CS 474 FINAL PROJECT: SKIN CANCER DETECTION
COMPUTER SCIENCE AND MATHEMATICS

RORY ARNONE-WHEAT

Date: December 12, 2024.

1. Introduction

The project I chose is the one for detecting skin cancer. I chose this one because my grandpa has been struggling with skin cancer for as long as I can remember and seeing the effects of skin cancer has made it so that options like this that work the improve early detection are incredibly appealing. For this project I can implement a CNN I can use keras in the same way that I did for assignment 4. This will allow me to set up the code quickly and focus on adjusting parameters to get the results I want on this project.

2. Proposed Method

Since this project is based on image recognition, just like the project on determining handwritten numbers, I decided to use a Convolutional Neural Network. As such, I used the keras package to build and implement the CNN. I initially set up the CNN with 3 convolutional layers, a max-pooling layer, a flattening layer, a fully connected layer with 128 nodes, and a binary output layer. I compiled it using the Adam optimizer and the binary cross-entropy loss function. Based off the results gained I added more nodes to the fully connected layer and expanded the amount of epochs.

3. Experimental Results

3.1. **Initial Model.** This project came with dataset preparation built into the Jupyter Notebook file, so I was able to skip that part in the development process. For the initial model I built I had 3 convolutional layers followed by a max-pooling layer (to speed up computations). I then used a flatten layer to convert the 3D mapping to a 1D vector before passing it to the fully connected layer that sends to the output layer which is a binary output. To compile I used the Adam optimizer with the binary cross-entropy loss function as this is a binary classification.

This model was run for 10 epochs which started at an accuracy of 62.70% and by the second epoch had reached a training accuracy of 75.82%. It jumped up to around an 80% accuracy for the third epoch and hovered around there until the 7th epoch where it hit an accuracy of 84.36%. The 8th epoch gave slightly inferior results, while the 9th gave an accuracy of 86.81%. The final epoch gave a training accuracy of 88.07%. This can be seen in figure 1.

The test accuracy was 81.96%. This can be seen in figure 2 and a graphical analysis of the accuracy can be seen in figure 3.

3.2. **First adjustments.** I adjusted the amount of nodes for the fully connected layer to 256 and changed the amount of epochs to 30. The training accuracy started at 57.95% but quickly jumped to 75-80%. It hovered around low to mid 80's on training accuracy until the 11th epoch when it broke through and started climbing to the upper 80's before breaking into the 90's by the 17th epoch. It ended around 97-98% training accuracy. This can be seen in figures 4 and 5.

The testing accuracy was 82.7%. This can be seen in figure 6.

From the Accuracy Graph, in figure 7, it looks like it is having some over-fitting problems.

3.3. **Final adjustments.** For the final run of the model I tried increasing the number of nodes on the fully connect layer to 512, which just drastically worsened the problem of the over-fitting. This resulted in an testing accuracy of 85% and can be seen in figure 8.

4. Conclusion

The second attempt was the best the testing accuracy got, but it clearly showed signs of over-fitting. Attempting to increase the neural node amount did not fix this issue and in fact exacerbated it. To improve and fix this issue would require me to look over constructing a CNN again as I am clearly misunderstanding something. Overall I met the expectations of the project, but unfortunately made some mistakes in execution.

