arbash-malik-assignment-2

April 5, 2024

0.1 Loading helper libraries

[1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import keras

from sklearn.model_selection import train_test_split

from keras.models       import Sequential, Model
from keras.layers       import BatchNormalization, Conv2D, Dense, Dropout,␣
 ↪Flatten, GlobalAveragePooling2D, MaxPooling2D, Rescaling
from keras.callbacks    import EarlyStopping
from keras.optimizers   import Adam
from keras.applications import MobileNetV2
from keras.losses       import MeanSquaredError
from keras.utils        import to_categorical,set_random_seed

prng = np.random.RandomState(20240405)
set_random_seed(20240405)
keras.utils.set_random_seed(20240405)
```

0.2 Defining helper functions

[2]:

```
def plot_model_history(model_histories, labels, main_title):
    plt.figure(figsize=(14, 6))

    # Adding main title
    plt.suptitle(main_title, fontsize=14)

    # plotting training & validation accuracy
    plt.subplot(1, 2, 1)
    for model_history, label in zip(model_histories, labels):
        epochs = range(1, len(model_history.history['accuracy']) + 1)
        plt.plot(epochs, model_history.history['accuracy'], label=f'{label} -␣
 ↪Training')
```

```
        plt.plot(epochs, model_history.history['val_accuracy'], label=f'{label}␣
 ↪- Validation', linestyle="--")
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # plotting training & validation loss
    plt.subplot(1, 2, 2)
    for model_history, label in zip(model_histories, labels):
        epochs = range(1, len(model_history.history['loss']) + 1)
        plt.plot(epochs, model_history.history['loss'], label=f'{label} -␣
 ↪Training')
        plt.plot(epochs, model_history.history['val_loss'], label=f'{label} -␣
 ↪Validation', linestyle="--")
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

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

0.3 Loading Data

[84]:

```
from keras.datasets     import fashion_mnist

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
 ↪2, random_state=prng)

print(f"X_train: {X_train.shape}")
print(f"y_train: {y_train.shape}")
print('__')
print(f"X_val:  {X_val.shape}")
print(f"y_val:  {y_val.shape}")
print('__')
print(f"X_test:  {X_test.shape}")
print(f"y_test:  {y_test.shape}")
```

```
X_train: (48000, 28, 28)
y_train: (48000,)
__
X_val:  (12000, 28, 28)
y_val:  (12000,)
__
X_test:  (10000, 28, 28)
```

```
y_test:  (10000,)
```

0.4 1. What would be an appropriate metric to evaluate your models? Why?

An appropriate metric to evaluate models for the classification task on this dataset would be **Accuracy**. Accuracy measures the proportion of correctly classified instances out of the total instances. Accuracy is also very easy to interpret. It represents the percentage of correct predictions made by the model. Accuracy is also relevant in our case as it directly measures how well the model is performing (the goal is to correctly classify images into one of the ten categories

---

0.5 2. Get the data and show some example images from the data.

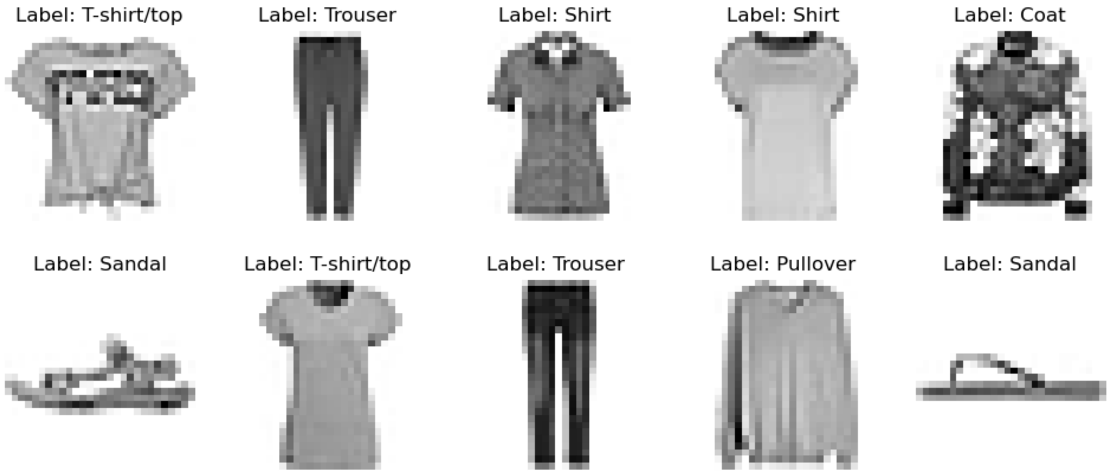
0.5.1 a) Training Data

[4]:
```python
class_names = ["T-shirt/top","Trouser","Pullover","Dress","Coat","Sandal","Shirt","Sneaker","Bag","Ankle boot"]

fig, axs = plt.subplots(2, 5, figsize=(12,5))
for i, ax in enumerate(axs.flatten()):
    ax.imshow(X_train[i], cmap="binary")
    ax.axis("off")
    ax.set_title(f"Label: {class_names[y_train[i]]}")
plt.tight_layout
plt.show()
```

0.5.2 b) Validation Data

[5]:

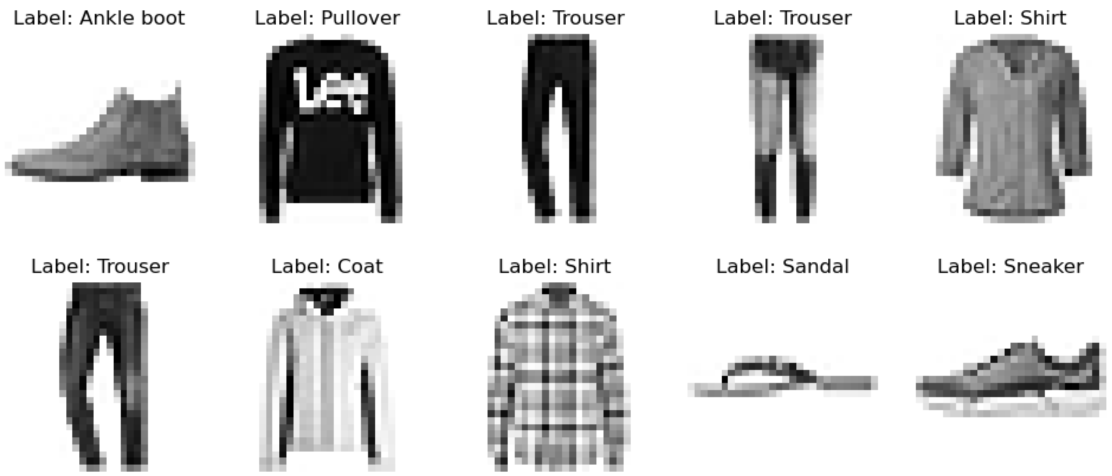
```
fig, axs = plt.subplots(2, 5, figsize=(12,5))
for i, ax in enumerate(axs.flatten()):
    ax.imshow(X_val[i], cmap="binary")
    ax.axis("off")
    ax.set_title(f"Label: {class_names[y_val[i]]}")
plt.tight_layout
plt.show()
```



0.5.3 c) Test Data

[6]:

```
fig, axs = plt.subplots(2, 5, figsize=(12,5))
for i, ax in enumerate(axs.flatten()):
    ax.imshow(X_test[i], cmap="binary")
    ax.axis("off")
    ax.set_title(f"Label: {class_names[y_test[i]]}")
plt.tight_layout
plt.show()
```



0.6 3. Train a simple fully connected single hidden layer network to predict the items.

0.6.1 Remember to normalize the data similar to what we did in class. Make sure that you use enough epochs so that the validation error begins to level off - provide a plot of the training history.

[85]:

```python
from keras.utils import to_categorical

class_names = ["T-shirt/top","Trouser","Pullover","Dress","Coat","Sandal","Shirt","Sneaker","Bag","Ankle boot"]
num_classes = len(class_names)
print(f"Dimension of y before transformation: {y_train.shape}")

# Convert target variables to categorical
y_sets = [y_train, y_test, y_val]
y_train, y_test, y_val = [to_categorical(y, num_classes=num_classes) for y in y_sets]
print(f"Dimension of y after transformation: {y_train.shape}")
```

```
Dimension of y before transformation: (48000,)
Dimension of y after transformation: (48000, 10)
```

[86]:

```python
from keras.models import Sequential
from keras.layers import Input, Flatten, Rescaling, Dense

model = Sequential([
    Input(shape=X_train.shape[1:]),
    Flatten(),
    Rescaling(1./255),
    Dense(100, activation='relu'),
    Dense(num_classes, activation='softmax')
])
print(model.summary())


# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

```
Model: "sequential_24"


 Layer (type)                        Output Shape                        ␣
 ↪Param #

 flatten_17 (Flatten)                (None, 784)                             ␣
 ↪  0

 rescaling_18 (Rescaling)            (None, 784)                             ␣
 ↪  0

 dense_42 (Dense)                    (None, 100)                          ␣
 ↪78,500

 dense_43 (Dense)                    (None, 10)                            ␣
 ↪1,010


 Total params: 79,510 (310.59 KB)


 Trainable params: 79,510 (310.59 KB)


 Non-trainable params: 0 (0.00 B)


None
```

[87]:

```
history = model.fit(X_train, y_train, validation_data=(X_val, y_val),␣
 ↪epochs=20,batch_size = 512)
```

```
Epoch 1/20
94/94                 1s 6ms/step -
accuracy: 0.6156 - loss: 1.1669 - val_accuracy: 0.8188 - val_loss: 0.5430
```

```
Epoch 2/20
94/94             0s 4ms/step -
accuracy: 0.8250 - loss: 0.5251 - val_accuracy: 0.8381 - val_loss: 0.4715
Epoch 3/20
94/94             0s 4ms/step -
accuracy: 0.8425 - loss: 0.4630 - val_accuracy: 0.8463 - val_loss: 0.4396
Epoch 4/20
94/94             0s 4ms/step -
accuracy: 0.8523 - loss: 0.4298 - val_accuracy: 0.8538 - val_loss: 0.4189
Epoch 5/20
94/94             0s 4ms/step -
accuracy: 0.8599 - loss: 0.4066 - val_accuracy: 0.8583 - val_loss: 0.4049
Epoch 6/20
94/94             0s 4ms/step -
accuracy: 0.8655 - loss: 0.3893 - val_accuracy: 0.8613 - val_loss: 0.3934
Epoch 7/20
94/94             0s 4ms/step -
accuracy: 0.8701 - loss: 0.3749 - val_accuracy: 0.8652 - val_loss: 0.3840
Epoch 8/20
94/94             0s 4ms/step -
accuracy: 0.8734 - loss: 0.3631 - val_accuracy: 0.8672 - val_loss: 0.3759
Epoch 9/20
94/94             0s 4ms/step -
accuracy: 0.8769 - loss: 0.3527 - val_accuracy: 0.8704 - val_loss: 0.3694
Epoch 10/20
94/94             0s 4ms/step -
accuracy: 0.8804 - loss: 0.3433 - val_accuracy: 0.8728 - val_loss: 0.3638
Epoch 11/20
94/94             0s 4ms/step -
accuracy: 0.8834 - loss: 0.3345 - val_accuracy: 0.8742 - val_loss: 0.3576
Epoch 12/20
94/94             0s 4ms/step -
accuracy: 0.8871 - loss: 0.3259 - val_accuracy: 0.8767 - val_loss: 0.3528
Epoch 13/20
94/94             0s 4ms/step -
accuracy: 0.8893 - loss: 0.3182 - val_accuracy: 0.8776 - val_loss: 0.3485
Epoch 14/20
94/94             0s 4ms/step -
accuracy: 0.8914 - loss: 0.3112 - val_accuracy: 0.8781 - val_loss: 0.3459
Epoch 15/20
94/94             0s 4ms/step -
accuracy: 0.8926 - loss: 0.3056 - val_accuracy: 0.8787 - val_loss: 0.3435
Epoch 16/20
94/94             0s 4ms/step -
accuracy: 0.8947 - loss: 0.3000 - val_accuracy: 0.8791 - val_loss: 0.3418
Epoch 17/20
94/94             0s 4ms/step -
accuracy: 0.8961 - loss: 0.2953 - val_accuracy: 0.8794 - val_loss: 0.3392
```

```
Epoch 18/20
94/94            0s 4ms/step -
accuracy: 0.8983 - loss: 0.2901 - val_accuracy: 0.8803 - val_loss: 0.3367
Epoch 19/20
94/94            0s 4ms/step -
accuracy: 0.8998 - loss: 0.2852 - val_accuracy: 0.8811 - val_loss: 0.3355
Epoch 20/20
94/94            0s 4ms/step -
accuracy: 0.9012 - loss: 0.2810 - val_accuracy: 0.8813 - val_loss: 0.3334
```

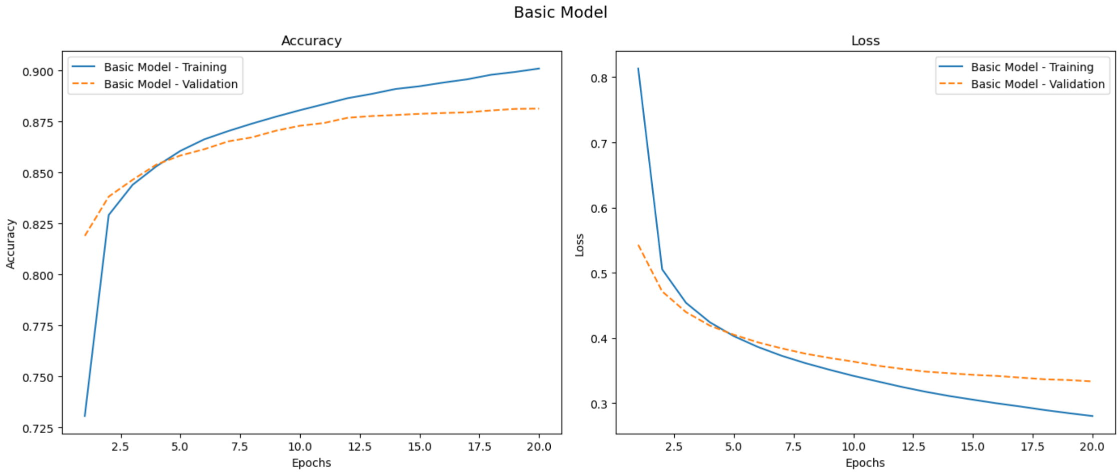
[88]:

```
# Evaluation of the model on the validation set
scores = model.evaluate(X_val, y_val)
print(f"Accuracy for Basic Model: {round(scores[1], 4)},Loss for Basic Model:␣
  ↪{round(scores[0], 4)}")
```

```
375/375              1s 1ms/step -
accuracy: 0.8842 - loss: 0.3255
Accuracy for Basic Model: 0.8813,Loss for Basic Model: 0.3309
```

[89]:

