Machine Learning Kaggle Competition Writeup

Yashas Manjunatha
CS 671D - Spring 2019

April 23, 2019

1 Exploratory Analysis

I began exploratory analysis of the Fashion MNIST dataset by attempting to visualize the data. First I needed to get a sense of the shape of the data and understand the classes present in the dataset. Then I wanted to get a sense what the 28x28 images looked like, so I reshaped the flattened data vector to the 28x28 image vector and used the imshow function from the matplotlib Python library to visualize the image. This helped me get a better understanding of the different items in the dataset. The image of the first data point in the dataset is pictured in Figure 1.

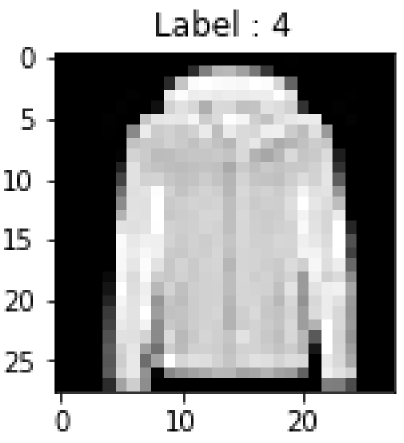

Figure 1: First Data Point Reshaped to 28x28 Vector and Visualized

Next, I used principal component analysis (PCA) to perform dimensionality reduction for visualization of the data. I used PCA to project from 784 to 2 dimensions. Then, I plotted the first two principal components of each data point to get Figure 2 where the color corresponds to the label of the point. Here I observed some separation between the categories, and especially in categories in the lower half of the figure, it looks like a linear separator would be capable of classifying the data, which immediately led to the first model idea I had to approach the problem.

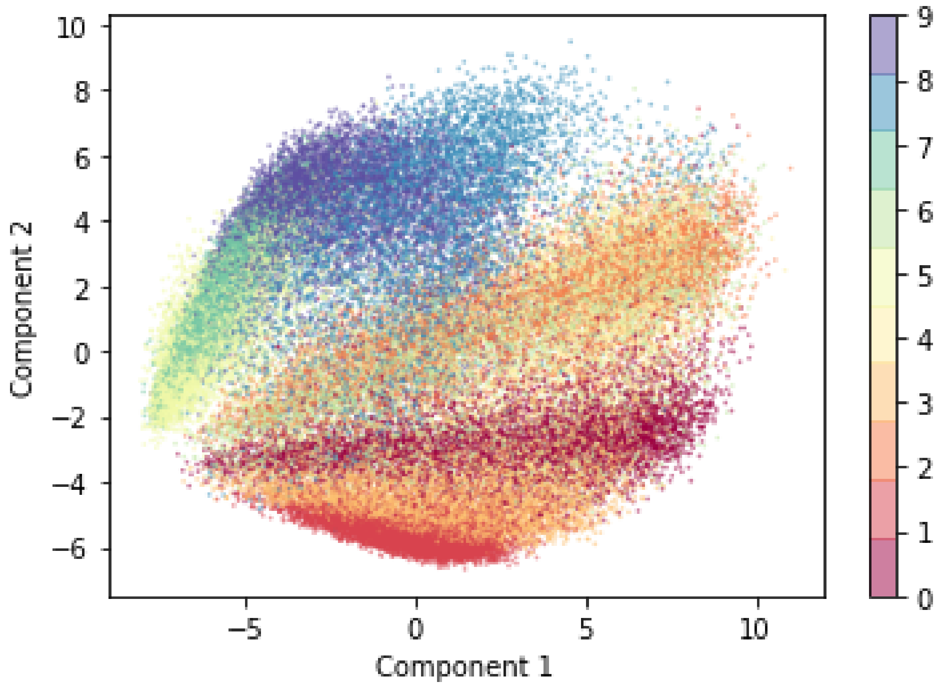

Figure 2: PCA of Dataset

2 Models

2.1 Support Vector Machines

From the PCA visualization, I observed that a linear separator might be a good start to classifying the data. So, I decided to first try support vector classification (SVC) with a linear kernel using the scikit-learn SVC module. This approach was further motivated by the promising comparison of the SVC with linear kernel classification performance in Figure 3 and PCA visualization of the dataset in Figure 2. However, since SVM solves an optimization problem of quadratic order this model was computationally expensive for this large dataset and required a high run time to train the model. So, I decided to move to an algorithm that would be less computationally expensive.

