Team 14

MSA 8150 - Final Project Report

Gotham and Botanist

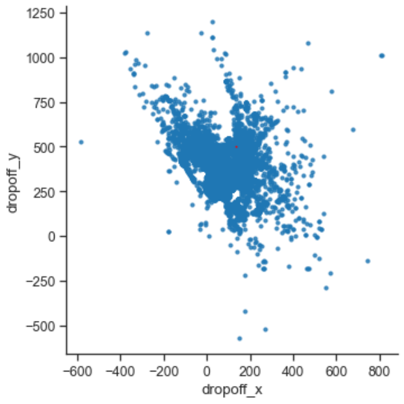
Type A - Gotham

Purpose:

Batman's butler, Alfred, is in need of our help to calculate a prediction time for taxi trips in order for Batman to get around for his missions. This regression problem aims to train a model to predict the travel time of a cab from one city to another. We are given a rich dataset of the recorded taxi trip durations between various parts of the city. The data includes *pickup_datetime, NumberOfPassengers, pickup_x, pickup_y, dropoff_x, and dropoff_y*. The columns provide information for which date and time of taxi rides, the number of passengers, as well as the x and y coordinates of where passengers are picked up and dropped off. We will be using the Train.csv file to fit our models. The results should then give us an additional column that predicts the quantity of *duration*.

Pre-Process Data and Model Setup:

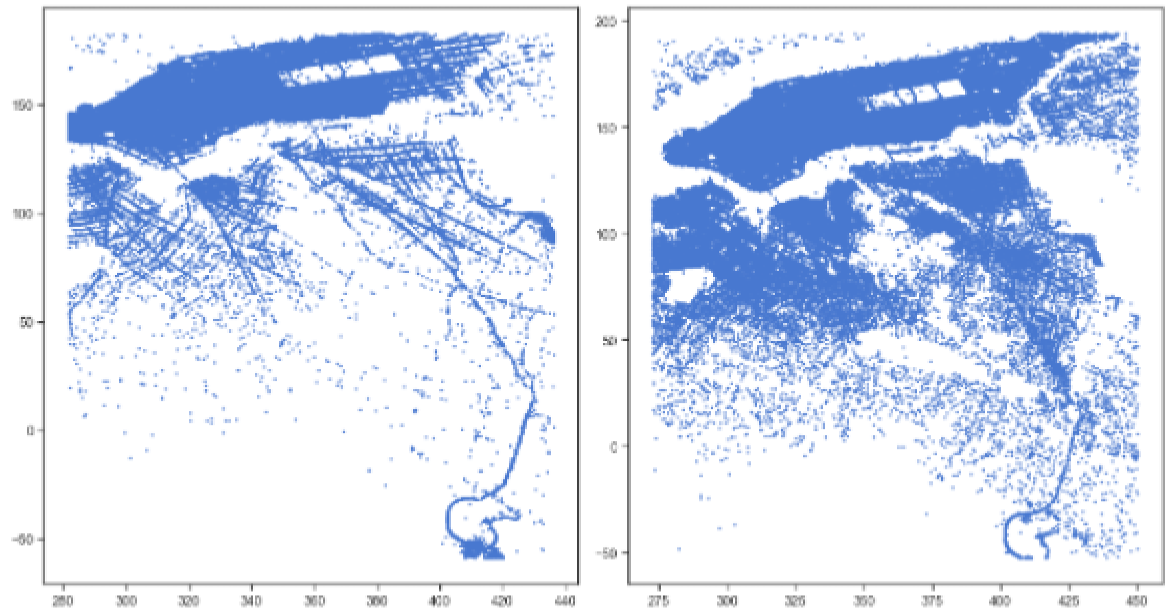
The first step of the data pre-processing is importing the package to build the plot and model. The essential packages to be imported are *datetime, matplotlib, sklearn, xgboost, seaborn, pandas, and numpy*. After importing the package, we make the scatter plot of dropoff and pickup spot to see how the data is distributed.



This is an example of the scatter plot of dropoff.

From this graph, we can see that this looks like many dots gathered into one big circle. If we reduce the data with a specific value amount of quantile, the dropoff and pickup scatter plot picture could show an image of New York City.

```
plot_lat_long(df,0.008)
```



After the data reduction with a quantile of 0.008, the dropoff and pickup scatter plot looks like New York City. We can notice that there are Manhattan, Brooklyn, and Flushing areas. After looking at these scatter plots, we decided to reduce data with the quantile of 0.008 for building and training the prediction model.

Before we built the model, we needed to create features that include euclidean distance, distance of X, and distance of Y. We want to train a model that is able to look at three types of distances rather than one. We did extract the date-time into a month, week, weekday, hour, minute, and minutes of the day. Doing so could help us with the prediction, and we want to see if these features would help the prediction of duration because we believe that time is a very crucial part of the duration. We build pickup and dropoff clusters by using the k-means method. This makes the pickup and dropoff location have many zones. It can help the model learn about how long the duration from pickup to dropoff place is by using the K-means cluster zone.

The last step is to make the model. Before we built the model, we decided to drop the features of pickup_x, pickup_y, dropoff_x, dropoff_y, and pickup datetime. Since we already breakdown these features into separate features, we don't use these features and it can lower the accuracy because of the redundancy. For the model, we try to use random forest, catboost, and xg boost.

