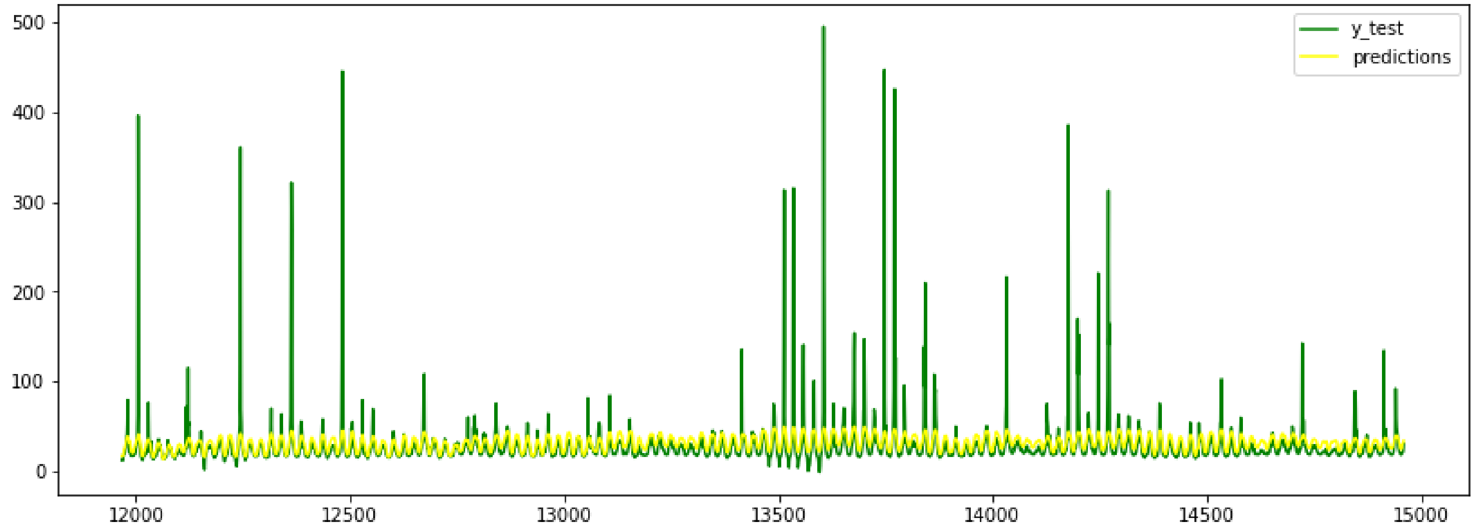


Original Prediction with OLS Model:

Data Splitting:

