.0	79.0	1.0	0.0
15	F	1.000000	1.0	81.0	0.0	0.0
16	G	1.000000	1.0	84.0	0.0	0.0
17	H	1.000000	1.0	83.0	0.0	0.0
18	J	0.986486	1.0	73.0	1.0	0.0
19	K	1.000000	1.0	64.0	0.0	0.0
20	L	1.000000	1.0	61.0	0.0	0.0
21	M	1.000000	1.0	73.0	0.0	0.0
22	N	1.000000	1.0	64.0	0.0	0.0
23	P	1.000000	1.0	63.0	0.0	0.0
24	R	1.000000	1.0	72.0	0.0	0.0
25	S	1.000000	1.0	84.0	0.0	0.0
26	T	1.000000	1.0	58.0	0.0	0.0
27	V	1.000000	1.0	76.0	0.0	0.0
28	W	1.000000	1.0	88.0	0.0	0.0
29	X	1.000000	1.0	58.0	0.0	0.0
30	Y	1.000000	1.0	76.0	0.0	0.0
31	Z	1.000000	1.0	72.0	0.0	0.0

Figure 75: Tabular results for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 15 pixels

B.3.5 Kernel Size 20 pixels

Figure 76 and Figure 77 show results for a Faster R-CNN ResNet 50 network trained and tested on a dataset with a motion blur simulated with a kernel size of 20 pixels. For this dataset there was a false negative for “G” and several false positives for “C”, “S”, “V”, and “X”.

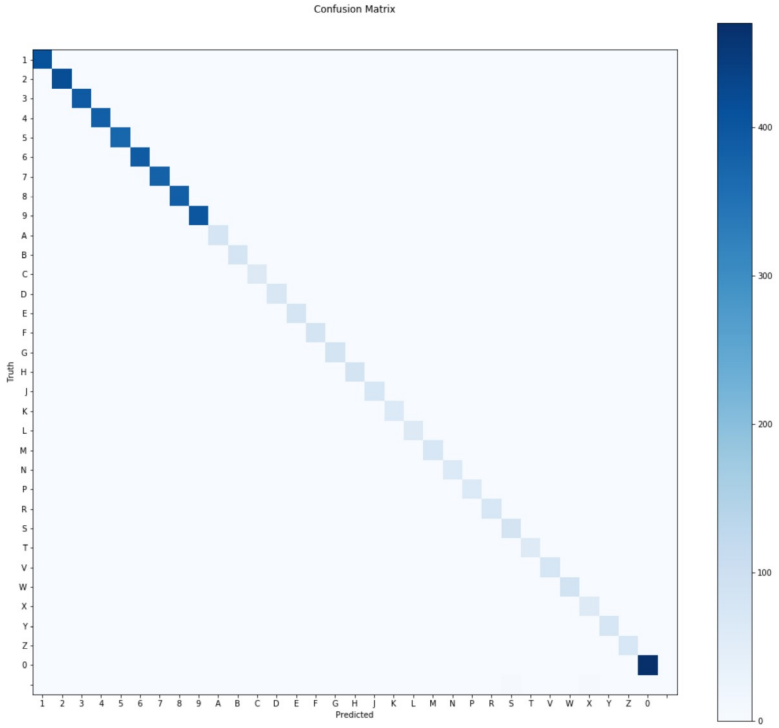

Figure 76: Confusion Matrix for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 20 pixels

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.000000	470.0	0.0	0.0
1	1	1.000000	1.000000	408.0	0.0	0.0
2	2	1.000000	1.000000	416.0	0.0	0.0
3	3	1.000000	1.000000	394.0	0.0	0.0
4	4	1.000000	1.000000	384.0	0.0	0.0
5	5	1.000000	1.000000	371.0	0.0	0.0
6	6	1.000000	1.000000	391.0	0.0	0.0
7	7	1.000000	1.000000	380.0	0.0	0.0
8	8	1.000000	1.000000	384.0	0.0	0.0
9	9	1.000000	1.000000	402.0	0.0	0.0
10	A	1.000000	1.000000	80.0	0.0	0.0
11	B	1.000000	1.000000	78.0	0.0	0.0
12	C	0.984127	1.000000	62.0	1.0	0.0
13	D	1.000000	1.000000	71.0	0.0	0.0
14	E	1.000000	1.000000	79.0	0.0	0.0
15	F	1.000000	1.000000	81.0	0.0	0.0
16	G	1.000000	0.988095	83.0	0.0	1.0
17	H	1.000000	1.000000	83.0	0.0	0.0
18	J	1.000000	1.000000	73.0	0.0	0.0
19	K	1.000000	1.000000	64.0	0.0	0.0
20	L	1.000000	1.000000	61.0	0.0	0.0
21	M	1.000000	1.000000	73.0	0.0	0.0
22	N	1.000000	1.000000	64.0	0.0	0.0
23	P	1.000000	1.000000	63.0	0.0	0.0
24	R	1.000000	1.000000	72.0	0.0	0.0
25	S	0.943820	1.000000	84.0	5.0	0.0
26	T	1.000000	1.000000	58.0	0.0	0.0
27	V	0.987013	1.000000	76.0	1.0	0.0
28	W	1.000000	1.000000	88.0	0.0	0.0
29	X	0.935484	1.000000	58.0	4.0	0.0
30	Y	1.000000	1.000000	76.0	0.0	0.0
31	Z	1.000000	1.000000	72.0	0.0	0.0

Figure 77: Tabular results for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 20 pixels

B.3.6 Kernel Size 25 pixels

Figure 78 and Figure 79 show results for a Faster R-CNN ResNet 50 network trained and tested on a dataset with a motion blur simulated with a kernel size of 255 pixels. At this size of the motion blur, the performance starts degrading significantly with a large number of false negatives as well as false positives.

