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
October, 2020

Abstract

Commercially available Automatic License Plate Recognition (ALPR) systems have limited ability to provide character recognition on low-quality license plate images [20]. Improving this ability would be beneficial for tasks currently requiring human involvement to read low-quality license plate characters. Recent advances in Deep Learning networks have shown that, for object detection tasks, Deep Learning networks can achieve levels of performance equal to or better than those of a human [2,6]. The aim of this thesis is to introduce a foundational Deep Learning framework for character recognition performance analysis. The analysis is carried out on license plate images that have undergone various types and levels of image quality reduction.

This thesis leverages the TensorFlow Object Detection API to enable rapid development and testing of different Machine Learning networks and configurations. The framework allows for the creation of synthetically generated datasets on which image augmentation techniques can be applied. The various image augmentation techniques expand the dataset, and enable the network to be robust to image quality reductions. Networks were trained on the Maryland Advanced Computing Center’s GPU system. Per-character metrics of precision and recall are framework outputs used to evaluate trained networks.

Network performance was evaluated using the framework for several Machine Learning models. The Faster R-CNN ResNet 50 network was found to have the best performance for character recognition on synthetically generated license plate images. On an ideal dataset, with no image degradation applied, the lower threshold of image size, on which the Faster R-CNN

ResNet 50 network can reliably perform character recognition, was found to be 32 x 16 pixels. Finally, the network was trained and tested on image datasets with various data augmentation techniques applied. The data augmentation techniques evaluated in this thesis are: JPEG Compression, motion blur, affine transforms, and Gaussian noise. The results showed that, when trained on augmented synthetic data, the network was robust to quality reduction from most of the applied augmentation techniques.

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

Secondary Reader: John Samsundar, Ph.D.

Secondary Reader: Cleon Davis, Ph.D.

Secondary Reader: Joseph S. Peri, Ph.D.

Table of Contents

List of Tables

List of Figures

Chapter 1

1 Introduction

1.1 Problem Statement

This thesis develops a foundation and a framework for license plate recognition on low-quality, noisy images. With regards to performance of license plate character identification, this thesis does not address reaching, nor evaluation against, human levels of performance. In the long term, automated recognition accuracy has the potential to be better than that of humans. If networks that meet human performance thresholds can be developed, automatic license plate recognition can be applied to novel applications, including digital forensics.

1.2 Problem Space

1.2.1 Forensics Problem

Automatic license plate recognition (ALPR) is a common computer vision task used across many domains including 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 for which commercial ALPR systems are useful. Other potential applications of ALPR, like digital forensics, have a need to identify license plates from noisy images. These noisy images are frequently obtained on non-dedicated camera equipment under poor lighting conditions with unknown camera specifications and locations. For example, consider an ALPR system, with closed-circuit television (CCTV) footage, attempting to identify the license plate of a car at a crime scene. For the purposes of forensics, license plate characters must be identified manually since commercial ALPR systems do not perform reliably on low-quality images. 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 has the potential to provide benefits where the identification of license plates is needed, or where the identification is not currently possible but would improve situational information.

1.2.2 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, like 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 is a subset of Machine Learning that allows for networks to learn increasing abstraction without significant human interaction by using many layers in a network. Simply put, Machine Learning and Deep Learning allow for computers to solve problems of logical abstraction by training on large datasets.

Machine Learning has begun to obtain results that are as good as, or even better than, those of human object detection and identification. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [2] evaluates algorithms for object detection and image classification across a large dataset with many classes. ILSVRC is run every year, which allows for observation of the improvement of state-of-the-art networks over time. Human performance has been evaluated on the dataset used for ILSVRC. Consequently, a comparison between network performance and that of humans can be determined. As shown in Figure 1, object classification networks began to exceed human performance on this dataset as early as 2015 with the Inception-v3 network [16].

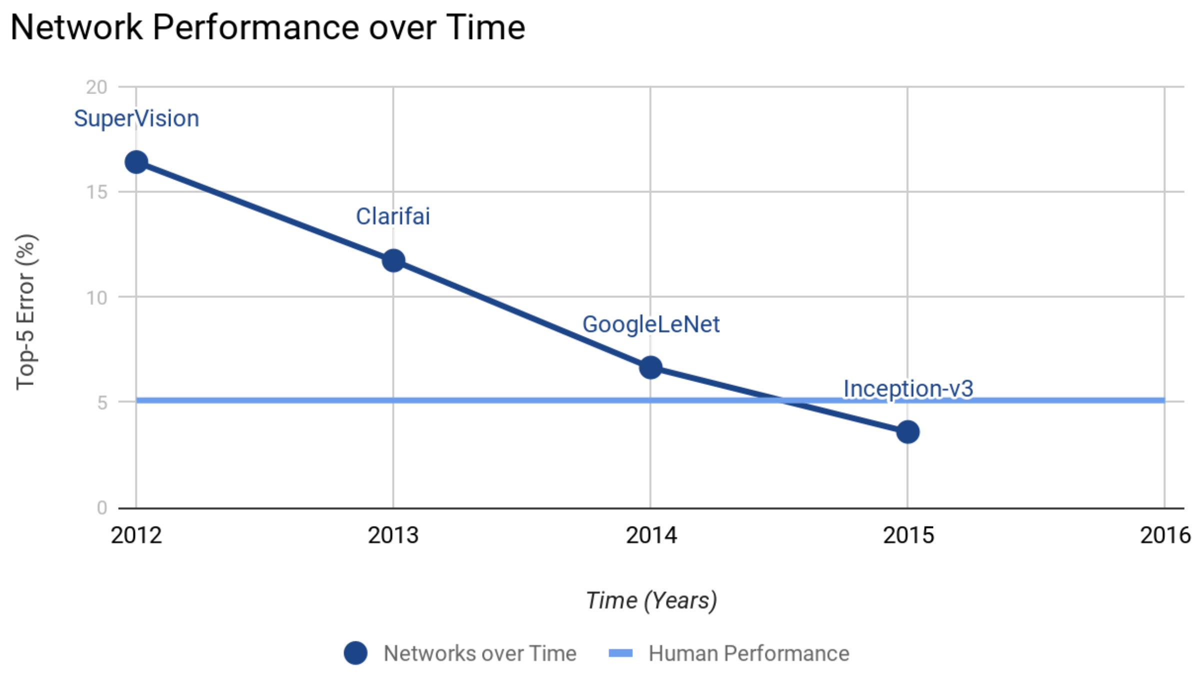


Figure 1: ILSVR Challenge scores over time vs. human performance, data from [2, 6].

1.3 Past Work

1.3.1 Limitations without Machine Learning

Not all previous ALPR approaches use Machine Learning. Optical character recognition (OCR) is a common non-Machine Learning-based ALPR solution. 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. 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] shows the implementation of an OCR driven license plate recognition system that achieves 95% accuracy for Arabic characters from high-resolution digital images. OCR, however, begins to face 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.

1.3.2 Multilayer Perceptron (MLP)

There is a need for systems that can perform license plate character recognition on images with low quality. Since OCR performance degrades on these images, alternate methods such as utilizing Machine Learning, without OCR, 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 perceptrons, 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. 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. 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 dimensional space) cannot be separated into classes by a linear function [1].

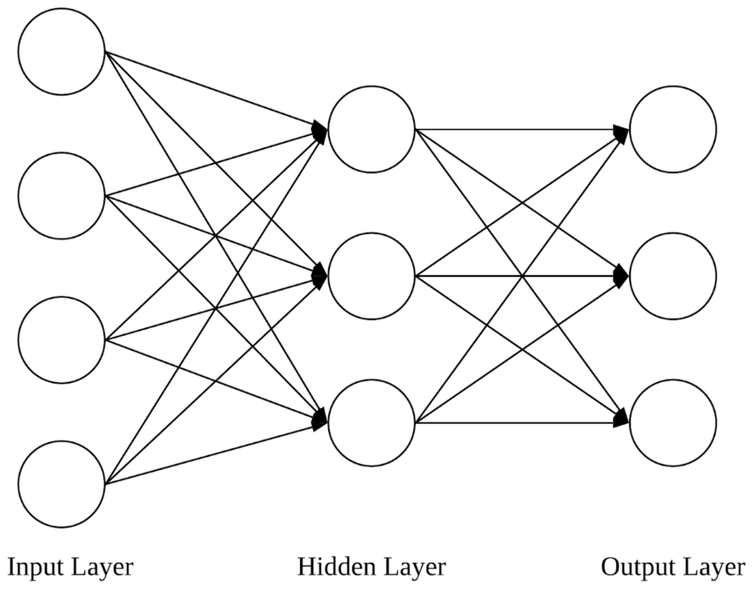


Figure 2: Basic MLP Architecture

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.

1.3.3 Convolutional Neural Networks (CNNs)

In object detection problems, Convolution Neural Networks (CNNs) are one of the most used Machine Learning networks. Basic neural nets like the Multilayer Perceptron are intended for one-dimensional data and do not scale well into image problems. CNNs were designed for multi-dimensional data. 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.

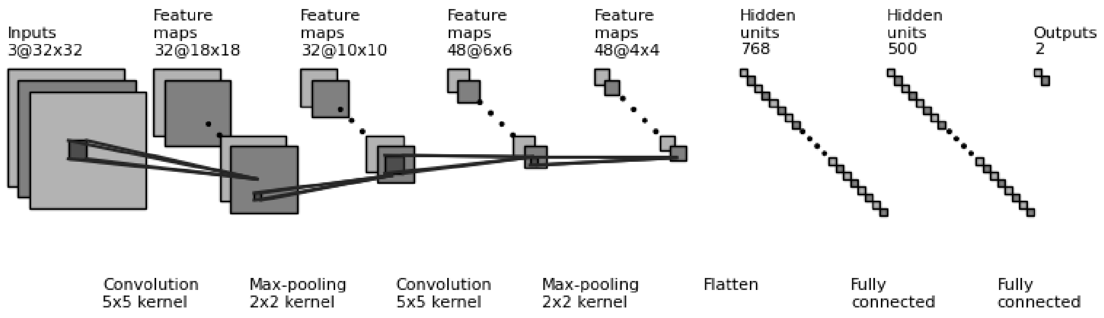


Figure 3: Basic CNN Architecture

Four key ideas drive the CNN structure: local connections, shared weights, pooling, and the use of many layers. In a CNN, local connections are employed by having each neuron connected to only a subset of the input image. This contrasts with 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 in a particular feature map share weights. These shared weights along with local connections help reduce the number of parameters in the system. Pooling allows the CNN to merge similar features and to progressively reduce the number of parameters. Reducing the number of parameters makes computation more efficient. Pooling also has the added benefit of suppressing noise. In a CNN, the first few stages are convolutional and pooling layers. These layers allow the network to take advantage of the improvements in computational efficiency and noise suppression. The result of these stages is then flattened into a column vector via a fully connected layer and fed into a feed-forward network like the MLP to perform classification. CNNs were neglected in the Machine Learning field for many years. The advent of more powerful computing resources and

GPUs allowed CNNs to go deeper, which means adding many layers. Making CNNs deeper dramatically improves their capability for object detection. This improvement 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 approach was improved further by the incorporation of temporal data from video capture.

1.3.4 Recurrent Neural Networks (RNNs)

The previously discussed networks are memory-less. The introduction of recurrence in Recurrent Neural Networks (RNNs) allows for the addition of short-term memory. Short-term memory provides improved performance for sequential information. Simple RNNs are similar to MLPs except that RNNs also have recurrence. Recurrence means having connections of a perceptron to itself, or to other perceptons. The recurrence acts as short-term memory. Figure 4 shows a simple RNN with the recurrence in red.

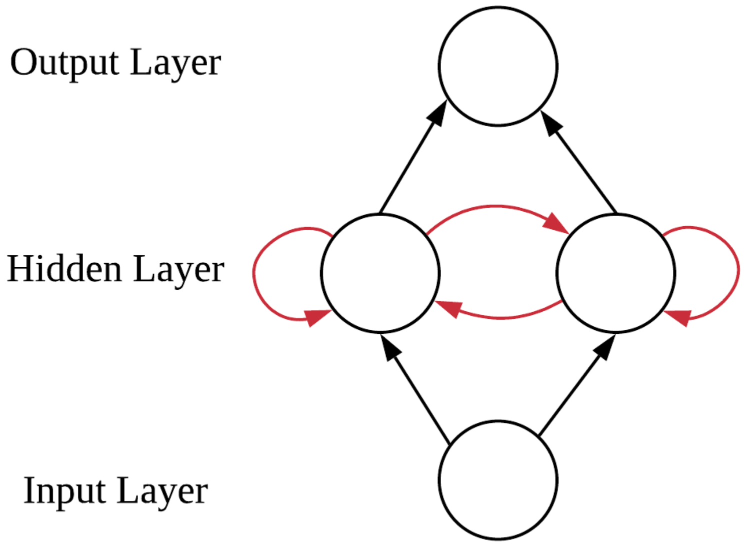


Figure 4: Simple RNN. Recurrence is shown in red

The capabilities of RNNs have been extended to allow for long time gaps in the sequence by using long short-term memory (LSTM) cells, shown in Figure 5. This memory is gated, meaning that the unit decides whether 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. Training is the process of learning a mapping function between input and output. The relationship between input and output variables can be described as a complex mathematical function. The goal of training is to learn the value of the parameters (weights) that result in the best function approximation. Finding optimal parameters requires solving a non-convex optimization problem. The weights, therefore, are learned by using an iterative process to navigate the error surface. A model with a specific set of weights can be evaluated on the training dataset to determine model error. A change to the model weights will result in a change to model error. A set of weights that results in the smallest possible error is sought. [1, 28]

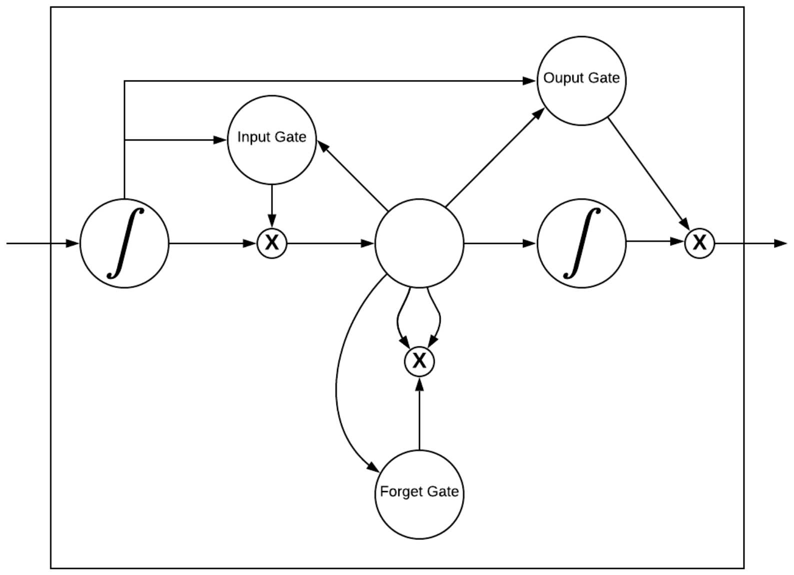


Figure 5: LSTM Cell

Li, *et al.* [8] uses RNNs with LSTMs to perform character recognition on license plates. One benefit of this method is that the system can process the whole string of license plate characters without the need for segmentation.

1.3.5 Poor Quality License Plates

The approaches discussed above have been trained on datasets that do not apply noise deliberately or evaluate performance with respect to the noise in the image. This thesis focuses on evaluating ALPR for degraded imagery. Previous work in identifying license plate characters under deliberate application of degradation are discussed in this section.

1.3.5.1 Motion Blur

Motion blur is an image degradation technique 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 the exposure time. However, shortening exposure time has the disadvantage of higher noise. Higher noise imposes higher requirements on image sensor quality. Motion blur can also be reduced by providing stronger illumination. These methods of reducing motion blur are not always possible or cost-effective. Therefore, deblurring in post-processing is considered a viable alternative. Svoboda, *et al.* [9] developed a CNN to perform character recognition on license plates with motion blurring artifacts. The CNN was a 15-layer network that consisted of only convolutional and Rectified Linear Unit (ReLU) layers. A ReLU is an activation function used in neural networks. If the input to a ReLU is positive, the input value is passed directly through to the output. If the input to a ReLU is negative, the ReLU output is zero. The CNN developed showed that Machine Learning had superior performance to OCR on motion-blurred images.

1.3.5.2 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. The SoftMax function is an activation function that outputs a probability distribution across the possible outcomes. This network provides 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 large dataset. These resources include: memory, computer processing unit (CPU) access, graphical processing unit (GPU) access, and time. GPU technology has been one of the pivotal advances that has allowed Machine Learning to solve problems with increasing complexity. Specifically, GPUs have allowed networks to go “deeper”. Therefore, this research 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]. SLURM is a resource management system commonly used by HPC centers. SLURM uses partitions to allocate resources to jobs. The partition, on which a job is requested, 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 interval during which a job can be run. Memory is tied in fixed amounts to a CPU core.

Therefore, the amount of memory available for a job is based on the number of 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

The project for this thesis was developed with a single GPU and six CPU cores for training. SLURM jobs are time limited. This project was designed such that training of the network is periodically saved. This means that if the SLURM job terminates in the middle of training, training can be resumed during a later job.

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. 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. 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

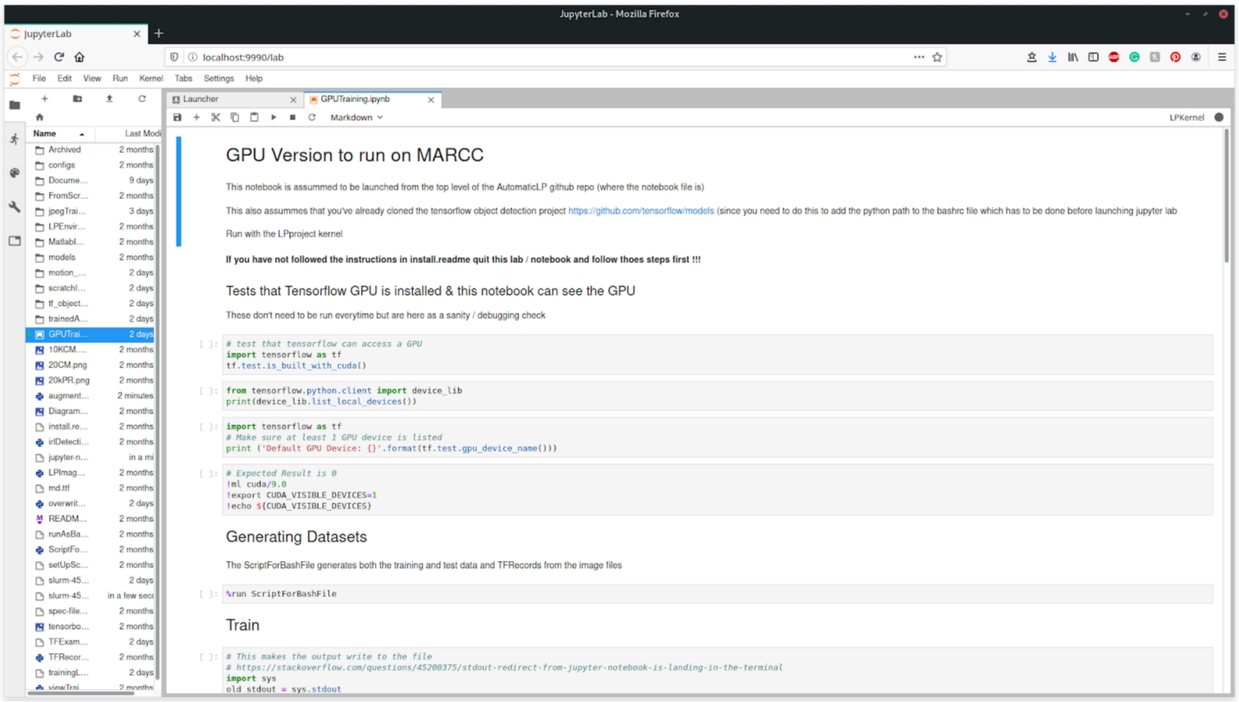
Given the barriers to using MATLAB, other Machine Learning frameworks were investigated. TensorFlow was ultimately selected. TensorFlow is a commonly used end-to-end open-source Machine Learning platform written in Python. TensorFlow provides a pre-built object detection API, which "makes it easy to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

2.1.3 User Interface

MARCC has a strict requirement that job execution shall not be run on system login. Rather, job execution on MARCC 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 to run Python scripts that leverage the TensorFlow library. Running purely through bash scripts, however, does not allow for user interaction in real-time. MARCC also offers the option of requesting a job on a partition that launches a Jupyter Notebook or Lab instance. Jupyter Notebook or Lab allows the user to run underlying scripts interactively. 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 a Jupyter Lab instance.

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. Running in real-time through Jupyter Lab allows different parts of the project framework to be run individually and allows for the modification of the configuration on the fly. A screenshot of the Jupyter Lab file for this project is shown in Figure 6.



GPU Version to run on MARCC

This notebook is assummed to be launched from the top level of the AutomaticLP github repo (where the notebook file is)

This also assummes that you've already cloned the tensorflow object detection project https://github.com/tensorflow/models (since you need to do this to add the python path to the bashrc file which has to be done before launching jupyter lab

Run with the LPproject kernel

If you have not followed the instructions in install.readme quit this lab / notebook and follow thoes steps first !!!