Increase the portion of Training data

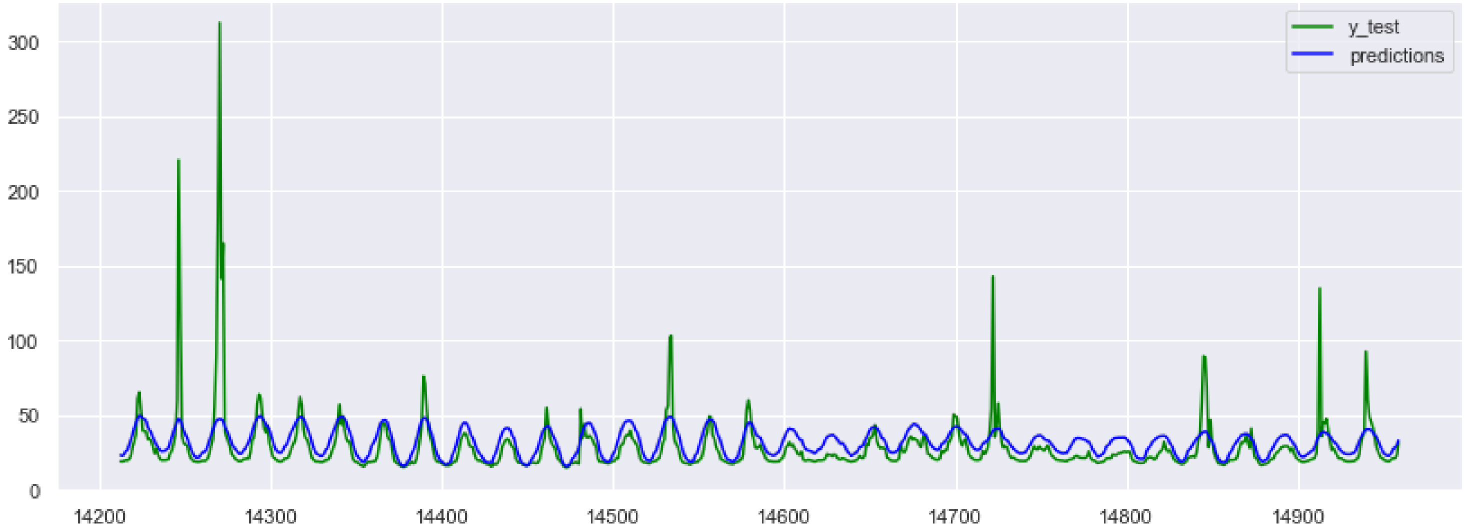
Predict shorter period

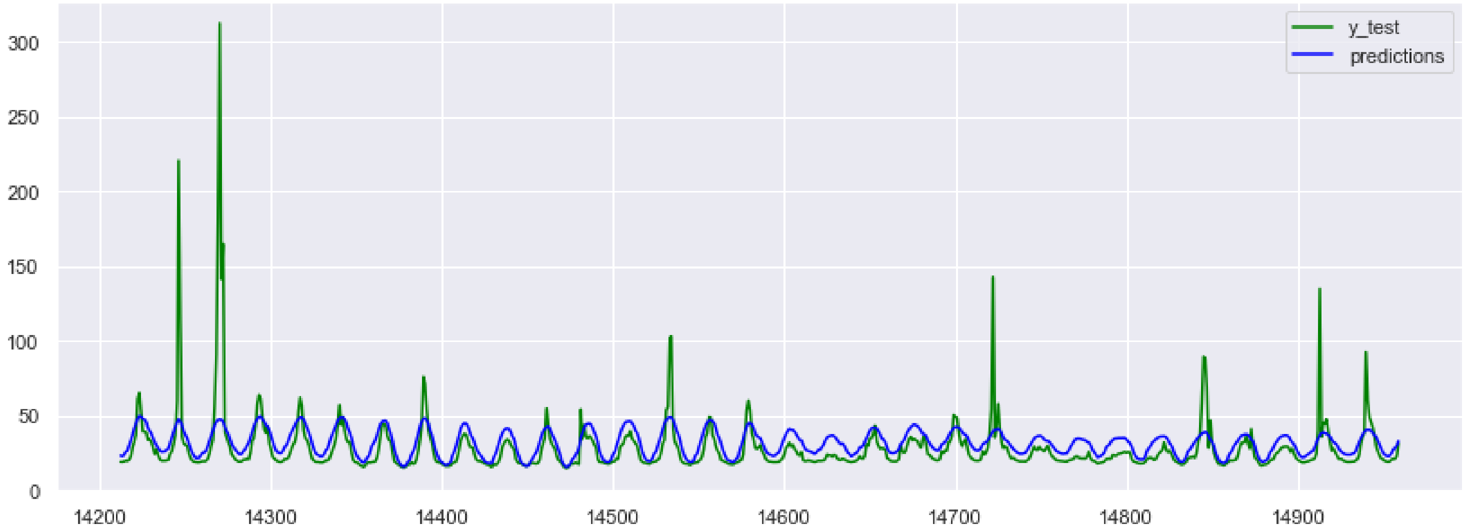
Model Selection reflection:

Discover the better model using Arima, Seasonal Arima, LSTM (currently studying), Informer (currently studying)