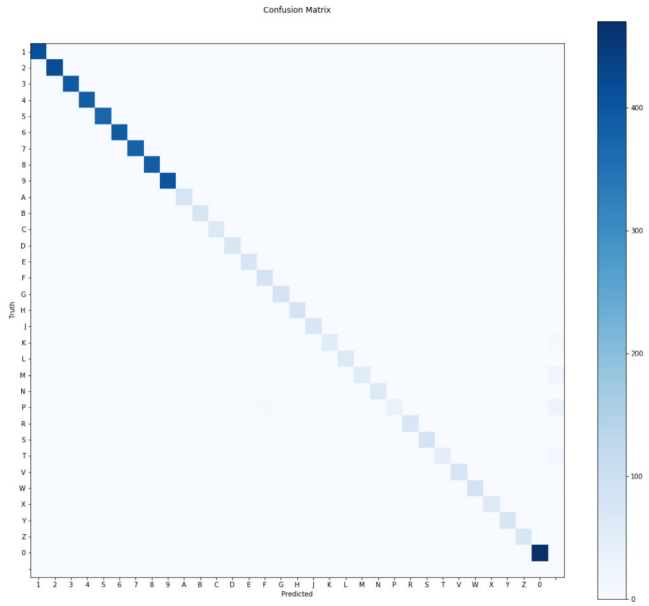

Figure 78: Confusion Matrix for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 25 pixels

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.000000	470.0	0.0	0.0
1	1	1.000000	1.000000	408.0	0.0	0.0
2	2	1.000000	1.000000	416.0	0.0	0.0
3	3	1.000000	1.000000	394.0	0.0	0.0
4	4	1.000000	1.000000	384.0	0.0	0.0
5	5	1.000000	1.000000	371.0	0.0	0.0
6	6	1.000000	1.000000	391.0	0.0	0.0
7	7	1.000000	1.000000	380.0	0.0	0.0
8	8	1.000000	1.000000	384.0	0.0	0.0
9	9	1.000000	1.000000	402.0	0.0	0.0
10	A	1.000000	1.000000	80.0	0.0	0.0
11	B	1.000000	0.974359	76.0	0.0	2.0
12	C	1.000000	1.000000	62.0	0.0	0.0
13	D	0.972603	1.000000	71.0	2.0	0.0
14	E	1.000000	0.974684	77.0	0.0	2.0
15	F	0.941860	1.000000	81.0	5.0	0.0
16	G	1.000000	0.976190	82.0	0.0	2.0
17	H	1.000000	0.987952	82.0	0.0	1.0
18	J	1.000000	1.000000	73.0	0.0	0.0
19	K	1.000000	0.859375	55.0	0.0	9.0
20	L	1.000000	1.000000	61.0	0.0	0.0
21	M	1.000000	0.739726	54.0	0.0	19.0
22	N	0.984615	1.000000	64.0	1.0	0.0
23	P	1.000000	0.492063	31.0	0.0	32.0
24	R	1.000000	1.000000	72.0	0.0	0.0
25	S	1.000000	1.000000	84.0	0.0	0.0
26	T	1.000000	0.775862	45.0	0.0	13.0
27	V	1.000000	1.000000	76.0	0.0	0.0
28	W	1.000000	0.977273	86.0	0.0	2.0
29	X	0.950820	1.000000	58.0	3.0	0.0
30	Y	1.000000	0.986842	75.0	0.0	1.0
31	Z	1.000000	1.000000	72.0	0.0	0.0

Figure 79: Tabular results for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 25 pixels

B.4 Affine Transforms

Affine transforms were applied as data augmentation, scaling the dataset from 50% to 150% and rotating between +/- 5 degrees.

B.4.1 50% Scale

Figure 80 and Figure 81 show results for a Faster R-CNN ResNet 50 network trained with affine transformed dataset and tested against a dataset at 50% scale. Out of all data augmentations, this has the worst performance. This is somewhat surprising since when the images were made half size the network had no issue. The difference between the 50% scale and the half-size images is the black border surrounding the images in the 50% scaled images. Although various scaled images fit the problem space as a data augmentation technique this black border that is used when the Imgaug library generates the augmented images doesn't, and since the features of interest in the plate image are the same color as the border it is possible that this is causing part of the issue.

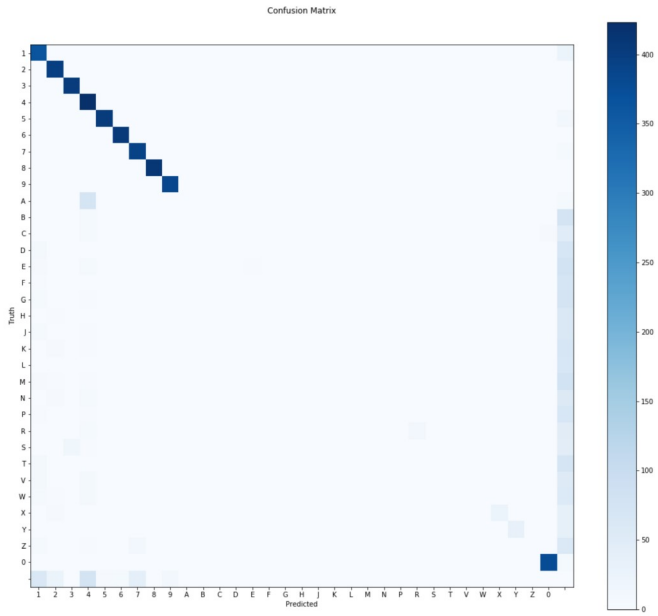

Figure 80: Confusion Matrix for Faster R-CNN ResNet 50 on images at 50% scale

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	0.986877	0.986877	376.0	5.0	5.0
1	1	0.749482	0.942708	362.0	121.0	22.0
2	2	0.896396	1.000000	398.0	46.0	0.0
3	3	0.961722	1.000000	402.0	16.0	0.0
4	4	0.671429	1.000000	423.0	207.0	0.0
5	5	0.990148	0.975728	402.0	4.0	10.0
6	6	0.985401	1.000000	405.0	6.0	0.0
7	7	0.888383	0.984848	390.0	49.0	6.0
8	8	1.000000	1.000000	410.0	0.0	0.0
9	9	0.967254	1.000000	384.0	13.0	0.0
10	A	NaN	0.000000	0.0	0.0	79.0
11	B	NaN	0.000000	0.0	0.0	79.0
12	C	NaN	0.000000	0.0	0.0	59.0
13	D	NaN	0.000000	0.0	0.0	76.0
14	E	1.000000	0.022222	2.0	0.0	88.0
15	F	NaN	0.000000	0.0	0.0	75.0
16	G	NaN	0.000000	0.0	0.0	85.0
17	H	NaN	0.000000	0.0	0.0	65.0
18	J	1.000000	0.014706	1.0	0.0	67.0
19	K	NaN	0.000000	0.0	0.0	75.0
20	L	NaN	0.000000	0.0	0.0	66.0
21	M	NaN	0.000000	0.0	0.0	88.0
22	N	NaN	0.000000	0.0	0.0	66.0
23	P	NaN	0.000000	0.0	0.0	70.0
24	R	1.000000	0.200000	13.0	0.0	52.0
25	S	NaN	0.000000	0.0	0.0	63.0
26	T	NaN	0.000000	0.0	0.0	80.0
27	V	1.000000	0.014706	1.0	0.0	67.0
28	W	NaN	0.000000	0.0	0.0	71.0
29	X	1.000000	0.384615	25.0	0.0	40.0
30	Y	1.000000	0.469697	31.0	0.0	35.0
31	Z	NaN	0.000000	0.0	0.0	79.0

Figure 81: Tabular results for Faster R-CNN ResNet 50 on images at 50% scale

B.4.2 150% Scale

Figure 82 and Figure 82 show results for a Faster R-CNN ResNet 50 network trained with affine transformed dataset and tested against a dataset at 150% scale. While this has better performance than the 50% scale both recall and precision were quite poor.

