Spring 2020

SW551 - Capstone II

Implementation Phase

AI-based Diagnostic Tool for Autism Screening

Group 2

Chris Carter

Indigo Guerra

Thomas Marrinan

Emre Ozbalta

Hung Nguyen

April 23rd, 2020

Project Summary

Our project tasked us to create a neural network that looks at a series of questions asked by the AQ10 Child survey to determine the possibility that a patient has Autism Spectrum Disorder (ASD). This application must run on an Android mobile device and will use Tensorflow to apply a simple neural network to make predictions.

In Capstone 1 we were asked to focus on researching how to perform this task. Starting with our data model we realized our data model was lacking records and completeness. The approach also needed some changes and the implementation of all the metadata fields. During capstone 2 we focused on getting more data and cleaning up our data model.

As for the Android Application, the path forward was simple. With our prototype in hand we designed a functional Android application that integrates a model designed in Tensorflow into it by utilizing Tensorflow Lite. The SRS document includes images of a Prototype, the mockups that we used to design our mobile application.

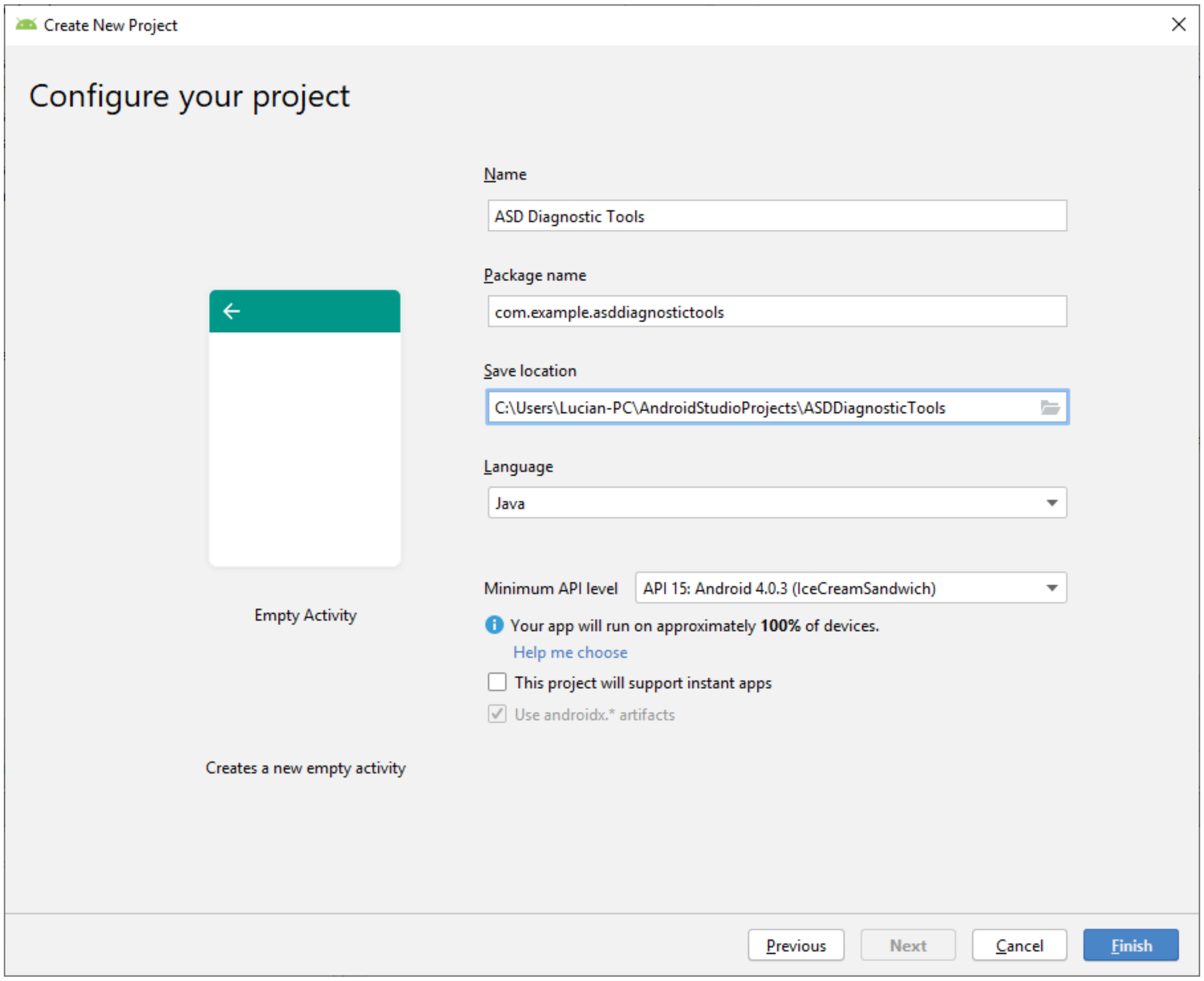
Development Environment Setup

To perform these tasks, a model must be created on a PC through Python using Tensorflow. To perform this task you first install Anaconda3 with the latest version of Python. Once installed, run the Anaconda3 command prompt to install pip. Then, use pip to install Tensorflow and Keras.

