ML project neural network - final

March 4, 2024

Neural network model - ML project

Neural network structure

```
  Input Layer           Hidden Layer 1        Hidden Layer 2         Output Layer
     (X)                    (A[1])                (A[2])                (A[3])
      |                       |                     |                     |
      v                       v                     v                     v
      |                       |                     |                     |
      +----> [W1, b1] -----> ReLU ----> [W2, b2] -----> ReLU ----> [W_out, b_out] -----> Sigmo
                                           ↑
                                           |
                                        ReLU
                                           ↑
                                        ReLU
```

Data Source: "https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset/data" Objective: utilize neural network models to predict default events

General sections 1. run a single model and check its metrics like accuracy, etc 2. fine-tune the model by changing hyerparameters like batch number, etc, to check variation of models regarding metrics 3. fine-tune the model by changing stracture, by changing neuron numbers in each layer

To compare the differences between the models, analyze various aspects such as their architecture, performance metrics (e.g., accuracy, loss), convergence behavior, and any changes made during fine-tuning. Below, I'll outline steps to compare these aspects:

Performance Metrics Comparison: Compare the performance metrics (e.g., accuracy, loss) of the models on the test dataset.

Type 0 trial:

Multiple training sessions and results comparison.

Type 1 trial and finetuning:

Architecture Comparison: Check if there were any changes in the architecture of the models during fine-tuning.

Type 2 trial and finetuning:

Hyperparameter Changes: adjust any hyperparameters during fine-tuning, compare these changes and their impact on model performance.

Convergence Behavior: Plot the training histories of the models.

Section 1 Single model

Step 1: Data Preprocessing(better treating categorical variables)

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import tkinter as tk
from tkinter import filedialog

# No full GUI, keep the root window from appearing
root = tk.Tk()
root.withdraw()

# Open file dialog to choose the dataset file
file_path = filedialog.askopenfilename()
if not file_path:
    print("No file selected. Exiting...")
    exit()

# Data Loading
print("Loading data from file...")
df = pd.read_csv(file_path)

# Separate features and target variable
X = df.drop(columns=['ID', 'default.payment.next.month']).values
y = df['default.payment.next.month'].values

# Categorical columns
categorical_cols = ['SEX', 'EDUCATION', 'MARRIAGE', 'PAY_0', 'PAY_2', 'PAY_3',␣
↪'PAY_4', 'PAY_5', 'PAY_6']

# One-hot encoding for categorical variables
encoder = OneHotEncoder(sparse=False)
X_cat = encoder.fit_transform(df[categorical_cols])

# Combine one-hot encoded features with numerical features
X = pd.concat([pd.DataFrame(X_cat), df.drop(columns=categorical_cols+['ID',␣
↪'default.payment.next.month'])], axis=1).values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
Loading data from file…
```

```
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\preprocessing\_encoders.py:868: FutureWarning: `sparse` was
renamed to `sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

Step 2: Model Architecture

[ ]:
```
#pip install tensorflow
```

[ ]:
```
import tensorflow as tf

# Define the neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train.
 ↪shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
WARNING:tensorflow:From C:\Users\wjbea\AppData\Roaming\Python\Python311\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

WARNING:tensorflow:From C:\Users\wjbea\AppData\Roaming\Python\Python311\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.
```

Step 3: Training

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

# Train the model
original_history = model.fit(X_train, y_train, epochs=20, batch_size=32,␣
 ↪validation_split=0.2)

# Train the original model verbose?
# original_history = model.fit(X_train, y_train, epochs=20, batch_size=32,␣
 ↪validation_split=0.2, verbose=0)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
```

```python
print("Original Model Test Accuracy:", accuracy)
```

```
WARNING:tensorflow:From C:\Users\wjbea\AppData\Roaming\Python\Python311\site-
packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/20
WARNING:tensorflow:From C:\Users\wjbea\AppData\Roaming\Python\Python311\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\wjbea\AppData\Roaming\Python\Python311\site-
packages\keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

600/600 [==============================] - 2s 2ms/step - loss: 0.4559 -
accuracy: 0.8127 - val_loss: 0.4424 - val_accuracy: 0.8190
Epoch 2/20
600/600 [==============================] - 1s 1ms/step - loss: 0.4303 -
accuracy: 0.8239 - val_loss: 0.4486 - val_accuracy: 0.8138
Epoch 3/20
600/600 [==============================] - 1s 1ms/step - loss: 0.4237 -
accuracy: 0.8238 - val_loss: 0.4428 - val_accuracy: 0.8146
Epoch 4/20
600/600 [==============================] - 1s 1ms/step - loss: 0.4206 -
accuracy: 0.8256 - val_loss: 0.4497 - val_accuracy: 0.8150
Epoch 5/20
600/600 [==============================] - 1s 1ms/step - loss: 0.4176 -
accuracy: 0.8260 - val_loss: 0.4531 - val_accuracy: 0.8156
Epoch 6/20
600/600 [==============================] - 1s 1ms/step - loss: 0.4142 -
accuracy: 0.8281 - val_loss: 0.4471 - val_accuracy: 0.8115
Epoch 7/20
600/600 [==============================] - 1s 2ms/step - loss: 0.4103 -
accuracy: 0.8304 - val_loss: 0.4492 - val_accuracy: 0.8169
Epoch 8/20
600/600 [==============================] - 1s 1ms/step - loss: 0.4087 -
accuracy: 0.8303 - val_loss: 0.4559 - val_accuracy: 0.8158
Epoch 9/20
600/600 [==============================] - 1s 1ms/step - loss: 0.4053 -
accuracy: 0.8322 - val_loss: 0.4640 - val_accuracy: 0.8135
Epoch 10/20
600/600 [==============================] - 1s 1ms/step - loss: 0.4023 -
accuracy: 0.8329 - val_loss: 0.4574 - val_accuracy: 0.8133
Epoch 11/20
600/600 [==============================] - 1s 2ms/step - loss: 0.3998 -
accuracy: 0.8331 - val_loss: 0.4561 - val_accuracy: 0.8106
```

```
Epoch 12/20
600/600 [==============================] - 1s 1ms/step - loss: 0.3968 -
accuracy: 0.8349 - val_loss: 0.4645 - val_accuracy: 0.8160
Epoch 13/20
600/600 [==============================] - 1s 2ms/step - loss: 0.3936 -
accuracy: 0.8358 - val_loss: 0.4627 - val_accuracy: 0.8085
Epoch 14/20
600/600 [==============================] - 1s 2ms/step - loss: 0.3914 -
accuracy: 0.8374 - val_loss: 0.4721 - val_accuracy: 0.8069
Epoch 15/20
600/600 [==============================] - 1s 1ms/step - loss: 0.3874 -
accuracy: 0.8394 - val_loss: 0.4686 - val_accuracy: 0.8050
Epoch 16/20
600/600 [==============================] - 1s 1ms/step - loss: 0.3846 -
accuracy: 0.8396 - val_loss: 0.4691 - val_accuracy: 0.8010
Epoch 17/20
600/600 [==============================] - 1s 2ms/step - loss: 0.3800 -
accuracy: 0.8432 - val_loss: 0.4822 - val_accuracy: 0.8037
Epoch 18/20
600/600 [==============================] - 1s 1ms/step - loss: 0.3770 -
accuracy: 0.8442 - val_loss: 0.4839 - val_accuracy: 0.7996
Epoch 19/20
600/600 [==============================] - 1s 2ms/step - loss: 0.3768 -
accuracy: 0.8443 - val_loss: 0.4778 - val_accuracy: 0.8048
Epoch 20/20
600/600 [==============================] - 1s 1ms/step - loss: 0.3690 -
accuracy: 0.8464 - val_loss: 0.4947 - val_accuracy: 0.7931
188/188 [==============================] - 0s 1ms/step - loss: 0.4836 -
accuracy: 0.8033
Original Model Test Accuracy: 0.8033333420753479
```

[ ]:
```python
import numpy as np

# Print the summary of the model

model.summary()

# Extracting metrics from the original_history object
training_accuracy = original_history.history['accuracy']
validation_accuracy = original_history.history['val_accuracy']
training_loss = original_history.history['loss']
validation_loss = original_history.history['val_loss']

# Calculating final metrics and variance
final_training_accuracy = training_accuracy[-1]
final_validation_accuracy = validation_accuracy[-1]
final_training_loss = training_loss[-1]
```

```
final_validation_loss = validation_loss[-1]
variance_accuracy = np.var(validation_accuracy)
variance_loss = np.var(validation_loss)

# Printing the metrics
print("\nModel Performance Metrics:")
print(f"Final Training Accuracy: {final_training_accuracy*100:.2f}%")
print(f"Final Validation Accuracy: {final_validation_accuracy*100:.2f}%")
print(f"Training Accuracy Variance: {np.var(training_accuracy):.4f}")
print(f"Validation Accuracy Variance: {variance_accuracy:.4f}")
print(f"Final Training Loss: {final_training_loss:.4f}")
print(f"Final Validation Loss: {final_validation_loss:.4f}")
print(f"Training Loss Variance: {np.var(training_loss):.4f}")
print(f"Validation Loss Variance: {variance_loss:.4f}")
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               11776

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 20097 (78.50 KB)
Trainable params: 20097 (78.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

Model Performance Metrics:
Final Training Accuracy: 84.64%
Final Validation Accuracy: 79.31%
Training Accuracy Variance: 0.0001
Validation Accuracy Variance: 0.0000
Final Training Loss: 0.3690
Final Validation Loss: 0.4947
Training Loss Variance: 0.0004
Validation Loss Variance: 0.0002
```

Step 4: Evaluation by adding additional training sessions

Compare additional three training sessions

[ ]:
```
# Function to create and compile model
def create_model():
    model = tf.keras.Sequential([
```

```python
        tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train.
 ↪shape[1],)),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
    return model

models = []
histories = []
val_accuracies = []
test_metrics = []

for i in range(3):  # Example: fine-tuning three times
    model = create_model()
    history = model.fit(X_train, y_train, epochs=20, batch_size=32,␣
 ↪validation_split=0.2, verbose=0)
    loss, accuracy = model.evaluate(X_test, y_test, verbose=0)  # Set verbose␣
 ↪to 0 to reduce log noise
    print(f"Model {i+1} - Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
    histories.append(history)
    models.append(model)
    test_metrics.append((loss, accuracy))

    # Get the last epoch metrics from the training history
    final_training_accuracy = history.history['accuracy'][-1]
    final_training_loss = history.history['loss'][-1]
    final_val_accuracy = history.history['val_accuracy'][-1]
    final_val_loss = history.history['val_loss'][-1]
    val_accuracies.append(final_val_accuracy)

    # Printing the detailed metrics
    print(f"Model {i+1} Training Metrics - Final Training Accuracy:␣
 ↪{final_training_accuracy:.4f}, Final Training Loss: {final_training_loss:.
 ↪4f}")
    print(f"Model {i+1} Validation Metrics - Final Validation Accuracy:␣
 ↪{final_val_accuracy:.4f}, Final Validation Loss: {final_val_loss:.4f}")

    model.summary()

