Spring 5-22-2019

Traffic Flow Prediction Using Convolutional Neural Network accelerated by Spark Distributed Cluster

Yihang Tang
San Jose State University

Traffic Flow Prediction Using Convolutional Neural Network accelerated by Spark Distributed Cluster

A Project Report

Presented to

Dr. Melody Moh

Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Class

CS298

By

Yihang Tang

May. 2019

The Designated Project Committee Approves the Project Titled

Traffic Flow Prediction Using Convolutional Neural Network accelerated by Spark Distributed Cluster

By

Yihang Tang

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2019

Abstract

Obtain information from historical data to forecast traffic flow in a city can be difficult because a precision forecasting demands large amount of data and accurate pattern analysis. Meanwhile, it is also meaningful because it provides a detailed and accurate point-to-point prediction for users. In this project, I use CNN (Convolutional Neural Network) to train the model based on the images captured by webcams in New York City. Then I deploy the training process on a Spark distributed Cluster so that the whole training process is accelerated. To efficiently combine CNN and Apache Spark, the prediction model is re-designed and optimized, and the distributed cluster is tuned. By using 5-fold validation, multiple test results are presented to provides a support for the analysis about the model optimization and distributed cluster tuning. The aim of this project is to find the most accurate prediction model for the traffic flow prediction with acceptable time cost.

Acknowledgements

I would like to thank Dr. Melody Moh for her continued guidance and support throughout my project. I would like to thank my committee members Dr. Teng Moh and Dr. Ching-Seh Wu for their guidance and review of the project. Their experience and knowledge on different fields inspire and enlighten me. A huge thanks to my parents for their patience and support. A huge thanks to my girlfriend for her support and encouragement.

Table of Contents

List of Tables

List of Figures

I. Introduction:

The prediction of traffic flow has been widely applied in modern society. Many industries intend use real-time data to generate an instant prediction. However, it is still a painful process as it is challenging to find a balance between accuracy and time cost. Accurate results often need complicated analysis based on a large amount of data. Alternatively, one can sacrifice accuracy to save some time, but it may lead to an unacceptable prediction result. For traffic flow forecasting, in this project, I propose a method that runs a prediction on images containing moving vehicles. By analyzing the movement of vehicles and the relationship between vehicles and locations, a classification of traffic situation among all available coordinates in the route is conducted, and it can display the overall traffic flow from the desired starting point to the destination.

Furthermore, I study the characteristics and performance of CNN to explore how it may utilize a distributed environment. Specifically, three components of CNN are discussed for object recognition in the image: learning rate, activation function, and pooling layer. By adjusting these three components, my CNN model becomes an effective tool to recognize vehicles from captured images and transfer them to classifications between the number of vehicles and coordinates.

The optimization of CNN mainly aims to increase prediction accuracy. However, training of the model takes an enormous amount of time because CNN is a complicated neural network, and the size of the dataset is big. Therefore, to accelerate the entire training process Apache Spark [1] is utilized. It is a cluster-computing framework that supports the MapReduce paradigm yet does not depend on it [2]. Different modes of the Spark cluster are implemented and compared. The performance of GPU and CPU accelerations is also explored based on a series of experiments.

II. History and Background

a. Machine Learning

Walter Pitts and Warren McCulloch are the first guys who tried to mimic the way a neuron was thought to work in the 1940s [3]. This is considered as the starting point of machine learning since it established a concept about how a machine can simulate the mechanism of a human being's brain. Then, ML (Machine Learning) and AI (Artificial Intelligence) entered its first winter, not many breakthroughs happened until the '70s, Rosenblatt's perceptron began to gain a bit of attention [3]. However, his study result got doubt by Seymor Parpert because Seymor proved that perceptron was incapable of learning the simple exclusive-or function. This leads to the first AI winter.

Then, after over 3 decades' silence, Hinton, once again, declared that he found out how the brain actually works in 2006. He introduced the idea of unsupervised pretraining and deep belief nets [3], which brings the interest back to this field. Sooner, more and more paper about neural network appeared and attracted more attention. In 2012, a significant breakthrough happened, JS Denker [3], lead a project that successfully using deep nets for speech recognition. Then, the use of graphics processing units (GPU) to train the model break through the limitation of physical resources, and the research of AI and ML literally ushered in a new era.

b. Deep Learning

Before DL (Deep Learning) came into view, ML struggled with handmade features. Obviously, this is not "intelligent" enough, but the appearing of Deep Learning solved this question. In general, the most important difference between traditional ML and DL is how features are extracted [4]. Unlike traditional ML, DL learned and extracted features "automatically" and represented hierarchically in multiple levels [4]. This means that DL is stronger than traditional ML in most practical scenarios. Below is a figure [3] that briefly describe the development of ML and DL.

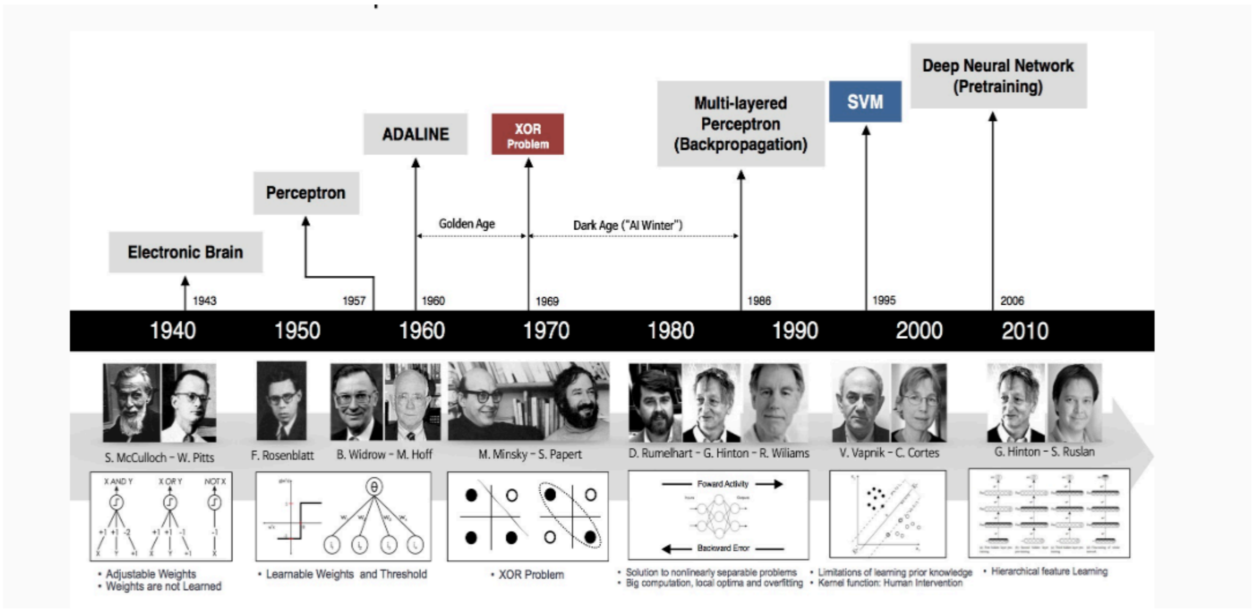

Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
ADALINE
XOR Problem
Perceptron
Golden Age
Dark Age ("AI Winter")
Electronic Brain
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

Figure 1: Development of ML and DL

c. CNN

CNN, as one of the most famous DL classes, is no doubt the superstar in visual imagery analysis. To pursue the origin of CNN, we need to go back to 1980's, where Kunihiko Fukushima introduced the concept called "neocognitron" [5]. He introduced two significant components in CNN: convolutional layers and downsampling layers. These two components are still the core of CNN even after several decades.

d. Spark Cluster and Distributed Environment Acceleration

The development of Apache Spark is a journey from academia to industry. In 2009, a class project was proposed in UC Berkeley, which is about building a cluster management framework that can support different types of cluster computing systems [6]. This project was called mesos, and it is the rudiment of Apache Spark.

Scalable and efficient LPR (SELPR) approach is an approach that uses YOLO network to detect the location of the license plate, which is introduced by W.Zhang, et [7]. This approach is deployed on Apache Spark and take advantage of it to improve the processing speed.

Zhang and other authors also used CNN to construct their model since it relates to the object recognition in the image, and CNN is no doubt the best choice for this task. Similar to the project proposed in this paper, Zhang also used Spark cluster to establish a distributed environment, but it was mainly used for data preprocessing. Although it indeed improves the performance of CNN model, I believe there is more potential to exploit.

III. Existing Methods

Before I chose CNN, I did a lot of research on the traditional ML classifiers and DL model CNN. My research mainly focused on two aspects: the accuracy and time cost for a model/classifier to identify the specific object from images. The traditional ML classifiers that I chose includes Decision Tree, Random Forest, SVM (Support Vector Machine), and KNN (K Nearest Neighbors).

a. Decision Tree

Decision Tree is a tree-like model which is frequently being used in the decision process. In general, this classifier mainly used conditional control statement to train data and extract features. Each node in the decision tree is a “test”, and each test generates separate class labels. This classifier is considered expensive to construct and fast [8]. In the dataset, there exists a large number of records that have redundant or repeated attributes, and the decision tree is a good tool to handle this type of data.

b. Random Forest

Random Forest can be considered as an advanced version of the decision tree, so sometimes people also called it a random decision tree. Compare to the decision tree, random forest constructs multiple decision trees randomly and take the mean or mode to classify data while training data.

