Logan Kelsch

COSC 311 - Fall 2024

Homework 2 Report

INTRODUCTION

In this homework assignment, we were asked to use the “Food type recognition dataset” which contained 65 features and one target. This target consisted of 20 different classifications, and the features consisted of already normalized values.

We were then asked to split the data into training and testing (80/20), select and build a classification model, and attempt to obtain the highest accuracy we can. After this, use a confusion matrix to understand the model’s performance on test data.

After the model's test performance data is observed, we are asked to explain the misclassifications.

MODEL:
BUILDING AND RESULTS

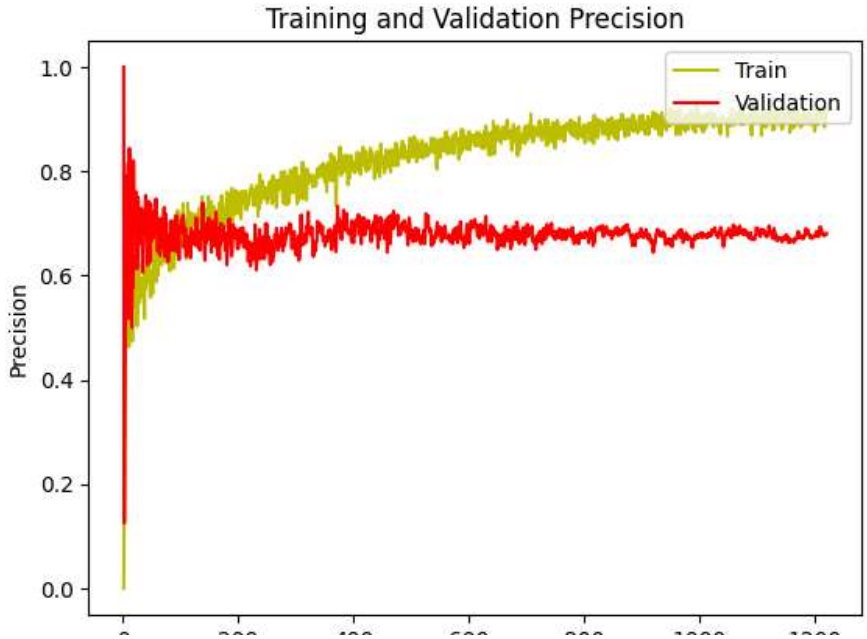
- To split the data, I used sklearn’s pre-built ‘train_test_split()’ function.
- I decided to go with building a neural network for my classification model, as it is what I am currently most comfortable with. I used the following build_model() function as seen on the page below.

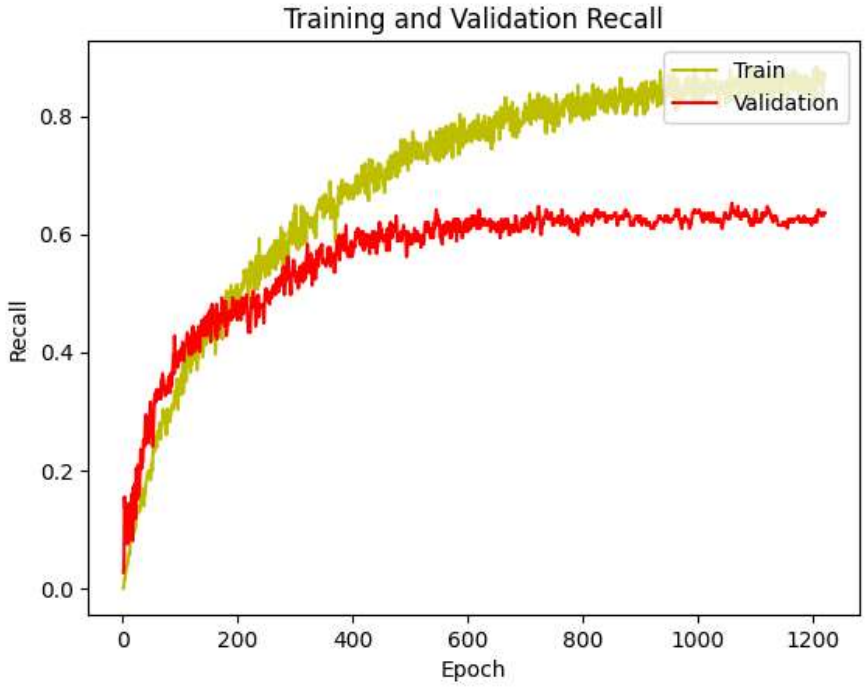
```
#function to contain construction of
#- neuron architecture
#- custom metric declaration
#- sequential model compilation
def build_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(512),#neuron layer
        #Excessive BN layers for small + complex dataset
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation('relu'),
        tf.keras.layers.Dropout(0.6),#high dropout
        tf.keras.layers.Dense(256),#decreasing vals
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation('relu'),
        tf.keras.layers.Dropout(0.5),#decreasing vals
        tf.keras.layers.Dense(128),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation('relu'),
        tf.keras.layers.Dropout(0.4),
        tf.keras.layers.Dense(64),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation('relu'),
        tf.keras.layers.Dropout(0.3),
        #output layer - 20 binarized classes
        tf.keras.layers.Dense(20, activation='softmax')
    ])
    #AUC=tf.keras.metrics.AUC(curve='PR')
    met = ['precision','recall','accuracy']
    model.compile(optimizer=opt4,
                  loss='categorical_crossentropy'
                  ,metrics=met)
    return model
```