Tests that Tensorflow GPU is installed & this notebook can see the GPU

These don't need to be run everytime but are here as a sanity / debugging check

```
# test that tensorflow can access a GPU
import tensorflow as tf
tf.test.is_built_with_cuda()
```

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
import tensorflow as tf
# Make sure at least 1 GPU device is listed
print ('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
```

```
# Expected Result is 0
!ml cuda/9.0
!export CUDA_VISIBLE_DEVICES=1
!echo ${CUDA_VISIBLE_DEVICES}
```

Generating Datasets

The ScriptForBashFile generates both the training and test data and TFRecords from the image files

```
%run ScriptForBashFile
```

Train

```
# This makes the output write to the file
# https://stackoverflow.com/questions/45200375/stdout-redirect-from-jupyter-notebook-is-landing-in-the-terminal
import sys
old stdout = sys.stdout
```

Figure 6: Screenshot of Jupyter Lab instance running on MARCC

2.2 Framework

Having decided upon the computing resources through MARCC and a Machine Learning toolset, 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 here in brief and discussed in detail in the following sections.

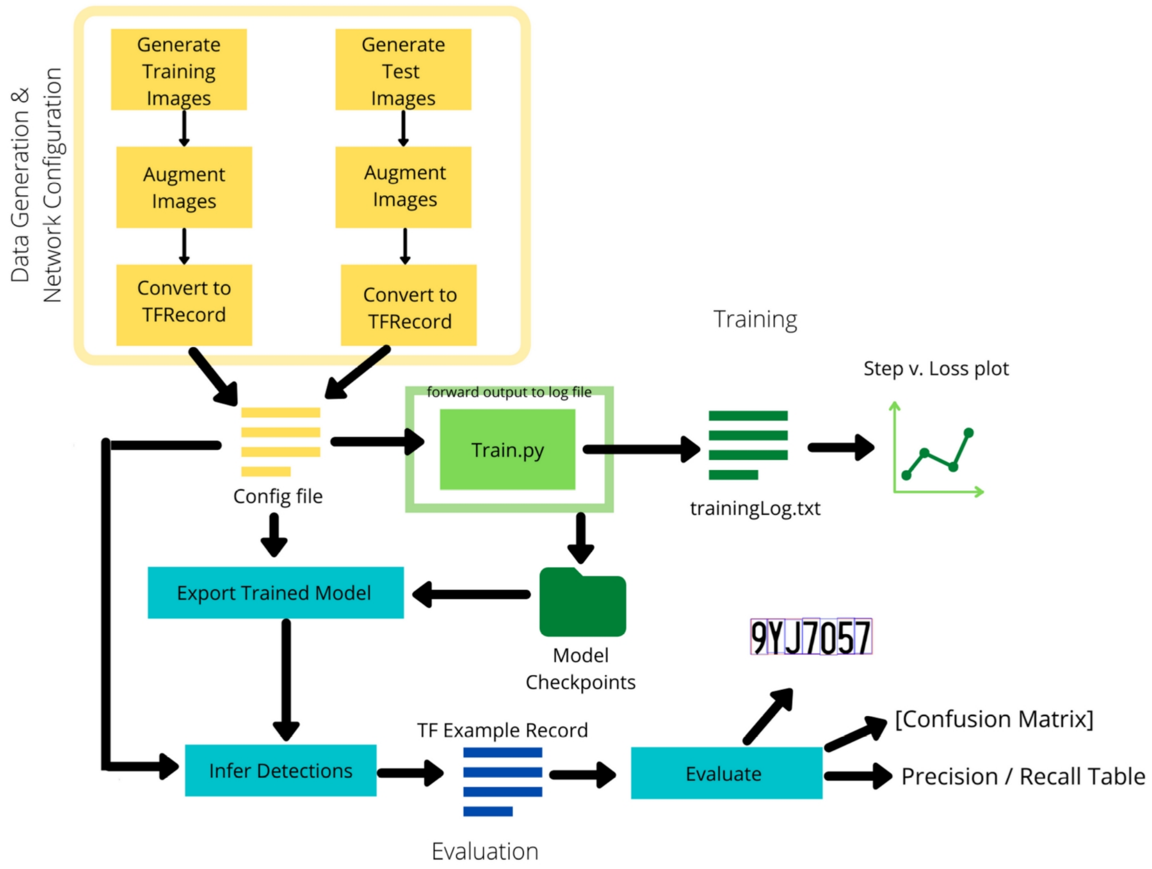


Figure 7: High-level Framework Flow

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. The goal of this project is to understand how image noise impact network performance. Network performance with respect to image noise is easier to evaluate on synthetic datasets where the level of introduced noise can be controlled. Therefore, images were synthetically generated. Once generated, the images were split into training and test data. Training data comprised 60% of the original dataset and test data the remaining 40%.

A dataset can be expanded past its initial size by applying data augmentation techniques. Data augmentation increases the amount of available data by applying various algorithms to transform the original dataset. Data augmentation also adds variation to an “ideal”, that is a non-degraded, dataset. Applying data augmentation techniques to the training dataset can help the trained network become robust to variations in the input data. Additionally, data augmentation on the test dataset allows performance evaluation of Deep Learning algorithms for the applied degradation.

Once a training dataset is generated, a network needs to be defined. The network is defined through a configuration file. The configuration file specifies the type of network architecture and training variables. Training variables specified in the configuration file include the hyperparameters, batch size and learning rate. Batch size is the number of samples, in this case training images, to process before updating model parameters. Learning rate controls the amount by which the network weights and biases change.

Once data is generated and a network is defined, the training data is fed into the TensorFlow Object Detection API executable. Labeled data is converted into a file format called a TFRecord to feed into the executable. As shown in Figure 7, this produces two outputs (when run through the Jupyter Lab Configuration): a log file and “Model Checkpoints”. The log file is converted into plots of training loss. “Model Checkpoints” are essentially saved files of training progress that capture the state of the network at a given time during training. These “Model Checkpoints” allow training to be restarted if processing is interrupted due to a MARCC processing queue timeout.

Evaluation is broken into three steps. First, the “Model Checkpoints” are converted into a “frozen graph” that 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. Finally, the information in the TF Example Record is converted to various object detection and classification performance metrics.

2.2.1 Configuration

2.2.1.1 Dataset Generation

The images generated to develop baseline performance are license plate strings that are compliant with the rules regarding the most recent Maryland passenger plates on a blank background. 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. 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.

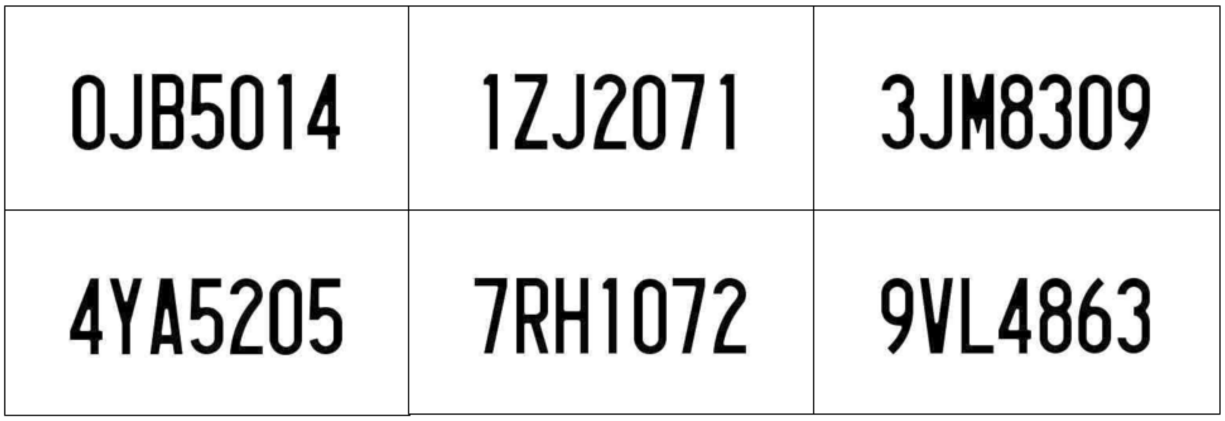


Figure 8: Sample Generated "Ideal" License Plate Images

2.2.1.1.1 Data Augmentation

Data augmentation techniques are applied to generated training images in order to increase the training dataset size and produce degraded images. Common data augmentation techniques include: JPEG compression, motion Blur, affine transforms, and Gaussian noise. Examples of some data augmentation techniques are shown in Figure 9.

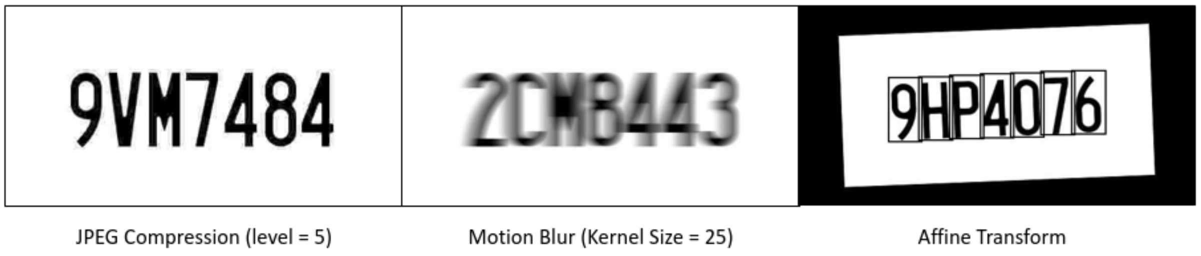


Figure 9: Commonly used data augmentation techniques

A Python library called Imgaug was used to apply data augmentation techniques to the generated datasets. Any augmentation technique used on a dataset needs to be used in the context of the problem being solved. Even if a data augmentation technique can be applied, the question of whether 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 whether the technique is applicable to the problem space and whether it was evaluated in this project. As with any project, this one is limited in scope and focused on a subset of applicable 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 black and white 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 technique was implemented via configuration file not Imgaug

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

2.2.1.1.2 Implementation

The license plate image generation script, run through this project's Jupyter Lab interface, produces two directories of "ideal" license plate images and a comma separated value (csv) file of ground truth data for each directory. One directory is comprised of 60% of the samples and is used for training. The other is comprised of the remaining 40% and is used for evaluation. In order to apply data augmentation techniques 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 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.

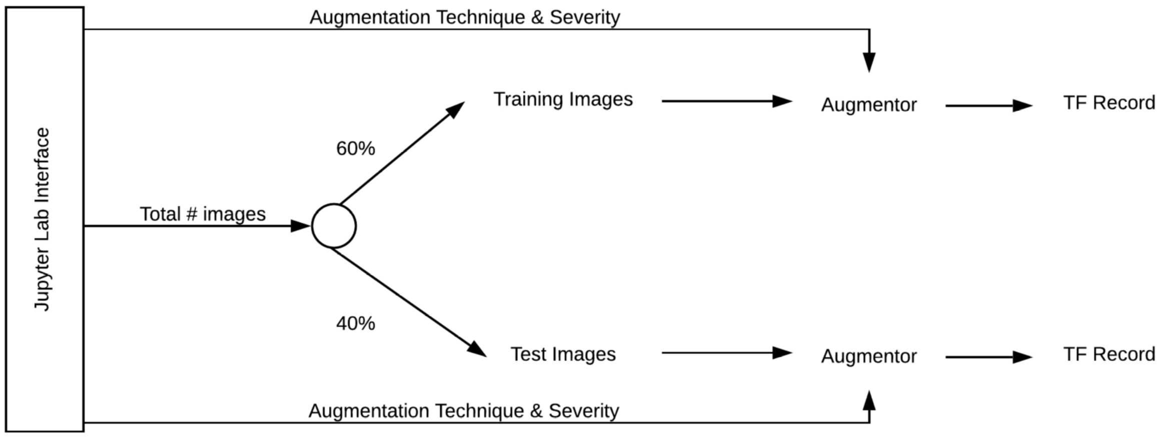


Figure 10: Data Generation process flow

2.2.1.1.3 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.

Protocol buffers 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 output to the correct class and ground truth. During training, knowing ground truth allows the network to determine the error and adjust the model weights and biases. Knowing ground truth of the test dataset allows for calculation of network performance metrics on the test dataset.

2.2.1.2 Network Definition and Setup

Once a synthetic dataset is generated, a network needs to be developed for training and testing with the data. For the TensorFlow Object Detection API, networks are defined through a configuration file. The configuration file 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 by simply changing a section of the configuration file. The configuration file can also define variables associated with training, like the hyperparameters. Driving network definition through this file allowed for rapid testing.

2.2.1.2.1 TensorFlow Object Detection Model Zoo

The TensorFlow Object Detection API provides the TensorFlow Object Detection Model Zoo. This consists of a collection of pre-trained models on common computer vision datasets and the configuration files associated with them. These configuration files and pre-trained models, for object classification tasks, provide a baseline for this project to perform transfer learning. Transfer learning is the process of taking a model trained on one task, e.g. object classification, and re-purposing the model to perform on a second task, e.g. character recognition. All the networks in the TensorFlow Object Detection API Model Zoo have shown high performance on object detection and classification tasks, but every network has its trade-offs. Therefore, testing different networks for the license plate character recognition task is prudent. The TensorFlow Object Detection Model Zoo allows for the rapid development of these different models for analysis of their capability for performance on the license plate character recognition task.

2.2.1.2.2 Image Resizer

The configuration file can control how image sizing is treated on input to the network. Control of image sizing is important because in Machine Learning every pixel of an image is treated as a discrete piece of information. Therefore, if the number of pixels representing an object, or in this case a character, are decreased by reducing the size of the image, the network has less information. Size, therefore, can be a limiting factor in the ability of a Machine Learning network to perform on a problem set. For network configuration, image size drives layer size and number to make localization and class determinations. Therefore, all images must be the same size when passed into the network. When raw input images vary in size, as 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.

When an image is fed into the network, the TensorFlow Object Detection API can resize the image in one of two ways. 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 configuration file. The “keep aspect ratio resizer” adjusts 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 adds zeros to the bottom and right of the image. For this project, a “fixed shape resizer” was used. The network was configured to expect images of a single size. 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, i.e. the application of an affine transform, the augmented images were already padded to the default size before processing to the TFRecord.

2.2.1.2.3 Configuration File Data Augmentation

The Imgaug library was used for applying most of the augmentation techniques. The Imgaug library has the capability to provide basically any augmentation technique that could be desired. Using an external library, however, requires additional piping. The options for data augmentation techniques in the configuration file are limited, but they are easy to apply with little additional effort. The implementation of Gaussian noise, for example, evaluated in this project, was implemented through the configuration file, rather than through an external augmenter.

2.2.1.2.4 Hyperparameters

The first hyperparameter controlled by the configuration file is the batch size. Batch size is the number of samples evaluated between updates of network weights and biases. Networks are trained as optimization problems, where weights and biases are updated based on error estimates. The more training examples (larger batch size) used in an estimate the more accurate the estimate will be. Even though a larger batch size, in theory, has more accurate estimates, 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 configuration file is the learning rate. The learning rate controls the amount the model changes with respect to the error, when model weights are updated. 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 can get stuck in a local minimum. A learning rate that is too high, on the other hand, can jump over minima missing the optimal 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 configuration file: 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. Manual step rate allows the learning rate to be explicitly specified based on what step of training is occurring. Manual step rate allows custom scheduling of learning rates, because the learning rate can be modified at as many training steps as desired.