In this project, the random forest is also chosen because it usually provides a more accurate prediction result than the decision tree does, even though it also takes more time and computational cost. Besides that, the random forest provides a congestion state which cannot be achieved by using decision tree because it can reflect the traffic in an area rather than just a data point, which is very effective for this project [10].

c. SVM

Support Vector Machine (SVM) is a discriminative classifier that defines hyperplane to categorize data. A hyperplane can be linear or non-linear, and since most roads in are not distributed linearly, the non-linear hyperplane is a better choice for my project. Another key concept is the kernel. A kernel is a function that implies the correlation of coordinates without computing it. This function can help with avoiding overfitting [11]. In this project, I use a kernel function called Radial basis function (RBF). RBF takes two samples x and x' as feature vectors in some input space, and define the squared Euclidean distance based on it [12].

d. KNN

K nearest neighbor is another ML classifier that has been widely applied. K is the number of constant samples that the user defined. This method aims to "remember" the relative position of previous samples in multi-dimension coordination. A sample will be assigned to a class most common among its k nearest neighbors. KNN is relatively expensive compared to the decision tree or random forest. Further performance comparison will be given in the following section. Another concept that affects how KNN categorize data is the way to "vote". The most common way to vote for deciding which class a data point belongs to is the majority vote, which means this data point will be categorized to the class that is major among k nearest neighbors. However, this is not the best fit sometimes, and a custom vote is an alternative method in that case.

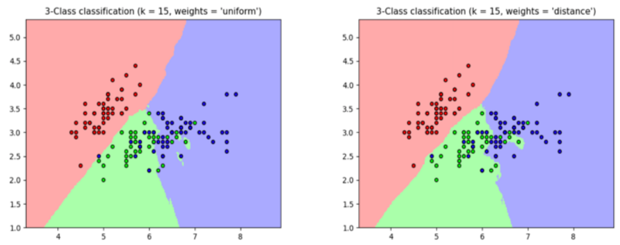


Figure 2: Comparison: uniform majority votes vs. custom majority votes

Above are two figures [13] that show the difference between KNN with uniform majority votes and KNN with custom majority votes. A custom vote using Euclidean distance performs more accurate than the default majority vote in this dataset. One of the small green areas in blue class is excluded by using the customize voting method.

e. CNN

CNN, as mentioned above, is a popular class of DL. CNN requires a conversion of the input: for image processing, the raw image need to be converted to a matrix of pixels. Based on the image resolution, the height, the width, and the dimension of the matrix are various. Then, these matrices of pixels are passed into a structure called the hidden layer. Hidden layer typically contains convolutional layers with activation functions as filters, pooling layer, which is also known as subsampling layer, and fully connected layer, which sometimes come with another pooling layer. Then, the final result is passed to the output layer. In my project, the design and optimization of CNN model also follow this structure. Details about the modification on the model will be provided in the following section.

A figure displays the basic structure of CNN [4].

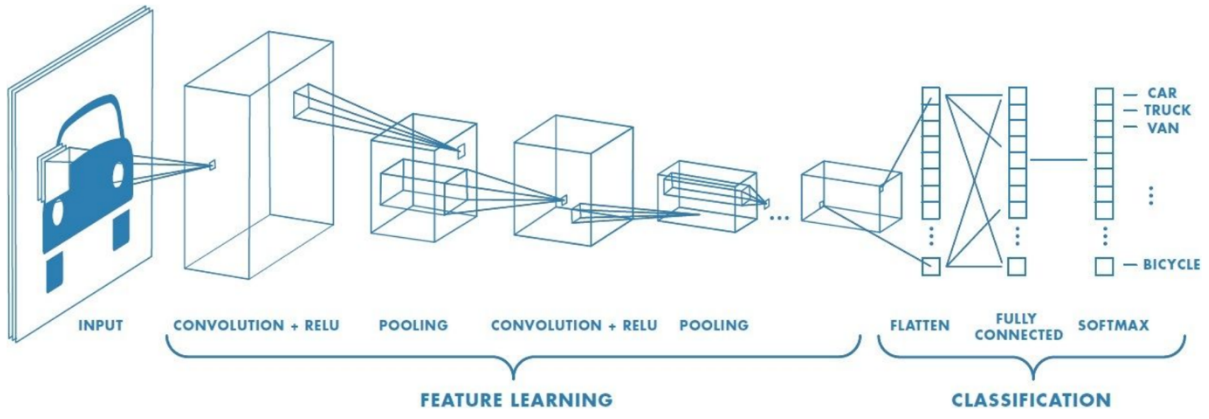


Figure 3: Structure of CNN

f. Comparison of Performance

Now, the question becomes: which of the above tools has better performance in case of object detecting in the image? The best way to answer this question is to set up an experiment and compare them directly. Therefore, I implement all ML classifiers using a python library called scikit-learn and implement CNN using a framework called deeplearning4j. Below is the test result based on a sample contains 5000 frames in 3, 6, 9 and 12 months:

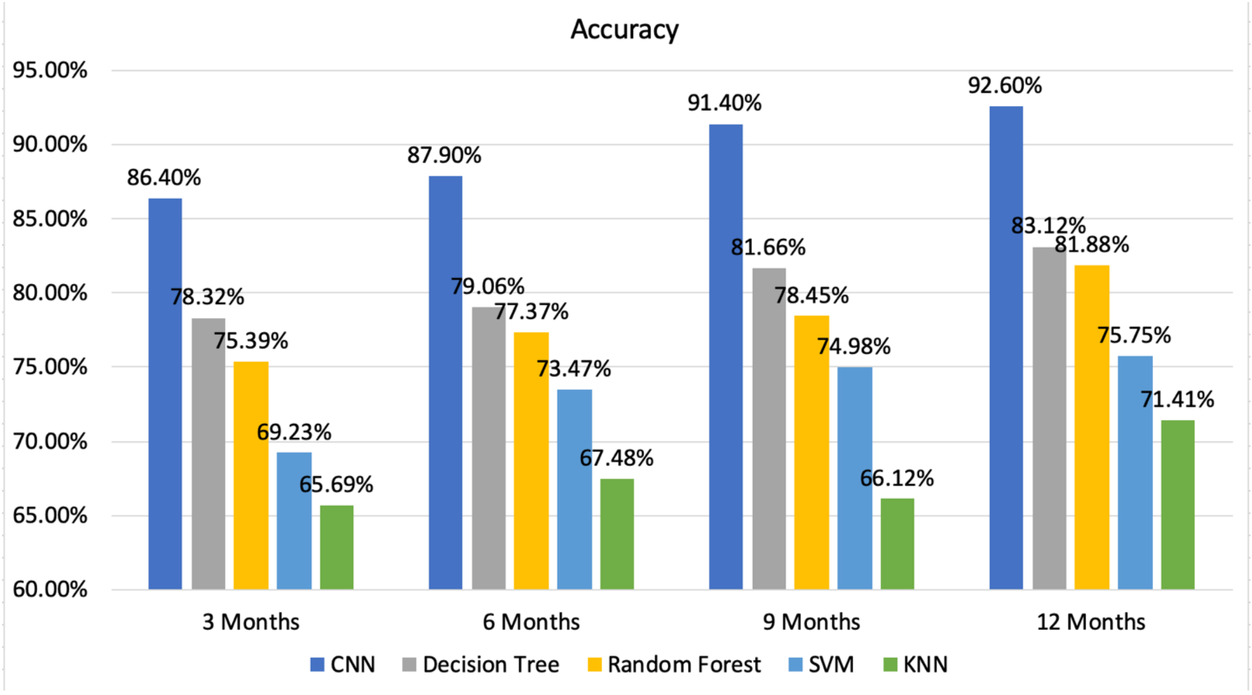


Figure 4: Accuracy - 5 ML Classifiers and CNN

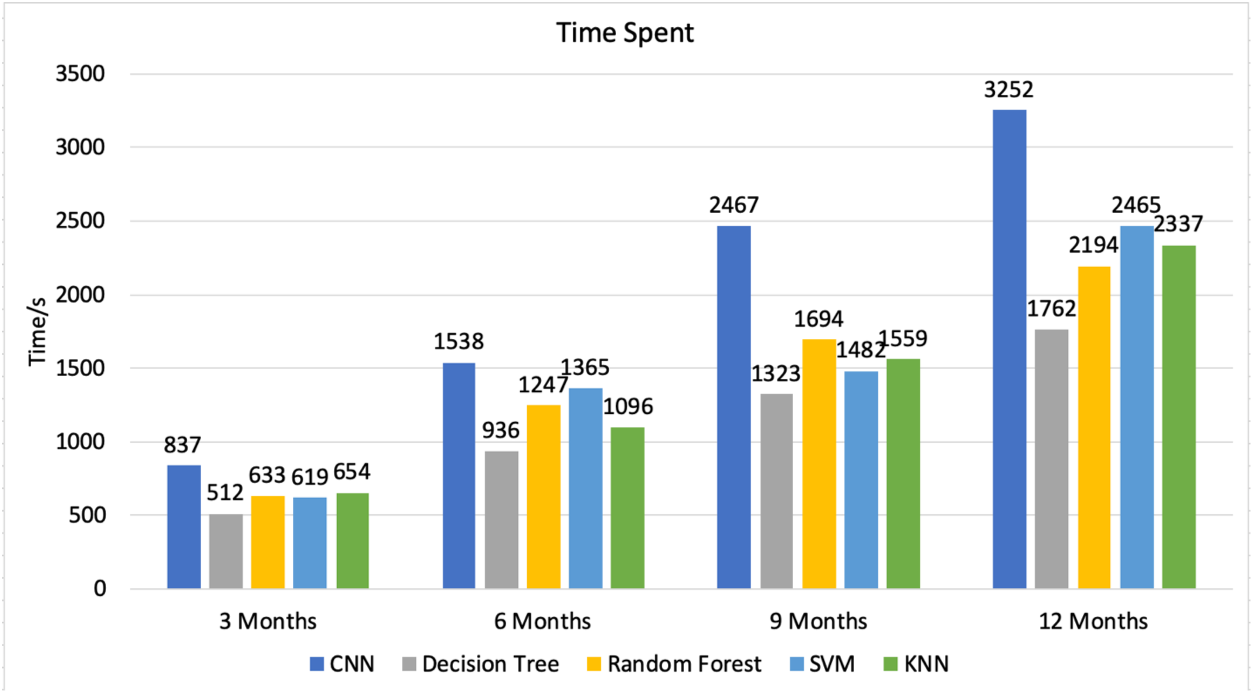


Figure 5: Training Time - 5 ML Classifiers and CNN

It is obvious that CNN generates much more accurate prediction than all 5 ML classifiers. This is because it has a more complicated structure and avoids the bias caused by pre-defined classes. Unfortunately, CNN takes much more time to train and to get the final results. As shown in Figure.

5, the time CNN spends on a sample contains frames in 12 months is almost twice as much as Decision Tree does. This leads to the core of my project: how to accelerate the training process of CNN? As stated above, my answer is Spark cluster, and a detailed explanation about how to do it is given in the following section.

IV. Improved Method

a. Workflow

After evaluating the performance of CNN and the traditional ML methods, it is not hard to understand that CNN has excellent accuracy on object detection on image. However, how to convert this advantage of CNN into the prediction of traffic flow is also a challenge. In the CityCam dataset, the location of each camera is given. Then, if I connect the detected number of vehicles with the location of each camera, I can easily get an estimation of the number of vehicles between two points on the map. Besides that, when I consider this question, the number of vehicles is not the only factor that affects traffic. The situation of the road also does. Thus, instead of simply detect how many vehicles is in an image, I tried to get a percentage that represents how many pixels belongs to detected vehicles in the whole matrix that converted from an image. Then, no matter what the width of a road is, this percentage can still display the traffic situation at that time. Thus, the workflow of making a prediction is: 1. CNN obtain the percentage that represents the region belongs to a detected vehicles/ the size of the whole image. 2. Each point that represents a camera from the starting point to the destination is assigned with a percentage that is the average of the percentage obtained in step 1. 3. Final prediction between two points on the map is formed by each camera with a percentage on the route. Details about how to get the percentage of an image are provided in the next section since it relates to the specific component of CNN.

Thus, the workflow of making a prediction is as follows:

1. *CNN obtain the percentage that represents the region belongs to a detected vehicles/ the size of the whole image.*
2. *Each point that represents a camera from the starting point to the destination is assigned with a percentage that is the average of the percentage obtained in step 1.*
3. *Final prediction between two points on the map is formed by each camera with a percentage on the route. Details about how to get the percentage of an image are provided in the next section since it relates to the specific component of CNN.*

b. CNN Model design and optimization

i. Learning rate

Learning rate is an important hyper-parameter that affects the adjusting of the weight of the neural network with respect to the loss gradient [14]. In other words, the learning rate is the most important factor that decides the rate of weight changing in the neural network model. A naïve neural network usually takes a stable learning rate when it is configured. However, if you use one constant learning rate from beginning to the end, one of the shortcomings is that the gradient loss can be inefficient.

The relationship between loss gradient and the learning rate is changing along with the increasing of the epochs. So, a constant learning rate cannot guarantee the best result among all the time.

I did an experiment to explore the relationship between learning rate and loss gradient for this project, and below is the result:

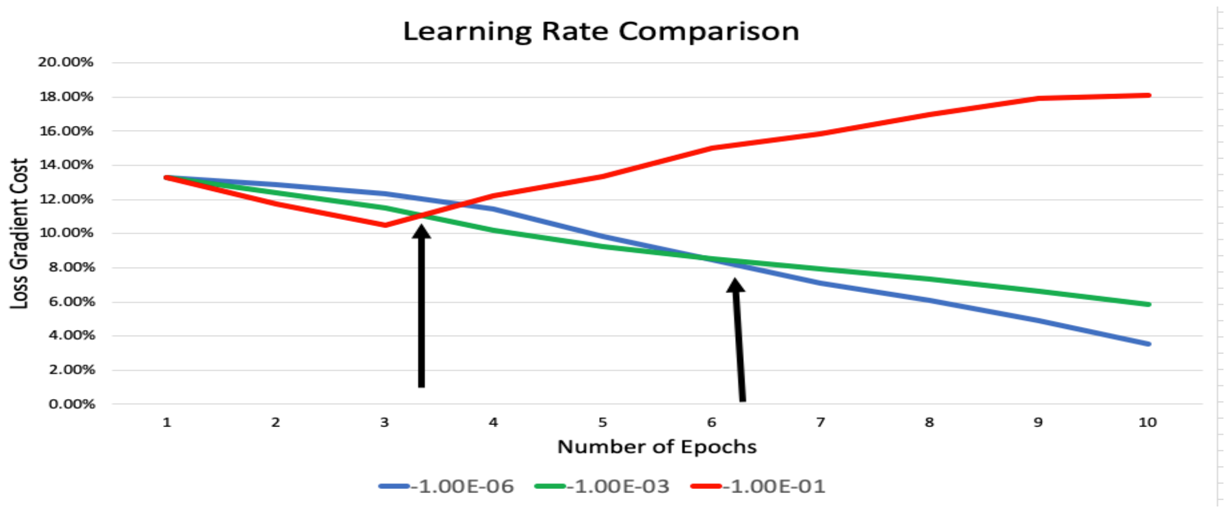


Figure 6: Comparison of Loss Gradient Decent between Different Learning Rates

Figure 6 shows the loss of gradient decent of three different learning rates: 1.00E-01, 1.00E-03, and 1.00E-06. Obviously, there is no learning rate that is the best choice among all the time, so a better strategy is to adjust the learning rate to achieve the best combination. In my project, obviously, starting with learning rate equals to 1.00E-01, then switch to 1.00E-03 after the training of third epochs finished, and change to 1.00E-06 after the training of sixth epochs finished is the best strategy. The rate of loss gradient is kept at the lowest rate even though the learning rate changes from fastest to slowest.

ii. Activation Function

Activation function is the function that converts the linear output generated from convolutional layer to a non-linear result. For example, convolutional layer generates serious of values from 0 to 10, activation function basically says: “everyone above 5 belongs to class ‘yes’, and the rest belongs to the class ‘no’.” Then, all values are classified into two complimentary classes.

In almost all of the convolutional neural network, ReLU (Rectified Linear Unit) is used as the activation function. Of course, compare to other activation function, such as Sigmod, ReLU is popular for a reason: ReLU can reduce the likelihood of vanishing gradient. Before further explains these two benefits, recall that the definition of a ReLU is $h = \max(0, a)$ where $a = Wx + b$. For the reason mentioned above, when $a > 0$, the gradient has a constant value. On the contrary, the gradient of Sigmoid becomes increasingly small as the absolute value of x increases. Therefore, the constant gradient of ReLU results in faster learning. Meanwhile, this characteristic also leads to less run time which no doubt can speed up the training process. Although ReLU can potentially cause "deader" cell due to its nature when $a < 0$, it is still the activation that closest to "perfect" so far.

iii. Pooling layer

Similar to convolutional layer, pooling layer is also responsible for reducing the spatial size of the features extracted from the input matrix. The benefit that pooling layer brings is reducing the cost of the resource. More specifically, the most straightforward purpose of pooling layer is extracting domain features within part of the input matrix. There are many ways to achieve this purpose. The most popular method is called Max Pooling. Basically, Max Pooling takes the max value from each partition of the general input and re-construct a matrix that contains only the max value extracted. Below is a figure [15] that shows how it works:

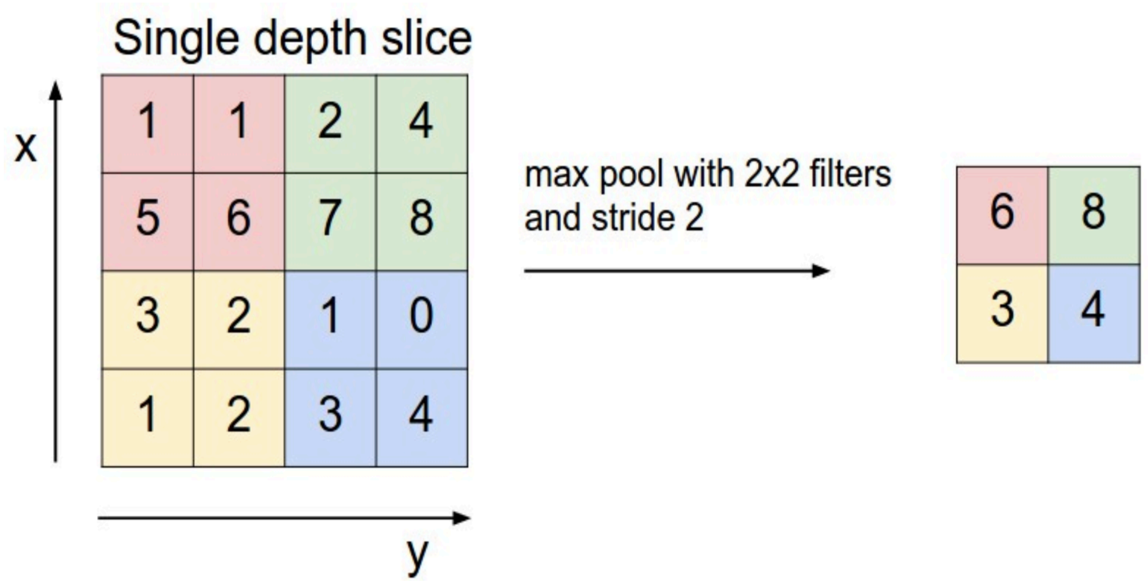


Figure 7: Max Pooling

Above is a model with stride equals to 2 and 2*2 filters, the result may vary based on different combination of stride and dimension of the filter. In this project, max pooling is also the first choice that I tried, and it performs well. However, the performance of it is not as “perfect” as I expect. Therefore, I constantly change the sample data and end up finding out the problem is the overlapping between vehicles. When two vehicles overlapped in an image, the selection of major features will ignore partial or even all of one vehicle. This leads to high standard deviation depends on the partition of the dataset.

In order to solve this problem, another type of pooling layer, SoftMax is introduced. SoftMax still classify features based on the input, except that it assigns each feature a confidence level rather than takes only the max value. This means that SoftMax, compare to Max, change from the top feature extracted from a specific region to the most possible top feature extracted from a specific region. A threshold is set, and features that have confidence levels higher than the threshold is counted, the possibilities of all qualified features are added. All features that have confidence level larger than the threshold can be considered as a "liked" one, and the rest are features that the model does not like in this image. Below is an example [16] of SoftMax:

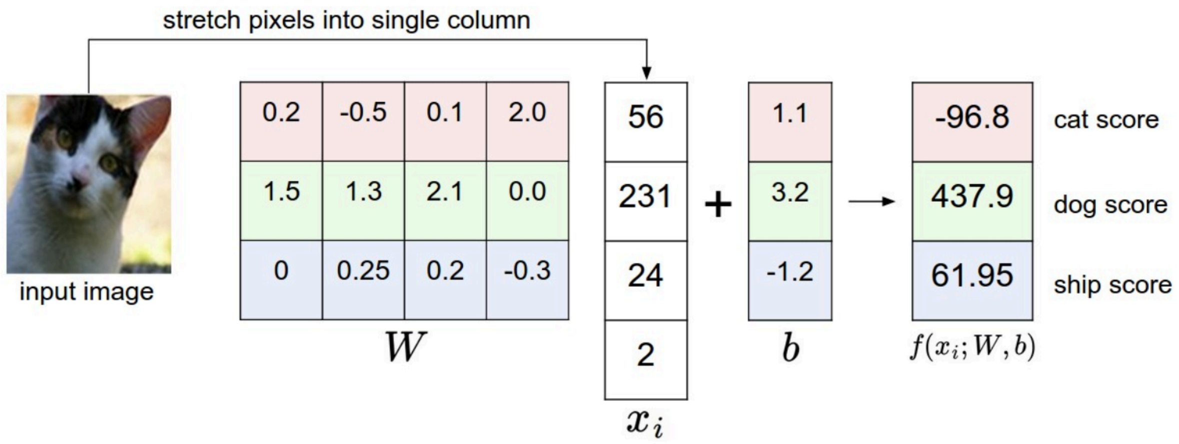


Figure 8: SoftMax

In this example, image pixels are stretched into a column and a matrix multiplication is performed to get the score for each class. Then the weight of each weight is adjusted based on the score obtained.

However, nothing is perfect, SoftMax spends more resources since it may keep track of multiple classes instead of only one max value Thus, use SoftMax thoroughly is also not wise. My method is using Max after the first convolutional layer and activation function and uses SoftMax after the fully connected layer. This strategy, in my opinion, combines the strength of two pooling functions and set off the disadvantage of them.

iv. Final CNN Model

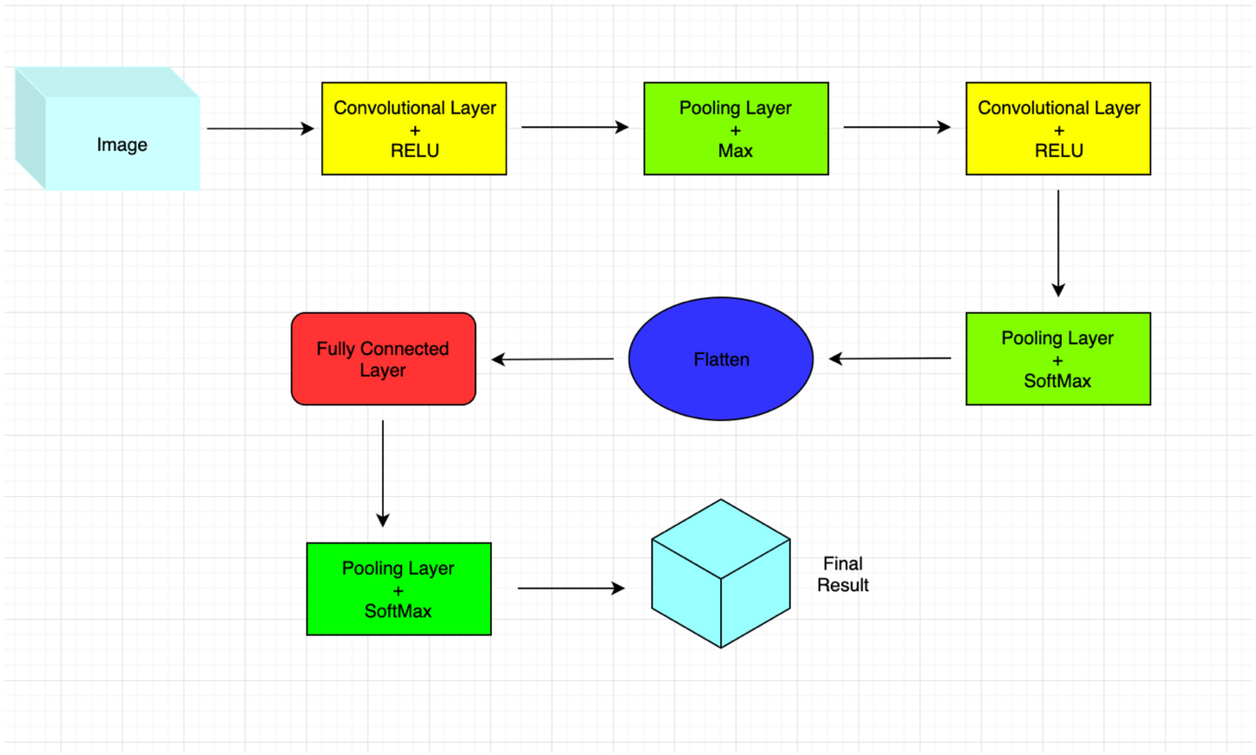


Figure 9: Final CNN Model

Figure 8 is my final CNN model. As shown in the figure, this is a 7-layer convolutional neural network model. First four layers are convolutional layer with RELU and pooling layer using Max & SoftMax. Then, a flatten layer is implemented to convert extracted features to classes. Last but not least, a fully connected layer is implemented to group pixels into different classes, and additional pooling layer is implemented to finish the final filtering.

c. Spark Cluster Configuration

After optimizing the model itself, a more important and practical question is speed. As introduced above, the DL model is way more complicated than traditional ML classifiers, so it takes much more time to get the result. In fact, before I implemented any optimization, the training of the model in this project takes over 10 hours to get a result with accuracy > 90%, which is not acceptable. Thus, it is necessary to find a way to accelerate the training process. Spark cluster is the answer I found. It provides a distributed environment that allows the model being trained in parallel. It is a great solution that efficiently allocates hardware resources to improve the performance of software. Below is a figure [18] that show the structure of a Spark cluster.