```
conda install pip
pip install --upgrade tensorflow
pip install --keras
```

Once this is done the script to create and train the model through either Spyder or Jupyter notebook (we used Jupyter notebook) both of which are included in the Anaconda3 installer. After training is complete the model must be saved to a Tensorflow Lite model in the code otherwise it cannot be loaded into the mobile application.

With the Tensorflow Lite model trained we can now discuss deploying to an Android app. To start, we first install Android studio using default options. Once installed we then setup the default device for emulation during the debug process. For this process we chose a Pixel 2, using API 29. With these things out of the way we can then create a new project using Java as our language. For this we set the minimum API on android to API 15: Android 4.0.3 (Ice Cream Sandwich). I've included a screenshot to show the options used when configuring our project.



Configure your project

Name: ASD Diagnostic Tools

Package name: com.example.asddiagnostictools

Save location: C:\Users\Lucian-PC\AndroidStudioProjects\ASDDiagnosticTools

Language: Java

Minimum API level: API 15: Android 4.0.3 (IceCreamSandwich)

Your app will run on approximately **100%** of devices.

Help me choose

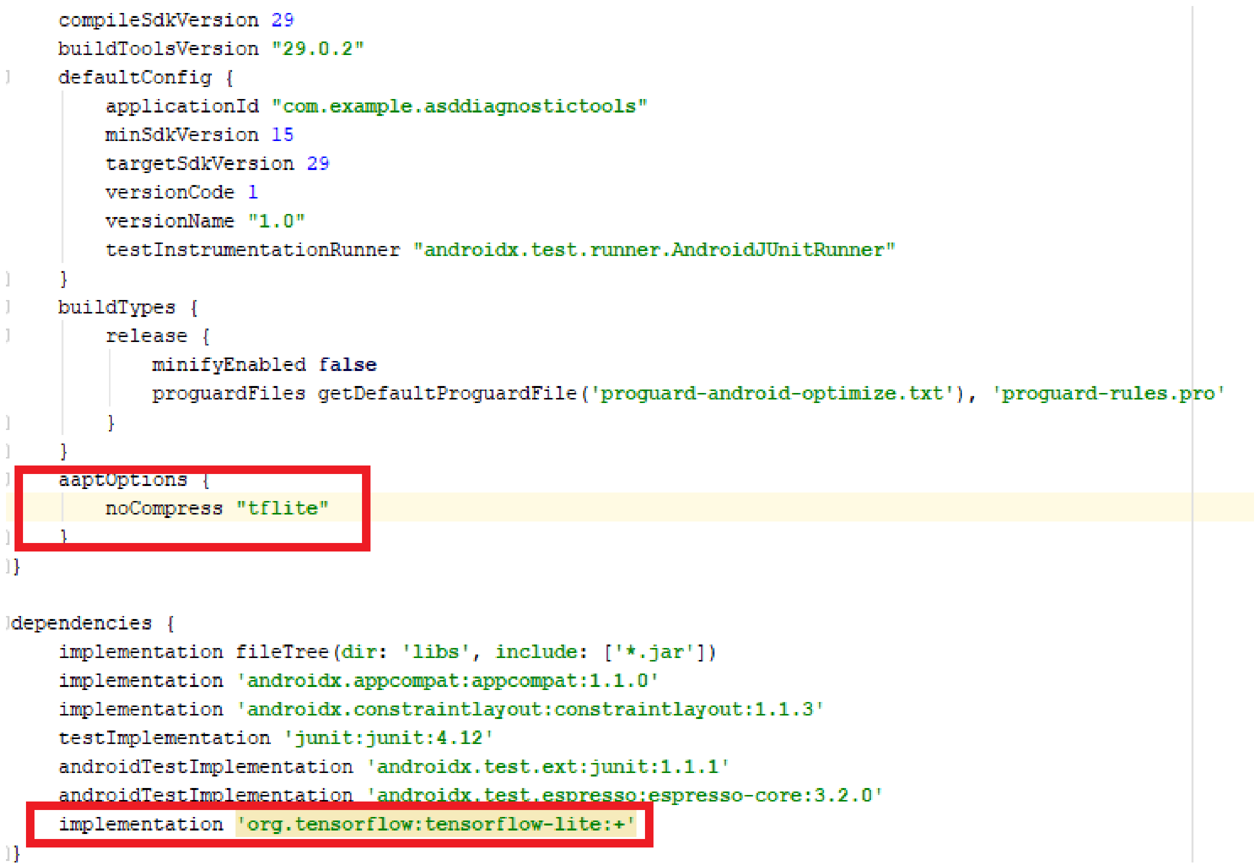
☐ This project will support instant apps