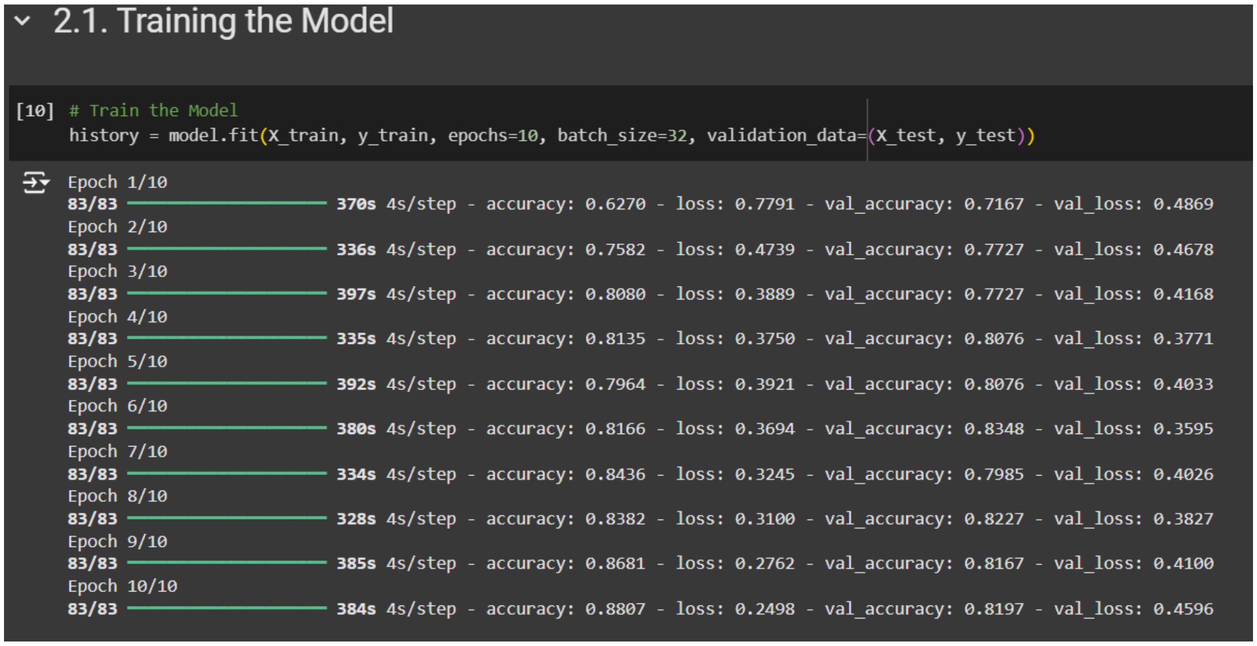
5. References

https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign

https://keras.io/guides/

6. Screenshots

I ended up having to put these images at the bottom of the document as I could not get the formatting for them to work properly otherwise.