# Identifying the best model based on validation accuracy
best_model_index = val_accuracies.index(max(val_accuracies))
best_model = models[best_model_index]
best_test_loss, best_test_accuracy = test_metrics[best_model_index]
```

```
print(f"\nBest Model is Model {best_model_index + 1} with Test Loss:␣
↪{best_test_loss:.4f} and Test Accuracy: {best_test_accuracy:.4f}")
```

```
Model 1 - Test Loss: 0.4874, Test Accuracy: 0.8038
Model 1 Training Metrics - Final Training Accuracy: 0.8461, Final Training Loss:
0.3688
Model 1 Validation Metrics - Final Validation Accuracy: 0.8071, Final Validation
Loss: 0.4817
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 128)               11776

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 20097 (78.50 KB)
Trainable params: 20097 (78.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model 2 - Test Loss: 0.4910, Test Accuracy: 0.8093
Model 2 Training Metrics - Final Training Accuracy: 0.8467, Final Training Loss:
0.3696
Model 2 Validation Metrics - Final Validation Accuracy: 0.8012, Final Validation
Loss: 0.4936
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 128)               11776

 dense_7 (Dense)             (None, 64)                8256

 dense_8 (Dense)             (None, 1)                 65

=================================================================
Total params: 20097 (78.50 KB)
Trainable params: 20097 (78.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model 3 - Test Loss: 0.4809, Test Accuracy: 0.8118
Model 3 Training Metrics - Final Training Accuracy: 0.8476, Final Training Loss:
0.3652
Model 3 Validation Metrics - Final Validation Accuracy: 0.8065, Final Validation
Loss: 0.4934
```

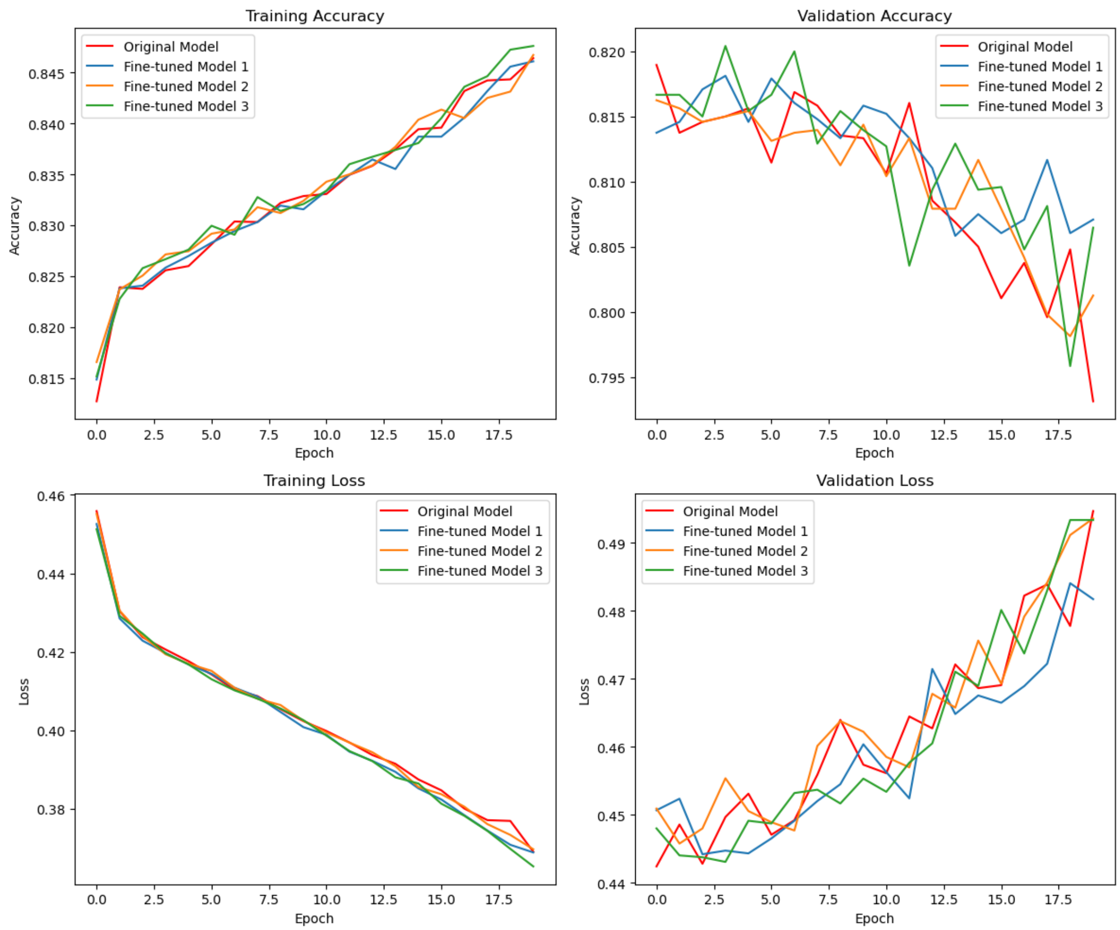
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 128)               11776

 dense_10 (Dense)            (None, 64)                8256

 dense_11 (Dense)            (None, 1)                 65

=================================================================
Total params: 20097 (78.50 KB)
Trainable params: 20097 (78.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
Best Model is Model 1 with Test Loss: 0.4874 and Test Accuracy: 0.8038
```

Step 5: Visual Comparison

[ ]:
```
import matplotlib.pyplot as plt

# Set figure size
plt.figure(figsize=(12, 10))

# Compare training accuracy
plt.subplot(2, 2, 1)
plt.plot(original_history.history['accuracy'], label='Original Model',␣
 ↪color='red')
for i, history in enumerate(histories):
    plt.plot(history.history['accuracy'], label=f'Fine-tuned Model {i+1}')
plt.title('Training Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Compare validation accuracy
plt.subplot(2, 2, 2)
plt.plot(original_history.history['val_accuracy'], label='Original Model',␣
 ↪color='red')
for i, history in enumerate(histories):
    plt.plot(history.history['val_accuracy'], label=f'Fine-tuned Model {i+1}')
plt.title('Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Compare training loss
```

```
plt.subplot(2, 2, 3)
plt.plot(original_history.history['loss'], label='Original Model', color='red')
for i, history in enumerate(histories):
    plt.plot(history.history['loss'], label=f'Fine-tuned Model {i+1}')
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Compare validation loss
plt.subplot(2, 2, 4)
plt.plot(original_history.history['val_loss'], label='Original Model',␣
 ↪color='red')
for i, history in enumerate(histories):
    plt.plot(history.history['val_loss'], label=f'Fine-tuned Model {i+1}')
plt.title('Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

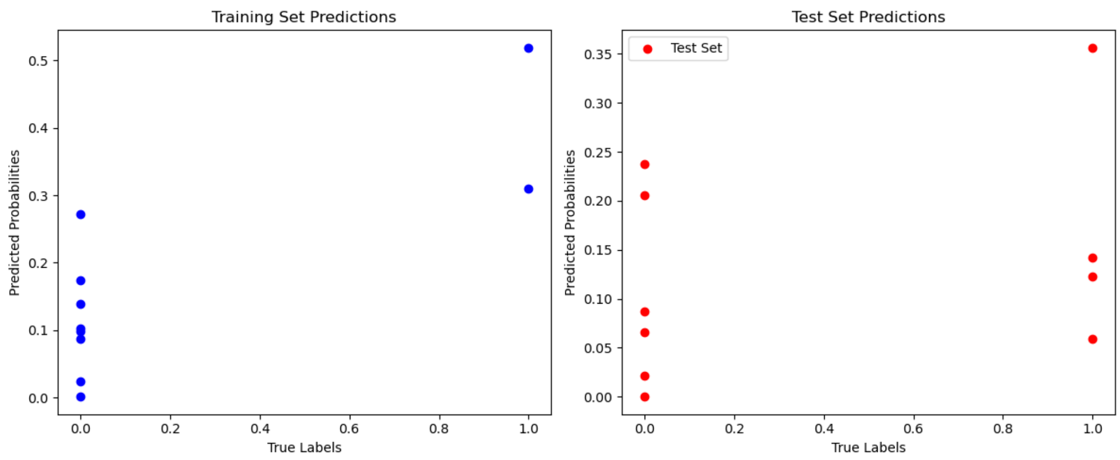


Step 6: Sampling 20 cases to check the model

```
[ ]: import random
import matplotlib.pyplot as plt

# Randomly select 10 cases from the training set
random.seed(42)  # for reproducibility
train_indices = random.sample(range(len(X_train)), k=10)
test_indices = random.sample(range(len(X_test)), k=10)

# Function to decode model output
def decode_output(output):
    return 1 if output >= 0.5 else 0

# Initialize lists to store predictions
train_true_labels = []
train_predicted_labels = []
train_predicted_probs = []
```

```
test_true_labels = []
test_predicted_labels = []
test_predicted_probs = []

# Make predictions on the training set
for idx in train_indices:
    input_sample = X_train[idx]
    true_label = y_train[idx]
    predicted_prob = model.predict(input_sample.reshape(1, -1))[0, 0]
    predicted_label = decode_output(predicted_prob)

    train_true_labels.append(true_label)
    train_predicted_labels.append(predicted_label)
    train_predicted_probs.append(predicted_prob)

# Make predictions on the test set
for idx in test_indices:
    input_sample = X_test[idx]
    true_label = y_test[idx]
    predicted_prob = model.predict(input_sample.reshape(1, -1))[0, 0]
    predicted_label = decode_output(predicted_prob)

    test_true_labels.append(true_label)
    test_predicted_labels.append(predicted_label)
    test_predicted_probs.append(predicted_prob)

# Plotting
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Plot training predictions
axes[0].scatter(train_true_labels, train_predicted_probs, color='blue',␣
 ↪label='Training Set')
axes[0].set_title('Training Set Predictions')
axes[0].set_xlabel('True Labels')
axes[0].set_ylabel('Predicted Probabilities')

# Plot test predictions
axes[1].scatter(test_true_labels, test_predicted_probs, color='red',␣
 ↪label='Test Set')
axes[1].set_title('Test Set Predictions')
axes[1].set_xlabel('True Labels')
axes[1].set_ylabel('Predicted Probabilities')

plt.tight_layout()
plt.legend()
plt.show()
```

```
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
```



step 7: output model layers

```
[ ]: for layer in model.layers:
         print(layer.get_config())  # Print layer's configuration
         print(layer.get_weights())  # Print layer's weights
```

```
{'name': 'dense_9', 'trainable': True, 'dtype': 'float32', 'batch_input_shape':
(None, 91), 'units': 128, 'activation': 'relu', 'use_bias': True,
'kernel_initializer': {'module': 'keras.initializers', 'class_name':
'GlorotUniform', 'config': {'seed': None}, 'registered_name': None},
'bias_initializer': {'module': 'keras.initializers', 'class_name': 'Zeros',
```

