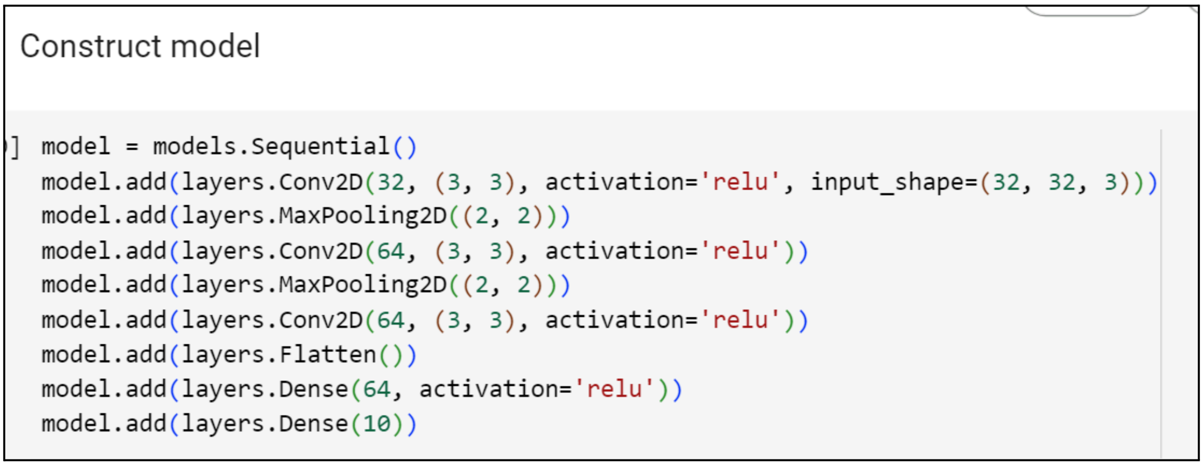
Lab 04 - Applied Machine Learning

Bui Minh Thuan - 104486358

For week 5 lab, we learned about CNN - Neural Network used for computer vision, and how to apply it to classify images with Cifar 10 dataset.

1. Base CNN model

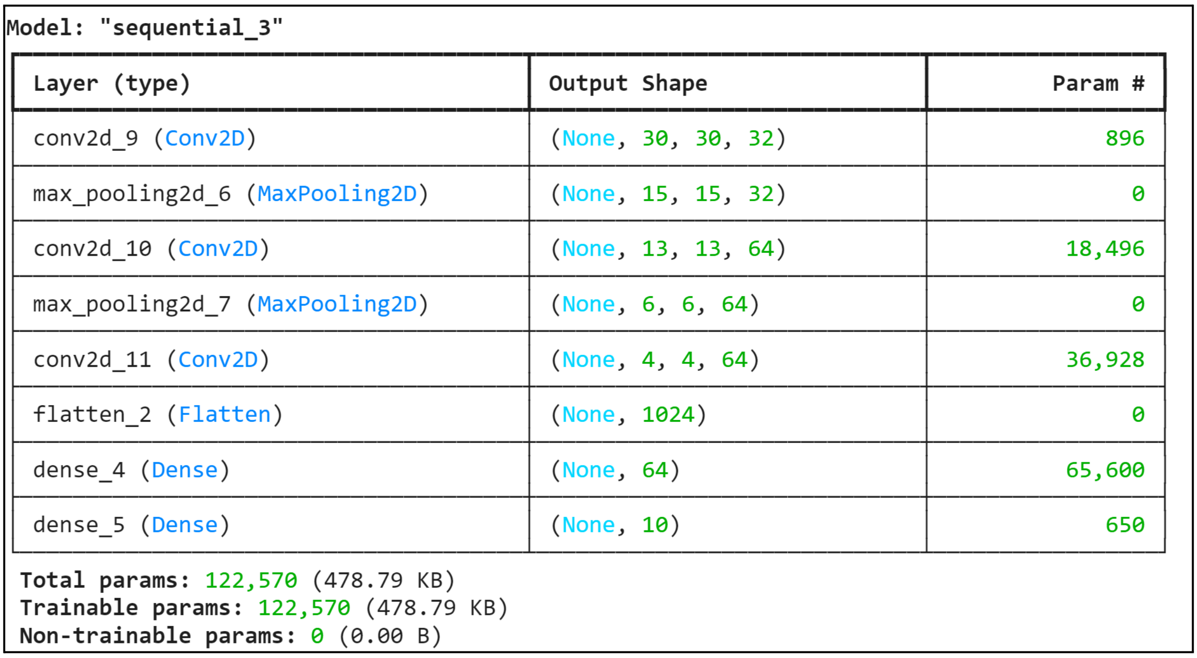
In the lab instruction, we are provided with a simple CNN model, trained on the Cifar 10 dataset. Even though this model can successfully classify 3 images I provided to it, the overall accuracy is quite low, only 71%. That means we can improve it by modifying the model architecture



