Project 1

Leaf Classification

Name: - Amany Azzam

ID: - 20399133

Supervised by

Prof. Hazem Abbas

Part I: Data Preparation

1. Describe the data

Training Data contain 990 rows, 193 columns

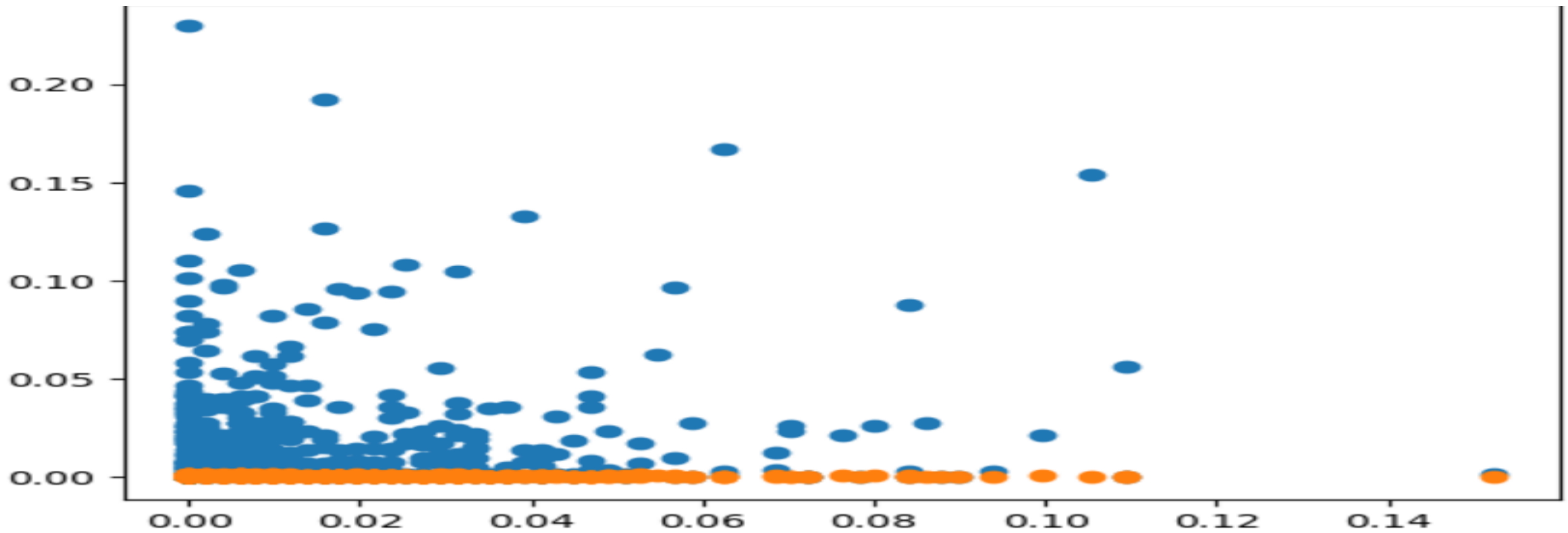
2. Clean the data

Data not contain missing values and there aren't contain outliers

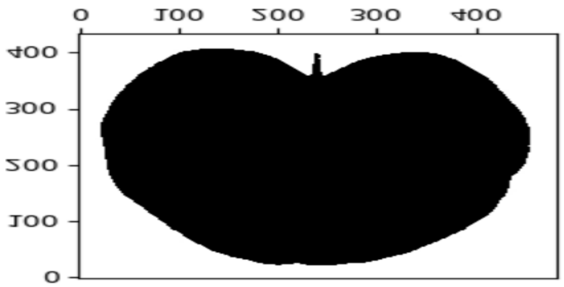
3. Check the data for missing values or duplicates

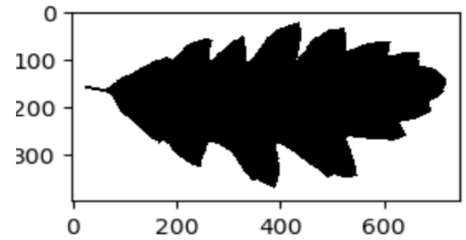
no missing values and no duplicates

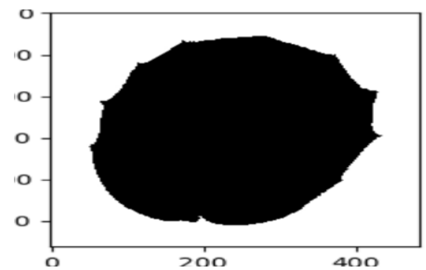
4. Visualize the data using proper visualization methods.



5. Draw some of the images







6. Carry out required correlation analysis

```
corr = df.corr()
corr.style.background_gradient(cmap="Spectral")
```

	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	margin10
margin1	1.000000	0.806390	-0.182829	-0.297807	-0.475874	0.767718	0.066273	-0.094137	-0.181496	0.397138
margin2	0.806390	1.000000	-0.204640	-0.315953	-0.444312	0.825762	-0.083273	-0.086428	-0.120276	0.162587
margin3	-0.182829	-0.204640	1.000000	0.120042	-0.185007	-0.163976	0.095449	0.024350	-0.000042	0.008772
margin4	-0.297807	-0.315953	0.120042	1.000000	0.029480	-0.261437	-0.268271	-0.047693	0.227543	-0.173986
margin5	-0.475874	-0.444312	-0.185007	0.029480	1.000000	-0.438587	-0.108178	0.056557	0.196745	-0.320647
margin6	0.767718	0.825762	-0.163976	-0.261437	-0.438587	1.000000	-0.093780	-0.112896	-0.136961	0.215141

7. divide the data into a training and test set

divide data to 80% for training and 20% for test

8. standardize the data

It is clear that the data is already normalized

Part II: Training a neural network

- **We decided to tune the following hyperparameter**
  - Batch Size
  - Hidden Node Size
  - Drop Rate
  - Optimizer
- Define Function that contain the Deep Learning architecture That has 3-layer MLP model (one input layer, one hidden layer with tanh activation and one output layer)

```
from tensorflow.keras.optimizers import Adam
def fun_model(optim = Adam() , bat_size = 32,  hid_nodes = 512, drop_rate = 0.5):

  model = Sequential()
  model.add(Dense(hid_nodes, activation='tanh', input_shape=(192,), kernel_initializer = 'glorot_uniform', bias_initializer='zeros', name = 'Layer_1'))
  model.add(Dropout(drop_rate))
  model.add(Dense(99 , activation='softmax', name = 'Output'))

  model.compile(optimizer = optim ,loss='sparse_categorical_crossentropy' , metrics=['accuracy'])

  history = model.fit(X_train , y_train , epochs=100 , batch_size=bat_size , validation_data=(X_val, y_val))


  return model, history
```

- Define Function for Draw accuracy for training against accuracy for test

```
def acc_(model_name, history):
    model_history = pd.DataFrame(history.history)
    model_history['epoch'] = history.epoch
    fig, ax = plt.subplots(figsize=(14,8))
    num_epochs = model_history.shape[0]
    ax.plot(np.arange(0, num_epochs), model_history["accuracy"], label="Training accuracy", lw=3, color='#f4b400')
    ax.plot(np.arange(0, num_epochs), model_history["val_accuracy"], label="Validation accuracy", lw=3, color='#0f9d58')
    ax.legend()
    plt.tight_layout()
    plt.show()
```