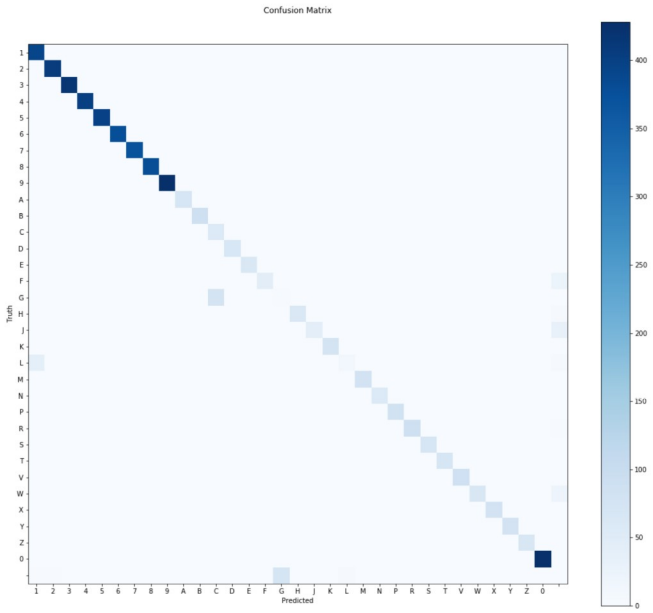

Figure 82: Confusion Matrix for Faster R-CNN ResNet 50 on images at 150% scale

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.000000	427.0	0.0	0.0
1	1	0.900693	1.000000	390.0	43.0	0.0
2	2	0.992683	1.000000	407.0	3.0	0.0
3	3	1.000000	1.000000	417.0	0.0	0.0
4	4	1.000000	1.000000	401.0	0.0	0.0
5	5	1.000000	1.000000	399.0	0.0	0.0
6	6	1.000000	1.000000	377.0	0.0	0.0
7	7	1.000000	1.000000	370.0	0.0	0.0
8	8	1.000000	1.000000	379.0	0.0	0.0
9	9	1.000000	1.000000	428.0	0.0	0.0
10	A	1.000000	1.000000	70.0	0.0	0.0
11	B	1.000000	1.000000	90.0	0.0	0.0
12	C	0.431818	1.000000	57.0	75.0	0.0
13	D	1.000000	1.000000	66.0	0.0	0.0
14	E	0.983871	1.000000	61.0	1.0	0.0
15	F	1.000000	0.671642	45.0	0.0	22.0
16	G	0.040541	0.038462	3.0	71.0	75.0
17	H	1.000000	0.939394	62.0	0.0	4.0
18	J	1.000000	0.569444	41.0	0.0	31.0
19	K	1.000000	1.000000	76.0	0.0	0.0
20	L	0.705882	0.214286	12.0	5.0	44.0
21	M	1.000000	1.000000	78.0	0.0	0.0
22	N	1.000000	0.983051	58.0	0.0	1.0
23	P	1.000000	1.000000	82.0	0.0	0.0
24	R	1.000000	0.967742	90.0	0.0	3.0
25	S	1.000000	1.000000	67.0	0.0	0.0
26	T	1.000000	1.000000	70.0	0.0	0.0
27	V	1.000000	1.000000	86.0	0.0	0.0
28	W	1.000000	0.755814	65.0	0.0	21.0
29	X	1.000000	1.000000	77.0	0.0	0.0
30	Y	1.000000	1.000000	77.0	0.0	0.0
31	Z	1.000000	1.000000	64.0	0.0	0.0

Figure 83: Tabular results for Faster R-CNN ResNet 50 on images at 150% scale

B.4.3 +5 Degree Rotation

Figure 84 and Figure 85 show results for a Faster R-CNN ResNet 50 network trained with affine transformed dataset and tested against a dataset at +5-degree rotation. Recall and precision were above 80%, which while as good as the other data augmentation techniques is better than the other affine transforms. Note that the ability to train and test on the rotated image was limited to this range by the ground truth bounding boxes rotation function which does not rotate well.

