Group 3 - MA 2 (Batch 2)

Dimaculangan, Rob Ian

Gutierrez, Gaerlan John

Hernandez, John Robynn

Pangilinan, Patrick

Recto, Antonio Julian

Introduction

The data set that is used for the model used in this project is from a study conducted on the recurrence of breast cancer in female patients at the University of Chicago's Billings Hospital collected by Matjaz, Z., & Milan S. (1988). The dataset considers several factors from the patients: their age, menopause status, tumor size, number of involved axillary nodes, Tumor capsular penetration or lack thereof, the degree of tumor malignancy, specific location of the tumor on the breast, and whether the patient has undergone radiation therapy. Recurrence is classified in the data set as a binary value.

Going into further detail of each of the items in the dataset, the class feature indicates whether the patient has recurrent cancer (recurrence-events) or non-recurrent cancer (no-recurrence-events). Age is divided into four ranges: 30-39, 40-49, 50-59 and 60-69. Menopause is divided into three groups: premeno, ge40 and lt40. Tumor-size is divided into six ranges: 0-4, 5-9, 10-14, 15-19, 25-29 and 35-39. Inv-nodes are divided into three ranges: 0-2, 3-5 and 6-8. The node-caps feature is divided into two groups: yes and no. Deg-malig is a measure of the malignancy of the cancer and is divided into three groups: 1, 2 and 3. Breast is divided into two groups: left and right. Breast-quad is divided into nine groups: left_up, left_low, right_up, right_low, central, left_central, right_central, central_upper and central_lower. Irradiat is divided into two groups: yes and no.

	class	age	menopause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat
0	no-recurrence-events	30-39	premeno	30-34	0-2	no	3	left	left_low	no
1	no-recurrence-events	40-49	premeno	20-24	0-2	no	2	right	right_up	no
2	no-recurrence-events	40-49	premeno	20-24	0-2	no	2	left	left_low	no
3	no-recurrence-events	60-69	ge40	15-19	0-2	no	2	right	left_up	no
4	no-recurrence-events	40-49	premeno	0-4	0-2	no	2	right	right_low	no

Figure 1: A Snippet of the Dataset

This data is useful in finding connections between various factors in a patient's case of breast cancer before and after medical treatment and to infer whether certain factors affect the recurrence of breast cancer in patients. The goal of using this dataset is to use machine learning techniques to build a model that can predict whether a patient will have recurrent or non-recurrent cancer based on these features. This information can help doctors and healthcare professionals make more informed decisions about the treatment of patients with breast cancer.

Going over the exploratory data analysis that we conducted in the python notebook, we have 4 questions that we answered which are:

1. What is the most prominent number of involved auxillary nodes among patients with recurrent cancer?
2. How many patients with recurrent cancer have premenopause status vs non-recurrent patients?
3. On which side does breast cancer occur among patients with non-recurrent cancer vs patients with recurrent cancer?
4. What is the most common age range for patients with non-recurrent cancer vs patients with recurrent cancer?

The answers and conclusions for these questions are written in the Python notebook, providing visualization in the form of tables and conclusions in the form of comments on the code.

Discussion of the Experiment

Pre-Processing and Cleaning the Data

```
[ ]  # Drop null rows
     cancerdf = cancerdf.replace('?', np.nan)
     cancerdf = cancerdf.dropna()
```

Figure 2: Dropping Null Rows

We cleaned up the values in the table that don't have data labeled with a "?" symbol. We first replaced the "?" symbol with a null value and then proceeded to use the dropna() function to drop all the null values in the table.

```
[ ]  # ONE-HOT ENCODE THE CATEGORICAL VARIABLES
     # Get the column names of dataframe
     columns = cancerdf.columns.tolist()
     # Remove the column to not one-hot encode
     columns.remove('class')
     # One-hot encode all columns except 'class'
     cancerdf_encoded = pd.get_dummies(cancerdf, columns=columns)

     # Replace "no-recurrence-events" with 0 and "recurrence-events" with 1
     cancerdf_encoded["class"] = cancerdf_encoded["class"].replace("no-recurrence-events", 0).replace("recurrence-events", 1)
     cancerdf_encoded.head()
```

	class	age_20-29	age_30-39	age_40-49	age_50-59	age_60-69	age_70-79	menopause_ge40	menopause_lt40	menopause_premeno	...	breast-quad_right_low
0	0	0	1	0	0	0	0	0	0	1	...	0
1	0	0	0	1	0	0	0	0	0	1	...	0
2	0	0	0	1	0	0	0	0	0	1	...	0
3	0	0	0	0	0	1	0	1	0	0	...	0
4	0	0	0	1	0	0	0	0	0	1	...	1