- And Define Function for Draw loss for training against loss for test

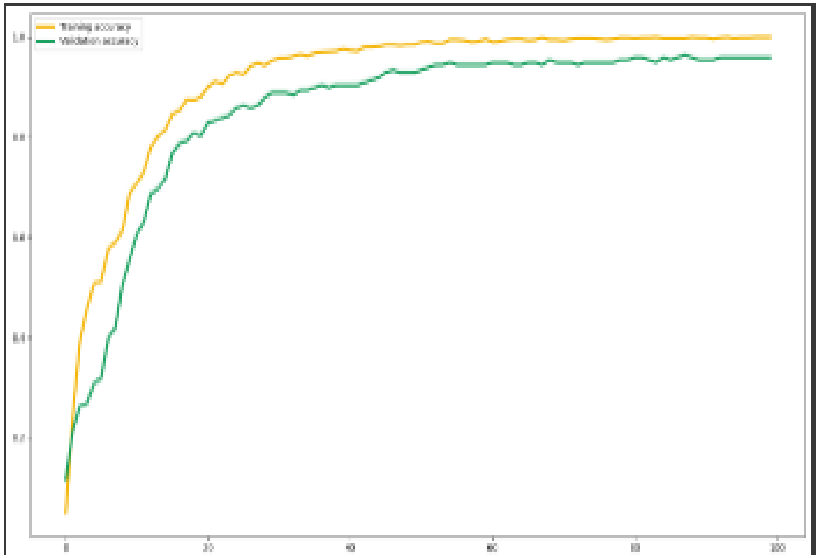
```
def loss_(model_name, history):
    model_history = pd.DataFrame(history.history)
    model_history['epoch'] = history.epoch
    fig, ax = plt.subplots(figsize=(14,8))
    num_epochs = model_history.shape[0]
    ax.plot(np.arange(0, num_epochs), model_history["loss"], label="Training loss", lw=3, color='#f4b400')
    ax.plot(np.arange(0, num_epochs), model_history["val_loss"], label="Validation loss", lw=3, color='#0f9d58')
    ax.legend()
    plt.tight_layout()
    plt.show()
```

The trails with Adam Optimizer

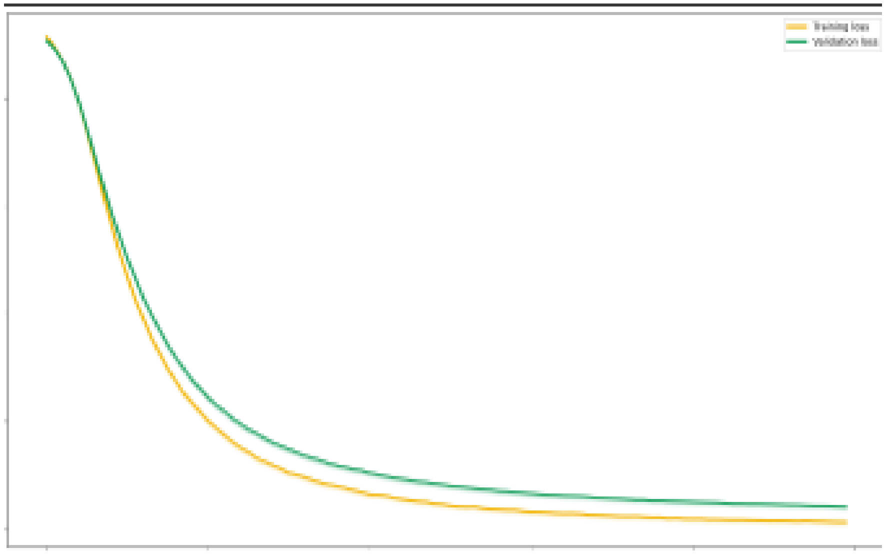
Trail_1: - I used Adam optimizer in our model with 32 batch size 512 hidden nodes and 0.5 drop out ratio

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=1e-3, decay_steps=10000, decay_rate=0.9)
adam = Adam(learning_rate = lr_schedule)
model_1, history_1 = fun_model(adam , 32)
model_1.summary()
model_1.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

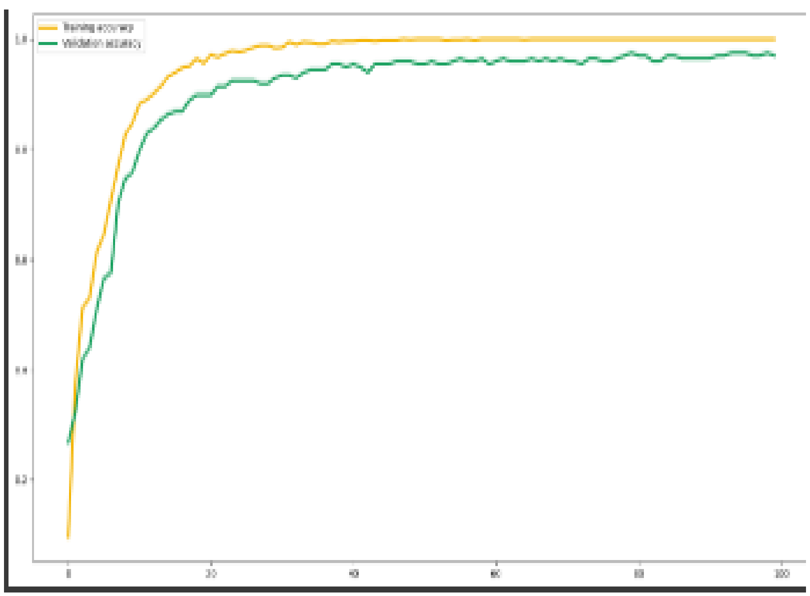


[loss: - 0.192669329404831, accuracy: - 0.959595782600403]

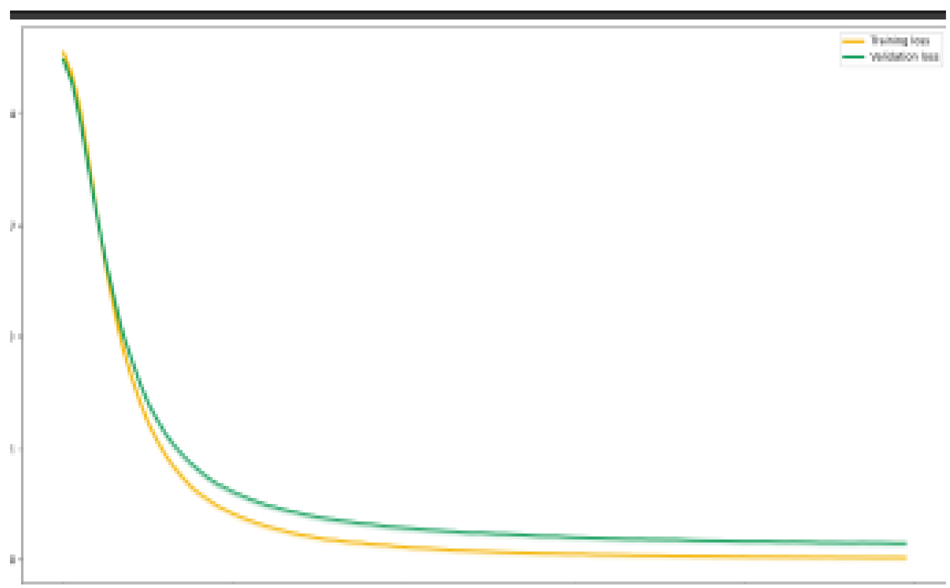
Trail_2: - I used Adam optimizer in our model with 16 batch size and 0.3 drop out ratio

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=1e-3, decay_steps=10000, decay_rate=0.9)
adam = Adam(learning_rate = lr_schedule)
model_2, history_2 = fun_model(adam , 16 , 512 , 0.3)
model_2.summary()
model_2.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

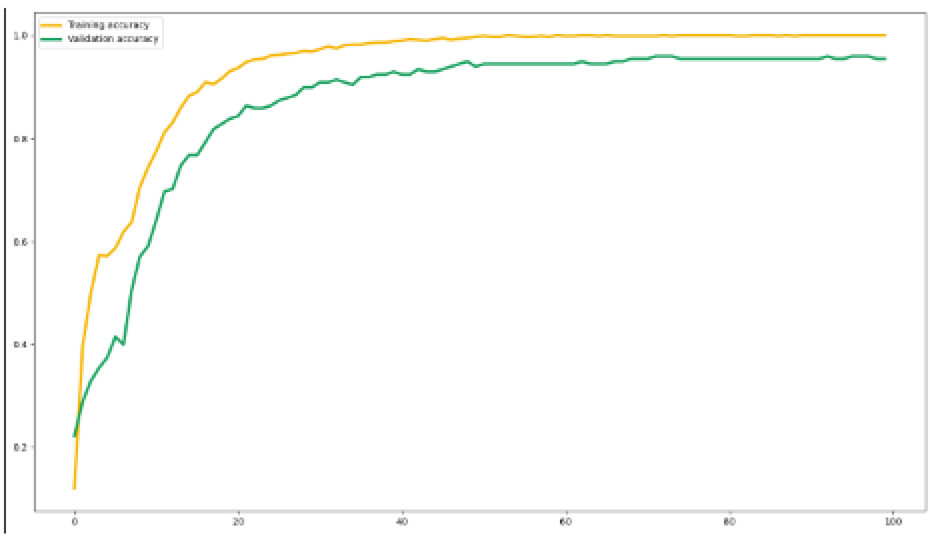


[loss: - 0.13997921347618103, accuracy: - 0.9696969985961914]