2.2.1.2.5 Label Map

The TensorFlow Object Detection API requires a label map that provides an integer identification value to each class. However, TensorFlow does not allow zero to be used as an identification value. Based on this, 1-9 are mapped to their respective numerical values, and letters used in Maryland license plates are mapped to 10-31. Zero is mapped to 32.

2.2.2 Training

In Machine Learning, training is the process where a mapping function between inputs and outputs is learned. This is achieved by updating the weights of the network in response to an error between the network output and the training dataset. Updates are made continually to reduce error until loss is below the desired threshold or the maximum epochs are reached. Loss is the sum of classification loss and localization loss. Exactly how loss is determined depends on the specific network being trained. Once the loss is below the desired threshold, the model can be evaluated against test data. The evaluation process does not iteratively update weights and biases and, consequently, it is significantly less computationally intensive. Training and test data are distinct. After training has executed, the test data is used for network performance evaluation. An overview of the training process is shown in Figure 11 and described in detail in the following sections.

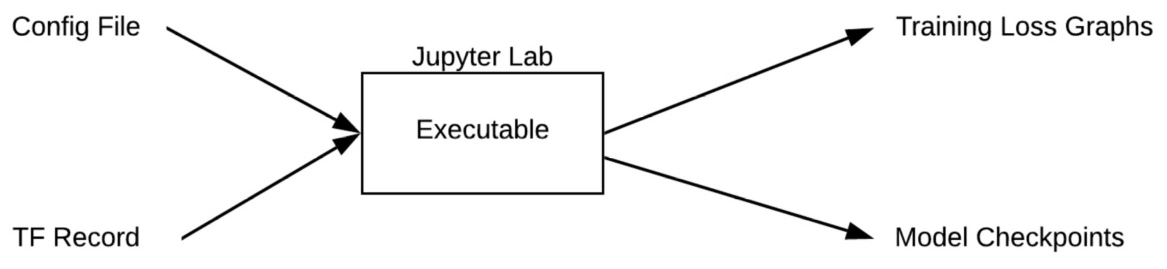


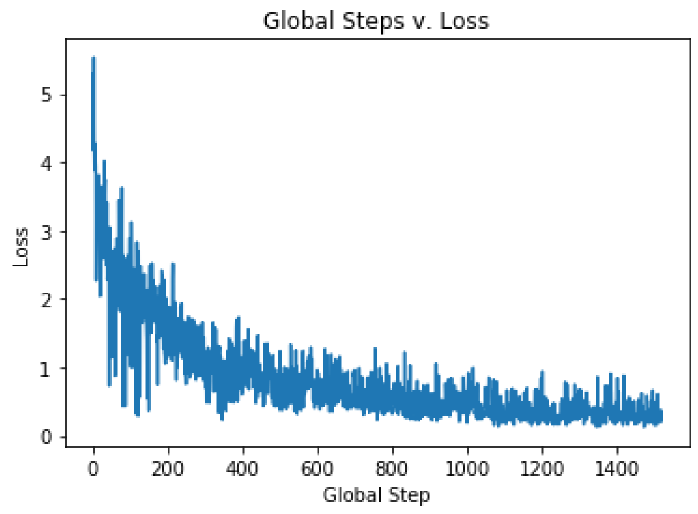
Figure 11: Flowchart of training process

2.2.2.1 Executable

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

2.2.3 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 performed.



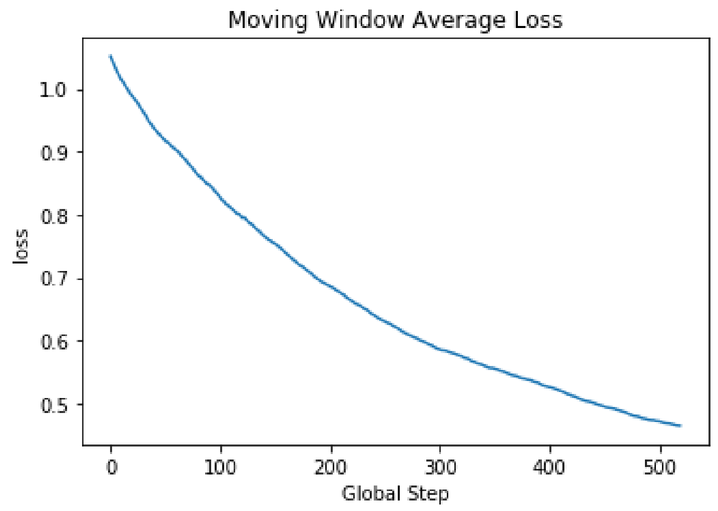


Figure 12: Sample plots of training loss. Top is per-step, bottom is moving average.

Sample plots of what is produced in the Jupyter Lab instance are shown in Figure 12. The top plot shows loss per training step. The loss per training step plot is frequently very noisy and, therefore, difficult to read especially with longer training runs which increase the density of the data. The bottom plot 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.

2.2.4 Evaluation

2.2.4.1 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.

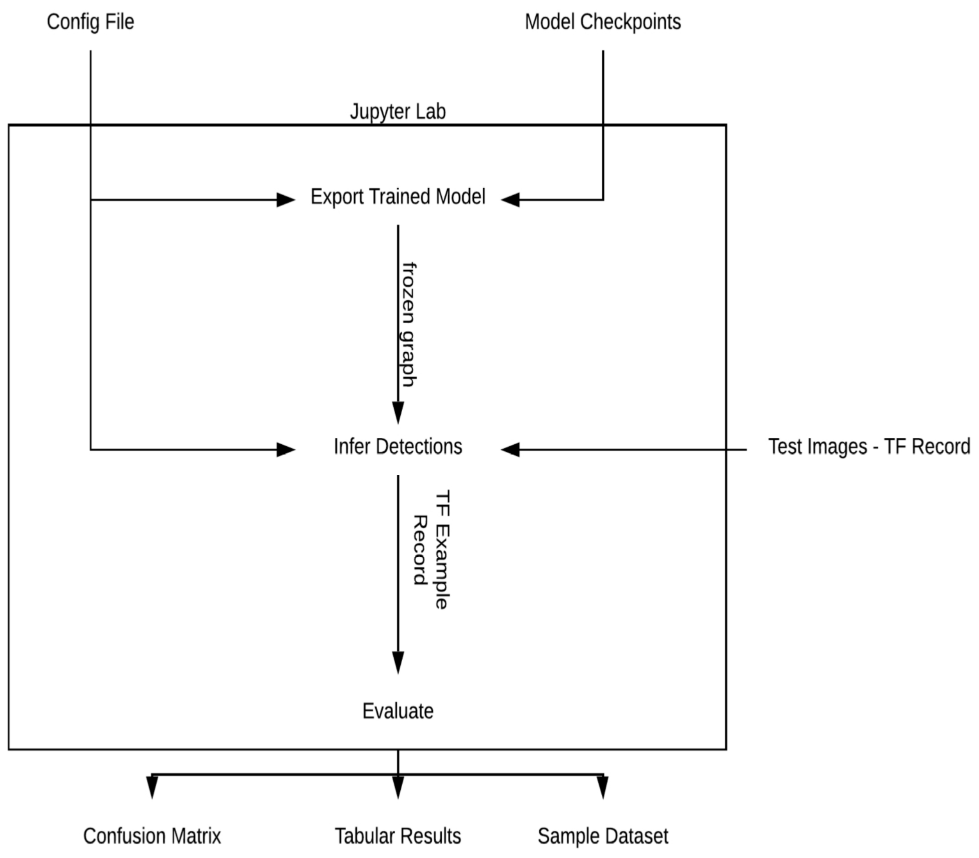


Figure 13: Flow of Evaluation

2.2.4.2 Object Detection Performance Tools & Metrics

In order to understand the results of the test images, the different performance metrics used are discussed in the following subsections.

2.2.4.2.1 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; the closer the prediction is to the truth, the higher the value. IOU is defined mathematically as, $IOU=\frac{truth\cap prediction}{truth\cup prediction}$. The intersection, $\cap$, of truth and prediction is the overlap area of the two bounding boxes. The union, $\cup$, of truth and prediction is the total area of the two bounding boxes. If the truth and predicted bounding boxes match exactly the IOU will be one.

2.2.4.2.2 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 in which the model correctly predicts the correct class. A False Positive (FP) is an outcome in which the model incorrectly predicts the correct class. A False Negative (FN) is the outcome in which the model fails to predict a class, when one exists.

2.2.4.2.3 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.

2.2.4.2.4 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 shown in Figure 14. Note that the last column shows false negatives and the bottom row shows false positives.

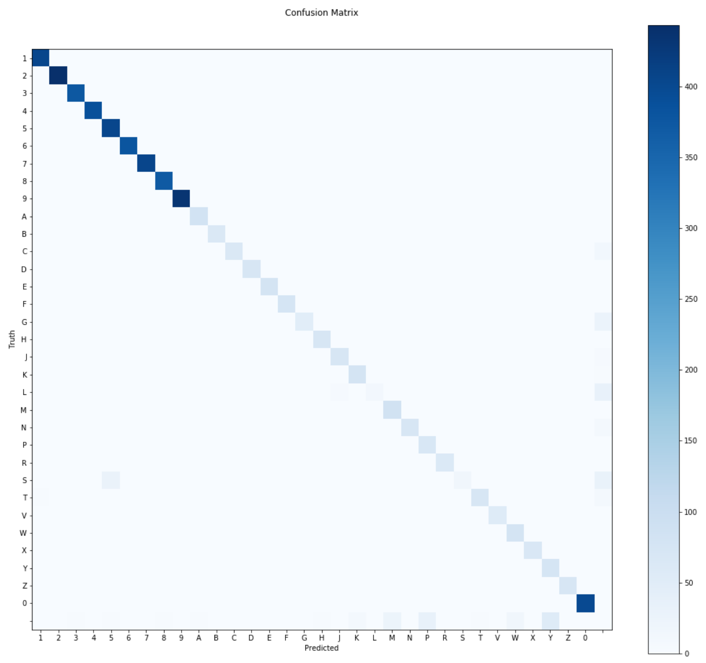


Figure 14: Sample Confusion Matrix. Note that for the license plate problem letters occur significantly less frequently than numbers which accounts for the relative shading.

2.2.4.3 Framework Outputs

When the evaluation is run through the Jupyter Lab instance, three outputs are generated. First, 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. Generation of the sample detections 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 Architectures and Object Detection Approaches

A Machine Learning architecture is the structure of the neural network and defines the type and number of layers. Object detection approaches define the way objects are localized and extracted. A model is the combination of an architecture and an object detection approach. An architecture can be combined with several different object detection approaches. While all of these approaches 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. For the license plate problem posed in this thesis, it is useful to evaluate the performance of these architectures and approaches. One can, then, understand the trade-offs that a given network 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 depth-wise separable convolutions, shown in Figure 15, which are a form of factorized convolutions that break down into 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, depth-wise convolutional layers, and ends with an average pooling layer, a 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 16.

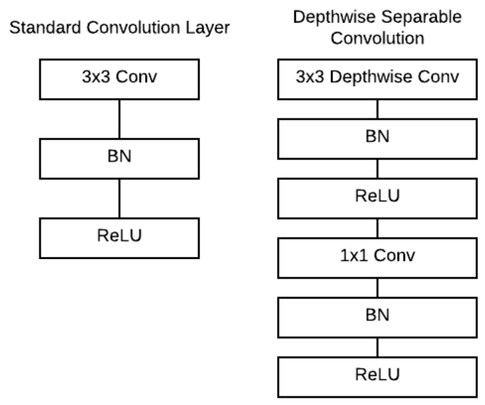


Figure 15: Comparison between standard convolution layer and depth-wise separable convolution that is used in MobileNet

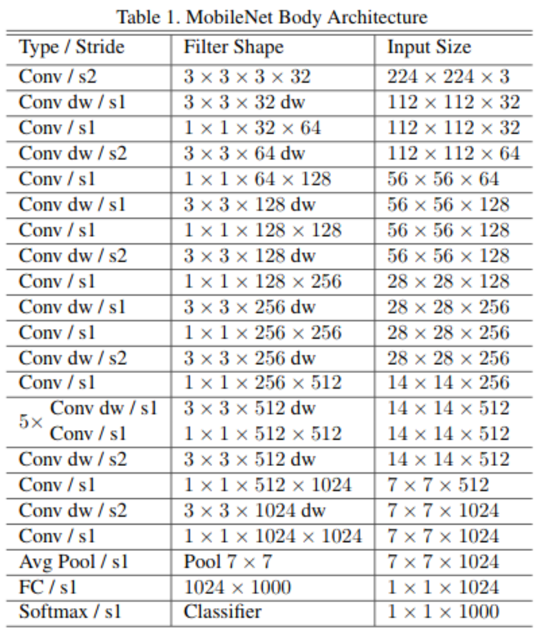


Table 1. MobileNet Body Architecture

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	$14 \times 14 \times 512$
	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	$1 \times 1 \times 1000$

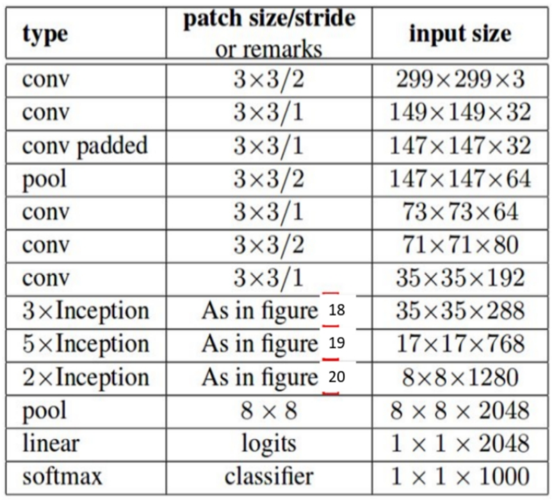
Figure 16: Full Architecture of MobileNet [13]

Although not the point of this 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. Given that MobileNet is developed to perform well on resource-restricted systems it may be a good choice for ALPR systems on mobile phones or embedded 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: (1) choosing a kernel size can be difficult and (2) increasing depth of networks can result 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. This means 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 more deeply to achieve performance. Deep networks have two downsides; first, they are 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 the “representational bottleneck”, 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. The 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.

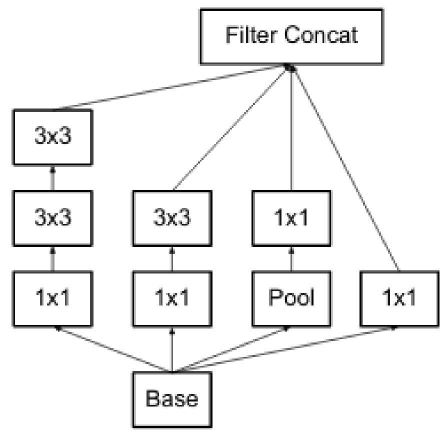


Figure 18: Inception module where 5x5 convolution is replaced by two 3x3 convolutions [14]

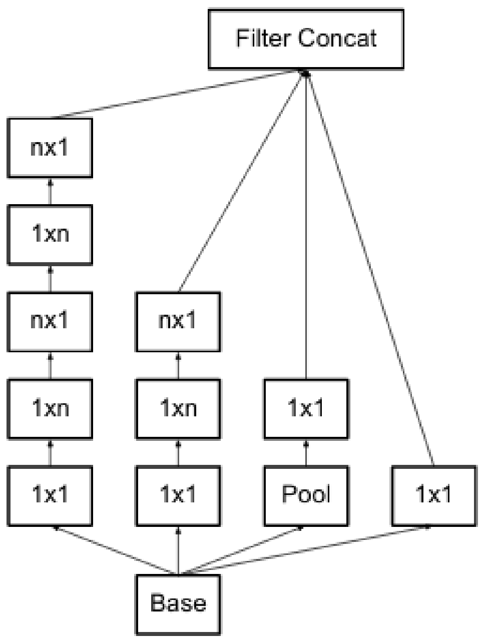


Figure 19: Inception modules after the factorization of the nxn convolutions [14]

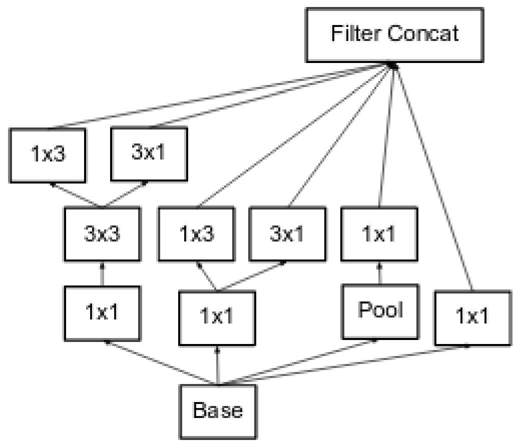
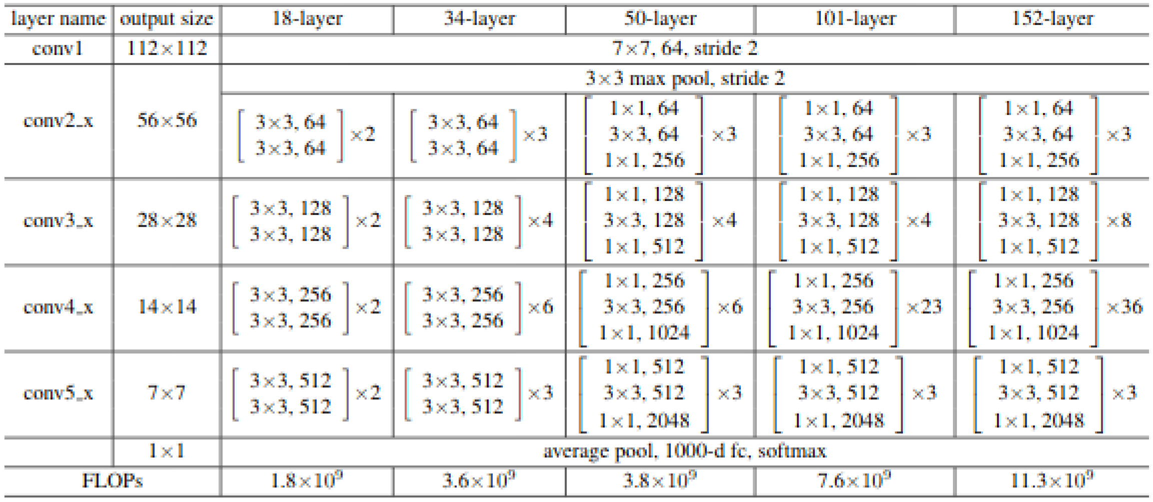


Figure 20: Inception module with an expanded filter bank [14]

Inception v2 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]

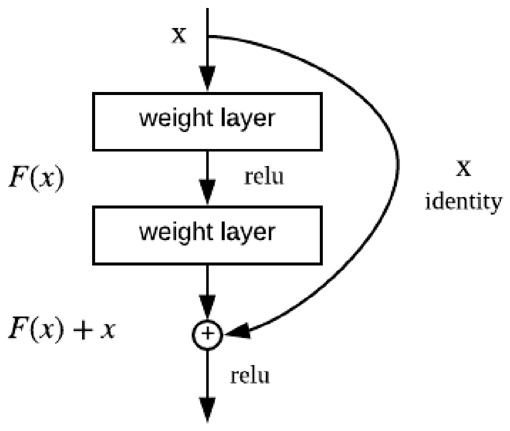


Figure 22: ResNet building block [15]

ResNet networks are classified by the number of layers of which they are composed. The 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 allow 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.

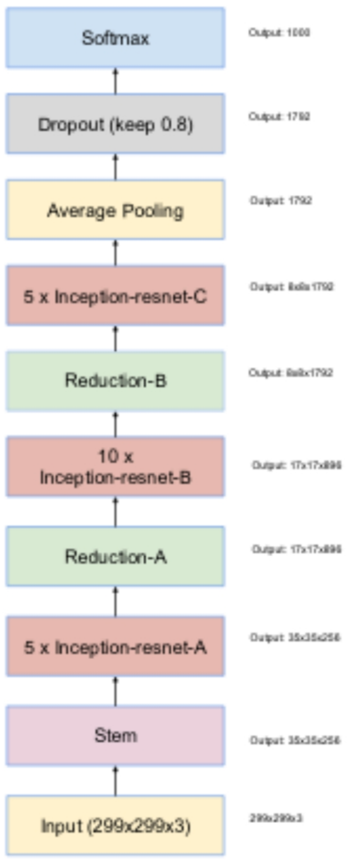


Figure 23: Schema for Inception-ResNet network [16]

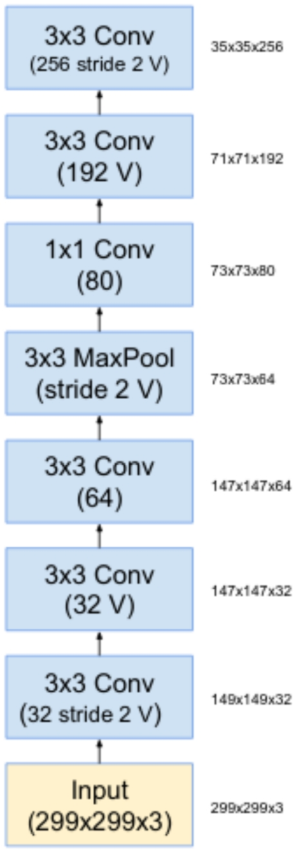


Figure 24: Stem for the Inception-ResNet-v1 network [16]

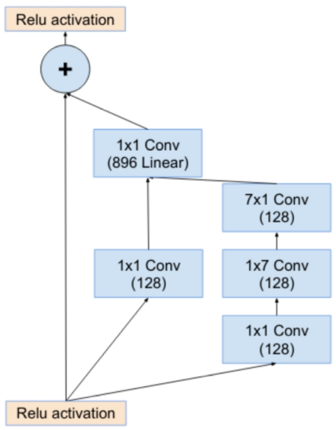


Figure 25: Schema for the 17 x 17 (Inception-ResNet-B) module of the Inception-ResNet-v1 network [16]

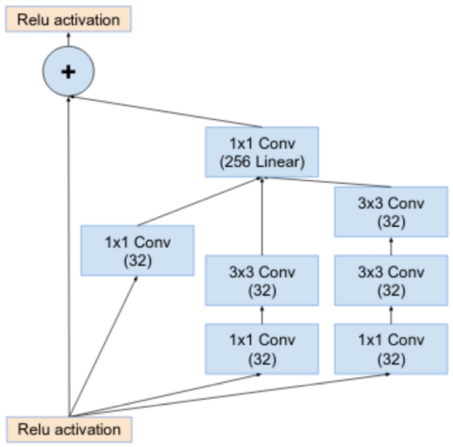


Figure 26: Schema for 35 x 35 grid (Inception-ResNet-A) module of the Inception-ResNet-v1 network

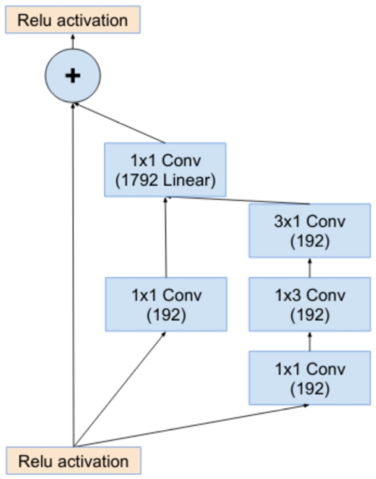


Figure 27: Schema for 8 x8 grid (Inception-ResNet-C) module of Inception-ResNet-v1 network [16]

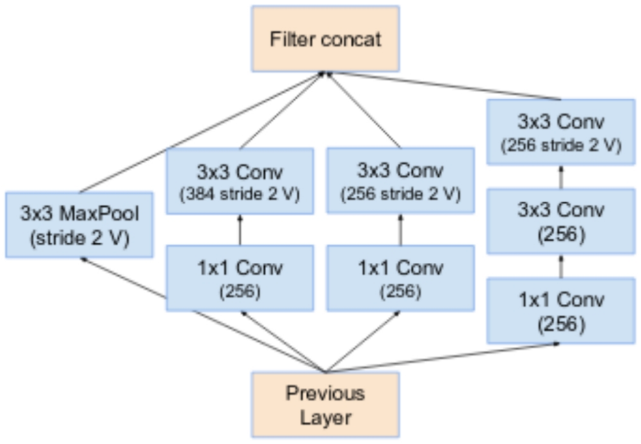


Figure 28: "Reduction-B" 17x17 to 8x8 grid-reduction module [16]

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 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) is not an architecture per-say but rather the process of automating architecture engineering. The process behind NAS finds the optimal architecture from all possible architectures by following a search strategy that will maximize performance for the given problem. In the TensorFlow Object Detection API, NAS is implemented in the same way that architecture is selected (through the configuration file) and is included here for that reason. NAS is appealing as an option for selecting an architecture. If the optimal architecture for any given problem can always be found with this approach, in principle this should be the default option to solve any problem including that of the ALPR task.

2.3.2 Object Detection Approaches

2.3.2.1 Single Shot Multibox Detector (SSD)

Single Shot Multibox Detector (SSD) is an object detection approach that reduces the computational intensity by eliminating bounding box proposals and subsequent pixel or feature resampling. Compared to previous approaches 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 will adjust 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.

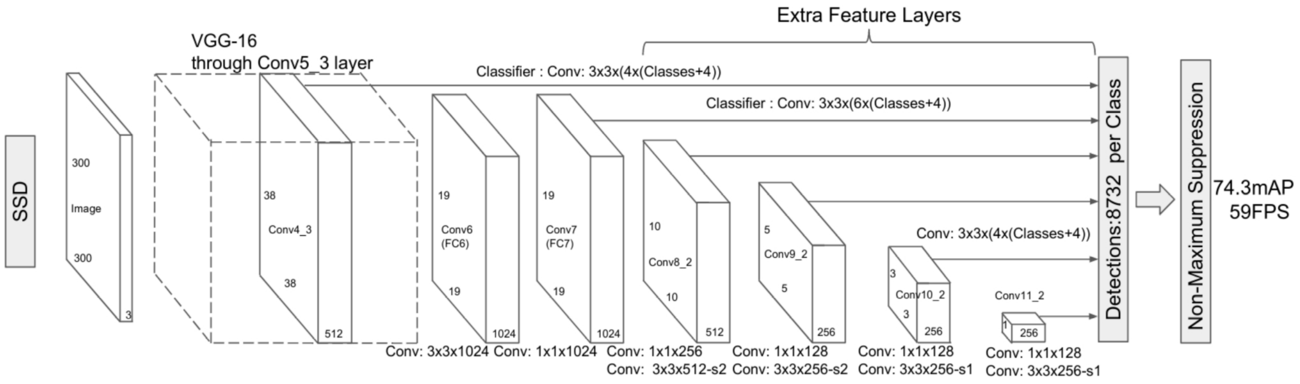


Figure 29: SSD schema[17]

2.3.2.2 Faster R-CNN

Faster R-CNN is based on Fast R-CNN which in turn is based on R-CNN. R-CNN was developed to detect and localize an object within an image. In order to minimize the number of potential regions that may contain the object, 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 box 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, 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. At each point, it outputs 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 it to a classifier [19] [27]. The overall network process for the Faster R-CNN is shown in Figure 30.

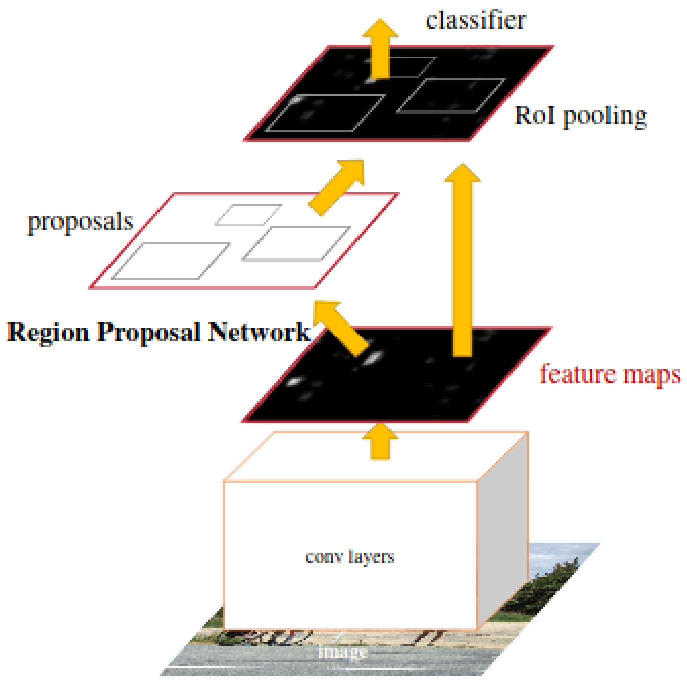


Figure 30: Faster R-CNN network for object detection [19]

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

Previous region-based detectors have the problem of applying a network to an image hundreds of times to determine ROIs. This reapplication of the network on each region is a computationally costly endeavor. R-FCN, like Faster R-CNN, attempt to solve this problem. In R-FCN, the same region proposal networks from the R-CNN networks is used. Unlike the Faster R-CNN network, the fully connected layers are moved to before the ROI pooling layer for R-FCN. The fully connected layers generate score maps which perform average voting to determine scores [25]. The schema for the R-FCN approach is shown in Figure 31.

