Build the Image classification model by dividing the model into the following fourstages:

- a. Loading and preprocessing the image data - b. Defining the model's architecture - c. Training the model - d. Estimating the model's performance

-Tushar Bhagat Roll_No. 07

In [4]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.utils import plot_model
```

In [5]:
```
# Load the CIFAR-10 data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Preprocess the data
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

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

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 103s 1us/step
```

In [6]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

def create_model():
    model = Sequential([
        Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(64, kernel_size=(3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(128, kernel_size=(3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(10, activation='softmax')
    ])
    return model
```

In [7]:
```
# Create the model
model = create_model()

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

# Train the model
history = model.fit(x_train, y_train,
                    epochs=20,
                    batch_size=64,
                    validation_split=0.2)
```

```
Epoch 1/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 37s 51ms/step - accuracy: 0.2580 - loss: 1.9782 - val_accuracy: 0.4780 - val_loss: 1.41
91
Epoch 2/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 33s 53ms/step - accuracy: 0.4831 - loss: 1.4366 - val_accuracy: 0.5689 - val_loss: 1.21
07
Epoch 3/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 42s 54ms/step - accuracy: 0.5467 - loss: 1.2563 - val_accuracy: 0.6056 - val_loss: 1.12
18
Epoch 4/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 31s 50ms/step - accuracy: 0.5933 - loss: 1.1406 - val_accuracy: 0.6360 - val_loss: 1.04
51
Epoch 5/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 29s 46ms/step - accuracy: 0.6275 - loss: 1.0587 - val_accuracy: 0.6447 - val_loss: 1.01
42
Epoch 6/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 28s 45ms/step - accuracy: 0.6466 - loss: 0.9973 - val_accuracy: 0.6710 - val_loss: 0.94
70
Epoch 7/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 30s 48ms/step - accuracy: 0.6750 - loss: 0.9303 - val_accuracy: 0.6819 - val_loss: 0.90
68
Epoch 8/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 33s 52ms/step - accuracy: 0.6898 - loss: 0.8837 - val_accuracy: 0.6871 - val_loss: 0.90
91
Epoch 9/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 32s 51ms/step - accuracy: 0.7073 - loss: 0.8287 - val_accuracy: 0.6939 - val_loss: 0.89
15
Epoch 10/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 33s 53ms/step - accuracy: 0.7202 - loss: 0.7991 - val_accuracy: 0.6978 - val_loss: 0.87
49
Epoch 11/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 32s 51ms/step - accuracy: 0.7332 - loss: 0.7657 - val_accuracy: 0.6989 - val_loss: 0.87
37
Epoch 12/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 32s 51ms/step - accuracy: 0.7453 - loss: 0.7298 - val_accuracy: 0.6933 - val_loss: 0.89
70
Epoch 13/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 31s 50ms/step - accuracy: 0.7487 - loss: 0.7095 - val_accuracy: 0.7137 - val_loss: 0.85
04
Epoch 14/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 32s 52ms/step - accuracy: 0.7656 - loss: 0.6690 - val_accuracy: 0.7059 - val_loss: 0.88
46
Epoch 15/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 33s 53ms/step - accuracy: 0.7713 - loss: 0.6431 - val_accuracy: 0.7137 - val_loss: 0.86
36
Epoch 16/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 31s 50ms/step - accuracy: 0.7855 - loss: 0.6172 - val_accuracy: 0.7157 - val_loss: 0.86
13
Epoch 17/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 32s 50ms/step - accuracy: 0.7930 - loss: 0.5888 - val_accuracy: 0.7148 - val_loss: 0.88
66
Epoch 18/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 32s 51ms/step - accuracy: 0.7992 - loss: 0.5698 - val_accuracy: 0.7122 - val_loss: 0.89
23
Epoch 19/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 32s 51ms/step - accuracy: 0.8068 - loss: 0.5522 - val_accuracy: 0.7125 - val_loss: 0.89
97
Epoch 20/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 17s 27ms/step - accuracy: 0.8085 - loss: 0.5304 - val_accuracy: 0.7152 - val_loss: 0.90
40
```

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

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_3 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_4 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_5 (Conv2D)	(None, 4, 4, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten_1 (Flatten)	(None, 512)	0
dense_2 (Dense)	(None, 128)	65,664
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1,290

Total params: 480,608 (1.83 MB)