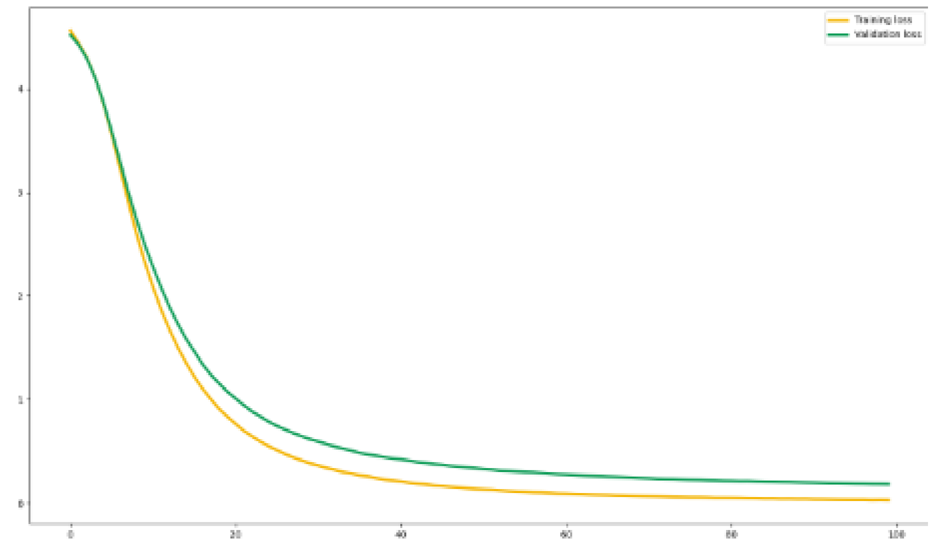
Trail_3: - I used Adam optimizer in our model with 64 batch size, 1024 hidden nodes and 0.2 drop out ratio

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=1e-3, decay_steps=10000, decay_rate=0.9)
adam = Adam(learning_rate = lr_schedule)
model_3, history_3 = fun_model(adam , 64 , 1024 , 0.2)
model_3.summary()
model_3.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

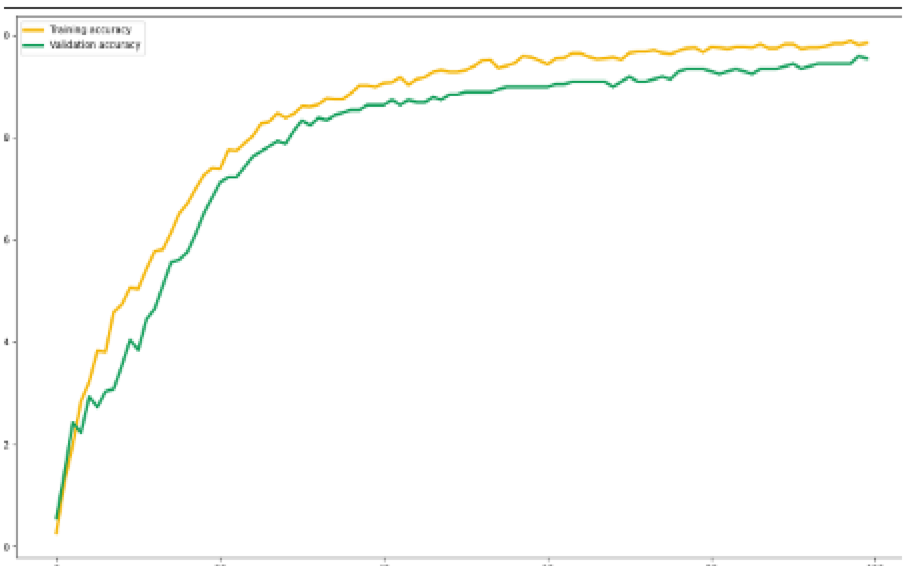


[loss: - 0.18521615862846375, accuracy: - 0.9545454382896423]

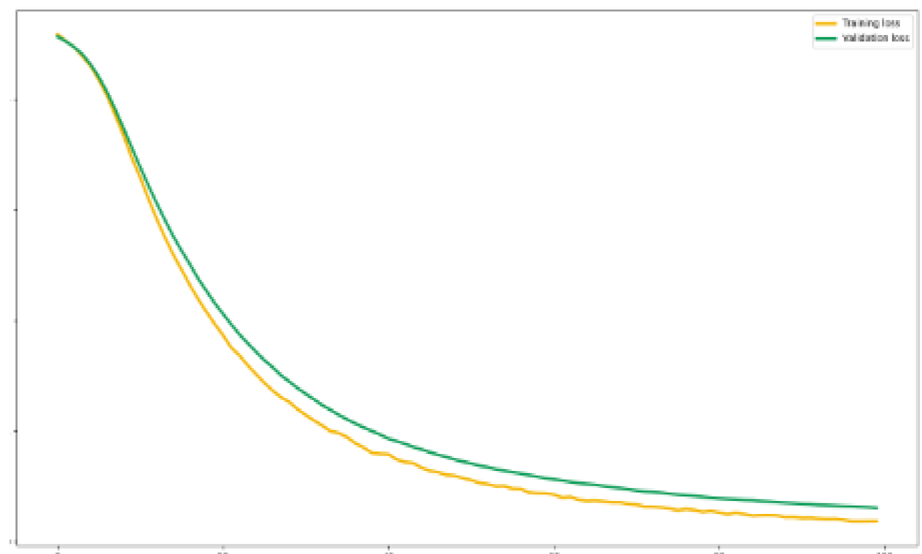
Trail_4: - I used Adam optimizer in our model with 32 batch size, 256 hidden nodes and 0.6 drop out ratio

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=1e-3, decay_steps=10000, decay_rate=0.9)
adam = Adam(learning_rate = lr_schedule)
model_4, history_4 = fun_model(adam , 32 , 256 , 0.6)
model_4.summary()
model_4.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

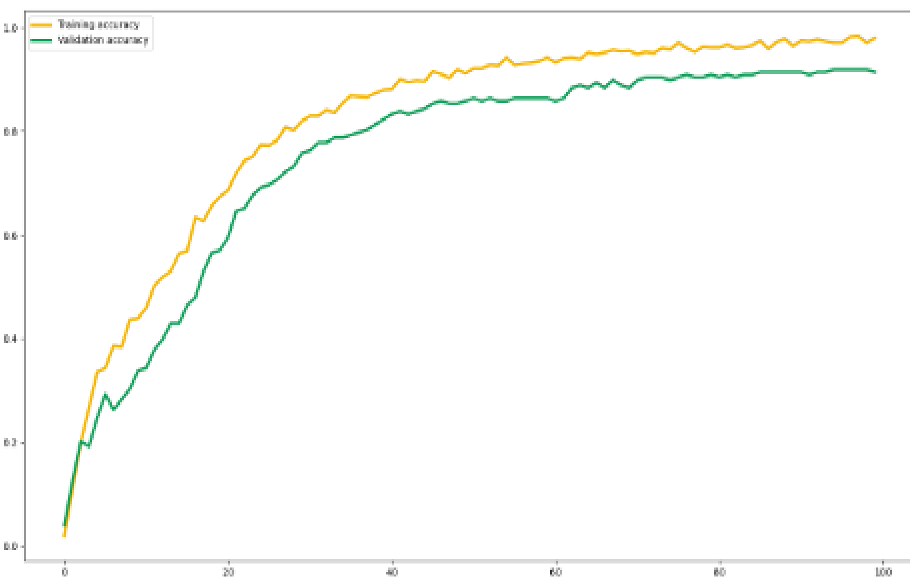


[loss: - 0.3047192096710205, accuracy: - 0.954545438289642З]