```
plot_model_history([history], ['Basic Model'],'Basic Model')
```



0.7 4. Experiment with different network architectures and settings (number of hidden layers, number of nodes, regularization, etc.)

0.7.1 Train at least 3 models. Explain what you have tried and how it worked.

What I will iterate in my models are node numbers in each layer, adding hidden layers, adding dropout, adding early stopping clause.

Increasing the number of nodes allows for the model to capture the relationship between images and labels more clearly. Adding a hidden layer in a neural network increases the model’s capacity to learn complex patterns and representations from the input data.

Model 1: Nodes increased to 256 on first layer, added a second hidden layer with 100 nodes.

[90]:
```
model1 = Sequential([
    Input(shape=X_train.shape[1:]),
    Flatten(),
    Rescaling(1./255),
    Dense(256, activation='relu'),
    Dense(100, activation='relu'),
    Dense(num_classes, activation='softmax')
])
print(model1.summary())


# Compile the model
model1.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

```
Model: "sequential_25"


 Layer (type)                          Output Shape                        ␣
↪Param #

 flatten_18 (Flatten)                  (None, 784)                             ␣
↪   0

 rescaling_19 (Rescaling)              (None, 784)                             ␣
↪   0

 dense_44 (Dense)                      (None, 256)                         ␣
↪200,960

 dense_45 (Dense)                      (None, 100)                          ␣
↪25,700

 dense_46 (Dense)                      (None, 10)                            ␣
↪1,010


 Total params: 227,670 (889.34 KB)


 Trainable params: 227,670 (889.34 KB)
 Non-trainable params: 0 (0.00 B)
```

```
None
```

[91]:

```
history1 = model1.fit(X_train, y_train, validation_data=(X_val, y_val),␣
 ↪epochs=20,batch_size = 512)
```

```
Epoch 1/20
94/94           2s 8ms/step -
accuracy: 0.6732 - loss: 0.9736 - val_accuracy: 0.8355 - val_loss: 0.4664
Epoch 2/20
94/94           1s 6ms/step -
accuracy: 0.8434 - loss: 0.4493 - val_accuracy: 0.8580 - val_loss: 0.4026
Epoch 3/20
94/94           1s 6ms/step -
accuracy: 0.8604 - loss: 0.3944 - val_accuracy: 0.8702 - val_loss: 0.3727
Epoch 4/20
94/94           1s 6ms/step -
accuracy: 0.8716 - loss: 0.3625 - val_accuracy: 0.8748 - val_loss: 0.3566
Epoch 5/20
94/94           1s 6ms/step -
accuracy: 0.8807 - loss: 0.3392 - val_accuracy: 0.8793 - val_loss: 0.3441
Epoch 6/20
94/94           1s 6ms/step -
accuracy: 0.8875 - loss: 0.3201 - val_accuracy: 0.8825 - val_loss: 0.3352
Epoch 7/20
94/94           1s 6ms/step -
accuracy: 0.8911 - loss: 0.3049 - val_accuracy: 0.8810 - val_loss: 0.3286
Epoch 8/20
94/94           1s 6ms/step -
accuracy: 0.8948 - loss: 0.2929 - val_accuracy: 0.8863 - val_loss: 0.3199
Epoch 9/20
94/94           1s 6ms/step -
accuracy: 0.8990 - loss: 0.2823 - val_accuracy: 0.8900 - val_loss: 0.3114
Epoch 10/20
94/94           1s 6ms/step -
accuracy: 0.9026 - loss: 0.2706 - val_accuracy: 0.8917 - val_loss: 0.3060
Epoch 11/20
94/94           1s 6ms/step -
accuracy: 0.9046 - loss: 0.2621 - val_accuracy: 0.8924 - val_loss: 0.3023
Epoch 12/20
94/94           1s 6ms/step -
accuracy: 0.9079 - loss: 0.2520 - val_accuracy: 0.8915 - val_loss: 0.3026
Epoch 13/20
94/94           1s 6ms/step -
accuracy: 0.9111 - loss: 0.2454 - val_accuracy: 0.8900 - val_loss: 0.3062
Epoch 14/20
```

```
94/94              1s 6ms/step -
accuracy: 0.9130 - loss: 0.2371 - val_accuracy: 0.8915 - val_loss: 0.3069
Epoch 15/20
94/94              1s 6ms/step -
accuracy: 0.9171 - loss: 0.2287 - val_accuracy: 0.8924 - val_loss: 0.3074
Epoch 16/20
94/94              1s 7ms/step -
accuracy: 0.9203 - loss: 0.2213 - val_accuracy: 0.8894 - val_loss: 0.3133
Epoch 17/20
94/94              1s 8ms/step -
accuracy: 0.9226 - loss: 0.2142 - val_accuracy: 0.8900 - val_loss: 0.3153
Epoch 18/20
94/94              1s 8ms/step -
accuracy: 0.9248 - loss: 0.2085 - val_accuracy: 0.8929 - val_loss: 0.3103
Epoch 19/20
94/94              1s 8ms/step -
accuracy: 0.9263 - loss: 0.2027 - val_accuracy: 0.8916 - val_loss: 0.3087
Epoch 20/20
94/94              1s 8ms/step -
accuracy: 0.9281 - loss: 0.1993 - val_accuracy: 0.8917 - val_loss: 0.3122
```

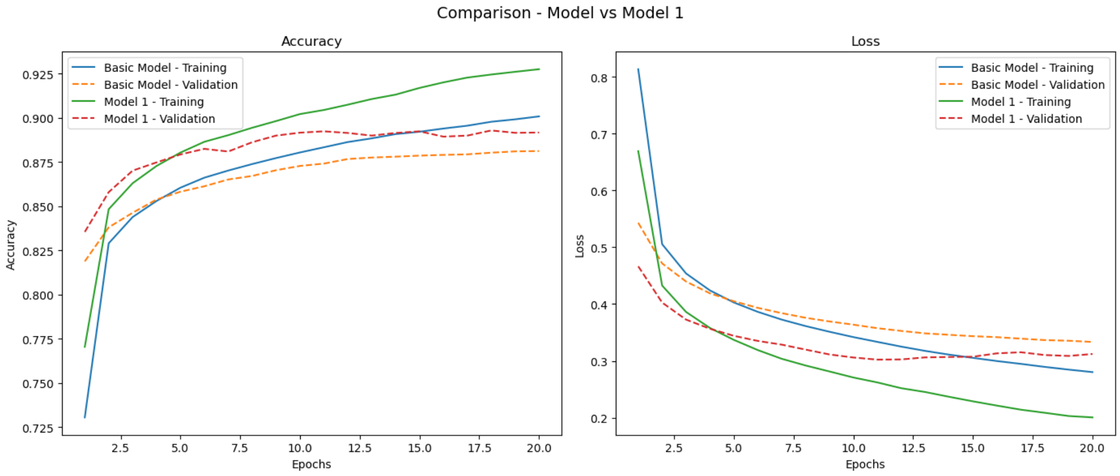
[92]:

```
# Evaluation of the model on the validation set
scores1 = model1.evaluate(X_val, y_val)
print("\n")
print(f"Accuracy for Model 1: {round(scores1[1], 4)},Loss for Model 1:␣
  ↪{round(scores1[0], 4)}")
```

```
375/375              1s 2ms/step -
accuracy: 0.8960 - loss: 0.3036


Accuracy for Model 1: 0.8917,Loss for Model 1: 0.3103
```

[93]:

```
plot_model_history([history,history1], ['Basic Model','Model 1'],'Comparison -␣
  ↪Model vs Model 1')
```



Comparing to our basic model, our first iteration of the model where increased the nodes on the first layer and and added a layer worked pretty well as both accuracy and loss improve

Model 2: 256 Nodes on first layer, added a second hidden layer with 100 nodes, added a third hidden layer with 50 nodes

[16]:
```
model2 = Sequential([
    Input(shape=X_train.shape[1:]),
    Flatten(),
    Rescaling(1./255),
    Dense(256, activation='relu'),
    Dense(100, activation='relu'),
    Dense(50, activation='relu'),
    Dense(num_classes, activation='softmax')
])

# Compile the model
model2.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

print(model2.summary())
```

```
Model: "sequential_2"


 Layer (type)                       Output Shape                          ␣
↪Param #

 flatten_2 (Flatten)                (None, 784)                             ␣
↪   0
```

```
 rescaling_2 (Rescaling)              (None, 784)                              ␣
 ↪  0

 dense_5 (Dense)                      (None, 256)                          ␣
 ↪200,960

 dense_6 (Dense)                      (None, 100)                           ␣
 ↪25,700

 dense_7 (Dense)                      (None, 50)                             ␣
 ↪5,050

 dense_8 (Dense)                      (None, 10)                               ␣
 ↪510



 Total params: 232,220 (907.11 KB)


 Trainable params: 232,220 (907.11 KB)


 Non-trainable params: 0 (0.00 B)


None
```

[17]:

```
history2 = model2.fit(X_train, y_train, validation_data=(X_val, y_val),␣
 ↪epochs=20, batch_size = 512)
```

```
Epoch 1/20
94/94                 4s 17ms/step -
accuracy: 0.6416 - loss: 1.0570 - val_accuracy: 0.8225 - val_loss: 0.5074
Epoch 2/20
94/94                 1s 12ms/step -
accuracy: 0.8382 - loss: 0.4625 - val_accuracy: 0.8390 - val_loss: 0.4593
Epoch 3/20
94/94                 1s 11ms/step -
accuracy: 0.8543 - loss: 0.4182 - val_accuracy: 0.8641 - val_loss: 0.3879
Epoch 4/20
94/94                 1s 13ms/step -
accuracy: 0.8642 - loss: 0.3822 - val_accuracy: 0.8685 - val_loss: 0.3720
Epoch 5/20
94/94                 1s 13ms/step -
accuracy: 0.8734 - loss: 0.3568 - val_accuracy: 0.8701 - val_loss: 0.3631
Epoch 6/20
94/94                 1s 14ms/step -
accuracy: 0.8792 - loss: 0.3360 - val_accuracy: 0.8734 - val_loss: 0.3522
```

```
Epoch 7/20
94/94                  1s 12ms/step -
accuracy: 0.8848 - loss: 0.3201 - val_accuracy: 0.8753 - val_loss: 0.3422
Epoch 8/20
94/94                  1s 13ms/step -
accuracy: 0.8884 - loss: 0.3074 - val_accuracy: 0.8798 - val_loss: 0.3374
Epoch 9/20
94/94                  1s 13ms/step -
accuracy: 0.8925 - loss: 0.2943 - val_accuracy: 0.8783 - val_loss: 0.3371
Epoch 10/20
94/94                  1s 12ms/step -
accuracy: 0.8965 - loss: 0.2839 - val_accuracy: 0.8827 - val_loss: 0.3239
Epoch 11/20
94/94                  1s 12ms/step -
accuracy: 0.9012 - loss: 0.2710 - val_accuracy: 0.8862 - val_loss: 0.3197
Epoch 12/20
94/94                  1s 13ms/step -
accuracy: 0.9025 - loss: 0.2639 - val_accuracy: 0.8843 - val_loss: 0.3231
Epoch 13/20
94/94                  1s 13ms/step -
accuracy: 0.9062 - loss: 0.2560 - val_accuracy: 0.8817 - val_loss: 0.3342
Epoch 14/20
94/94                  1s 15ms/step -
accuracy: 0.9080 - loss: 0.2488 - val_accuracy: 0.8773 - val_loss: 0.3479
Epoch 15/20
94/94                  1s 13ms/step -
accuracy: 0.9114 - loss: 0.2404 - val_accuracy: 0.8773 - val_loss: 0.3472
Epoch 16/20
94/94                  1s 13ms/step -
accuracy: 0.9152 - loss: 0.2298 - val_accuracy: 0.8827 - val_loss: 0.3312
Epoch 17/20
94/94                  1s 12ms/step -
accuracy: 0.9187 - loss: 0.2200 - val_accuracy: 0.8834 - val_loss: 0.3334
Epoch 18/20
94/94                  1s 13ms/step -
accuracy: 0.9220 - loss: 0.2139 - val_accuracy: 0.8845 - val_loss: 0.3295
Epoch 19/20
94/94                  1s 12ms/step -
accuracy: 0.9254 - loss: 0.2083 - val_accuracy: 0.8821 - val_loss: 0.3341
Epoch 20/20
94/94                  1s 13ms/step -
accuracy: 0.9247 - loss: 0.2060 - val_accuracy: 0.8796 - val_loss: 0.3519
```

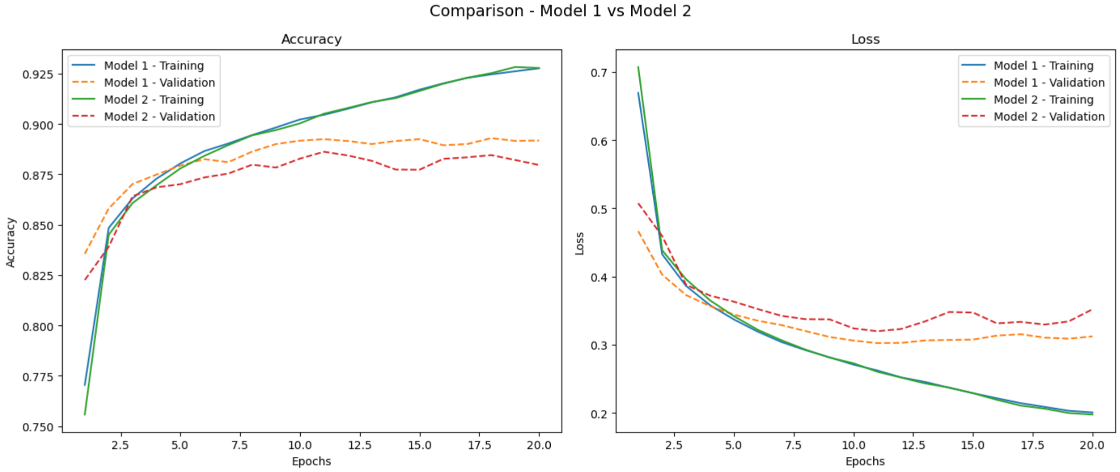
[18]:

```
# Evaluation of the model on the validation set
scores2 = model2.evaluate(X_val, y_val)
```

```
print(f"Accuracy for Model 2: {round(scores2[1], 4)},Loss for Model 2:␣
  ↪{round(scores2[0], 4)}")

print("\n")
```

```
375/375               1s 3ms/step -
accuracy: 0.8781 - loss: 0.3501


Accuracy for Model 2: 0.8796,Loss for Model 2: 0.3535
```

[94]:
```
plot_model_history([history1,history2], ['Model 1','Model 2'],'Comparison -␣
  ↪Model 1 vs Model 2')
```



Comparing to our first iteration model, our second iteration of the model where we added a third hidden layer is the worse in accuracy as well as the loss is worse in validation set