5 rows × 48 columns

Figure 3: Encoding Categorical Variables into Numerical

Since the majority of the values in the csv file is categorical, we need to convert them into numerical values in order to be suitable for training the model. The methods we used were one-hot encoding the categorical variables to have them be easily identifiable through their own columns and having their values be 1 or 0 depending on which variable the data in a row belongs to. We also used the replace function for the class column which is the main column that the model will predict the values for. This is done so that the class column stays as one column and is easier to be entered into the model.

```
[ ]  # Divide the data frame into input and output
     X = cancerdf_encoded.drop('class', axis=1)
     Y = cancerdf_encoded['class']
```

```
[ ]  # Split the data into training and test sets
     from sklearn.model_selection import train_test_split

     xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size = 0.3)
```

```
[ ]  # Import necessary libraries
     import tensorflow as tf
     from tensorflow import keras
     from tensorflow.python.keras import layers
     from tensorflow.python.keras.layers import Dropout
```

Figure 4: Splitting the Data

The data is then divided into the input and output and split by using Sklearn in order to have random training test splits, using a test size of 0.3. We then imported the tensorflow and keras libraries in order to begin with writing the model for predicting the necessary results.

Testing and Fine-Tuning the Model

Legend

L = Loss Function

O = Optimizer

E = Epochs

First Test

- **L**: binary_cross_entropy; **O**: adam; **E**: 200;
    - **Layer 1 Nodes**: 50
    - **Output Layer Nodes**: 1
    - **Accuracy**: 64%; **Loss**: 1.1945;

```
[8]  model = keras.Sequential()
     model.add(layers.Dense(50, activation='relu'))
     model.add(layers.Dense(1,activation='sigmoid'))
```

For the first test, we started with one dense layer with 50 nodes and an output layer with one node using RELU as the activation function and sigmoid for the output layer.

```
[9]  # compile and fit the model
     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
     history = model.fit(xtrain, ytrain, validation_split=0.22, epochs=200)

     Epoch 172/200
     5/5 [==============================] - 0s 8ms/step - loss: 0.0699 - accuracy: 1.0000 - val_loss: 0.9538 - val_accuracy: 0.7209
     Epoch 173/200
     5/5 [==============================] - 0s 11ms/step - loss: 0.0693 - accuracy: 1.0000 - val_loss: 0.9582 - val_accuracy: 0.7209
     Epoch 174/200
     5/5 [==============================] - 0s 9ms/step - loss: 0.0684 - accuracy: 1.0000 - val_loss: 0.9581 - val_accuracy: 0.7209
     Epoch 175/200
     5/5 [==============================] - 0s 7ms/step - loss: 0.0677 - accuracy: 1.0000 - val_loss: 0.9597 - val_accuracy: 0.7209
     Epoch 176/200
     5/5 [==============================] - 0s 8ms/step - loss: 0.0669 - accuracy: 1.0000 - val_loss: 0.9615 - val_accuracy: 0.7209
     Epoch 177/200
     5/5 [==============================] - 0s 9ms/step - loss: 0.0661 - accuracy: 1.0000 - val_loss: 0.9637 - val_accuracy: 0.7209
     Epoch 178/200
```

We then compiled and fit the model with the binary cross entropy loss function and the adam optimizer, using 200 epochs and a validation split of 0.22