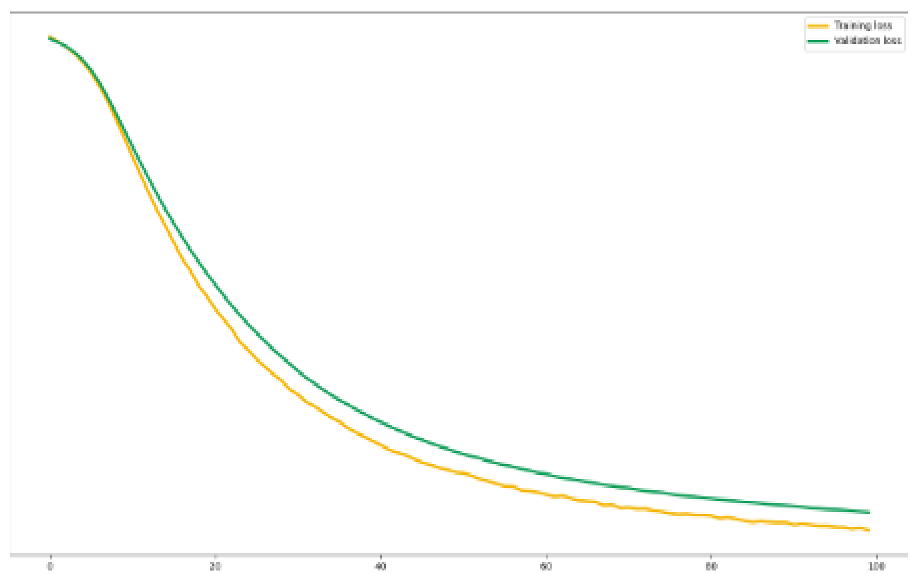
Trail_5: - I used Adam optimizer in our model with 32 batch size, 128 hidden nodes and 0.4 drop out ratio

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=1e-3, decay_steps=10000, decay_rate=0.9)
adam = Adam(learning_rate = lr_schedule)
model_5, history_5 = fun_model(adam , 32 , 128 , 0.4)
model_5.summary()
model_5.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve



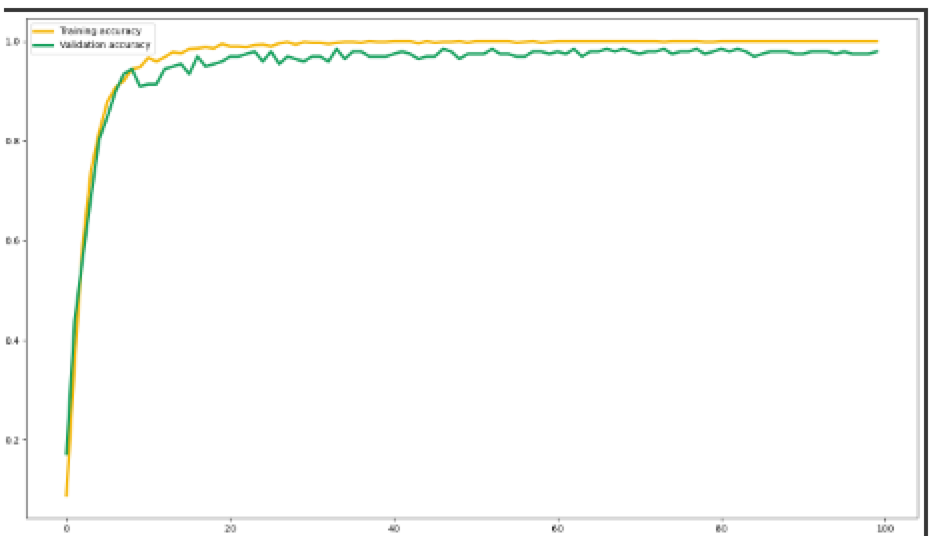
[loss: - 0.391522616147995, accuracy: - 0.9141414165496826]

The Trials with RMSProp optimizer

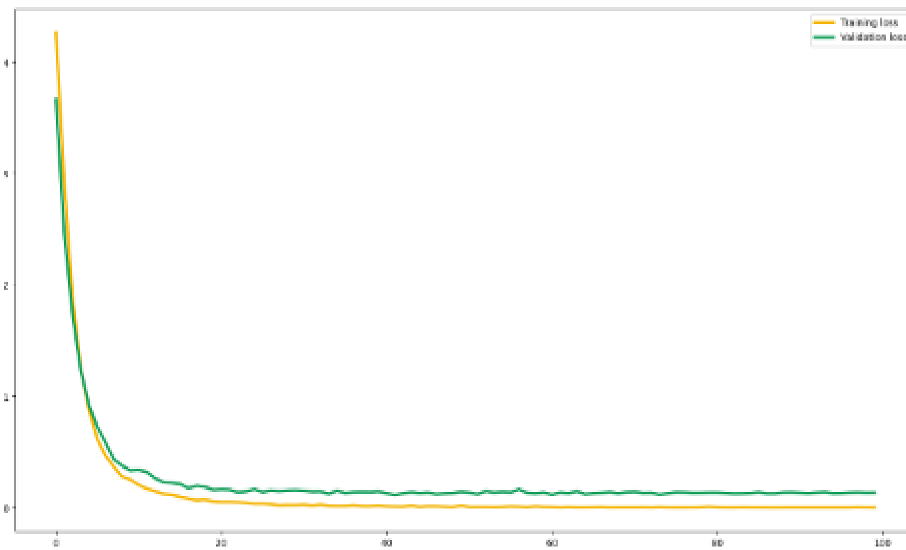
Trail_1: - I used RMSProp optimizer with 32 batch size and 0.5 drop out ratio.

```
rms_prop = tf.keras.optimizers.RMSprop(learning_rate=0.01,rho=0.9,momentum=0.0,epsilon=1e-07,centered=False, name="RMSprop")

model_6,history_6 = fun_model(rms_prop , 32 , 512 , 0.5)

model_6.summary()

model_6.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

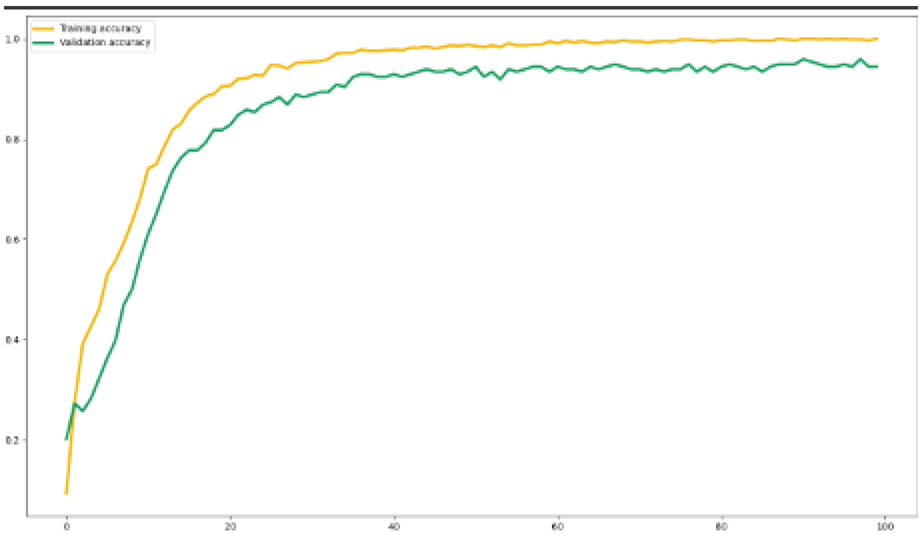


[loss: - 0.1346735656261444, accuracy: - 0.9797979593276978]

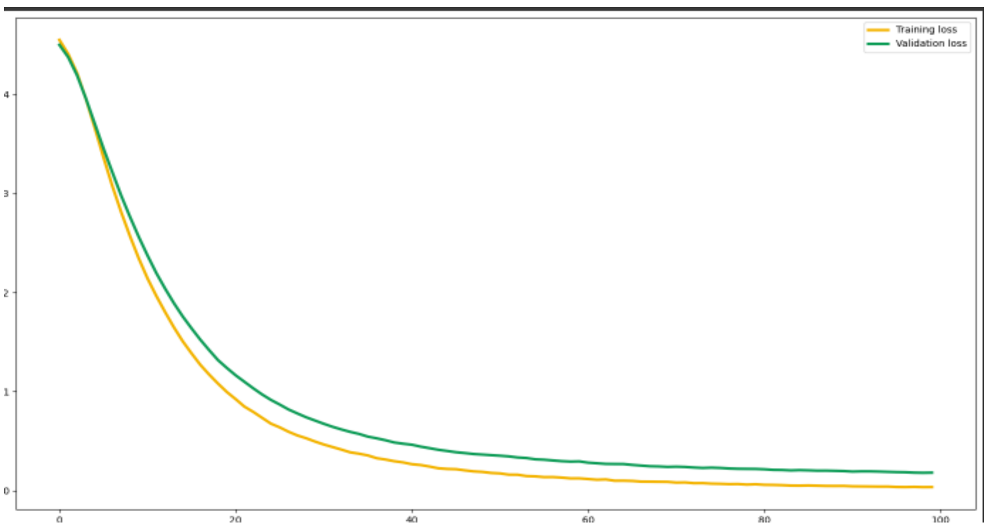
Trail_2: - I used RMSProp optimizer with 16 batch size and 0.3 drop out