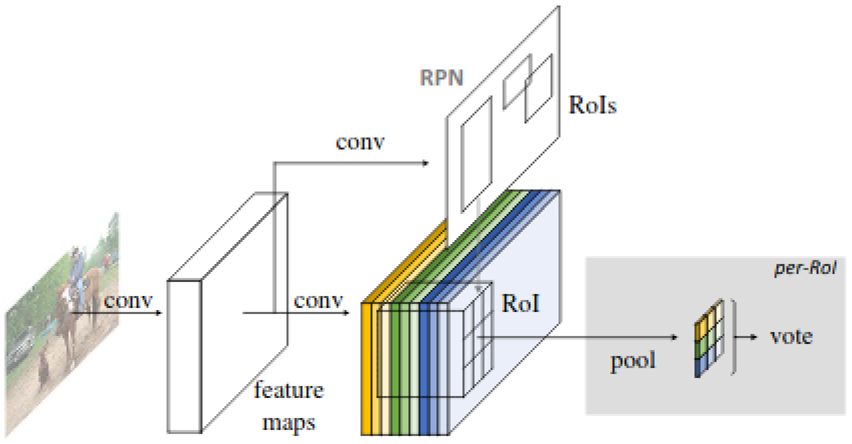


Figure 31: Overall schema of R-FCN [25]

Chapter 3

3 Results

3.1 Model Results

In order to determine the optimal network for identifying license plate characters in the idealized dataset, several architecture / object detection approach combinations were trained and evaluated against the baseline dataset. A brief summary of the tested approaches 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

Object Detection Approaches	Architecture	Relative Training 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 low. With the SSD Inception v2 network, however, not all characters in the test dataset license plates were detected. For example, all 2063 instances of “3” in the test dataset were false negatives. This poor performance is shown in the confusion matrix Figure 32 and listed in 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 predictions match truth.

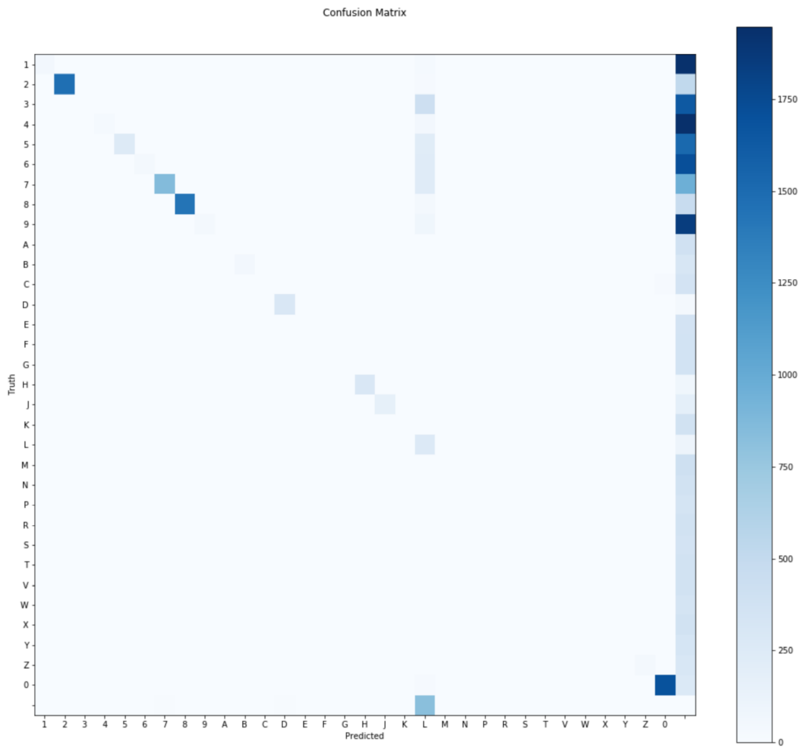


Figure 32: Confusion Matrix for SSD Inception v2

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. Although precision was mostly in the 80 and 90 percent range, recall was even worse than the SSD Inception v2 network. The results are shown in the confusion matrix, Figure 33, and in more detail in Table 5.

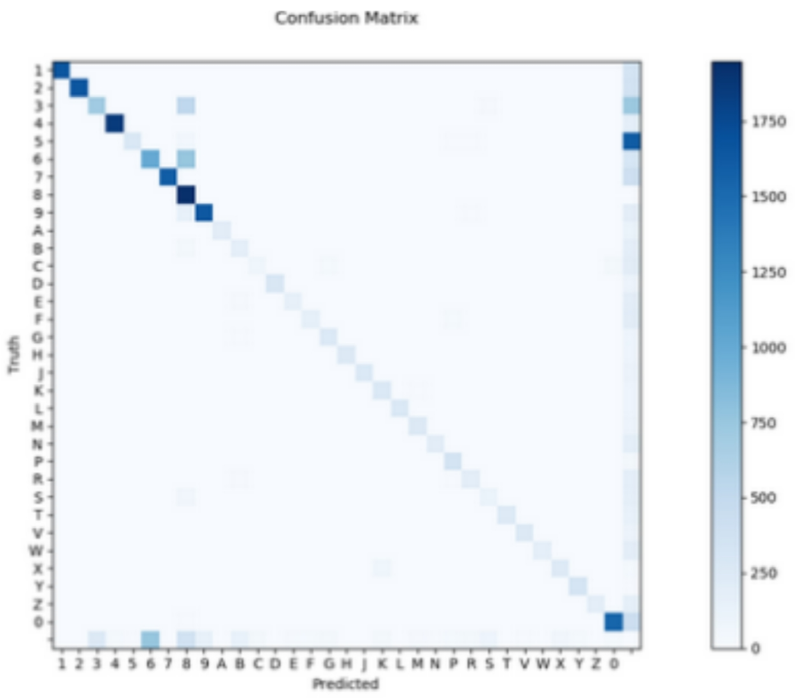


Figure 33: Confusion Matrix for SSD MobileNet

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. There also were 75 false negatives. The results are shown in the confusion matrix in Figure 34 and the tabular results in Table 6.

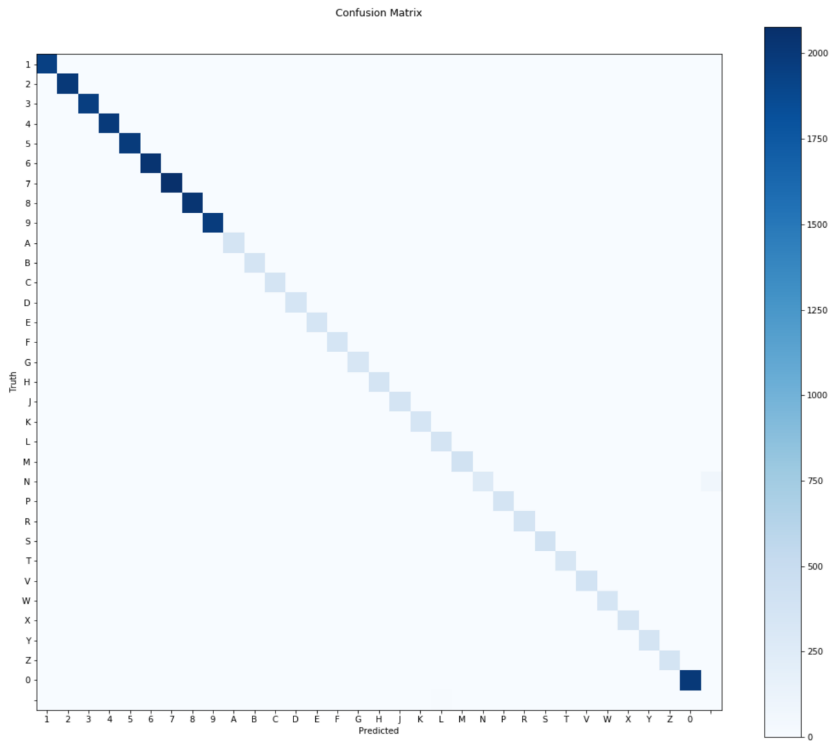


Figure 34: Confusion Matrix for Faster R-CNN Inception v2

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. The network did have false positives for “1”, “P”, “T”, and “Y” as well as a false negative for “Y”. The results for this network are shown in the confusion matrix, Figure 35, and more clearly in the tabular results in Table 7.

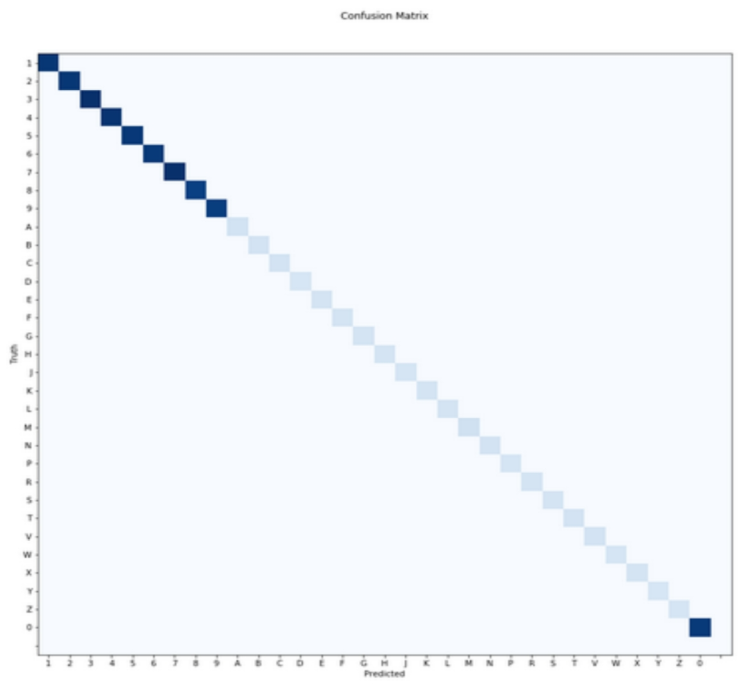


Figure 35: Confusion Matrix for Faster R-CNN ResNet 101

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. There is a single false positive on “L”. The results for this network are shown in the confusion matrix, Figure 36, and tabular results in Table 8. These results show that this network has slightly better performance than Faster R-CNN ResNet 101.

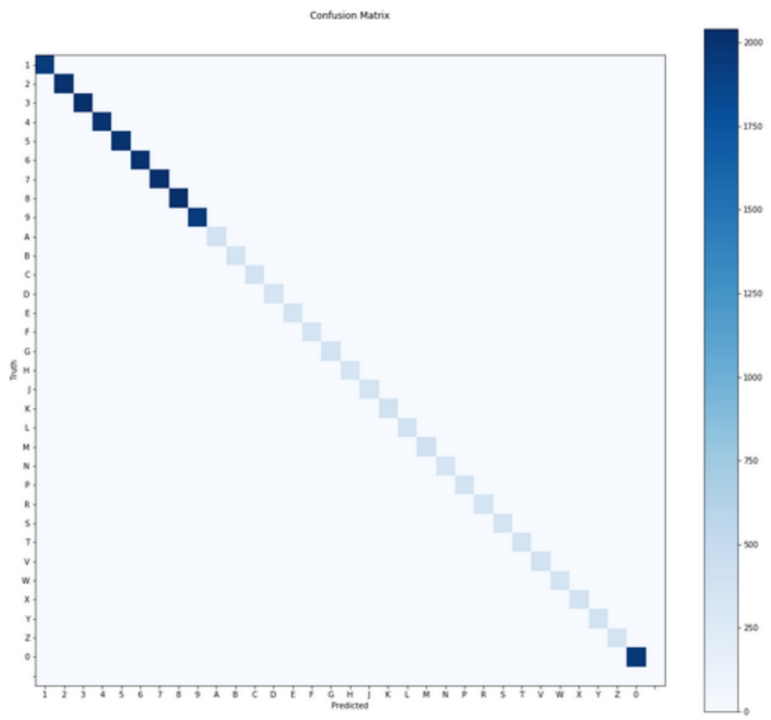


Figure 36: Confusion Matrix for Faster R-CNN ResNet 50

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 slow 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.

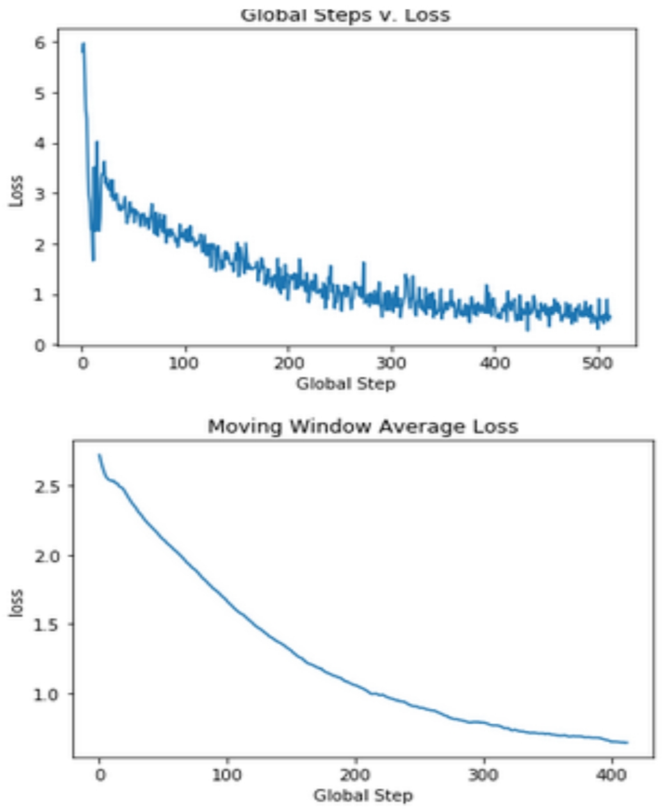


Figure 37: Training loss for Faster R-CNN Inception-ResNet

3.1.7 Faster R-CNN NAS

Faster R-CNN was trained with the Network Architecture Search. Compared to training Faster R-CNN with ResNet 101, ResNet 50, Inception v2, and even Inception-ResNet, training and evaluating test images 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.

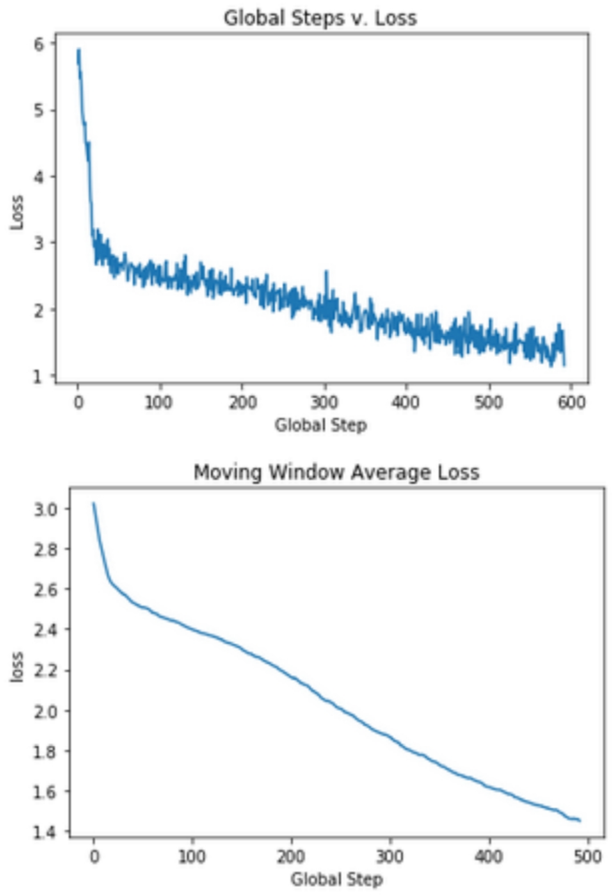


Figure 38: Training loss for Faster R-CNN NAS

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”.

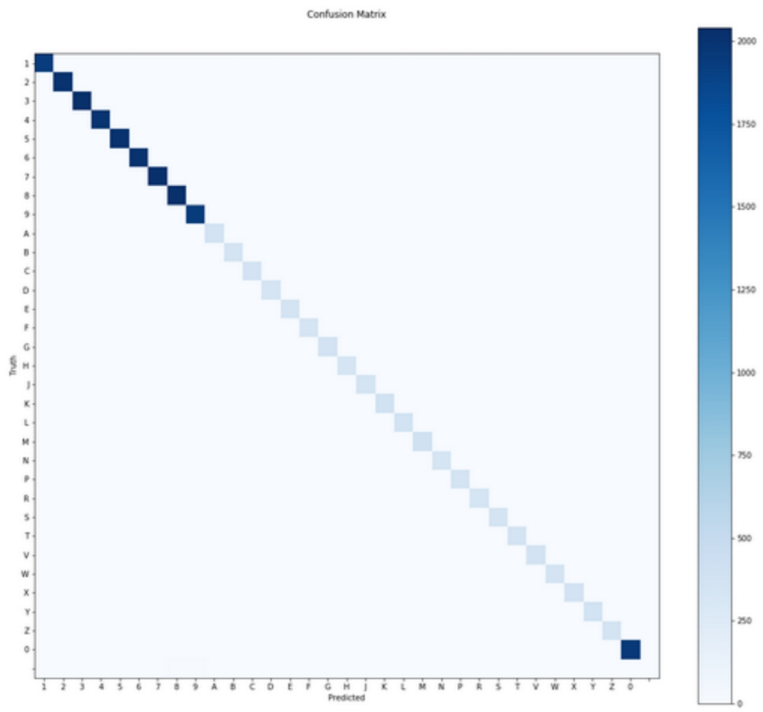


Figure 39: Confusion Matrix for R-FCN ResNet 101

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

Several models were trained against the baseline license plate dataset. The resulting precision and recall were evaluated and averaged across all possible license plate characters for Maryland plates. The results of this comparison are shown in Figure 40.

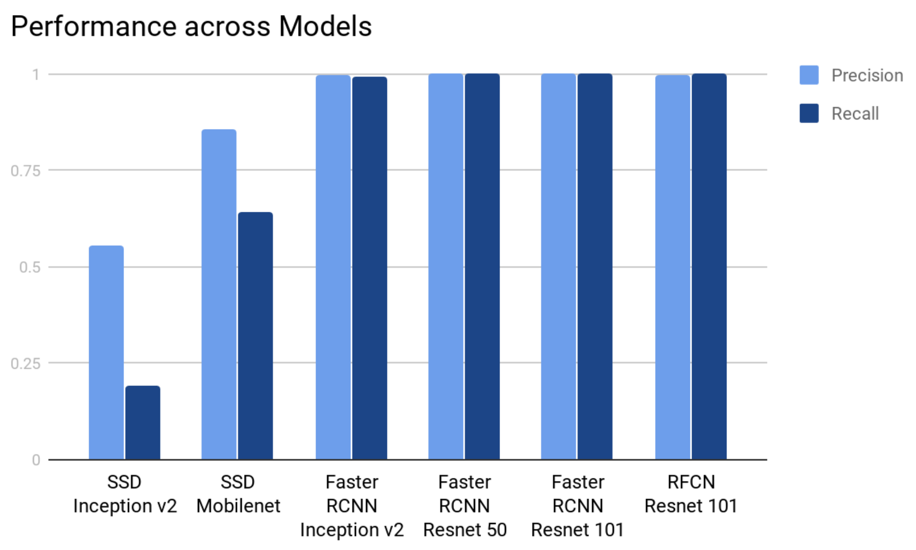


Figure 40: Performance across different architectures

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 on the Faster R-CNN and R-FCN approaches. Of these three top-performing networks, Faster R-CNN ResNet 50 was chosen as the network to proceed with in this thesis.

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 runs on datasets with augmentation techniques applied, 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.

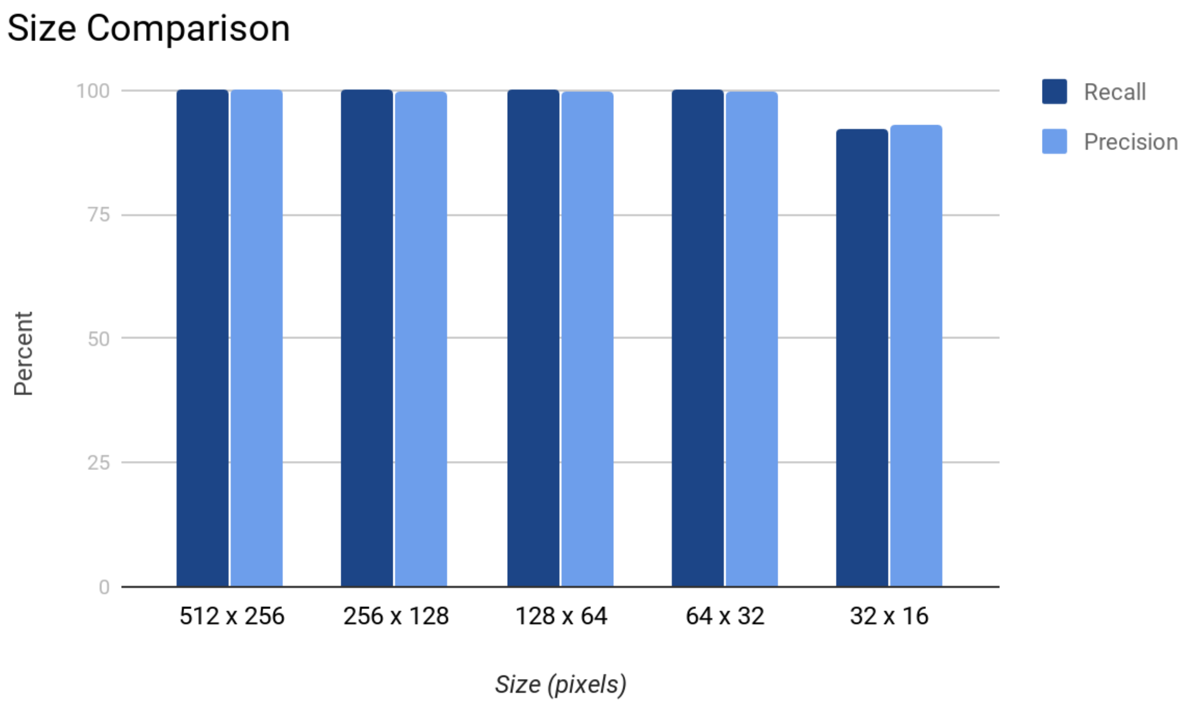


Figure 41: Performance across different image sizes

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. 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.

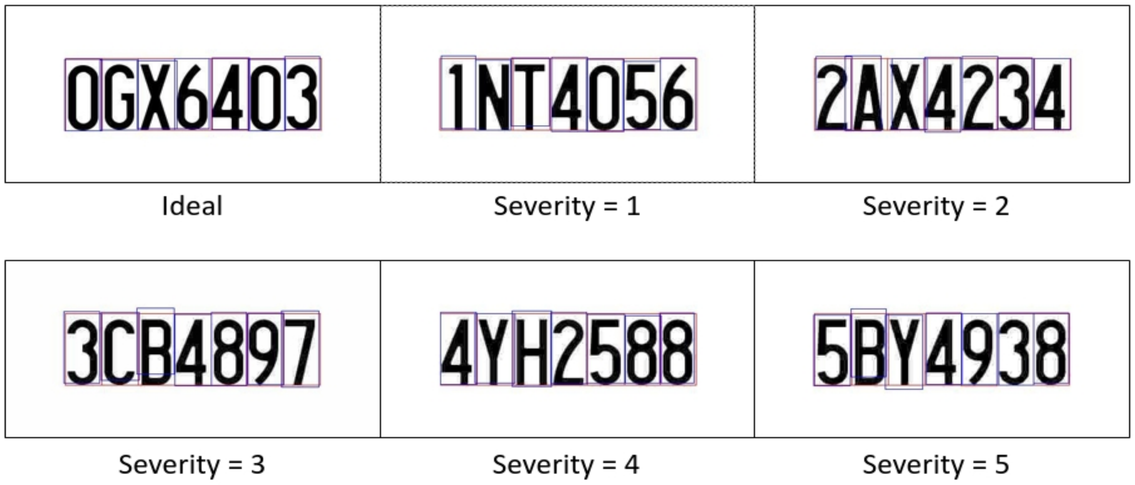


Figure 42: Test Images at a variety of Imgaug JPEG Compression severity levels. Red boxes are ground truth, blue boxes are detections.

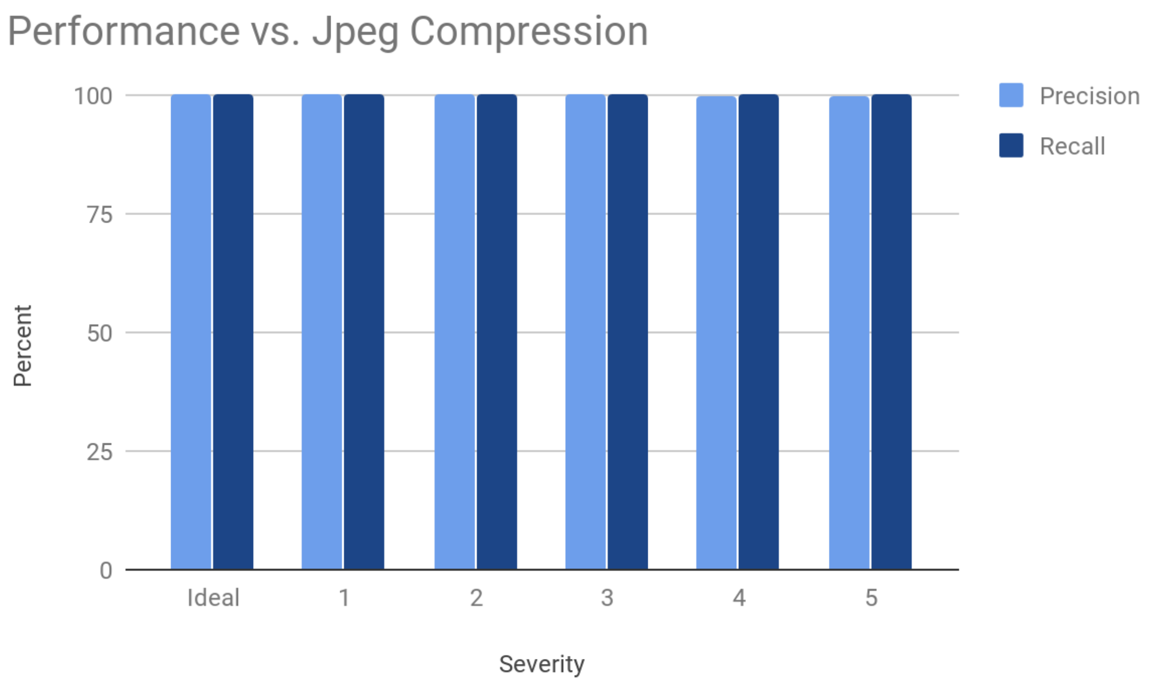


Figure 43: Plot showing performance at different JPEG Compression severity levels

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 values, 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 44 and performance with different kernel sizes is shown in Figure 45.

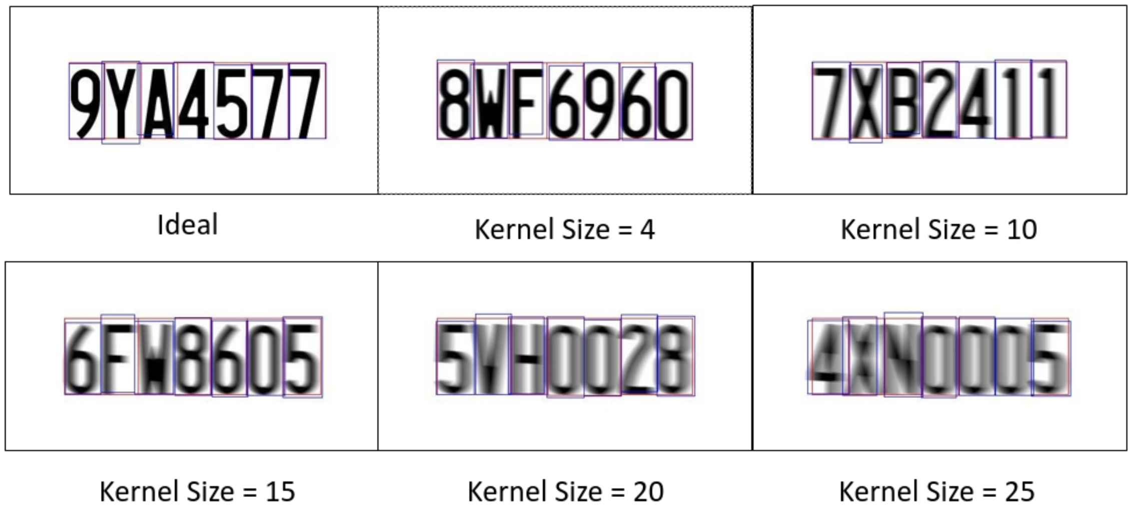


Figure 44: Sample images of license plates augmented with different kernel sizes. Red boxes are ground truth, blue boxes are detections.

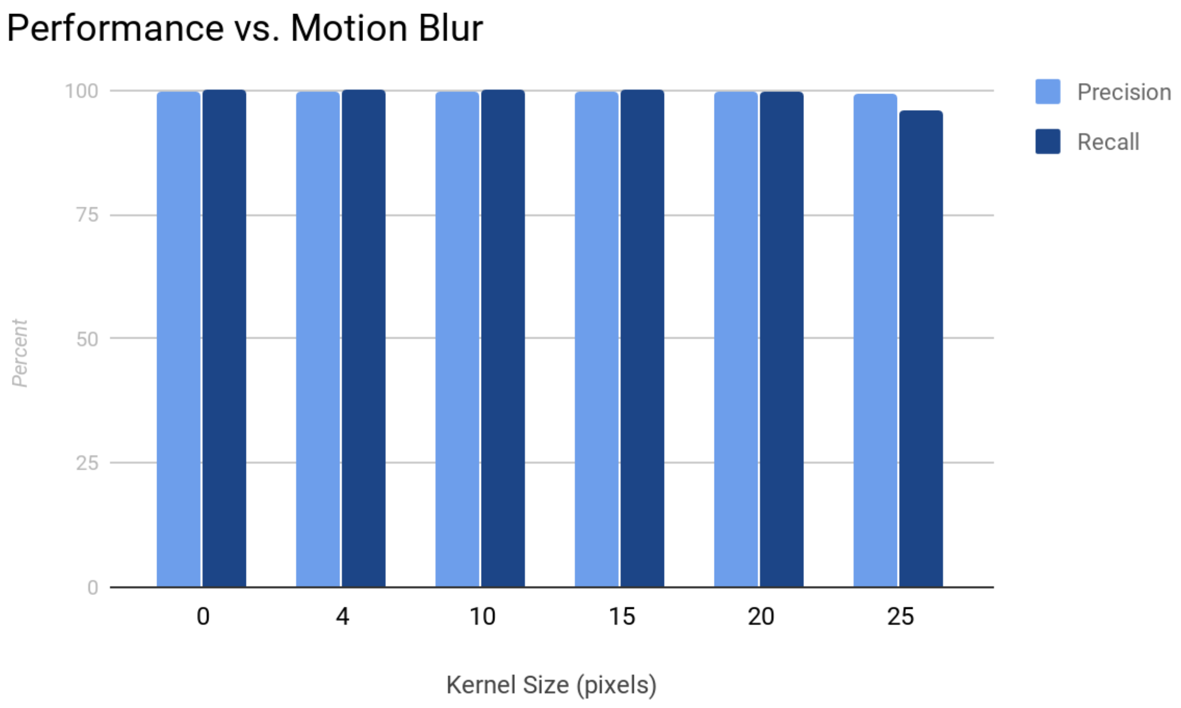


Figure 45: Plot of performance with test images of varying motion blur kernel sizes

3.2.4 Affine Transform

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

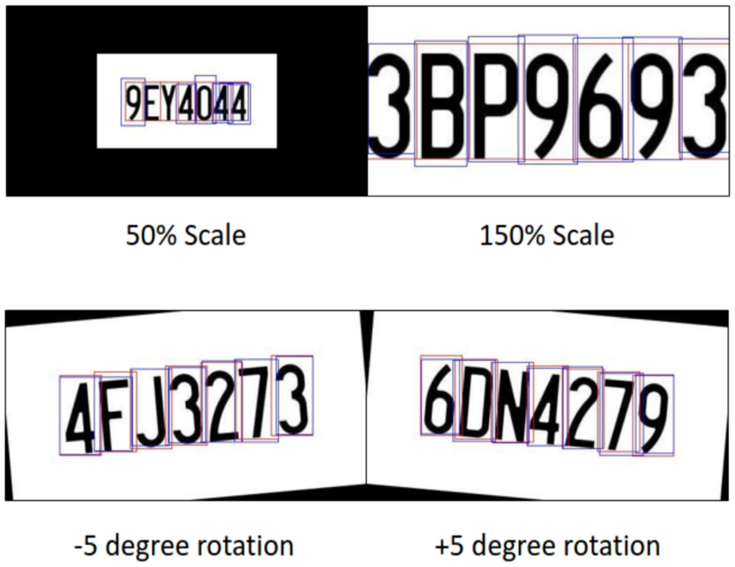


Figure 46: Sample affine transform augmented images. Red boxes are ground truth, blue boxes are detections.

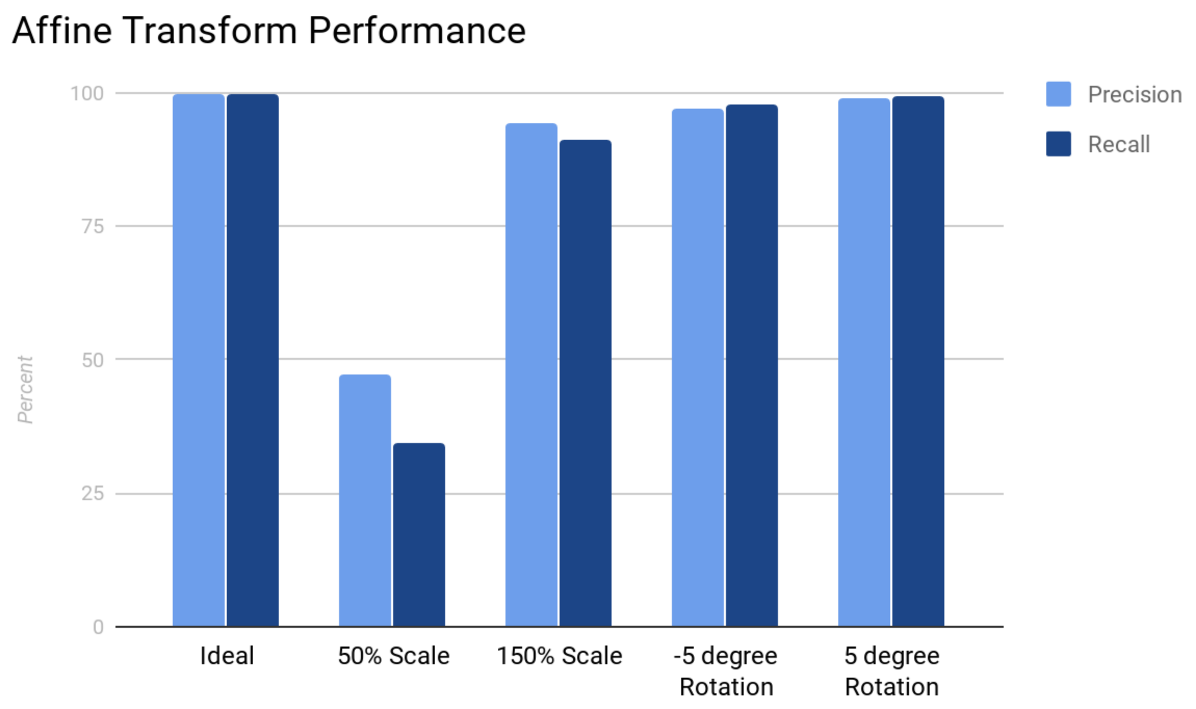


Figure 47: Plot of performance for different affine transform implemented on test images

Out of all data augmentation techniques, affine transform has the worst performance. The performance is particularly poor for the 50% scale images. This is somewhat surprising because the network had no issues when the images were made half size. 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, does not fit the problem space. Since the features of interest in the plate image are the same color as that of the border, it is possible that this is causing 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. Because only the corners are rotated, the bounding box lines are still drawn straight. If the whole box was rotated, rather than just the corners, better ground truth boxes for affine augmented images could be produced.

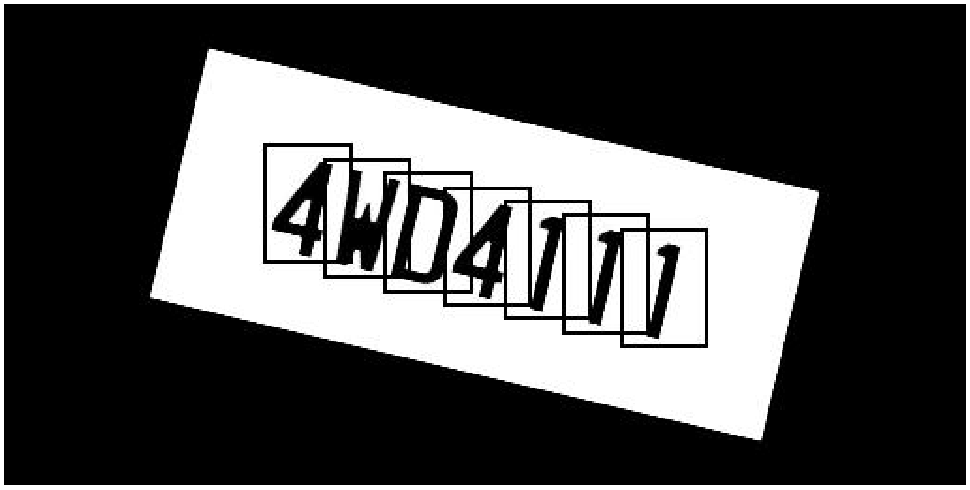


Figure 48: Bounding boxes at higher rotations

3.2.5 Gaussian Noise

The Faster R-CNN ResNet 50 network was trained on a Gaussian noise augmented dataset. This was the only data augmentation technique 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, against which the network was tested, are shown in Figure 49. Network performance is shown in Figure 50. Severity levels 1-5 of Gaussian noise correspond to random additive noise scaled by the magnitudes 0.08, 0.12, 0.18, 0.26, and 0.38, respectively.

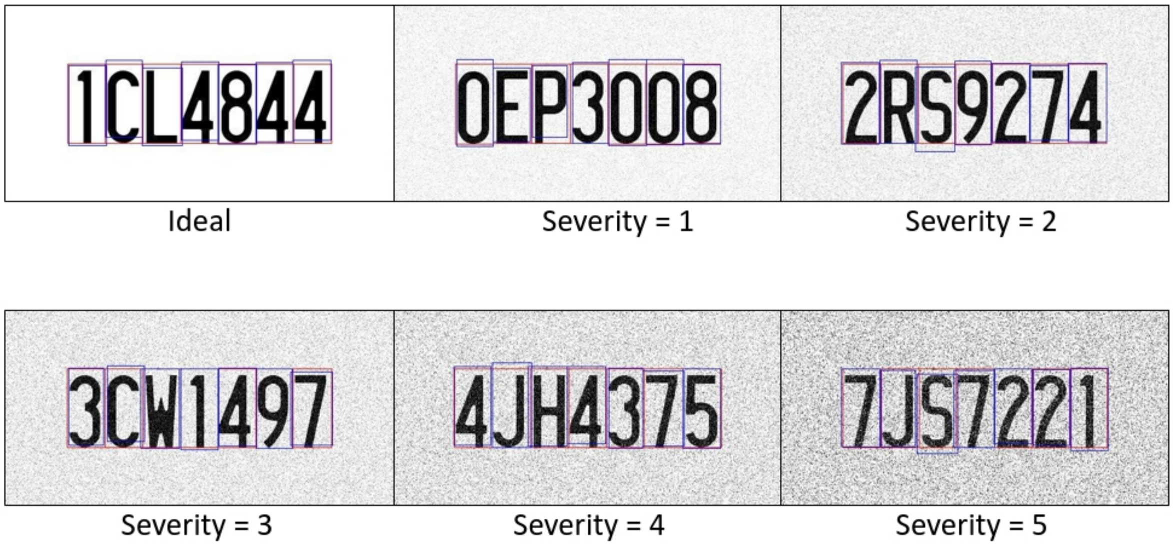


Figure 49: Sample Gaussian noise augmented images at various severity levels. Red boxes are ground truth, blue boxes are detections.

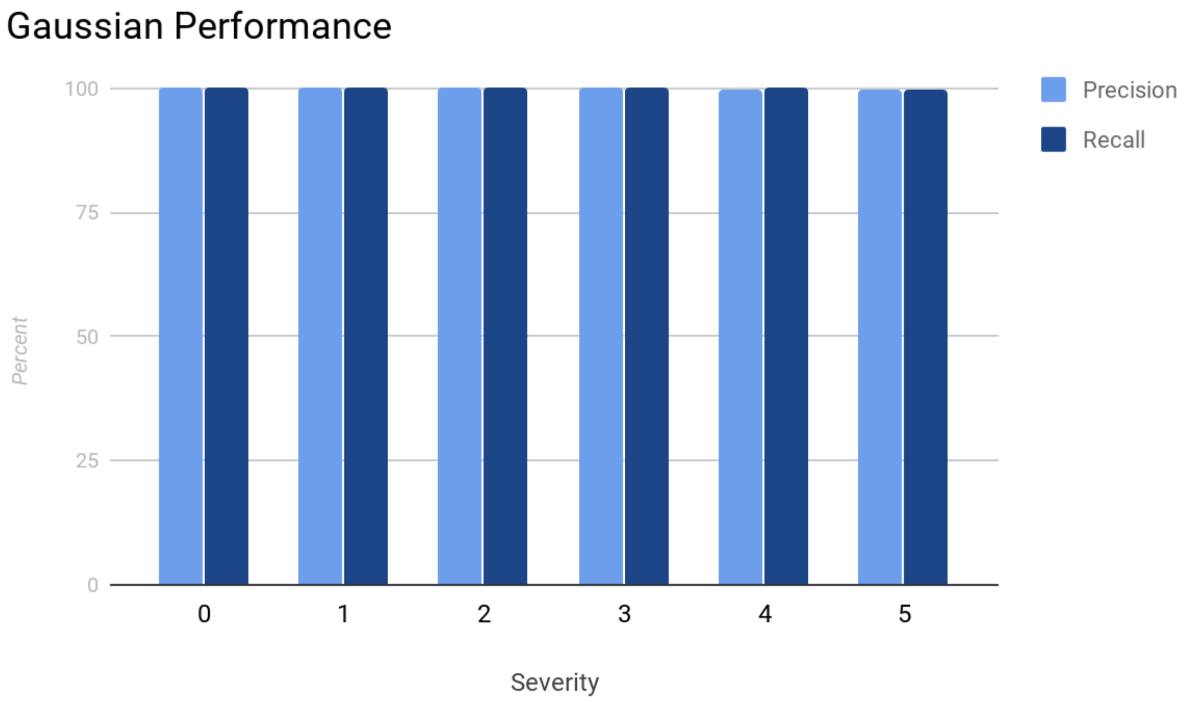


Figure 50: Plot of performance for test image at different levels of Gaussian noise

3.3 Conclusions

In this thesis, a Deep Learning framework has been developed on MARCC to perform character recognition on synthetically generated license plate images of varying qualities. The performance of a suite of Deep Learning architectures and object detection approaches was evaluated using the plate images. Initially, all architectures and approaches were evaluated with non-degraded license plate images. The results indicate that the Faster R-CNN and related R-FCN approaches, along with the ResNet architecture, at varying layer depths, have the best performance for license plate character recognition problem. The network used for all further evaluations was the Faster R-CNN ResNet 50 network. In addition, it was shown that for robust performance, the lower bound of the ideal (no degradation) license plate image size was 32 x 16 pixels.

The Faster R-CNN ResNet 50 network was trained and tested on degraded license plate images. The network was robust to image degradation due to Gaussian noise, JPEG compression, or motion blur. Performance was slightly worse at the low end of the tested quality spectrum for these augmentation techniques. The application of affine transforms caused larger issues and revealed holes in the overall framework. An issue uncovered with the affine transform was the limit of the Imgaug library. The Imgaug library has a built-in ground truth bounding box transform. This bounding box transform performs poorly for this dataset at high rotations (above 5 degrees). A potential remedy for the poor performance would be to define a custom bounding box transform. The design of a new image augmentor was outside the scope of this thesis but would be a future improvement. The fifty percent scaling affine transform case had the worst performance of any data augmentation technique applied. Compared to the half-size images evaluated when the fifty percent scaling data augmentation technique is applied, the image itself does not become smaller rather it is padded. The network did not handle this padding well. The poor performance on the fifty percent scaled images could potentially be that the padding is the same pixel value (black) as the characters.

Due to limitations of scope, this thesis is far from a comprehensive analysis of license plate character recognition on low-quality images. There is much additional work that could be done. A custom bounding box transform could be developed to allow for accurate ground truth information when applying data augmentation techniques that require modifications to ground truth. There are also numerous data augmentation techniques which are relevant to this dataset that were not applied and tested due to scope limitations. Table 2 lists some of these in brief. A fully developed system for performing character recognition on license plates would need to be robust to many more variations of degradation, as well as robust to the potential for multiple methods of distortion introduced in one image. This thesis focused on using license plate images with a white background as a baseline for performance. Future analysis on license plates with the traditional image background would be useful and would open the door for additional color-dependent data augmentation techniques 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. There is much work between this system and one that could be deployed in digital forensics, but this thesis shows that a Deep Learning-based approach has potential 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

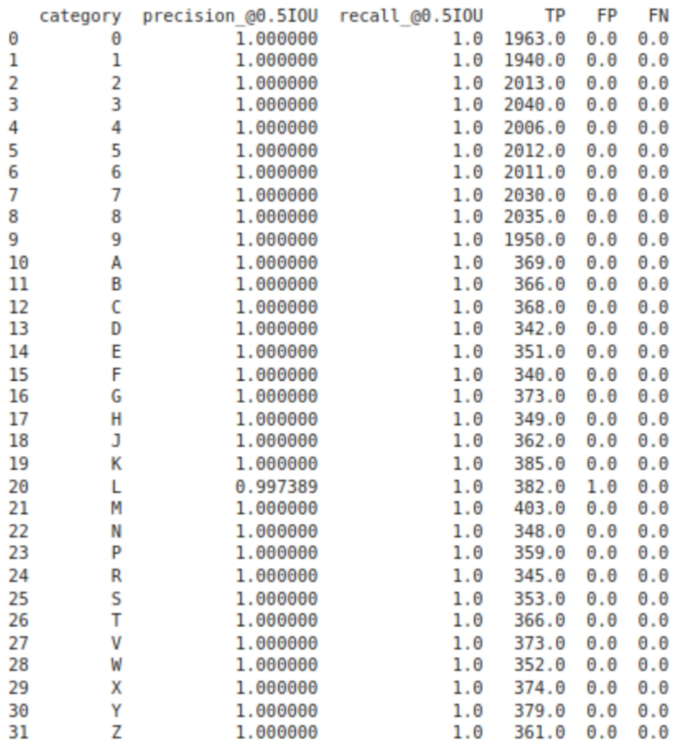
This appendix contains the available confusion matrices and tabular results for test runs on datasets with augmentation techniques applied.

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

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”.

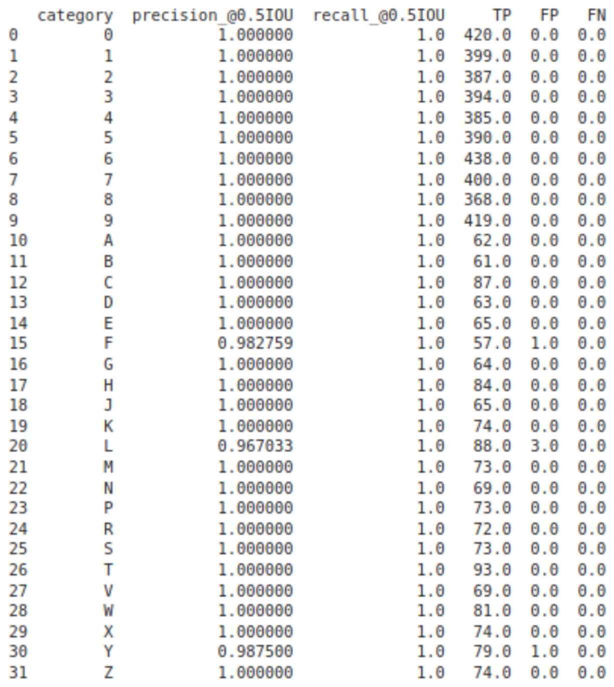