2.2 Random Forests

I landed on random forests because of its simple parameterization (number of trees), availability of a high-quality Python library (scikit-learn's RandomForestClassifier), and computational efficiency (i.e. faster training runtime). This approach also seemed promising with a comparison of random forest classification performance in Figure 4 and the PCA visualization of the dataset in Figure 2. Additionally, I believed random forests would be a powerful model because of its simplicity and flexibility, and it does not typically suffer from overfitting.

Citation of External Library:

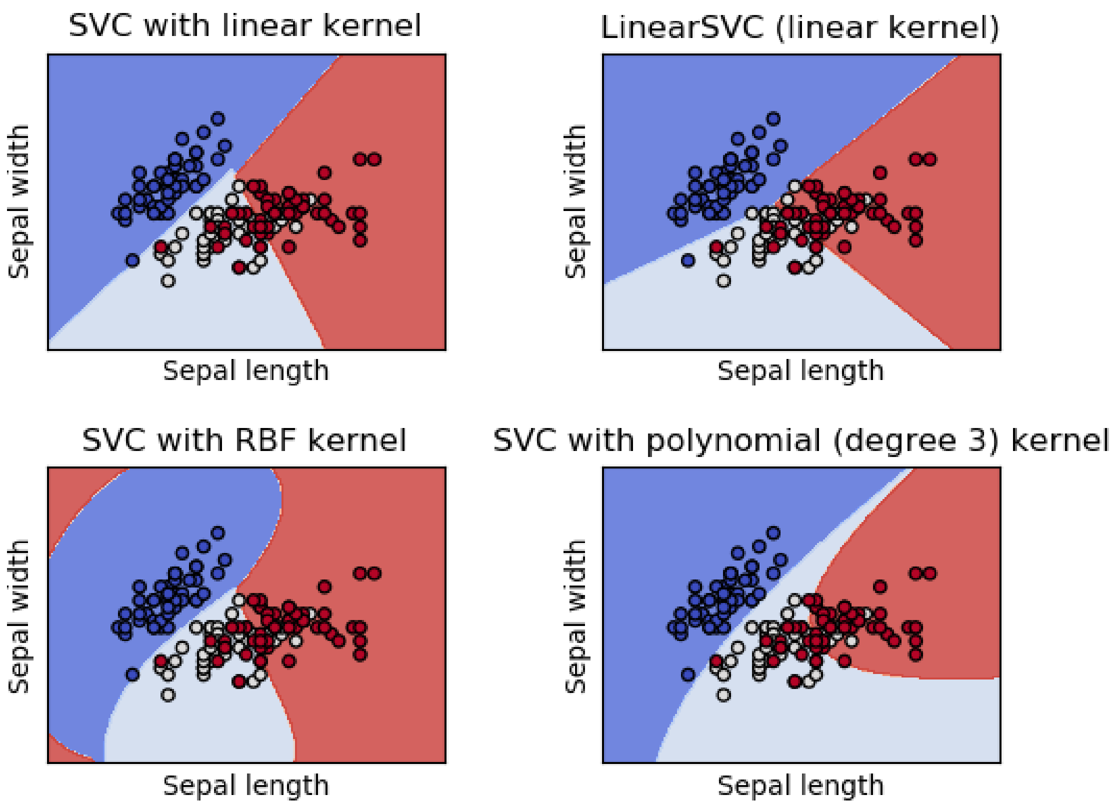

Figure 3: Visualization of SVC Algorithm; Figure adapted from https://scikit-learn.org/stable/modules/svm.html#classification