Model 3: 100 Nodes on first layer, added a second hidden layer with 25 nodes, added a third hidden layer with 50 nodes

[19]:
```
model3 = Sequential([
    Input(shape=X_train.shape[1:]),
    Flatten(),
    Rescaling(1./255),
    Dense(100, activation='relu'),
    Dense(75, activation='relu'),
    Dense(50, activation='relu'),
    Dense(num_classes, activation='softmax')
])
print(model3.summary())
```

```
# Compile the model
model3.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

```
Model: "sequential_3"


 Layer (type)                           Output Shape                        ␣
 ↪Param #

 flatten_3 (Flatten)                    (None, 784)                             ␣
 ↪   0

 rescaling_3 (Rescaling)                (None, 784)                             ␣
 ↪   0

 dense_9 (Dense)                        (None, 100)                          ␣
 ↪78,500

 dense_10 (Dense)                       (None, 75)                            ␣
 ↪7,575

 dense_11 (Dense)                       (None, 50)                            ␣
 ↪3,800

 dense_12 (Dense)                       (None, 10)                              ␣
 ↪510


 Total params: 90,385 (353.07 KB)

 Trainable params: 90,385 (353.07 KB)

 Non-trainable params: 0 (0.00 B)

None
```

[20]:
```
history3 = model3.fit(X_train, y_train, validation_data=(X_val, y_val),␣
 ↪epochs=100, batch_size = 512
                      ,callbacks=[EarlyStopping(monitor='val_accuracy',␣
 ↪patience=10)])
```

```
Epoch 1/100
```

```
94/94             4s 16ms/step -
accuracy: 0.6037 - loss: 1.2183 - val_accuracy: 0.8240 - val_loss: 0.4995
Epoch 2/100
94/94             2s 17ms/step -
accuracy: 0.8270 - loss: 0.4932 - val_accuracy: 0.8407 - val_loss: 0.4456
Epoch 3/100
94/94             2s 20ms/step -
accuracy: 0.8488 - loss: 0.4281 - val_accuracy: 0.8549 - val_loss: 0.4106
Epoch 4/100
94/94             2s 16ms/step -
accuracy: 0.8606 - loss: 0.3963 - val_accuracy: 0.8641 - val_loss: 0.3821
Epoch 5/100
94/94             2s 23ms/step -
accuracy: 0.8697 - loss: 0.3699 - val_accuracy: 0.8708 - val_loss: 0.3646
Epoch 6/100
94/94             2s 16ms/step -
accuracy: 0.8749 - loss: 0.3511 - val_accuracy: 0.8740 - val_loss: 0.3526
Epoch 7/100
94/94             1s 9ms/step -
accuracy: 0.8803 - loss: 0.3371 - val_accuracy: 0.8783 - val_loss: 0.3428
Epoch 8/100
94/94             1s 12ms/step -
accuracy: 0.8843 - loss: 0.3248 - val_accuracy: 0.8790 - val_loss: 0.3358
Epoch 9/100
94/94             1s 11ms/step -
accuracy: 0.8874 - loss: 0.3128 - val_accuracy: 0.8813 - val_loss: 0.3331
Epoch 10/100
94/94             1s 12ms/step -
accuracy: 0.8899 - loss: 0.3017 - val_accuracy: 0.8828 - val_loss: 0.3295
Epoch 11/100
94/94             1s 9ms/step -
accuracy: 0.8932 - loss: 0.2932 - val_accuracy: 0.8815 - val_loss: 0.3279
Epoch 12/100
94/94             1s 9ms/step -
accuracy: 0.8960 - loss: 0.2847 - val_accuracy: 0.8828 - val_loss: 0.3252
Epoch 13/100
94/94             1s 11ms/step -
accuracy: 0.8977 - loss: 0.2776 - val_accuracy: 0.8834 - val_loss: 0.3210
Epoch 14/100
94/94             1s 12ms/step -
accuracy: 0.8999 - loss: 0.2693 - val_accuracy: 0.8857 - val_loss: 0.3204
Epoch 15/100
94/94             1s 10ms/step -
accuracy: 0.9032 - loss: 0.2618 - val_accuracy: 0.8873 - val_loss: 0.3185
Epoch 16/100
94/94             1s 11ms/step -
accuracy: 0.9053 - loss: 0.2562 - val_accuracy: 0.8874 - val_loss: 0.3183
Epoch 17/100
```

```
94/94              1s 11ms/step -
accuracy: 0.9077 - loss: 0.2494 - val_accuracy: 0.8857 - val_loss: 0.3225
Epoch 18/100
94/94              1s 10ms/step -
accuracy: 0.9096 - loss: 0.2456 - val_accuracy: 0.8848 - val_loss: 0.3219
Epoch 19/100
94/94              1s 10ms/step -
accuracy: 0.9136 - loss: 0.2384 - val_accuracy: 0.8855 - val_loss: 0.3265
Epoch 20/100
94/94              1s 10ms/step -
accuracy: 0.9142 - loss: 0.2345 - val_accuracy: 0.8839 - val_loss: 0.3307
Epoch 21/100
94/94              1s 10ms/step -
accuracy: 0.9156 - loss: 0.2296 - val_accuracy: 0.8822 - val_loss: 0.3378
Epoch 22/100
94/94              1s 10ms/step -
accuracy: 0.9170 - loss: 0.2262 - val_accuracy: 0.8808 - val_loss: 0.3466
Epoch 23/100
94/94              1s 10ms/step -
accuracy: 0.9181 - loss: 0.2232 - val_accuracy: 0.8769 - val_loss: 0.3588
Epoch 24/100
94/94              1s 11ms/step -
accuracy: 0.9190 - loss: 0.2223 - val_accuracy: 0.8847 - val_loss: 0.3414
Epoch 25/100
94/94              1s 10ms/step -
accuracy: 0.9210 - loss: 0.2173 - val_accuracy: 0.8848 - val_loss: 0.3428
Epoch 26/100
94/94              1s 10ms/step -
accuracy: 0.9223 - loss: 0.2133 - val_accuracy: 0.8867 - val_loss: 0.3391
```

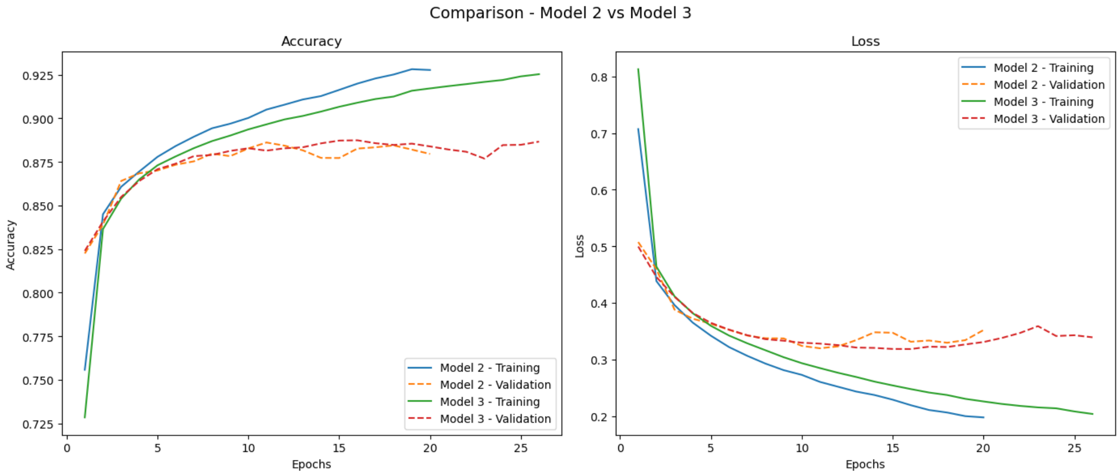
[21]:

```
# Evaluation of the model on the validation set
scores3 = model3.evaluate(X_val, y_val)

print(f"Accuracy for Model 3: {round(scores3[1], 4)},Loss for Model 3:␣
 ↪{round(scores3[0], 4)}")

print("\n")
```

```
375/375              1s 3ms/step -
accuracy: 0.8874 - loss: 0.3328


Accuracy for Model 3: 0.8867,Loss for Model 3: 0.3393
```

[95]:

```
plot_model_history([history2,history3], ['Model 2','Model 3'],'Comparison -␣
 ↪Model 2 vs Model 3')
```



Our third model where we reduced the number of nodes in each layer as well as added an early stopping with a patience level of 5 with 100 epochs does better but it is not comparable as number of epochs are higher

Model 4: Nodes increased to 512 on first layer, added a Dropout value of 50%, added a second hidden layer with 256 nodes, added a Dropout value of 50%, added a third hidden layer with 100 nodes.

[22]:
```
model4 = Sequential([
    Input(shape=X_train.shape[1:]),
    Flatten(),
    Rescaling(1./255),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(100, activation='relu'),
    Dense(num_classes, activation='softmax')
])
print(model4.summary())


# Compile the model
model4.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

```
Model: "sequential_4"



 Layer (type)                        Output Shape                       ␣
↪Param #
```

```
 flatten_4 (Flatten)                  (None, 784)                               ␣
↪   0

 rescaling_4 (Rescaling)              (None, 784)                               ␣
↪   0

 dense_13 (Dense)                     (None, 512)                           ␣
↪401,920

 dropout (Dropout)                    (None, 512)                               ␣
↪   0

 dense_14 (Dense)                     (None, 256)                           ␣
↪131,328

 dropout_1 (Dropout)                  (None, 256)                               ␣
↪   0

 dense_15 (Dense)                     (None, 100)                            ␣
↪25,700

 dense_16 (Dense)                     (None, 10)                              ␣
↪1,010


 Total params: 559,958 (2.14 MB)


 Trainable params: 559,958 (2.14 MB)


 Non-trainable params: 0 (0.00 B)


None
```

[23]:

```
history4 = model4.fit(X_train, y_train, validation_data=(X_val, y_val),␣
 ↪epochs=100, batch_size = 512
                      ,callbacks=[EarlyStopping(monitor='val_accuracy',␣
 ↪patience=10)])
```

```
Epoch 1/100
94/94                 5s 27ms/step -
accuracy: 0.5372 - loss: 1.2692 - val_accuracy: 0.8238 - val_loss: 0.4889
Epoch 2/100
94/94                 2s 23ms/step -
accuracy: 0.8032 - loss: 0.5466 - val_accuracy: 0.8415 - val_loss: 0.4186
```

```
Epoch 3/100
94/94             2s 23ms/step -
accuracy: 0.8294 - loss: 0.4753 - val_accuracy: 0.8597 - val_loss: 0.3760
Epoch 4/100
94/94             2s 23ms/step -
accuracy: 0.8412 - loss: 0.4398 - val_accuracy: 0.8625 - val_loss: 0.3719
Epoch 5/100
94/94             2s 23ms/step -
accuracy: 0.8517 - loss: 0.4163 - val_accuracy: 0.8664 - val_loss: 0.3618
Epoch 6/100
94/94             2s 25ms/step -
accuracy: 0.8553 - loss: 0.4015 - val_accuracy: 0.8705 - val_loss: 0.3426
Epoch 7/100
94/94             2s 25ms/step -
accuracy: 0.8553 - loss: 0.3908 - val_accuracy: 0.8726 - val_loss: 0.3408
Epoch 8/100
94/94             3s 27ms/step -
accuracy: 0.8609 - loss: 0.3747 - val_accuracy: 0.8772 - val_loss: 0.3344
Epoch 9/100
94/94             3s 29ms/step -
accuracy: 0.8672 - loss: 0.3706 - val_accuracy: 0.8754 - val_loss: 0.3409
Epoch 10/100
94/94             3s 28ms/step -
accuracy: 0.8678 - loss: 0.3638 - val_accuracy: 0.8788 - val_loss: 0.3235
Epoch 11/100
94/94             3s 28ms/step -
accuracy: 0.8728 - loss: 0.3501 - val_accuracy: 0.8801 - val_loss: 0.3201
Epoch 12/100
94/94             3s 28ms/step -
accuracy: 0.8722 - loss: 0.3469 - val_accuracy: 0.8837 - val_loss: 0.3150
Epoch 13/100
94/94             2s 25ms/step -
accuracy: 0.8751 - loss: 0.3395 - val_accuracy: 0.8850 - val_loss: 0.3181
Epoch 14/100
94/94             2s 25ms/step -
accuracy: 0.8752 - loss: 0.3417 - val_accuracy: 0.8817 - val_loss: 0.3232
Epoch 15/100
94/94             2s 24ms/step -
accuracy: 0.8795 - loss: 0.3310 - val_accuracy: 0.8825 - val_loss: 0.3205
Epoch 16/100
94/94             2s 25ms/step -
accuracy: 0.8804 - loss: 0.3273 - val_accuracy: 0.8838 - val_loss: 0.3127
Epoch 17/100
94/94             2s 26ms/step -
accuracy: 0.8814 - loss: 0.3227 - val_accuracy: 0.8877 - val_loss: 0.3071
Epoch 18/100
94/94             3s 28ms/step -
accuracy: 0.8810 - loss: 0.3240 - val_accuracy: 0.8868 - val_loss: 0.3074
```

```
Epoch 19/100
94/94             2s 25ms/step -
accuracy: 0.8838 - loss: 0.3155 - val_accuracy: 0.8873 - val_loss: 0.3069
Epoch 20/100
94/94             2s 24ms/step -
accuracy: 0.8816 - loss: 0.3158 - val_accuracy: 0.8859 - val_loss: 0.3074
Epoch 21/100
94/94             2s 24ms/step -
accuracy: 0.8835 - loss: 0.3129 - val_accuracy: 0.8868 - val_loss: 0.3094
Epoch 22/100
94/94             2s 23ms/step -
accuracy: 0.8850 - loss: 0.3068 - val_accuracy: 0.8875 - val_loss: 0.3082
Epoch 23/100
94/94             3s 27ms/step -
accuracy: 0.8866 - loss: 0.3017 - val_accuracy: 0.8882 - val_loss: 0.3135
Epoch 24/100
94/94             3s 28ms/step -
accuracy: 0.8879 - loss: 0.3013 - val_accuracy: 0.8907 - val_loss: 0.3020
Epoch 25/100
94/94             3s 28ms/step -
accuracy: 0.8892 - loss: 0.2990 - val_accuracy: 0.8909 - val_loss: 0.2952
Epoch 26/100
94/94             3s 27ms/step -
accuracy: 0.8903 - loss: 0.2931 - val_accuracy: 0.8936 - val_loss: 0.2950
Epoch 27/100
94/94             3s 26ms/step -
accuracy: 0.8905 - loss: 0.2963 - val_accuracy: 0.8896 - val_loss: 0.3036
Epoch 28/100
94/94             2s 24ms/step -
accuracy: 0.8954 - loss: 0.2850 - val_accuracy: 0.8926 - val_loss: 0.2981
Epoch 29/100
94/94             2s 24ms/step -
accuracy: 0.8916 - loss: 0.2881 - val_accuracy: 0.8923 - val_loss: 0.2961
Epoch 30/100
94/94             3s 31ms/step -
accuracy: 0.8955 - loss: 0.2822 - val_accuracy: 0.8928 - val_loss: 0.2974
Epoch 31/100
94/94             3s 36ms/step -
accuracy: 0.8934 - loss: 0.2816 - val_accuracy: 0.8931 - val_loss: 0.2950
Epoch 32/100
94/94             3s 29ms/step -
accuracy: 0.8958 - loss: 0.2794 - val_accuracy: 0.8932 - val_loss: 0.2993
Epoch 33/100
94/94             3s 29ms/step -
accuracy: 0.8965 - loss: 0.2781 - val_accuracy: 0.8936 - val_loss: 0.2936
Epoch 34/100
94/94             3s 30ms/step -
accuracy: 0.8979 - loss: 0.2777 - val_accuracy: 0.8898 - val_loss: 0.3049
```

```
Epoch 35/100
94/94                 2s 23ms/step -
accuracy: 0.8964 - loss: 0.2766 - val_accuracy: 0.8926 - val_loss: 0.2944
Epoch 36/100
94/94                 2s 24ms/step -
accuracy: 0.8989 - loss: 0.2761 - val_accuracy: 0.8920 - val_loss: 0.2963
```