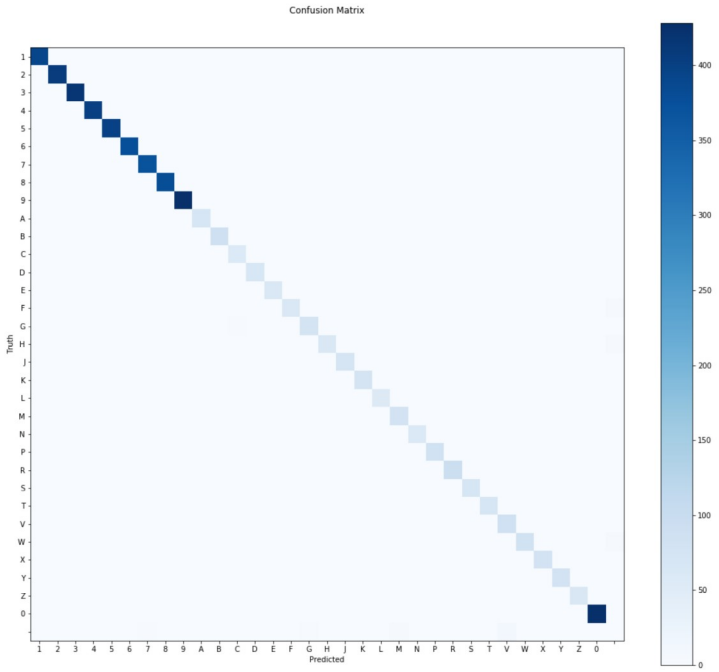

Figure 84: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated +5 degrees

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	0.997664	1.000000	427.0	1.0	0.0
1	1	1.000000	1.000000	390.0	0.0	0.0
2	2	1.000000	1.000000	407.0	0.0	0.0
3	3	1.000000	1.000000	417.0	0.0	0.0
4	4	1.000000	1.000000	401.0	0.0	0.0
5	5	1.000000	1.000000	399.0	0.0	0.0
6	6	1.000000	1.000000	377.0	0.0	0.0
7	7	0.994624	1.000000	370.0	2.0	0.0
8	8	1.000000	1.000000	379.0	0.0	0.0
9	9	1.000000	1.000000	428.0	0.0	0.0
10	A	1.000000	1.000000	70.0	0.0	0.0
11	B	1.000000	1.000000	90.0	0.0	0.0
12	C	0.950000	1.000000	57.0	3.0	0.0
13	D	1.000000	1.000000	66.0	0.0	0.0
14	E	1.000000	1.000000	61.0	0.0	0.0
15	F	1.000000	0.940299	63.0	0.0	4.0
16	G	0.961538	0.961538	75.0	3.0	3.0
17	H	1.000000	0.924242	61.0	0.0	5.0
18	J	1.000000	1.000000	72.0	0.0	0.0
19	K	1.000000	1.000000	76.0	0.0	0.0
20	L	1.000000	1.000000	56.0	0.0	0.0
21	M	0.939759	1.000000	78.0	5.0	0.0
22	N	1.000000	1.000000	59.0	0.0	0.0
23	P	1.000000	1.000000	82.0	0.0	0.0
24	R	1.000000	1.000000	93.0	0.0	0.0
25	S	0.985294	1.000000	67.0	1.0	0.0
26	T	1.000000	1.000000	70.0	0.0	0.0
27	V	0.877551	1.000000	86.0	12.0	0.0
28	W	1.000000	0.941860	81.0	0.0	5.0
29	X	1.000000	1.000000	77.0	0.0	0.0
30	Y	1.000000	1.000000	77.0	0.0	0.0
31	Z	1.000000	1.000000	64.0	0.0	0.0

Figure 85: Tabular results for Faster R-CNN ResNet 50 on images rotated +5 degrees

B.4.4 -5 Degree Rotation

Figure 86 and Figure 87 show results for a Faster R-CNN ResNet 50 network trained with affine transformed dataset and tested against a dataset at -5-degree rotation. Recall and precision were above 50%, which while as good as the other data augmentation techniques is better than the other affine transforms. Note that the ability to train and test on the rotated image was limited to this range by the ground truth bounding boxes rotation function which does not rotate well.

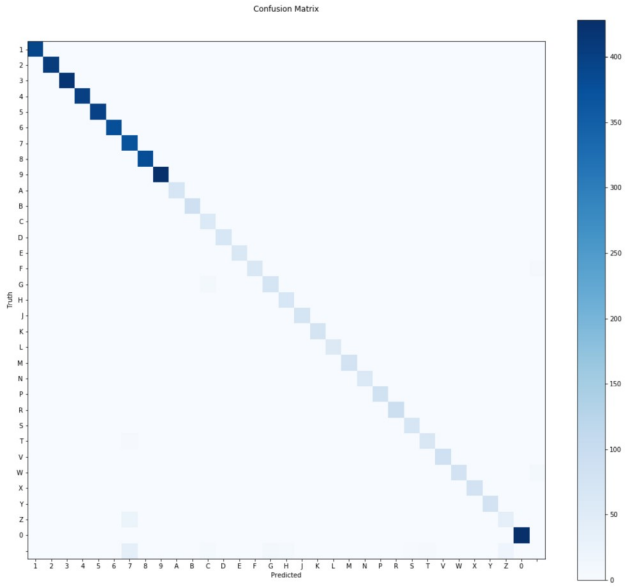

Figure 86: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated -5 degrees

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.000000	427.0	0.0	0.0
1	1	1.000000	1.000000	390.0	0.0	0.0
2	2	1.000000	1.000000	407.0	0.0	0.0
3	3	1.000000	1.000000	417.0	0.0	0.0
4	4	1.000000	1.000000	401.0	0.0	0.0
5	5	1.000000	1.000000	399.0	0.0	0.0
6	6	1.000000	1.000000	377.0	0.0	0.0
7	7	0.842825	1.000000	370.0	69.0	0.0
8	8	1.000000	1.000000	379.0	0.0	0.0
9	9	1.000000	1.000000	428.0	0.0	0.0
10	A	1.000000	1.000000	70.0	0.0	0.0
11	B	1.000000	1.000000	90.0	0.0	0.0
12	C	0.814286	1.000000	57.0	13.0	0.0
13	D	1.000000	1.000000	66.0	0.0	0.0
14	E	1.000000	1.000000	61.0	0.0	0.0
15	F	1.000000	0.925373	62.0	0.0	5.0
16	G	0.910256	0.910256	71.0	7.0	7.0
17	H	0.916667	1.000000	66.0	6.0	0.0
18	J	1.000000	1.000000	72.0	0.0	0.0
19	K	1.000000	1.000000	76.0	0.0	0.0
20	L	1.000000	1.000000	56.0	0.0	0.0
21	M	1.000000	1.000000	78.0	0.0	0.0
22	N	1.000000	1.000000	59.0	0.0	0.0
23	P	1.000000	1.000000	82.0	0.0	0.0
24	R	1.000000	1.000000	93.0	0.0	0.0
25	S	0.971014	1.000000	67.0	2.0	0.0
26	T	0.928571	0.928571	65.0	5.0	5.0
27	V	1.000000	1.000000	86.0	0.0	0.0
28	W	1.000000	0.895349	77.0	0.0	9.0
29	X	0.987179	1.000000	77.0	1.0	0.0
30	Y	1.000000	1.000000	77.0	0.0	0.0
31	Z	0.655172	0.593750	38.0	20.0	26.0

Figure 87: Tabular results for Faster R-CNN ResNet 50 on images rotated -5 degrees

B.5 Gaussian Noise

Gaussian Noise data augmentation was applied to training data via the configuration file, and to test data with varying severity from the Imuaug library. Performance remained relatively high across all tested values.

B.5.1 Ideal

Figure 88 and Figure 89 show results for an ideal dataset with no Gaussian noise applied. Recall and precision for this dataset are both 100%.

