License Plate Detection Under Various Supervised Algorithms

Akshay Raman
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA
akshayra@andrew.cmu.edu

Shang Shi
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA
shangs@andrew.cmu.edu

Leo Chen
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA
ikaic@andrew.cmu.edu

Abstract

The current state of parking is extremely outdated and inefficient. Most parking lots still use a physical ticketing system that requires many manual steps such as holding onto a physical ticket, paying at a pay station, etc. Automatic License Plate Detection (ALPD) can revolutionize the parking industry by eliminating all these manual components, providing a fully automated and streamlined parking experience. This paper aims to find the optimal Machine Learning (ML) pipeline for ALPD. YOLOv5 is first used to find the license plate from an image of a car. Computer Vision (CV) techniques are then used to separate the characters. Finally, to predict the characters, three classification algorithms, trained on the EMNIST balanced dataset, have been compared: (1) K-nearest neighbor (KNN), (2) feed-forward neural network (FNN), and (3) convoluted neural network (CNN). Ablation studies were done on each algorithm to find the optimal hyperparameters and regularization settings. Results showed the FNN proved to be the best method, providing a testing accuracy of 88%, compared to CNN (67%) and KNN (19%). Due to the small size of the EMNIST dataset images (28 x 28), removing features with pooling in CNN makes it harder for the model to train. For KNN, the model is not able to identify patterns as the similar nearest neighbors are in very different places for the testing images, which are printed characters, and the training images, which are handwritten characters.

1. Introduction

The current state of parking is extremely outdated and inefficient. Most parking lots still use a physical ticketing system that requires a multitude of manual steps such as entering/exiting the lot with a physical ticket or keycard, physically paying at a pay station, etc. Those that don't use such a system either rely on a human parking manager, which greatly increases cost, or the integrity of drivers, which means a lot of revenue is lost. This can all be completely streamlined with Automatic License Plate Detection (ALPD).

Automatic License Plate Detection (ALPD) removes the need for a physical ticket or keycard as it can identify the driver with their car's license plate. If the license plate is linked to the driver's credit card via a cellphone app, the driver can automatically be charged upon exit, eliminating the need for a pay station. Drivers will be able to enter/exit parking lots seamlessly, resulting in a truly automated parking experience.

Developing a parking management cellphone app is rather straightforward these days. The difficult part is the license plate detection component. Detecting a license plate boils down to three components: (1) finding the license plate bounding box (2) separating the characters and (3) predicting the characters. Character separation can be omitted if the whole license plate is to be predicted together.

There have been many approaches to this problem already. Current solutions fully rely on either Machine Learning (ML) algorithms or Computer Vision (CV) techniques. For ML-based solutions, some form of YOLO or CNN is generally used for all three steps. For CV-based solutions, edge detection and the bounding box method are used to find the license plate and separate the characters. Template matching is used to predict the characters.

For our solution, we wanted to combine the best of these two approaches. We firstly used YOLO to find the license plate bounding box. CV contour techniques were used to separate the characters. K-means and DBSCAN clustering methods were also tested, but proved ineffective in comparison to CV techniques.

Since we are dealing with images, the obvious ML solution for predicting license plate characters is to use CNN. This also happened to be the most common approach amongst the literature reviewed. However, the team was interested to see if other ML algorithms could produce better results. So, in addition to using CNN, the team modeled FNN and KNN solutions to predict characters.

2. Related Work

To determine the optimal methodology, the team considered several technical papers. In particular, the team looked for strategies to: (1) find the bounding box, (2) separate the characters, and (3) predict the characters. The two papers focused on have methodologies described below:

License Plate Detection and Recognition in Unconstrained Scenarios [14]:

- Warped Planar Object Detection Network (a type of CNN) to find the license plate
- YOLO to recognize license plate characters (doesn't separate license plate, predicts entire license plate at the same time)

Vehicle Number Plate Detection and Recognition using Bounding Box Method [4]:

- Sobel edge detection to find license plate
- Bounding box method to separate characters
- Template matching to recognize characters

The first paper has a Machine Learning (ML) approach, while the second paper has a Computer Vision (CV) approach. A thoughtful combination would provide the best of both worlds.