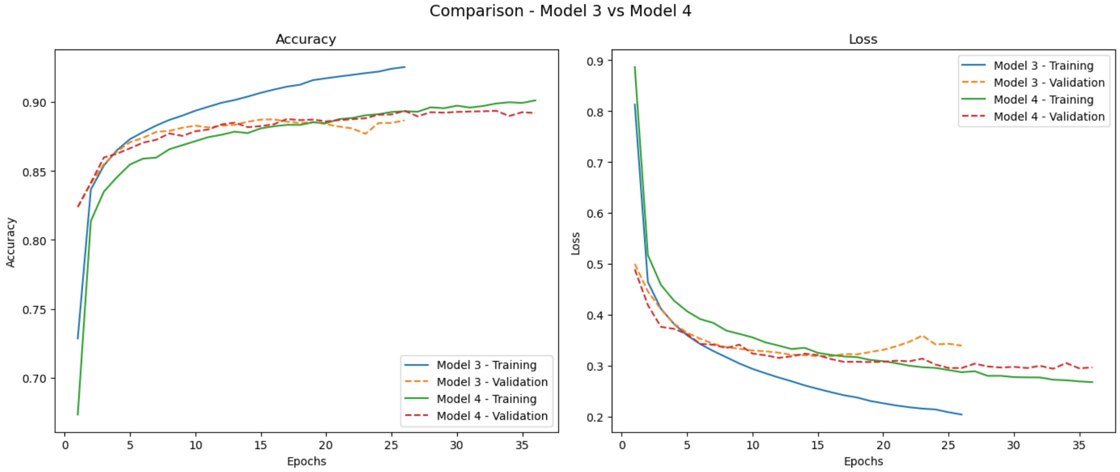
[24]:

```
# Evaluation of the model on the validation set
scores4 = model4.evaluate(X_val, y_val)

print(f"Accuracy for Model 4: {round(scores4[1], 4)},Loss for Model 4:␣
  ↪{round(scores4[0], 4)}")

print("\n")
```

```
375/375               1s 4ms/step -
accuracy: 0.8915 - loss: 0.2966


Accuracy for Model 4: 0.892,Loss for Model 4: 0.2971
```

[96]:

```
plot_model_history([history3,history4], ['Model 3','Model 4'],'Comparison -␣
  ↪Model 3 vs Model 4')
```



Here the 4th model improves on 3rd model slightly when we add a dropout on the first two layers of 50%

Model 5: Nodes increased to 256 on first layer, added a Dropout value of 50%, added a second hidden layer with 100 nodes, added a Dropout value of 50%, added a third hidden layer with 20 nodes.

[25]:
```
model5 = Sequential([
    Input(shape=X_train.shape[1:]),
    Flatten(),
    Rescaling(1./255),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(100, activation='relu'),
    Dropout(0.5),
    Dense(20, activation='relu'),
    Dense(num_classes, activation='softmax')
])
print(model5.summary())


# Compile the model
model5.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

```
Model: "sequential_5"



 Layer (type)                          Output Shape                          ␣
 ↪Param #

 flatten_5 (Flatten)                   (None, 784)                               ␣
 ↪   0

 rescaling_5 (Rescaling)               (None, 784)                               ␣
 ↪   0

 dense_17 (Dense)                      (None, 256)                           ␣
 ↪200,960

 dropout_2 (Dropout)                   (None, 256)                               ␣
 ↪   0

 dense_18 (Dense)                      (None, 100)                            ␣
 ↪25,700

 dropout_3 (Dropout)                   (None, 100)                               ␣
 ↪   0

 dense_19 (Dense)                      (None, 20)                              ␣
 ↪2,020
 dense_20 (Dense)                     (None, 10)                                     ␣
 ↪210
```

```
 Total params: 228,890 (894.10 KB)
```

```
 Trainable params: 228,890 (894.10 KB)
```

```
 Non-trainable params: 0 (0.00 B)
```

```
None
```

[26]:

```
history5 = model5.fit(X_train, y_train, validation_data=(X_val, y_val),␣
  ↪epochs=100, batch_size = 512
                      ,callbacks=[EarlyStopping(monitor='val_accuracy',␣
  ↪patience=10)])
```

```
Epoch 1/100
94/94             5s 20ms/step -
accuracy: 0.3851 - loss: 1.6822 - val_accuracy: 0.7893 - val_loss: 0.5955
Epoch 2/100
94/94             1s 14ms/step -
accuracy: 0.7534 - loss: 0.7016 - val_accuracy: 0.8225 - val_loss: 0.4785
Epoch 3/100
94/94             2s 16ms/step -
accuracy: 0.7957 - loss: 0.5778 - val_accuracy: 0.8420 - val_loss: 0.4273
Epoch 4/100
94/94             1s 15ms/step -
accuracy: 0.8192 - loss: 0.5173 - val_accuracy: 0.8511 - val_loss: 0.4002
Epoch 5/100
94/94             1s 14ms/step -
accuracy: 0.8298 - loss: 0.4851 - val_accuracy: 0.8553 - val_loss: 0.3911
Epoch 6/100
94/94             2s 16ms/step -
accuracy: 0.8388 - loss: 0.4581 - val_accuracy: 0.8604 - val_loss: 0.3790
Epoch 7/100
94/94             1s 15ms/step -
accuracy: 0.8447 - loss: 0.4409 - val_accuracy: 0.8652 - val_loss: 0.3674
Epoch 8/100
94/94             2s 15ms/step -
accuracy: 0.8496 - loss: 0.4280 - val_accuracy: 0.8694 - val_loss: 0.3577
Epoch 9/100
94/94             2s 19ms/step -
accuracy: 0.8519 - loss: 0.4185 - val_accuracy: 0.8659 - val_loss: 0.3595
Epoch 10/100
94/94             2s 16ms/step -
```

```
accuracy: 0.8537 - loss: 0.4082 - val_accuracy: 0.8725 - val_loss: 0.3443
Epoch 11/100
94/94             2s 18ms/step -
accuracy: 0.8585 - loss: 0.4010 - val_accuracy: 0.8714 - val_loss: 0.3453
Epoch 12/100
94/94             2s 17ms/step -
accuracy: 0.8631 - loss: 0.3938 - val_accuracy: 0.8723 - val_loss: 0.3448
Epoch 13/100
94/94             2s 19ms/step -
accuracy: 0.8637 - loss: 0.3872 - val_accuracy: 0.8760 - val_loss: 0.3346
Epoch 14/100
94/94             2s 16ms/step -
accuracy: 0.8690 - loss: 0.3742 - val_accuracy: 0.8783 - val_loss: 0.3329
Epoch 15/100
94/94             2s 16ms/step -
accuracy: 0.8688 - loss: 0.3672 - val_accuracy: 0.8800 - val_loss: 0.3308
Epoch 16/100
94/94             1s 15ms/step -
accuracy: 0.8715 - loss: 0.3609 - val_accuracy: 0.8791 - val_loss: 0.3297
Epoch 17/100
94/94             2s 17ms/step -
accuracy: 0.8717 - loss: 0.3593 - val_accuracy: 0.8786 - val_loss: 0.3250
Epoch 18/100
94/94             2s 16ms/step -
accuracy: 0.8744 - loss: 0.3550 - val_accuracy: 0.8810 - val_loss: 0.3213
Epoch 19/100
94/94             2s 16ms/step -
accuracy: 0.8744 - loss: 0.3527 - val_accuracy: 0.8806 - val_loss: 0.3213
Epoch 20/100
94/94             2s 15ms/step -
accuracy: 0.8755 - loss: 0.3455 - val_accuracy: 0.8805 - val_loss: 0.3200
Epoch 21/100
94/94             2s 16ms/step -
accuracy: 0.8768 - loss: 0.3404 - val_accuracy: 0.8832 - val_loss: 0.3180
Epoch 22/100
94/94             1s 15ms/step -
accuracy: 0.8796 - loss: 0.3375 - val_accuracy: 0.8815 - val_loss: 0.3192
Epoch 23/100
94/94             2s 16ms/step -
accuracy: 0.8784 - loss: 0.3364 - val_accuracy: 0.8845 - val_loss: 0.3194
Epoch 24/100
94/94             2s 15ms/step -
accuracy: 0.8814 - loss: 0.3345 - val_accuracy: 0.8855 - val_loss: 0.3135
Epoch 25/100
94/94             2s 15ms/step -
accuracy: 0.8834 - loss: 0.3272 - val_accuracy: 0.8849 - val_loss: 0.3146
Epoch 26/100
94/94             2s 16ms/step -
```

```
accuracy: 0.8803 - loss: 0.3292 - val_accuracy: 0.8857 - val_loss: 0.3114
Epoch 27/100
94/94             2s 15ms/step -
accuracy: 0.8825 - loss: 0.3258 - val_accuracy: 0.8866 - val_loss: 0.3073
Epoch 28/100
94/94             1s 14ms/step -
accuracy: 0.8847 - loss: 0.3215 - val_accuracy: 0.8880 - val_loss: 0.3048
Epoch 29/100
94/94             1s 14ms/step -
accuracy: 0.8832 - loss: 0.3162 - val_accuracy: 0.8860 - val_loss: 0.3109
Epoch 30/100
94/94             2s 15ms/step -
accuracy: 0.8869 - loss: 0.3131 - val_accuracy: 0.8889 - val_loss: 0.3094
Epoch 31/100
94/94             1s 15ms/step -
accuracy: 0.8865 - loss: 0.3133 - val_accuracy: 0.8873 - val_loss: 0.3036
Epoch 32/100
94/94             2s 19ms/step -
accuracy: 0.8847 - loss: 0.3135 - val_accuracy: 0.8860 - val_loss: 0.3075
Epoch 33/100
94/94             1s 14ms/step -
accuracy: 0.8870 - loss: 0.3092 - val_accuracy: 0.8877 - val_loss: 0.3083
Epoch 34/100
94/94             2s 17ms/step -
accuracy: 0.8888 - loss: 0.3080 - val_accuracy: 0.8923 - val_loss: 0.3032
Epoch 35/100
94/94             2s 17ms/step -
accuracy: 0.8912 - loss: 0.3060 - val_accuracy: 0.8882 - val_loss: 0.3039
Epoch 36/100
94/94             2s 16ms/step -
accuracy: 0.8888 - loss: 0.3072 - val_accuracy: 0.8886 - val_loss: 0.3071
Epoch 37/100
94/94             1s 15ms/step -
accuracy: 0.8907 - loss: 0.3023 - val_accuracy: 0.8897 - val_loss: 0.3066
Epoch 38/100
94/94             2s 22ms/step -
accuracy: 0.8913 - loss: 0.3016 - val_accuracy: 0.8906 - val_loss: 0.3038
Epoch 39/100
94/94             2s 21ms/step -
accuracy: 0.8897 - loss: 0.3021 - val_accuracy: 0.8923 - val_loss: 0.3000
Epoch 40/100
94/94             2s 16ms/step -
accuracy: 0.8916 - loss: 0.2985 - val_accuracy: 0.8920 - val_loss: 0.2981
Epoch 41/100
94/94             2s 16ms/step -
accuracy: 0.8936 - loss: 0.2942 - val_accuracy: 0.8929 - val_loss: 0.3004
Epoch 42/100
94/94             2s 15ms/step -
```

```
accuracy: 0.8908 - loss: 0.2942 - val_accuracy: 0.8945 - val_loss: 0.2986
Epoch 43/100
94/94            2s 16ms/step -
accuracy: 0.8932 - loss: 0.2932 - val_accuracy: 0.8924 - val_loss: 0.2965
Epoch 44/100
94/94            2s 16ms/step -
accuracy: 0.8930 - loss: 0.2944 - val_accuracy: 0.8940 - val_loss: 0.2962
Epoch 45/100
94/94            1s 15ms/step -
accuracy: 0.8964 - loss: 0.2825 - val_accuracy: 0.8916 - val_loss: 0.2992
Epoch 46/100
94/94            2s 16ms/step -
accuracy: 0.8957 - loss: 0.2877 - val_accuracy: 0.8904 - val_loss: 0.3037
Epoch 47/100
94/94            1s 15ms/step -
accuracy: 0.8930 - loss: 0.2901 - val_accuracy: 0.8917 - val_loss: 0.2970
Epoch 48/100
94/94            2s 17ms/step -
accuracy: 0.8966 - loss: 0.2834 - val_accuracy: 0.8904 - val_loss: 0.3068
Epoch 49/100
94/94            1s 15ms/step -
accuracy: 0.8946 - loss: 0.2869 - val_accuracy: 0.8930 - val_loss: 0.3005
Epoch 50/100
94/94            1s 15ms/step -
accuracy: 0.8998 - loss: 0.2769 - val_accuracy: 0.8888 - val_loss: 0.3050
Epoch 51/100
94/94            1s 15ms/step -
accuracy: 0.8972 - loss: 0.2820 - val_accuracy: 0.8894 - val_loss: 0.3074
Epoch 52/100
94/94            2s 16ms/step -
accuracy: 0.8983 - loss: 0.2813 - val_accuracy: 0.8916 - val_loss: 0.3034
```

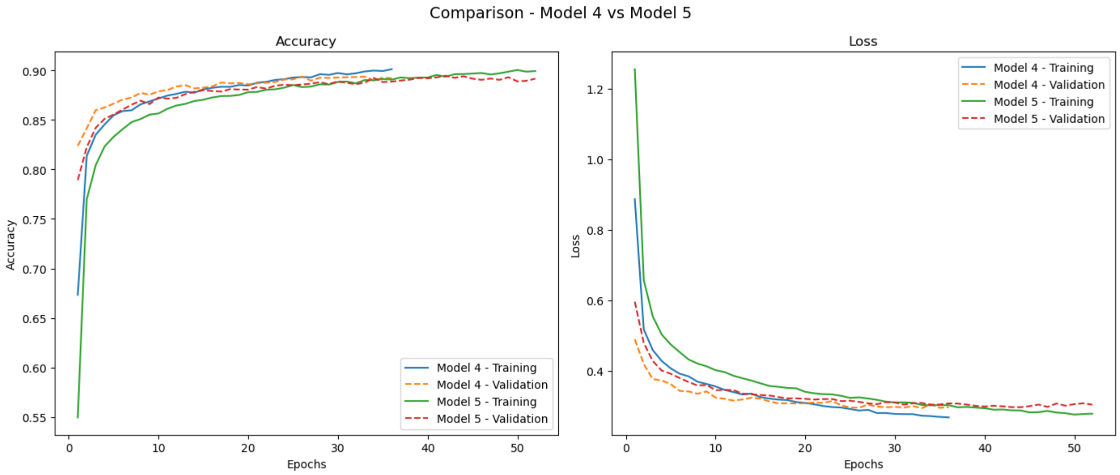
[27]:

```
# Evaluation of the model on the validation set
scores5 = model5.evaluate(X_val, y_val)

print("\n")
print(f"Accuracy for Model 5: {round(scores5[1], 4)},Loss for Model 5:␣
  ↪{round(scores5[0], 4)}")
```

```
375/375              1s 3ms/step -
accuracy: 0.8881 - loss: 0.3049


Accuracy for Model 5: 0.8916,Loss for Model 5: 0.3041
```

[98]:

```
plot_model_history([history4,history5], ['Model 4','Model 5'],'Comparison -␣
  ↪Model 4 vs Model 5')
```



The 5th model is the same as 4th, it only differs in the number of nodes, where we reduce them. As you can see it does not improve the accuracy or the loss

---

0.8 5. Try to improve the accuracy of your model by using convolution.

0.8.1 Train at least two different models (you can vary the number of convolutional and pooling layers or whether you include a fully connected layer before the output, etc.)

[28]:
```
from keras.layers import Reshape

preprocess = Sequential([
    Reshape(target_shape=(X_train.shape[1], X_train.shape[2], 1)),  #␣
  ↪explicitly state the 4th (channel) dimension
    Rescaling(1./255)
])

X_sets = [X_train, X_test, X_val]
X_train_4D, X_test_4D, X_val_4D = [preprocess(X) for X in X_sets]
```

Model 6: Adding Convolution, Pooling, and Fully Connected Layer with No Hidden Layer

[29]:
```
from keras.layers import Conv2D, MaxPooling2D

# Build the model
model6 = Sequential([
    Input(shape=X_train_4D.shape[1:]),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(num_classes, activation='softmax')
])

# Compile the model
model6.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
print(model6.summary())
```

```
Model: "sequential_7"


 Layer (type)                        Output Shape                     ␣
 ↪Param #

 conv2d (Conv2D)                     (None, 26, 26, 32)                   ␣
 ↪320

 max_pooling2d (MaxPooling2D)        (None, 13, 13, 32)                   ␣
 ↪  0

 flatten_6 (Flatten)                 (None, 5408)                         ␣
 ↪  0

 dense_21 (Dense)                    (None, 10)                        ␣
 ↪54,090


 Total params: 54,410 (212.54 KB)

 Trainable params: 54,410 (212.54 KB)

 Non-trainable params: 0 (0.00 B)

None
```

[30]:

```
# Fit the model
history6 = model6.fit(
    X_train_4D, y_train, validation_data=(X_val_4D, y_val), epochs=100,␣
 ↪batch_size=2048,
    callbacks=[EarlyStopping(monitor='val_accuracy', patience=10)]
)
```

```
Epoch 1/100
```

```
24/24              5s 160ms/step -
accuracy: 0.4792 - loss: 1.7679 - val_accuracy: 0.7388 - val_loss: 0.8033
Epoch 2/100
24/24              3s 143ms/step -
accuracy: 0.7583 - loss: 0.7315 - val_accuracy: 0.7792 - val_loss: 0.6113
Epoch 3/100
24/24              4s 157ms/step -
accuracy: 0.7979 - loss: 0.5844 - val_accuracy: 0.8102 - val_loss: 0.5344
Epoch 4/100
24/24              4s 150ms/step -
accuracy: 0.8251 - loss: 0.5165 - val_accuracy: 0.8267 - val_loss: 0.4903
Epoch 5/100
24/24              4s 156ms/step -
accuracy: 0.8380 - loss: 0.4740 - val_accuracy: 0.8378 - val_loss: 0.4633
Epoch 6/100
24/24              4s 151ms/step -
accuracy: 0.8484 - loss: 0.4457 - val_accuracy: 0.8444 - val_loss: 0.4432
Epoch 7/100
24/24              4s 146ms/step -
accuracy: 0.8561 - loss: 0.4250 - val_accuracy: 0.8501 - val_loss: 0.4262
Epoch 8/100
24/24              4s 146ms/step -
accuracy: 0.8620 - loss: 0.4083 - val_accuracy: 0.8558 - val_loss: 0.4118
Epoch 9/100
24/24              4s 146ms/step -
accuracy: 0.8672 - loss: 0.3944 - val_accuracy: 0.8601 - val_loss: 0.4001
Epoch 10/100
24/24              4s 150ms/step -
accuracy: 0.8710 - loss: 0.3827 - val_accuracy: 0.8636 - val_loss: 0.3905
Epoch 11/100
24/24              4s 160ms/step -
accuracy: 0.8746 - loss: 0.3727 - val_accuracy: 0.8657 - val_loss: 0.3822
Epoch 12/100
24/24              4s 147ms/step -
accuracy: 0.8771 - loss: 0.3641 - val_accuracy: 0.8689 - val_loss: 0.3749
Epoch 13/100
24/24              4s 147ms/step -
accuracy: 0.8801 - loss: 0.3564 - val_accuracy: 0.8707 - val_loss: 0.3684
Epoch 14/100
24/24              4s 146ms/step -
accuracy: 0.8813 - loss: 0.3494 - val_accuracy: 0.8724 - val_loss: 0.3625
Epoch 15/100
24/24              4s 149ms/step -
accuracy: 0.8830 - loss: 0.3431 - val_accuracy: 0.8742 - val_loss: 0.3572
Epoch 16/100
24/24              4s 154ms/step -
accuracy: 0.8843 - loss: 0.3373 - val_accuracy: 0.8756 - val_loss: 0.3523
Epoch 17/100
```

```
24/24             4s 161ms/step -
accuracy: 0.8865 - loss: 0.3319 - val_accuracy: 0.8777 - val_loss: 0.3478
Epoch 18/100
24/24             4s 158ms/step -
accuracy: 0.8881 - loss: 0.3269 - val_accuracy: 0.8788 - val_loss: 0.3436
Epoch 19/100
24/24             4s 146ms/step -
accuracy: 0.8898 - loss: 0.3222 - val_accuracy: 0.8801 - val_loss: 0.3398
Epoch 20/100
24/24             4s 147ms/step -
accuracy: 0.8914 - loss: 0.3179 - val_accuracy: 0.8815 - val_loss: 0.3363
Epoch 21/100
24/24             4s 144ms/step -
accuracy: 0.8924 - loss: 0.3138 - val_accuracy: 0.8821 - val_loss: 0.3330
Epoch 22/100
24/24             4s 154ms/step -
accuracy: 0.8932 - loss: 0.3099 - val_accuracy: 0.8829 - val_loss: 0.3299
Epoch 23/100
24/24             4s 160ms/step -
accuracy: 0.8945 - loss: 0.3063 - val_accuracy: 0.8836 - val_loss: 0.3270
Epoch 24/100
24/24             4s 168ms/step -
accuracy: 0.8956 - loss: 0.3028 - val_accuracy: 0.8849 - val_loss: 0.3244
Epoch 25/100
24/24             4s 150ms/step -
accuracy: 0.8965 - loss: 0.2996 - val_accuracy: 0.8857 - val_loss: 0.3218
Epoch 26/100
24/24             4s 159ms/step -
accuracy: 0.8976 - loss: 0.2965 - val_accuracy: 0.8866 - val_loss: 0.3195
Epoch 27/100
24/24             4s 148ms/step -
accuracy: 0.8985 - loss: 0.2935 - val_accuracy: 0.8874 - val_loss: 0.3173
Epoch 28/100
24/24             4s 158ms/step -
accuracy: 0.8995 - loss: 0.2907 - val_accuracy: 0.8887 - val_loss: 0.3152
Epoch 29/100
24/24             4s 175ms/step -
accuracy: 0.9003 - loss: 0.2880 - val_accuracy: 0.8892 - val_loss: 0.3132
Epoch 30/100
24/24             4s 162ms/step -
accuracy: 0.9010 - loss: 0.2854 - val_accuracy: 0.8904 - val_loss: 0.3114
Epoch 31/100
24/24             4s 184ms/step -
accuracy: 0.9016 - loss: 0.2829 - val_accuracy: 0.8908 - val_loss: 0.3096
Epoch 32/100
24/24             4s 154ms/step -
accuracy: 0.9020 - loss: 0.2806 - val_accuracy: 0.8911 - val_loss: 0.3080
Epoch 33/100
```

```
24/24             4s 167ms/step -
accuracy: 0.9028 - loss: 0.2783 - val_accuracy: 0.8916 - val_loss: 0.3064
Epoch 34/100
24/24             4s 157ms/step -
accuracy: 0.9037 - loss: 0.2761 - val_accuracy: 0.8917 - val_loss: 0.3049
Epoch 35/100
24/24             4s 157ms/step -
accuracy: 0.9049 - loss: 0.2740 - val_accuracy: 0.8922 - val_loss: 0.3035
Epoch 36/100
24/24             4s 153ms/step -
accuracy: 0.9055 - loss: 0.2720 - val_accuracy: 0.8927 - val_loss: 0.3022
Epoch 37/100
24/24             4s 148ms/step -
accuracy: 0.9063 - loss: 0.2700 - val_accuracy: 0.8932 - val_loss: 0.3009
Epoch 38/100
24/24             4s 148ms/step -
accuracy: 0.9072 - loss: 0.2681 - val_accuracy: 0.8929 - val_loss: 0.2997
Epoch 39/100
24/24             4s 150ms/step -
accuracy: 0.9077 - loss: 0.2662 - val_accuracy: 0.8932 - val_loss: 0.2985
Epoch 40/100
24/24             4s 156ms/step -
accuracy: 0.9084 - loss: 0.2644 - val_accuracy: 0.8936 - val_loss: 0.2974
Epoch 41/100
24/24             4s 152ms/step -
accuracy: 0.9089 - loss: 0.2627 - val_accuracy: 0.8939 - val_loss: 0.2963
Epoch 42/100
24/24             4s 158ms/step -
accuracy: 0.9095 - loss: 0.2610 - val_accuracy: 0.8938 - val_loss: 0.2952
Epoch 43/100
24/24             4s 154ms/step -
accuracy: 0.9102 - loss: 0.2593 - val_accuracy: 0.8940 - val_loss: 0.2942
Epoch 44/100
24/24             4s 160ms/step -
accuracy: 0.9108 - loss: 0.2577 - val_accuracy: 0.8946 - val_loss: 0.2933
Epoch 45/100
24/24             4s 148ms/step -
accuracy: 0.9113 - loss: 0.2562 - val_accuracy: 0.8948 - val_loss: 0.2924
Epoch 46/100
24/24             4s 150ms/step -
accuracy: 0.9118 - loss: 0.2547 - val_accuracy: 0.8951 - val_loss: 0.2915
Epoch 47/100
24/24             4s 150ms/step -
accuracy: 0.9126 - loss: 0.2532 - val_accuracy: 0.8957 - val_loss: 0.2906
Epoch 48/100
24/24             4s 159ms/step -
accuracy: 0.9128 - loss: 0.2517 - val_accuracy: 0.8959 - val_loss: 0.2898
Epoch 49/100
```

```
24/24            4s 153ms/step -
accuracy: 0.9135 - loss: 0.2503 - val_accuracy: 0.8960 - val_loss: 0.2890
Epoch 50/100
24/24            4s 154ms/step -
accuracy: 0.9138 - loss: 0.2489 - val_accuracy: 0.8962 - val_loss: 0.2883
Epoch 51/100
24/24            4s 156ms/step -
accuracy: 0.9142 - loss: 0.2476 - val_accuracy: 0.8959 - val_loss: 0.2876
Epoch 52/100
24/24            4s 149ms/step -
accuracy: 0.9147 - loss: 0.2462 - val_accuracy: 0.8958 - val_loss: 0.2869
Epoch 53/100
24/24            4s 149ms/step -
accuracy: 0.9152 - loss: 0.2450 - val_accuracy: 0.8961 - val_loss: 0.2862
Epoch 54/100
24/24            4s 155ms/step -
accuracy: 0.9158 - loss: 0.2437 - val_accuracy: 0.8966 - val_loss: 0.2856
Epoch 55/100
24/24            4s 145ms/step -
accuracy: 0.9162 - loss: 0.2424 - val_accuracy: 0.8967 - val_loss: 0.2850
Epoch 56/100
24/24            4s 147ms/step -
accuracy: 0.9167 - loss: 0.2412 - val_accuracy: 0.8967 - val_loss: 0.2844
Epoch 57/100
24/24            4s 158ms/step -
accuracy: 0.9174 - loss: 0.2400 - val_accuracy: 0.8966 - val_loss: 0.2838
Epoch 58/100
24/24            4s 149ms/step -
accuracy: 0.9177 - loss: 0.2388 - val_accuracy: 0.8968 - val_loss: 0.2833
Epoch 59/100
24/24            4s 149ms/step -
accuracy: 0.9181 - loss: 0.2377 - val_accuracy: 0.8972 - val_loss: 0.2828
Epoch 60/100
24/24            4s 148ms/step -
accuracy: 0.9187 - loss: 0.2366 - val_accuracy: 0.8973 - val_loss: 0.2823
Epoch 61/100
24/24            4s 158ms/step -
accuracy: 0.9190 - loss: 0.2354 - val_accuracy: 0.8973 - val_loss: 0.2818
Epoch 62/100
24/24            4s 150ms/step -
accuracy: 0.9192 - loss: 0.2343 - val_accuracy: 0.8978 - val_loss: 0.2812
Epoch 63/100
24/24            4s 152ms/step -
accuracy: 0.9196 - loss: 0.2332 - val_accuracy: 0.8979 - val_loss: 0.2807
Epoch 64/100
24/24            4s 149ms/step -
accuracy: 0.9200 - loss: 0.2321 - val_accuracy: 0.8980 - val_loss: 0.2803
Epoch 65/100
```