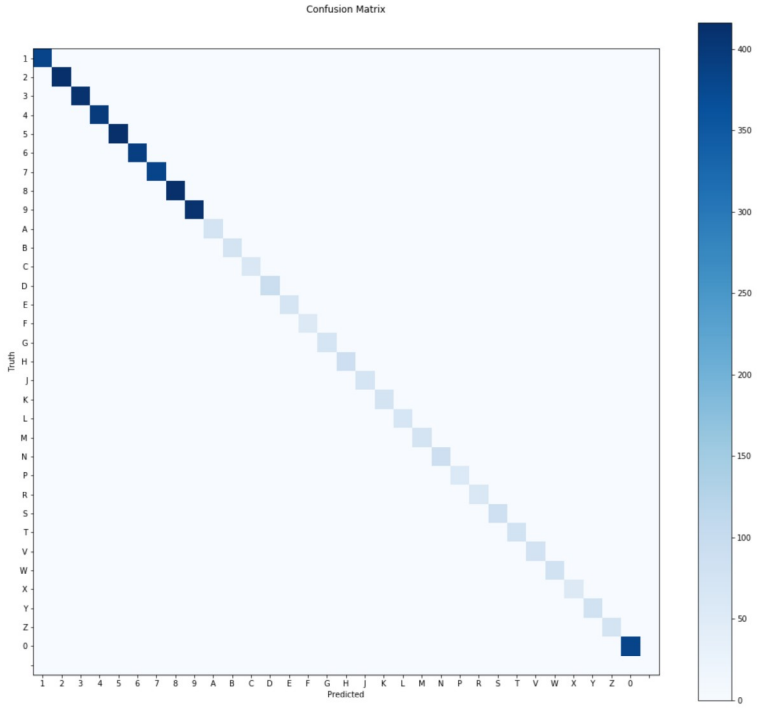

Figure 88: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with no Gaussian noise

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.0	1.0	383.0	0.0	0.0
1	1	1.0	1.0	382.0	0.0	0.0
2	2	1.0	1.0	413.0	0.0	0.0
3	3	1.0	1.0	411.0	0.0	0.0
4	4	1.0	1.0	398.0	0.0	0.0
5	5	1.0	1.0	414.0	0.0	0.0
6	6	1.0	1.0	391.0	0.0	0.0
7	7	1.0	1.0	382.0	0.0	0.0
8	8	1.0	1.0	416.0	0.0	0.0
9	9	1.0	1.0	410.0	0.0	0.0
10	A	1.0	1.0	73.0	0.0	0.0
11	B	1.0	1.0	74.0	0.0	0.0
12	C	1.0	1.0	60.0	0.0	0.0
13	D	1.0	1.0	93.0	0.0	0.0
14	E	1.0	1.0	70.0	0.0	0.0
15	F	1.0	1.0	55.0	0.0	0.0
16	G	1.0	1.0	69.0	0.0	0.0
17	H	1.0	1.0	89.0	0.0	0.0
18	J	1.0	1.0	69.0	0.0	0.0
19	K	1.0	1.0	74.0	0.0	0.0
20	L	1.0	1.0	67.0	0.0	0.0
21	M	1.0	1.0	74.0	0.0	0.0
22	N	1.0	1.0	89.0	0.0	0.0
23	P	1.0	1.0	56.0	0.0	0.0
24	R	1.0	1.0	61.0	0.0	0.0
25	S	1.0	1.0	86.0	0.0	0.0
26	T	1.0	1.0	77.0	0.0	0.0
27	V	1.0	1.0	76.0	0.0	0.0
28	W	1.0	1.0	81.0	0.0	0.0
29	X	1.0	1.0	53.0	0.0	0.0
30	Y	1.0	1.0	80.0	0.0	0.0
31	Z	1.0	1.0	74.0	0.0	0.0

Figure 89: Tabular results for Faster R-CNN ResNet 50 on images rotated with no Gaussian noise

B.5.2 Severity 1

Figure 90 and Figure 91 show results for an ideal dataset with Gaussian noise of severity 1 applied. Recall and precision for this dataset are both 100%.

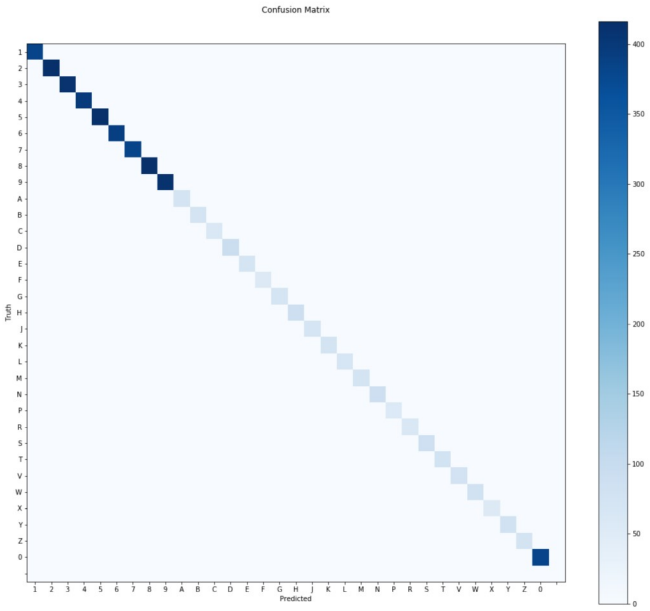

Figure 90: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 1

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.0	1.0	383.0	0.0	0.0
1	1	1.0	1.0	382.0	0.0	0.0
2	2	1.0	1.0	413.0	0.0	0.0
3	3	1.0	1.0	411.0	0.0	0.0
4	4	1.0	1.0	398.0	0.0	0.0
5	5	1.0	1.0	414.0	0.0	0.0
6	6	1.0	1.0	391.0	0.0	0.0
7	7	1.0	1.0	382.0	0.0	0.0
8	8	1.0	1.0	416.0	0.0	0.0
9	9	1.0	1.0	410.0	0.0	0.0
10	A	1.0	1.0	73.0	0.0	0.0
11	B	1.0	1.0	74.0	0.0	0.0
12	C	1.0	1.0	60.0	0.0	0.0
13	D	1.0	1.0	93.0	0.0	0.0
14	E	1.0	1.0	70.0	0.0	0.0
15	F	1.0	1.0	55.0	0.0	0.0
16	G	1.0	1.0	69.0	0.0	0.0
17	H	1.0	1.0	89.0	0.0	0.0
18	J	1.0	1.0	69.0	0.0	0.0
19	K	1.0	1.0	74.0	0.0	0.0
20	L	1.0	1.0	67.0	0.0	0.0
21	M	1.0	1.0	74.0	0.0	0.0
22	N	1.0	1.0	89.0	0.0	0.0
23	P	1.0	1.0	56.0	0.0	0.0
24	R	1.0	1.0	61.0	0.0	0.0
25	S	1.0	1.0	86.0	0.0	0.0
26	T	1.0	1.0	77.0	0.0	0.0
27	V	1.0	1.0	76.0	0.0	0.0
28	W	1.0	1.0	81.0	0.0	0.0
29	X	1.0	1.0	53.0	0.0	0.0
30	Y	1.0	1.0	80.0	0.0	0.0
31	Z	1.0	1.0	74.0	0.0	0.0

Figure 91: Tabular results for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 1

B.5.3 Severity 2

Figure 92 and Figure 93 show results for an ideal dataset with Gaussian noise of severity 2 applied. Recall and precision for this dataset are both 100%.