```
   category  precision_@0.5IOU  recall_@0.5IOU      TP   FP   FN
0         0           1.000000             1.0  1963.0  0.0  0.0
1         1           1.000000             1.0  1940.0  0.0  0.0
2         2           1.000000             1.0  2013.0  0.0  0.0
3         3           1.000000             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

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”.

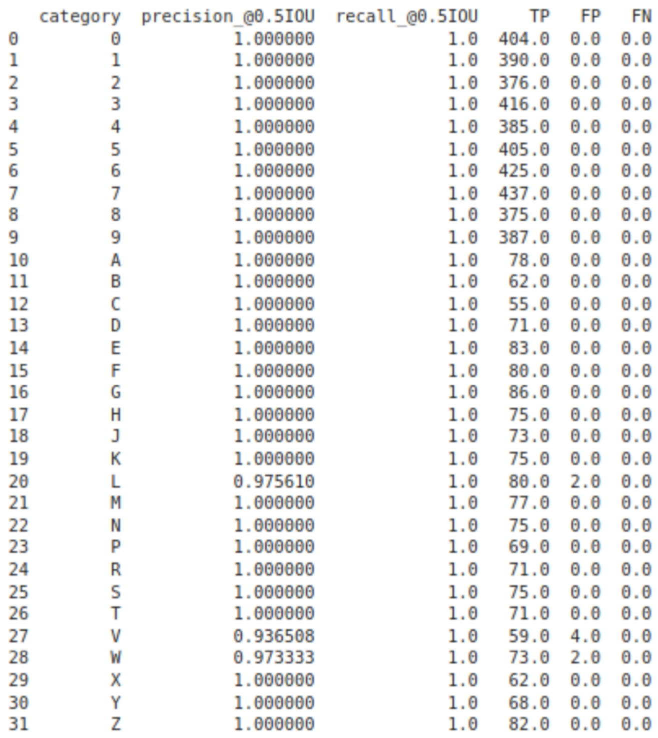


```
    category  precision_@0.5IOU  recall_@0.5IOU     TP   FP   FN
0          0           1.000000             1.0  420.0  0.0  0.0
1          1           1.000000             1.0  399.0  0.0  0.0
2          2           1.000000             1.0  387.0  0.0  0.0
3          3           1.000000             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

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”.

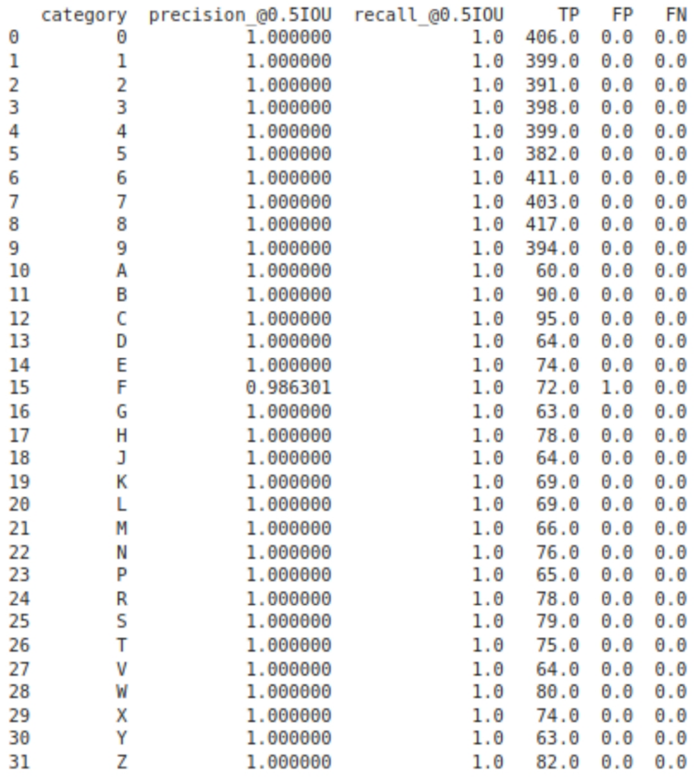


```
   category  precision_@0.5IOU  recall_@0.5IOU     TP   FP   FN
0         0           1.000000             1.0  404.0  0.0  0.0
1         1           1.000000             1.0  390.0  0.0  0.0
2         2           1.000000             1.0  376.0  0.0  0.0
3         3           1.000000             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

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”.