Results:

Before we conclude the final result, we did try to add dummies variables for predicting the duration by using Random Forest and Catboost. The results with dummy variables did not give a great RMSE. Therefore we decided to do without the dummy variables, and using XG Boost provided a better result. We also did not use *np.log* function for the duration with XG Boost, and the RMSE is 267.94. After receiving feedback from the presentation, we attempted Catboost and Random Forest. The RMSE for Catboost is 267.65, and Random Forest is 279.24. Catboost is showing a slightly better result compared to XG Boost. Random Forest gives a higher RMSE score, but not a huge spike. In conclusion, either Catboost or XG Boost will provide the best result for the RMSE.

Type B - Botanist

Purpose:

This project aims to train a model with images of leaves and predict a label corresponding to the type of the plant. We are given 50000 images of different leaves and 38 classifications to make our predictions. The purpose is to help find if these leaves are meant to be the way they are or find any problems with them. We may not have a botanist in our back pocket at all times, but maybe we can recreate the mind of one via code. The classifications apple, blueberry, cherry, corn, grape, orange, peach, pepper, potato, raspberry, soybean, squash, strawberry, and tomato will be labeled numerically as 1 to 38.

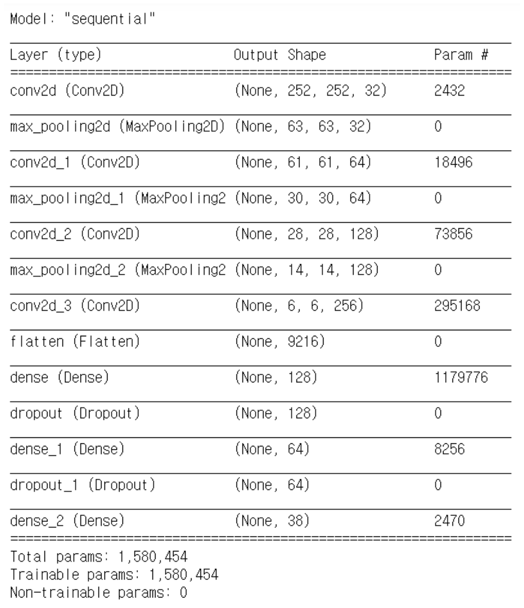
Image Data Pre-Process/Generator:

In order to make the most out of our training dataset, we augmented them via several random transformations. This step will help overfitting prevention and allow the model to generalize better. From Keras preprocessing image, we imported *ImageDataGenerator*. This class gave us the ability to configure the random transformations and normalize the operations during training. We used the inputs *wifth_shift_range, height_shift_range, rotation_range, rescale, validation_split, horizontal_flip,* and *vertical_flip*. These inputs helped with ensuring that every image was scaled/rotated/facing in the correct position and specifying that we will be splitting our training and validation dataset by 75-25.

The instantiate generators of the augmented images, and their labels are done via *.flow_from_dataframe*. We used the inputs: *dataframe, directory, x_col, y_col, class_mode, target_size, batch_size, validate_filenames,* and *subset*. We noticed that our images were considered non-validated images, so we needed to set the *validate_filnames = False* in order to read all 50000 images. Now we are able to split the data into two generators that read the input *subset = training* and *subset = validation*. These generators allow us to use the image data generator with the Keras model method inputs, such as *fit_generator, evaluate_generator,* and *predict_generator*.

Building a Model:

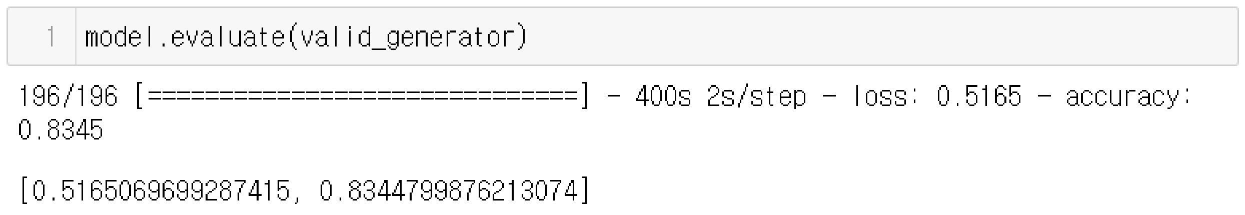
Then we build the convolutional model. Once the input layer is done, we added four convolutional 2D layers with ReLU in the sizes 32, 64, 128, and 256 used for the activation. The first three convolutional 2d layers includes a max-pooling 2d layer with the size of 4x4 for the first layer and 2x2 for the rest. Next we used flatten, dense, and dropout functions. The size of 128 and 64 with ReLU are used to activate the dense function. For the output layer, softmax is used as it enables us to identify various types of data classes.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 252, 252, 32)      2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 6, 6, 256)         295168
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 128)               1179776
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 38)                2470
=================================================================
Total params: 1,580,454
Trainable params: 1,580,454
Non-trainable params: 0
```

Results:

Prior to fitting the model, we compile the convolutional model with *Adam* as an optimizer, *categorical cross-entropy* as a loss function, and *accuracy* as a metric function. The 10 epochs we used helps the model with better data learning and calculation. We tried to run a model with 15 epochs because it could also improve or worsen the model's accuracy. The accuracy is almost the same as 10 epochs. We figure that 10 epochs already give a great accuracy – 83.45% accuracy with the test dataset. Therefore, we keep it for future model tests. I would suggest using more convolutional layers since it would allow the model to learn more about the image dataset and to read granular details of the images. Also, we recommend using more epochs because it allowed the model to learn more about the images.



```
model.evaluate(valid_generator)
```

```
196/196 [==============================] - 400s 2s/step - loss: 0.5165 - accuracy:
0.8345

[0.5165069699287415, 0.8344799876213074]
```