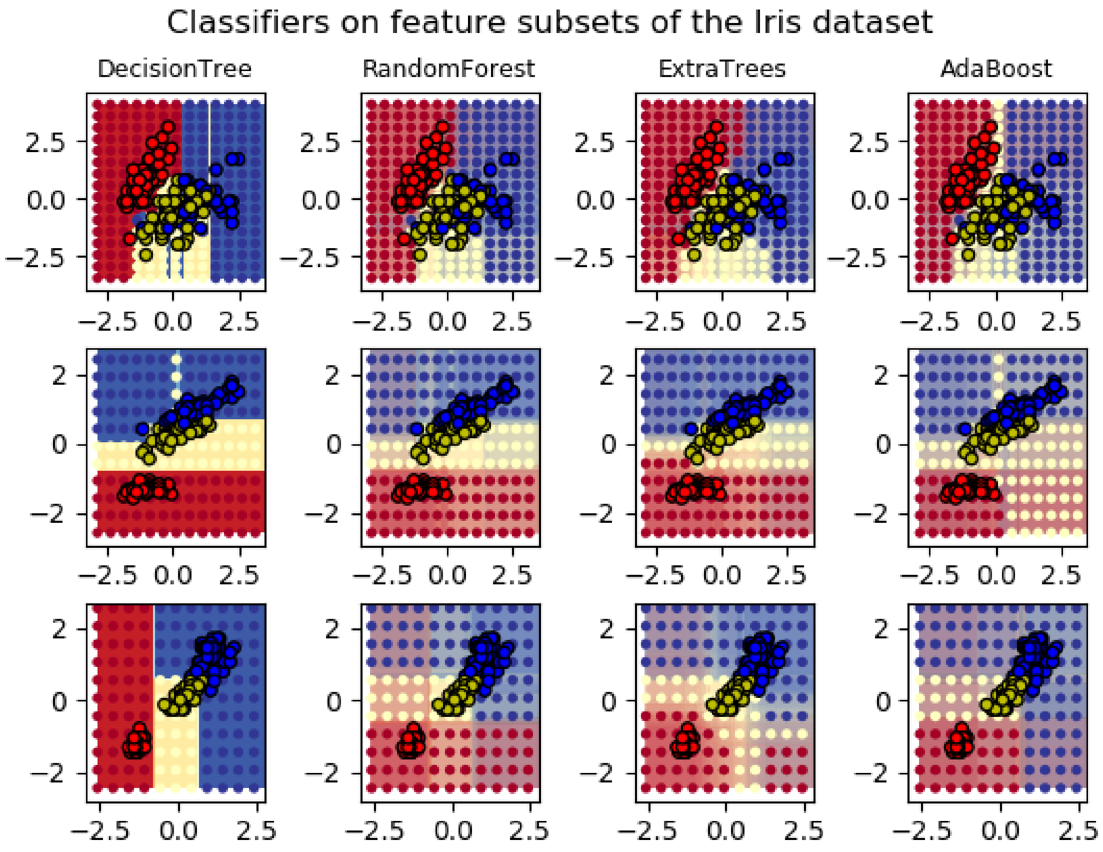

Figure 4: Visualization of RF Algorithm; Figure adapted from https://scikit-learn.org/stable/modules/ensemble.html#extremely-randomized-trees

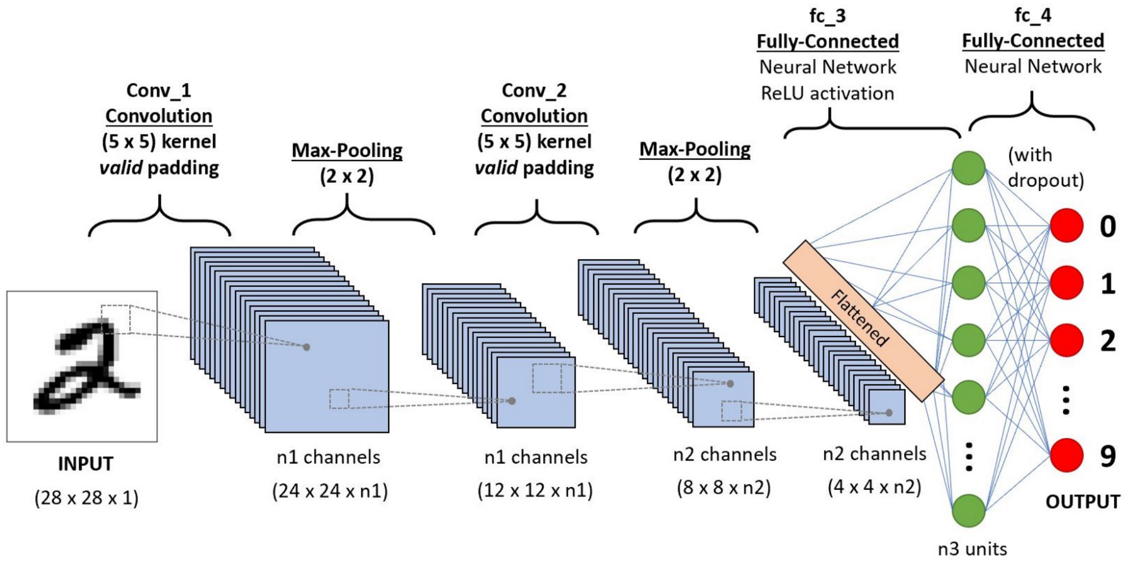

Figure 5: Visualization of CNN Algorithm; Figure adapted from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks\-the-eli5-way-3bd2b1164a53