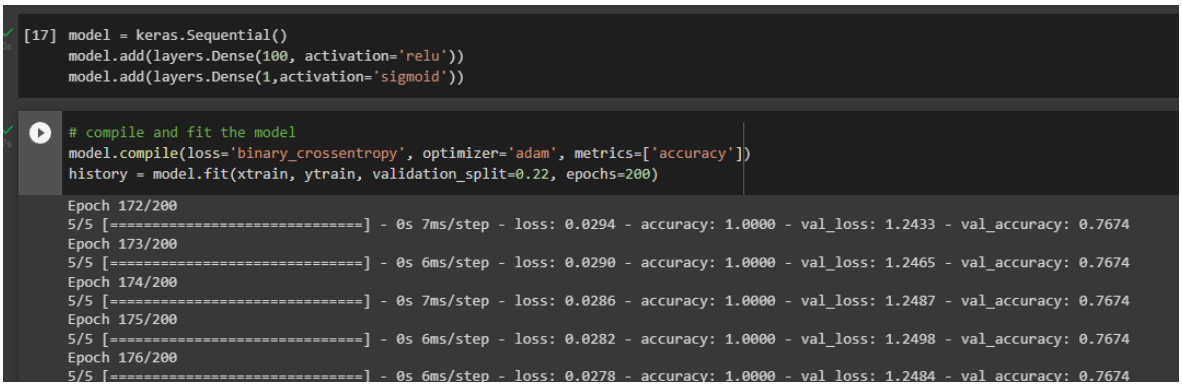
```
     model.evaluate(xtest, ytest)

     3/3 [==============================] - 0s 4ms/step - loss: 1.1945 - accuracy: 0.6429
     [1.1944921016693115, 0.6428571343421936]

[11] model.evaluate(xtrain,ytrain)

     7/7 [==============================] - 0s 2ms/step - loss: 0.2661 - accuracy: 0.9430
     [0.26612594723701477, 0.9430052042007446]

[12] import sklearn
     from sklearn.metrics import confusion_matrix
     ypred = (model.predict(xtest)>0.50).astype('int32')

     3/3 [==============================] - 0s 3ms/step

[13] confusion_matrix(ytest, ypred)

     array([[44, 10],
            [20, 10]])
```

The model was evaluated and came out to a loss of 1.1945 and an accuracy of 0.6429 which suggest that the model's predictions on the test set are not very accurate. The loss value of 1.1945 indicates that the model's predictions are far from the true

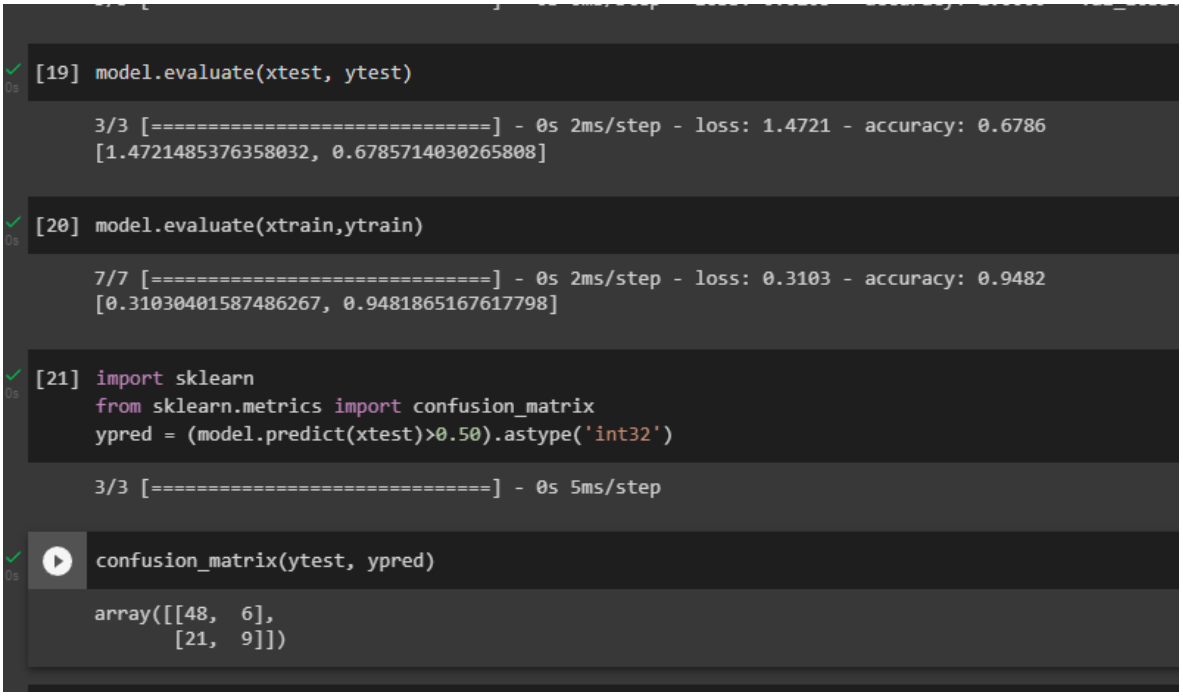
values. The accuracy of 0.6429 means that only 64.29% of the predictions made by the model on the test set were correct. This is a relatively low accuracy, and it suggests that the model is not performing well on the test set. It could be an indication that the model is overfitting on the training set, or that the model is not generalizing well to new data. This first test will be used as a baseline for future tests to be based on.