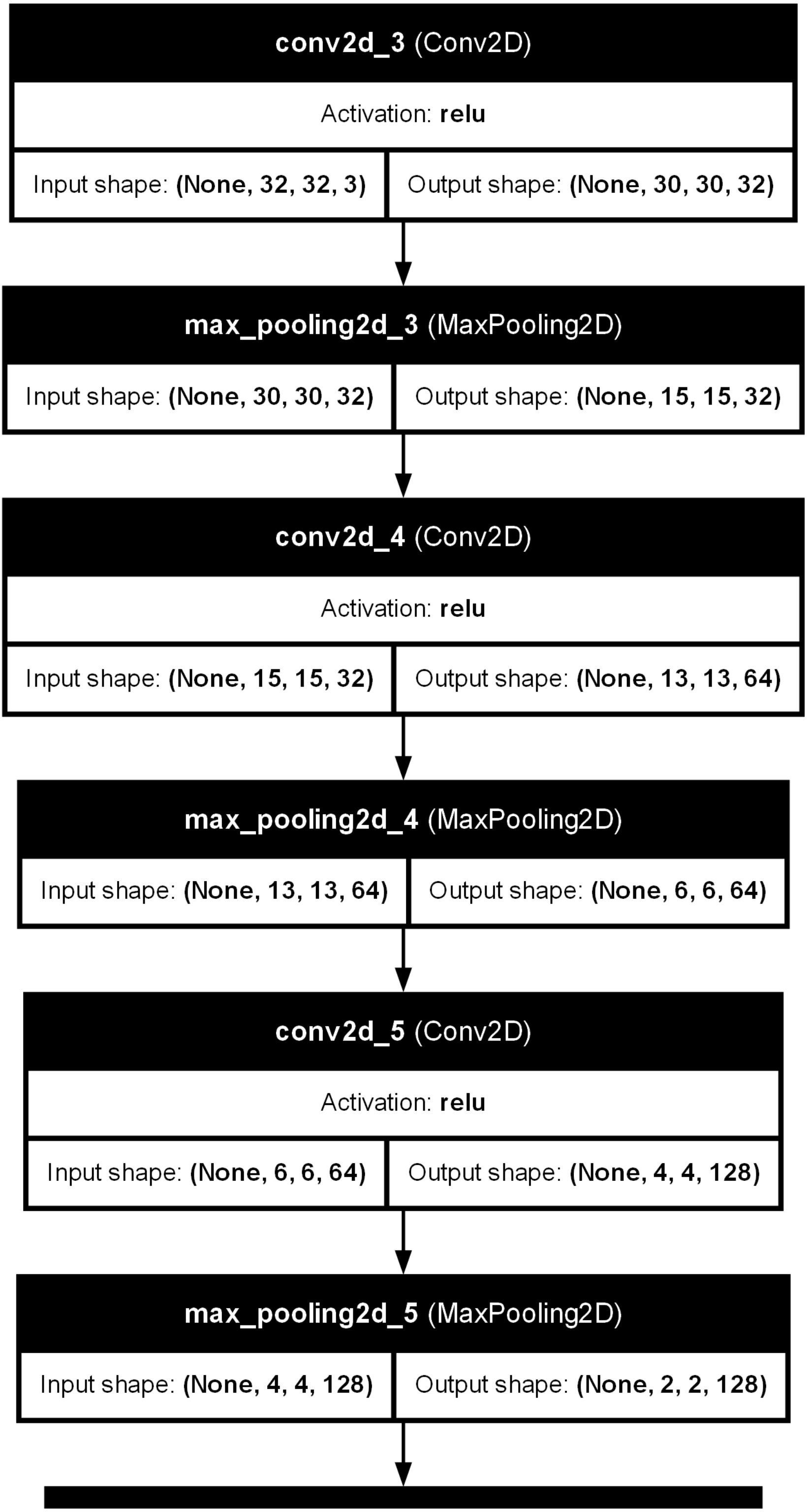
Trainable params: 160,202 (625.79 KB)