```
'config': {}, 'registered_name': None}, 'kernel_regularizer': None,
'bias_regularizer': None, 'activity_regularizer': None, 'kernel_constraint':
None, 'bias_constraint': None}
[array([[ 0.13239932, -0.03825978, -0.10319452, …, -0.16205542,
        -0.13940139,  0.2677684 ],
       [ 0.00473295, -0.12608987, -0.18208343, …,  0.1472235 ,
         0.13748474, -0.2805138 ],
       [ 0.29030785, -0.08532255,  0.22759183, …,  0.10514984,
        -0.03552388,  0.27803218],
       …,
       [-0.08121741, -0.09746833, -0.19118749, …,  0.05381406,
        -0.30578998, -0.04809258],
       [ 0.13815905, -0.01368143,  0.18784279, …,  0.23073445,
         0.00439576, -0.14108647],
       [-0.30082905, -0.03564675,  0.08768641, …, -0.03130113,
        -0.06376801, -0.08907747]], dtype=float32), array([-0.06314587,
-0.02974103, -0.0700342 , -0.11583614, -0.17640002,
       -0.2322554 , -0.16806479, -0.09214032, -0.15590343, -0.29983562,
       -0.17640348,  0.02554844, -0.36677706, -0.21298784, -0.126944  ,
       -0.22415736, -0.27387822, -0.24064156, -0.1783337 , -0.08097191,
       -0.30222082, -0.11500244, -0.03020082,  0.03047959, -0.11861157,
        0.02099592,  0.00854484, -0.21871714, -0.21738867,  0.00888074,
       -0.06536833, -0.26920047, -0.16733673, -0.36044002, -0.18762176,
       -0.19258292, -0.24224396, -0.12067862, -0.03029143, -0.23478661,
       -0.29432067, -0.25759685, -0.11637391, -0.11599952, -0.17844893,
       -0.0723611 , -0.20233198,  0.01161837, -0.0689031 , -0.18270268,
       -0.12455187, -0.18856804, -0.2686821 , -0.15163168, -0.11408707,
       -0.13761647, -0.21396627, -0.17154607, -0.2436982 , -0.24545157,
       -0.24467805, -0.12399559, -0.17881367, -0.08267391,  0.01478142,
       -0.09625816,  0.02176718, -0.16933236, -0.25569078,  0.03455385,
       -0.12796807, -0.2685009 , -0.08787989, -0.01828   , -0.27891597,
       -0.22707304, -0.14136721, -0.1512532 , -0.24737933, -0.2013958 ,
       -0.13348156, -0.0013908 , -0.31744325, -0.13146073, -0.2641084 ,
       -0.04728441, -0.17210115, -0.24750565, -0.2916154 , -0.14546978,
       -0.0177034 , -0.04906127, -0.08304112, -0.04939458, -0.24545719,
       -0.02917488, -0.16505745, -0.22719473, -0.03564568, -0.00334169,
       -0.06662806, -0.09785035, -0.2731515 , -0.23868786,  0.13806614,
       -0.1946871 , -0.31707096, -0.22114572, -0.16728005, -0.08742393,
       -0.08580276, -0.11595045, -0.02900333, -0.18309851, -0.15776159,
       -0.00449369, -0.12166932,  0.07699926, -0.02268643, -0.02592161,
       -0.1688097 , -0.1507025 , -0.1880744 , -0.11428418, -0.18733458,
       -0.02719883, -0.01845168, -0.12324347], dtype=float32)]
{'name': 'dense_10', 'trainable': True, 'dtype': 'float32', 'units': 64,
'activation': 'relu', 'use_bias': True, 'kernel_initializer': {'module':
'keras.initializers', 'class_name': 'GlorotUniform', 'config': {'seed': None},
'registered_name': None}, 'bias_initializer': {'module': 'keras.initializers',
'class_name': 'Zeros', 'config': {}, 'registered_name': None},
'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer':
```

```
None, 'kernel_constraint': None, 'bias_constraint': None}
[array([[-0.22011697, -0.1402252 ,  0.1395788 , …,  0.1388353 ,
        -0.24116401,  0.24727479],
       [-0.30878422, -0.12471921, -0.16031046, …,  0.00356248,
         0.15338708, -0.08754006],
       [-0.14594913, -0.12022878, -0.3207866 , …,  0.02234133,
        -0.10106713,  0.01767357],
       …,
       [-0.20023239, -0.19484839, -0.01543508, …,  0.04308372,
        -0.1426682 ,  0.05813215],
       [-0.14201584, -0.03673085,  0.17813236, …,  0.109411  ,
        -0.06412992,  0.10969793],
       [-0.24960716,  0.01865408,  0.01901884, …,  0.2142758 ,
        -0.05285589,  0.09265448]], dtype=float32), array([ 0.0238386 ,
-0.05295807, -0.01194536, -0.07486983, -0.0208216 ,
       -0.01433623, -0.03451939,  0.00328265, -0.04093596,  0.07009771,
        0.13951778, -0.06355715, -0.06291004, -0.05227482,  0.00918209,
        0.16383398,  0.06719777, -0.00381986, -0.04094479, -0.10450248,
        0.06949019, -0.03519735, -0.03222349, -0.03947103,  0.01237652,
       -0.0536177 ,  0.03858345,  0.11107745, -0.09571528,  0.13125886,
        0.01128598, -0.02818724, -0.0422563 ,  0.03987476, -0.06938097,
        0.10354959,  0.09867904, -0.0526472 ,  0.0467023 , -0.10349622,
        0.05768659,  0.07218007,  0.11918143, -0.02044393,  0.07951221,
       -0.04103069,  0.03245994,  0.00359879, -0.10939439, -0.02530224,
        0.0070388 ,  0.13065603, -0.02626172, -0.02957447,  0.16042773,
        0.02988331, -0.07671733,  0.08386246, -0.03480882,  0.06925093,
       -0.05640384,  0.1654793 , -0.02632102,  0.10154916], dtype=float32)]
{'name': 'dense_11', 'trainable': True, 'dtype': 'float32', 'units': 1,
'activation': 'sigmoid', 'use_bias': True, 'kernel_initializer': {'module':
'keras.initializers', 'class_name': 'GlorotUniform', 'config': {'seed': None},
'registered_name': None}, 'bias_initializer': {'module': 'keras.initializers',
'class_name': 'Zeros', 'config': {}, 'registered_name': None},
'kernel_regularizer': None, 'bias_regularizer': None, 'activity_regularizer':
None, 'kernel_constraint': None, 'bias_constraint': None}
[array([[ 0.4338064 ],
       [ 0.00093213],
       [ 0.48423794],
       [ 0.43717209],
       [ 0.70104045],
       [ 0.55573255],
       [ 0.34861368],
       [ 0.43271264],
       [ 0.22647694],
       [-0.18504947],
       [-0.20204873],
       [ 0.39965478],
       [ 0.2908424 ],
       [ 0.25911856],
```

```
[ 0.32803735],
[-0.3193789 ],
[-0.13965467],
[ 0.25816926],
[ 0.49188796],
[ 0.04936615],
[-0.09829216],
[ 0.21363284],
[ 0.3800024 ],
[ 0.28971595],
[-0.13094781],
[ 0.43317962],
[-0.45168683],
[-0.11722303],
[ 0.39437443],
[-0.22337526],
[-0.3687599 ],
[ 0.47107086],
[ 0.15801112],
[-0.23352239],
[ 0.10032287],
[-0.32461742],
[-0.05347043],
[ 0.15408881],
[ 0.00164975],
[ 0.08574565],
[-0.2731378 ],
[-0.33189943],
[-0.35994083],
[-0.10292064],
[-0.09082615],
[ 0.21505448],
[-0.14595073],
[-0.08609369],
[ 0.35251582],
[ 0.5171532 ],
[-0.07837456],
[-0.29163086],
[ 0.11652065],
[ 0.5479267 ],
[-0.18389061],
[-0.15051468],
[ 0.39740297],
[-0.33310184],
[ 0.42630106],
[-0.02723908],
[ 0.40839666],
[-0.25522384],
```

```
[-0.3117061 ],
[-0.2334225 ]], dtype=float32), array([-0.12800233], dtype=float32)]
```

Section 2: Hyperparameter fintuning

Step 1: variations setting now creat more models to fine tune : epochs +- 10 holding others constant. Batch +-10 holding others constant. Validation split+- 0,1 holding others constant. Changing hyper parameters to check different models.

[ ]:
```python
import numpy as np

# Function to create, train, and evaluate the model with variations
def train_model_with_variation(epochs_variation=0, batch_size_variation=0,␣
↪validation_split_variation=0.0):
    model_variation = create_model()  # Assuming create_model() creates the␣
↪same model architecture
    epochs = 20 + epochs_variation
    batch_size = 32 + batch_size_variation
    validation_split = 0.2 + validation_split_variation
    history = model_variation.fit(X_train, y_train, epochs=epochs,␣
↪batch_size=batch_size, validation_split=validation_split, verbose=0)
    loss, accuracy = model_variation.evaluate(X_test, y_test, verbose=0)  # Set␣
↪verbose to 0 for cleaner output
    # Extract the final training and validation metrics
    final_training_accuracy = history.history['accuracy'][-1]
    final_validation_accuracy = history.history['val_accuracy'][-1]
    final_training_loss = history.history['loss'][-1]
    final_validation_loss = history.history['val_loss'][-1]
    return history, loss, accuracy, final_training_loss, final_validation_loss,␣
↪final_training_accuracy, final_validation_accuracy

# Create and train models with variations
variations = [
    (1, 0, 0.0),  # Increase epochs by 1
    (-1, 0, 0.0),  # Decrease epochs by 1
    (0, 10, 0.0),  # Increase batch size by 10
    (0, -10, 0.0),  # Decrease batch size by 10
    (0, 0, 0.1),  # Increase validation split by 0.1
    (0, 0, -0.1),  # Decrease validation split by 0.1
    (1, 10, 0.0),  # Increase epochs by 1 and batch size by 10
    (-1, -10, 0.0),  # Decrease epochs by 1 and batch size by 10
    (1, 0, -0.1),  # Increase epochs by 1 and decrease validation split by 0.1
    (-1, 0, 0.1)   # Decrease epochs by 1 and increase validation split by 0.1
]

histories = []
losses = []
accuracies = []
```

```
training_losses = []
validation_losses = []
training_accuracies = []
validation_accuracies = []

for variation in variations:
    history, test_loss, test_accuracy, final_training_loss,␣
↪final_validation_loss, final_training_accuracy, final_validation_accuracy =␣
↪train_model_with_variation(*variation)
    histories.append(history)
    losses.append(test_loss)
    accuracies.append(test_accuracy)
    training_losses.append(final_training_loss)
    validation_losses.append(final_validation_loss)
    training_accuracies.append(final_training_accuracy)
    validation_accuracies.append(final_validation_accuracy)