Second Test

- **L**: binary_cross_entropy; **O**: adam; **E**: 200;
    - **Layer 1 Nodes**: 100
    - **Output Layer Nodes**: 1
    - **Accuracy**: 67.8%; **Loss**: 1.4721;



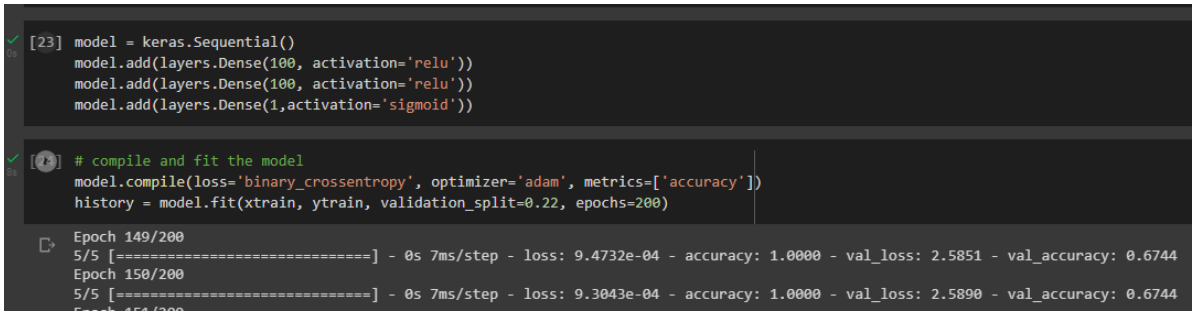
For the second test, we decided to increase the amount of nodes to try to increase the accuracy of the model, still using the previous other parameters that were used in the previous test.



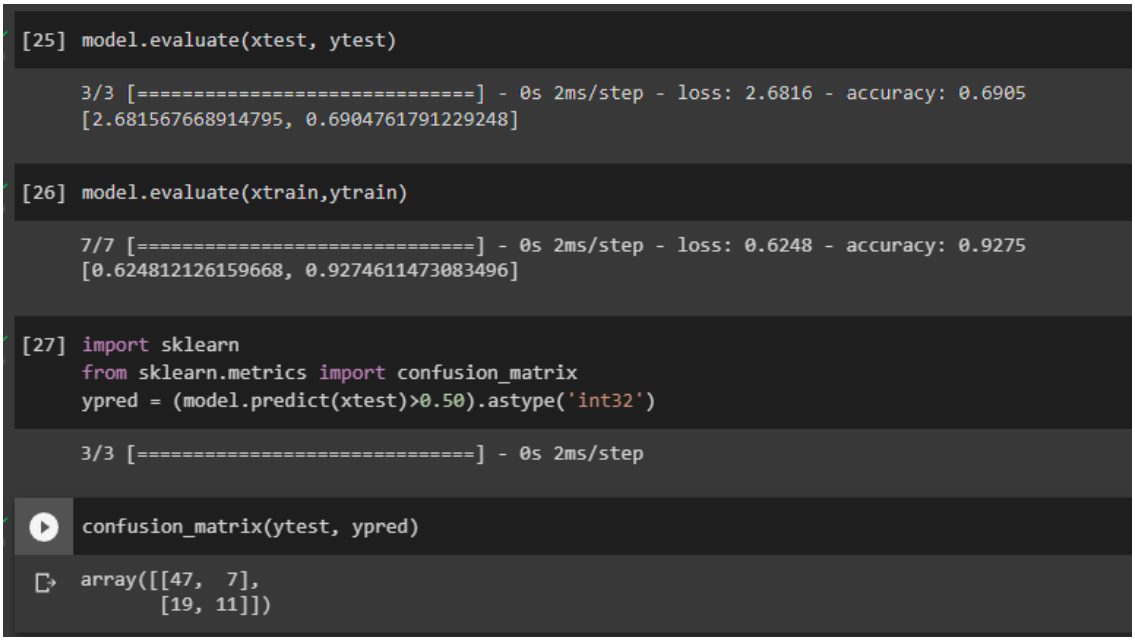
The second test results came out to a loss of 1.4721 and an accuracy of 0.6786 which means that the model is not performing well on the test set. The loss value is high, indicating that the model is making large errors in its predictions. The accuracy of 0.6786 also suggests that the model is not making accurate predictions and isn't much of a rise from the previous test. This could be due to overfitting or poor model design seeing as the loss also rose up slightly as well.

