Implementing Feed-forward neural networks with Keras and TensorFlow

-Tushar Bhagat Roll_No. 07

In [23]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Input
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.utils import plot_model
```

In [24]:
```
# Load MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess data
x_train = x_train.reshape((x_train.shape[0], 28 * 28)).astype('float32') / 255
x_test = x_test.reshape((x_test.shape[0], 28 * 28)).astype('float32') / 255

# One-hot encode labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

In [25]:
```
# Define the network architecture
model = Sequential([
    Input(shape=(28 * 28,)),  # Define the input shape here
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),
    Dense(10, activation='softmax')
])
```

In [26]:
```
# Compile the model
model.compile(optimizer=SGD(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train,
                    epochs=20,
                    batch_size=32,
                    validation_split=0.2)
```

```
Epoch 1/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7532 - loss: 0.9979 - val_accuracy: 0.9118 - val_loss: 0.31
31
Epoch 2/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9129 - loss: 0.3045 - val_accuracy: 0.9302 - val_loss: 0.24
35
Epoch 3/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9306 - loss: 0.2419 - val_accuracy: 0.9386 - val_loss: 0.21
81
Epoch 4/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9401 - loss: 0.2056 - val_accuracy: 0.9479 - val_loss: 0.18
87
Epoch 5/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9477 - loss: 0.1812 - val_accuracy: 0.9531 - val_loss: 0.16
87
Epoch 6/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9556 - loss: 0.1618 - val_accuracy: 0.9563 - val_loss: 0.15
47
Epoch 7/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9600 - loss: 0.1420 - val_accuracy: 0.9594 - val_loss: 0.14
29
Epoch 8/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9635 - loss: 0.1303 - val_accuracy: 0.9628 - val_loss: 0.13
37
Epoch 9/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9684 - loss: 0.1133 - val_accuracy: 0.9647 - val_loss: 0.12
67
Epoch 10/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9719 - loss: 0.1036 - val_accuracy: 0.9668 - val_loss: 0.11
83
Epoch 11/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9740 - loss: 0.0969 - val_accuracy: 0.9684 - val_loss: 0.11
31
Epoch 12/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9753 - loss: 0.0889 - val_accuracy: 0.9680 - val_loss: 0.11
18
Epoch 13/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9788 - loss: 0.0813 - val_accuracy: 0.9693 - val_loss: 0.10
55
Epoch 14/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9802 - loss: 0.0765 - val_accuracy: 0.9705 - val_loss: 0.10
27
Epoch 15/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9822 - loss: 0.0690 - val_accuracy: 0.9721 - val_loss: 0.09
81
Epoch 16/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9829 - loss: 0.0634 - val_accuracy: 0.9718 - val_loss: 0.09
67
Epoch 17/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9843 - loss: 0.0608 - val_accuracy: 0.9722 - val_loss: 0.09
58
Epoch 18/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9854 - loss: 0.0569 - val_accuracy: 0.9738 - val_loss: 0.09
03
Epoch 19/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9858 - loss: 0.0538 - val_accuracy: 0.9744 - val_loss: 0.08
92
Epoch 20/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9872 - loss: 0.0489 - val_accuracy: 0.9733 - val_loss: 0.09
33
```