# Print metrics for each variation
for i, variation in enumerate(variations):
    print(f"Variation {i+1}: Training Accuracy = {training_accuracies[i]*100:.
↪2f}%, Validation Accuracy = {validation_accuracies[i]*100:.2f}%, Training␣
↪Loss = {training_losses[i]:.4f}, Validation Loss = {validation_losses[i]:.
↪4f}, Test Accuracy = {accuracies[i]*100:.2f}%")
```

```
Variation 1: Training Accuracy = 84.94%, Validation Accuracy = 80.10%, Training
Loss = 0.3615, Validation Loss = 0.4924, Test Accuracy = 81.00%
Variation 2: Training Accuracy = 84.47%, Validation Accuracy = 80.17%, Training
Loss = 0.3701, Validation Loss = 0.4917, Test Accuracy = 81.20%
Variation 3: Training Accuracy = 84.44%, Validation Accuracy = 80.23%, Training
Loss = 0.3677, Validation Loss = 0.4864, Test Accuracy = 80.40%
Variation 4: Training Accuracy = 84.90%, Validation Accuracy = 80.19%, Training
Loss = 0.3664, Validation Loss = 0.4837, Test Accuracy = 80.83%
Variation 5: Training Accuracy = 85.09%, Validation Accuracy = 80.83%, Training
Loss = 0.3642, Validation Loss = 0.4861, Test Accuracy = 81.15%
Variation 6: Training Accuracy = 84.32%, Validation Accuracy = 79.96%, Training
Loss = 0.3731, Validation Loss = 0.5004, Test Accuracy = 80.68%
Variation 7: Training Accuracy = 84.79%, Validation Accuracy = 79.56%, Training
Loss = 0.3663, Validation Loss = 0.4819, Test Accuracy = 80.60%
Variation 8: Training Accuracy = 84.82%, Validation Accuracy = 80.08%, Training
Loss = 0.3694, Validation Loss = 0.4849, Test Accuracy = 81.27%
Variation 9: Training Accuracy = 84.69%, Validation Accuracy = 79.79%, Training
Loss = 0.3671, Validation Loss = 0.4998, Test Accuracy = 81.10%
Variation 10: Training Accuracy = 84.73%, Validation Accuracy = 80.57%, Training
Loss = 0.3691, Validation Loss = 0.4844, Test Accuracy = 81.15%
```

Print best model

[ ]:
```
# Initialize variables for the best model
best_model_index = np.argmax(accuracies)  # Index of the model with the highest␣
↪test accuracy
best_test_loss = losses[best_model_index]
best_test_accuracy = accuracies[best_model_index]
best_training_loss = training_losses[best_model_index]
best_validation_loss = validation_losses[best_model_index]
best_variation = variations[best_model_index]

# Print the best model's details
print("\nBest Model Details:")
print(f"Variation {best_model_index + 1}:")
print(f"Test Loss = {best_test_loss}, Test Accuracy = {best_test_accuracy}")
print(f"Training Loss = {best_training_loss}, Validation Loss =␣
↪{best_validation_loss}")
print(f"Variation Parameters: Epochs Variation = {best_variation[0]}, Batch␣
↪Size Variation = {best_variation[1]}, Validation Split Variation =␣
↪{best_variation[2]}")

# Print the summary of the best model
print("\nBest Model Summary:")
if 0 <= best_model_index < len(models):  # Ensure the index is valid
    best_model = models[best_model_index]  # Retrieve the best model based on␣
↪the index
    best_model.summary()
else:
    print("Invalid model index:", best_model_index)
```

```
Best Model Details:
Variation 8:
Test Loss = 0.4763542115688324, Test Accuracy = 0.812666654586792
Training Loss = 0.36942601203918457, Validation Loss = 0.48487588763237
Variation Parameters: Epochs Variation = -1, Batch Size Variation = -10,
Validation Split Variation = 0.0

Best Model Summary:
Invalid model index: 7
```

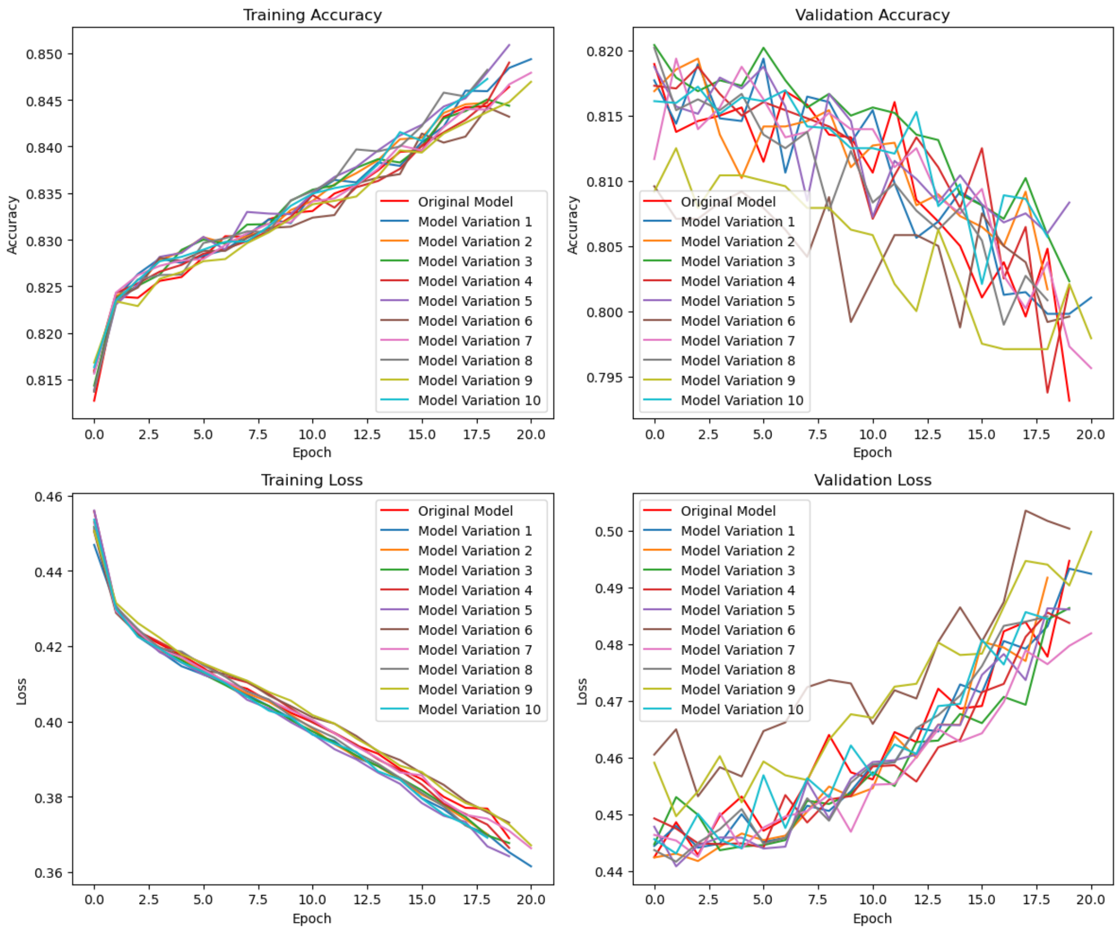
Step 2: plot function preparation

[ ]:
```
import matplotlib.pyplot as plt

# Function to plot comparison graphs
def plot_comparison_graphs(original_history, histories):
    # Set figure size
    plt.figure(figsize=(12, 10))
    # Compare training accuracy
```

```python
    plt.subplot(2, 2, 1)
    plt.plot(original_history.history['accuracy'], label='Original Model', color='red')
    for i, history in enumerate(histories):
        plt.plot(history.history['accuracy'], label=f'Model Variation {i+1}')
    plt.title('Training Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    # Compare validation accuracy
    plt.subplot(2, 2, 2)
    plt.plot(original_history.history['val_accuracy'], label='Original Model', color='red')
    for i, history in enumerate(histories):
        plt.plot(history.history['val_accuracy'], label=f'Model Variation {i+1}')
    plt.title('Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    # Compare training loss
    plt.subplot(2, 2, 3)
    plt.plot(original_history.history['loss'], label='Original Model', color='red')
    for i, history in enumerate(histories):
        plt.plot(history.history['loss'], label=f'Model Variation {i+1}')
    plt.title('Training Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    # Compare validation loss
    plt.subplot(2, 2, 4)
    plt.plot(original_history.history['val_loss'], label='Original Model', color='red')
    for i, history in enumerate(histories):
        plt.plot(history.history['val_loss'], label=f'Model Variation {i+1}')
    plt.title('Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

Step 3: print summaries and graphs

```
[ ]: # Plot comparison graphs
plot_comparison_graphs(original_history, histories)
```