```
rms_prop = tf.keras.optimizers.RMSprop(learning_rate=0.001,rho=0.9,momentum=0.0,epsilon=1e-07,centered=False, name="RMSprop")

model_7,history_7 = fun_model(rms_prop , 16 , 512 , 0.3)

model_7.summary()

model_7.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

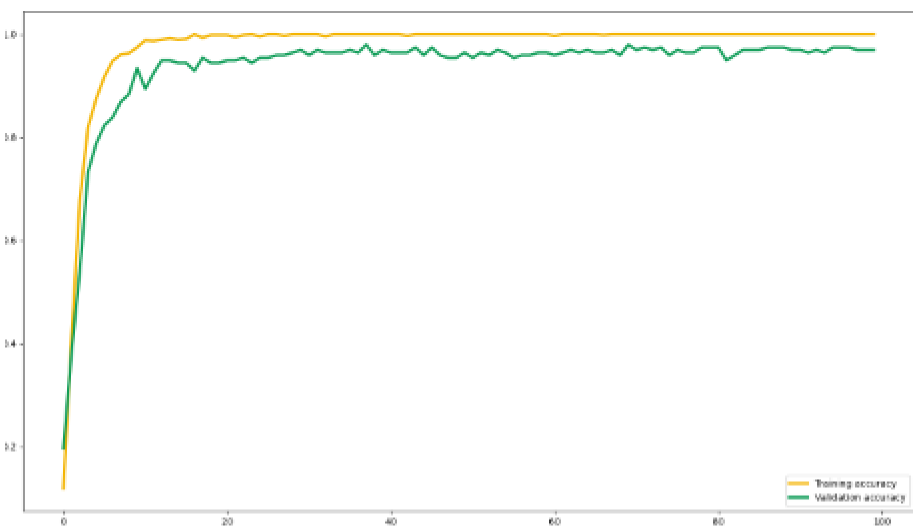


[loss: - 0.1821233183145523, accuracy: - 0.9444444179534912]

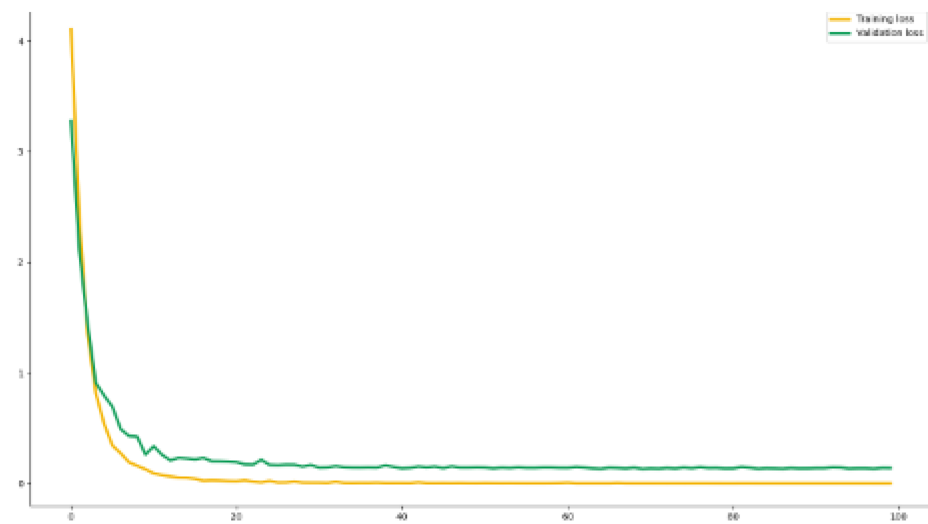
Trail_3: - I used RMSProp optimizer with 64 batch size 1024 hidden nodes and 0.2 drop out

```
rms_prop = tf.keras.optimizers.RMSprop(learning_rate=0.01,rho=0.9,momentum=0.0,epsilon=1e-07,centered=False, name="RMSprop")

model_8,history_8 = fun_model(rms_prop , 46 , 1024 , 0.2)

model_8.summary()

model_8.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

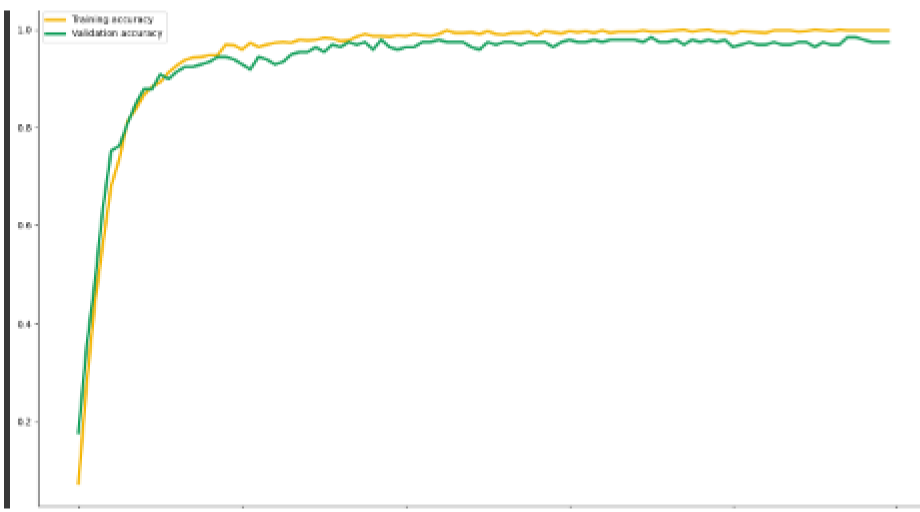


[loss: - 0.13985563640594448, accuracy: - 0.9696969985961914]

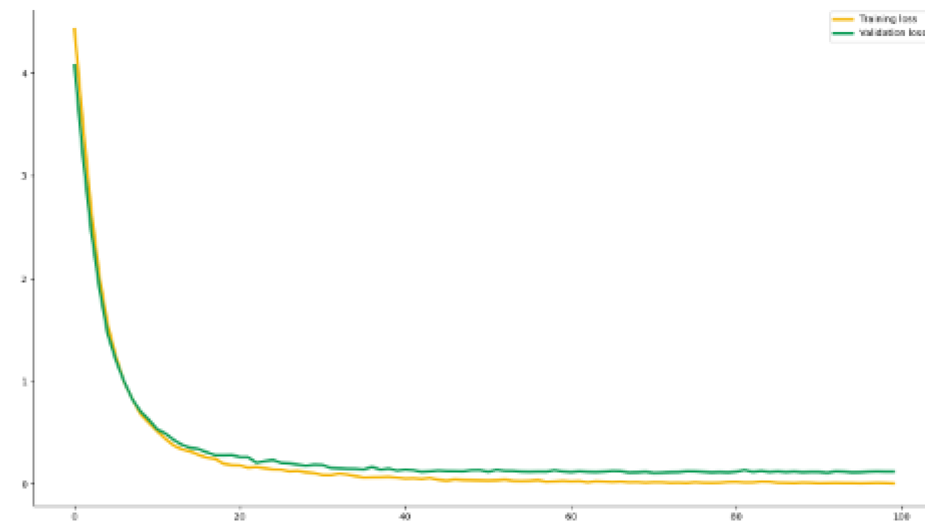
Trail_4: - I used RMSProp optimizer with 32 batch size 256 hidden nodes and 0.6 drop out ratio.