☑ Use androidx.* artifacts

Empty Activity

Creates a new empty activity

With a new project created and ready, we then added the Tensorflow Lite model to the Assets folder of the project and modified the Gradel build script to include the dependency for Tensorflow Lite and set the aaptOptions to prevent the model from being compressed.



```
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.example.asddiagnostictools"
        minSdkVersion 15
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    aaptOptions {
        noCompress "tflite"
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation 'org.tensorflow:tensorflow-lite:+'
}
```

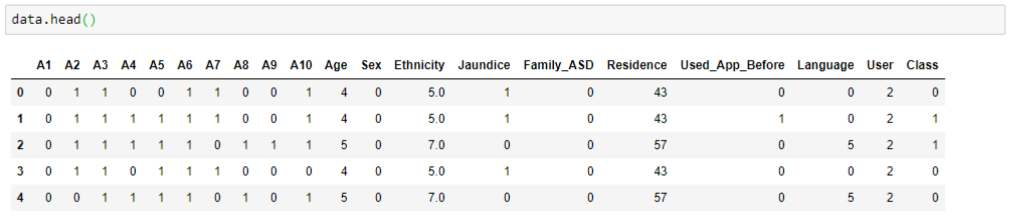
With this configuration completed I was then able to write Java code to load the model and push data into it to create a prediction.

Analysis of the Dataset

The original dataset for the AQ10 child survey we utilized for our network originated from the UCI Machine Learning Repository (see UCI Machine Learning Repository) and contained only 292 instances of data with 21 attributes. The data was incomplete and we needed more records.

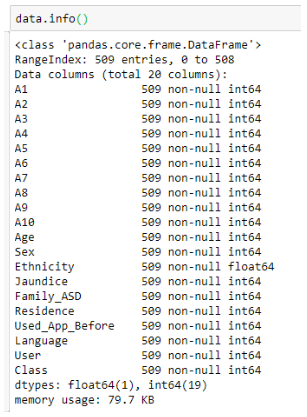
The new dataset for this project was provided by the same guy who posted the first version to the UCI Machine Learning Repository, Fadi Fayez Thabtah (see https://fadifayez.com/autism-datasets/). This dataset has 509 instances of data and 24 attributes. Only 20 attributes are useful to us as the other four are irrelevant, such as the Score field which simply adds the scores of all AQ10 questions together. As this is a derived fact, the summation of a value we already have broken out into individual answers is of no value to us.