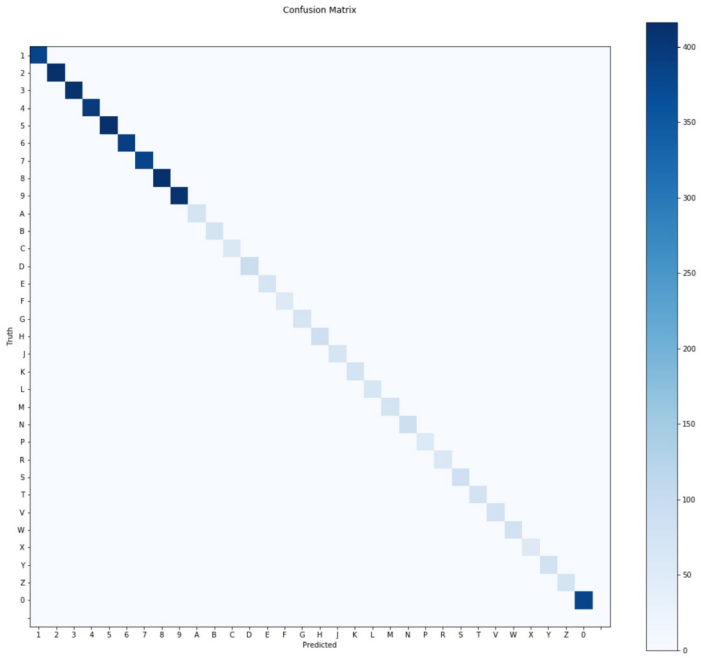

Figure 92: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 2

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.0	1.0	383.0	0.0	0.0
1	1	1.0	1.0	382.0	0.0	0.0
2	2	1.0	1.0	413.0	0.0	0.0
3	3	1.0	1.0	411.0	0.0	0.0
4	4	1.0	1.0	398.0	0.0	0.0
5	5	1.0	1.0	414.0	0.0	0.0
6	6	1.0	1.0	391.0	0.0	0.0
7	7	1.0	1.0	382.0	0.0	0.0
8	8	1.0	1.0	416.0	0.0	0.0
9	9	1.0	1.0	410.0	0.0	0.0
10	A	1.0	1.0	73.0	0.0	0.0
11	B	1.0	1.0	74.0	0.0	0.0
12	C	1.0	1.0	60.0	0.0	0.0
13	D	1.0	1.0	93.0	0.0	0.0
14	E	1.0	1.0	70.0	0.0	0.0
15	F	1.0	1.0	55.0	0.0	0.0
16	G	1.0	1.0	69.0	0.0	0.0
17	H	1.0	1.0	89.0	0.0	0.0
18	J	1.0	1.0	69.0	0.0	0.0
19	K	1.0	1.0	74.0	0.0	0.0
20	L	1.0	1.0	67.0	0.0	0.0
21	M	1.0	1.0	74.0	0.0	0.0
22	N	1.0	1.0	89.0	0.0	0.0
23	P	1.0	1.0	56.0	0.0	0.0
24	R	1.0	1.0	61.0	0.0	0.0
25	S	1.0	1.0	86.0	0.0	0.0
26	T	1.0	1.0	77.0	0.0	0.0
27	V	1.0	1.0	76.0	0.0	0.0
28	W	1.0	1.0	81.0	0.0	0.0
29	X	1.0	1.0	53.0	0.0	0.0
30	Y	1.0	1.0	80.0	0.0	0.0
31	Z	1.0	1.0	74.0	0.0	0.0

Figure 93: Tabular results for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 2

B.5.4 Severity 3

Figure 94 and Figure 95 show results for an ideal dataset with Gaussian noise of severity 3 applied. Recall and precision for this dataset are both 100%.

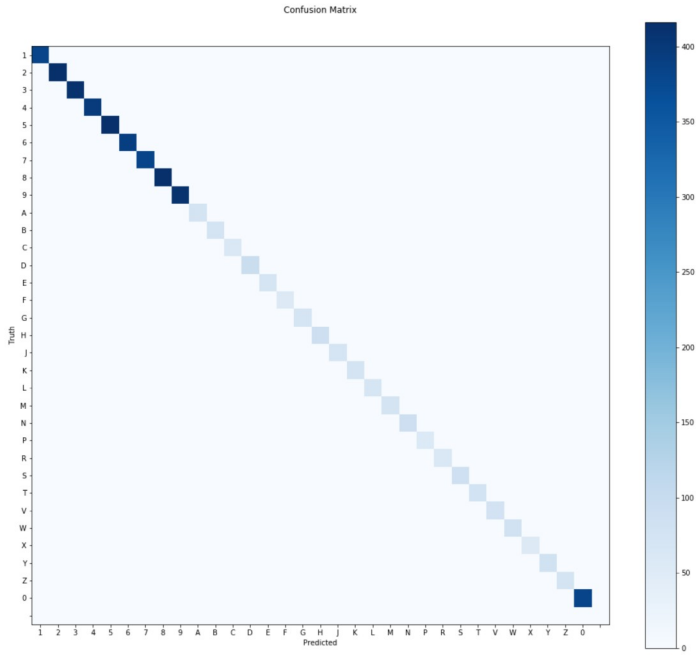

Figure 94: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 3

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.0	1.0	383.0	0.0	0.0
1	1	1.0	1.0	382.0	0.0	0.0
2	2	1.0	1.0	413.0	0.0	0.0
3	3	1.0	1.0	411.0	0.0	0.0
4	4	1.0	1.0	398.0	0.0	0.0
5	5	1.0	1.0	414.0	0.0	0.0
6	6	1.0	1.0	391.0	0.0	0.0
7	7	1.0	1.0	382.0	0.0	0.0
8	8	1.0	1.0	416.0	0.0	0.0
9	9	1.0	1.0	410.0	0.0	0.0
10	A	1.0	1.0	73.0	0.0	0.0
11	B	1.0	1.0	74.0	0.0	0.0
12	C	1.0	1.0	60.0	0.0	0.0
13	D	1.0	1.0	93.0	0.0	0.0
14	E	1.0	1.0	70.0	0.0	0.0
15	F	1.0	1.0	55.0	0.0	0.0
16	G	1.0	1.0	69.0	0.0	0.0
17	H	1.0	1.0	89.0	0.0	0.0
18	J	1.0	1.0	69.0	0.0	0.0
19	K	1.0	1.0	74.0	0.0	0.0
20	L	1.0	1.0	67.0	0.0	0.0
21	M	1.0	1.0	74.0	0.0	0.0
22	N	1.0	1.0	89.0	0.0	0.0
23	P	1.0	1.0	56.0	0.0	0.0
24	R	1.0	1.0	61.0	0.0	0.0
25	S	1.0	1.0	86.0	0.0	0.0
26	T	1.0	1.0	77.0	0.0	0.0
27	V	1.0	1.0	76.0	0.0	0.0
28	W	1.0	1.0	81.0	0.0	0.0
29	X	1.0	1.0	53.0	0.0	0.0
30	Y	1.0	1.0	80.0	0.0	0.0
31	Z	1.0	1.0	74.0	0.0	0.0