Third Test

- **L**: binary_cross_entropy; **O**: adam; **E**: 200;
    - **Layer 1 Nodes**: 100
    - **Layer 2 Nodes**: 100
    - **Output Layer Nodes**: 1
    - **Accuracy**: 69.05%; **Loss**: 2.6816;



For the third test, we attempted adding an additional layer to the model, retaining the values of the other parameters. We reasoned that perhaps adding an additional layer would increase the accuracy of the predictions calculated.



The results however were lackluster, with the loss value increasing to 2.6816, and accuracy only slightly increasing to 69%.

Fourth Test

- **L**: binary_cross_entropy; **O**: adam; **E**: 200;
    - **Layer 1 Nodes**: 100
    - **Layer 2 Nodes**: 100
    - **Layer 3 Nodes**: 100
    - **Output Layer Nodes**: 1
    - **Accuracy**: 72.62%; **Loss**: 4.7649;

```
model = keras.Sequential()
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(1,activation='sigmoid'))
```

```
[31] # compile and fit the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(xtrain, ytrain, validation_split=0.22, epochs=200)
```

```
Epoch 172/200
5/5 [==============================] - 0s 6ms/step - loss: 6.0733e-08 - accuracy: 1.0000 - val_loss: 4.4418 - val_accuracy: 0.6977
Epoch 173/200
5/5 [==============================] - 0s 7ms/step - loss: 6.0093e-08 - accuracy: 1.0000 - val_loss: 4.4431 - val_accuracy: 0.6977
```

For the fourth test, we decided to add another hidden layer with 100 nodes, again using the parameters that were used in the previous tests. At this point, we were trying to get the accuracy higher through just adding more layers and nodes, not realizing that it's also raising the loss value for the model.

```
[32] model.evaluate(xtest, ytest)
```

```
3/3 [==============================] - 0s 3ms/step - loss: 4.7649 - accuracy: 0.7262
[4.7649245262146, 0.726190447807312]
```

```
[33] model.evaluate(xtrain,ytrain)
```

```
7/7 [==============================] - 0s 2ms/step - loss: 1.0057 - accuracy: 0.9326
[1.0057212114334106, 0.9326424598693848]
```

```
[34] import sklearn
from sklearn.metrics import confusion_matrix
ypred = (model.predict(xtest)>0.50).astype('int32')
```

```
3/3 [==============================] - 0s 2ms/step
```

```
confusion_matrix(ytest, ypred)
```

```
array([[47,  7],
       [16, 14]])
```

As shown here, the model is still not generalizing well to the test set seeing as it still only has an accuracy of 72% and a massive loss of 4.7649 which is not ideal when you're trying to get a model to predict the values accurately. In the next tests, we try to keep the loss value more in mind and put less emphasis on adding more and more layers and nodes just to raise the accuracy of the results of the model.