3. Data

The model design requires three sets of data images to successfully train and detect individual license plate characters. The first dataset is the images of cars with the license plate location labelled to train the YOLO method so the license plate bounding box can be found. The second dataset is images of cars and the third dataset is images of individual characters to train the different models on.

3.1. Images of Vehicles with Labelled License Plate

The first step in the method is to find the bounding box of the license plate. A trained YOLOv5 model was used to do this. To train it, a Kaggle dataset with vehicle images that had its license plate location annotated was used [8].

3.2. Images of Vehicles

The test images of vehicles were specifically selected for the algorithm to extract the boundaries of license plates and split up characters into individual images. The team selected images where the license plate is not so slanted that it is barely recognizable. Even so, vehicles within the images were oriented in different angles to closely mimic camera image captures of vehicles on an intersection of parking lot. The versatility of angles added an extra field of complexity with extracting the characters from the license plate.

The license plate images were obtained from platesmania [1]. This website contained a database of car images where the license plate is visible. However, for the team's model to work, some processing on the images such as splitting the characters on these license plates had to be performed. This is discussed more in depth in the methods section.

3.3. EMNIST Character Training Images

After splitting the license plate into separate characters, a different model was selected to train the characters. The chosen training dataset is the EMNIST Balanced dataset which contains 131,600 characters of 47 different classes [3]. The classes include digits from 0 to 9 as well as upper and lowercase letters which are displayed in Figure 1 [13]. The EMNIST dataset was chosen since it contains all the required classes and it is widely used. The images are 28 by 28 which is quite small so it is computationally inexpensive. The balanced version is chosen to ensure the model isn't biased to any specific classes. Since license plates are

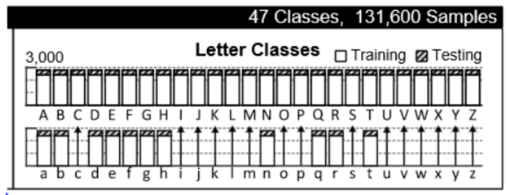


Figure 1. 47 Classes of EMNIST Balanced Dataset

only composed of digits and uppercase letters, the lowercase classes from the dataset were removed. As a result, the dataset yielded 10 digit classes and 26 letter classes, for a total of 36 classes as shown in Figure 2. However, since the dataset was originally flipped and rotated, before we trained the model, the images were modified into the desired orientation. The team performed some augmentations on the images such as inverting the colors on the images and adjusting the contrast to prevent over-training.

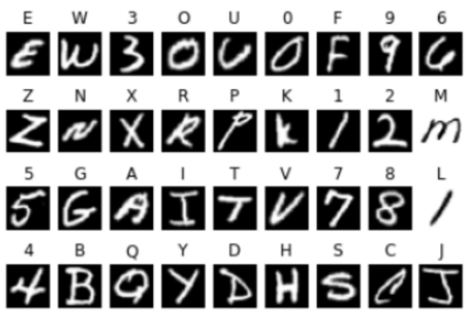


Figure 2. 36 Classes of EMNIST Balanced Dataset Used

4. Methods

The model was designed around the idea of making the license detection algorithm as computationally friendly as possible. As mentioned earlier, current license detection algorithms exist, but are computationally expensive - mostly built from computer vision techniques or learning algorithms centered around detecting all characters at once [11].

Before the individual models can be tested on the license plate characters, the raw vehicle images need to first be processed and split into the individual characters. As mentioned before, each model is trained on the training dataset before an ablation study is conducted to evaluate the performance of each model. Figure 3 below gives the overall overview of the model for a single car image. This model is repeated over each analyzed license plate image. Each step is explained in detail in the coming subsections.