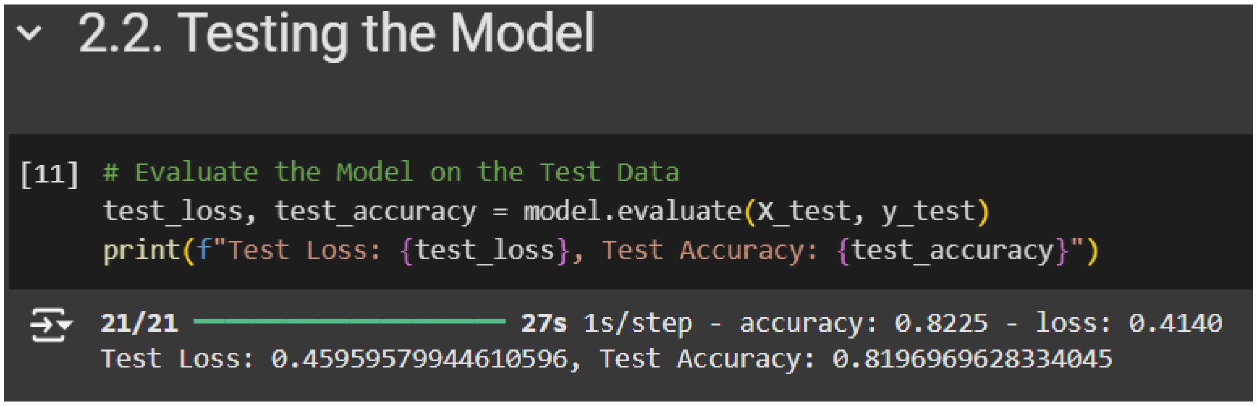


2.1. Training the Model

```
[10] # Train the Model
     history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

     Epoch 1/10
     83/83 ━━━━━━━━━━━━━━━━━━━━ 370s 4s/step - accuracy: 0.6270 - loss: 0.7791 - val_accuracy: 0.7167 - val_loss: 0.4869
     Epoch 2/10
     83/83 ━━━━━━━━━━━━━━━━━━━━ 336s 4s/step - accuracy: 0.7582 - loss: 0.4739 - val_accuracy: 0.7727 - val_loss: 0.4678
     Epoch 3/10
     83/83 ━━━━━━━━━━━━━━━━━━━━ 397s 4s/step - accuracy: 0.8080 - loss: 0.3889 - val_accuracy: 0.7727 - val_loss: 0.4168
     Epoch 4/10
     83/83 ━━━━━━━━━━━━━━━━━━━━ 335s 4s/step - accuracy: 0.8135 - loss: 0.3750 - val_accuracy: 0.8076 - val_loss: 0.3771
     Epoch 5/10
     83/83 ━━━━━━━━━━━━━━━━━━━━ 392s 4s/step - accuracy: 0.7964 - loss: 0.3921 - val_accuracy: 0.8076 - val_loss: 0.4033
     Epoch 6/10
     83/83 ━━━━━━━━━━━━━━━━━━━━ 380s 4s/step - accuracy: 0.8166 - loss: 0.3694 - val_accuracy: 0.8348 - val_loss: 0.3595
     Epoch 7/10
     83/83 ━━━━━━━━━━━━━━━━━━━━ 334s 4s/step - accuracy: 0.8436 - loss: 0.3245 - val_accuracy: 0.7985 - val_loss: 0.4026
     Epoch 8/10
     83/83 ━━━━━━━━━━━━━━━━━━━━ 328s 4s/step - accuracy: 0.8382 - loss: 0.3100 - val_accuracy: 0.8227 - val_loss: 0.3827
     Epoch 9/10
     83/83 ━━━━━━━━━━━━━━━━━━━━ 385s 4s/step - accuracy: 0.8681 - loss: 0.2762 - val_accuracy: 0.8167 - val_loss: 0.4100
     Epoch 10/10
     83/83 ━━━━━━━━━━━━━━━━━━━━ 384s 4s/step - accuracy: 0.8807 - loss: 0.2498 - val_accuracy: 0.8197 - val_loss: 0.4596
```

Figure 1. Training Data



2.2. Testing the Model

```
[11] # Evaluate the Model on the Test Data
     test_loss, test_accuracy = model.evaluate(X_test, y_test)
     print(f"Test Loss: {test_loss}, Test Accuracy: {test_accuracy}")
```

```
21/21 ━━━━━━━━━━━━━━━━━━━━ 27s 1s/step - accuracy: 0.8225 - loss: 0.4140
Test Loss: 0.45959579944610596, Test Accuracy: 0.8196969628334045
```