Fifth Test

- **L**: binary_cross_entropy; **O**: adam; **E**: 300;
    - **Layer 1 Nodes**: 16
    - **Layer 2 Nodes**: 32
    - **Layer 3 Nodes**: 64
    - **Output Layer Nodes**: 1
    - **Accuracy**: 75%; **Loss**: 1.3916;

```
[ ]  model = keras.Sequential()
     model.add(layers.Dense(16, activation='relu'))
     model.add(Dropout(0.5))
     model.add(layers.Dense(32, activation='relu'))
     model.add(Dropout(0.5))
     model.add(layers.Dense(64, activation='relu'))
     model.add(Dropout(0.5))

     model.add(layers.Dense(1,activation='sigmoid'))
```

```
# compile and fit the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(xtrain, ytrain, epochs=300, validation_data=(xtest, ytest))
```

Now keeping the previous results in mind, we started focusing less on high amounts of nodes per layer and started focusing on reducing the loss value. And one of the ways that we used to help with the loss value issue is employing dropout layers to prevent overfitting while also being more conservative with the amount of nodes that are used per layer, having 16, 32 and 64 on the respective layers. We also started using a train test split of 0.3 at this point.

```
model.evaluate(xtest, ytest)

3/3 [==============================] - 0s 5ms/step - loss: 1.3916 - accuracy: 0.7500
[1.391640543937683, 0.75]

model.evaluate(xtrain,ytrain)

7/7 [==============================] - 0s 2ms/step - loss: 0.1431 - accuracy: 0.9534
[0.14306293427944183, 0.9533678889274597]

import sklearn
from sklearn.metrics import confusion_matrix
ypred = (model.predict(xtest)>0.50).astype('int32')

3/3 [==============================] - 0s 5ms/step

confusion_matrix(ytest, ypred)

array([[52,  7],
       [14, 11]])
```

The results come out to a loss of 1.3916 with an accuracy of 75% for the test set, but the training accuracy is very high, which may indicate that there is still some overfitting to the training data. However, there is a significant improvement to the test accuracy which is a good sign. The test loss also got relatively lower which signifies that we are headed to the right direction in terms of the parameters that we're setting for the model.

Sixth Test

- **L**: binary_cross_entropy; **O**: adam; **E**: 300;
    - **Layer 1 Nodes**: 32
    - **Layer 2 Nodes**: 32
    - **Layer 3 Nodes**: 32
    - **Output Layer Nodes**: 1
    - **Accuracy**: 83%; **Loss**: 0.5805;

```
model = keras.Sequential()
model.add(layers.Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(layers.Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(layers.Dense(32, activation='relu'))
model.add(Dropout(0.5))

model.add(layers.Dense(1,activation='sigmoid'))
```

```
[12] # compile and fit the model
     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
     history = model.fit(xtrain, ytrain, epochs=300, validation_data=(xtest, ytest))
```

```
Epoch 272/300
7/7 [==============================] - 0s 8ms/step - loss: 0.3156 - accuracy: 0.8808
Epoch 273/300
7/7 [==============================] - 0s 8ms/step - loss: 0.3298 - accuracy: 0.8446
```

For the sixth test, we equalized the node counts to 32 per layer to increase the accuracy of the model, retaining the same parameters for everything else.

```
model.evaluate(xtest, ytest)
```

```
3/3 [==============================] - 0s 5ms/step - loss: 0.5805 - accuracy: 0.8333
[0.5804913640022278, 0.8333333134651184]
```

```
[14] model.evaluate(xtrain,ytrain)
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.1500 - accuracy: 0.9430
[0.1499505490064621, 0.9430052042007446]
```

```
[15] import sklearn
     from sklearn.metrics import confusion_matrix
     ypred = (model.predict(xtest)>0.50).astype('int32')
```

```
3/3 [==============================] - 0s 8ms/step
```