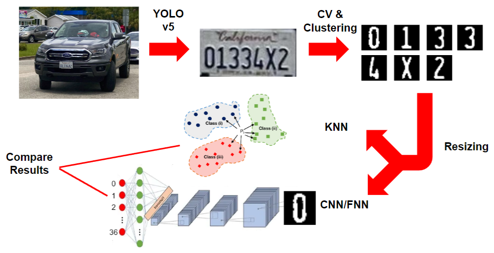


Figure 3. Sample Model Overview From Vehicle Image to Training Models

4.1. License Plate Extraction from Raw Image

After collecting a dataset of raw vehicle images, the first step is to automatically locate the license plate in each image. Initial hypothesis was to try and extract specific contours depths of the license plates to locate the license plate bounding box within the image. However, as seen in the below Figure 4, it proved t be very difficult to single out the license plate contours from the rest of the image. Hence, a pre-trained network, YOLOv5 was used to extract the bounding box of the license plate. YOLOv5s pre-trained weights were used to train an annotated license plate dataset downloaded from Kaggle, called ”Indian vehicle number plate yolo annotation”[5] [9]. After training, the algorithm was tested on car image training dataset. After training, YOLOv5 was able to reach an accuracy of about 70%. The network worked rather well on more angled images as well.

Figure 4. Straight On Image of Vehicle vs. Angled Image of Vehicle

4.2. Cropping Into License Plate

The output of YOLOv5 were image coordinates to the location of the license plate bounding boxes represented as [x_center, y_center, width, height]. The output bounding box from YOLOv5 was highlighted in red as shown in Figure 5 below [5]. All values are normalized. Using the size of each image, the location of the bounding box was determined for each image and the box cropped out of the image. The sole license plates, like the example shown in Figure 3, are saved as the new dataset.

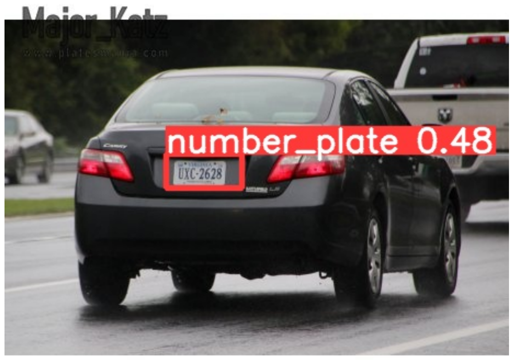


Figure 5. Bounding Box Image from YOLOv5 Before Being Cropped

4.3. Splitting Characters from License Plate

To keep the model as computationally friendly as possible, the model was designed to split the license plate images into separate characters before feeding into the various ML algorithms. This kept the number of classification classes to only 36 and reduced the computational complexity of the neural layers. Two main approaches were taken to split the plate images into separate characters: Machine Learning Unsupervised Clustering Approach Vs Computer Vision Contouring Approach. Both approaches are discussed below; however, ultimately the computer vision contouring approach was implemented into the final model as it provided the finer results.

4.3.1 Unsupervised Clustering Approach

In the United States, the number of characters in a license plate varies. Hence, it is impossible to split the characters using supervised learning techniques. K-means was initially used to try and cluster each character into a separate cluster. The license plate images were first binarized with the OTSU method before feeding into the K-means algorithm. However, K-means requires specifying the number of clusters to split the characters into. This proved impossible to solve as there was no way of cardinally knowing how many character are in the current license plate without manual inputs. DBSCAN proved to be a more promising algorithm to use as the number of clusters need not be specified. It is also able to split among non-linear lines [16]. Multiple parameter iterations of DBSCAN and K-means were used to optimize clustering the individual characters. Table 1 below shows how well each algorithm performed with different parameter settings for K-means. The outcome was ”yes” if the algorithm was successfully able to split all characters from non-character boundaries. For DBSCAN, Table 2 shows what area percentage DBSCAN was able to recognize of all characters in the image.

Table 1. KMeans Parameter Tuning for Character Clustering

# of Clusters	Initialization Type	Outcome
1	Random	No
2	Random	Yes
2	Naive Sharding	No
7	Random	No
7	Kmeans++	No

Table 2. DBSCAN Parameter Tuning for Character Clustering

EPS	Min Samples	Multiple Clusters?	% Recognized
1	10	No	0%
10	10	Yes	10%
20	10	No	1%
1	100	No	0%
10	100	No	40%
20	100	No	95%