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

2.3 Convolutional Neural Networks

Finally, the second algorithm I used for generating predictions was a convolutional neural network. I used this algorithm because convolutional neural network is a specialized neural network for processing data that has an input shape like a 2D matrix, which is useful for image data, and thus typically used for image recognition and classification, which is appropriate for this dataset. Figure 5 demonstrates a convolutional neural network's capabilities in processing image input.

Citation of External Library:
Keras, Chollet, François et al., GitHub, 2015. https://github.com/fchollet/keras

3 Training

3.1 Support Vector Machines

For linearly separable data, support vector machine tries to find two parallel hyperplanes such that the margin between two classes is maximized. Given training vectors $x_i \in \mathbb{R}^p$, $i = 1, , n$, in two classes, and a vector $y \in \{1, -1\}^n$, SVC solves the following primal problem:

$$\begin{aligned} \min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \\ \text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \\ \zeta_i \geq 0, i = 1, ..., n \end{aligned}$$

With a dual problem:

$$\begin{aligned} \min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\ \text{subject to } y^T \alpha = 0 \\ 0 \leq \alpha_i \leq C, i = 1, ..., n \end{aligned}$$

where $Q_{ij} \equiv y_i y_j K(x_i, x_j)$ and $K(x_i, x_j)$ is the kernel, and in the linear case, is $\langle x, x' \rangle$. As we also showed in Homework 4, this dual problem is a quadratic program, and thus solving this optimization problem was very computationally expensive on this dataset and run time to train a model on the dataset took on the order of hours to days, and thus was not conducive to quick hyperparameter tuning.

3.2 Random Forests

In the random forests algorithm, a diverse set of classifier trees are constructed by introducing randomness in the classifier construction and uses averaging to improve the predictive accuracy and control over-fitting. Specifically, each tree in the ensemble is built from a sample drawn with replacement (i.e. bootstrap) from the training data. And when splitting a node during the creation of the tree, the split that is picked is the best split among a random subset of the features. The scikit-learn implementation supports two criteria to measure the quality of a split: the Gini impurity and the information gain (entropy). Resulting from this randomness, the bias slightly increases, but because of averaging, the variance decreases, resulting in a powerful model. Additionally, the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class. Finally, this algorithm about 5 minutes of run time (wall time) to train the model on the given training dataset.

3.3 Convolutional Neural Networks

A convolutional neural network model takes in an input image, and assigns importance, learnable weights and biases, to various aspects in the image to differentiate separate classes. The architecture of a convolutional neural network is similar to that of neurons in the human brain and inspired by the organization of the visual cortex. The first layer is a convolutional layer used to extract the high-level features from the input image. This layer uses the Leaky ReLU activation function which helps the network learn non-linear decision boundaries. The next layer is the max-pooling layer used to decrease the computational expensiveness to process the data through dimensionality reduction, it also helps extracting dominant features