Section 3: model structure/additional hyperparameter fine tuning

step 1: changing numbers of neuron in each layer assume base case as 128 64 1, make 6 variations on 128 and 6 variations on 64, holding ohters constant in each try

```
[ ]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt

# Define the input shape
input_shape = X_train.shape[1]

# Function to create model with variations in neuron numbers for each layer
def create_model(first_layer_neurons, second_layer_neurons):
```

```
    model = Sequential()
    model.add(Dense(first_layer_neurons, activation='relu',␣
↪input_shape=(input_shape,)))
    model.add(Dense(second_layer_neurons, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    return model

# Define variations in neuron numbers for each layer
first_layer_neurons_variations = [64, 96, 128, 160, 192, 224]
second_layer_neurons_variations = [32, 48, 64, 80, 96, 112]

# Initialize a list to store the performance of each model
model_performances = []

# Initialize lists to store models and histories
models = []
histories = []

# Create and train models with variations
for first_layer_neurons in first_layer_neurons_variations:
    for second_layer_neurons in second_layer_neurons_variations:
        model = create_model(first_layer_neurons, second_layer_neurons)
        model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])
        history = model.fit(X_train, y_train, epochs=20, batch_size=32,␣
↪validation_split=0.2, verbose=0)
        test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
        # Extract training and validation metrics
        final_training_accuracy = history.history['accuracy'][-1]
        final_validation_accuracy = history.history['val_accuracy'][-1]
        final_training_loss = history.history['loss'][-1]
        final_validation_loss = history.history['val_loss'][-1]
        print(f"Model with {first_layer_neurons} neurons in the first layer and␣
↪{second_layer_neurons} neurons in the second layer - Test Accuracy:␣
↪{test_accuracy}, Training Accuracy: {final_training_accuracy}, Validation␣
↪Accuracy: {final_validation_accuracy}, Training Loss: {final_training_loss},␣
↪Validation Loss: {final_validation_loss}")
        model_performances.append((test_accuracy, final_training_accuracy,␣
↪final_validation_accuracy, final_training_loss, final_validation_loss,␣
↪first_layer_neurons, second_layer_neurons))

        # Save model and history
        models.append(model)
        histories.append(history)

# Find the best performing model based on test accuracy
best_performance = max(model_performances, key=lambda x: x[0])
```

```
best_test_accuracy, best_training_accuracy, best_validation_accuracy,␣
↪best_training_loss, best_validation_loss, best_first_layer_neurons,␣
↪best_second_layer_neurons = best_performance

# Print the summary of the best model
print(f"\nThe best model has {best_first_layer_neurons} neurons in the first␣
↪layer and {best_second_layer_neurons} neurons in the second layer with Test␣
↪Accuracy: {best_test_accuracy}, Training Accuracy: {best_training_accuracy},␣
↪Validation Accuracy: {best_validation_accuracy}, Training Loss:␣
↪{best_training_loss}, Validation Loss: {best_validation_loss}.\n")
```

```
Model with 64 neurons in the first layer and 32 neurons in the second layer -
Test Accuracy: 0.812666654586792, Training Accuracy: 0.8387500047683716,
Validation Accuracy: 0.8079166412353516, Training Loss: 0.3872203528881073,
Validation Loss: 0.4688476026058197
Model with 64 neurons in the first layer and 48 neurons in the second layer -
Test Accuracy: 0.8146666884422302, Training Accuracy: 0.8394270539283752,
Validation Accuracy: 0.809583306312561, Training Loss: 0.3832058906555176,
Validation Loss: 0.4705822765827179
Model with 64 neurons in the first layer and 64 neurons in the second layer -
Test Accuracy: 0.8108333349227905, Training Accuracy: 0.8416146039962769,
Validation Accuracy: 0.8066666722297668, Training Loss: 0.3821476399898529,
Validation Loss: 0.47421035170555115
Model with 64 neurons in the first layer and 80 neurons in the second layer -
Test Accuracy: 0.8133333325386047, Training Accuracy: 0.840833306312561,
Validation Accuracy: 0.8052083253860474, Training Loss: 0.3782062828540802,
Validation Loss: 0.46927282214164734
Model with 64 neurons in the first layer and 96 neurons in the second layer -
Test Accuracy: 0.8088333606719971, Training Accuracy: 0.8419791460037231,
Validation Accuracy: 0.8043749928474426, Training Loss: 0.37621212005615234,
Validation Loss: 0.47140589356422424
Model with 64 neurons in the first layer and 112 neurons in the second layer -
Test Accuracy: 0.8105000257492065, Training Accuracy: 0.8450520634651184,
Validation Accuracy: 0.8081250190734863, Training Loss: 0.37170642614364624,
Validation Loss: 0.4826148450374603
Model with 96 neurons in the first layer and 32 neurons in the second layer -
Test Accuracy: 0.8133333325386047, Training Accuracy: 0.8417708277702332,
Validation Accuracy: 0.8075000047683716, Training Loss: 0.3775798976421356,
Validation Loss: 0.474557101726532
Model with 96 neurons in the first layer and 48 neurons in the second layer -
Test Accuracy: 0.8116666674613953, Training Accuracy: 0.843177080154419,
Validation Accuracy: 0.8037499785423279, Training Loss: 0.37569084763526917,
Validation Loss: 0.4807412326335907
Model with 96 neurons in the first layer and 64 neurons in the second layer -
Test Accuracy: 0.8133333325386047, Training Accuracy: 0.8438541889190674,
Validation Accuracy: 0.8031250238418579, Training Loss: 0.3739531338214874,
Validation Loss: 0.4831506907939911
Model with 96 neurons in the first layer and 80 neurons in the second layer -
```

Test Accuracy: 0.8119999766349792, Training Accuracy: 0.8467708230018616,
Validation Accuracy: 0.8058333396911621, Training Loss: 0.3704271614551544,
Validation Loss: 0.4868190586566925
Model with 96 neurons in the first layer and 96 neurons in the second layer -
Test Accuracy: 0.809499979019165, Training Accuracy: 0.844739556312561,
Validation Accuracy: 0.8025000095367432, Training Loss: 0.37028738856315613,
Validation Loss: 0.48446670174598694
Model with 96 neurons in the first layer and 112 neurons in the second layer -
Test Accuracy: 0.8116666674613953, Training Accuracy: 0.8483333587646484,
Validation Accuracy: 0.8004166483879089, Training Loss: 0.36756575107574463,
Validation Loss: 0.483987420797348
Model with 128 neurons in the first layer and 32 neurons in the second layer -
Test Accuracy: 0.8163333535194397, Training Accuracy: 0.8432812690734863,
Validation Accuracy: 0.8070833086967468, Training Loss: 0.3765762448310852,
Validation Loss: 0.4808409512042999
Model with 128 neurons in the first layer and 48 neurons in the second layer -
Test Accuracy: 0.8009999990463257, Training Accuracy: 0.84744793176651,
Validation Accuracy: 0.8010416626930237, Training Loss: 0.3700962960720062,
Validation Loss: 0.4862491488456726
Model with 128 neurons in the first layer and 64 neurons in the second layer -
Test Accuracy: 0.8136666417121887, Training Accuracy: 0.8454687595367432,
Validation Accuracy: 0.8031250238418579, Training Loss: 0.3675951659679413,
Validation Loss: 0.4863320291042328
Model with 128 neurons in the first layer and 80 neurons in the second layer -
Test Accuracy: 0.8036666512489319, Training Accuracy: 0.8463020920753479,
Validation Accuracy: 0.79666668176651, Training Loss: 0.36680057644844055,
Validation Loss: 0.4863210618495941
Model with 128 neurons in the first layer and 96 neurons in the second layer -
Test Accuracy: 0.8053333163261414, Training Accuracy: 0.8493229150772095,
Validation Accuracy: 0.7964583039283752, Training Loss: 0.3636026382446289,
Validation Loss: 0.4930282533168793
Model with 128 neurons in the first layer and 112 neurons in the second layer -
Test Accuracy: 0.8003333210945129, Training Accuracy: 0.8491666913032532,
Validation Accuracy: 0.7995833158493042, Training Loss: 0.3628787100315094,
Validation Loss: 0.4973815381526947
Model with 160 neurons in the first layer and 32 neurons in the second layer -
Test Accuracy: 0.812333345413208, Training Accuracy: 0.8455208539962769,
Validation Accuracy: 0.8010416626930237, Training Loss: 0.3746674954891205,
Validation Loss: 0.4798887073993683
Model with 160 neurons in the first layer and 48 neurons in the second layer -
Test Accuracy: 0.8063333630561829, Training Accuracy: 0.8486979007720947,
Validation Accuracy: 0.8006250262260437, Training Loss: 0.3659965693950653,
Validation Loss: 0.48492029309272766
Model with 160 neurons in the first layer and 64 neurons in the second layer -
Test Accuracy: 0.8118333220481873, Training Accuracy: 0.8496354222297668,
Validation Accuracy: 0.8054166436195374, Training Loss: 0.36426082253456116,
Validation Loss: 0.48917099833488464
Model with 160 neurons in the first layer and 80 neurons in the second layer -

```
Test Accuracy: 0.809166669845581, Training Accuracy: 0.8461979031562805,
Validation Accuracy: 0.8043749928474426, Training Loss: 0.3641396462917328,
Validation Loss: 0.4911010265350342
Model with 160 neurons in the first layer and 96 neurons in the second layer -
Test Accuracy: 0.8040000200271606, Training Accuracy: 0.8519791960716248,
Validation Accuracy: 0.7989583611488342, Training Loss: 0.3583512306213379,
Validation Loss: 0.5052505731582642
Model with 160 neurons in the first layer and 112 neurons in the second layer -
Test Accuracy: 0.8086666464805603, Training Accuracy: 0.8526041507720947,
Validation Accuracy: 0.8041666746139526, Training Loss: 0.35638508200645447,
Validation Loss: 0.49052751064300537
Model with 192 neurons in the first layer and 32 neurons in the second layer -
Test Accuracy: 0.812833309173584, Training Accuracy: 0.8439062237739563,
Validation Accuracy: 0.8070833086967468, Training Loss: 0.37143778800964355,
Validation Loss: 0.48632359504699707
Model with 192 neurons in the first layer and 48 neurons in the second layer -
Test Accuracy: 0.8013333082199097, Training Accuracy: 0.8473437428474426,
Validation Accuracy: 0.79708331823349, Training Loss: 0.3649325668811798,
Validation Loss: 0.49191176891326904
Model with 192 neurons in the first layer and 64 neurons in the second layer -
Test Accuracy: 0.809499979019165, Training Accuracy: 0.847083330154419,
Validation Accuracy: 0.8006250262260437, Training Loss: 0.36377066373825073,
Validation Loss: 0.4958113729953766
Model with 192 neurons in the first layer and 80 neurons in the second layer -
Test Accuracy: 0.8025000095367432, Training Accuracy: 0.8490625023841858,
Validation Accuracy: 0.8022916913032532, Training Loss: 0.36053144931793213,
Validation Loss: 0.4845883250236511
Model with 192 neurons in the first layer and 96 neurons in the second layer -
Test Accuracy: 0.8050000071525574, Training Accuracy: 0.8507291674613953,
Validation Accuracy: 0.79666668176651, Training Loss: 0.35723981261253357,
Validation Loss: 0.5070580840110779
Model with 192 neurons in the first layer and 112 neurons in the second layer -
Test Accuracy: 0.8059999942779541, Training Accuracy: 0.8520833253860474,
Validation Accuracy: 0.800208330154419, Training Loss: 0.3575883209705353,
Validation Loss: 0.49499502778053284
Model with 224 neurons in the first layer and 32 neurons in the second layer -
Test Accuracy: 0.8134999871253967, Training Accuracy: 0.8458853960037231,
Validation Accuracy: 0.8052083253860474, Training Loss: 0.37150782346725464,
Validation Loss: 0.480500191450119
Model with 224 neurons in the first layer and 48 neurons in the second layer -
Test Accuracy: 0.8088333606719971, Training Accuracy: 0.846875011920929,
Validation Accuracy: 0.8025000095367432, Training Loss: 0.3660374879837036,
Validation Loss: 0.4818115234375
Model with 224 neurons in the first layer and 64 neurons in the second layer -
Test Accuracy: 0.8101666569709778, Training Accuracy: 0.852135419845581,
Validation Accuracy: 0.7995833158493042, Training Loss: 0.3612259030342102,
Validation Loss: 0.5021263360977173
Model with 224 neurons in the first layer and 80 neurons in the second layer -
```