However, as seen in the Figures 6 and 7 below, both algorithms failed to properly split the characters when more than one cluster was specified. For K-means, there is no input to specify the distance between pixels to aid the clustering. Hence, K-means has no method of grouping characters based on its distance. A similar issue is seen with DBSCAN as well. Since DBSCAN only incorporates pixel color and not relative distance, it is not able to differentiate that a single cluster can in fact be split into multiple clusters [10]. Potential solutions for K-means and DBSCAN for better cluster prediction are given below:

- Kmeans:
  - Add a relative distance to some arbitrary origin point to each RGB pixel. Kmeans can incorporate each pixel color as well its relative distance.
  - Change the color of each contour identified and split clusters based on the color of each character.
- DBSCAN:
  - Add a relative distance to some arbitrary origin point to each RGB pixel. Allow DBSCAN to filter clusters out based on distance as well.
  - Change the sample size of DBSCAN to match the different color for each character. Would need to change the color of each character based on identified contours.

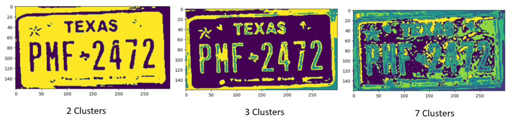


Figure 6. Kmeans Clustering for Different Specified # of Clusters

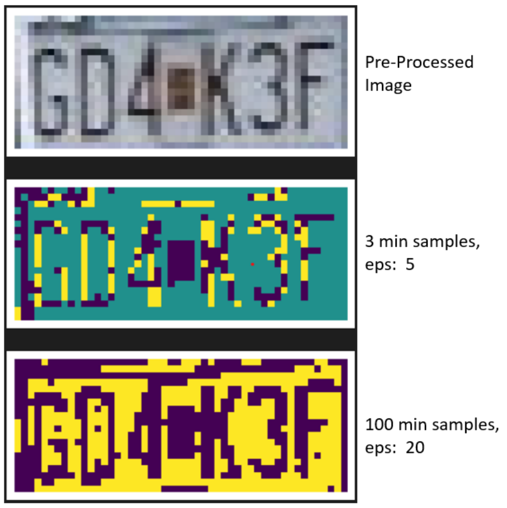


Figure 7. DBSCAN for Different EPS and N_Clusters

4.3.2 Computer Vision Contour Approach

Although, as mentioned above, K-means was able to accurately cluster all letters as one cluster, computer vision techniques would still need to be used to split each character. Instead of identifying the characters using K-means, contour techniques can be used to extract each letter individually. After applying a padding to make all images symmetrical, the license plates were binarized with the OTSU method. After binarization, erosion and dilation techniques were applied to the images before finding the contours. Erosion and dilation removed all smaller text and thin contours, leaving only the main thick characters [6]. Figure 8 below gives a visual representation of the steps taken to obtain the different contours from an image.

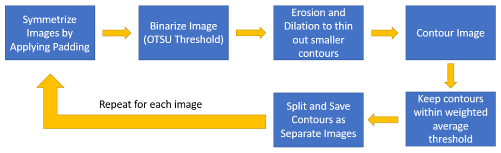

Symmetrize Images by Applying Padding
Binarize Image (OTSU Threshold)
Erosion and Dilation to thin out smaller contours
Contour Image
Keep contours within weighted average threshold
Split and Save Contours as Separate Images
Repeat for each image

Figure 8. Contouring Method to Extract Individual Character from License Plate Image

Unfortunately, as seen in the 'mask' tab in Figure 9 above, there are other unnecessary texts such as state of origin of the license plate that appear in the contour maps. Since the actual license plate character contours have a much greater area than that of the unnecessary text, only contours that remained within an area threshold were kept. To keep consistent with differences between each license plate design, the area ranges were calculated for each license plate. For each image, a weighted average for contour sizes was calculated and a set range was applied to each end. After applying this threshold, the image in 'cont' tab of Figure 9 shows the remaining contours.

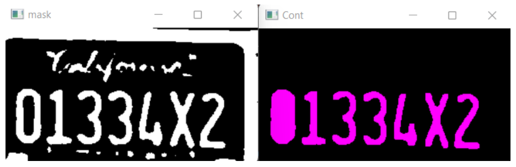


Figure 9. Contouring Method to Extract Individual Character from License Plate Image

4.4. Resizing Character Images

Each contour is split into separate character images and resized/binarized to match the EMNIST character training dataset. The individual character images were saved into a folder named after the label of the license plate.