Figure 2. Testing Data

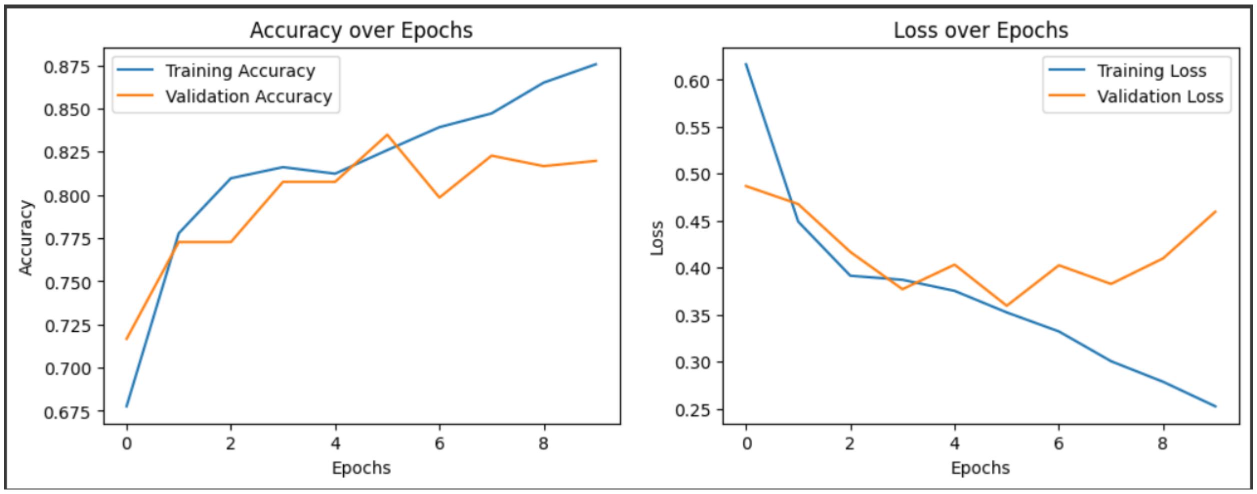
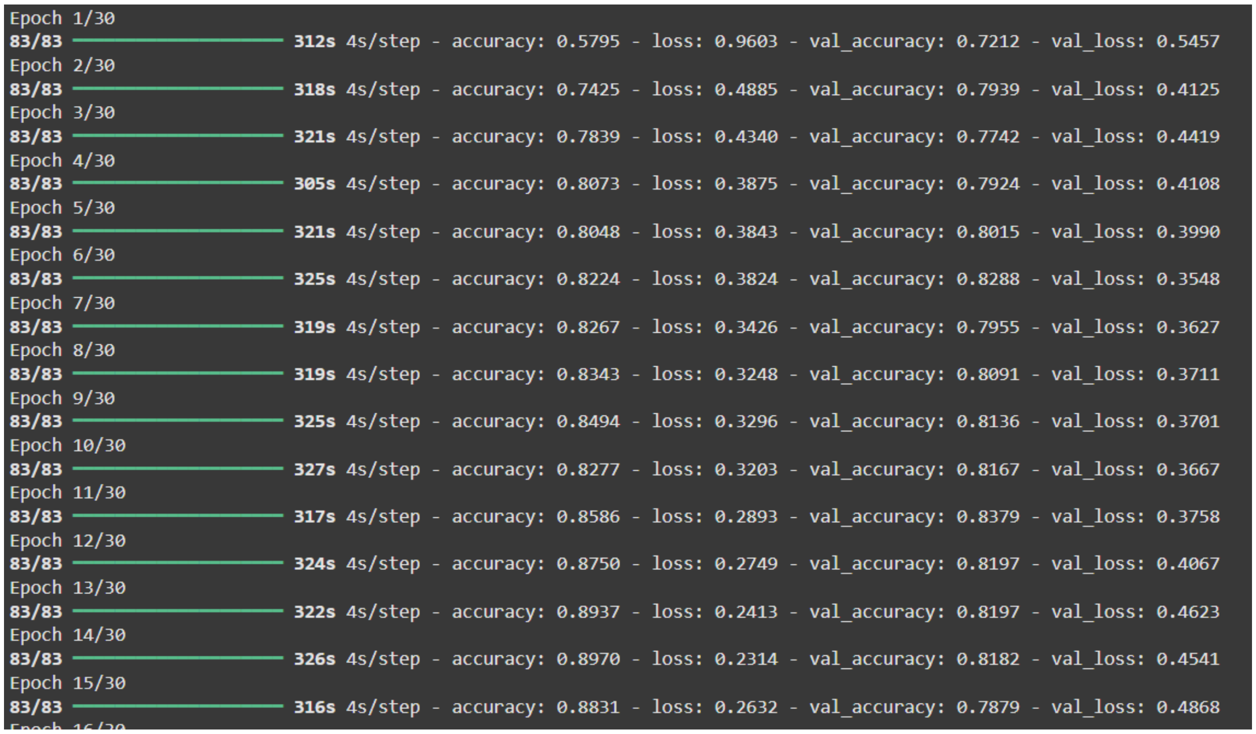
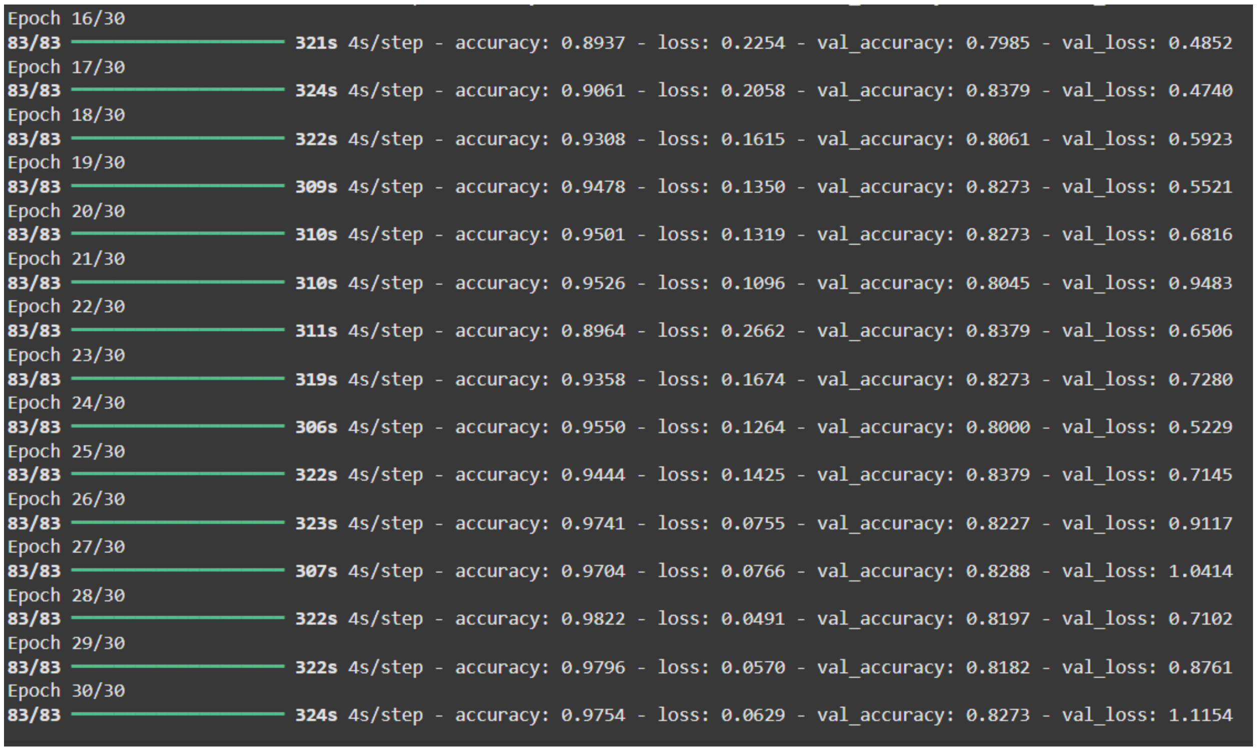