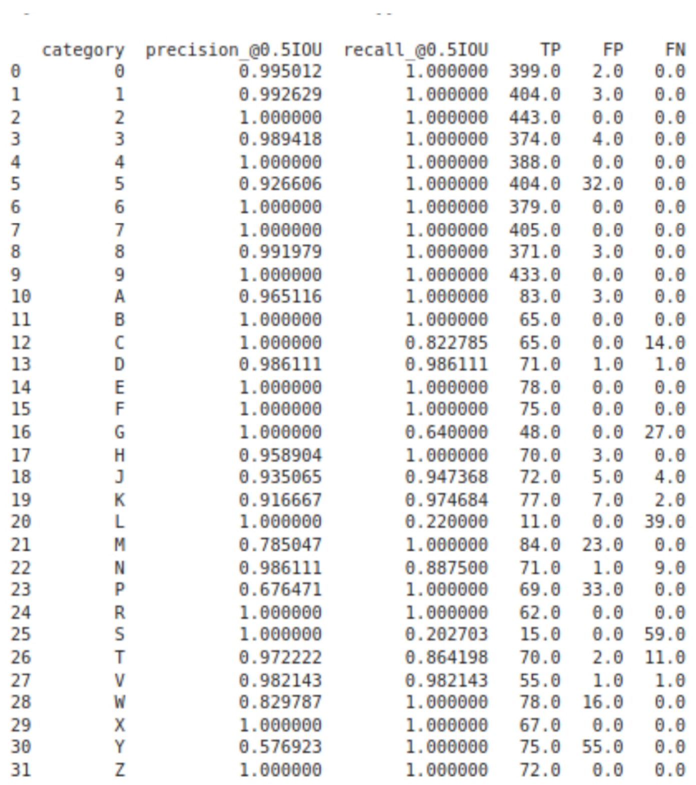


```
   category  precision_@0.5IOU  recall_@0.5IOU     TP   FP   FN
0         0           1.000000             1.0  406.0  0.0  0.0
1         1           1.000000             1.0  399.0  0.0  0.0
2         2           1.000000             1.0  391.0  0.0  0.0
3         3           1.000000             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

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 at which 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 technique 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.

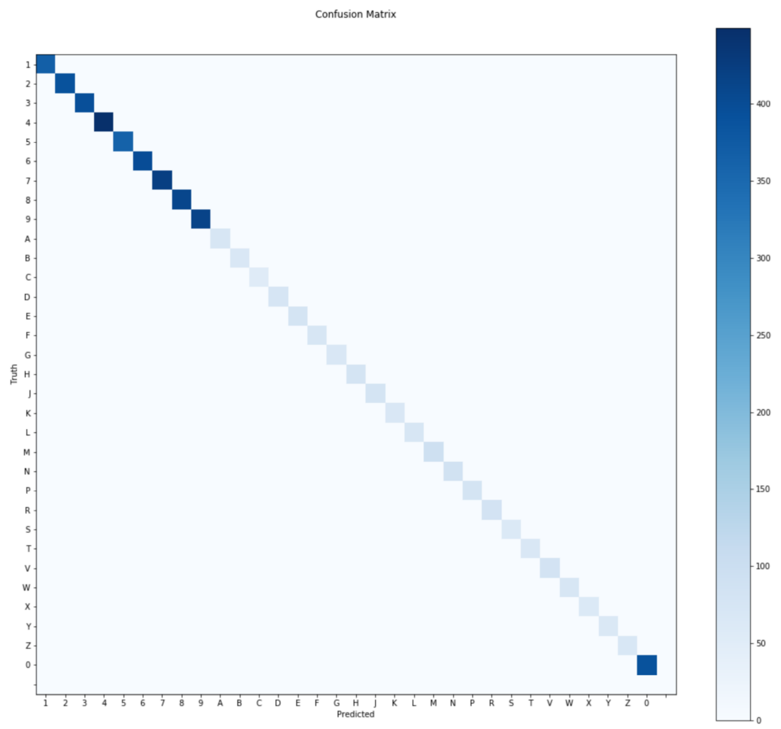
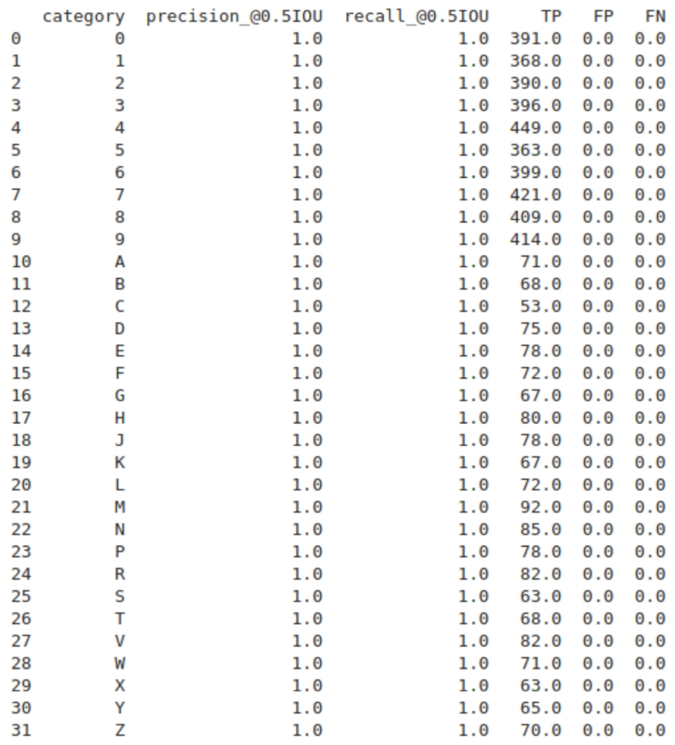


Figure 56: Confusion Matrix for Faster R-CNN ResNet 50 on images with no JPEG Compression applied



```
   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%.

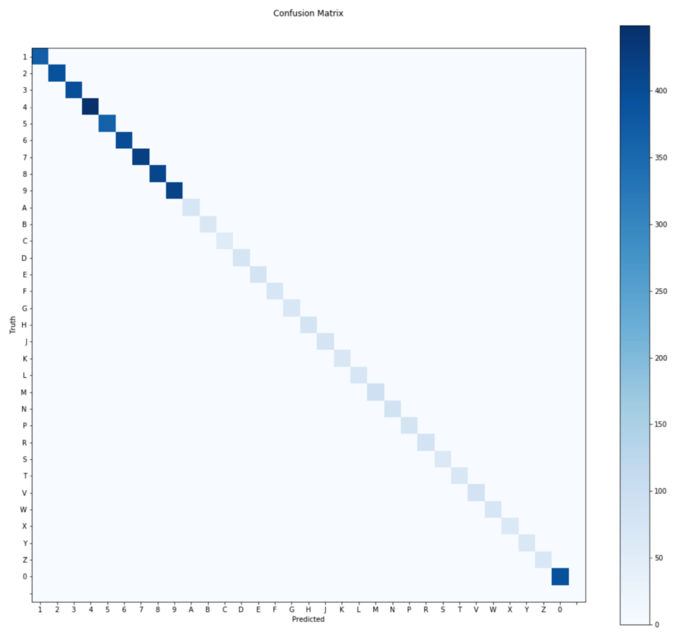
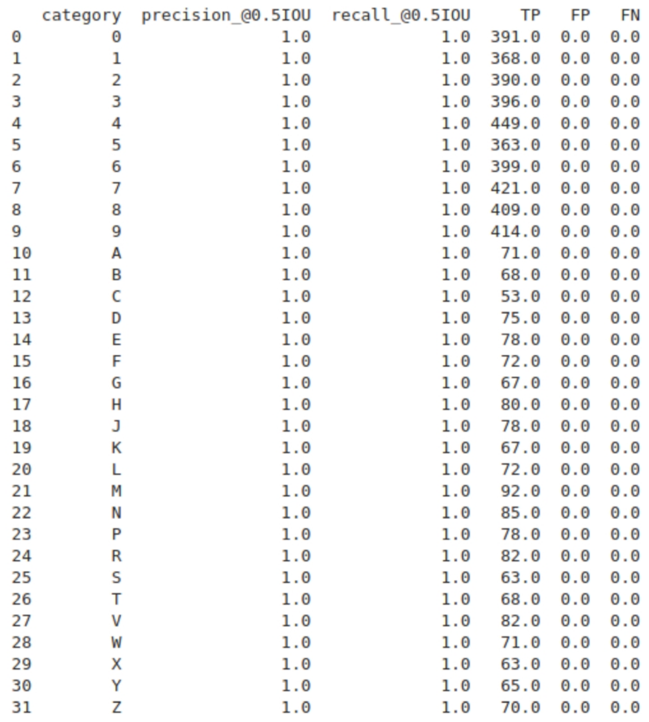


Figure 58: Confusion Matrix for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 1 (Quality 25)



```
   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%.

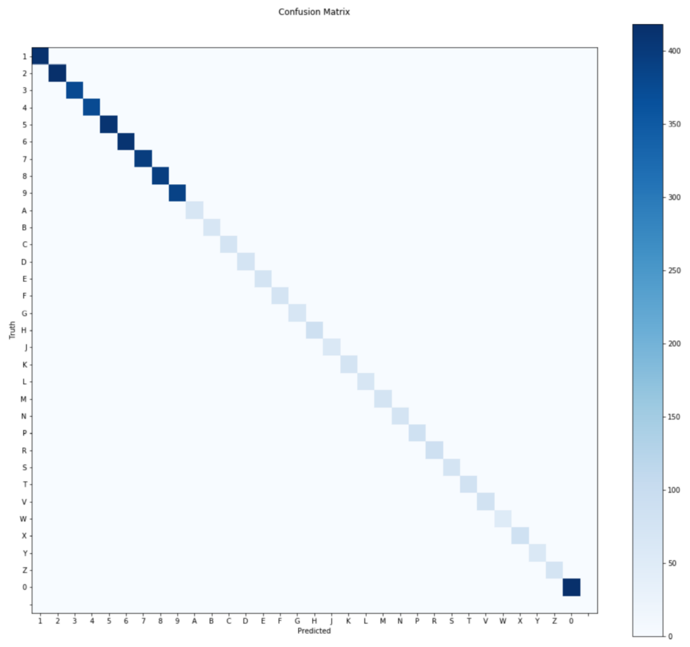
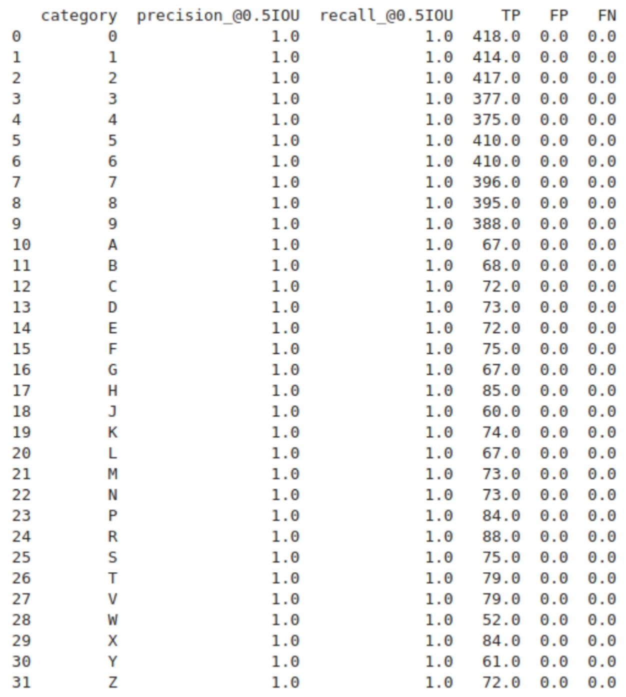


Figure 60: Confusion Matrix for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 2 (Quality 18)



```
   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%.

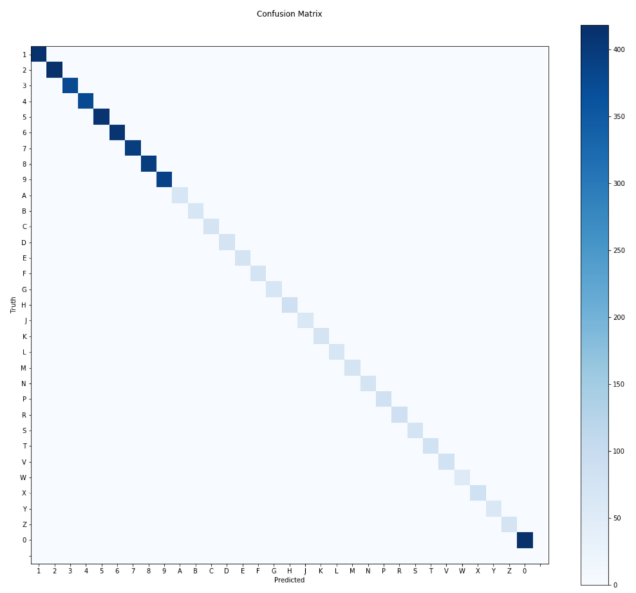
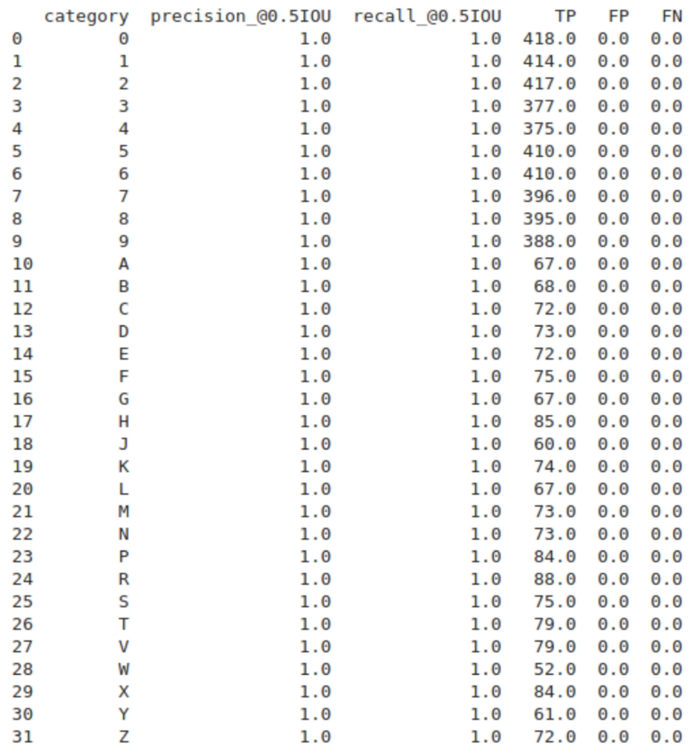


Figure 62: Confusion Matrix for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 3 (Quality 15)



```
    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”.

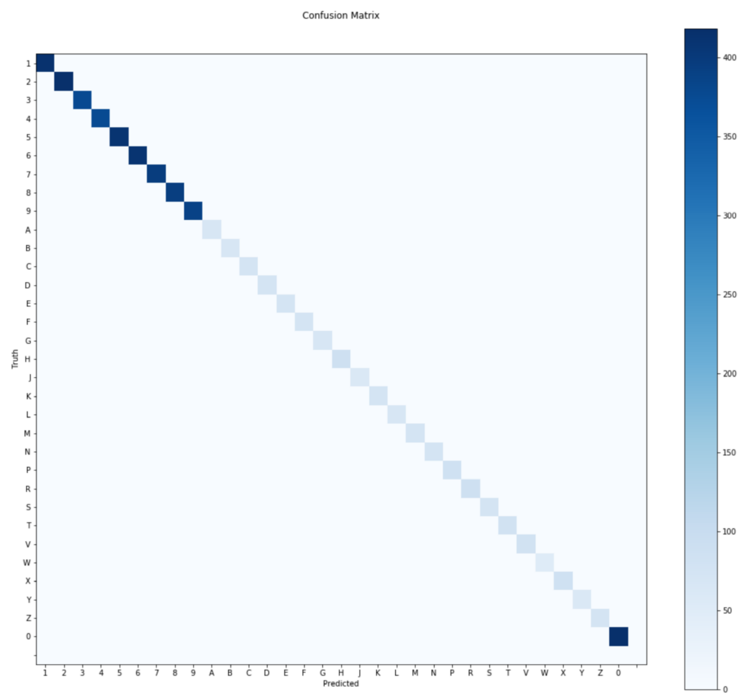
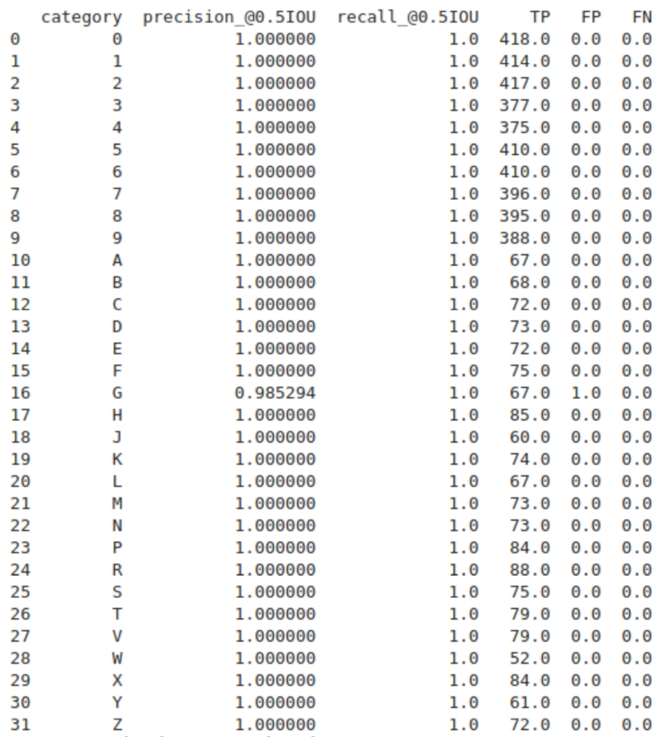


Figure 64: Confusion Matrix for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 4 (Quality 10)



```
   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 66 and Figure 67 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”.

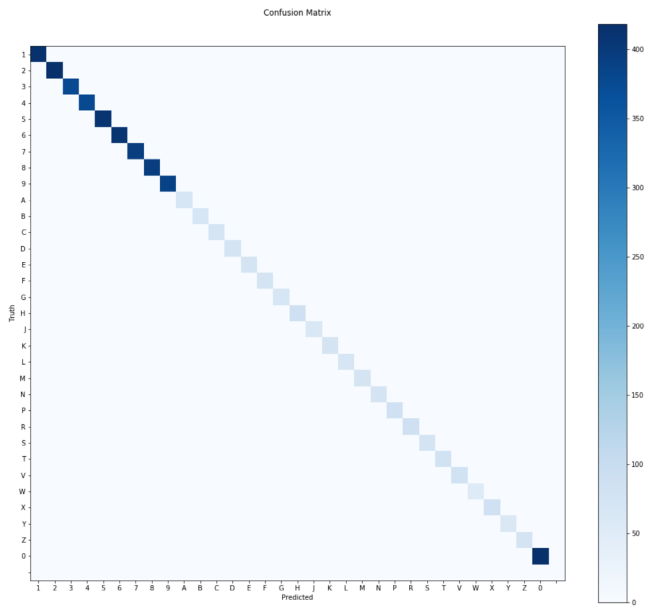
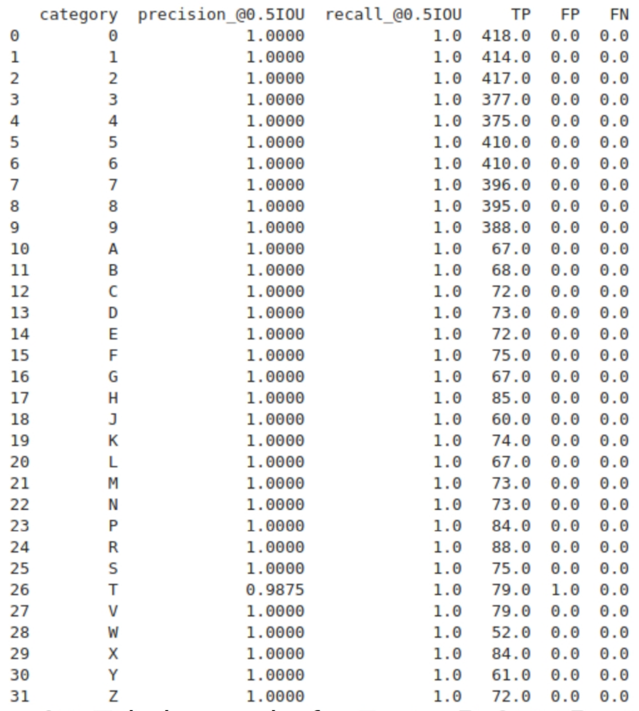


Figure 66: Confusion Matrix for Faster R-CNN ResNet 50 on images with JPEG Compression of Severity 5 (Quality 7)



```
   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 technique 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 camera's field of view 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".

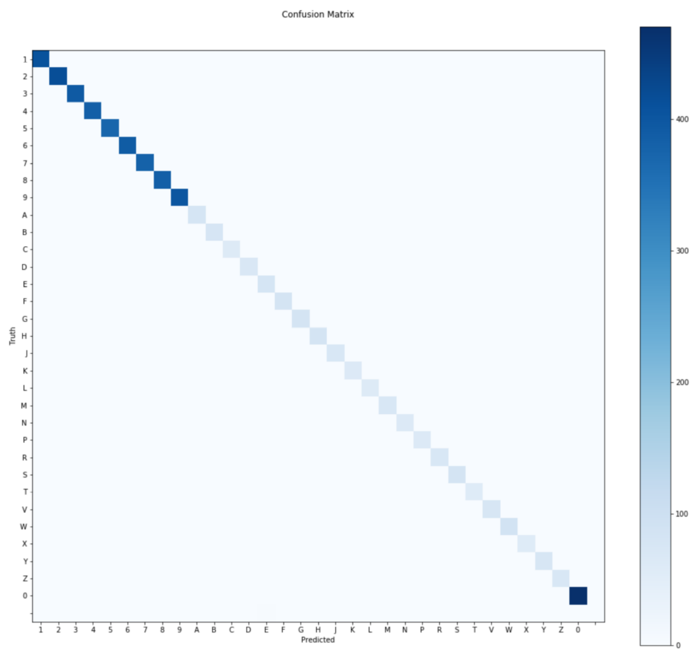
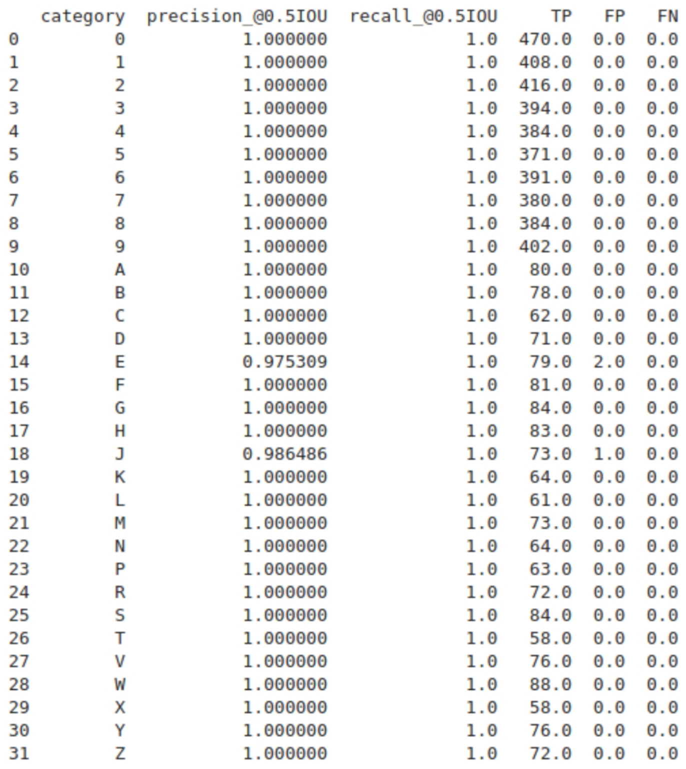


Figure 68: Confusion Matrix for Faster R-CNN ResNet 50 on images with no motion blur



```
   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”.