4.5. Conducting Ablation Study

The main purpose of this project has been to find more cost efficient methods to solve license plate detection problems. As mentioned before, by splitting up the characters, it is more efficient to run ML algorithms on individual characters as there are less classification parameters. To figure out the most cost efficient algorithm, an ablation study was conducted. The models chosen were kept as simple as possible while tuning parameters to maximize classification accuracy. Final chosen models are explained in more detail below:

- **K-Nearest Neighbors (KNN):** KNN is a supervised learning classifier that uses proximity to make predictions on test images. KNN finds patterns by locating most popular points surrounding a test image point to classify which label the image belongs too. The model is very strong in recognizing patterns in text and numbers in handwritten formats [2]. Hence, the model was chosen due to its strong performance in handwritten character classification.

- **Feed-Forward Neural Networks (FNN):** Feed forward neural network is an artificial neural network with connected nodes that calculate weights between features to find the optimal classification. The model uses an iterative process to constantly update weights by reducing the loss between the ground truth and predicted classification. For images, FNN creates weights and relationships between each pixel[7]. Due to the large number of weight networks, FNN are slightly computationally expensive. However, since the dataset being trained on is a very simple 28x28 pixel image, the model is still significantly cost efficient compared to current methods.

- **Convolutional Neural Networks (CNN):** CNNs are a deep learning algorithm that uses convolutions to scan over an image and find features across the image. The convolutions allow the model to identify differences between images to accurately classify the image regardless of rotation, translation, and dilation. Since the dataset being used was handwritten letters compared to printed characters as shown in Figure 10, CNNs would still be able to accurately classify characters despite the variations between printed and handwritten characters [15]. CNN was chosen as the most promising model due to its ability to simplify features and its versatility in image detection.

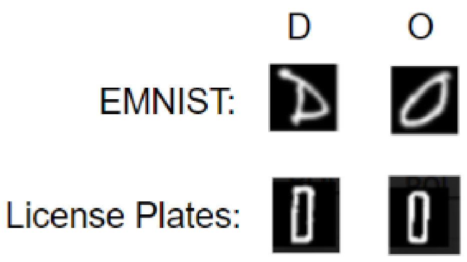


Figure 10. Handwritten Vs Printed Differences for Letters ’D’ and ’O’

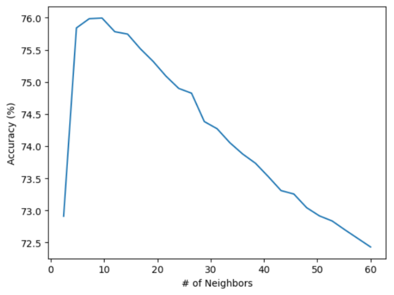


Figure 11. KNN EMNIST Data Testing Accuracy for Increasing N Neighbors Parameters

5. Experiments

To find the optimal classification method to predict the individual characters, three algorithms were tested and compared: (1) KNN (2) FNN and (3) CNN.

5.1. K-Nearest Neighbors Model

For KNN to accurately recognize patterns, both EMNIST train and license plate test data were binarized, padded to the same size, and flattened. An ablation study was conducted on the KNN parameters to maximize test accuracy. Gridsearch was used to optimize the hyper-parameters. Gridsearch is an automated tuning technique that tries to find the optimized solution by iterating through different iterations of hyper-parameters. For KNN, Gridsearch was used to iterate over n_neighbors (1 to 60) and weights (uniform or distance). The neighbors parameter is used to place number of points to find neighbors next too while weights is the method to calculate the distance between the neighbor and the points on the test image [12]. See Table 3 for results on EMNIST testing data before and after applying Gridsearch. The optimal parameters from Gridsearch is 12 n_neighbors and ’uniform’ weights. Figure 11 gives a visual representation of the EMNIST test accuracy as the n neighbors parameter is increased from 1 to 60. The optimal parameters from KNN Gridsearch algorithm was used to predict individual license plate characters. Surprisingly, even though the model produced 76% EMNIST testing accuracy, when tasked to predict license plate characters, the results barely crossed a testing accuracy of 18.6%. Using different number of n_neighbors reduced the testing accuracy by about 4%. Applying bagging techniques could slightly increase testing results but not improve by much. KNN’s poor performance is largely due to training on handwritten characters but predicting printed plate characters. As seen in Figure 10, there are large differences in how characters are represented between EMNIST and the license plate. The model is not able to identify patterns as the similar nearest neighbors are in very different places from the labeled image and the training images.