Figure 95: Tabular results for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 3

B.5.5 Severity 4

Figure 96 and Figure 97 show results for an ideal dataset with Gaussian noise of severity 4 applied. Recall was 100% but performance began to degrade with several false positives for “M”.

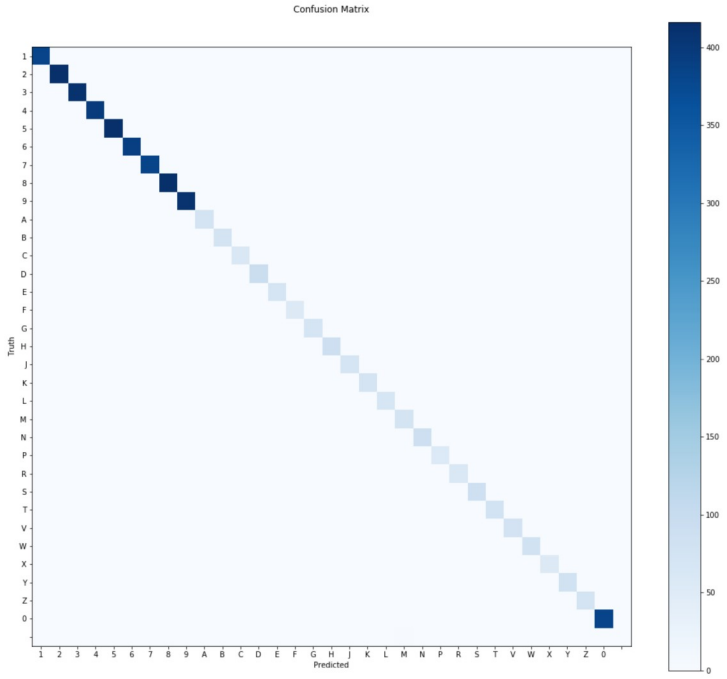

Figure 96: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 4

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.0	383.0	0.0	0.0
1	1	1.000000	1.0	382.0	0.0	0.0
2	2	1.000000	1.0	413.0	0.0	0.0
3	3	1.000000	1.0	411.0	0.0	0.0
4	4	1.000000	1.0	398.0	0.0	0.0
5	5	1.000000	1.0	414.0	0.0	0.0
6	6	1.000000	1.0	391.0	0.0	0.0
7	7	1.000000	1.0	382.0	0.0	0.0
8	8	1.000000	1.0	416.0	0.0	0.0
9	9	1.000000	1.0	410.0	0.0	0.0
10	A	1.000000	1.0	73.0	0.0	0.0
11	B	1.000000	1.0	74.0	0.0	0.0
12	C	1.000000	1.0	60.0	0.0	0.0
13	D	1.000000	1.0	93.0	0.0	0.0
14	E	1.000000	1.0	70.0	0.0	0.0
15	F	1.000000	1.0	55.0	0.0	0.0
16	G	1.000000	1.0	69.0	0.0	0.0
17	H	1.000000	1.0	89.0	0.0	0.0
18	J	1.000000	1.0	69.0	0.0	0.0
19	K	1.000000	1.0	74.0	0.0	0.0
20	L	1.000000	1.0	67.0	0.0	0.0
21	M	0.961039	1.0	74.0	3.0	0.0
22	N	1.000000	1.0	89.0	0.0	0.0
23	P	1.000000	1.0	56.0	0.0	0.0
24	R	1.000000	1.0	61.0	0.0	0.0
25	S	1.000000	1.0	86.0	0.0	0.0
26	T	1.000000	1.0	77.0	0.0	0.0
27	V	1.000000	1.0	76.0	0.0	0.0
28	W	1.000000	1.0	81.0	0.0	0.0
29	X	1.000000	1.0	53.0	0.0	0.0
30	Y	1.000000	1.0	80.0	0.0	0.0
31	Z	1.000000	1.0	74.0	0.0	0.0

Figure 97: Tabular results for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 4

B.5.6 Severity 5

Figure 98 and Figure 99 show results for an ideal dataset with Gaussian noise of severity 5 applied. Performance further degraded for this data augmentation technique with several false positives for “M” and false negatives for “N”.