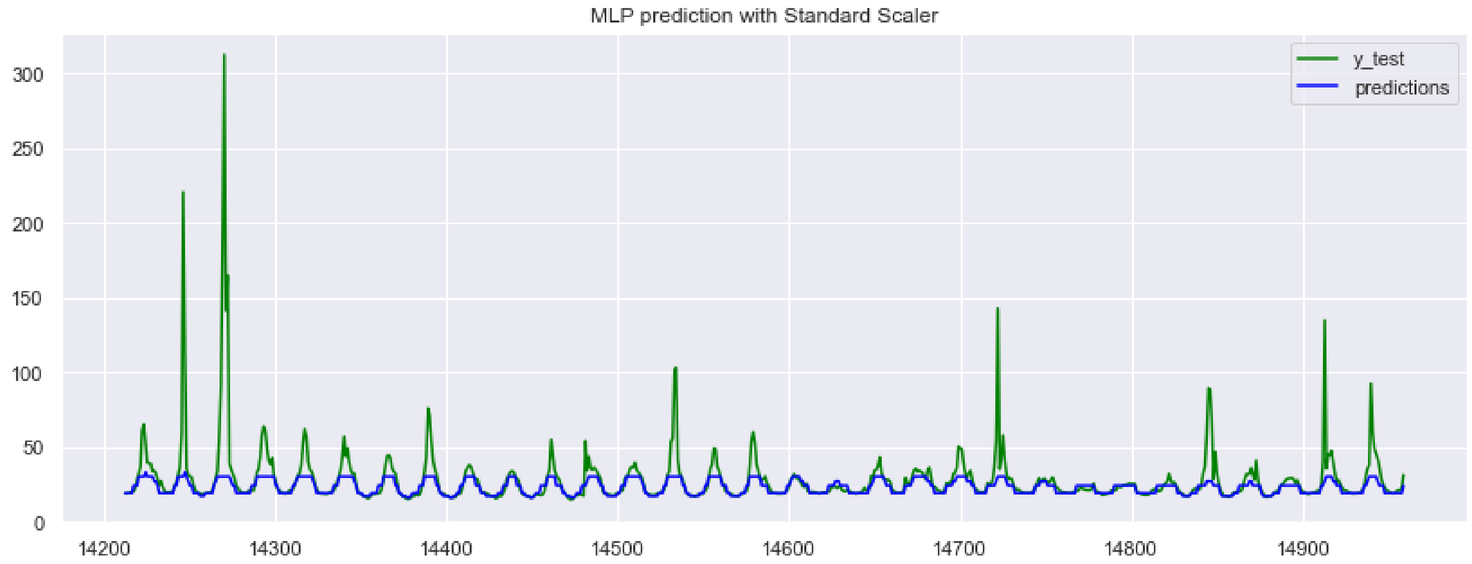
User the higher standard to conduct model selection: I only tested the stationarity of my data, and I could compare and find a better model while contrasting the R squared, MSE or RMSE scores, or  the aic and bic.

OLS Models under 2 Scalers:

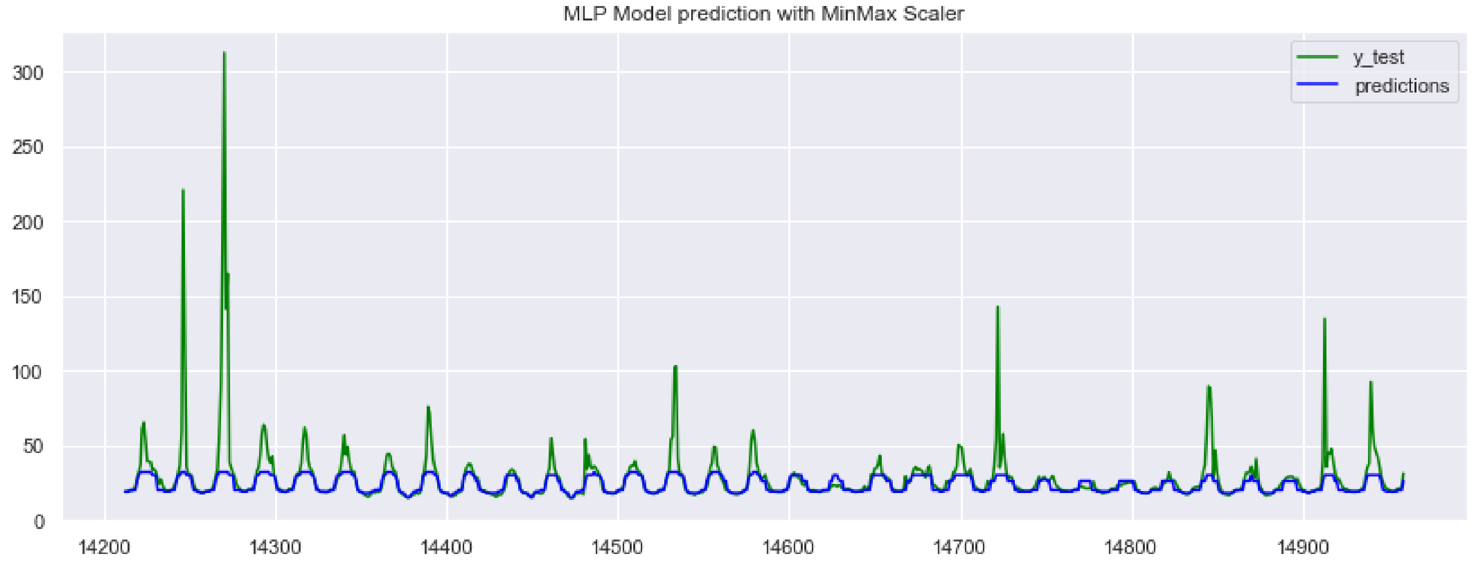

OLS Model prediction with Standard Scaler


OLS Model prediction with MinMax Scaler

My MLP Models under 2 Scalers:



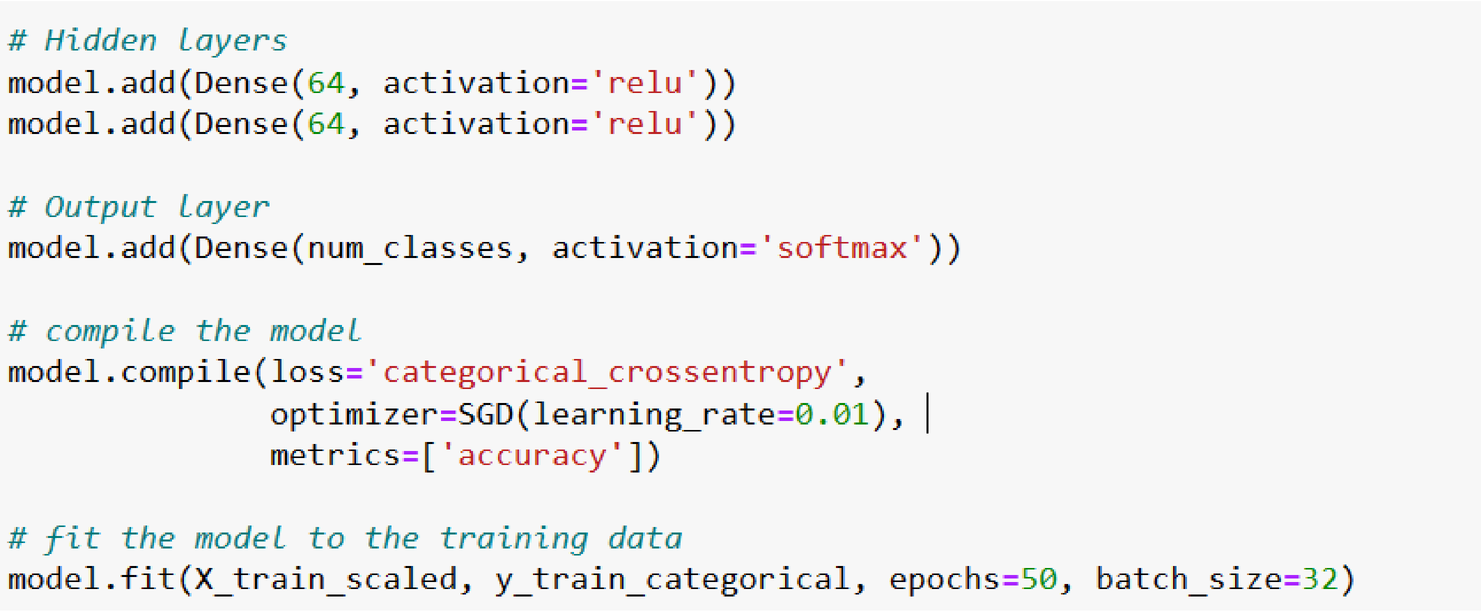
MLP prediction with Standard Scaler



MLP Model prediction with MinMax Scaler

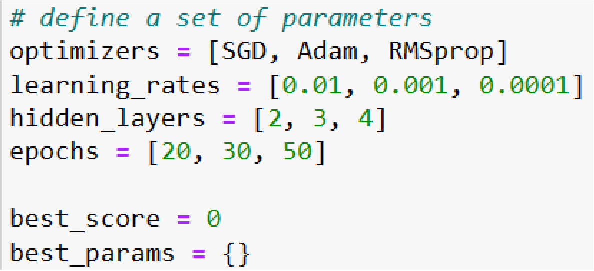
Because of the Loop I am using to find the best parameters includes different Optimizers, Learning Rates, Hidden Layers and Epochs, for the purpose of avoiding Overfitting, I also put the following code into the loop:

Old:

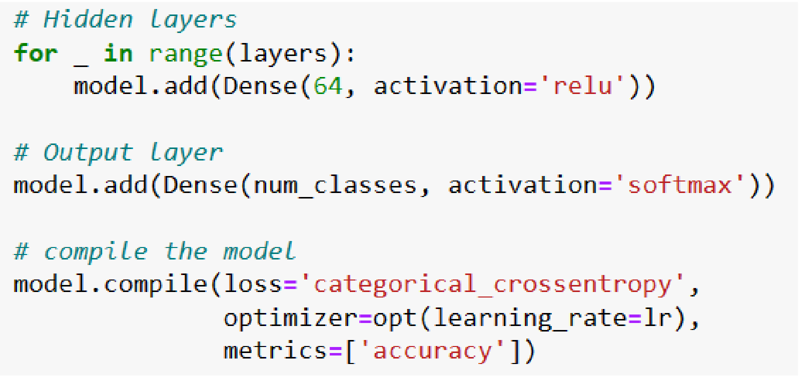


```
# Hidden layers
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))

# Output layer
model.add(Dense(num_classes, activation='softmax'))

# compile the model
model.compile(loss='categorical_crossentropy',
              optimizer=SGD(learning_rate=0.01),
              metrics=['accuracy'])

# fit the model to the training data
model.fit(X_train_scaled, y_train_categorical, epochs=50, batch_size=32)
```

I defined a set of parameters and running them in the loop:

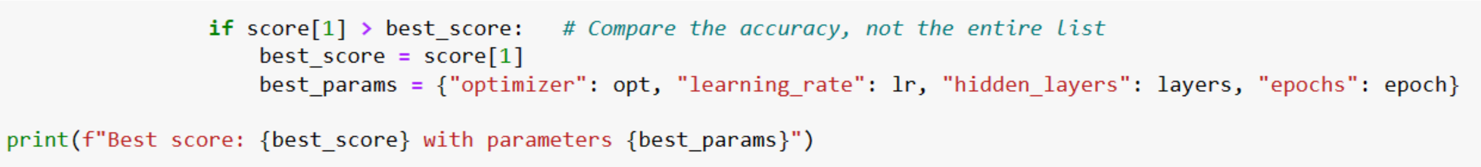


```
# define a set of parameters
optimizers = [SGD, Adam, RMSprop]
learning_rates = [0.01, 0.001, 0.0001]
hidden_layers = [2, 3, 4]
epochs = [20, 30, 50]

best_score = 0
best_params = {}
```



```
# Hidden layers
for _ in range(layers):
    model.add(Dense(64, activation='relu'))

# Output layer
model.add(Dense(num_classes, activation='softmax'))

# compile the model
model.compile(loss='categorical_crossentropy',
              optimizer=opt(learning_rate=lr),
              metrics=['accuracy'])
```

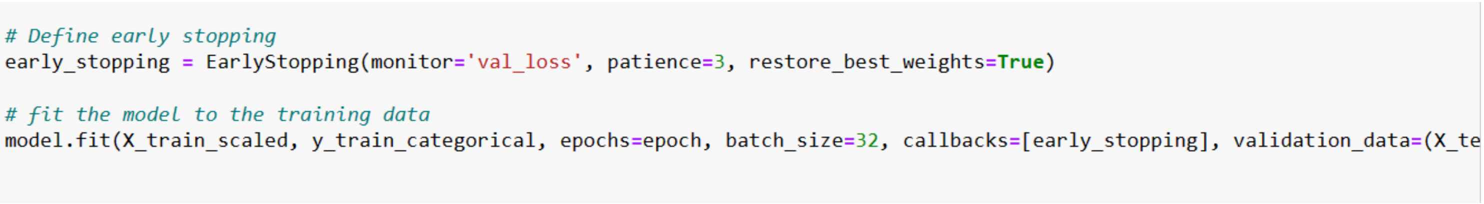
And I will receive the best parameters:



```
                if score[1] > best_score:   # Compare the accuracy, not the entire list
                    best_score = score[1]
                    best_params = {"optimizer": opt, "learning_rate": lr, "hidden_layers": layers, "epochs": epoch}

print(f"Best score: {best_score} with parameters {best_params}")
```