Table 3. EMNIST Test Accuracy With Different KNN Parameters

N Neighbors	Weight	Gridsearch	Test Accuracy
3	Uniform	No	75.6%
50	Uniform	No	72.43%
12	Uniform	Yes	77.8%

5.2. Feed-forward Neural Network Model

An ablation study was also done on a feed-forward neural network to maximize test accuracy. The team first started out with a 410-node hidden layer, using ReLU as the activation function. 410 nodes were chosen as it was close to the halfway point between the number of output (36) and input dimensions (784). First one hidden layer was chosen arbitrarily, just to see if it was enough to achieve an acceptable accuracy. With a learning rate of 0.0001, batch size and number of epochs of 20, the model converged with a test accuracy of 84% (testing was done on our own dataset). See Table 4 for results.

Table 4. Accuracies for Standard FNN (%)

Training	Validation	Testing
91.2	83.8	85.0

The accuracy vs. epochs plot can be seen in Figure 12.

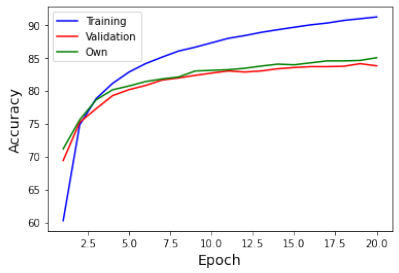


Figure 12. Accuracy vs. Epochs for Standard FNN. ”Own” refers to our own dataset, which corresponds to ”Testing” in the tables.

A model with highly accurate results was already received. However, a tuning further could proved a more promising model. To achieve this, dropout is added to the hidden layer to remove any overfitting that could have happened.

However, the observed a drop accuracy throughout (see Table 5 for results). This is likely due to both the simplicity of the model and the small image size. Because the model is already very simple and the images has very few features, removing nodes makes it even harder to train.

Table 5. Accuracies for FNN with Dropout (%)

Training	Validation	Testing
86.6	82.4	82.4

The accuracy vs. epochs plot can be seen in Figure 13.

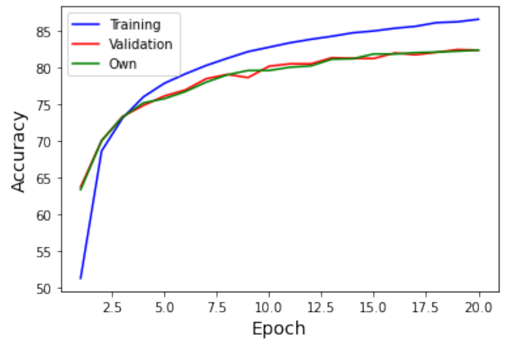


Figure 13. Accuracy vs. Epochs for FNN with Dropout

Dropout proved ineffective in improving accuracy, so hyperparameter tuning was done on the learning rate and batch size (see Table 6 for results).

Table 6. Testing Accuracy for FNN Hyperparameter Tuning (%)

Learning Rate / Batch Size	50	100	200
0.001	86.9	86.8	86.5
0.0015	87.1	86.7	86.1
0.003	84.7	84.5	83.7

The best test accuracy, at 87.1%, was given with a learning rate of 0.0015 and a batch size of 50. This is 2% higher than the test accuracy prior to hyperparameter tuning, which was 85.0% (learning rate of 0.0001 and batch size of 20). The batch size is also much greater than that of the pre-tuned model. This means the tuned model is able to achieve a higher accuracy with a smaller number of iterations, improving computational efficiency.

Considering this, using a learning rate of 0.0015 and a batch size 100 makes the most sense, as it gives a similar testing accuracy (86.7%) with the least number of iterations. Hyperparameter tuning also showed that a learning rate of 0.003 is too large and should not be used as the testing accuracy drops to an average of around 84.3% (lower than the pre-tuned accuracy).