```
rms_prop = tf.keras.optimizers.RMSprop(learning_rate=0.01,rho=0.9,momentum=0.0,epsilon=1e-07,centered=False, name="RMSprop")

model_9,history_9 = fun_model(rms_prop , 32 , 256 , 0.6)

model_9.summary()

model_9.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

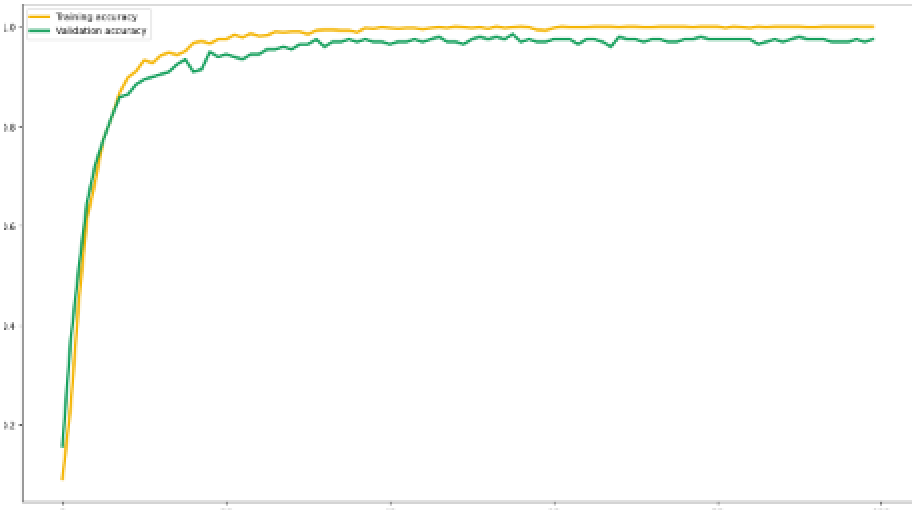


[loss: - 0.12103080749511719, accuracy: - 0.9747474789619446]