Ten of these attributes are the AQ10 question outcomes. Nine others are metadata fields that describe the patient in some way such as their age, ethnicity, family history, etc. We want to use these metadata fields in conjunction with the AQ10 questions to help predict ASD.



```
data.head()
```

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Age	Sex	Ethnicity	Jaundice	Family_ASD	Residence	Used_App_Before	Language	User	Class
0	0	1	1	0	0	1	1	0	0	1	4	0	5.0	1	0	43	0	0	2	0
1	0	1	1	1	1	1	1	0	0	1	4	0	5.0	1	0	43	1	0	2	1
2	0	1	1	1	1	1	0	1	1	1	5	0	7.0	0	0	57	0	5	2	1
3	0	1	1	0	1	1	1	0	0	0	4	0	5.0	1	0	43	0	0	2	0
4	0	0	1	1	1	1	0	1	0	1	5	0	7.0	0	0	57	0	5	2	0



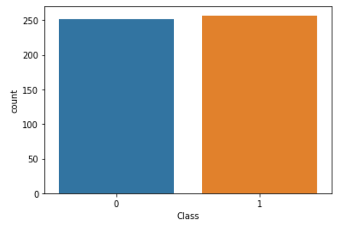
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 509 entries, 0 to 508
Data columns (total 20 columns):
A1                 509 non-null int64
A2                 509 non-null int64
A3                 509 non-null int64
A4                 509 non-null int64
A5                 509 non-null int64
A6                 509 non-null int64
A7                 509 non-null int64
A8                 509 non-null int64
A9                 509 non-null int64
A10                509 non-null int64
Age                509 non-null int64
Sex                509 non-null int64
Ethnicity          509 non-null float64
Jaundice           509 non-null int64
Family_ASD         509 non-null int64
Residence          509 non-null int64
Used_App_Before    509 non-null int64
Language           509 non-null int64
User               509 non-null int64
Class              509 non-null int64
dtypes: float64(1), int64(19)
memory usage: 79.7 KB
```

A count plot of our actual label to see the number of instances per label and see if it is a well balanced problem:

```
# Well balanced actual label
sns.countplot(x = 'Class', data = data)
```

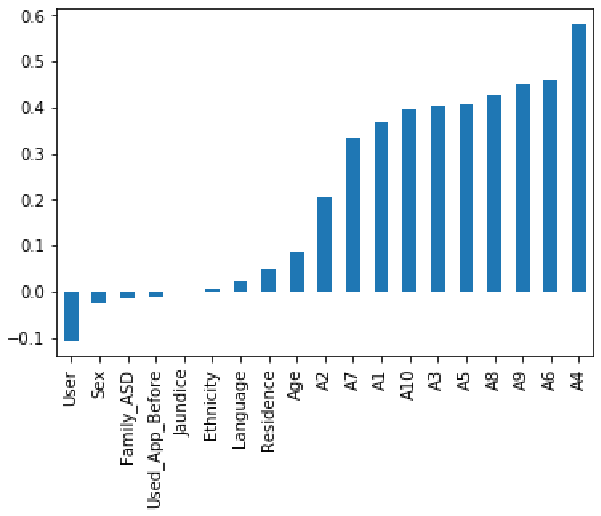
```
<matplotlib.axes._subplots.AxesSubplot at 0x2c2abdb0148>
```



We checked out the correlation between the features themselves using bar plot and heatmap. Bar plot doesn't include label variable but heatmap does. We can see that what is highly positively correlated as well as what's negatively correlated with our label. It is easy to come up with that the correlation between AQ10 question outcomes and our label variable (Class) is stronger than the correlation between any other feature and our label variable.

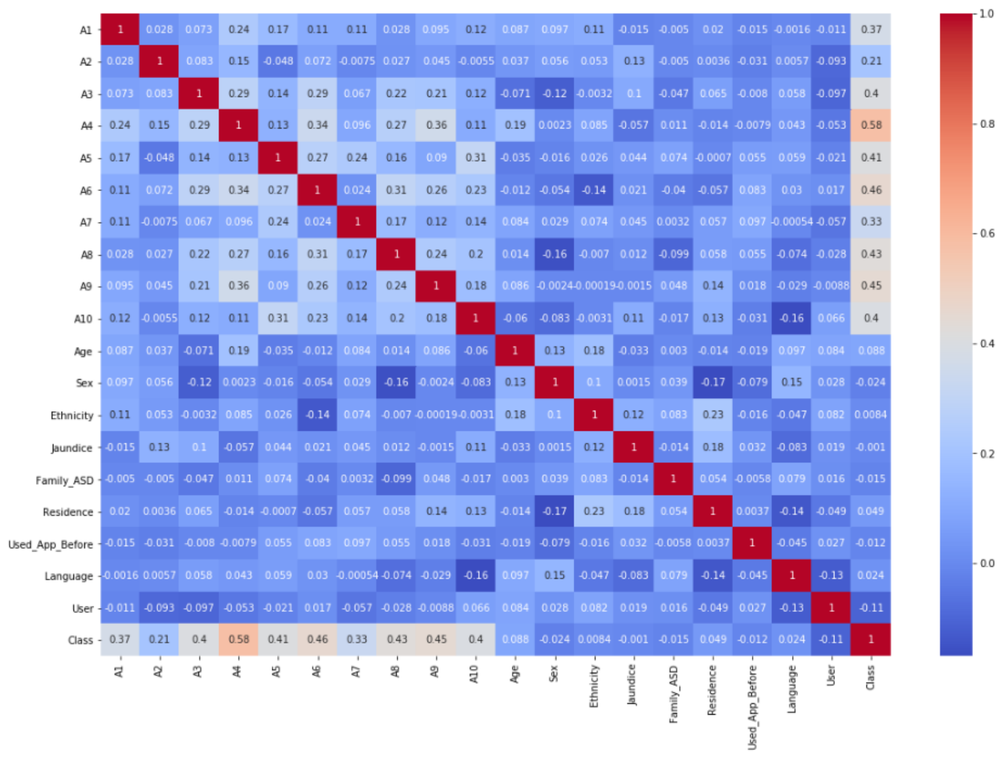
```
# Correlation between the features (bar plot)
data.corr()['Class'][:-1].sort_values().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2c2ac072bc8>
```



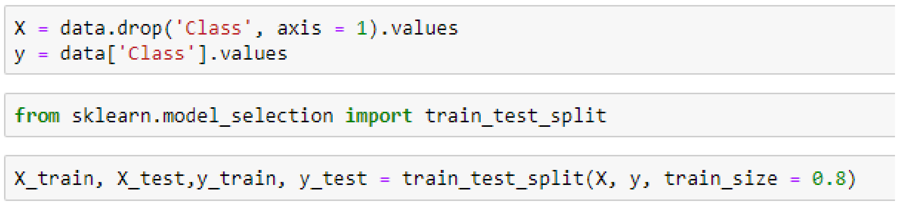
```
plt.figure(figsize=(18,12))
sns.heatmap(data.corr(),annot=True,cmap='coolwarm')
plt.ylim(20, 0)
```