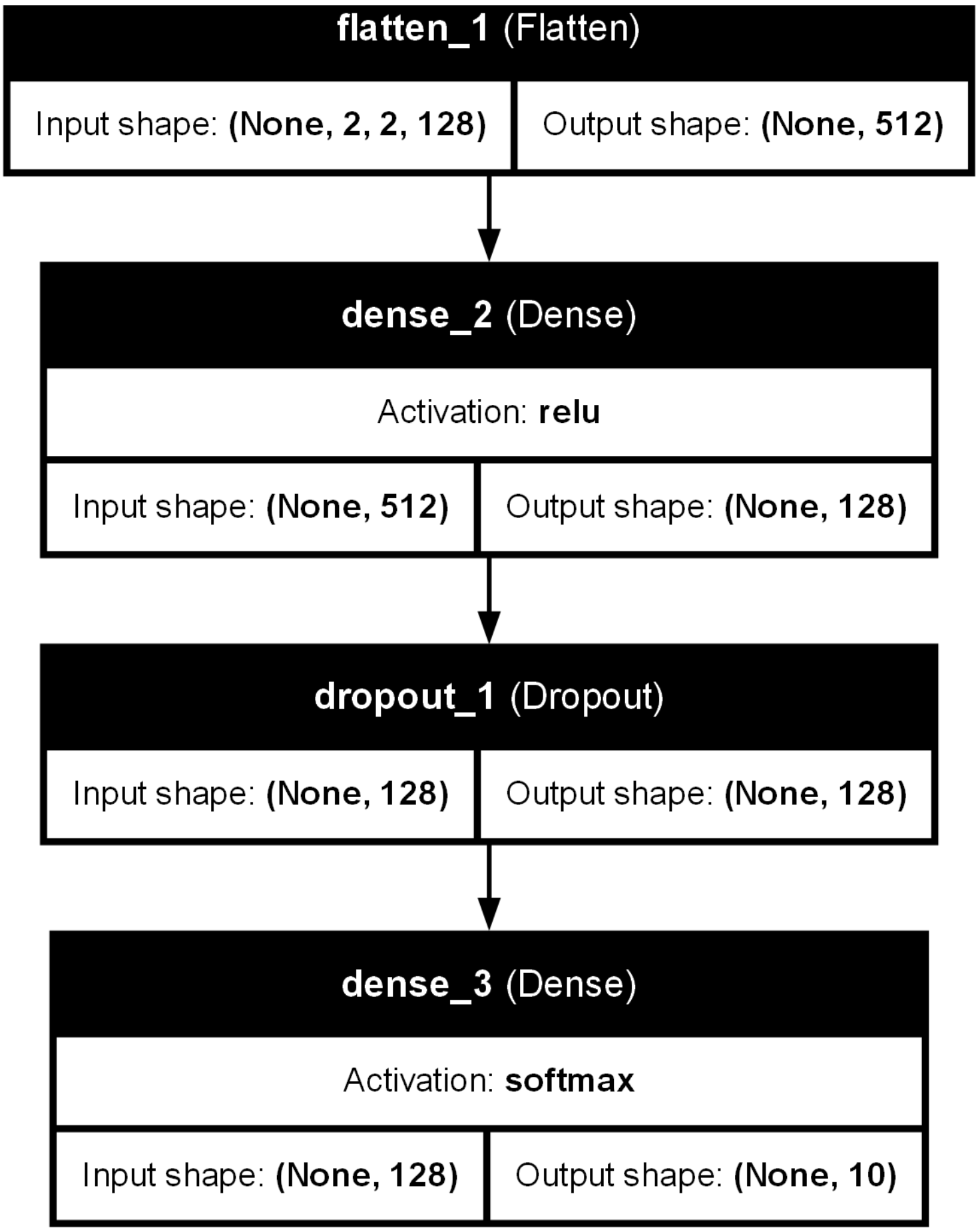
Non-trainable params: 0 (0.00 B)

Optimizer params: 320,406 (1.22 MB)

In [9]:
```
#plotting the model
plot_model(model,to_file='C:/Users/tmbha/Downloads/DLprac3_model.png',show_shapes=True,show_layer_names=True,show_l
```