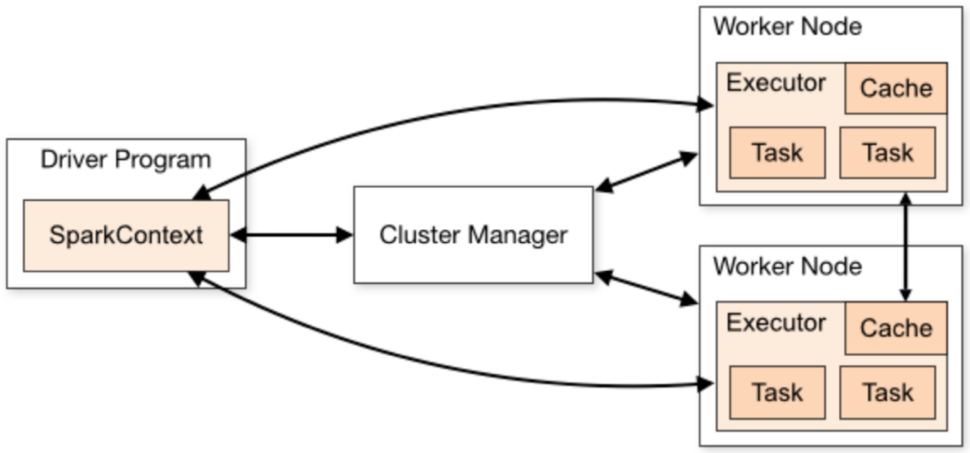


Figure 10: Apache Spark Cluster Mode

One important challenge of deploying CNN in a Spark cluster is that it converts the entire training project into a series of independent sets of processes and run all of the jobs on each worker node concurrently. Therefore, how to manage these nodes so that they can work more efficiently on CNN training tasks is the core problem when I configure the spark cluster.

i. Four Modes

So far, the definition of the distributed cluster is still ambiguous. In other words, there are many ways to construct a distributed Spark cluster and many of them have been proved to be good choices in specific fields. In other to compare and find the best mode for this project, the four most popular modes of Spark are implemented:

- Single server – CPU
- Single server – GPU
- Single server – Multi-GPUs
- Multi server – Single GPU per Server

ii. Memory Layout

In Spark, each worker node has physical resources to use, like CPU, GPU, Memory, etc. Memory is one of the crucial components that directly relate to the performance of each worker node. Due to the consideration of fault tolerance, partial memory is restricted to use while an application is running on the node because they are preserved for failure recovery. In Spark, the size of this type of memory can be adjusted, which means I can choose how to partition the memory and find a balance between performance and fault tolerance. DeepLearning4j sets a boundary for the adjustable memory, which is 3.5 GB / 4 GB, so I choose types of memory Layout: 3 GB / 4GB, 3.25 GB / 4GB, 3.5 GB / 4GB, and compare the performance of them. Details about the result of the comparison and related analysis are provided in the next section.

iii. Cache Modes

Cache is the significant factor for Spark's fast performance. In Spark, or any similar data processing framework, a large amount of I/O operations cannot be avoided, and I/O operations usually cost lots of resources. Therefore, it is necessary to find the best mode of cache if you want the best performance. DeepLearning4J provides 3 modes of cache for users: default, *.cache()*, *.persist()*. Default is the moderate cache scheduling mode, it neither always persist to put a new file into the cache, nor always directly store it into the disk. *.cache()* is the most "aggressive" way to use cache, it stores the file whenever cache is available, and evict the file whenever there is not enough space*, .persist()* is the most "conservative" method, it uses cache as less as possible so that the integrity of the data can be protected as much as possible. The comparison result of three cache modes and related analysis is given in the next section.

iv. Shuffle Optimization

After monitoring and estimating the performance of each executor in the cluster. I found out that the write function in Shuffle operation takes a long running time and a large amount of memory. In Spark cluster, an executor will create a file to store the result for each task. No matter how large the result is, the size of this file is fixed. This brings a problem, in my project, based on the statistics I observed from the profiling tool, most results of tasks cannot fill the whole file. Therefore, a large amount of memory is waste, and the resources used for writing files into buffer are also wasted because the system actually does not need that much file to store results. Thus, by combining results from different tasks together and put it into the same file until the file is filled, I can significantly reduce the number of files need to be written into the buffer, and also memory reserved for creating files.

V. Performance Evaluation

a. Experiment Setup

i. Dataset

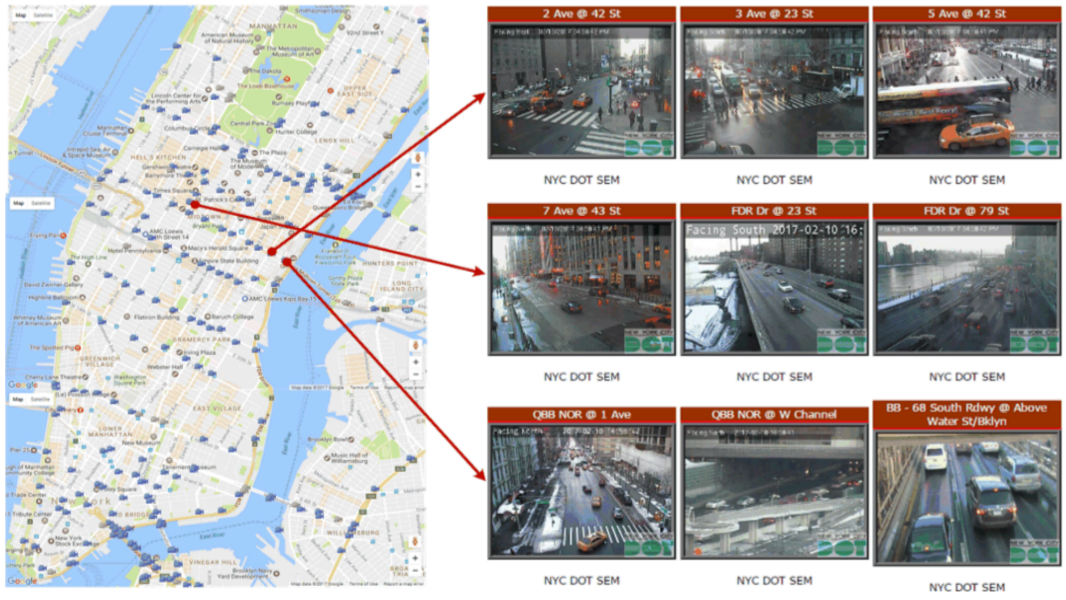


Figure 11: CityCam

The dataset that I used in this project is called CityCam [8], which is collected and organized by a team from Carnegie Mellon University. This dataset contains about 60,000 frames leading to 900,000 annotated objects. The resolution of each frame in the dataset is 352 * 240. All of the images are captured by cameras placed in New York City, and the total size of the dataset is 1.4 terabytes. Below is a figure [8] visualizing the distribution of cameras in NYC and how these cameras take pictures.

ii. Profiling Tool

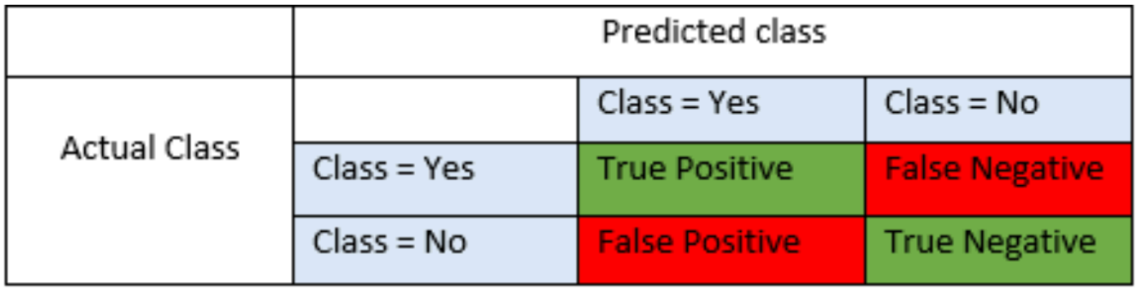
The profiling tool that I used to monitor and estimate the performance of Spark cluster is SparkLens [10]. SparkLens extend the Spark built-in monitoring tool WebUI and display more information. SparkLens can provides information such as:

- If the cluster used more cores than needed
- Running time and memory usage of each function
- Minimum memory required by each worker node

Based on the statistics I obtained by using SparkLens, I noticed that shuffle is the operation that wastes large amount of time and memory. Specifically, the write function in shuffle is the core of the problem as I mentioned above.

iii. Four measures

The dataset is divided into two parts: training data and test data. Training dataset contains 48,000 frames and test dataset contains 12,000 frames. In order to validate the test result, 5-fold cross validation is conducted during the test, which means, for example, as I mentioned above 1/5 of the whole dataset is test data, and cross-validation means that each time I randomly generate this 1/5 data as the test data, then evaluate it with the model trained by the rest 4/5 data and record the result. Then another 1/5 of the whole dataset is marked as test data and repeat the process again. The final result is based on the result that I record for 5 times.