In [27]:
```python
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 512)	401,920
dense_7 (Dense)	(None, 256)	131,328
dense_8 (Dense)	(None, 10)	2,570

Total params: 535,820 (2.04 MB)

Trainable params: 535,818 (2.04 MB)

Non-trainable params: 0 (0.00 B)

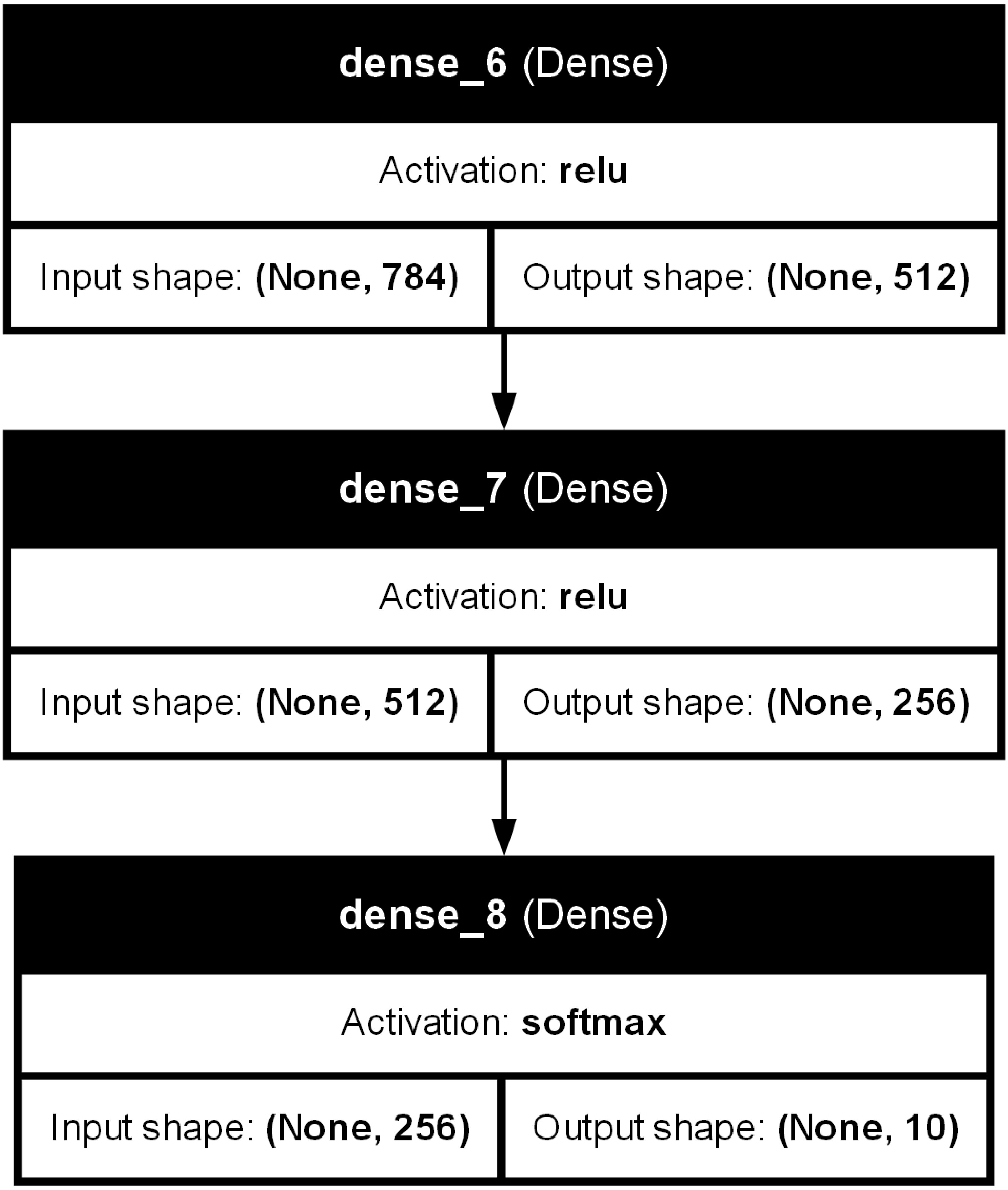
Optimizer params: 2 (12.00 B)

In [28]:
```python
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9695 - loss: 0.1002
Test Loss: 0.0863
Test Accuracy: 0.9732
```

In [29]:
```
#plotting the model