In addition to tuning the learning rate and batch size, the NN size was also tuned in search of a more accurate model. Two changes were made to the NN model: (1) increase the number of hidden layers (HL) to two (2) with 2 HLs, increase number of neurons from 410 to 600. The results can be see in Table 7.

Table 7. Testing Accuracy for FNN Model Size Tuning (%)

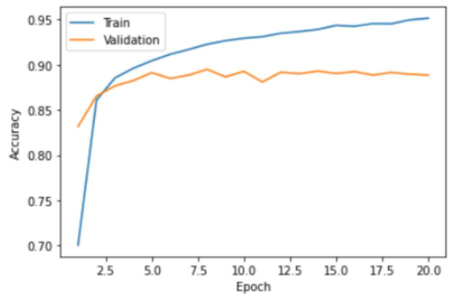
Modification	Training	Validation	Testing
2 HLs	96.6	87.8	88.0
2 HLs & 600 Neurons	97.3	87.6	87.7

Results show that 2 hidden layers increased the training accuracy drastically to 96.6%. The testing accuracy did see a 0.9% increase from the best-tuned model. With the increase from 410 to 600 neurons, training accuracy increased to 97.3%, but testing accuracy didn't change much at 87.7%. Overall, adding an additional HL was useful in improving testing accuracy. However, increasing the number of neurons did not prove as effective.

Whether or not the additional hidden layer should be added ultimately depends on the computational time and resources required. A more in depth computation study should be done in the future to determine if the 0.9% improvement in testing accuracy is worth the extra computational cost.

5.3. Convolutional Neural Network Model (CNN)

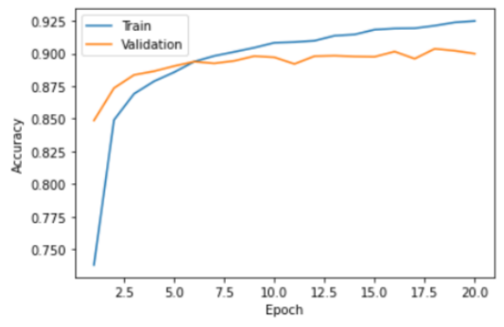
For the CNN, many different model structures and hyper-parameters were experimented with. The overall CNN structure included convolution layers attached to pooling and ReLU activation layers. First started with just a single convolution layer attached to a hidden classification layer which outputs 36 classes. This model achieved a training accuracy of over 95% but when tested on the actual license plate images, the accuracy was only 55%.



```
Final Training Accuracy: 0.9515335648148148
Final Validation Accuracy: 0.8885416666666667
```

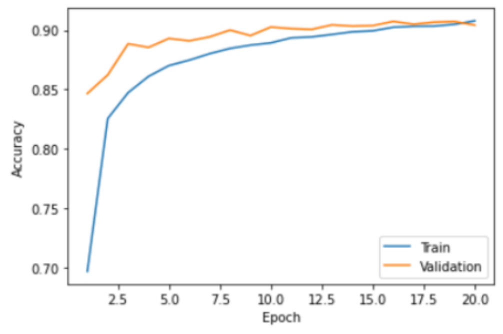
Figure 14. First CNN Model

The team then introduced two additional convolution layers to increase feature extraction capabilities. With this, the new model achieved a training accuracy of 92.4% as shown in Figure 15 but a testing accuracy on the license plates of only 50%. To increase accuracy, dropout layers were added since the training accuracy is continuing to grow while that of the validation has stagnated. With the introduction of dropout, the model is no longer overtrained in Figure 16. When testing this model on the license plates, it increased the testing accuracy to 67%.



```
Final Training Accuracy: 0.9248408564814815
Final Validation Accuracy: 0.8998263888888889
```

Figure 15. CNN Model Without Dropout



```
Final Training Accuracy: 0.9078125
Final Validation Accuracy: 0.9042824074074074
```

Figure 16. CNN Model with Dropout

Parameters were further tuned to find the 3 layer CNN to produce the best testing results. Up to this point, the model has been run with a batch size (BS) of 100 and learning rate (LR) of 0.0015. Therefore, the next step was to tune these hyperparameters. To tune the batch size and learning rates, one variable was set constant while changing the other one to see the effects. Tables 8 and 9 below summarizes the results. It is observed that the parameters that started with 100 BS and 0.0015 LR performed the best, which is consistent with the FNN results.