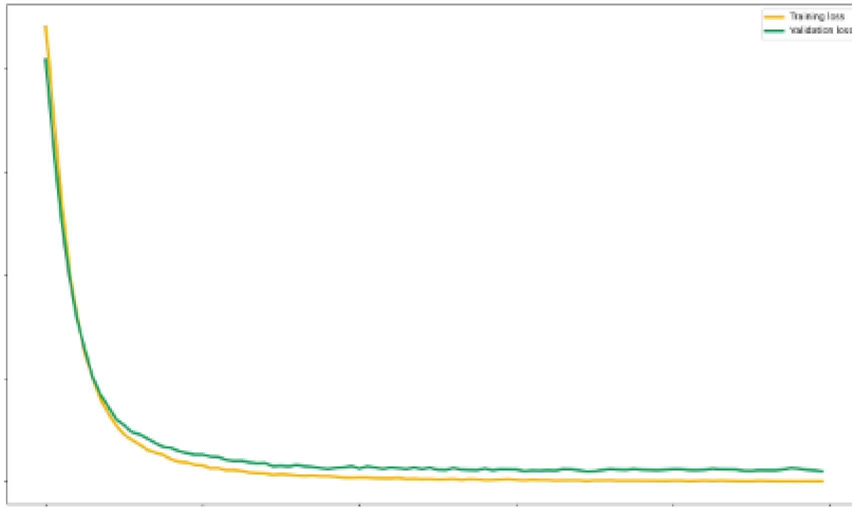
Trail_5: - I used RMSProp optimizer with 32 batch size 128 hidden nodes and 0.4 drop out ratio.

```
rms_prop = tf.keras.optimizers.RMSprop(learning_rate=0.01,rho=0.9,momentum=0.0,epsilon=1e-07,centered=False, name="RMSprop")

model_10,history_10 = fun_model(rms_prop , 32 , 128 , 0.4)

model_10.summary()

model_10.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve



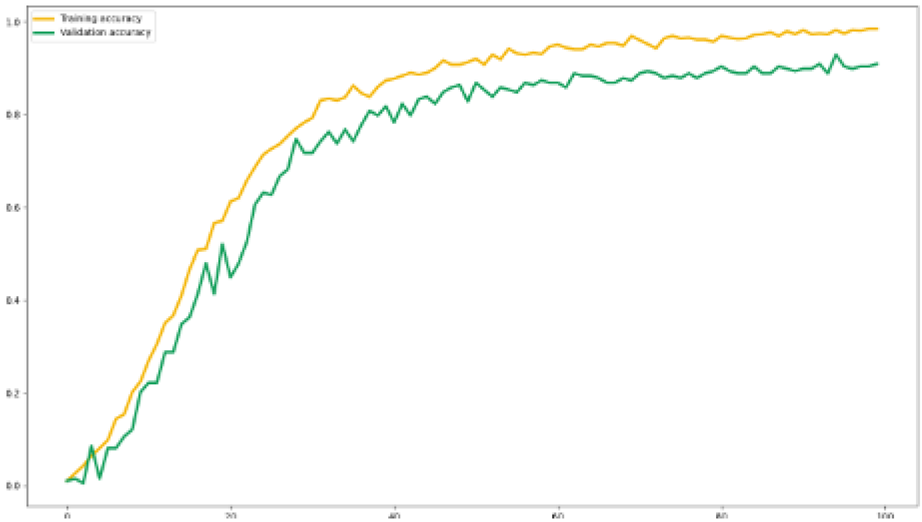
[loss: - 0.09994658827781677, accuracy: - 0.9747474789619446]

The Trials with SGD optimizer

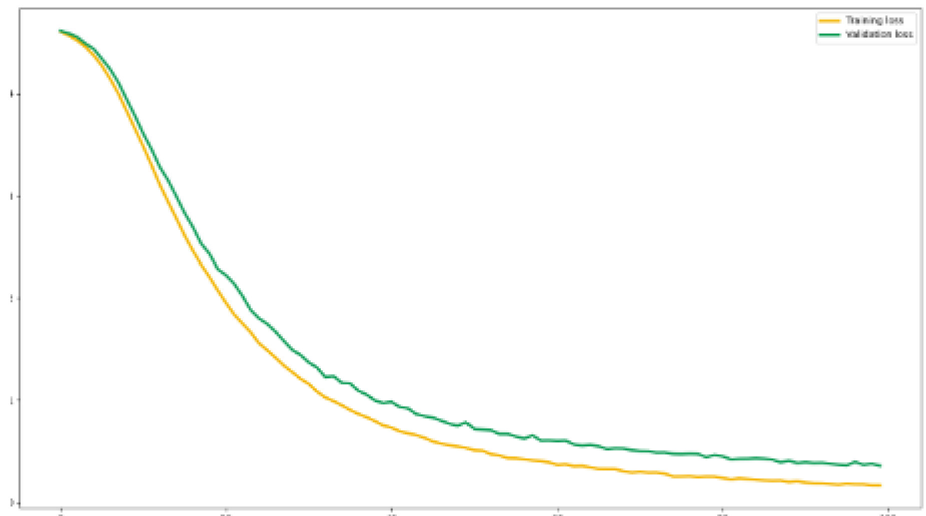
Trail_1: - I used SGD optimizer with 32 batch size and 0.5 drop out ratio.

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.9, momentum=0.0, nesterov=False, name="SGD")

model_11,history_11 = fun_model(sgd ,32, 512 , 0.5 )

model_11.summary()
model_11.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

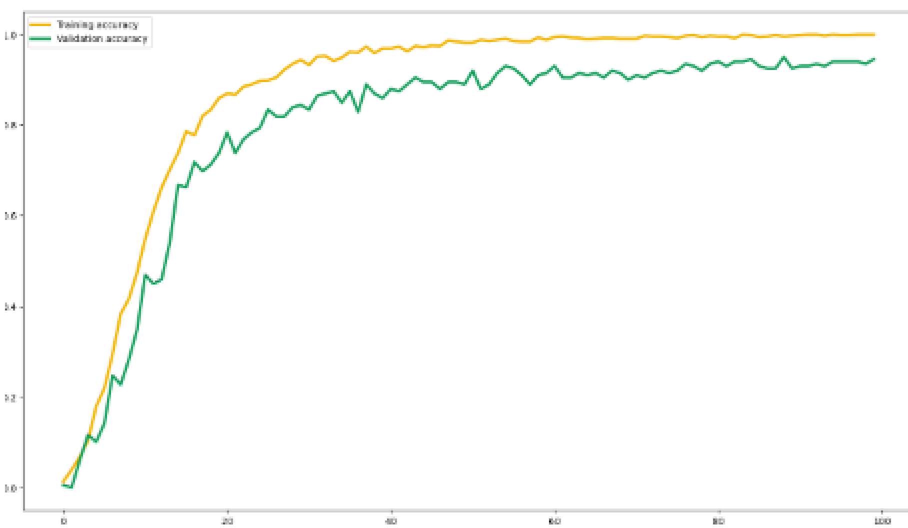


[loss: - 0.3594973087310791, accuracy: - 0.9090909361839294]

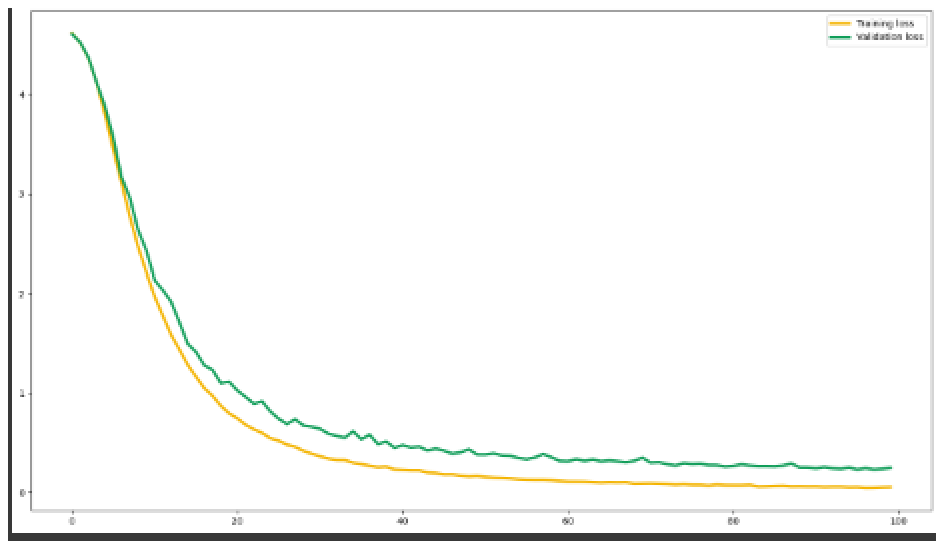
Trail_2: - I used SGD optimizer with 16 batch size and 0.3 drop out ratio.

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.9, momentum=0.0, nesterov=False, name="SGD")

model_12,history_12 = fun_model(sgd ,16, 512 , 0.3 )

model_12.summary()
model_12.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

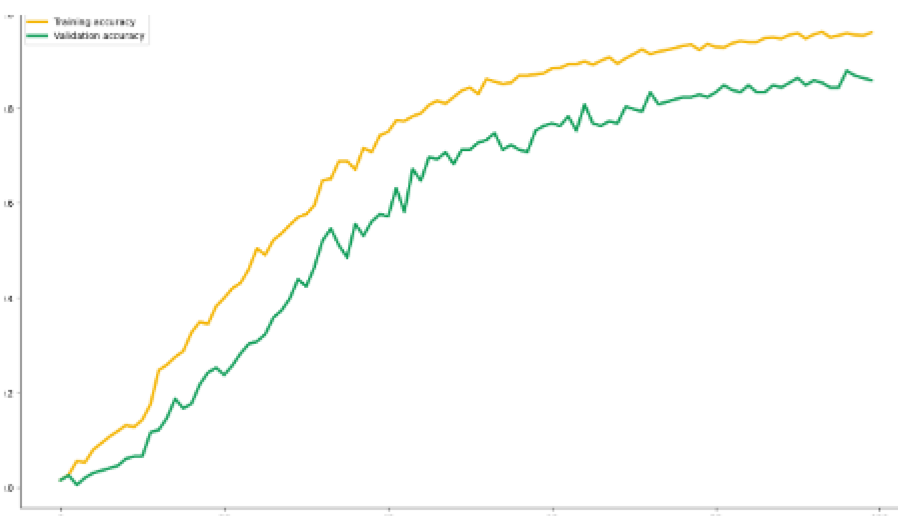


[loss: - 0.24715982377529144, accuracy: - 0.9444444179534912]

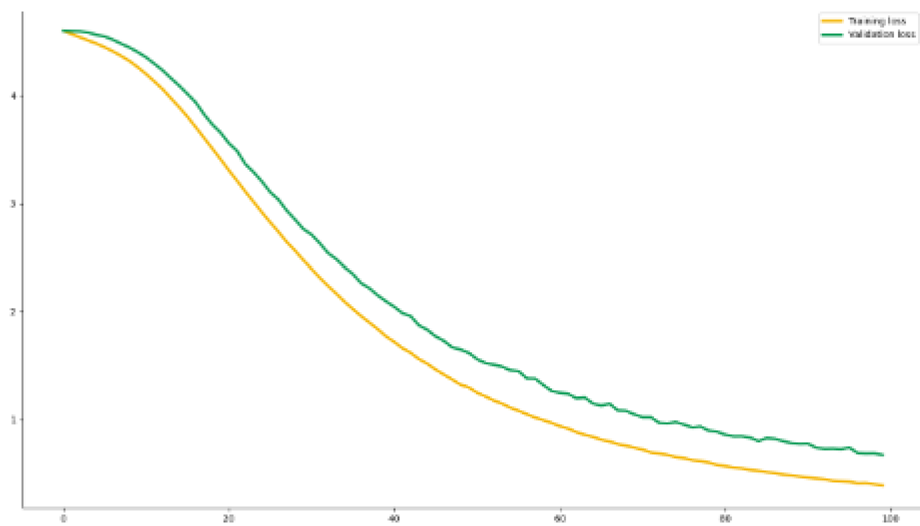
Trail_3: - I used SGD optimizer with 64 batch size 1024 hidden nodes and 0.2 drop out ratio.

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.9, momentum=0.0, nesterov=False, name="SGD")

model_13,history_13 = fun_model(sgd ,64, 1024 , 0.2 )

model_13.summary()
model_13.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

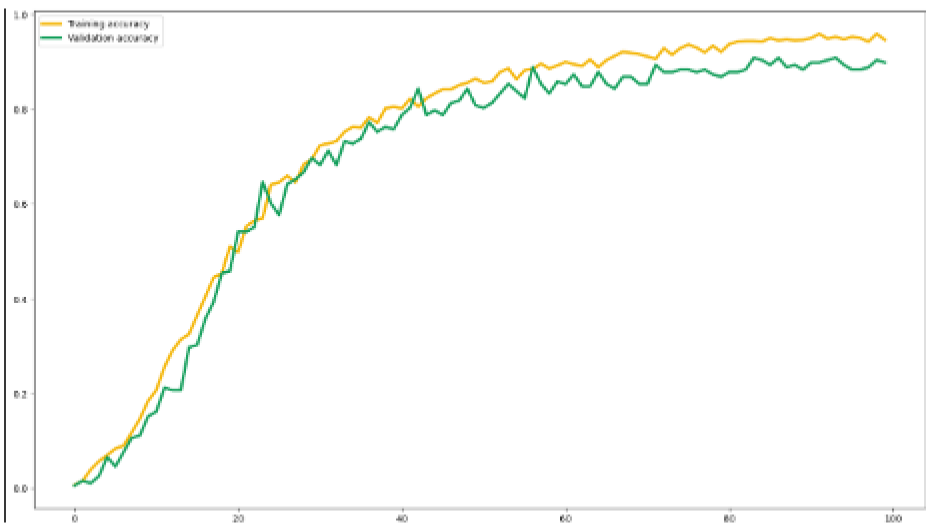


[loss: - 0.6672252416610718, accuracy: - 0.8585858345031738]

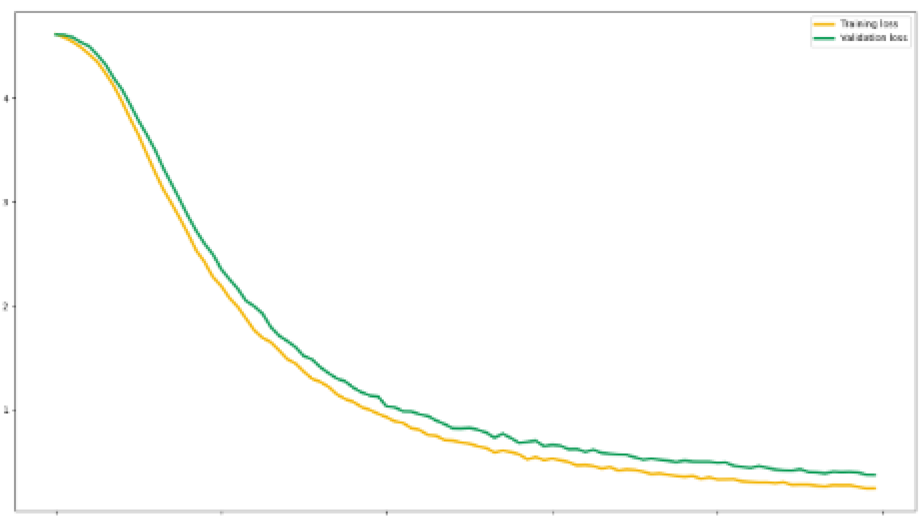
Trail_4: - I used SGD optimizer with 32 batch size 256 hidden nodes and 0.6 drop out ratio.

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.9, momentum=0.0, nesterov=False, name="SGD")

model_14,history_14 = fun_model(sgd ,32, 256 , 0.6 )

model_14.summary()
model_14.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve

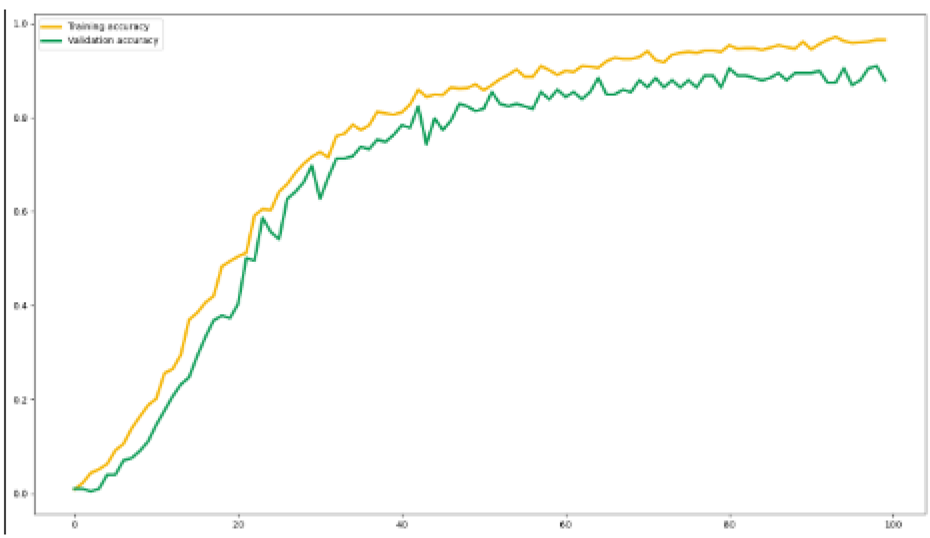


[loss: - 0.3718492984771 7285, accuracy: - 0.8989899158477783]

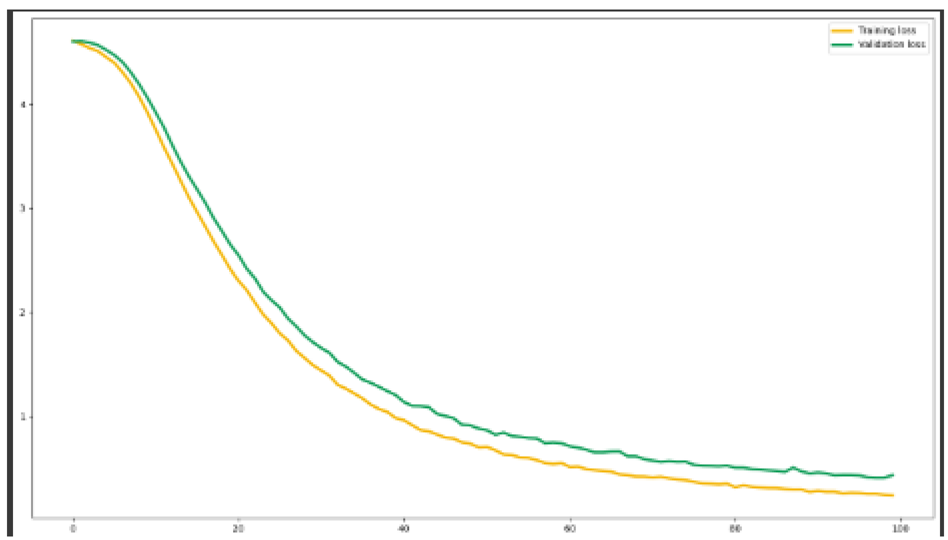
Trail_5: - SGD optimizer with 32 batch size 128 hidden nodes and 0.4 drop out ratio.

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.9, momentum=0.0, nesterov=False, name="SGD")

model_15,history_15 = fun_model(sgd ,32, 128 , 0.4 )

model_15.summary()
model_15.evaluate(X_val, y_val)
```