```
24/24           4s 150ms/step -
accuracy: 0.9205 - loss: 0.2311 - val_accuracy: 0.8982 - val_loss: 0.2798
Epoch 66/100
24/24           4s 152ms/step -
accuracy: 0.9210 - loss: 0.2300 - val_accuracy: 0.8982 - val_loss: 0.2793
Epoch 67/100
24/24           4s 149ms/step -
accuracy: 0.9213 - loss: 0.2290 - val_accuracy: 0.8985 - val_loss: 0.2788
Epoch 68/100
24/24           4s 147ms/step -
accuracy: 0.9220 - loss: 0.2279 - val_accuracy: 0.8990 - val_loss: 0.2784
Epoch 69/100
24/24           4s 146ms/step -
accuracy: 0.9225 - loss: 0.2269 - val_accuracy: 0.8989 - val_loss: 0.2779
Epoch 70/100
24/24           4s 187ms/step -
accuracy: 0.9227 - loss: 0.2259 - val_accuracy: 0.8989 - val_loss: 0.2774
Epoch 71/100
24/24           4s 153ms/step -
accuracy: 0.9232 - loss: 0.2249 - val_accuracy: 0.8989 - val_loss: 0.2770
Epoch 72/100
24/24           4s 153ms/step -
accuracy: 0.9235 - loss: 0.2239 - val_accuracy: 0.8992 - val_loss: 0.2766
Epoch 73/100
24/24           4s 156ms/step -
accuracy: 0.9241 - loss: 0.2230 - val_accuracy: 0.8993 - val_loss: 0.2761
Epoch 74/100
24/24           4s 153ms/step -
accuracy: 0.9243 - loss: 0.2220 - val_accuracy: 0.8996 - val_loss: 0.2757
Epoch 75/100
24/24           4s 154ms/step -
accuracy: 0.9245 - loss: 0.2210 - val_accuracy: 0.8995 - val_loss: 0.2753
Epoch 76/100
24/24           4s 155ms/step -
accuracy: 0.9250 - loss: 0.2201 - val_accuracy: 0.8998 - val_loss: 0.2749
Epoch 77/100
24/24           4s 151ms/step -
accuracy: 0.9253 - loss: 0.2192 - val_accuracy: 0.8997 - val_loss: 0.2746
Epoch 78/100
24/24           4s 152ms/step -
accuracy: 0.9257 - loss: 0.2183 - val_accuracy: 0.9003 - val_loss: 0.2742
Epoch 79/100
24/24           4s 153ms/step -
accuracy: 0.9262 - loss: 0.2174 - val_accuracy: 0.9001 - val_loss: 0.2738
Epoch 80/100
24/24           4s 153ms/step -
accuracy: 0.9263 - loss: 0.2165 - val_accuracy: 0.9004 - val_loss: 0.2735
Epoch 81/100
```

```
24/24           4s 152ms/step -
accuracy: 0.9267 - loss: 0.2156 - val_accuracy: 0.9007 - val_loss: 0.2732
Epoch 82/100
24/24           4s 153ms/step -
accuracy: 0.9269 - loss: 0.2148 - val_accuracy: 0.9012 - val_loss: 0.2729
Epoch 83/100
24/24           4s 154ms/step -
accuracy: 0.9271 - loss: 0.2139 - val_accuracy: 0.9013 - val_loss: 0.2726
Epoch 84/100
24/24           4s 152ms/step -
accuracy: 0.9272 - loss: 0.2131 - val_accuracy: 0.9014 - val_loss: 0.2723
Epoch 85/100
24/24           4s 157ms/step -
accuracy: 0.9275 - loss: 0.2123 - val_accuracy: 0.9018 - val_loss: 0.2721
Epoch 86/100
24/24           4s 152ms/step -
accuracy: 0.9278 - loss: 0.2115 - val_accuracy: 0.9020 - val_loss: 0.2718
Epoch 87/100
24/24           4s 156ms/step -
accuracy: 0.9282 - loss: 0.2107 - val_accuracy: 0.9020 - val_loss: 0.2716
Epoch 88/100
24/24           4s 154ms/step -
accuracy: 0.9283 - loss: 0.2099 - val_accuracy: 0.9023 - val_loss: 0.2713
Epoch 89/100
24/24           4s 152ms/step -
accuracy: 0.9289 - loss: 0.2091 - val_accuracy: 0.9023 - val_loss: 0.2711
Epoch 90/100
24/24           4s 163ms/step -
accuracy: 0.9293 - loss: 0.2083 - val_accuracy: 0.9024 - val_loss: 0.2709
Epoch 91/100
24/24           5s 203ms/step -
accuracy: 0.9295 - loss: 0.2076 - val_accuracy: 0.9025 - val_loss: 0.2707
Epoch 92/100
24/24           4s 173ms/step -
accuracy: 0.9297 - loss: 0.2068 - val_accuracy: 0.9027 - val_loss: 0.2705
Epoch 93/100
24/24           4s 159ms/step -
accuracy: 0.9303 - loss: 0.2060 - val_accuracy: 0.9029 - val_loss: 0.2703
Epoch 94/100
24/24           4s 159ms/step -
accuracy: 0.9308 - loss: 0.2053 - val_accuracy: 0.9030 - val_loss: 0.2701
Epoch 95/100
24/24           4s 152ms/step -
accuracy: 0.9309 - loss: 0.2046 - val_accuracy: 0.9034 - val_loss: 0.2699
Epoch 96/100
24/24           4s 155ms/step -
accuracy: 0.9312 - loss: 0.2038 - val_accuracy: 0.9038 - val_loss: 0.2698
Epoch 97/100
```

```
24/24                 4s 156ms/step -
accuracy: 0.9312 - loss: 0.2031 - val_accuracy: 0.9039 - val_loss: 0.2696
Epoch 98/100
24/24                 4s 174ms/step -
accuracy: 0.9314 - loss: 0.2024 - val_accuracy: 0.9039 - val_loss: 0.2694
Epoch 99/100
24/24                 4s 172ms/step -
accuracy: 0.9317 - loss: 0.2017 - val_accuracy: 0.9042 - val_loss: 0.2693
Epoch 100/100
24/24                 4s 176ms/step -
accuracy: 0.9321 - loss: 0.2010 - val_accuracy: 0.9043 - val_loss: 0.2691
```

[31]:

```
# Evaluation of the model on the validation set
scores6 = model6.evaluate(X_val_4D, y_val)

print(f"Accuracy for Model: {round(scores6[1], 4)},Loss for Model:␣
 ↪{round(scores6[0], 4)}")

print("\n")
```

```
375/375               2s 5ms/step -
accuracy: 0.9027 - loss: 0.2692
Accuracy for Model: 0.9043,Loss for Model: 0.2692
```

Model 7: Adding Convolution, Pooling, and Fully Connected Layer with a hidden layer (100 Nodes)

[32]:

```
model7 = Sequential([
    Input(shape=X_train_4D.shape[1:]),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(100, activation='relu'),
    Dense(num_classes, activation='softmax')
])

# Compile the model
model7.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
print(model7.summary())
```

```
Model: "sequential_8"



 Layer (type)                          Output Shape                          ␣
 ↪Param #
 conv2d_1 (Conv2D)                    (None, 26, 26, 32)                    ␣
↪320

 max_pooling2d_1 (MaxPooling2D)       (None, 13, 13, 32)                    ␣
↪  0

 flatten_7 (Flatten)                  (None, 5408)                          ␣
↪  0

 dense_22 (Dense)                     (None, 100)                         ␣
↪540,900

 dense_23 (Dense)                     (None, 10)                           ␣
↪1,010



 Total params: 542,230 (2.07 MB)


 Trainable params: 542,230 (2.07 MB)


 Non-trainable params: 0 (0.00 B)


None
```

[33]:

```python
# Fit the model
history7 = model7.fit(
    X_train_4D, y_train, validation_data=(X_val_4D, y_val), epochs=100,␣
 ↪batch_size=2048,
    callbacks=[EarlyStopping(monitor='val_accuracy', patience=5)]
)
```

```
Epoch 1/100
24/24                 7s 202ms/step -
accuracy: 0.5619 - loss: 1.4860 - val_accuracy: 0.7822 - val_loss: 0.6049
Epoch 2/100
24/24                 5s 188ms/step -
accuracy: 0.7997 - loss: 0.5637 - val_accuracy: 0.8307 - val_loss: 0.4818
Epoch 3/100
24/24                 5s 196ms/step -
accuracy: 0.8411 - loss: 0.4618 - val_accuracy: 0.8477 - val_loss: 0.4290
Epoch 4/100
24/24                 5s 200ms/step -
accuracy: 0.8554 - loss: 0.4183 - val_accuracy: 0.8562 - val_loss: 0.4070
Epoch 5/100
```

```
24/24             5s 193ms/step -
accuracy: 0.8636 - loss: 0.3931 - val_accuracy: 0.8654 - val_loss: 0.3856
Epoch 6/100
24/24             5s 193ms/step -
accuracy: 0.8731 - loss: 0.3684 - val_accuracy: 0.8709 - val_loss: 0.3710
Epoch 7/100
24/24             5s 187ms/step -
accuracy: 0.8792 - loss: 0.3495 - val_accuracy: 0.8754 - val_loss: 0.3564
Epoch 8/100
24/24             5s 187ms/step -
accuracy: 0.8840 - loss: 0.3349 - val_accuracy: 0.8784 - val_loss: 0.3468
Epoch 9/100
24/24             5s 186ms/step -
accuracy: 0.8881 - loss: 0.3242 - val_accuracy: 0.8810 - val_loss: 0.3391
Epoch 10/100
24/24             5s 186ms/step -
accuracy: 0.8911 - loss: 0.3148 - val_accuracy: 0.8838 - val_loss: 0.3319
Epoch 11/100
24/24             5s 194ms/step -
accuracy: 0.8938 - loss: 0.3054 - val_accuracy: 0.8857 - val_loss: 0.3247
Epoch 12/100
24/24             5s 201ms/step -
accuracy: 0.8968 - loss: 0.2966 - val_accuracy: 0.8875 - val_loss: 0.3183
Epoch 13/100
24/24             4s 184ms/step -
accuracy: 0.8995 - loss: 0.2889 - val_accuracy: 0.8893 - val_loss: 0.3131
Epoch 14/100
24/24             5s 207ms/step -
accuracy: 0.9014 - loss: 0.2823 - val_accuracy: 0.8909 - val_loss: 0.3086
Epoch 15/100
24/24             5s 187ms/step -
accuracy: 0.9032 - loss: 0.2762 - val_accuracy: 0.8920 - val_loss: 0.3045
Epoch 16/100
24/24             5s 199ms/step -
accuracy: 0.9053 - loss: 0.2703 - val_accuracy: 0.8939 - val_loss: 0.3005
Epoch 17/100
24/24             5s 209ms/step -
accuracy: 0.9070 - loss: 0.2646 - val_accuracy: 0.8942 - val_loss: 0.2968
Epoch 18/100
24/24             5s 222ms/step -
accuracy: 0.9088 - loss: 0.2594 - val_accuracy: 0.8971 - val_loss: 0.2916
Epoch 19/100
24/24             5s 218ms/step -
accuracy: 0.9105 - loss: 0.2540 - val_accuracy: 0.8964 - val_loss: 0.2907
Epoch 20/100
24/24             5s 214ms/step -
accuracy: 0.9120 - loss: 0.2496 - val_accuracy: 0.8972 - val_loss: 0.2889
Epoch 21/100
```

```
24/24               5s 193ms/step -
accuracy: 0.9135 - loss: 0.2454 - val_accuracy: 0.8968 - val_loss: 0.2867
Epoch 22/100
24/24               5s 201ms/step -
accuracy: 0.9148 - loss: 0.2411 - val_accuracy: 0.8971 - val_loss: 0.2850
Epoch 23/100
24/24               5s 188ms/step -
accuracy: 0.9163 - loss: 0.2372 - val_accuracy: 0.8985 - val_loss: 0.2830
Epoch 24/100
24/24               5s 188ms/step -
accuracy: 0.9181 - loss: 0.2331 - val_accuracy: 0.8994 - val_loss: 0.2813
Epoch 25/100
24/24               5s 194ms/step -
accuracy: 0.9201 - loss: 0.2293 - val_accuracy: 0.8999 - val_loss: 0.2800
Epoch 26/100
24/24               5s 193ms/step -
accuracy: 0.9216 - loss: 0.2258 - val_accuracy: 0.9014 - val_loss: 0.2792
Epoch 27/100
24/24               5s 189ms/step -
accuracy: 0.9232 - loss: 0.2226 - val_accuracy: 0.9011 - val_loss: 0.2784
Epoch 28/100
24/24               5s 204ms/step -
accuracy: 0.9239 - loss: 0.2196 - val_accuracy: 0.9022 - val_loss: 0.2784
Epoch 29/100
24/24               5s 212ms/step -
accuracy: 0.9248 - loss: 0.2170 - val_accuracy: 0.9013 - val_loss: 0.2786
Epoch 30/100
24/24               5s 191ms/step -
accuracy: 0.9261 - loss: 0.2147 - val_accuracy: 0.9018 - val_loss: 0.2785
Epoch 31/100
24/24               5s 190ms/step -
accuracy: 0.9271 - loss: 0.2127 - val_accuracy: 0.9017 - val_loss: 0.2753
Epoch 32/100
24/24               5s 190ms/step -
accuracy: 0.9274 - loss: 0.2110 - val_accuracy: 0.9038 - val_loss: 0.2683
Epoch 33/100
24/24               5s 189ms/step -
accuracy: 0.9285 - loss: 0.2076 - val_accuracy: 0.9046 - val_loss: 0.2640
Epoch 34/100
24/24               5s 208ms/step -
accuracy: 0.9307 - loss: 0.2030 - val_accuracy: 0.9048 - val_loss: 0.2636
Epoch 35/100
24/24               5s 201ms/step -
accuracy: 0.9326 - loss: 0.1983 - val_accuracy: 0.9046 - val_loss: 0.2630
Epoch 36/100
24/24               5s 200ms/step -
accuracy: 0.9334 - loss: 0.1950 - val_accuracy: 0.9049 - val_loss: 0.2625
Epoch 37/100
```