		Predicted class	
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

Figure 12: Four Parameters

Above is a figure show four parameters [16]. Before I interpret the performance, it is necessary to explain four parameters that are important for evaluating a DL model.

- True Positive: Correctly predicted positive values: both expected and actual classes are yes.
- True Negative: Correctly predicted negative values: both expected and actual classes are no.
- False Positive: Wrongly predicted positive value: expected yes but actually is no.

- False Negative: Wrongly predicted negative value: expected not but actually yes.

Accuracy is the most straightforward and intuitive performance measure: it is the ratio of correctly predicted observation to the total observations [17]. In this project, it equals to ratio of the correctly classified objects to all classified objects.

Accuracy = (TP+TN) / (TP + FP + FN + TN)

Precision is the ration of correctly predicted positive observations to the total predicted positive observations [15]. In this project, it equals to the ratio of correctly classified vehicles to all classified vehicles.

Precision = TP / (TP + FP)

Recall is relatively complicated, basically it is the ratio of correctly predicted positive observations to the all observations that is classified as “yes”. In this project, it equals to the ratio of correctly classified vehicles to the classified vehicles.

Recall = TP / (TP + FN)

F1 score is the most complicated performance measure. It is the weighted average of Precision and Recall [15]. For my project, it is more helpful than accuracy since the distribution of vehicles is uneven in the city.

F1 Score = 2 * (Recall * Precision) / (Recall + Precision)

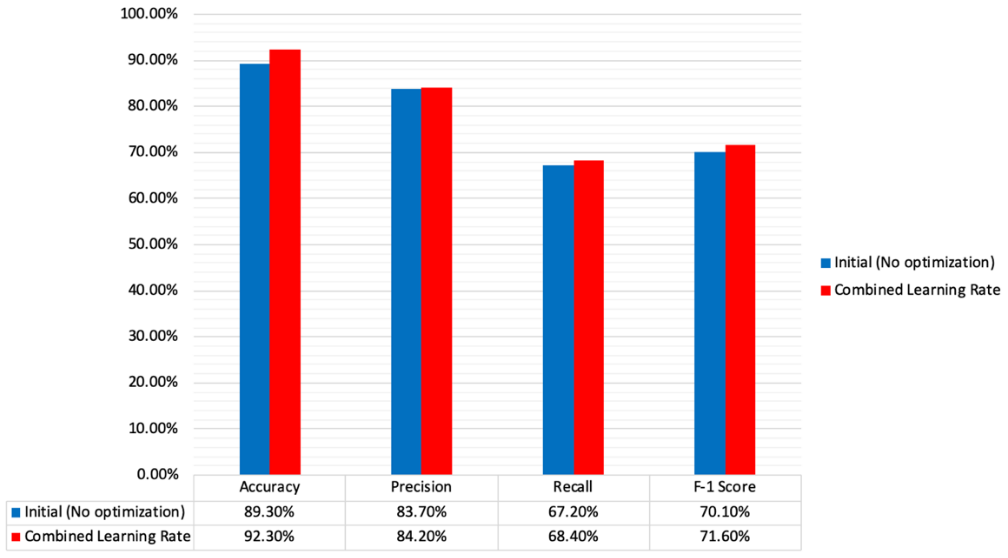
Above three parts are components that I think affect the performance of the model at most in this project after running a series of tests. Below are results that I got in the experiments

b. Results and Analysis

i. CNN Model Optimization

Below are figures that show the comparison between default constant learning rate and combined learning rate, the comparison between default Max Pooling layers and both Max and

SoftMax Pooling layers, and the comparison between the model with default setting and the model with optimization.



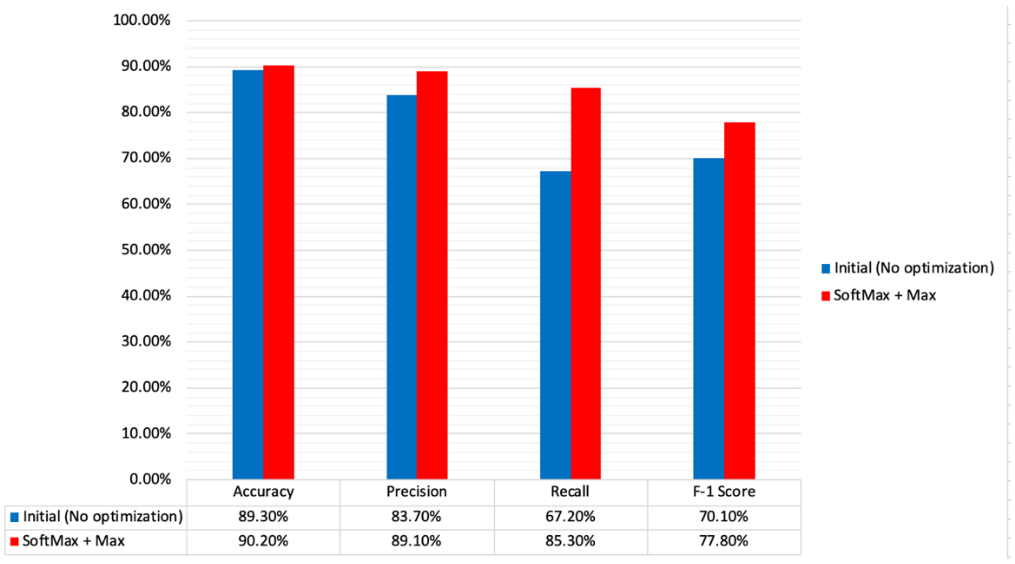
	Accuracy	Precision	Recall	F-1 Score
🟦 Initial (No optimization)	89.30%	83.70%	67.20%	70.10%
🟥 Combined Learning Rate	92.30%	84.20%	68.40%	71.60%

Figure 13: Prediction Result: Combined Learning Rate

	Default-Average	Default-Std	Combined Learning Rate-Average	Combined Learning Rate-Std	Improved-Average	Increased-Std
Accuracy	0.8930	0.0074	0.9230	0.0081	3.3%	9.4%
Precision	0.8370	0.0043	0.8420	0.0044	0.12%	2.3%
Recall	0.6720	0.0132	0.6840	0.0127	1.8%	-3.8%
F-1 Score	0.7010	0.0142	0.7160	0.0123	2.1%	-13.4%

TABLE I: 5-Fold Cross Validation: Default vs Combined Learning Rate

Figure 12 and Table 1 show the comparison between default (constant) learning rate and combined learning rate. Note that all the performance metrics are improved, among which accuracy is improved most significantly after implementing combined learning rates.



	Accuracy	Precision	Recall	F-1 Score
🟦 Initial (No optimization)	89.30%	83.70%	67.20%	70.10%
🟥 SoftMax + Max	90.20%	89.10%	85.30%	77.80%

Figure 14: Prediction Result: SoftMax and Max

	Default-Average	Default-Std	SoftMax&Max-Average	SoftMax&Max-Std	Improved-Average	Increased-Std
Accuracy	0.8930	0.0074	0.9020	0.0131	1.1%	77%
Precision	0.8370	0.0043	0.8910	0.0229	6.4%	433%
Recall	0.6720	0.132	0.8530	0.0352	27%	167%
F-1 Score	0.7010	0.142	0.7780	0.0296	11%	108%

TABLE II: 5-Fold Cross Validation: Default vs SoftMax&Max

Figure 13 and Table II show the comparison between default (constant) pooling layer using only Max and the optimized pooling layer using both Max and SoftMax (as described in Section IV.B.3). Note that all the performance metrics are improved, among which Precision, Recall, and F- Score all improved significantly with optimized pooling layer.

The performance of the model after and before optimization on above three components is displayed in following figure:

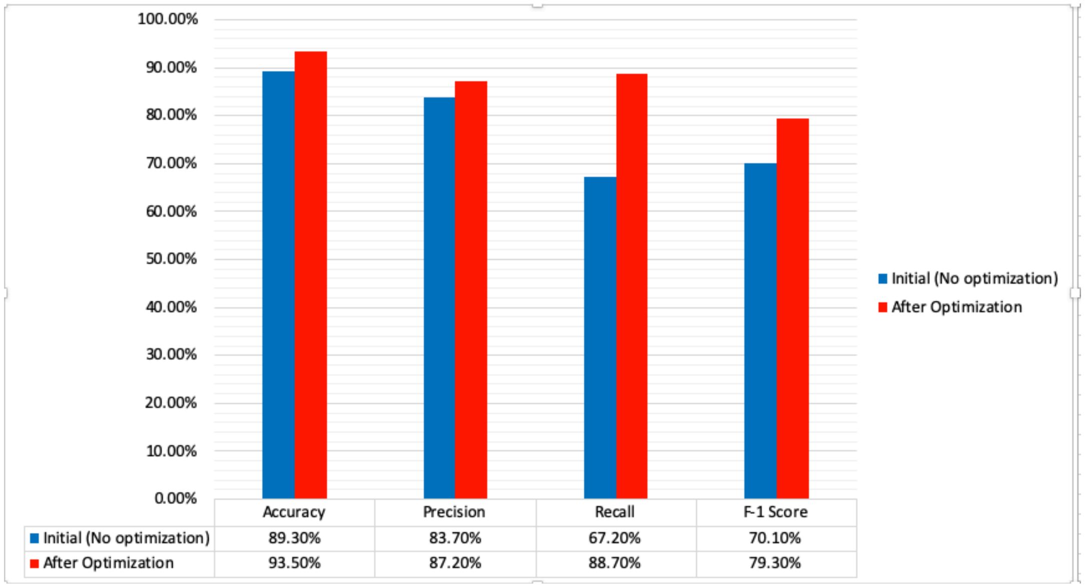


Figure 15: Prediction Result: Before & After Optimization

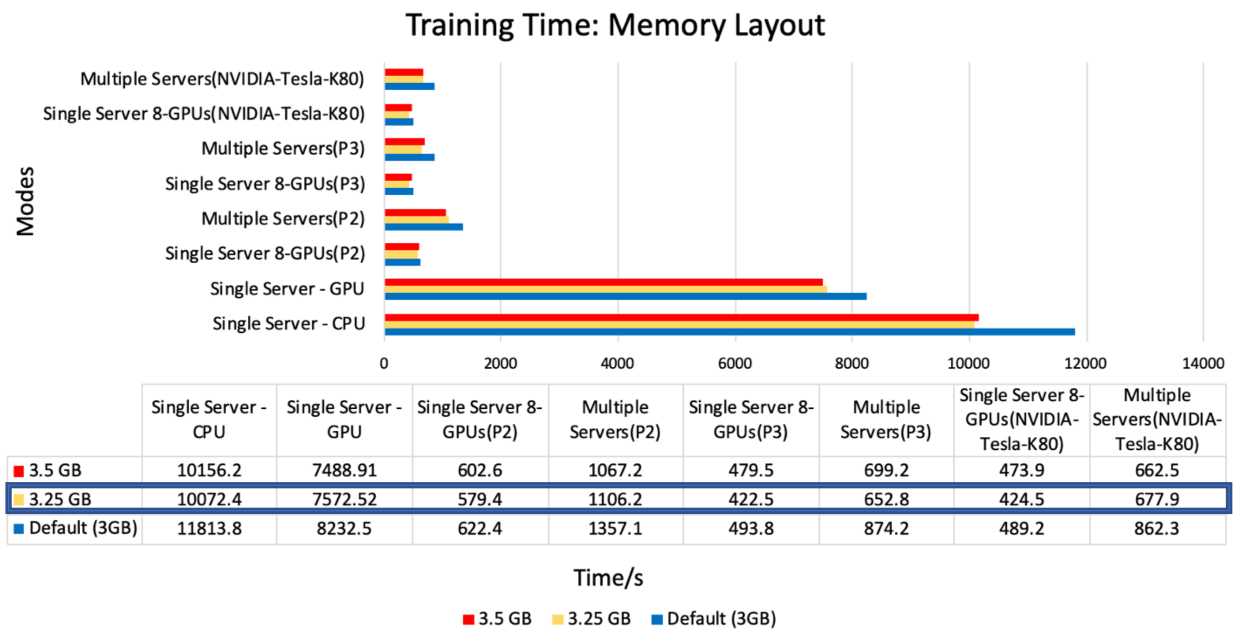
	Default-Average	Default-Std	Optimization-Average	Optimization-Std	Improved-Average	Increased-Std
Accuracy	0.8930	0.0074	0.9350	0.0092	4.7%	24%
Precision	0.8370	0.0043	0.8720	0.0077	4.2%	79%
Recall	0.6720	0.0132	0.8870	0.0212	32%	61%
F-1 Score	0.7010	0.0142	0.7930	0.0179	13%	25%

TABLE III: 5-Fold Cross Validation: Default vs Optimization

Based on the test result, I can tell that my optimization is successful. Generally, all four measures are better after optimized the model. Specifically, the improvement on Recall and F1 Score is great, even though the improvement on Accuracy and Precision is alright. I believe introducing SoftMax combining with Max is the most significant reason improve the Recall of the model. The combination of different learning rate contributes a lot on the improvement of F1 Score.

ii. Spark Cluster Tuning

Below are figures that show the comparison between three types of memory distribution and the comparison between three types of cache modes. Each comparison is based on the four distributed modes that I mentioned in the previous section.



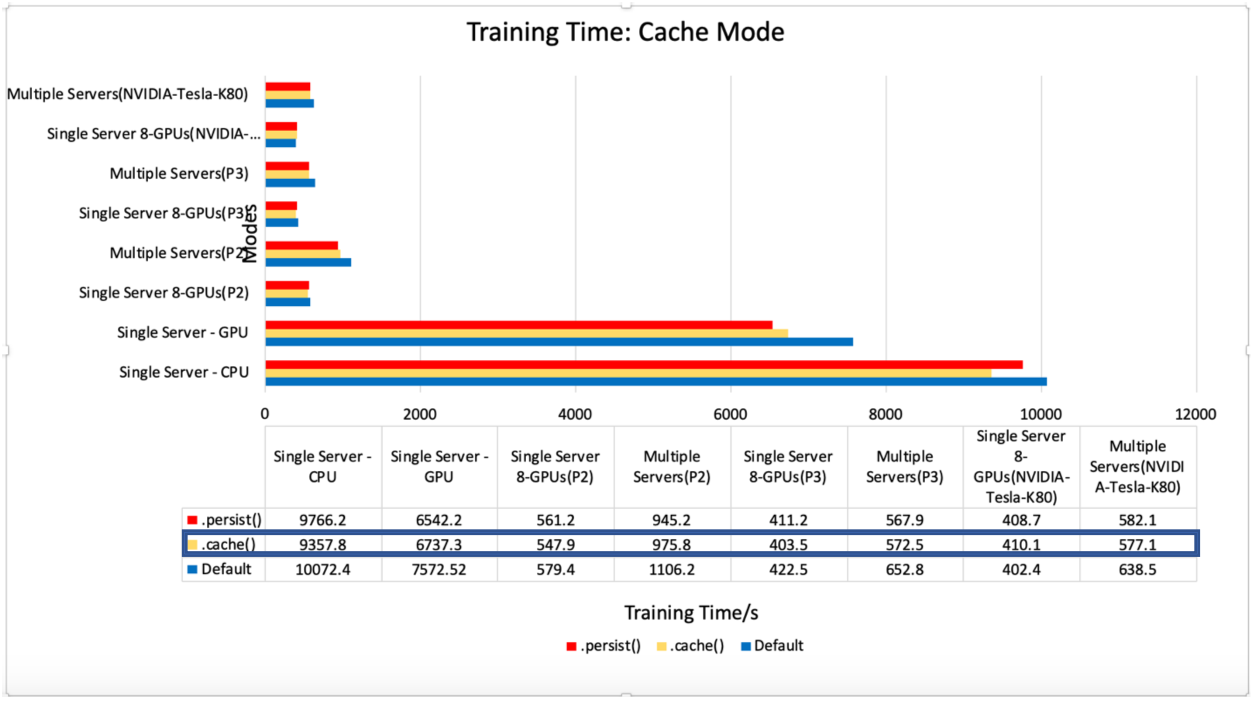
	Single Server - CPU	Single Server - GPU	Single Server 8-GPUs(P2)	Multiple Servers(P2)	Single Server 8-GPUs(P3)	Multiple Servers(P3)	Single Server 8-GPUs(NVIDIA-Tesla-K80)	Multiple Servers(NVIDIA-Tesla-K80)
🟥 3.5 GB	10156.2	7488.91	602.6	1067.2	479.5	699.2	473.9	662.5
🟨 3.25 GB	10072.4	7572.52	579.4	1106.2	422.5	652.8	424.5	677.9
🟦 Default (3GB)	11813.8	8232.5	622.4	1357.1	493.8	874.2	489.2	862.3

Figure 16: Training Time between Three Types of Memory Layout

	3.5 GB-Average	3.5 GB-Std	3.25 GB-Average	3.25 GB-Std	3 GB-Average	3 GB-Std
Training Time	479.5s	22.3s	422.5s	20.1s	493.8s	21.8s

TABLE IV: 5-Fold Cross Validation: Training Time of 3 Memory Layout (Single Server – 8 GPUs, P3 instance)

Figure 15 and Table IV show the cost of time among three different memory layout. Note that 3.25 GB/4GB (yellow lines) has the best performance in all four distributed environments.