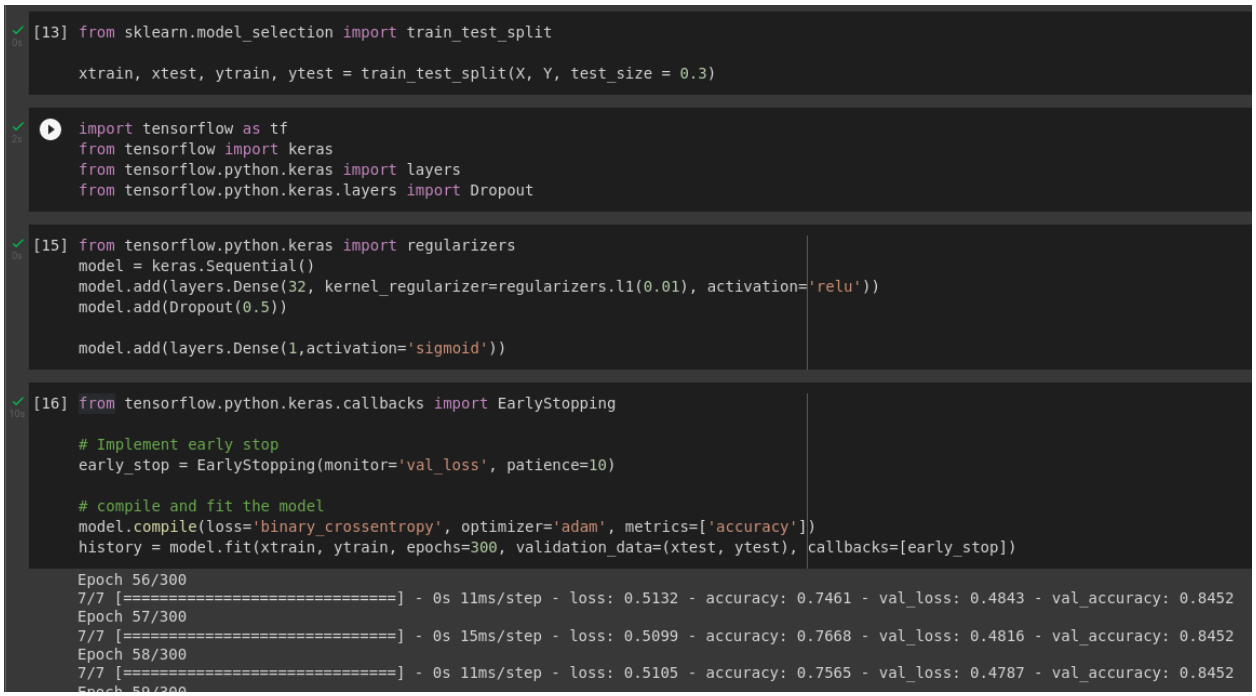
```
[16] confusion_matrix(ytest, ypred)
```

```
array([[61,  5],
       [ 9,  9]])
```

The results were promising with a significant improvement in accuracy from the previous test, going from 0.75 to 0.8333. The loss was also reduced by more than half, from 1.3916 to 0.5805. This all indicates that the model is able to fit the training data well. However, the difference in performance between the training set and test set, such as a big gap in accuracy, may indicate that the model is still overfitting to the training data. This is all addressed by a technique that is introduced in the next and last test.

Seventh Test

- **L**: binary_crossentropy; **O**: adam; **E**: 300;
    - **Layer 1 Nodes**: 32
    - **Output Layer Nodes**: 1
    - **Accuracy**: 82%; **Loss**: 0.4765;



```
[13] from sklearn.model_selection import train_test_split

     xtrain, xtest, ytrain, ytest = train_test_split(X, Y, test_size = 0.3)

     import tensorflow as tf
     from tensorflow import keras
     from tensorflow.python.keras import layers
     from tensorflow.python.keras.layers import Dropout

[15] from tensorflow.python.keras import regularizers
     model = keras.Sequential()
     model.add(layers.Dense(32, kernel_regularizer=regularizers.l1(0.01), activation='relu'))
     model.add(Dropout(0.5))

     model.add(layers.Dense(1,activation='sigmoid'))

[16] from tensorflow.python.keras.callbacks import EarlyStopping

     # Implement early stop
     early_stop = EarlyStopping(monitor='val_loss', patience=10)

     # compile and fit the model
     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
     history = model.fit(xtrain, ytrain, epochs=300, validation_data=(xtest, ytest), callbacks=[early_stop])

     Epoch 56/300
     7/7 [==============================] - 0s 11ms/step - loss: 0.5132 - accuracy: 0.7461 - val_loss: 0.4843 - val_accuracy: 0.8452
     Epoch 57/300
     7/7 [==============================] - 0s 15ms/step - loss: 0.5099 - accuracy: 0.7668 - val_loss: 0.4816 - val_accuracy: 0.8452
     Epoch 58/300
     7/7 [==============================] - 0s 11ms/step - loss: 0.5105 - accuracy: 0.7565 - val_loss: 0.4787 - val_accuracy: 0.8452
```

