# Tushar Bhagat

In [4]:
```
#importing necessary libraries

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 [5]:
```
# 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 [6]:
```
# 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 [7]:
```
# 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 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.7351 - loss: 1.0444 - val_accuracy: 0.9071 - val_loss: 0.3205
Epoch 2/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9116 - loss: 0.3111 - val_accuracy: 0.9312 - val_loss: 0.2441
Epoch 3/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9353 - loss: 0.2365 - val_accuracy: 0.9403 - val_loss: 0.2134
Epoch 4/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9411 - loss: 0.2068 - val_accuracy: 0.9463 - val_loss: 0.1875
Epoch 5/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9488 - loss: 0.1812 - val_accuracy: 0.9505 - val_loss: 0.1756
Epoch 6/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9544 - loss: 0.1655 - val_accuracy: 0.9557 - val_loss: 0.1572
Epoch 7/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9584 - loss: 0.1473 - val_accuracy: 0.9588 - val_loss: 0.1453
Epoch 8/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9626 - loss: 0.1277 - val_accuracy: 0.9613 - val_loss: 0.1353
Epoch 9/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9658 - loss: 0.1213 - val_accuracy: 0.9650 - val_loss: 0.1304
Epoch 10/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9694 - loss: 0.1086 - val_accuracy: 0.9655 - val_loss: 0.1228
Epoch 11/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9722 - loss: 0.0987 - val_accuracy: 0.9671 - val_loss: 0.1174
Epoch 12/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9749 - loss: 0.0924 - val_accuracy: 0.9691 - val_loss: 0.1123
Epoch 13/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9754 - loss: 0.0864 - val_accuracy: 0.9684 - val_loss: 0.1074
Epoch 14/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9788 - loss: 0.0780 - val_accuracy: 0.9698 - val_loss: 0.1059
Epoch 15/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9803 - loss: 0.0745 - val_accuracy: 0.9706 - val_loss: 0.1001
Epoch 16/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9813 - loss: 0.0682 - val_accuracy: 0.9712 - val_loss: 0.0999
Epoch 17/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9842 - loss: 0.0616 - val_accuracy: 0.9718 - val_loss: 0.0952
Epoch 18/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9839 - loss: 0.0594 - val_accuracy: 0.9707 - val_loss: 0.0965
Epoch 19/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9856 - loss: 0.0560 - val_accuracy: 0.9718 - val_loss: 0.0934
Epoch 20/20
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9862 - loss: 0.0517 - val_accuracy: 0.9732 - val_loss: 0.0892
```

In [8]:
```
model.summary()
```

Model: "sequential_1"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_3 (Dense) | (None, 512) | 401,920 |
| dense_4 (Dense) | (None, 256) | 131,328 |
| dense_5 (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 [9]:

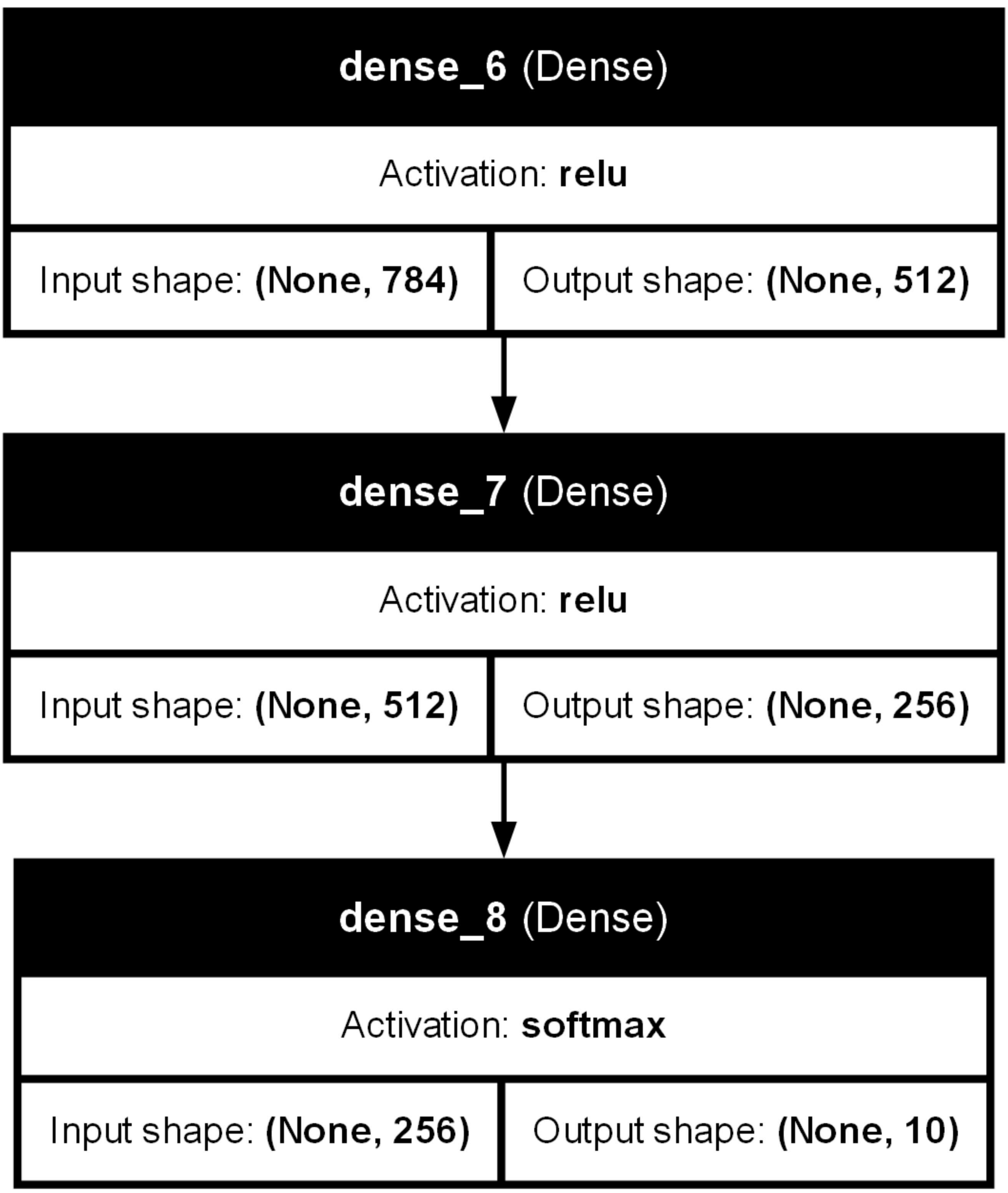
```
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9727 - loss: 0.0970
Test Loss: 0.0831
Test Accuracy: 0.9764
```

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_laye
```

Out[29]:




In [10]:

```
# 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()
```