```
(20, 0)
```



Creating the Neural Network

Our Neural Network was created using Python, which is the most popular and favored programming language amongst developers due to its extensive selection of libraries and frameworks, simplicity and amount of support derived from being an open source programming language. We used many of the built in functions in both the Tensorflow and Scikit-learn libraries as we constructed our model. To begin this neural network creation process, we started off by splitting our data into two sets: the training and testing set. Using common variable naming, we created X, which are the features and Y, which is the label for each of those feature rows. We used the Scikit-learn library function **train_test_split()** and passed in X, Y and train size as the parameters. Scikit-learn automatically outputs our training set and testing set so we have X_train and y_train, and X_test and y_test. At this point, we basically have our labels for our training and testing data, as well as our features for training and testing data.

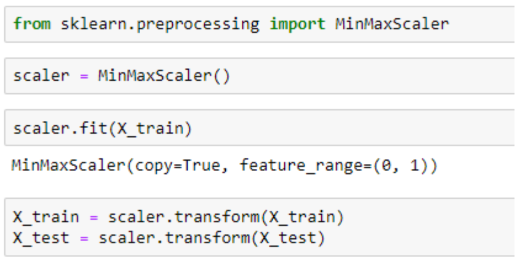


```
X = data.drop('Class', axis = 1).values
y = data['Class'].values
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test,y_train, y_test = train_test_split(X, y, train_size = 0.8)
```

We work with weights and biases inside of a neural network, which is why it is common to normalize or scale our data to reduce errors that can derive from having large values in your feature set. To avoid encountering any issues while training our network Scikit-learn has some built in functions such as **minmaxscaler** that can help with easily scaling our feature data. It essentially is going to transform our data based on the standard deviation of our data as well as the min and max values. We only need to scale the features since that's essentially what is being passed through the actual network. Final label is just a comparison done at the end. **Fit** function essentially calculates the standard deviation. We only run the fit on the training sets to prevent what is commonly referred to as data leakage from the test set. We never want to assume that we have prior information about the test set, which is why we only fit our scaler to the training set so that it does not try to cheat and look into the test set. Finally, we transform our training data and test data.