Figure 3. Accuracy and Loss Graph



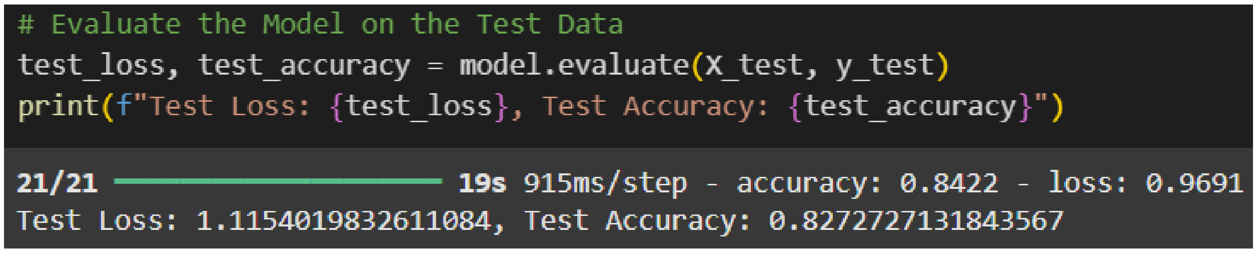
```
Epoch 1/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 312s 4s/step - accuracy: 0.5795 - loss: 0.9603 - val_accuracy: 0.7212 - val_loss: 0.5457
Epoch 2/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 318s 4s/step - accuracy: 0.7425 - loss: 0.4885 - val_accuracy: 0.7939 - val_loss: 0.4125
Epoch 3/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 321s 4s/step - accuracy: 0.7839 - loss: 0.4340 - val_accuracy: 0.7742 - val_loss: 0.4419
Epoch 4/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 305s 4s/step - accuracy: 0.8073 - loss: 0.3875 - val_accuracy: 0.7924 - val_loss: 0.4108
Epoch 5/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 321s 4s/step - accuracy: 0.8048 - loss: 0.3843 - val_accuracy: 0.8015 - val_loss: 0.3990
Epoch 6/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 325s 4s/step - accuracy: 0.8224 - loss: 0.3824 - val_accuracy: 0.8288 - val_loss: 0.3548
Epoch 7/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 319s 4s/step - accuracy: 0.8267 - loss: 0.3426 - val_accuracy: 0.7955 - val_loss: 0.3627
Epoch 8/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 319s 4s/step - accuracy: 0.8343 - loss: 0.3248 - val_accuracy: 0.8091 - val_loss: 0.3711
Epoch 9/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 325s 4s/step - accuracy: 0.8494 - loss: 0.3296 - val_accuracy: 0.8136 - val_loss: 0.3701
Epoch 10/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 327s 4s/step - accuracy: 0.8277 - loss: 0.3203 - val_accuracy: 0.8167 - val_loss: 0.3667
Epoch 11/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 317s 4s/step - accuracy: 0.8586 - loss: 0.2893 - val_accuracy: 0.8379 - val_loss: 0.3758
Epoch 12/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 324s 4s/step - accuracy: 0.8750 - loss: 0.2749 - val_accuracy: 0.8197 - val_loss: 0.4067
Epoch 13/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.8937 - loss: 0.2413 - val_accuracy: 0.8197 - val_loss: 0.4623
Epoch 14/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 326s 4s/step - accuracy: 0.8970 - loss: 0.2314 - val_accuracy: 0.8182 - val_loss: 0.4541
Epoch 15/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 316s 4s/step - accuracy: 0.8831 - loss: 0.2632 - val_accuracy: 0.7879 - val_loss: 0.4868
```