Accuracy Curve



Loss Curve



[loss: - 0.42986518144607544, accuracy: - 0.8787878751754761]

Model	Loss	Accuracy
Model_1	0.192666929404831	0.959595782600403
Model_2	0.1399792134761810З	0.969696998596191 4
Model_3	0.185216158628463 75	0.954545438289642 3
Model_4	0.304719209671020 5	0.954545438289642 3
Model_5	0.39152261614799 5	0.914141416549682 6
Model_6	0.134673565626144 4	0.979797959327697 8
Model_7	0.182123318314552 3	0.944444417953491 2
Model_8	0.139855636405944 8	0.969696998596191 4
Model_9	0.121030807495117 19	0.974747478961944 6
Model_10	0.0999465882778167 7	0.974747478961944 6
Model_11	0.359497308731079 1	0.909090936183929 4
Model_12	0.247159823775291 44	0.944444417953491 2
Model_13	0.667225241661071 8	0.858585834503173 8
Model_14	0.371849298477172 85	0.898989915847778 3
Model_15	0.429865181446075 44	0.878787875175476 1

Model_10 is the best model

Best hyperparameter

- Optimizer RMSProp
- batch size 32
- drop out 0.4
- hidden nodes 128

reference: -

https://www.tensorflow.org/agents/api_docs/python/tf_agents/train/Learner