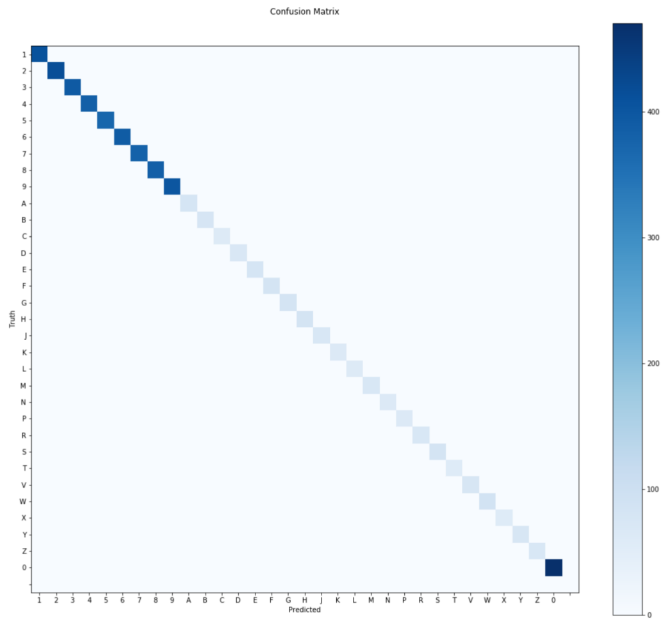
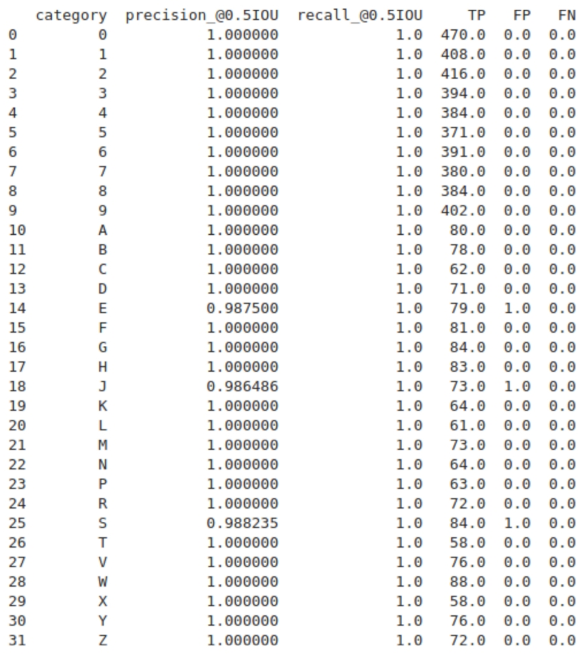


Figure 70: Confusion Matrix for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 4 pixels



```
   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”.

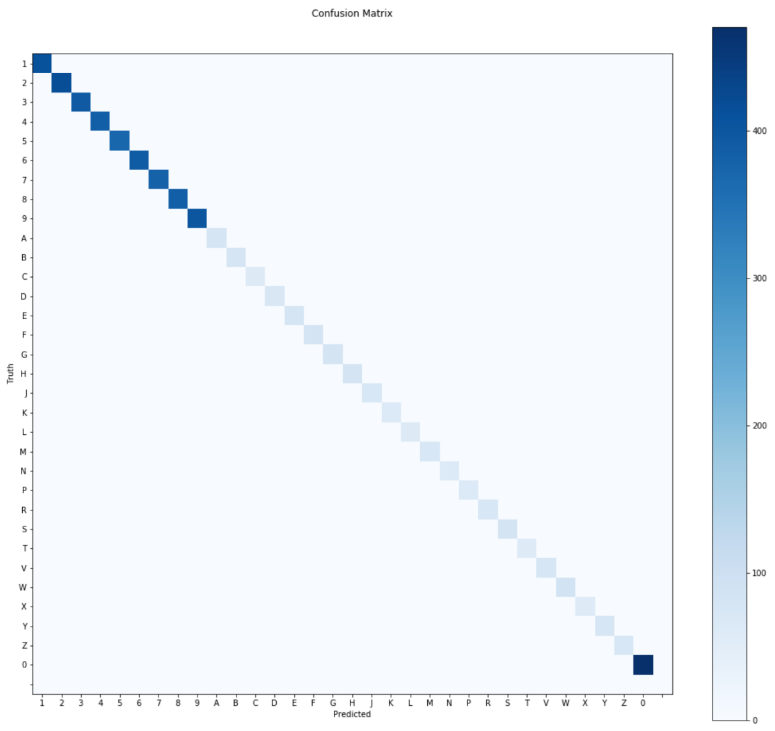
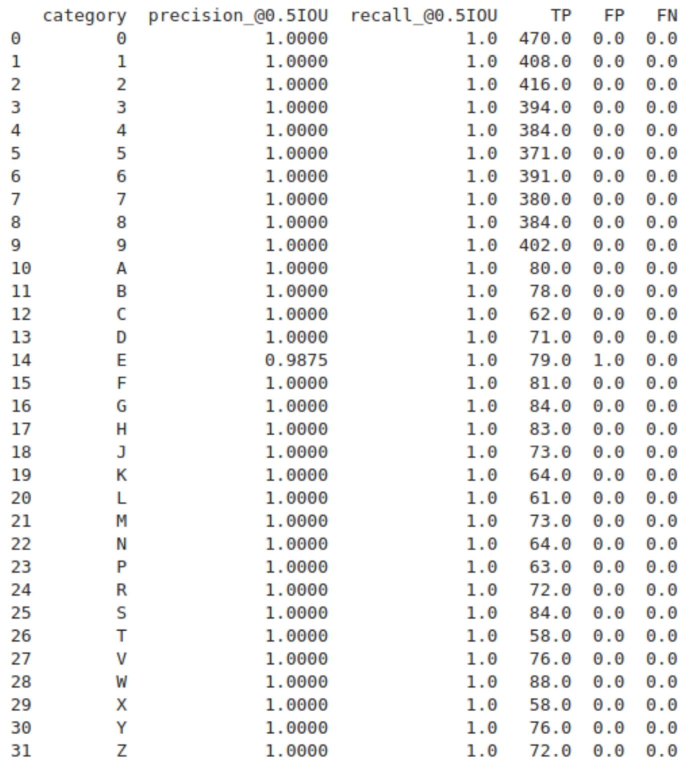


Figure 72: Confusion Matrix for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 10 pixels



```
   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”.

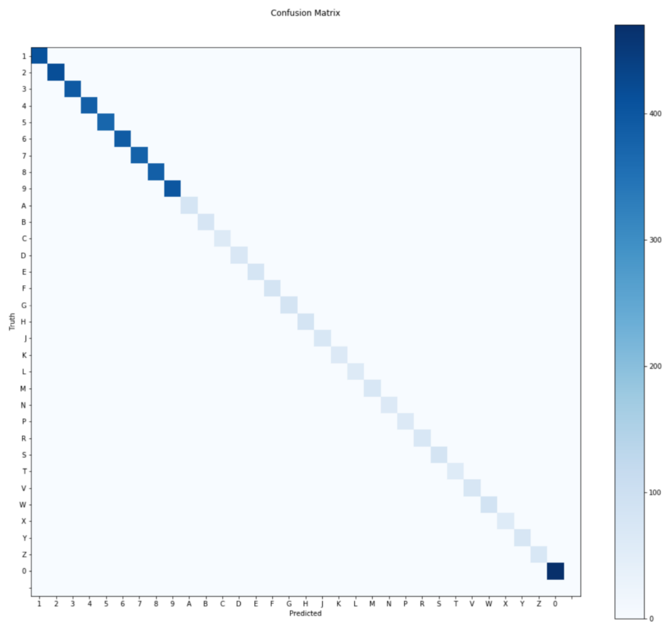
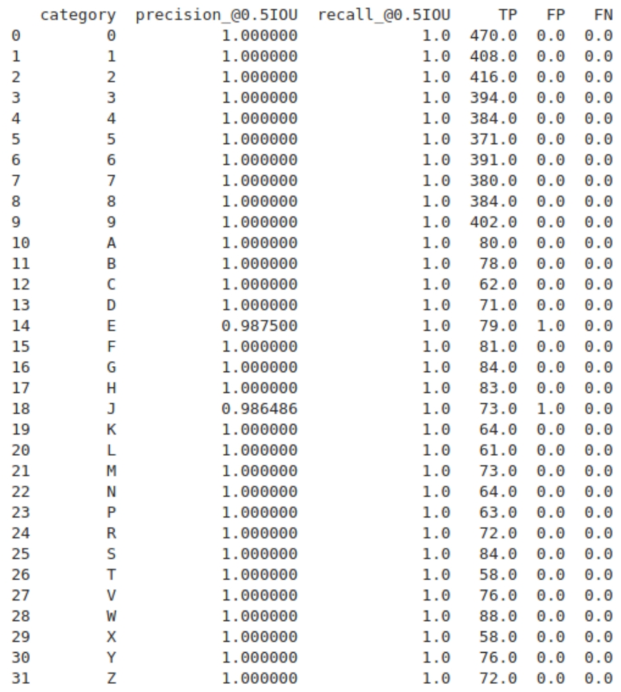


Figure 74: Confusion Matrix for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 15 pixels



```
   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”.

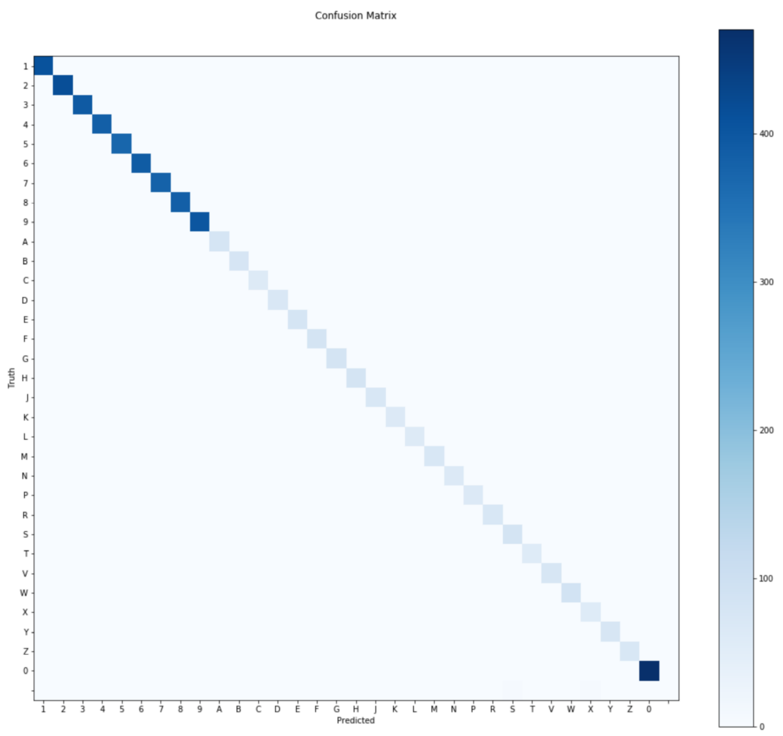
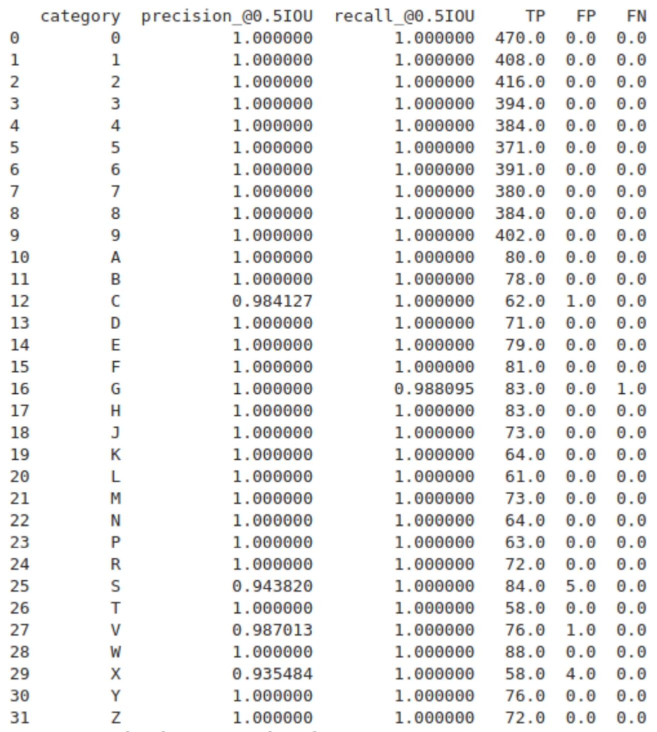


Figure 76: Confusion Matrix for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 20 pixels



```
    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 25 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.

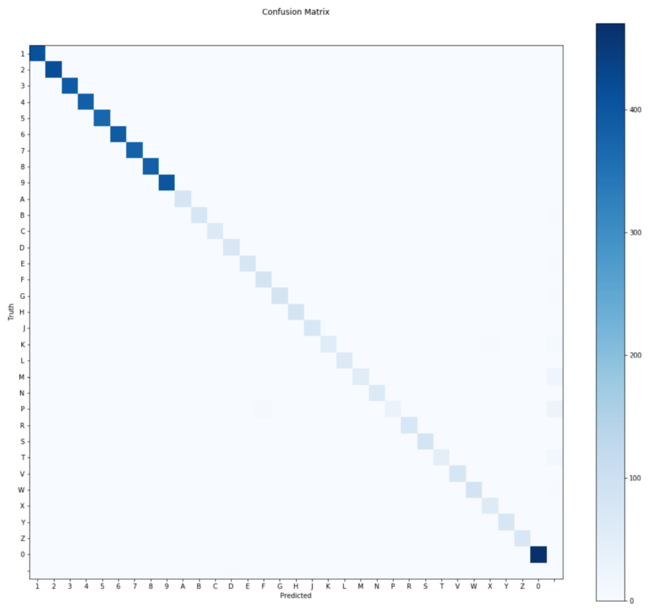
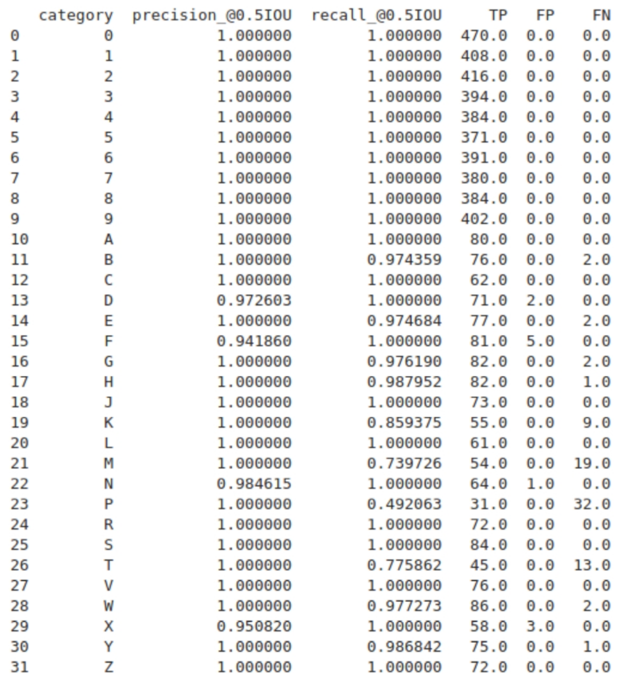


Figure 78: Confusion Matrix for Faster R-CNN ResNet 50 on images with a motion blur with a kernel size of 25 pixels



```
   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 a data augmentation technique, 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 augmentation techniques, this has the worst performance. This is somewhat surprising because the network had no issue when the images were made half size. 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, does not fit the problem space. Since the features of interest in the plate image are the same color as that of the border it is possible that this is causing part of the issue.

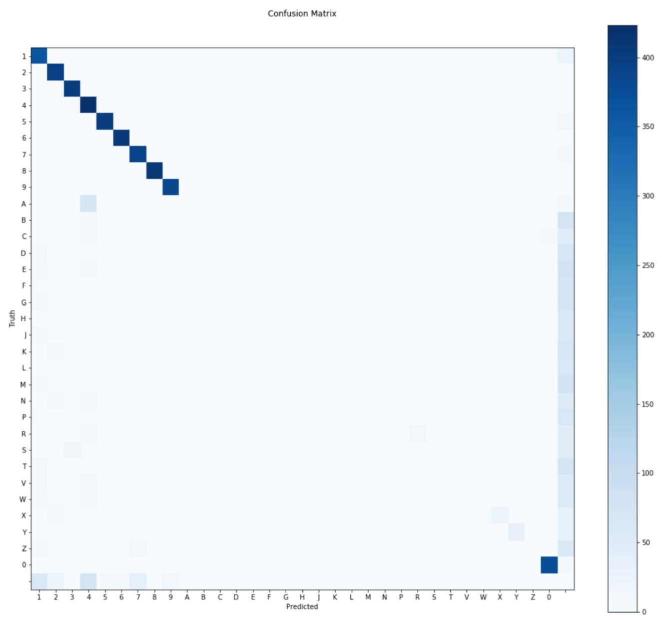
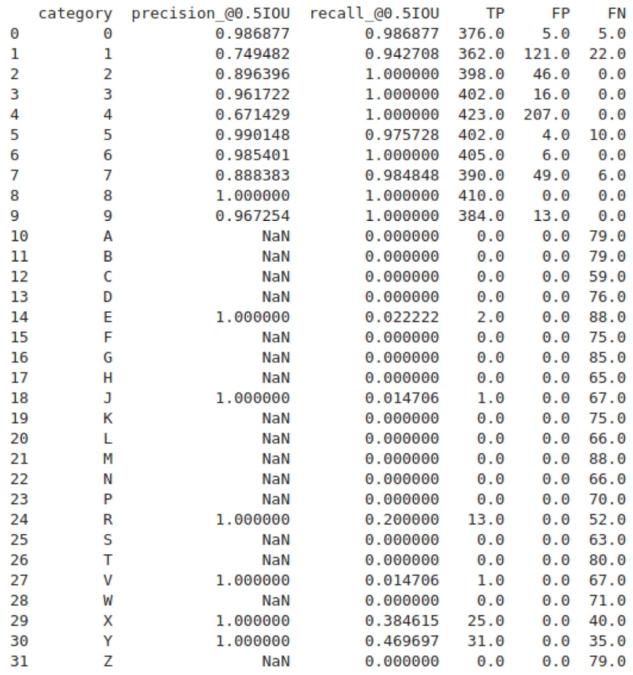


Figure 80: Confusion Matrix for Faster R-CNN ResNet 50 on images at 50% scale



```
   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 83 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.

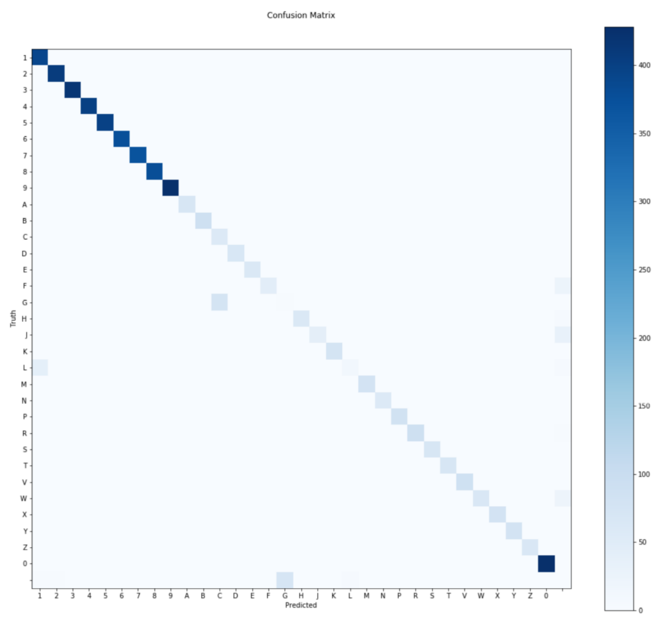


Figure 82: Confusion Matrix for Faster R-CNN ResNet 50 on images at 150% scale

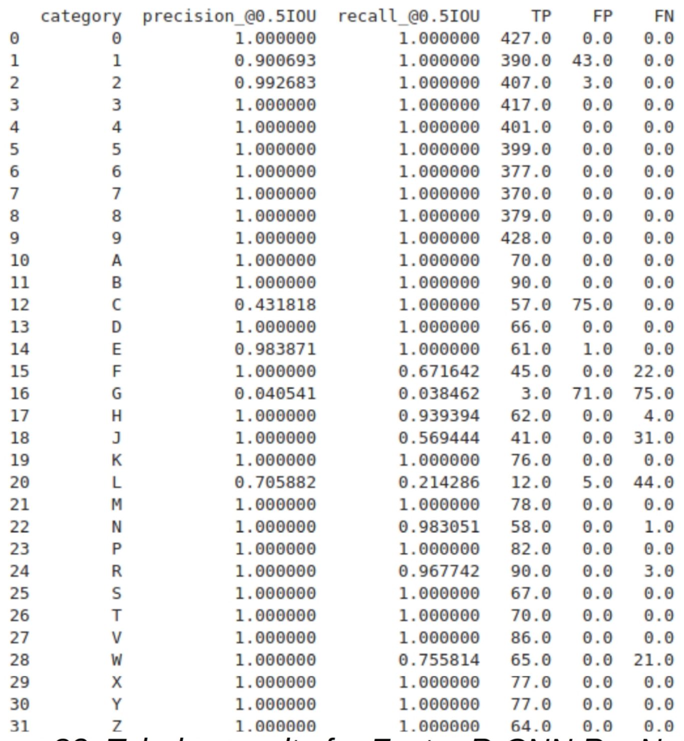

```
   category  precision_@0.5IOU  recall_@0.5IOU     TP    FP    FN
0         0           1.000000        1.000000  427.0   0.0   0.0
1         1           0.900693        1.000000  390.0  43.0   0.0
2         2           0.992683        1.000000  407.0   3.0   0.0
3         3           1.000000        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 not as good as the other data augmentation techniques, were 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.

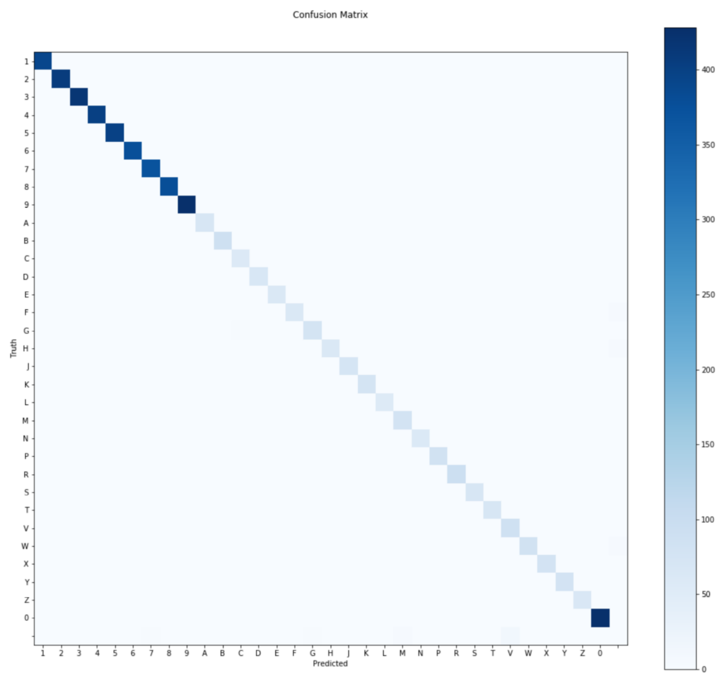
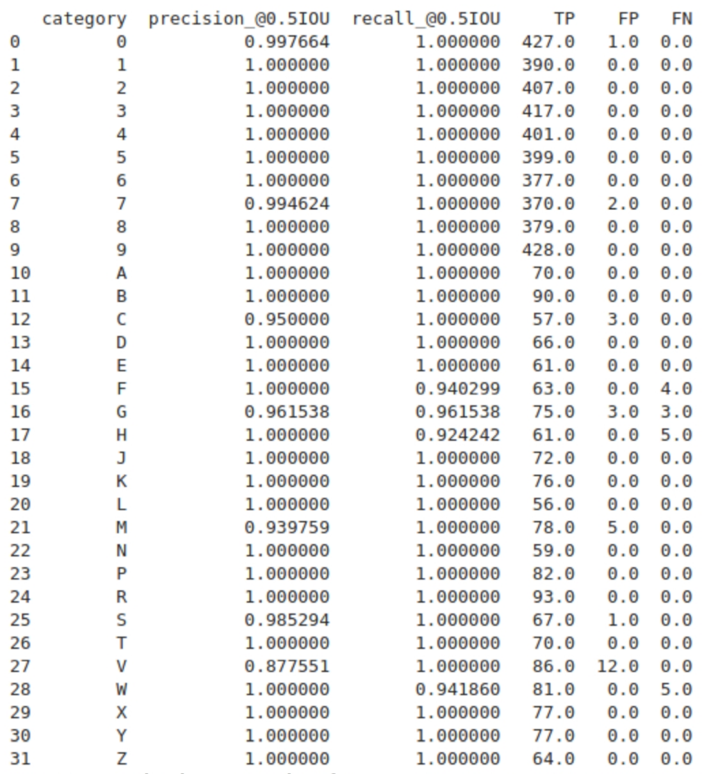


Figure 84: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated +5 degrees



```
   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 not as good as the other data augmentation techniques, were 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.