Figure 4. Training Data 2.1



```
Epoch 16/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 321s 4s/step - accuracy: 0.8937 - loss: 0.2254 - val_accuracy: 0.7985 - val_loss: 0.4852
Epoch 17/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 324s 4s/step - accuracy: 0.9061 - loss: 0.2058 - val_accuracy: 0.8379 - val_loss: 0.4740
Epoch 18/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.9308 - loss: 0.1615 - val_accuracy: 0.8061 - val_loss: 0.5923
Epoch 19/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 309s 4s/step - accuracy: 0.9478 - loss: 0.1350 - val_accuracy: 0.8273 - val_loss: 0.5521
Epoch 20/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 310s 4s/step - accuracy: 0.9501 - loss: 0.1319 - val_accuracy: 0.8273 - val_loss: 0.6816
Epoch 21/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 310s 4s/step - accuracy: 0.9526 - loss: 0.1096 - val_accuracy: 0.8045 - val_loss: 0.9483
Epoch 22/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 311s 4s/step - accuracy: 0.8964 - loss: 0.2662 - val_accuracy: 0.8379 - val_loss: 0.6506
Epoch 23/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 319s 4s/step - accuracy: 0.9358 - loss: 0.1674 - val_accuracy: 0.8273 - val_loss: 0.7280
Epoch 24/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 306s 4s/step - accuracy: 0.9550 - loss: 0.1264 - val_accuracy: 0.8000 - val_loss: 0.5229
Epoch 25/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.9444 - loss: 0.1425 - val_accuracy: 0.8379 - val_loss: 0.7145
Epoch 26/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 323s 4s/step - accuracy: 0.9741 - loss: 0.0755 - val_accuracy: 0.8227 - val_loss: 0.9117
Epoch 27/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 307s 4s/step - accuracy: 0.9704 - loss: 0.0766 - val_accuracy: 0.8288 - val_loss: 1.0414
Epoch 28/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.9822 - loss: 0.0491 - val_accuracy: 0.8197 - val_loss: 0.7102
Epoch 29/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.9796 - loss: 0.0570 - val_accuracy: 0.8182 - val_loss: 0.8761
Epoch 30/30
83/83 ━━━━━━━━━━━━━━━━━━━━ 324s 4s/step - accuracy: 0.9754 - loss: 0.0629 - val_accuracy: 0.8273 - val_loss: 1.1154
```

Figure 5. Training Data 2.2



```
# Evaluate the Model on the Test Data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_accuracy}")

21/21 ━━━━━━━━━━━━━━━━━━━━ 19s 915ms/step - accuracy: 0.8422 - loss: 0.9691
Test Loss: 1.1154019832611084, Test Accuracy: 0.8272727131843567
```