This is the final test that we did that we deemed satisfactory. First off, we started using regularizers to be added on the dense layers to have an additional measure to lessen overfitting. We also implemented early stopping to monitor if the loss hasn't improved in 10 epochs, which will then prompt it to stop the testing early. Additionally, from this stage of the testing, we wanted to experiment with starting off with a small number of nodes and just one layer and gradually start going up from there.

```
model.evaluate(xtest, ytest)

3/3 [==============================] - 0s 6ms/step - loss: 0.4765 - accuracy: 0.8571
[0.4765106737613678, 0.8571428656578064]

model.evaluate(xtrain,ytrain)

7/7 [==============================] - 0s 3ms/step - loss: 0.4506 - accuracy: 0.8135
[0.4506348669528961, 0.8134714961051941]

[19] import sklearn
     from sklearn.metrics import confusion_matrix
     ypred = (model.predict(xtest)>0.50).astype('int32')

     3/3 [==============================] - 0s 6ms/step

[20] confusion_matrix(ytest, ypred)

     array([[62,  4],
            [ 8, 10]])
```

Fortunately, downscaling the amount of layers and nodes produced astounding results and a massive improvement from the previous test in terms of both loss and accuracy. The test set performance seems to have improved compared to previous outputs, with a lower loss and higher accuracy. The training set performance also improved, which means that the model is able to generalize well to the test set. The accuracy of the test set is also acceptable as it is above 0.8.

In the confusion matrix, 62 is the number of true positives, 4 is the number of false positives, 8 is the number of false negatives, and 10 is the number of true negatives. From that, it can be said that the model has a good performance in terms of identifying true positives but may have less accuracy with false negatives.

All subsequent tests then yielded either diminishing returns or decline in the results which made us deem this test to be satisfactory as the final test.

Conclusion

Throughout this project, we learned a lot about choosing the right datasets to train a model on as well as being cognizant of the implications of the goals and purpose of the datasets that we find. We ultimately landed on the breast cancer data seeing as there is a lot of potential for analysis and exploration.

According to Barkved, K. (2022), an accuracy of 70% to 90% is a generally acceptable outcome for machine learning models which by the time we reached our 4th test, we reached the threshold of. Nevertheless, since the data is dealing with a serious disease, the consequences can be drastic if the prediction doesn't turn out to be accurate. Ultimately, we had our threshold for satisfactory results to be at 80-90% which our neural network model passed.

Going over the tests that we did, the first few tests that we conducted were getting our bearings on the data and trying to get the accuracy up by using more and more layers, but disregarding the loss values of the results. This resulted in unsatisfactory outcomes for both accuracy and loss values, with the accuracy only slightly going up by increments and the loss value following along the uptrend which wasn't a good sign. By the 4th test, we realized this mistake and started putting more emphasis on getting the loss down while still retaining accuracy, and this was done by being more conservative with the amount of layers and nodes that were used. From the fifth test, we concluded that by using dropout layers, more conservative amounts of nodes per layer, and a larger train-test split, we have been able to improve the model's test performance and reduce the overfitting. The subsequent tests were then about refining the results until a satisfactory result was reached.

Overall, the final test has shown that by using regularizers, early stopping, and starting with a small number of nodes and layers, we have been able to improve the model's performance, reduce overfitting, and achieve an acceptable level of accuracy. The bottom line is that less is more and having more layers and nodes on your model doesn't necessarily mean that it's going to be more accurate due to risk of overfitting and diminishing returns.

From that, we can conclude that even though the model is not perfect, it may still be accurate enough in some cases to predict if a female patient will have recurrent breast cancer. It may not be good enough in a medical setting as it may require a higher level of accuracy, but from a research standpoint, it is a point of interest to look more into.

References

Barkved, K. (March, 2022). How To Know if Your Machine Learning Model Has Good Performance. *Obviously AI.* Retrieved from https://www.obviously.ai/post/machine-learning-model-performance

Matjaz, Z., & Milan S. (1988). Breast Cancer. *UCI Machine Learning Repository*. Retrieved from https://archive-beta.ics.uci.edu/dataset/14/breast+cancer