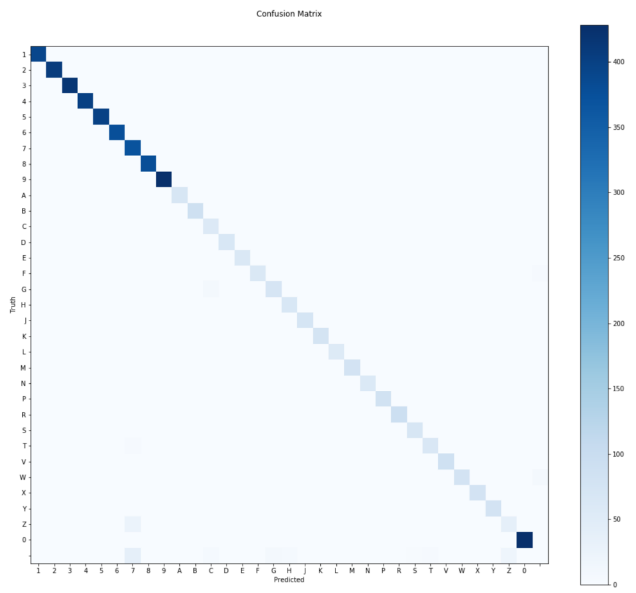


Figure 86: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated -5 degrees

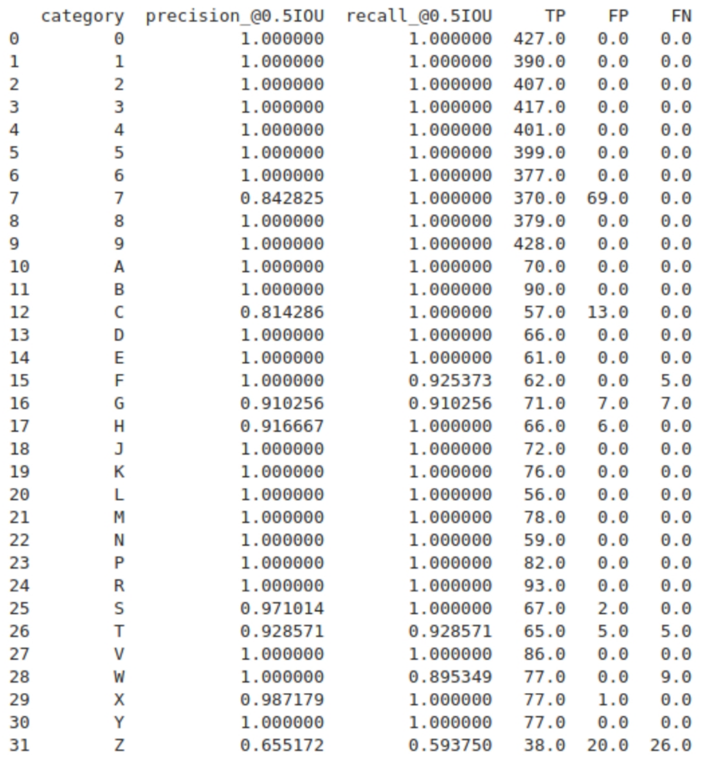

```
   category  precision_@0.5IOU  recall_@0.5IOU     TP    FP    FN
0         0           1.000000        1.000000  427.0   0.0   0.0
1         1           1.000000        1.000000  390.0   0.0   0.0
2         2           1.000000        1.000000  407.0   0.0   0.0
3         3           1.000000        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 technique 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%.

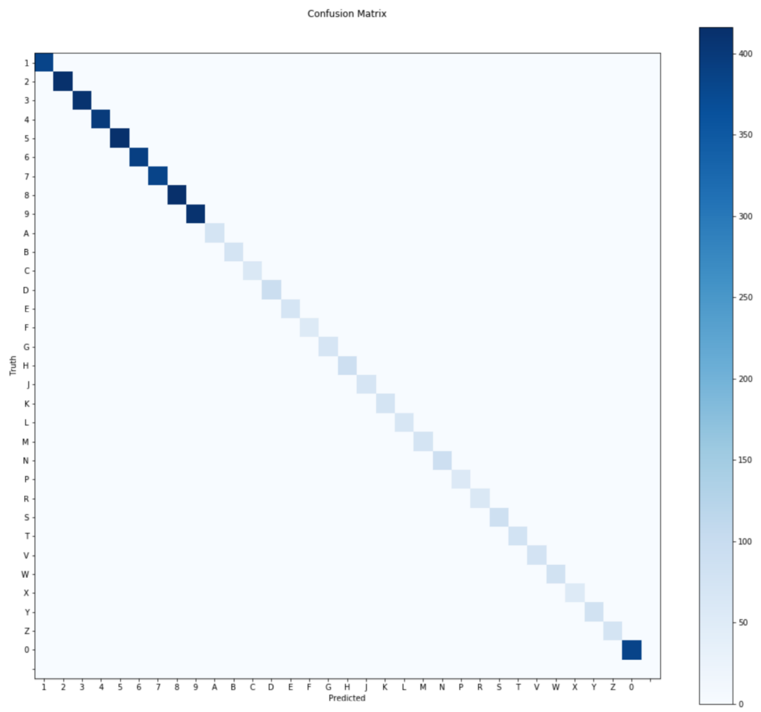
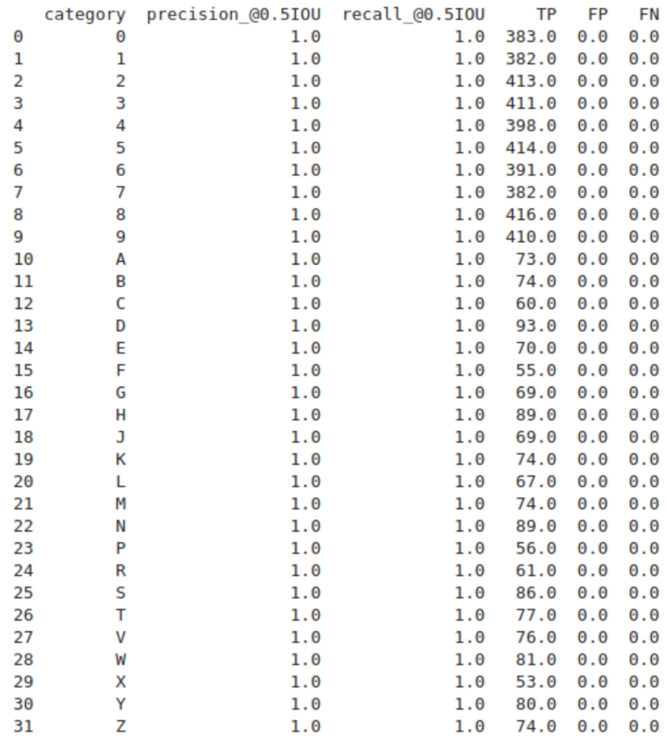


Figure 88: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with no Gaussian noise



```
   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%.

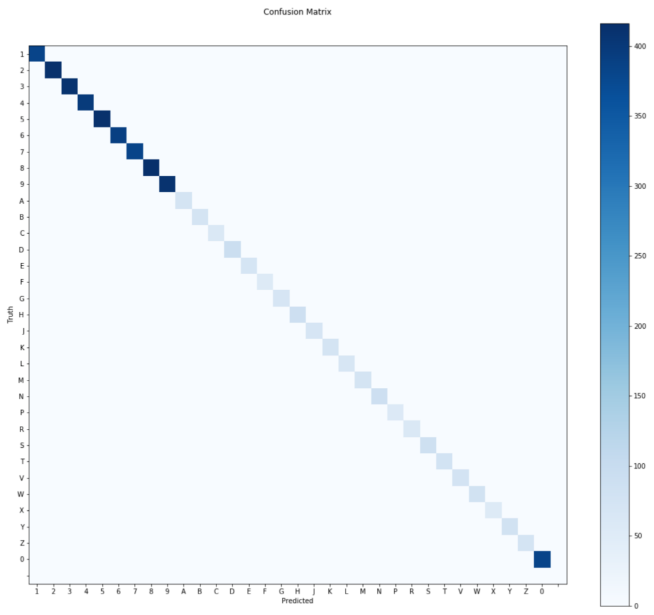
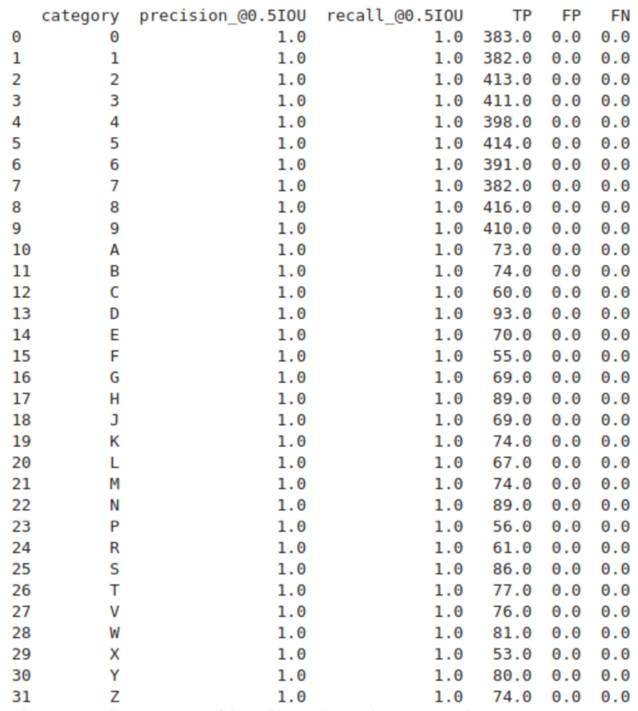


Figure 90: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 1



```
    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%.

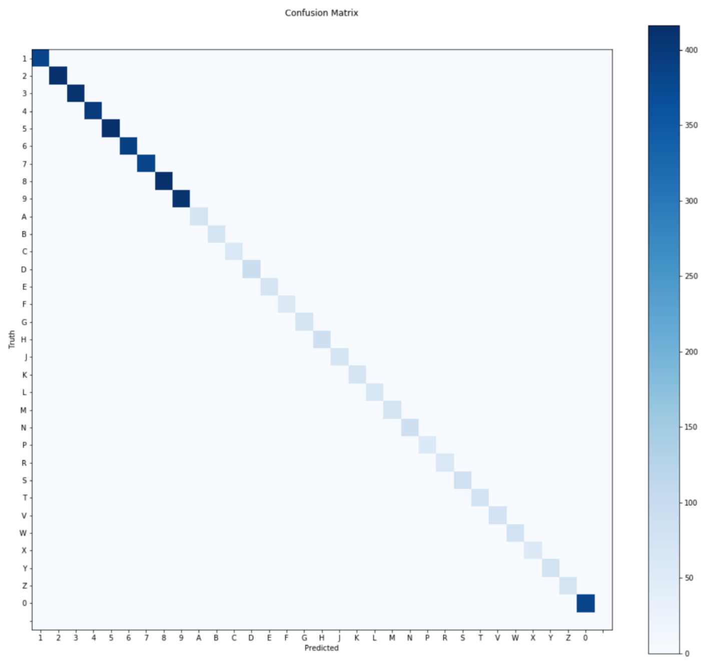
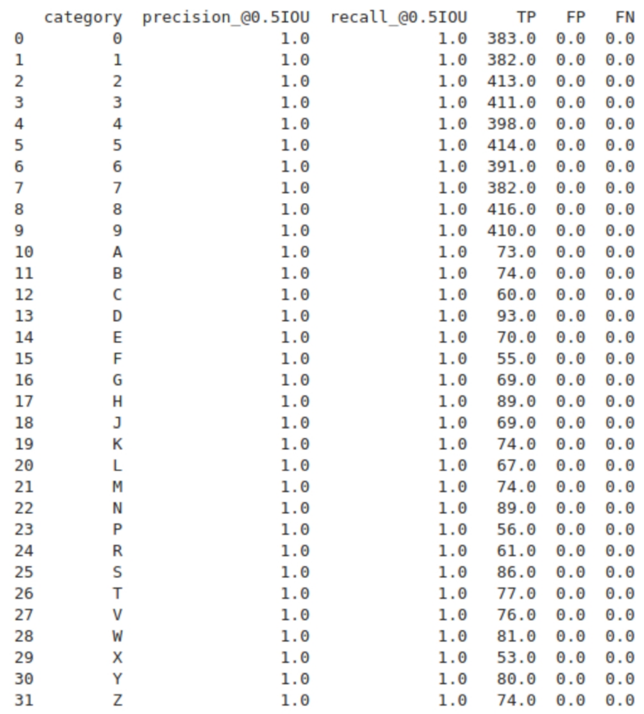


Figure 92: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 2



```
   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%.

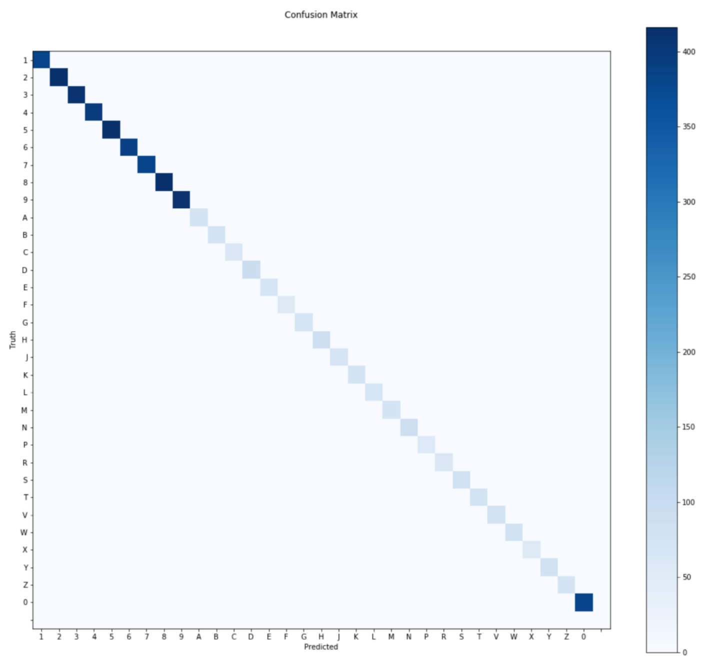
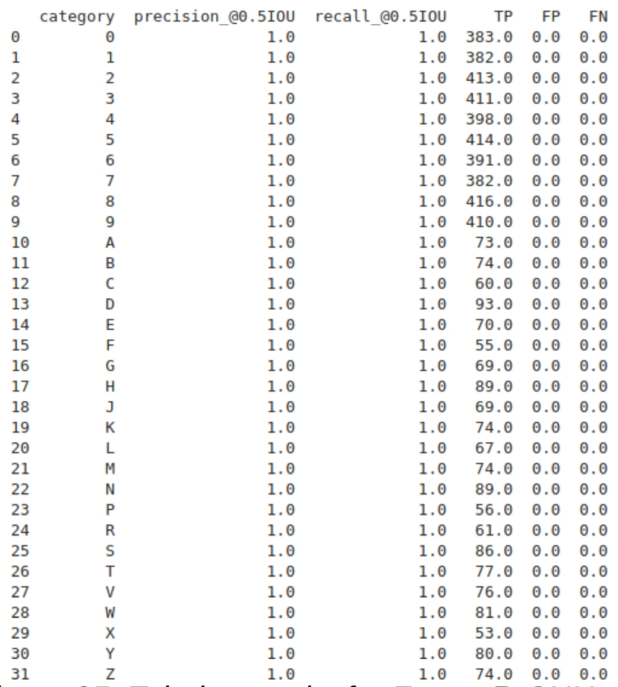


Figure 94: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 3



```
    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".

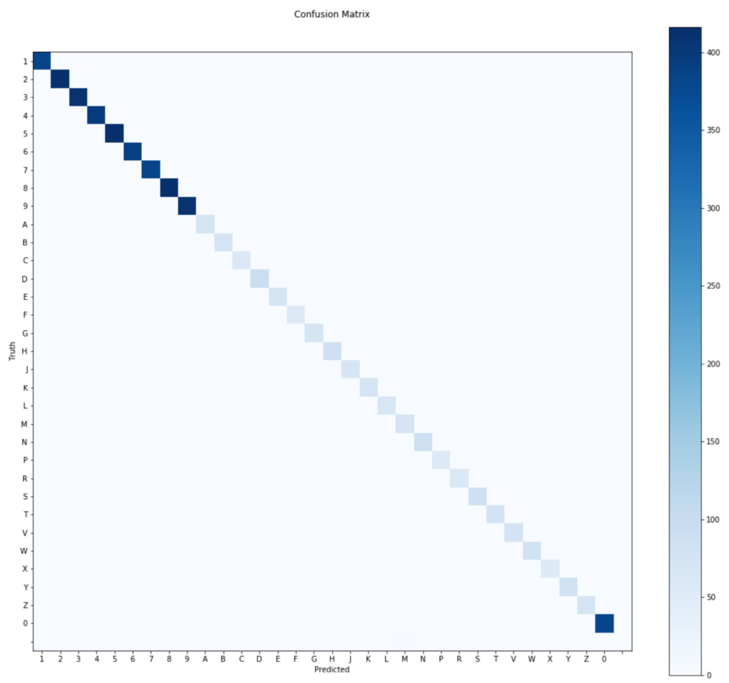
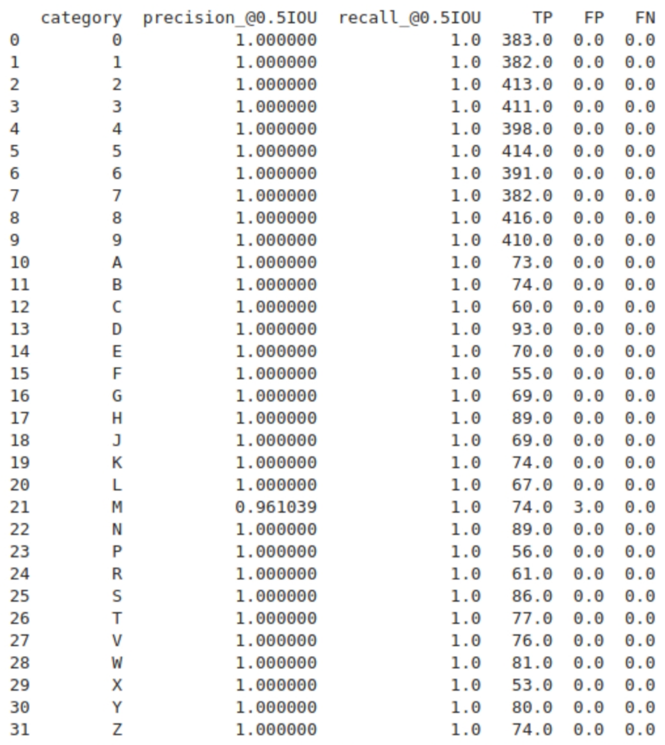


Figure 96: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 4



```
   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”.

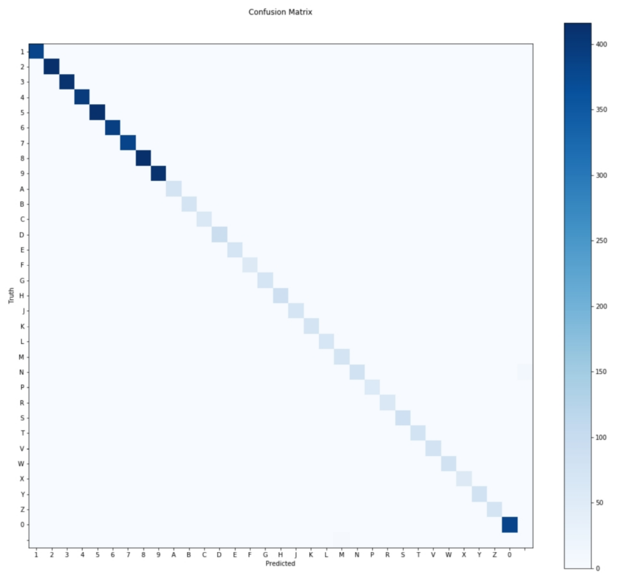
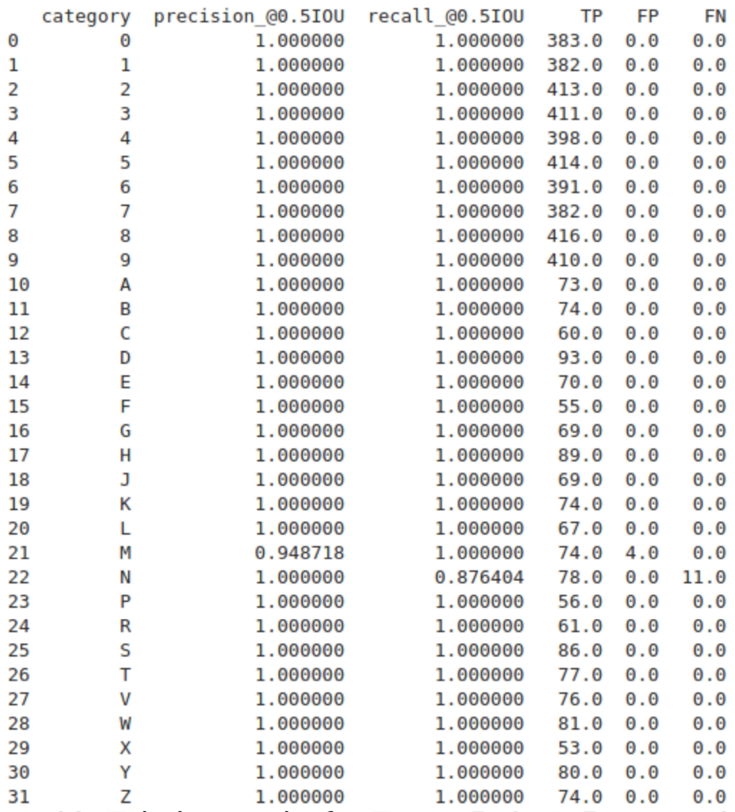


Figure 98: Confusion Matrix for Faster R-CNN ResNet 50 on images rotated with Gaussian noise of severity 5


```
   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.

[28]J. Brownlee, "A Gentle Introduction to the Challenge of Training Deep Learning Neural Network Models", *Machine Learning Mastery*, 2020. [Online]. Available: https://machinelearningmastery.com/a-gentle-introduction-to-the-challenge-of-training-deep-learning-neural-network-models/. [Accessed: 20- Sep- 2020].

Vita

Christina Paolicelli is originally from Pound Ridge, NY. She received her B.S. in 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.