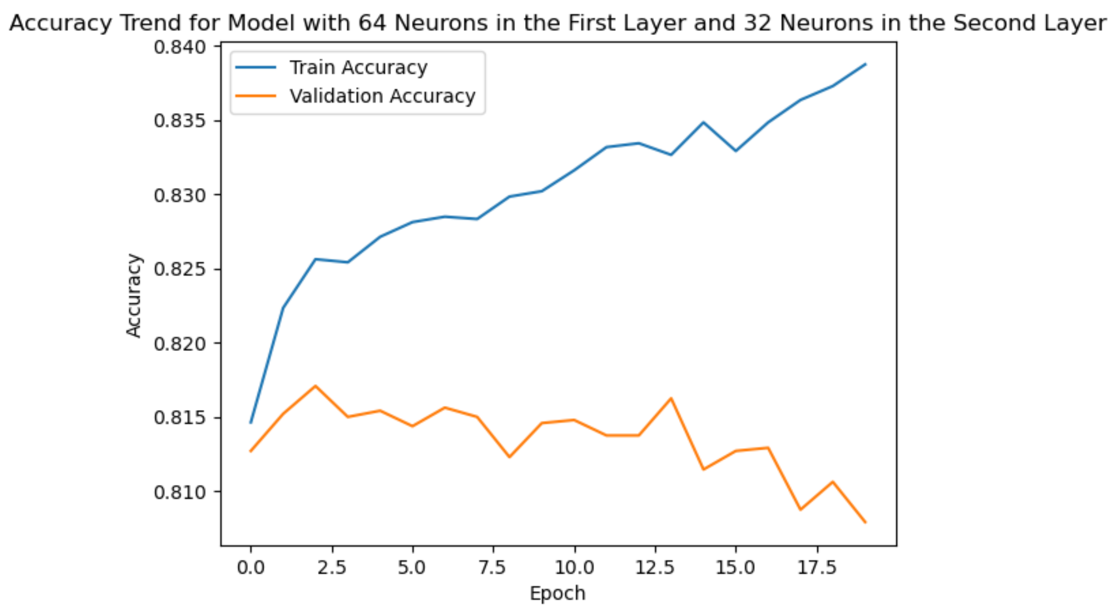
```
Test Accuracy: 0.8103333115577698, Training Accuracy: 0.8515625, Validation
Accuracy: 0.8035416603088379, Training Loss: 0.3581635653972626, Validation
Loss: 0.49874821305274963
Model with 224 neurons in the first layer and 96 neurons in the second layer -
Test Accuracy: 0.8086666464805603, Training Accuracy: 0.8512499928474426,
Validation Accuracy: 0.7983333468437195, Training Loss: 0.3610871136188507,
Validation Loss: 0.5040059089660645
Model with 224 neurons in the first layer and 112 neurons in the second layer -
Test Accuracy: 0.793666660785675, Training Accuracy: 0.8516666889190674,
Validation Accuracy: 0.7856249809265137, Training Loss: 0.3562335968017578,
Validation Loss: 0.5108903050422668

The best model has 128 neurons in the first layer and 32 neurons in the second
layer with Test Accuracy: 0.8163333535194397, Training Accuracy:
0.8432812690734863, Validation Accuracy: 0.8070833086967468, Training Loss:
0.3765762448310852, Validation Loss: 0.4808409512042999.
```

step 2: print summaries, metrics, graphs

```
[ ]: # Model Summary and Accuracy Trend Graph Block
for model, history, first_layer_neurons, second_layer_neurons in zip(models,␣
 ↪histories, first_layer_neurons_variations, second_layer_neurons_variations):
    # Print Model Summary
    print(f"Model Summary with {first_layer_neurons} neurons in the first layer␣
 ↪and {second_layer_neurons} neurons in the second layer:")
    model.summary()

    # Plot training history
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'Accuracy Trend for Model with {first_layer_neurons} Neurons in␣
 ↪the First Layer and {second_layer_neurons} Neurons in the Second Layer')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

    # Evaluate model and print Test Accuracy
    test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
    print(f"Test Accuracy: {test_accuracy}\n")
```

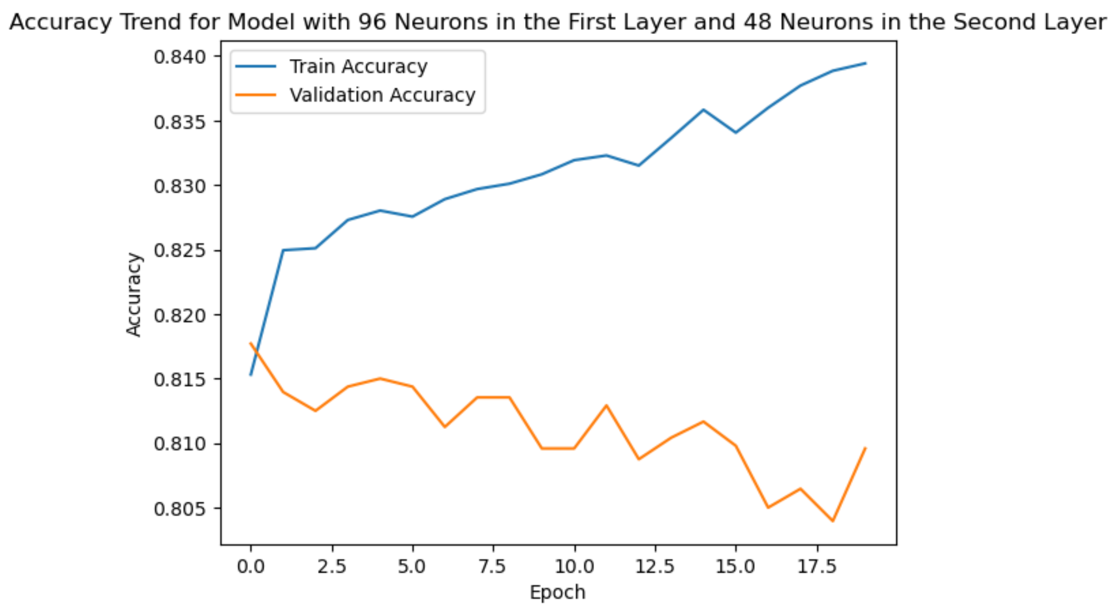
```
Model Summary with 64 neurons in the first layer and 32 neurons in the second
layer:
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 dense_42 (Dense)            (None, 64)                5888

 dense_43 (Dense)            (None, 32)                2080

 dense_44 (Dense)            (None, 1)                 33

=================================================================
Total params: 8001 (31.25 KB)
Trainable params: 8001 (31.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

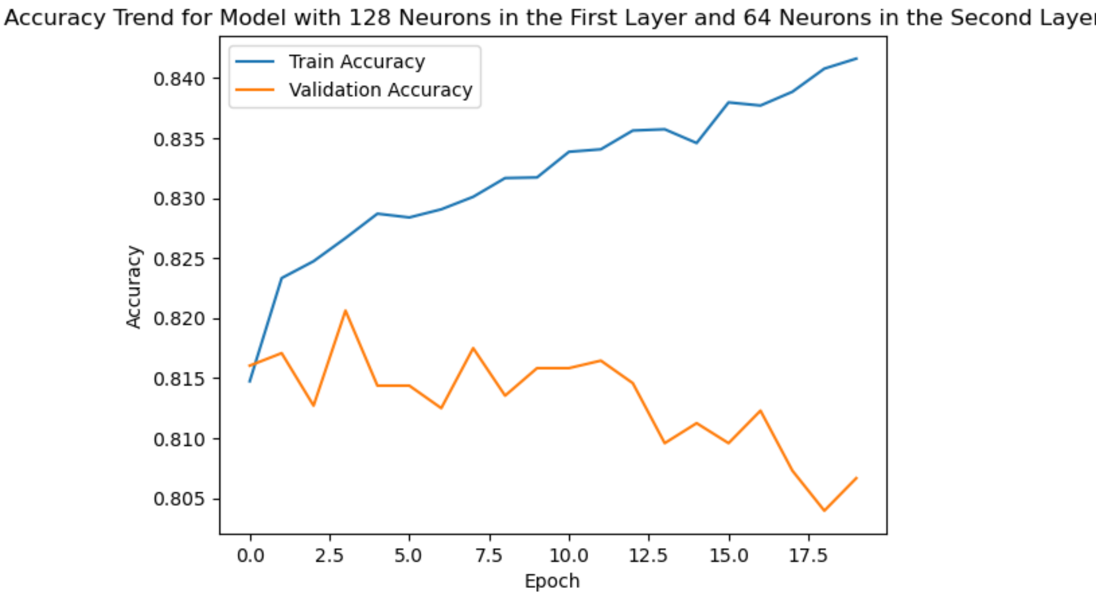



```
Test Accuracy: 0.812666654586792

Model Summary with 96 neurons in the first layer and 48 neurons in the second
layer:
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_45 (Dense)            (None, 64)                5888

 dense_46 (Dense)            (None, 48)                3120

 dense_47 (Dense)            (None, 1)                 49

=================================================================
Total params: 9057 (35.38 KB)
```

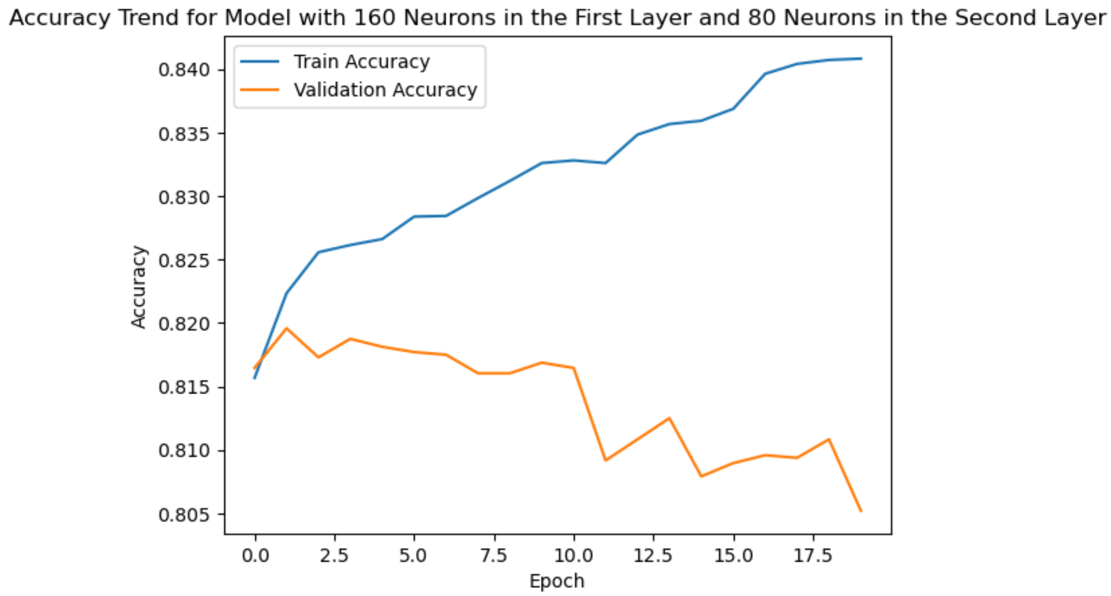
```
Trainable params: 9057 (35.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