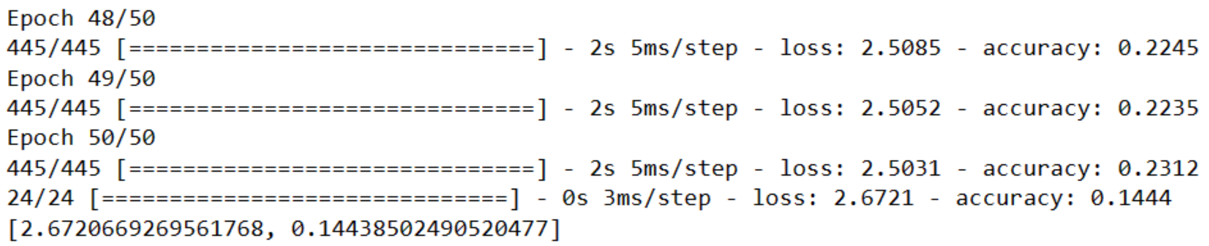
However, due to the endless epochs in this loop, there are multiple overfitting problems.

To Avoid this problem, I define the early stopping term:



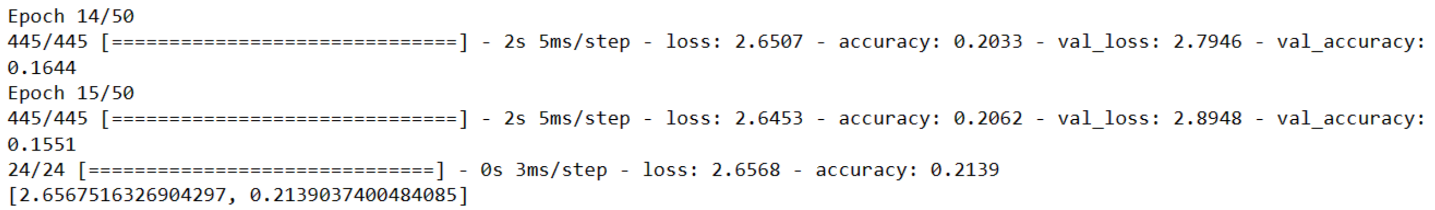
```
# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# fit the model to the training data
model.fit(X_train_scaled, y_train_categorical, epochs=epoch, batch_size=32, callbacks=[early_stopping], validation_data=(X_te
```

Without:



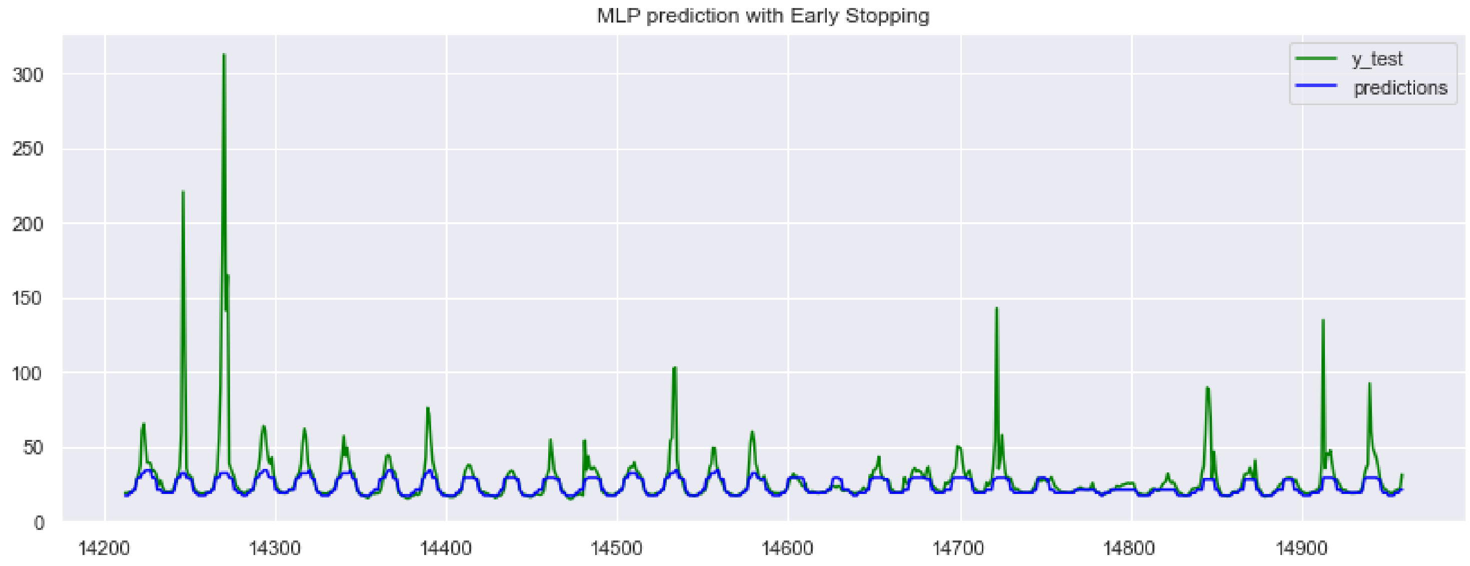
```
Epoch 48/50
445/445 [==============================] - 2s 5ms/step - loss: 2.5085 - accuracy: 0.2245
Epoch 49/50
445/445 [==============================] - 2s 5ms/step - loss: 2.5052 - accuracy: 0.2235
Epoch 50/50
445/445 [==============================] - 2s 5ms/step - loss: 2.5031 - accuracy: 0.2312
24/24 [==============================] - 0s 3ms/step - loss: 2.6721 - accuracy: 0.1444
[2.6720669269561768, 0.14438502490520477]
```

With early Stopping, the model won’t remember the specific feature and cause overfitting:

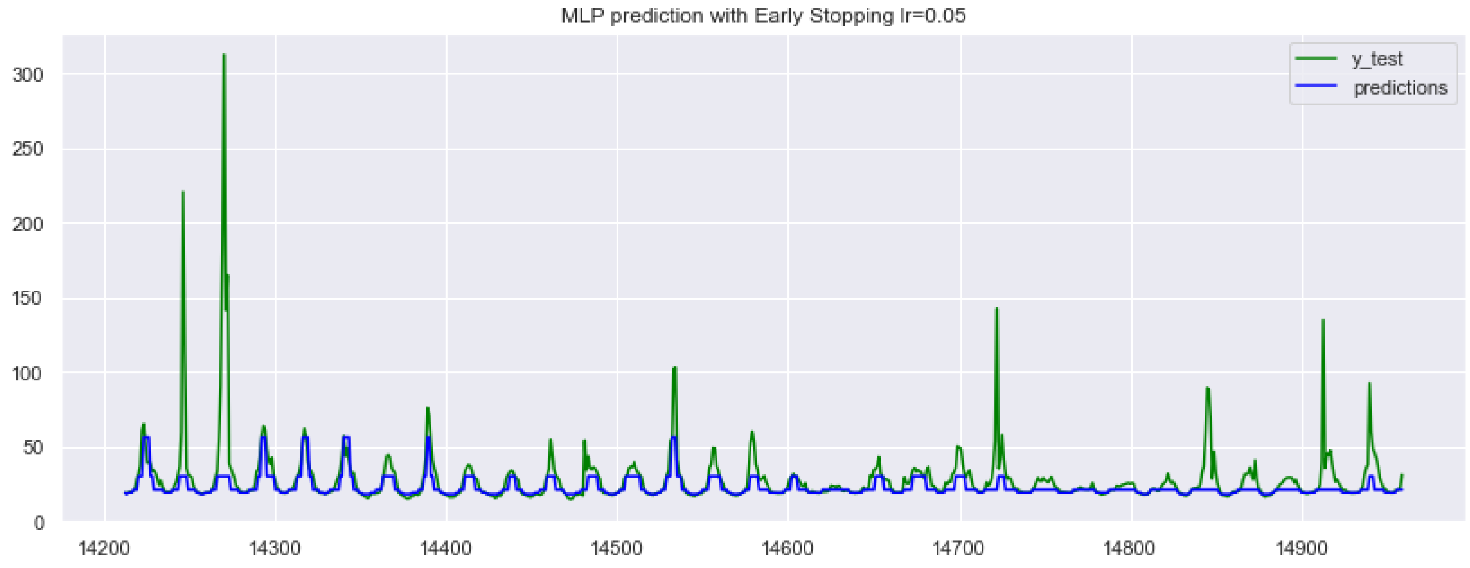


```
Epoch 14/50
445/445 [==============================] - 2s 5ms/step - loss: 2.6507 - accuracy: 0.2033 - val_loss: 2.7946 - val_accuracy:
0.1644
Epoch 15/50
445/445 [==============================] - 2s 5ms/step - loss: 2.6453 - accuracy: 0.2062 - val_loss: 2.8948 - val_accuracy:
0.1551
24/24 [==============================] - 0s 3ms/step - loss: 2.6568 - accuracy: 0.2139
[2.6567516326904297, 0.2139037400484085]
```