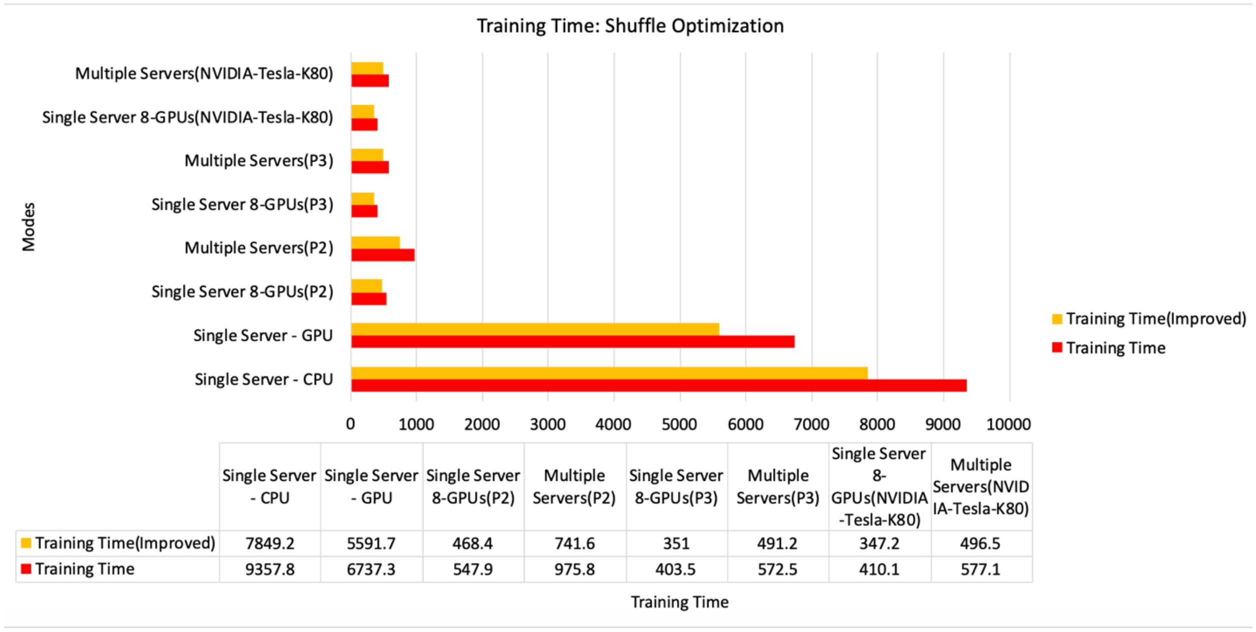
	Single Server - CPU	Single Server - GPU	Single Server 8-GPUs(P2)	Multiple Servers(P2)	Single Server 8-GPUs(P3)	Multiple Servers(P3)	Single Server 8-GPUs(NVIDIA-Tesla-K80)	Multiple Servers(NVIDIA-Tesla-K80)
🟥 .persist()	9766.2	6542.2	561.2	945.2	411.2	567.9	408.7	582.1
🟨 .cache()	9357.8	6737.3	547.9	975.8	403.5	572.5	410.1	577.1
🟦 Default	10072.4	7572.52	579.4	1106.2	422.5	652.8	402.4	638.5

Figure 17: Training Time between Three Types of Cache Modes

	.persist()-Average	*.persist()*-Std	*.cache()*-Average	*.cache()*-Std	default-Average	default-Std
Training Time	411.2s	19.2s	403.5s	21.3s	422.5s	22.6s

TABLE V: 5-Fold Cross Validation: Training Time of 3 Cache Modes (Single Server – 8 GPUs, P3 instance)

Figure 16 and Table V show the cost of time among three modes of cache. Note that the aggressive *.cache()* is better than the most conservative *.persist()*, while both improved over the moderate *default* setting.



	Single Server - CPU	Single Server - GPU	Single Server 8-GPUs(P2)	Multiple Servers(P2)	Single Server 8-GPUs(P3)	Multiple Servers(P3)	Single Server 8-GPUs(NVIDIA-Tesla-K80)	Multiple Servers(NVIDIA-Tesla-K80)
🟧 Training Time(Improved)	7849.2	5591.7	468.4	741.6	351	491.2	347.2	496.5
🟥 Training Time	9357.8	6737.3	547.9	975.8	403.5	572.5	410.1	577.1

Figure 18: Training Time Before and After Shuffle Optimization

	Before Optimization - Avg	Before Optimization - SD	After Optimization - Avg	After Optimization - SD
Training Time	403.5s	21.3s	351.0s	19.2s

TABLE VI: 5-Fold Cross Validation: Training Time of 3 Memory Distributions (Single Server – 8 GPUs, P3 instance)

	Single server – CPU	Single server – GPU	Multi server – Single GPU per Server (AWS P2)	Multi server – Single GPU per Server (AWS P3)	Multi server – Single GPU per Server (Google Cloud-K80)	Single server – Multi-GPUs (AWS P2)	Single server – Multi-GPUs (AWS P3):	Single server – Multi-GPUs(Google Cloud-K80)
Before	11813.8s	8232.5s	1357.1s	874.2s	862.3	622.4s	493.5s	489.2
After	7849.2s	5591.7s	741.6s	491.2s	496.5s	468.4s	351.0s	347.2s
Reduced	34%	32%	45%	44%	43%	26%	29%	29%

TABLE VII: 5-Fold Cross Validation: General improvement on Training Time

The test result shows that my optimization on shuffle makes a great improvement on the training time. This result is based on the configuration with best performance: 3.25 GB / 0.75 GB + *.cache()* + shuffle optimization.

iii. Summary of Performance Evaluation

In this section, I compare accuracy and cost of training time of CNN before and after using Spark. First of all, note that before using Spark, the results shown in Section III, Figures 1 and 2, are for 5,000 frames, while using Spark the results shown in Section V are for 60,000 frames. Next, for accuracy results, comparing those in Figure 1 with those in Tables I, II, and III, I can see that the accuracy remains high, while all the performance metrics (including precision, recall, and F1 score) are significantly higher after optimization methods are applied. Lastly, comparing cost of training time shown in Figure 2 and those in Tables IV and V, the time needed is much reduced, from the range of 837-3252 seconds to 423-494 when using CNN, and down to 404-423 seconds when applying optimization methods.

VI. Conclusion

Based on the result presented in the performance evaluation section, my improvement solution reaches the expectation that I made before starting this project. It improves both the prediction accuracy and the time cost of training. By deploying the training of the model in a distributed environment, the training time has been significantly decreased while the prediction result is still kept at a high level.

However, the improvement at the time cost is not sufficient. Or, not perfect yet. In other words, there is more potential to explore. Meanwhile, the experiment result shows that the improvement that I obtained currently is not stable based on the standard deviation that I got from 5-fold cross-validation. Therefore, in the future, I expect to run more experiments to validate the progress that I obtained so far.

VII. Reference

[1] "Apache spark - lightning-fast cluster computing", [online] Available: https://spark.apache.org/. , Accessed on May 2019.
[2] D. Manzi and D. Tompkins, "Exploring GPU Acceleration of Apache Spark," 2016 IEEE International Conference on Cloud Engineering (IC2E), Berlin, 2016, pp. 222-223.
[3] Andrew L.Beam, “Deep Learning 101 – Part1: History and Background”, [online] Available: https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html , Accessed on May 2019.
[4] Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Hasan, M., Van Esesn, B. C., Awwal, A. A. S., and Asari, V. K. (2018). The history began from alexnet: A comprehensive survey on deep learning approaches. arXiv preprint arXiv:1803.01164.
[5] Fukushima, K. (2007). "Neocognitron". Scholarpedia. 2 (1): 1717.
[6] Madhukar, “History of Apache Spark: Journey from Academia to Industry”, [online] Available: http://blog.madhukaraphatak.com/history-of-spark/ , Accessed on May 2019.
[7] W. Zhang, B. Xue, J. Zhou, X. Liu and H. Lv, "A Scalable and Efficient Multi-Label CNN-Based License Plate Recognition on Spark," 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), Guangzhou, 2018, pp. 1738-1744.
[8] CityCam, [online] Available: https://www.citycam-cmu.com/ , Accessed on May 2019.
[9] H. Gülaçar, Y. Yaslan and S. F. Oktuğ, "Short term traffic speed prediction using different feature sets and sensor clusters," NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, Istanbul, 2016, pp. 1265-1268.
[10] Y. Liu and H. Wu, "Prediction of Road Traffic Congestion Based on Random Forest," 2017 10th International Symposium on Computational Intelligence and Design (ISCID), Hangzhou, 2017, pp. 361-364.

[11] M. Duan, "Short-Time Prediction of Traffic Flow Based on PSO Optimized SVM," 2018 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS), Xiamen, 2018, pp. 41-45.
[12] "Radial basis function Kernel", Wikipedia, [online], Available: https://en.wikipedia.org/wiki/Radial_basis_fun ction_kernel , Accessed on May 2019.
[13] Nearest Neighbor Classification, Scikit-Learn v0.20.3 documentation, [online], Available: https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py , Accessed on May 2019.
[14] Hafidz Zulkifli, "Understanding Learning Rates and How It Improves Performance in Deep Learning", Towards Data Science, Jan 21, 2018
[15] Harsh Pokharna, "The best explanation of Convolutional Neural Networks on the Internet!", [online], Available: https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8 , Accessed on May 2019.
[16] CS231n Convolutional Neural Networks for Visual Recogniztion, [online], Available: http://cs231n.github.io/linear-classify/ , Accessed on May 2019.
[17] Renuka Joshi, "Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures", [online], Available: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/ , Accessed on May 2019.
[18] Cluster Mode Overview, Apache Spark, [online], Available: https://spark.apache.org/docs/latest/cluster-overview.html , Accessed on May 2019.