FIGURE 6. Testing Data 2

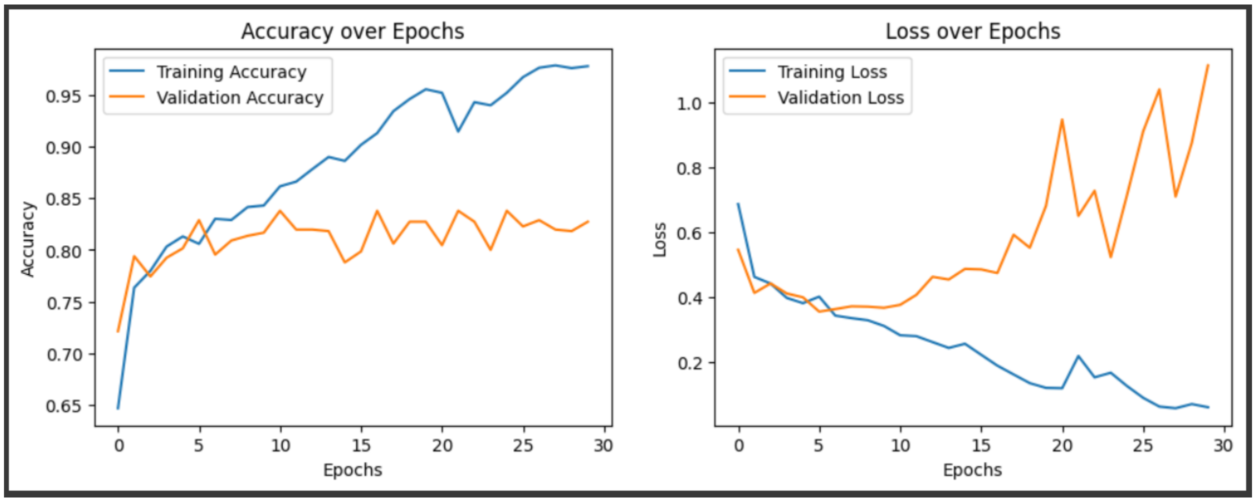


FIGURE 7. Acuracy and Loss Graph 2

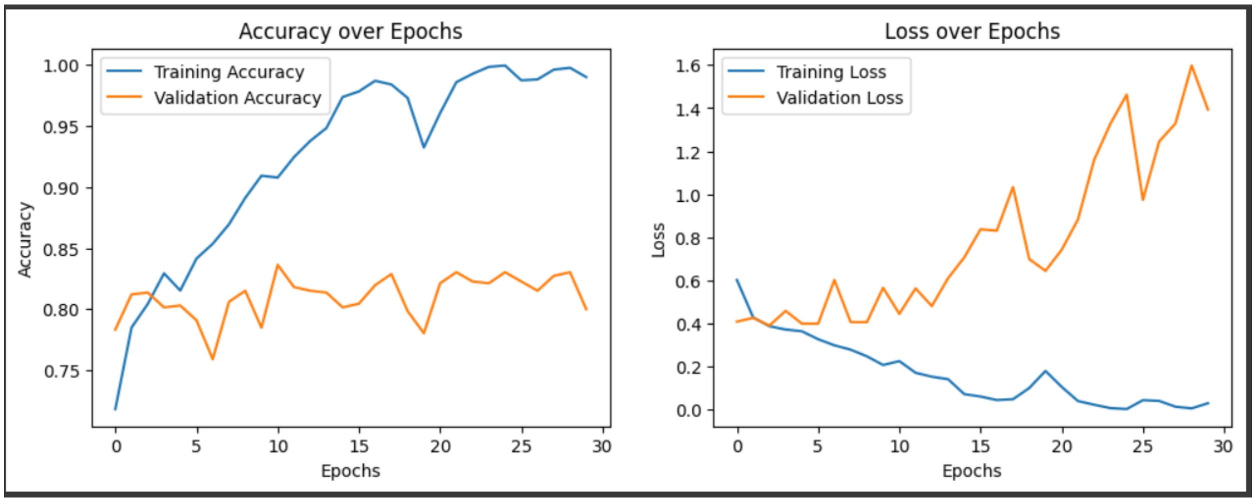


FIGURE 8. Acuracy and Loss Graph 3