I chose this architecture through some simple trial and error. I quickly found that the dataset was far too small for trying to accurately predict the complexity of the data. I also chose a purely ‘maximum validation accuracy’ approach, which consisted of dramatic loss and precision overfitting in a successful attempt for the model’s memorization to increase the validation recall, in turn increasing the test data accuracy.

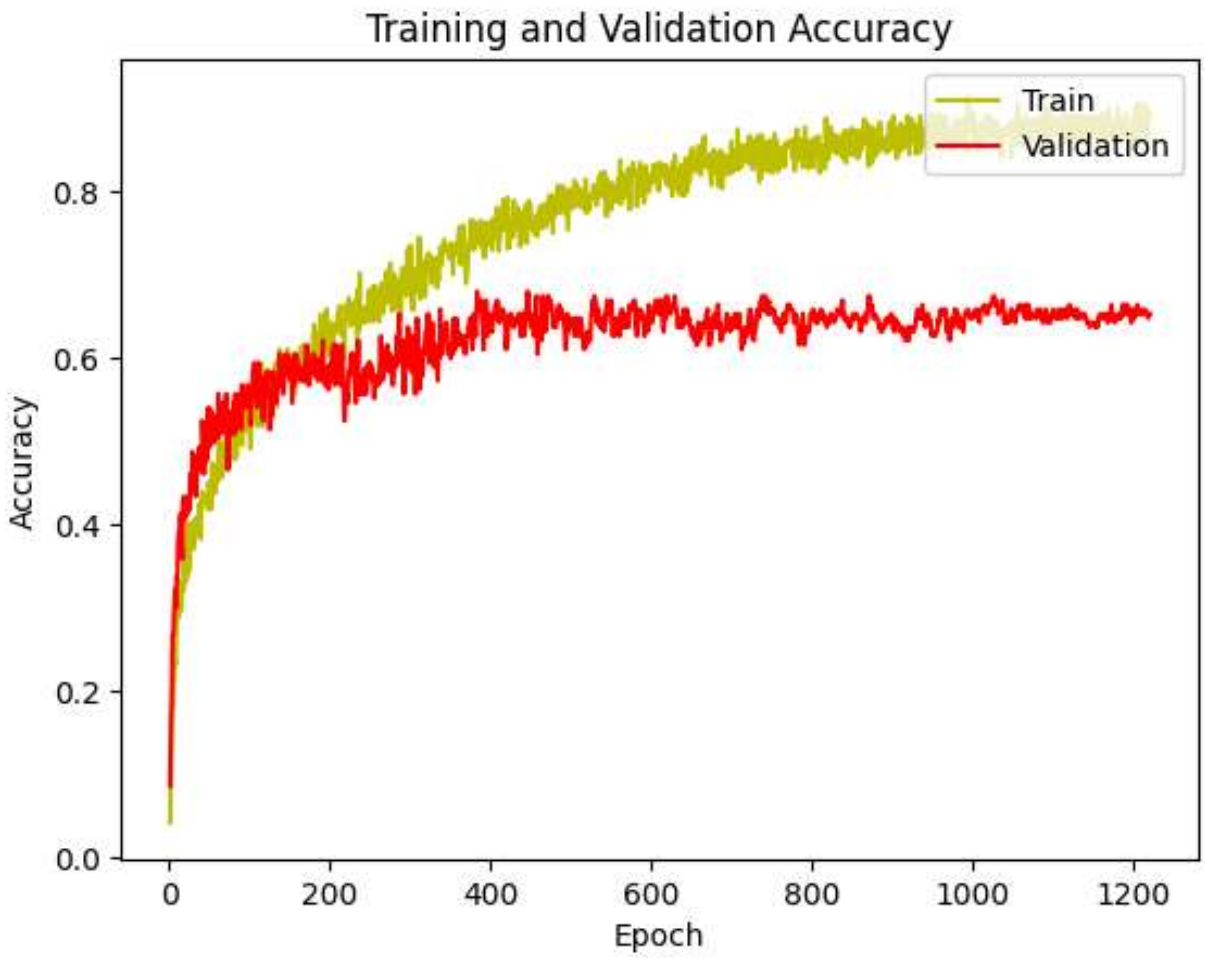
With this, the following graphs show obvious and aggressive overfitting just to maximize the validation accuracy. These graphs are of my model’s progress during training, tracking:

Accuracy, Precision, and Recall

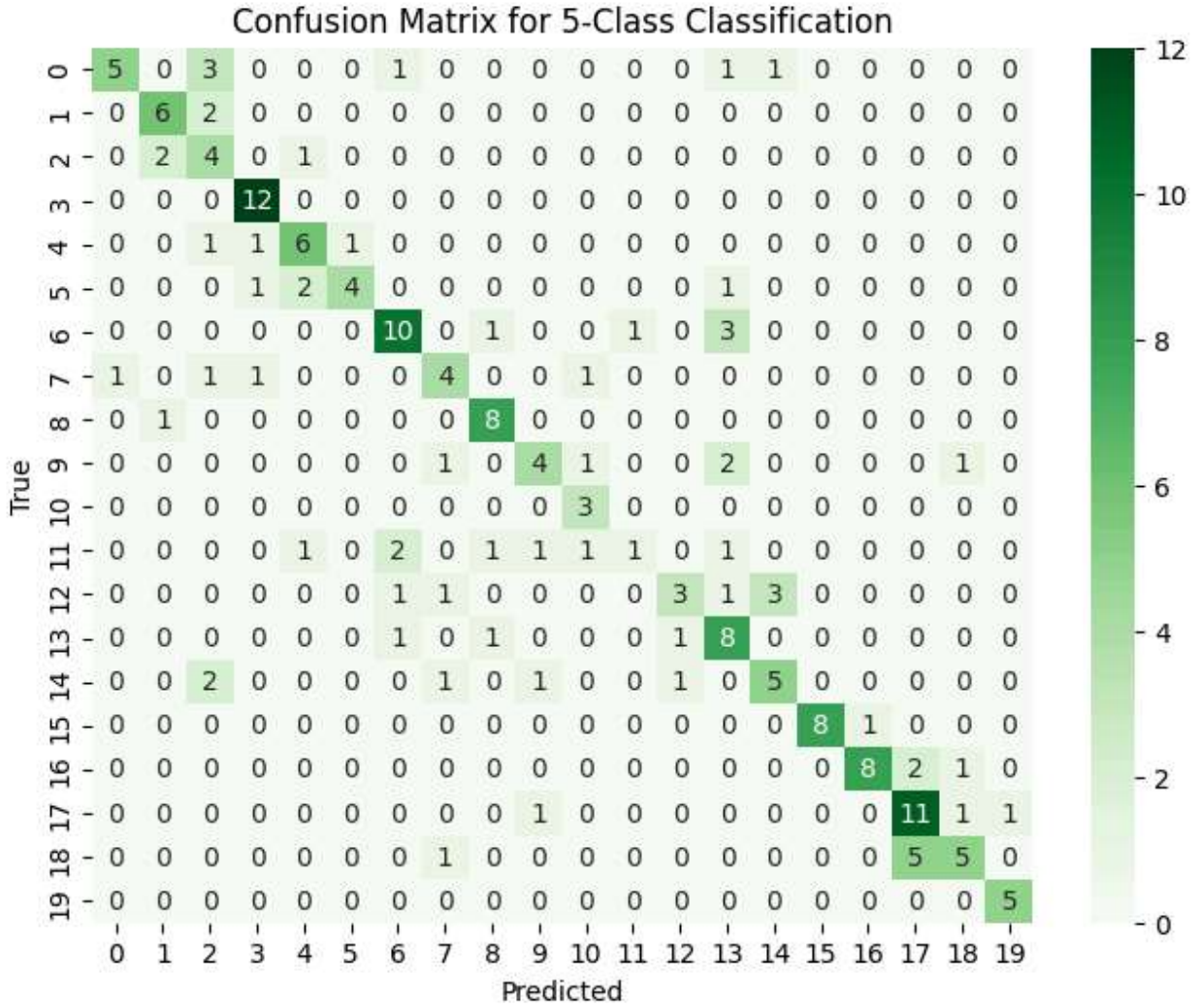




Accuracy on page below.


Training and Validation Accuracy
Train
Validation
Accuracy
Epoch
0.0
0.2
0.4
0.6
0.8
0
200
400
600
800
1000
1200

Here is the confusion matrix from the model



Here, finally is the final model output on the testing data set.

```
val_accuracy: 0.6524 - val_loss: 1.5961
val_precision: 0.6800 - val_recall: 0.6364
```

Although we were not provided information on exactly what each classification represents, there are clear misunderstandings by the model. For example, the model managed to accurately predict every true instance of classes 3, 10, and 19. However, the model almost never accurately predicted class 11.