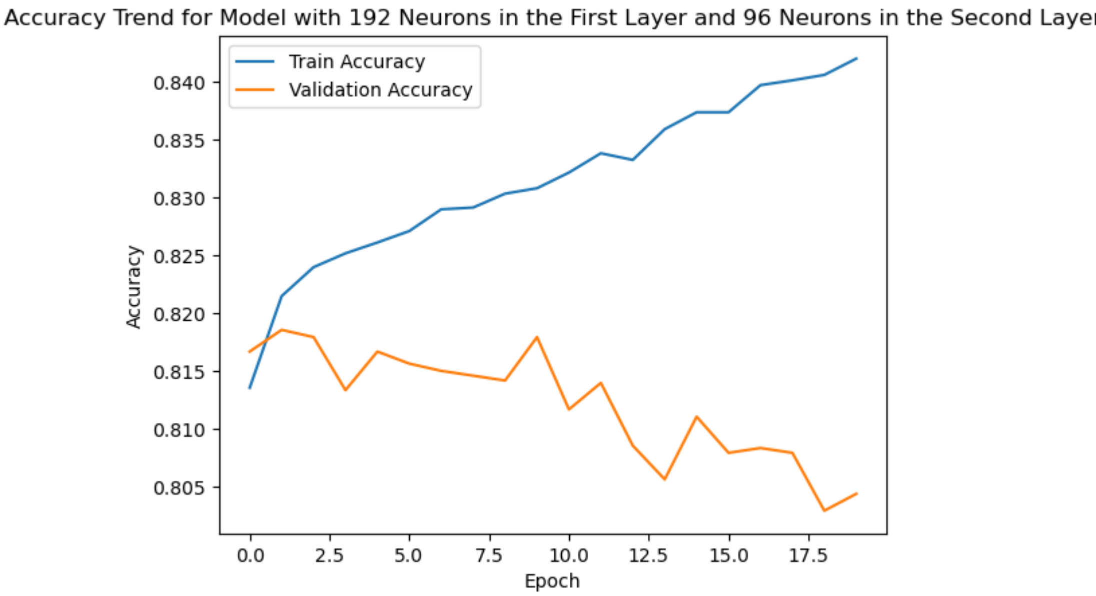

```
Test Accuracy: 0.8146666884422302

Model Summary with 128 neurons in the first layer and 64 neurons in the second
layer:
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_48 (Dense)            (None, 64)                5888

 dense_49 (Dense)            (None, 64)                4160

 dense_50 (Dense)            (None, 1)                 65

=================================================================
Total params: 10113 (39.50 KB)
Trainable params: 10113 (39.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



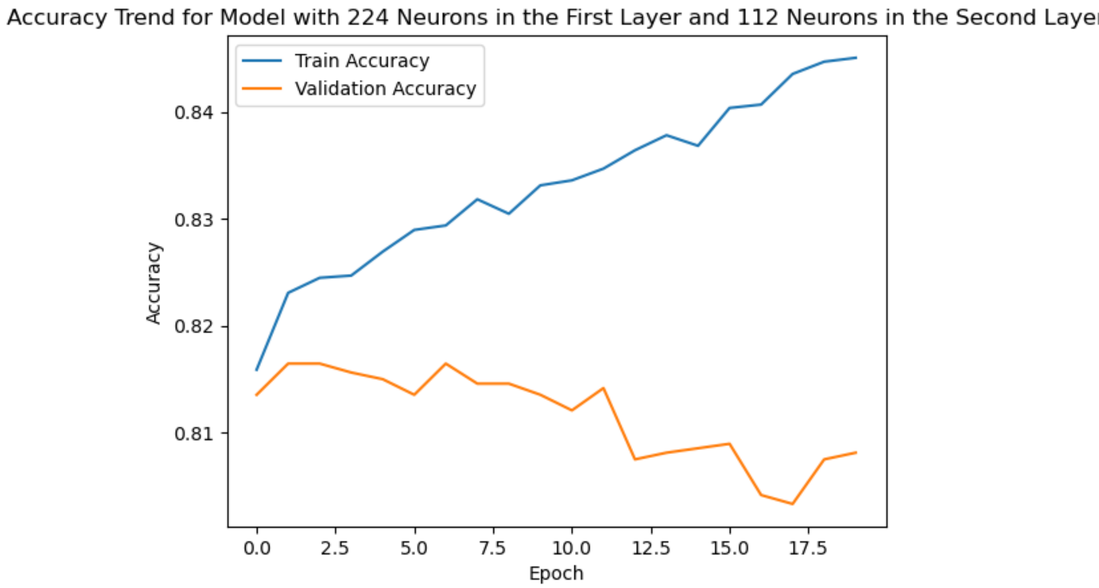

```
Test Accuracy: 0.8108333349227905

Model Summary with 160 neurons in the first layer and 80 neurons in the second
layer:
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_51 (Dense)            (None, 64)                5888

 dense_52 (Dense)            (None, 80)                5200

 dense_53 (Dense)            (None, 1)                 81

=================================================================
Total params: 11169 (43.63 KB)
Trainable params: 11169 (43.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```




```
Test Accuracy: 0.8133333325386047

Model Summary with 192 neurons in the first layer and 96 neurons in the second
layer:
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_54 (Dense)            (None, 64)                5888

 dense_55 (Dense)            (None, 96)                6240

 dense_56 (Dense)            (None, 1)                 97

=================================================================
Total params: 12225 (47.75 KB)
Trainable params: 12225 (47.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```




```
Test Accuracy: 0.8088333606719971

Model Summary with 224 neurons in the first layer and 112 neurons in the second
layer:
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_57 (Dense)            (None, 64)                5888

 dense_58 (Dense)            (None, 112)               7280

 dense_59 (Dense)            (None, 1)                 113

=================================================================
Total params: 13281 (51.88 KB)
Trainable params: 13281 (51.88 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

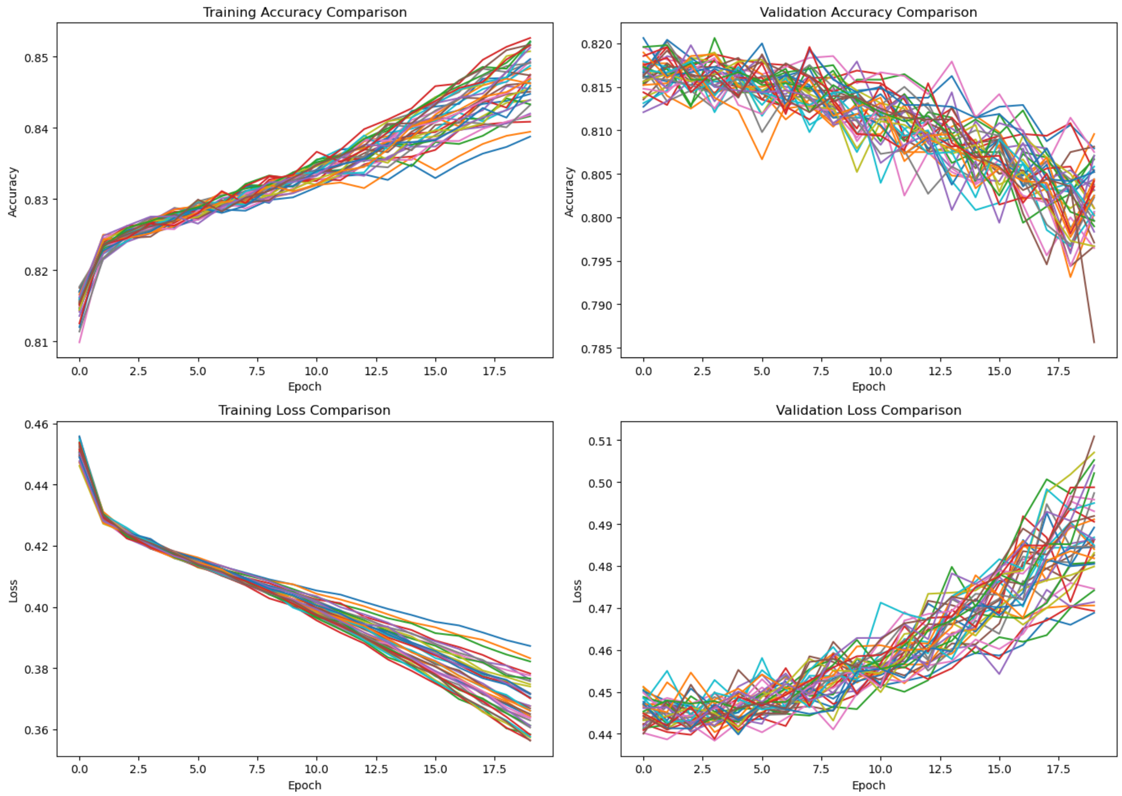



```
Test Accuracy: 0.8105000257492065
```

step 3: Compare models in collectively

[ ]:
```
import matplotlib.pyplot as plt

# Initialize subplots
fig, axs = plt.subplots(2, 2, figsize=(14, 10))

# Plot Training Accuracy Comparison
axs[0, 0].set_title('Training Accuracy Comparison')
for i, history in enumerate(histories):
    axs[0, 0].plot(history.history['accuracy'], label=f'Model {i+1}')
axs[0, 0].set_xlabel('Epoch')
axs[0, 0].set_ylabel('Accuracy')
#axs[0, 0].legend()

# Plot Validation Accuracy Comparison
axs[0, 1].set_title('Validation Accuracy Comparison')
for i, history in enumerate(histories):
    axs[0, 1].plot(history.history['val_accuracy'], label=f'Model {i+1}')
axs[0, 1].set_xlabel('Epoch')
axs[0, 1].set_ylabel('Accuracy')
#axs[0, 1].legend()

# Plot Training Loss Comparison
axs[1, 0].set_title('Training Loss Comparison')
```