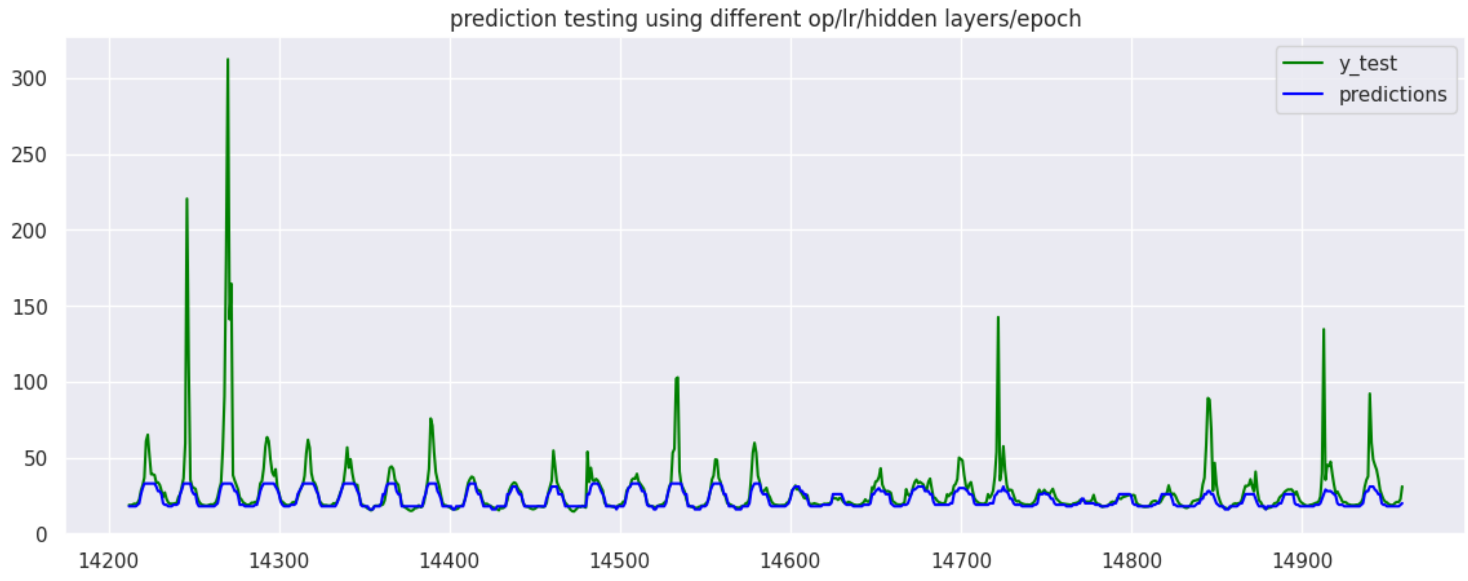
If val_accuracy starts decreasing, after patient (now =3) time, it will stop and generate the best Epoch for us.



Test 2, Learning rate = 0.05:



After the test, I could bring it back to the Loop I am using and generate the best parameters faster with higher accuracy:


prediction testing using different op/lr/hidden layers/epoch
y_test
predictions