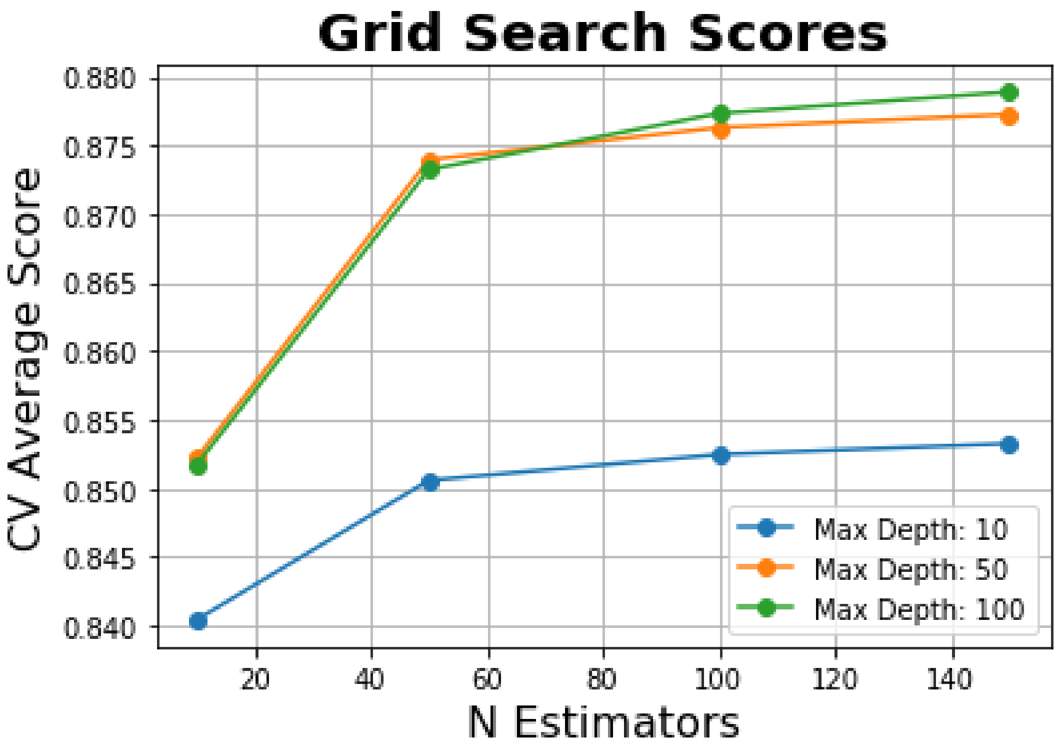

Figure 6: Grid Search Results from Hyperparameter Tuning on RF Model

from the image. Finally, the last layer is a dense layer with a softmax activation function with 10 units, which is needed for this multi-class classification problem. The fit function trains this generated model for a given number of epochs (or iterations on the dataset). It takes about 4 seconds for the each epoch to run, and since I used ran it for 20 epochs, it took a little over a minute for the model to train on the dataset.

4 Hyperparameter Selection

4.1 Random Forests

For the random forests model, I tuned the hyperparameters number of trees and max depth of trees by preforming a grid search over specified values for both the parameters and using a 3-fold cross validation to prevent overfitting on the training data. Figure 6 shows the results from the hyperparameter tuning, with optimal parameters on the plot of number of trees = 150 and max depth = 100. However, as we increase both the parameters the training time also increases. Thus, since the average predictive accuracy of the model was marginally increasing at this point in the tuning process, I decided to use the optimal parameters of number of trees = 150 and max depth = 100 for the final random forests model.

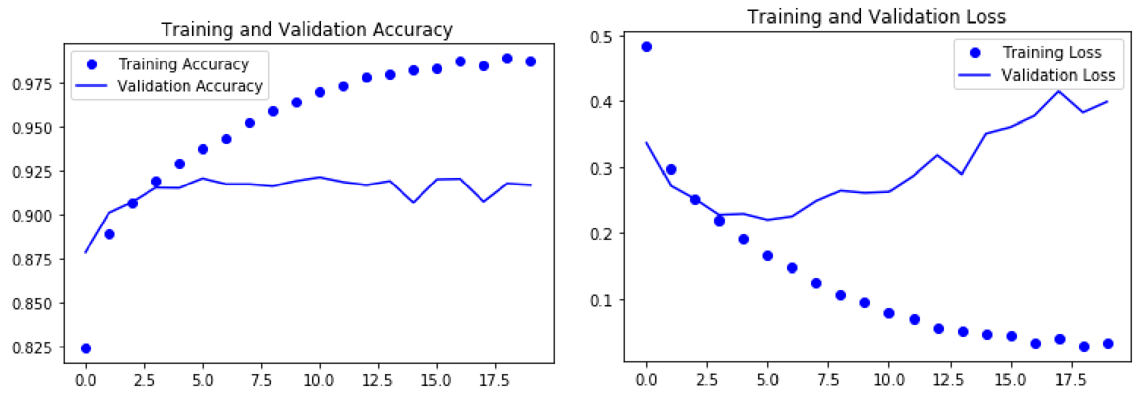

Figure 7: Results from Initial Convolutional Neural Network Model

4.2 Convolutional Neural Networks

For the convolutional neural network model, I tuned the hyperparameter of the number of neurons in the neural network architecture to reduce overfitting and to obtain a more optimal validation accuracy and validation loss. After training the initial convolutional neural network model with 3 layers, I immediately noticed overfitting in the model. As observed in Figure 7 the validation accuracy immediately stagnates after 3 to 4 epochs and the validation loss starts to increase after 3 to 4 epochs showing the signs of overfitting. Thus, I added a dropout layer to the network architecture. A dropout layer randomly turns off a fraction of neurons during the training process, which reduces dependency on the training data. The fractions of neurons turned off by the dropout layer depends on a hyperparameter, which I tuned to optimize validation accuracy. As observed in Figure 8, the optimal dropout rate was between 0.2 and 0.3, which is what I set the parameter to for my optimal model.