```
from sklearn.preprocessing import MinMaxScaler
```

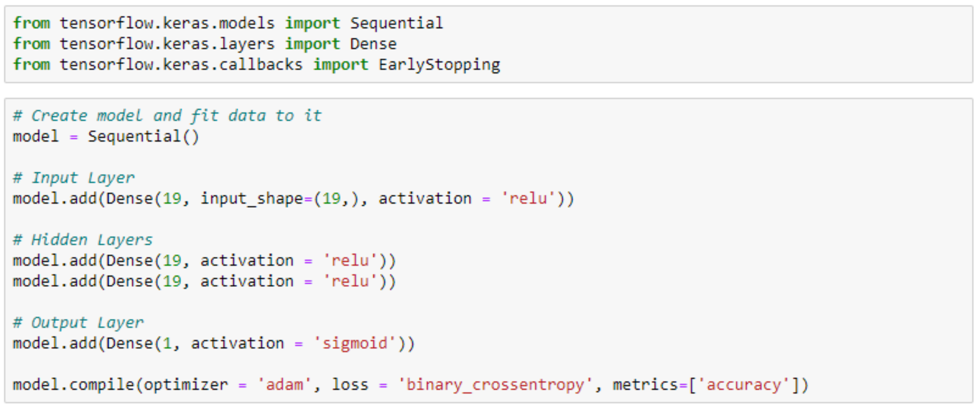
```
scaler = MinMaxScaler()
```

```
scaler.fit(X_train)
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Now that we have split and scaled the data, we can import the Sequential model and Dense layer to create our model. **Sequential** model is a linear stack of layers. We can create a Sequential model by passing a list of layer instances to the constructor. **Dense** is a regular densely connected neural network layer. It means if something is densely connected, it is going to be a normal feed forward network where every neuron is connected to every other neuron in the next layer. We try to base the number of neurons in our layers from the size of actual feature data. When we look at the shape or data, we will see that we have **19** incoming features and that’s a good range to have in our layer. We add one input layer, two hidden layers and one output layer into our model. We used only **one** neuron in the output layer because we deal with binary classification. If we had multi class classification, there could be more than one neuron in the output layer. Activation function for the input layer and hidden layers will be **relu (rectified linear unit)**. It has been found to have very good performance, especially when dealing with the issue of vanishing gradient. It is often used as the default due to its overall good performance. Activation function for the output layer will be **sigmoid** because this is a binary classification problem. Each neuron will output a value between 0 and 1, indicating the probability of having that class assigned to it. The **cross entropy loss function** is used for classification. This cross entropy loss function assumes that our model predicts a probability distribution for each class. As we have binary classification, we will use **binary cross entropy**. We have different options for optimizer but we use **adam optimizer.** Adam optimizer is an adaptive learning rate optimization algorithm that’s been designed for training deep neural networks. We also pass in accuracy metrics in our model to see our accuracy during the training.



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
```

```
# Create model and fit data to it
model = Sequential()