```
24/24           5s 196ms/step -
accuracy: 0.9349 - loss: 0.1923 - val_accuracy: 0.9057 - val_loss: 0.2620
Epoch 38/100
24/24           5s 216ms/step -
accuracy: 0.9362 - loss: 0.1899 - val_accuracy: 0.9059 - val_loss: 0.2617
Epoch 39/100
24/24           5s 194ms/step -
accuracy: 0.9367 - loss: 0.1875 - val_accuracy: 0.9061 - val_loss: 0.2612
Epoch 40/100
24/24           5s 189ms/step -
accuracy: 0.9378 - loss: 0.1849 - val_accuracy: 0.9062 - val_loss: 0.2608
Epoch 41/100
24/24           5s 190ms/step -
accuracy: 0.9392 - loss: 0.1823 - val_accuracy: 0.9066 - val_loss: 0.2603
Epoch 42/100
24/24           5s 197ms/step -
accuracy: 0.9401 - loss: 0.1798 - val_accuracy: 0.9068 - val_loss: 0.2600
Epoch 43/100
24/24           5s 223ms/step -
accuracy: 0.9407 - loss: 0.1773 - val_accuracy: 0.9073 - val_loss: 0.2597
Epoch 44/100
24/24           5s 225ms/step -
accuracy: 0.9415 - loss: 0.1749 - val_accuracy: 0.9076 - val_loss: 0.2594
Epoch 45/100
24/24           6s 239ms/step -
accuracy: 0.9426 - loss: 0.1726 - val_accuracy: 0.9081 - val_loss: 0.2589
Epoch 46/100
24/24           6s 229ms/step -
accuracy: 0.9432 - loss: 0.1702 - val_accuracy: 0.9082 - val_loss: 0.2588
Epoch 47/100
24/24           5s 186ms/step -
accuracy: 0.9440 - loss: 0.1680 - val_accuracy: 0.9087 - val_loss: 0.2588
Epoch 48/100
24/24           5s 209ms/step -
accuracy: 0.9444 - loss: 0.1658 - val_accuracy: 0.9088 - val_loss: 0.2586
Epoch 49/100
24/24           5s 219ms/step -
accuracy: 0.9454 - loss: 0.1635 - val_accuracy: 0.9087 - val_loss: 0.2584
Epoch 50/100
24/24           5s 216ms/step -
accuracy: 0.9461 - loss: 0.1614 - val_accuracy: 0.9095 - val_loss: 0.2586
Epoch 51/100
24/24           5s 193ms/step -
accuracy: 0.9469 - loss: 0.1594 - val_accuracy: 0.9106 - val_loss: 0.2587
Epoch 52/100
24/24           5s 191ms/step -
accuracy: 0.9478 - loss: 0.1573 - val_accuracy: 0.9107 - val_loss: 0.2587
Epoch 53/100
```

```
24/24                5s 199ms/step -
accuracy: 0.9484 - loss: 0.1553 - val_accuracy: 0.9101 - val_loss: 0.2589
Epoch 54/100
24/24                5s 209ms/step -
accuracy: 0.9492 - loss: 0.1533 - val_accuracy: 0.9103 - val_loss: 0.2590
Epoch 55/100
24/24                5s 200ms/step -
accuracy: 0.9498 - loss: 0.1513 - val_accuracy: 0.9102 - val_loss: 0.2587
Epoch 56/100
24/24                5s 201ms/step -
accuracy: 0.9507 - loss: 0.1491 - val_accuracy: 0.9103 - val_loss: 0.2590
Epoch 57/100
24/24                5s 193ms/step -
accuracy: 0.9517 - loss: 0.1470 - val_accuracy: 0.9101 - val_loss: 0.2588
```

[34]:

```
# Evaluation of the model on the validation set
scores7 = model7.evaluate(X_val_4D, y_val)
print(f"Accuracy for Model: {round(scores7[1], 4)},Loss for Model:␣
  ↪{round(scores7[0], 4)}")
print("\n")
```

```
375/375              2s 5ms/step -
accuracy: 0.9120 - loss: 0.2546
Accuracy for Model: 0.9101,Loss for Model: 0.2591
```

Model 8: Added 2 Convolution layers, and 2 Pooling layers, and Fully Connected Layer with a hidden layer (256 Nodes)

[35]:

```
model8 = Sequential([
    Input(shape=X_train_4D.shape[1:]),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(100, activation='relu'),
    Dense(num_classes, activation='softmax')
])

# Compile the model
model8.compile(loss='categorical_crossentropy', optimizer='adam',␣
  ↪metrics=['accuracy'])
print(model8.summary())
```

```
Model: "sequential_9"
```

```
 Layer (type)                        Output Shape                         ␣
↪Param #

 conv2d_2 (Conv2D)                   (None, 26, 26, 32)                     ␣
↪320

 max_pooling2d_2 (MaxPooling2D)      (None, 13, 13, 32)                     ␣
↪  0

 conv2d_3 (Conv2D)                   (None, 11, 11, 32)                   ␣
↪9,248

 max_pooling2d_3 (MaxPooling2D)      (None, 5, 5, 32)                       ␣
↪  0

 flatten_8 (Flatten)                 (None, 800)                            ␣
↪  0

 dense_24 (Dense)                    (None, 100)                         ␣
↪80,100

 dense_25 (Dense)                    (None, 10)                           ␣
↪1,010
```

```
 Total params: 90,678 (354.21 KB)

 Trainable params: 90,678 (354.21 KB)

 Non-trainable params: 0 (0.00 B)

None
```

[36]:

```python
# Fit the model
history8 = model8.fit(
    X_train_4D, y_train, validation_data=(X_val_4D, y_val), epochs=100,␣
↪batch_size=2048,
    callbacks=[EarlyStopping(monitor='val_accuracy', patience=5)]
)
```

```
Epoch 1/100
24/24                 9s 296ms/step -
accuracy: 0.3796 - loss: 2.0139 - val_accuracy: 0.6952 - val_loss: 0.8466
Epoch 2/100
```

```
24/24             7s 274ms/step -
accuracy: 0.7109 - loss: 0.7945 - val_accuracy: 0.7453 - val_loss: 0.6768
Epoch 3/100
24/24             6s 267ms/step -
accuracy: 0.7612 - loss: 0.6429 - val_accuracy: 0.7827 - val_loss: 0.5866
Epoch 4/100
24/24             6s 264ms/step -
accuracy: 0.7915 - loss: 0.5646 - val_accuracy: 0.8036 - val_loss: 0.5296
Epoch 5/100
24/24             6s 264ms/step -
accuracy: 0.8122 - loss: 0.5141 - val_accuracy: 0.8172 - val_loss: 0.4984
Epoch 6/100
24/24             6s 263ms/step -
accuracy: 0.8299 - loss: 0.4767 - val_accuracy: 0.8320 - val_loss: 0.4683
Epoch 7/100
24/24             6s 263ms/step -
accuracy: 0.8408 - loss: 0.4517 - val_accuracy: 0.8408 - val_loss: 0.4485
Epoch 8/100
24/24             6s 266ms/step -
accuracy: 0.8470 - loss: 0.4342 - val_accuracy: 0.8453 - val_loss: 0.4330
Epoch 9/100
24/24             8s 328ms/step -
accuracy: 0.8530 - loss: 0.4185 - val_accuracy: 0.8508 - val_loss: 0.4196
Epoch 10/100
24/24             9s 358ms/step -
accuracy: 0.8574 - loss: 0.4045 - val_accuracy: 0.8561 - val_loss: 0.4070
Epoch 11/100
24/24             9s 363ms/step -
accuracy: 0.8627 - loss: 0.3921 - val_accuracy: 0.8587 - val_loss: 0.3976
Epoch 12/100
24/24             7s 296ms/step -
accuracy: 0.8665 - loss: 0.3822 - val_accuracy: 0.8623 - val_loss: 0.3890
Epoch 13/100
24/24             6s 257ms/step -
accuracy: 0.8697 - loss: 0.3731 - val_accuracy: 0.8647 - val_loss: 0.3808
Epoch 14/100
24/24             7s 304ms/step -
accuracy: 0.8727 - loss: 0.3644 - val_accuracy: 0.8681 - val_loss: 0.3724
Epoch 15/100
24/24             7s 296ms/step -
accuracy: 0.8760 - loss: 0.3561 - val_accuracy: 0.8692 - val_loss: 0.3651
Epoch 16/100
24/24             6s 262ms/step -
accuracy: 0.8788 - loss: 0.3486 - val_accuracy: 0.8711 - val_loss: 0.3587
Epoch 17/100
24/24             7s 284ms/step -
accuracy: 0.8803 - loss: 0.3423 - val_accuracy: 0.8724 - val_loss: 0.3531
Epoch 18/100
```

```
24/24            7s 309ms/step -
accuracy: 0.8815 - loss: 0.3366 - val_accuracy: 0.8748 - val_loss: 0.3473
Epoch 19/100
24/24            10s 424ms/step -
accuracy: 0.8842 - loss: 0.3306 - val_accuracy: 0.8767 - val_loss: 0.3420
Epoch 20/100
24/24            8s 327ms/step -
accuracy: 0.8849 - loss: 0.3251 - val_accuracy: 0.8788 - val_loss: 0.3374
Epoch 21/100
24/24            8s 329ms/step -
accuracy: 0.8866 - loss: 0.3202 - val_accuracy: 0.8799 - val_loss: 0.3337
Epoch 22/100
24/24            7s 300ms/step -
accuracy: 0.8881 - loss: 0.3156 - val_accuracy: 0.8805 - val_loss: 0.3298
Epoch 23/100
24/24            7s 283ms/step -
accuracy: 0.8893 - loss: 0.3109 - val_accuracy: 0.8811 - val_loss: 0.3264
Epoch 24/100
24/24            7s 304ms/step -
accuracy: 0.8910 - loss: 0.3065 - val_accuracy: 0.8817 - val_loss: 0.3234
Epoch 25/100
24/24            7s 312ms/step -
accuracy: 0.8923 - loss: 0.3027 - val_accuracy: 0.8829 - val_loss: 0.3203
Epoch 26/100
24/24            7s 299ms/step -
accuracy: 0.8937 - loss: 0.2989 - val_accuracy: 0.8835 - val_loss: 0.3177
Epoch 27/100
24/24            7s 271ms/step -
accuracy: 0.8951 - loss: 0.2953 - val_accuracy: 0.8852 - val_loss: 0.3149
Epoch 28/100
24/24            7s 283ms/step -
accuracy: 0.8961 - loss: 0.2918 - val_accuracy: 0.8867 - val_loss: 0.3125
Epoch 29/100
24/24            7s 306ms/step -
accuracy: 0.8974 - loss: 0.2884 - val_accuracy: 0.8869 - val_loss: 0.3100
Epoch 30/100
24/24            7s 311ms/step -
accuracy: 0.8989 - loss: 0.2853 - val_accuracy: 0.8878 - val_loss: 0.3078
Epoch 31/100
24/24            8s 315ms/step -
accuracy: 0.9004 - loss: 0.2823 - val_accuracy: 0.8882 - val_loss: 0.3058
Epoch 32/100
24/24            7s 293ms/step -
accuracy: 0.9010 - loss: 0.2793 - val_accuracy: 0.8895 - val_loss: 0.3034
Epoch 33/100
24/24            7s 277ms/step -
accuracy: 0.9019 - loss: 0.2765 - val_accuracy: 0.8898 - val_loss: 0.3014
Epoch 34/100
```

```
24/24           7s 275ms/step -
accuracy: 0.9035 - loss: 0.2736 - val_accuracy: 0.8905 - val_loss: 0.2995
Epoch 35/100
24/24           7s 277ms/step -
accuracy: 0.9041 - loss: 0.2707 - val_accuracy: 0.8912 - val_loss: 0.2978
Epoch 36/100
24/24           7s 304ms/step -
accuracy: 0.9053 - loss: 0.2681 - val_accuracy: 0.8915 - val_loss: 0.2959
Epoch 37/100
24/24           7s 291ms/step -
accuracy: 0.9060 - loss: 0.2654 - val_accuracy: 0.8920 - val_loss: 0.2938
Epoch 38/100
24/24           7s 275ms/step -
accuracy: 0.9067 - loss: 0.2626 - val_accuracy: 0.8934 - val_loss: 0.2925
Epoch 39/100
24/24           7s 293ms/step -
accuracy: 0.9075 - loss: 0.2603 - val_accuracy: 0.8933 - val_loss: 0.2909
Epoch 40/100
24/24           7s 305ms/step -
accuracy: 0.9086 - loss: 0.2578 - val_accuracy: 0.8932 - val_loss: 0.2902
Epoch 41/100
24/24           7s 292ms/step -
accuracy: 0.9092 - loss: 0.2559 - val_accuracy: 0.8935 - val_loss: 0.2885
Epoch 42/100
24/24           7s 293ms/step -
accuracy: 0.9098 - loss: 0.2535 - val_accuracy: 0.8947 - val_loss: 0.2869
Epoch 43/100
24/24           8s 326ms/step -
accuracy: 0.9104 - loss: 0.2512 - val_accuracy: 0.8950 - val_loss: 0.2858
Epoch 44/100
24/24           7s 273ms/step -
accuracy: 0.9111 - loss: 0.2490 - val_accuracy: 0.8956 - val_loss: 0.2846
Epoch 45/100
24/24           7s 288ms/step -
accuracy: 0.9121 - loss: 0.2470 - val_accuracy: 0.8964 - val_loss: 0.2834
Epoch 46/100
24/24           7s 277ms/step -
accuracy: 0.9130 - loss: 0.2447 - val_accuracy: 0.8968 - val_loss: 0.2821
Epoch 47/100
24/24           8s 327ms/step -
accuracy: 0.9140 - loss: 0.2425 - val_accuracy: 0.8975 - val_loss: 0.2813
Epoch 48/100
24/24           8s 340ms/step -
accuracy: 0.9147 - loss: 0.2406 - val_accuracy: 0.8983 - val_loss: 0.2802
Epoch 49/100
24/24           8s 315ms/step -
accuracy: 0.9153 - loss: 0.2386 - val_accuracy: 0.8987 - val_loss: 0.2794
Epoch 50/100
```

```
24/24             7s 276ms/step -
accuracy: 0.9160 - loss: 0.2367 - val_accuracy: 0.8986 - val_loss: 0.2786
Epoch 51/100
24/24             7s 271ms/step -
accuracy: 0.9163 - loss: 0.2349 - val_accuracy: 0.8992 - val_loss: 0.2778
Epoch 52/100
24/24             7s 269ms/step -
accuracy: 0.9169 - loss: 0.2330 - val_accuracy: 0.8994 - val_loss: 0.2769
Epoch 53/100
24/24             7s 272ms/step -
accuracy: 0.9176 - loss: 0.2312 - val_accuracy: 0.8992 - val_loss: 0.2766
Epoch 54/100
24/24             7s 294ms/step -
accuracy: 0.9184 - loss: 0.2296 - val_accuracy: 0.9003 - val_loss: 0.2760
Epoch 55/100
24/24             7s 270ms/step -
accuracy: 0.9189 - loss: 0.2280 - val_accuracy: 0.9008 - val_loss: 0.2753
Epoch 56/100
24/24             6s 269ms/step -
accuracy: 0.9194 - loss: 0.2264 - val_accuracy: 0.9014 - val_loss: 0.2745
Epoch 57/100
24/24             7s 281ms/step -
accuracy: 0.9202 - loss: 0.2247 - val_accuracy: 0.9017 - val_loss: 0.2744
Epoch 58/100
24/24             7s 270ms/step -
accuracy: 0.9206 - loss: 0.2233 - val_accuracy: 0.9012 - val_loss: 0.2732
Epoch 59/100
24/24             7s 283ms/step -
accuracy: 0.9218 - loss: 0.2213 - val_accuracy: 0.9008 - val_loss: 0.2730
Epoch 60/100
24/24             7s 286ms/step -
accuracy: 0.9224 - loss: 0.2198 - val_accuracy: 0.9012 - val_loss: 0.2725
Epoch 61/100
24/24             6s 268ms/step -
accuracy: 0.9224 - loss: 0.2182 - val_accuracy: 0.9018 - val_loss: 0.2724
Epoch 62/100
24/24             7s 285ms/step -
accuracy: 0.9223 - loss: 0.2173 - val_accuracy: 0.9015 - val_loss: 0.2730
Epoch 63/100
24/24             7s 279ms/step -
accuracy: 0.9217 - loss: 0.2168 - val_accuracy: 0.9003 - val_loss: 0.2759
Epoch 64/100
24/24             7s 291ms/step -
accuracy: 0.9205 - loss: 0.2181 - val_accuracy: 0.8957 - val_loss: 0.2846
Epoch 65/100
24/24             7s 283ms/step -
accuracy: 0.9186 - loss: 0.2218 - val_accuracy: 0.9029 - val_loss: 0.2686
Epoch 66/100
```