5 Data Splits

I used a train test split on the training data provided and used 80% of the data to train the models and the remaining 20% of the data to provide validation accuracy scores. This helped ensure that I did not overfit the data. Furthermore, in the hyperparameter tuning for the random forests model, I utilized a grid search that was evaluated with 3-fold cross validation to prevent overfitting. Finally, in the convolutional neural network model, I utilized a dropout layer which I tuned and evaluated on the validation test set to prevent overfitting on the training data.

6 Errors and Mistakes

One of the biggest challenges of this competition was piecing together all the different concepts we discussed in class to perform a holistic and systematic approach to a real world data science problem, but one that was really rewarding once I was able to apply all the techniques we learned in class to develop and improve a model. Additionally, I wanted to

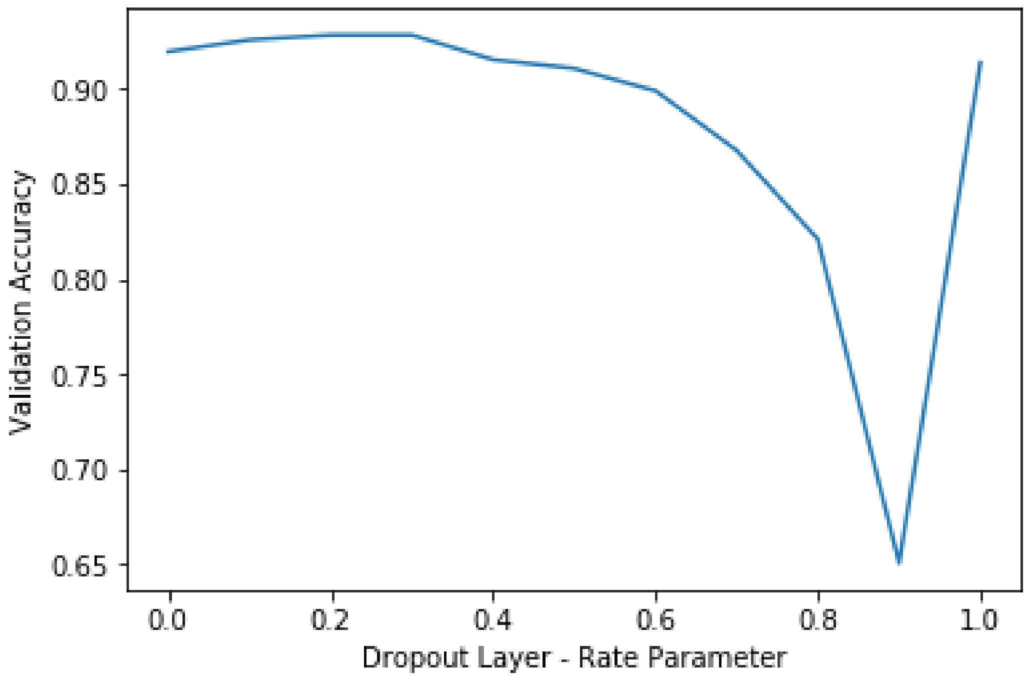

Figure 8: Relation between Predictive Accuracy and Dropout Rate

experiment with convolutional neural networks, so learning about and implementing that model was an additional challenge. A big misstep that I made was thinking that a model using linear separators would preform well on the dataset from the initial PCA visualization. Which led to the next misstep that really slowed my progress, which was attempting to use SVC for this dataset since the training time of that algorithm took quite a long time and I lost a significant amount of time attempting to tune the hyperparameter for the SVC model, especially since I accidentally closed the kernel multiple times and had to restart the process. Additionally, sometimes the algorithm would take so long to run I wasn't sure if there was a bug in my code or if that was just how long it took because it was computationally expensive.