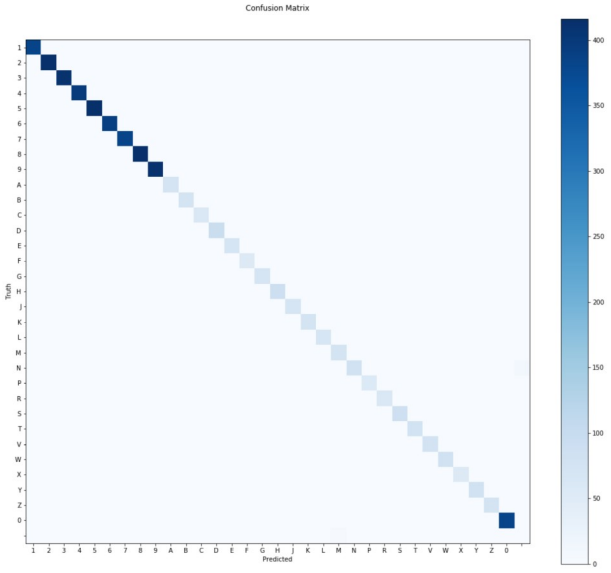

Figure 98: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 5

	category	precision_@0.5IOU	recall_@0.5IOU	TP	FP	FN
0	0	1.000000	1.000000	383.0	0.0	0.0
1	1	1.000000	1.000000	382.0	0.0	0.0
2	2	1.000000	1.000000	413.0	0.0	0.0
3	3	1.000000	1.000000	411.0	0.0	0.0
4	4	1.000000	1.000000	398.0	0.0	0.0
5	5	1.000000	1.000000	414.0	0.0	0.0
6	6	1.000000	1.000000	391.0	0.0	0.0
7	7	1.000000	1.000000	382.0	0.0	0.0
8	8	1.000000	1.000000	416.0	0.0	0.0
9	9	1.000000	1.000000	410.0	0.0	0.0
10	A	1.000000	1.000000	73.0	0.0	0.0
11	B	1.000000	1.000000	74.0	0.0	0.0
12	C	1.000000	1.000000	60.0	0.0	0.0
13	D	1.000000	1.000000	93.0	0.0	0.0
14	E	1.000000	1.000000	70.0	0.0	0.0
15	F	1.000000	1.000000	55.0	0.0	0.0
16	G	1.000000	1.000000	69.0	0.0	0.0
17	H	1.000000	1.000000	89.0	0.0	0.0
18	J	1.000000	1.000000	69.0	0.0	0.0
19	K	1.000000	1.000000	74.0	0.0	0.0
20	L	1.000000	1.000000	67.0	0.0	0.0
21	M	0.948718	1.000000	74.0	4.0	0.0
22	N	1.000000	0.876404	78.0	0.0	11.0
23	P	1.000000	1.000000	56.0	0.0	0.0
24	R	1.000000	1.000000	61.0	0.0	0.0
25	S	1.000000	1.000000	86.0	0.0	0.0
26	T	1.000000	1.000000	77.0	0.0	0.0
27	V	1.000000	1.000000	76.0	0.0	0.0
28	W	1.000000	1.000000	81.0	0.0	0.0
29	X	1.000000	1.000000	53.0	0.0	0.0
30	Y	1.000000	1.000000	80.0	0.0	0.0
31	Z	1.000000	1.000000	74.0	0.0	0.0

Figure 99: Tabular results for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 5

References

[1] E. Alpaydin, *Introduction to Machine Learning*, 3rd ed. Cambridge: MIT Press, The, 2014.

[2] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211-252, 2015.

[3] M. Sarfraz, M. J. Ahmed and S. A. Ghazi, "Saudi Arabian license plate recognition system," *2003 International Conference on Geometric Modeling and Graphics, 2003. Proceedings*, London, UK, 2003, pp. 36-41.

[4] A. LeNail, “NN-SVG: Publication-Ready Neural Network Architecture Schematics”, *The Journal of Open Source Software*, vol. 4, no. 33, p. 747, 2019. Available: 10.1007/s11263-015-0816-y.

[5] C. A. B. Mello and D. C. Costa, "A Complete System for Vehicle License Plate Recognition," *2009 16th International Conference on Systems, Signals and Image Processing*, Chalkida, 2009, pp. 1-4.

[6] Y. LeCun, Y. Bengio, G. Hinton, “Deep Learning,” *Nature.,* vol. 521, pp.436-*444*, May. 2015. Accessed on: June. 27, 2020. [Online]. Available doi:10.1038/nature14539

[7] R. Laroca *et al*., "A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector," *2018 International Joint Conference on Neural Networks (IJCNN)*, Rio de Janeiro, 2018, pp. 1-10.

[8] H. Li, P. Wang, M. You and C. Shen, “Reading Car License Plates Using Deep Convolutional Neural Networks and LSTMs,” *Image and Vision Computing*, vol. 72, pp. 14-23, Apr. 2018. Accessed on: July. 3, 2020. [Online].

[9] P. Svoboda, M. Hradiš, L. Maršík and P. Zemcík, "CNN for license plate motion deblurring," *2016 IEEE International Conference on Image Processing (ICIP)*, Phoenix, AZ, 2016, pp. 3832-3836.

[10] J. Huang *et al*., "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 3296-3297, doi: 10.1109/CVPR.2017.351.

[11]”Vehicle registration plates of Maryland", En.wikipedia.org , 2019. [Online]. Available: https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_Maryland. [Accessed: 09- Oct- 2019].

[12]"Maryland License Plates",Licenseplates.cc, 2019. [Online]. Available: https://www.licenseplates.cc/MD. [Accessed: 09- Oct- 2019].

[13] A. G. Howard, et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications.” *arXiv preprint arXiv:1704.04861* (2017).

[14] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision", 2015. Available: https://arxiv.org/pdf/1512.00567v3.pdf. [Accessed 2 June 2020].

[15] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. Available: 10.1109/cvpr.2016.90 [Accessed 2 June 2020].

[16] C. Szegedy, et al. “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.” *ArXiv* abs/1602.07261 (2017)
[17] W. Liu et al. (2016) SSD: Single Shot MultiBox Detector. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol 9905. Springer, Cham

[18] S. Agarwal, D. Tran, L. Torresani and H. Farid, "Deciphering Severely Degraded License Plates", *Electronic Imaging*, vol. 2017, no. 7, pp. 138-143, 2017. Available: 10.2352/issn.2470-1173.2017.7.mwsf-337.

[19] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 2017. Available: 10.1109/tpami.2016.2577031.

[20] "Camera Configuration — openalpr 2.8.101 documentation", Doc.openalpr.com, 2020. [Online]. Available: http://doc.openalpr.com/camera_placement.html. [Accessed: 19- Jul- 2020].

[21] D. M. F. Izidio, A. P. A. Ferreira and E. N. S. Barros, "An Embedded Automatic License Plate Recognition System Using Deep Learning," *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*, Salvador, Brazil, 2018, pp. 38-45.

[22] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computations 9(8):1735-1780*, 1997.

[23]N. Donges, "Recurrent neural networks 101: Understanding the basics of RNNs and LSTM", *Built In*, 2019. [Online]. Available: https://builtin.com/data-science/recurrent-neural-networks-and-lstm. [Accessed: 19- Jul- 2020].

[24] “Queuing system (SLURM)”, marcc.jhu.edu, 2020. [Online]. Available: https://www.marcc.jhu.edu/getting-started/running-jobs/

[25] J. Dai, Y. Li, K. He, and J.Sun, “R-FCN: Object Detection via Region-based Fully Convolutional Networks,” *In Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS’16).*

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, 86(11):2278-2324, November 1998.

[27] S. Kaul. “Region based convolutional neural networks for object detection and recognition in ADAS applications”. December 2017.

Vita

Christina Paolicelli is originally from Pound Ridge, NY. She received her B.S. inn Electrical Engineering from Rensselaer Polytechnic Institute, Troy, NY in 2017. She joined the Engineering for Professionals Master of Science in Electrical and Computer Engineering program at Johns Hopkins University in spring of 2018. She is broadly interested in Machine Learning and image processing.