plot_model(model,to_file='C:/Users/tmbha/Downloads/DLprac2_model.png',show_shapes=True,show_layer_names=True,show_l
```

Out[29]:

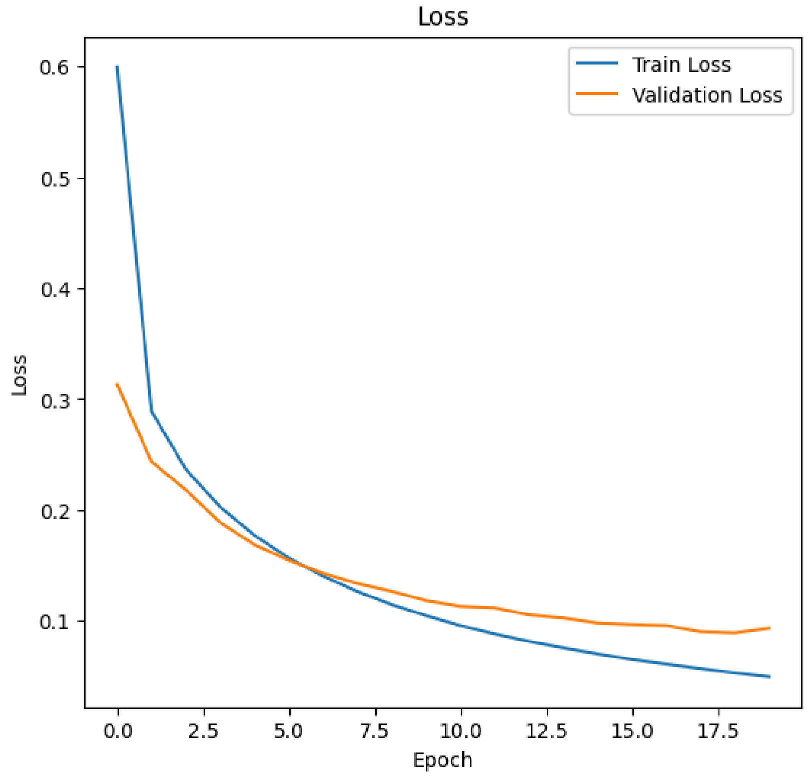


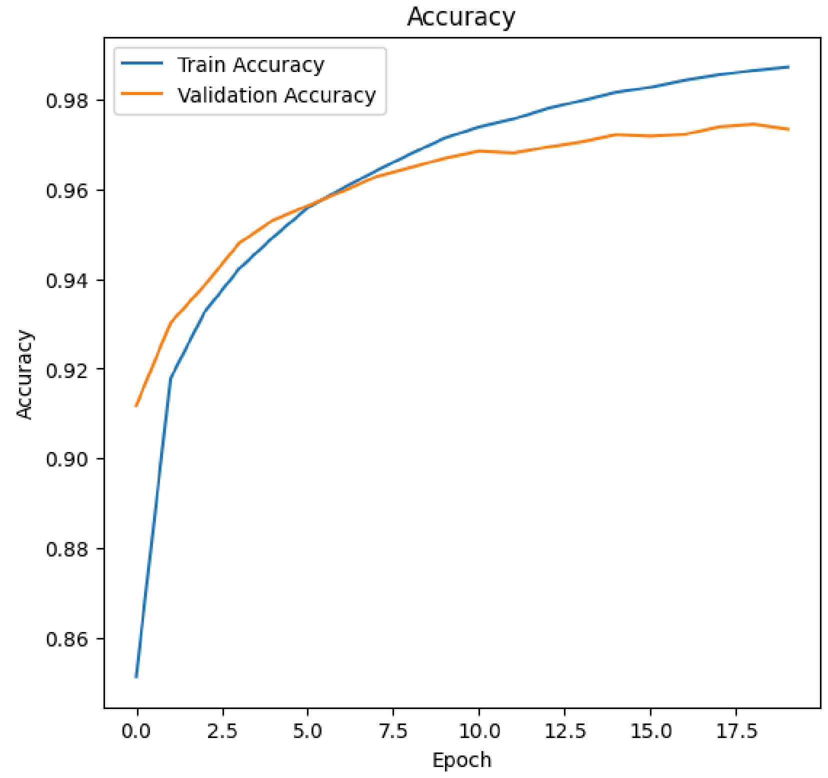
In [30]:
```
# Plotting
plt.figure(figsize=(14, 6))

# Plot training & validation loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot training & validation accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```


Loss
Train Loss
Validation Loss
Loss
Epoch
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5