# Input Layer
model.add(Dense(19, input_shape=(19,), activation = 'relu'))

# Hidden Layers
model.add(Dense(19, activation = 'relu'))
model.add(Dense(19, activation = 'relu'))

# Output Layer
model.add(Dense(1, activation = 'sigmoid'))

model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics=['accuracy'])
```

Now that we have our model on splitted and scaled data, we can train or fit our model on the training data. This is done through the model.fit() method. Basically we take our model and pass in our X training data which are the features of our data and pass in our y training data which are the training labels. We use early stopping to prevent possible overfitting. We use **early stopping callbacks** based on our validation loss before it gets out of hand. Keras can automatically stop training based on a loss condition on the validation data passed during the model.fit() call. **Monitor** is the quantity to be monitored, and that’s validation loss here. In ‘**min’** mode, training will stop when the quantity monitored has stopped decreasing. **Patience** is the number of epochs with no improvement after which training will be stopped. **Epoch** is an arbitrary cutoff, generally defined as "one pass over the entire dataset", used to separate training into distinct phases, which is useful for logging and periodic evaluation. One epoch means we’ve gone through the entire dataset one time. **Verbose** equals to 1 indicates the printing output during training. The higher the number of verbose parameter calls that means the more information is displayed. To explore whether or not we are overfitting the training data, we pass in the **validation data** along for training. It means after each epoch of training on the training data will quickly run the test data and check our loss and accuracy on the test data so that way we can keep a tracking of how well performing not just on our training data but also on our test data. Based on that performance we go back and adjust our model. We repeated the adjustment process over and over again with adding more neurons or layers until we are satisfied. This test data will not actually affect the weights or biases of our network.

```
early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 1, patience = 25)
```

```
model.fit(X_train,y_train,epochs = 200, validation_data = (X_test, y_test), callbacks=[early_stop])
```

We can actually take a look at our training history using model.history.history. We pass this into a data frame. We plot out loss, accuracy, validation loss and validation accuracy using the validation data that we pass in during the training. This exactly the kind of plot that we want to see both validation loss and loss decrease, and both validation accuracy and accuracy increases.

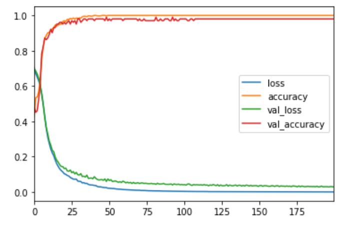
```
model_history = pd.DataFrame(model.history.history)
```

```
model_history.head()
```

	loss	accuracy	val_loss	val_accuracy
0	0.705202	0.427518	0.696133	0.480392
1	0.671215	0.533170	0.685416	0.450980
2	0.651087	0.538084	0.668315	0.460784
3	0.629195	0.570025	0.642979	0.529412
4	0.599545	0.643735	0.610088	0.607843

```
model_history.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x27bd517c048>
```



```
val_loss, val_acc = model.evaluate(X_test, y_test)
print(val_loss, val_acc)
```

```
102/102 [==============================] - 0s 59us/sample - loss: 0.0291 - accuracy: 0.9804
0.029052175741240967 0.98039216
```

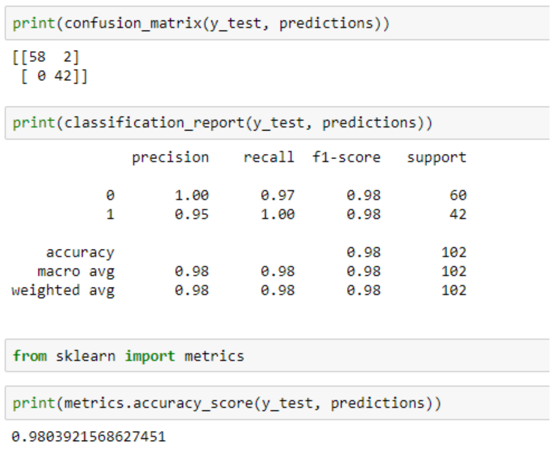
Now the model has been fit and trained on the training data. The model is ready to predict labels on the test set. We get predicted values using our finalized classification model in Keras using the predict_classes(). This function is only available for Sequential models. We pass in our test data those are our test features and then we get predictions off of the test data.

```
predictions = new_model.predict_classes(X_test)
```

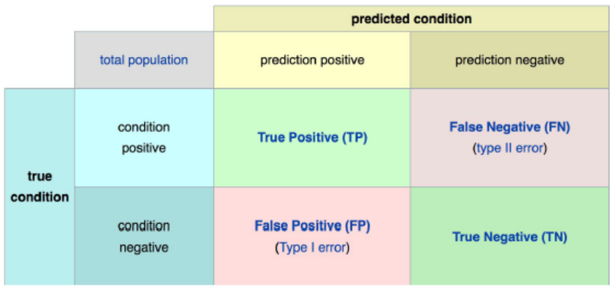
```
predictions[:10]
```