Table 8. Testing Accuracy for CNN BS Tuning (%)

Batch Size (LR = 0.001)	Accuracy
50	57
100	67
200	57

Table 9. Testing Accuracy for CNN LR Tuning (%)

Learning Rate (BS = 100)	Accuracy
0.001	55
0.0015	67
0.003	59

6. Conclusion

For this project, the team aimed to recognize license plates, split the characters, and accurately predict them. A pretrained YOLOv5 was used to get the bounding box from the car images. From there, 3 different methods were used to classify the characters: KNN, FNN, and CNN. These models were all trained on the EMNIST Balanced dataset as it was light computationally and easily accessible. The team initially believed CNN would have the optimal performance due to its feature extraction abilities. But, in fact, the FNN had the highest testing accuracy on actual license plates of 88%. Comparatively, CNN only achieved 67% and KNN, 19%. The team will need to further investigate these results but the initial hypothesis is that since the images are already so small, CNNs, which reduces the dimensions of the images even further, are not as effective with distinguishing the fine differences between characters. In addition, it can be observed that the testing accuracy on the EMNIST characters were much higher than when testing on actual plates. Therefore, in the future, it is wiser to gather a dataset of actual license plate characters to be used for training.

7. Codebase

All code is located in the following GitHub Repository Link: `https://github.com/Stevenshi12/License-Plate-Classification`

8. Individual Contributions

The members of this team are: Akshay Raman, Shang Shi, and Leo Chen. Each member was responsible for the ablation study of one ML algorithm: Akshay (KNN), Shang (CNN), and Leo (FNN). The pre-processing tasks (YOLOv5, bounding box extraction, character segmentation) were split equally amongst all members.

All members participated actively and made substantial contributions to the project. Overall, collaboration was extremely effective.

References

[1] Platesmania.

[2] What is the k-nearest neighbors algorithm?

[3] The emnist dataset, Mar 2019.

[4] K. M. Babu and M. V. Raghunadh. Vehicle number plate detection and recognition using bounding box method. *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, 2016.

[5] A. .c. Automatic vehicle number plate detection using yolov5 and extraction of text using ocr, Jan 2022.

[6] B. Catanzaro, B.-Y. Su, N. Sundaram, Y. Lee, M. Murphy, and K. Keutzer. Efficient, high-quality image contour detection. *2009 IEEE 12th International Conference on Computer Vision*, 2009.

[7] DeepAI. Feed forward neural network, May 2019.

[8] Deepak. Indian vehicle number plate yolo annotation.

[9] Deepak. Indian vehicle number plate yolo annotation, Sep 2021.

[10] R. Fatt. Image not segmenting properly using dbscan, Dec 2016.

[11] R. Laroca and D. Menotti. Automatic license plate recognition: An efficient and layout-independent system based on the yolo detector. *Anais Estendidos da Conference on Graphics, Patterns and Images (SIBRAPI Estendido 2020)*, 2020.

[12] T. Penumudy. A beginner's guide to knn and mnist handwritten digits recognition using knn from scratch, Jan 2021.

[13] A. Shawon, M. Jamil-Ur Rahman, F. Mahmud, and M. Arefin Zaman. Bangla handwritten digit recognition using deep cnn for large and unbiased dataset. *2018 International Conference on Bangla Speech and Language Processing (ICBSLP)*, 2018.

[14] S. M. Silva and C. R. Jung. License plate detection and recognition in unconstrained scenarios. *Computer Vision – ECCV 2018*, pages 593–609, 2018.

[15] R. Yamashita, M. Nishio, R. K. Do, and K. Togashi. Convolutional neural networks: An overview and application in radiology. *Insights into Imaging*, 9(4):611–629, 2018.

[16] Q. Ye, W. Gao, and W. Zeng. Color image segmentation using density-based clustering. *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).*, 2003.