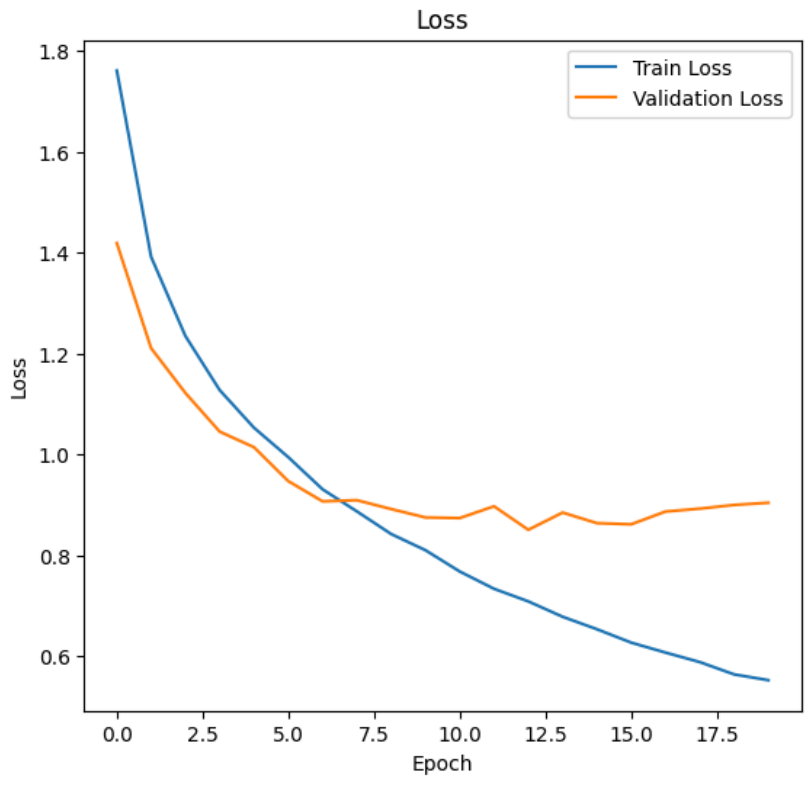
Out[9]:

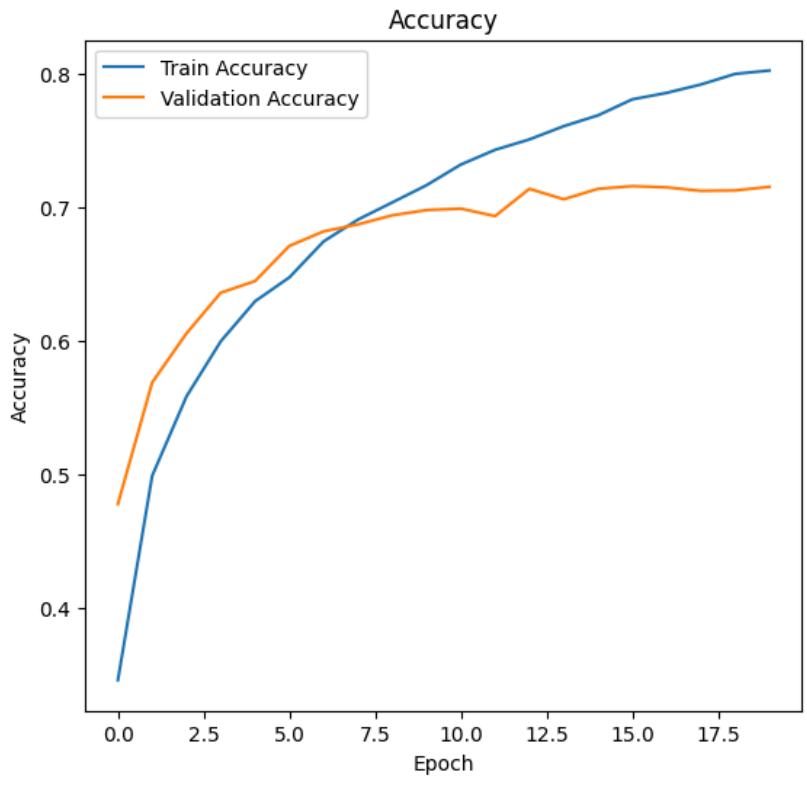

conv2d_3 (Conv2D)
Activation: relu
Input shape: (None, 32, 32, 3)
Output shape: (None, 30, 30, 32)
max_pooling2d_3 (MaxPooling2D)
Input shape: (None, 30, 30, 32)
Output shape: (None, 15, 15, 32)
conv2d_4 (Conv2D)
Activation: relu
Input shape: (None, 15, 15, 32)
Output shape: (None, 13, 13, 64)
max_pooling2d_4 (MaxPooling2D)
Input shape: (None, 13, 13, 64)
Output shape: (None, 6, 6, 64)
conv2d_5 (Conv2D)
Activation: relu
Input shape: (None, 6, 6, 64)
Output shape: (None, 4, 4, 128)
max_pooling2d_5 (MaxPooling2D)
Input shape: (None, 4, 4, 128)
Output shape: (None, 2, 2, 128)



In [10]:
```python
import matplotlib.pyplot as plt

# 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}')

# Plot training & validation loss
plt.figure(figsize=(14, 6))

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()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.7130 - loss: 0.9063
Test Loss: 0.9188
Test Accuracy: 0.7090
```


Loss
Train Loss
Validation Loss
Loss
Epoch
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epoch
0.8
0.7
0.6
0.5
0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5