```
24/24               7s 286ms/step -
accuracy: 0.9242 - loss: 0.2120 - val_accuracy: 0.9016 - val_loss: 0.2673
Epoch 67/100
24/24               7s 280ms/step -
accuracy: 0.9259 - loss: 0.2073 - val_accuracy: 0.9017 - val_loss: 0.2668
Epoch 68/100
24/24               7s 287ms/step -
accuracy: 0.9263 - loss: 0.2061 - val_accuracy: 0.9019 - val_loss: 0.2666
Epoch 69/100
24/24               7s 301ms/step -
accuracy: 0.9265 - loss: 0.2046 - val_accuracy: 0.9022 - val_loss: 0.2657
Epoch 70/100
24/24               7s 277ms/step -
accuracy: 0.9273 - loss: 0.2027 - val_accuracy: 0.9020 - val_loss: 0.2659
```

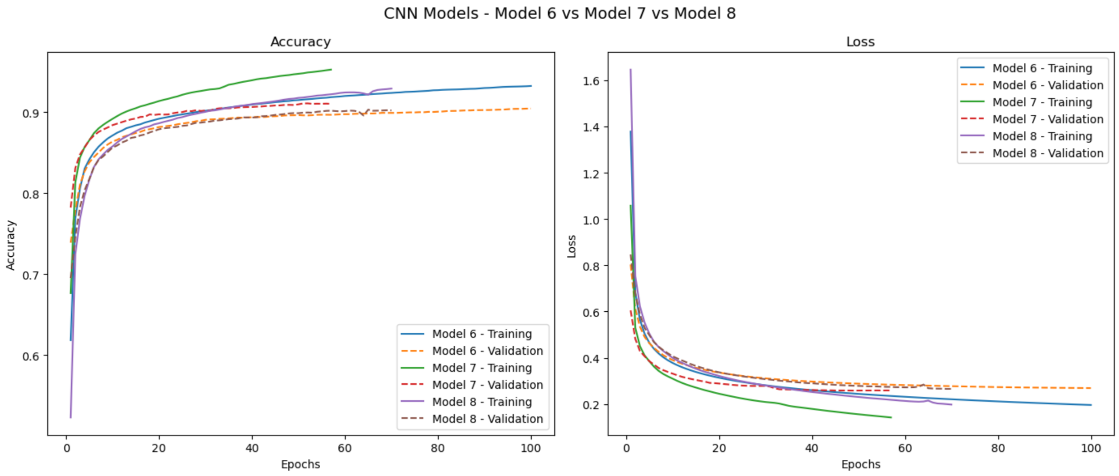
[37]:

```
# Evaluation of the model on the validation set
scores8 = model8.evaluate(X_val_4D, y_val)

print("\n")
print(f"Accuracy for Model 8: {round(scores8[1], 4)},Loss for Model 8:␣
  ↪{round(scores8[0], 4)}")
```

```
375/375               2s 6ms/step -
accuracy: 0.9026 - loss: 0.2604


Accuracy for Model 8: 0.902,Loss for Model 8: 0.2657
```

[99]:

```
plot_model_history([history6,history7,history8], ['Model 6','Model 7','Model␣
  ↪8'],'CNN Models - Model 6 vs Model 7 vs Model 8')
```



Model 6 has no middle layer, while model 7 has a middle layer with 100 nodes. Model 8 on the other hand has two convulation layers with a layer of 100 nodes. It is very clear that model 7 is by far the best on validation sets

---

0.9 6. Try to use a pre-trained network to improve accuracy.

[38]:

```
from keras.applications.resnet50 import ResNet50, preprocess_input
from keras.utils import load_img, img_to_array
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Flatten, Input
from skimage.color import gray2rgb
from skimage.transform import resize
```

[60]:

```
# Preprocess the data and convert grayscale images to RGB
X_train_resized = np.array([preprocess_input(gray2rgb(img)) for img in X_train])
X_test_resized = np.array([preprocess_input(gray2rgb(img)) for img in X_test])

# Resize the images to match the input shape expected by ResNet50
X_train_resized = np.array([resize(img, (32, 32)) for img in X_train])
X_val_resized = np.array([resize(img, (32, 32)) for img in X_val])

# Load pre-trained ResNet50 model without the top layer (include_top=False)
resnet_model = ResNet50(weights='imagenet', include_top=False, input_shape=(32,␣
 ↪32, 3))

# Freeze the layers in the pre-trained model
for layer in resnet_model.layers:
    layer.trainable = False

# Create a new model and add the pre-trained ResNet50 base
pretrained_model = Sequential()
pretrained_model.add(resnet_model)

# Add additional layers on top of the ResNet50 base
pretrained_model.add(Flatten())
pretrained_model.add(Dense(100, activation='relu'))
pretrained_model.add(Dense(10, activation='softmax'))  # FashionMNIST has 10␣
 ↪classes

# Compile the model
pretrained_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
print(pretrained_model.summary())
```

```
Model: "sequential_17"
```

```
 Layer (type)                         Output Shape                         ␣
↪Param #

 resnet50 (Functional)                ?                                 ␣
↪23,587,712

 flatten_11 (Flatten)                 ?                                 0␣
↪(unbuilt)

 dense_30 (Dense)                     ?                                 0␣
↪(unbuilt)

 dense_31 (Dense)                     ?                                 0␣
↪(unbuilt)



 Total params: 23,587,712 (89.98 MB)

 Trainable params: 0 (0.00 B)

 Non-trainable params: 23,587,712 (89.98 MB)

None
```

[44]:

```
# Train the model
history_pre = pretrained_model.fit(X_train_resized, y_train,␣
 ↪validation_data=(X_val_resized, y_val), epochs=20, batch_size=4096,␣
 ↪callbacks=[EarlyStopping(monitor='val_accuracy', patience=5)])
```

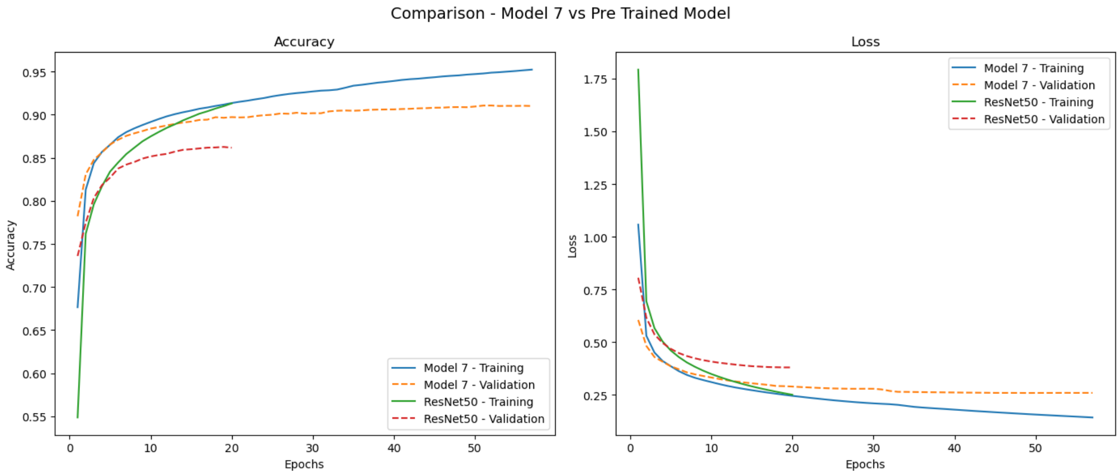
```
Epoch 1/20
12/12                 107s 8s/step -
accuracy: 0.4011 - loss: 2.8412 - val_accuracy: 0.7360 - val_loss: 0.8053
Epoch 2/20
12/12                 79s 7s/step -
accuracy: 0.7564 - loss: 0.7320 - val_accuracy: 0.7747 - val_loss: 0.6158
Epoch 3/20
12/12                 80s 7s/step -
accuracy: 0.7915 - loss: 0.5784 - val_accuracy: 0.8033 - val_loss: 0.5363
Epoch 4/20
12/12                 80s 7s/step -
accuracy: 0.8137 - loss: 0.5093 - val_accuracy: 0.8183 - val_loss: 0.4959
Epoch 5/20
12/12                 81s 7s/step -
accuracy: 0.8319 - loss: 0.4660 - val_accuracy: 0.8270 - val_loss: 0.4682
```

```
Epoch 6/20
12/12            81s 7s/step -
accuracy: 0.8424 - loss: 0.4333 - val_accuracy: 0.8375 - val_loss: 0.4480
Epoch 7/20
12/12            82s 7s/step -
accuracy: 0.8525 - loss: 0.4067 - val_accuracy: 0.8421 - val_loss: 0.4336
Epoch 8/20
12/12            89s 7s/step -
accuracy: 0.8605 - loss: 0.3853 - val_accuracy: 0.8451 - val_loss: 0.4227
Epoch 9/20
12/12            80s 7s/step -
accuracy: 0.8677 - loss: 0.3667 - val_accuracy: 0.8491 - val_loss: 0.4142
Epoch 10/20
12/12            88s 7s/step -
accuracy: 0.8737 - loss: 0.3509 - val_accuracy: 0.8515 - val_loss: 0.4075
Epoch 11/20
12/12            83s 7s/step -
accuracy: 0.8793 - loss: 0.3368 - val_accuracy: 0.8533 - val_loss: 0.4018
Epoch 12/20
12/12            80s 7s/step -
accuracy: 0.8836 - loss: 0.3240 - val_accuracy: 0.8546 - val_loss: 0.3974
Epoch 13/20
12/12            86s 7s/step -
accuracy: 0.8880 - loss: 0.3123 - val_accuracy: 0.8572 - val_loss: 0.3929
Epoch 14/20
12/12            77s 6s/step -
accuracy: 0.8928 - loss: 0.3016 - val_accuracy: 0.8593 - val_loss: 0.3891
Epoch 15/20
12/12            78s 7s/step -
accuracy: 0.8969 - loss: 0.2917 - val_accuracy: 0.8599 - val_loss: 0.3857
Epoch 16/20
12/12            81s 7s/step -
accuracy: 0.9009 - loss: 0.2825 - val_accuracy: 0.8608 - val_loss: 0.3837
Epoch 17/20
12/12            77s 7s/step -
accuracy: 0.9037 - loss: 0.2743 - val_accuracy: 0.8618 - val_loss: 0.3814
Epoch 18/20
12/12            78s 7s/step -
accuracy: 0.9071 - loss: 0.2663 - val_accuracy: 0.8620 - val_loss: 0.3805
Epoch 19/20
12/12            79s 7s/step -
accuracy: 0.9098 - loss: 0.2590 - val_accuracy: 0.8626 - val_loss: 0.3798
Epoch 20/20
12/12            80s 7s/step -
accuracy: 0.9128 - loss: 0.2520 - val_accuracy: 0.8617 - val_loss: 0.3797
```

[46]:
```
# Evaluation of the model on the validation set
scores_pre = pretrained_model.evaluate(X_val_resized, y_val)
print(f"Accuracy for ResNet50 : {round(scores_pre[1], 4)}")
```

```
375/375              31s 84ms/step -
accuracy: 0.8643 - loss: 0.3717
Accuracy for ResNet50 : 0.8617
```

[100]:
```
plot_model_history([history7,history_pre], ['Model 7','ResNet50'],'Comparison -␣
  ↪Model 7 vs Pre Trained Model')
```



Using ResNet50 as a pretrained model did not improve accuracy. One reason could be as we tried changin greyscale images to rgb which ResNet50 requires.

0.10 7. Select a final model and evaluate it on the test set. How does the test error compare to the validation error?

My best model was **Model 7: Adding Convolution, Pooling, and Fully Connected Layer with a hidden layer (100 Nodes)**

[70]:
```
print(f"Validation Accuracy: {round(scores7[1], 4)}, Validation Loss:␣
  ↪{round(scores7[0], 4)}")
print("\n")
```

```
Accuracy for Model: 0.9101,Loss for Model: 0.2591
```

[71]:
```
scores7_test = model7.evaluate(X_test_4D, y_test)
```

```
313/313              1s 3ms/step -
accuracy: 0.9104 - loss: 0.2754
```

[73]:

```
print(f"Test Accuracy: {round(scores7_test[1], 4)},Test Loss:␣
 ↪{round(scores7_test[0], 4)}")
print("\n")
```

```
Test Accuracy: 0.9084,Test Loss: 0.2681
```

[106]:

```
data = {
    "Model 7": ["Validation", "Test"],
    "Accuracy": [round(scores7[1], 4), round(scores7_test[1], 4)],
    "Loss": [round(scores7[0], 4), round(scores7_test[0], 4)]
}

# Create a dataframe from the dictionary
df = pd.DataFrame(data)

df
```

[106]:

```
      Model 7  Accuracy    Loss
0  Validation    0.9101  0.2591
1        Test    0.9084  0.2681
```

As you can see the accuracy for test dataset for our best model is pretty close to the validation set as well as the loss score. For reminder, find the details of the model below

[108]:

```
print(model7.summary())
```

```
Model: "sequential_8"


 Layer (type)                          Output Shape                      ␣
↪Param #

 conv2d_1 (Conv2D)                     (None, 26, 26, 32)                    ␣
↪320

 max_pooling2d_1 (MaxPooling2D)        (None, 13, 13, 32)                    ␣
↪  0

 flatten_7 (Flatten)                   (None, 5408)                          ␣
↪  0
```

```
 dense_22 (Dense)                      (None, 100)                          ␣
 ↪540,900

 dense_23 (Dense)                      (None, 10)                            ␣
 ↪1,010



 Total params: 1,626,692 (6.21 MB)


 Trainable params: 542,230 (2.07 MB)


 Non-trainable params: 0 (0.00 B)


 Optimizer params: 1,084,462 (4.14 MB)


None
```

```
[ ]:
```