```
array([[0],
       [1],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [1]])
```

We can evaluate our model by comparing our predictions to the correct values. Typically in any classification task our model can only achieve two results. Either our model is correct in its prediction or our model is incorrect in its prediction. Once we have the model's predictions from the X_test data, we compare it to the true y values (the correct labels). At the end we have a count of correct matches and a count of incorrect matches. We organize our predicted values compared to the real values in a confusion matrix. The key classification metrics we need to understand are: Accuracy, Recall, Precision, F1-Score. **Accuracy** in classification problems is the number of correct predictions made by the model divided by the total number of predictions. Accuracy is useful when target classes are well balanced. As we see in the count plot, our actual label is well balanced. **Recall** is the ability of a model to find all the relevant cases within a dataset. The precise definition of recall is the number of true positives divided by the number of true positives plus the number of false negatives. **Precision** is the ability of a classification model to identify only the relevant data points. **Precision** is defined as the number of true positives divided by the number of true positives plus the number of false positives. While recall expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant actually were relevant. The **F1-score** is the harmonic mean of precision and recall taking both metrics into account in the following equation. We use the harmonic mean instead of a simple average because it punishes extreme values.



```
print(confusion_matrix(y_test, predictions))

[[58  2]
 [ 0 42]]

print(classification_report(y_test, predictions))

              precision    recall  f1-score   support

           0       1.00      0.97      0.98        60
           1       0.95      1.00      0.98        42

    accuracy                           0.98       102
   macro avg       0.98      0.98      0.98       102
weighted avg       0.98      0.98      0.98       102

from sklearn import metrics

print(metrics.accuracy_score(y_test, predictions))
0.9803921568627451
```



		predicted condition	
	total population	prediction positive	prediction negative
true condition	condition positive	True Positive (TP)	False Negative (FN) (type II error)
	condition negative	False Positive (FP) (Type I error)	True Negative (TN)

Running the Software

The Android application was tested in Android Studio using the "**Pixel 2 XL API29**" device. Once the project (folder) is loaded into Android Studio it will spend some time (20+ minutes on some machines) to parse the data to organize the project in order to build. Once ready you should be able to click Run to proceed.

The tensorflow lite model is in the assets folder and may be overwritten with a new model generated from the Python scripts. No code changes are provided so long as fields aren't added or removed from the inputs.

Conclusions

We've improved the accuracy of our model significantly and added more data than last semester. The model is integrated with the mobile application and provides real-time predictions to determine if a child has ASD. Our model has a high degree of accuracy and works well against the clean dataset when performing a manual validation.

I have to mention that the dataset provided to us did not contain a real diagnosis from a doctor, nor could we find one with additional metadata as our project was defined to use. Dr. Fayez posted an updated dataset but this one also had no true diagnosis. After some digging we learned that this data originated from an app designed by Dr Fayez (ASD Quiz) because this information is nearly impossible to obtain due to protected health laws. The version of the app where this data was collected never asks the patient to return with a diagnosis after visiting a doctor.

Due to this these predictions are limited by the deterministic outcome so real data is necessary for a better, more realistic data model. Since the dataset outcome is solely based on scoring a 7 or higher on the survey then catching these cases will be rare. It's worth noting however that during testing the model I did run through a few samples and happened upon cases where the patient scored 6 out of 10 but our model predicted they may have ASD.

Also, we also found a project where Dr Fayez applied machine learning to this dataset. In this application we noted that his inputs were only the demographic and family history fields and the AQ10 questions were completely omitted. Since this model only predicts ASD while excluding the AQ10 questions this seems lacking and prone to provide wrong answers. The findings posted on this also show this to be the case and most likely, these findings guided him to add the dr's diagnosis field to the new ASD Quiz app. In time, hopefully he will post datasets that include this information so that a better neural network can be generated.

References

Tensorflow Lite Converter Process
https://www.tensorflow.org/lite/convert

AQ10 Survey
https://micmrc.org/system/files/webinars/AQ10-Child.pdf

AQ10 Survey Dataset from UCI Machine Learning Repository (used in Capstone 1)
https://archive.ics.uci.edu/ml/datasets/Autistic+Spectrum+Disorder+Screening+Data+for+Children++

AQ10 Survey Dataset v2 from Author’s Website (used in Capstone 2)
https://fadifayez.com/autism-datasets/ (select the Version. 2 link)