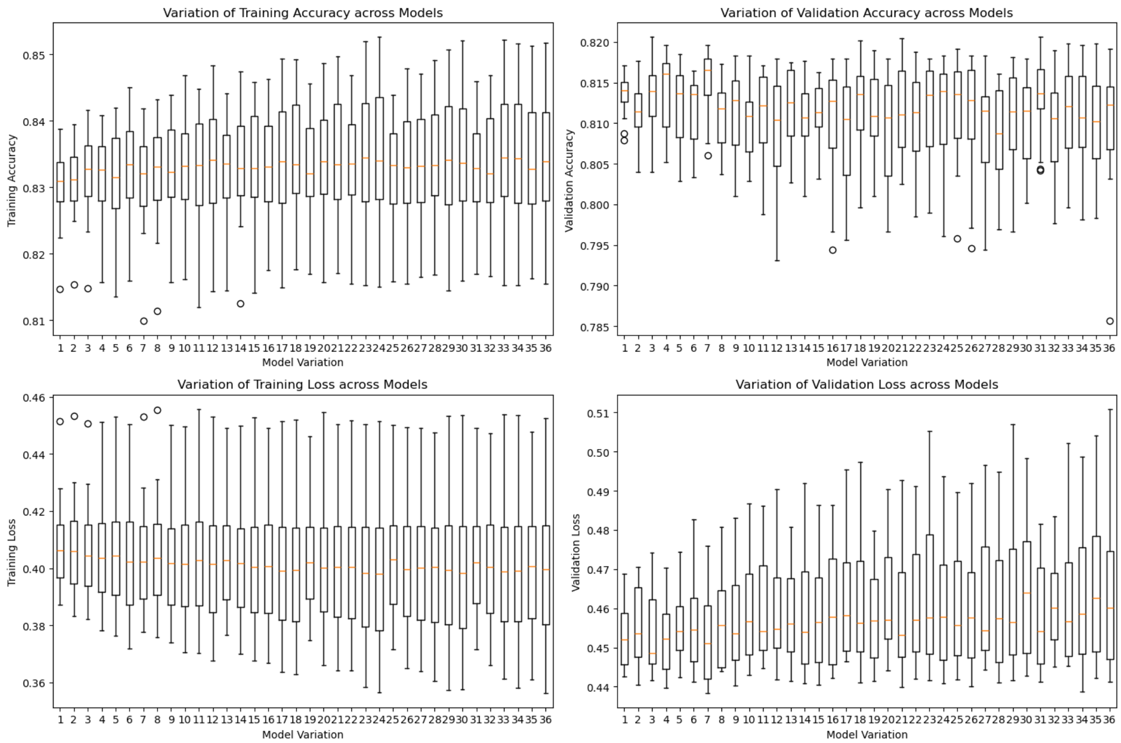
```
for i, history in enumerate(histories):
    axs[1, 0].plot(history.history['loss'], label=f'Model {i+1}')
axs[1, 0].set_xlabel('Epoch')
axs[1, 0].set_ylabel('Loss')
#axs[1, 0].legend()

# Plot Validation Loss Comparison
axs[1, 1].set_title('Validation Loss Comparison')
for i, history in enumerate(histories):
    axs[1, 1].plot(history.history['val_loss'], label=f'Model {i+1}')
axs[1, 1].set_xlabel('Epoch')
axs[1, 1].set_ylabel('Loss')
#axs[1, 1].legend()

plt.tight_layout()
plt.show()
```



step 4: box plot to see general variations of metrics

[ ]:
```
import matplotlib.pyplot as plt

# Extract metrics for all epochs
```

```python
train_accuracies = []
val_accuracies = []
train_losses = []
val_losses = []

for history in histories:
    train_accuracies.append(history.history['accuracy'])
    val_accuracies.append(history.history['val_accuracy'])
    train_losses.append(history.history['loss'])
    val_losses.append(history.history['val_loss'])

# Create subplots
fig, axs = plt.subplots(2, 2, figsize=(15, 10))

# Box plot for training accuracy
axs[0, 0].boxplot(train_accuracies, positions=range(1, len(train_accuracies)+1))
axs[0, 0].set_xlabel('Model Variation')
axs[0, 0].set_ylabel('Training Accuracy')
axs[0, 0].set_title('Variation of Training Accuracy across Models')

# Box plot for validation accuracy
axs[0, 1].boxplot(val_accuracies, positions=range(1, len(val_accuracies)+1))
axs[0, 1].set_xlabel('Model Variation')
axs[0, 1].set_ylabel('Validation Accuracy')
axs[0, 1].set_title('Variation of Validation Accuracy across Models')

# Box plot for training loss
axs[1, 0].boxplot(train_losses, positions=range(1, len(train_losses)+1))
axs[1, 0].set_xlabel('Model Variation')
axs[1, 0].set_ylabel('Training Loss')
axs[1, 0].set_title('Variation of Training Loss across Models')

# Box plot for validation loss
axs[1, 1].boxplot(val_losses, positions=range(1, len(val_losses)+1))
axs[1, 1].set_xlabel('Model Variation')
axs[1, 1].set_ylabel('Validation Loss')
axs[1, 1].set_title('Variation of Validation Loss across Models')

plt.tight_layout()
plt.show()
```

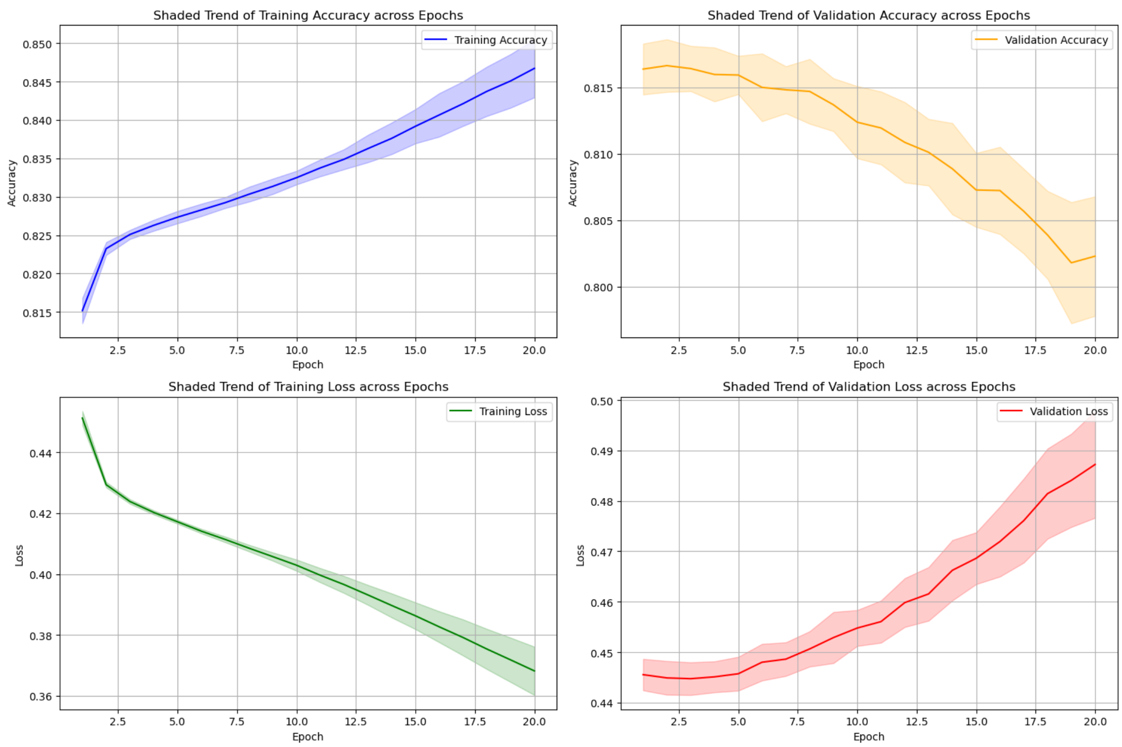


step 5: shaded variance trend graph

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Extract metrics for all epochs
train_accuracies = []
val_accuracies = []
train_losses = []
val_losses = []

for history in histories:
    train_accuracies.append(history.history['accuracy'])
    val_accuracies.append(history.history['val_accuracy'])
    train_losses.append(history.history['loss'])
    val_losses.append(history.history['val_loss'])

# Calculate mean and standard deviation for each epoch
mean_train_accuracy = np.mean(train_accuracies, axis=0)
std_train_accuracy = np.std(train_accuracies, axis=0)
mean_val_accuracy = np.mean(val_accuracies, axis=0)
std_val_accuracy = np.std(val_accuracies, axis=0)
mean_train_loss = np.mean(train_losses, axis=0)
std_train_loss = np.std(train_losses, axis=0)
```

```
mean_val_loss = np.mean(val_losses, axis=0)
std_val_loss = np.std(val_losses, axis=0)

# Create subplots
fig, axs = plt.subplots(2, 2, figsize=(15, 10))

# Plot shaded trend for training accuracy
axs[0, 0].plot(range(1, len(mean_train_accuracy) + 1), mean_train_accuracy,␣
↪label='Training Accuracy', color='blue')
axs[0, 0].fill_between(range(1, len(mean_train_accuracy) + 1),
                       mean_train_accuracy - std_train_accuracy,
                       mean_train_accuracy + std_train_accuracy,
                       color='blue', alpha=0.2)
axs[0, 0].set_xlabel('Epoch')
axs[0, 0].set_ylabel('Accuracy')
axs[0, 0].set_title('Shaded Trend of Training Accuracy across Epochs')
axs[0, 0].legend()
axs[0, 0].grid(True)

# Plot shaded trend for validation accuracy
axs[0, 1].plot(range(1, len(mean_val_accuracy) + 1), mean_val_accuracy,␣
↪label='Validation Accuracy', color='orange')
axs[0, 1].fill_between(range(1, len(mean_val_accuracy) + 1),
                       mean_val_accuracy - std_val_accuracy,
                       mean_val_accuracy + std_val_accuracy,
                       color='orange', alpha=0.2)
axs[0, 1].set_xlabel('Epoch')
axs[0, 1].set_ylabel('Accuracy')
axs[0, 1].set_title('Shaded Trend of Validation Accuracy across Epochs')
axs[0, 1].legend()
axs[0, 1].grid(True)

# Plot shaded trend for training loss
axs[1, 0].plot(range(1, len(mean_train_loss) + 1), mean_train_loss,␣
↪label='Training Loss', color='green')
axs[1, 0].fill_between(range(1, len(mean_train_loss) + 1),
                       mean_train_loss - std_train_loss,
                       mean_train_loss + std_train_loss,
                       color='green', alpha=0.2)
axs[1, 0].set_xlabel('Epoch')
axs[1, 0].set_ylabel('Loss')
axs[1, 0].set_title('Shaded Trend of Training Loss across Epochs')
axs[1, 0].legend()
axs[1, 0].grid(True)

# Plot shaded trend for validation loss
```

```
axs[1, 1].plot(range(1, len(mean_val_loss) + 1), mean_val_loss,␣
↪label='Validation Loss', color='red')
axs[1, 1].fill_between(range(1, len(mean_val_loss) + 1),
                        mean_val_loss - std_val_loss,
                        mean_val_loss + std_val_loss,
                        color='red', alpha=0.2)
axs[1, 1].set_xlabel('Epoch')
axs[1, 1].set_ylabel('Loss')
axs[1, 1].set_title('Shaded Trend of Validation Loss across Epochs')
axs[1, 1].legend()
axs[1, 1].grid(True)

plt.tight_layout()
plt.show()
```



step 6: best model

Previous code as redundant and incorporated in the main training block.

The best model has 64 neurons in the first layer and 80 neurons in the second layer with Test Accuracy: 0.8173333406448364, Training Accuracy: 0.8285416960716248, Validation Accuracy: 0.815833330154419, Training Loss: 0.4011897146701813, Validation Loss: 0.4485721290111542.

Any markdown about model results are at the moment of training at specific time. Differences are due to running models in another time.