7 Predictive Accuracy

My Kaggle username is Yashas Manjunatha. Figure 9 show the results of my models on the Kaggle competition dataset. While the random forest model had a score of 0.87133, my best performing model on the competition test dataset was the convolutional neural network model with a score of 0.91266 on the public test data. Figure 10 shows the training and validation accuracy results of the convolutional neural network model.

8 Code

My code is attached with the assignment submission as CS 671 HW #5.ipynb.

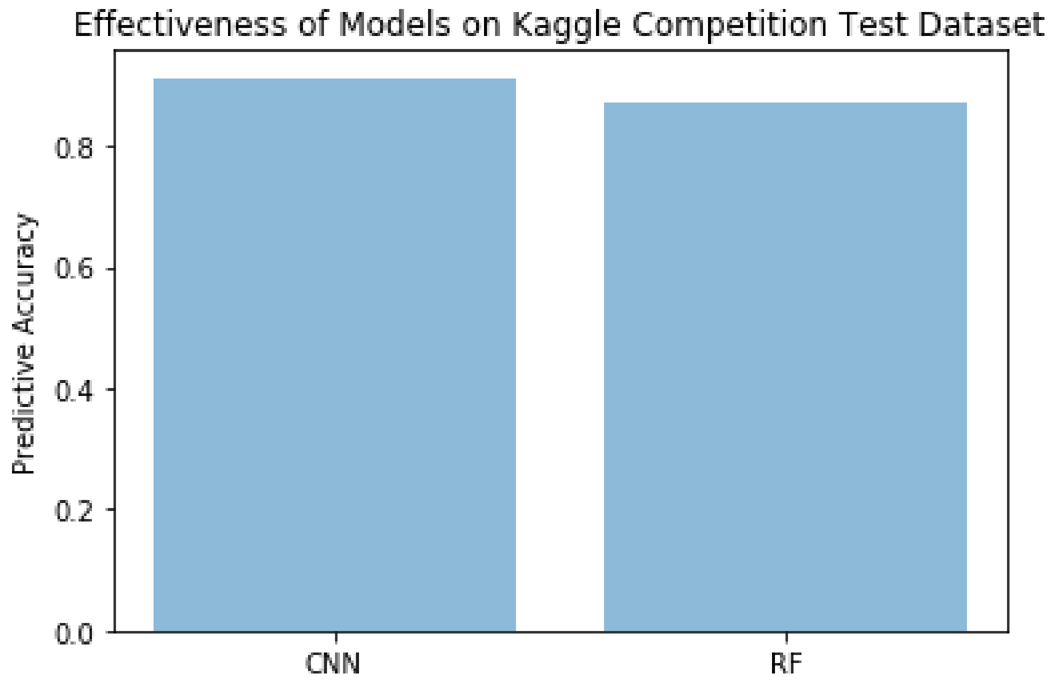

Figure 9: Results Kaggle Competition

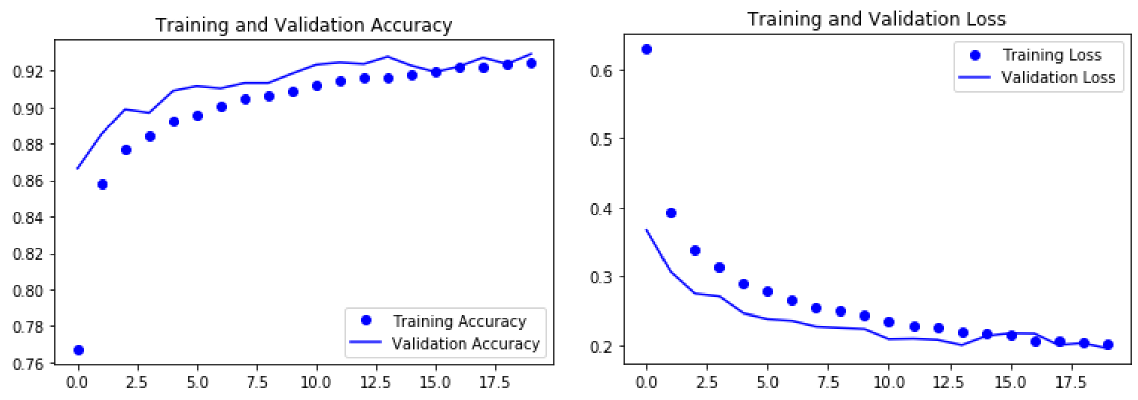

Figure 10: Results from Convolutional Neural Network Model with Tuned Dropout Layer