Construct model

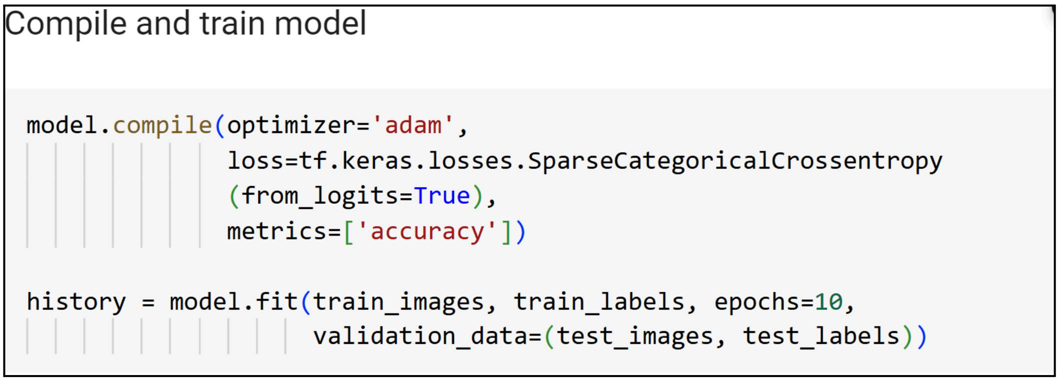
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Model construction



```
Model: "sequential_3"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape                 ┃        Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━┩
│ conv2d_9 (Conv2D)                │ (None, 30, 30, 32)           │            896 │
├──────────────────────────────────┼──────────────────────────────┼────────────────┤
│ max_pooling2d_6 (MaxPooling2D)   │ (None, 15, 15, 32)           │              0 │
├──────────────────────────────────┼──────────────────────────────┼────────────────┤
│ conv2d_10 (Conv2D)               │ (None, 13, 13, 64)           │         18,496 │
├──────────────────────────────────┼──────────────────────────────┼────────────────┤
│ max_pooling2d_7 (MaxPooling2D)   │ (None, 6, 6, 64)             │              0 │
├──────────────────────────────────┼──────────────────────────────┼────────────────┤
│ conv2d_11 (Conv2D)               │ (None, 4, 4, 64)             │         36,928 │
├──────────────────────────────────┼──────────────────────────────┼────────────────┤
│ flatten_2 (Flatten)              │ (None, 1024)                 │              0 │
├──────────────────────────────────┼──────────────────────────────┼────────────────┤
│ dense_4 (Dense)                  │ (None, 64)                   │         65,600 │
├──────────────────────────────────┼──────────────────────────────┼────────────────┤
│ dense_5 (Dense)                  │ (None, 10)                   │            650 │
└──────────────────────────────────┴──────────────────────────────┴────────────────┘
 Total params: 122,570 (478.79 KB)
 Trainable params: 122,570 (478.79 KB)
 Non-trainable params: 0 (0.00 B)
```

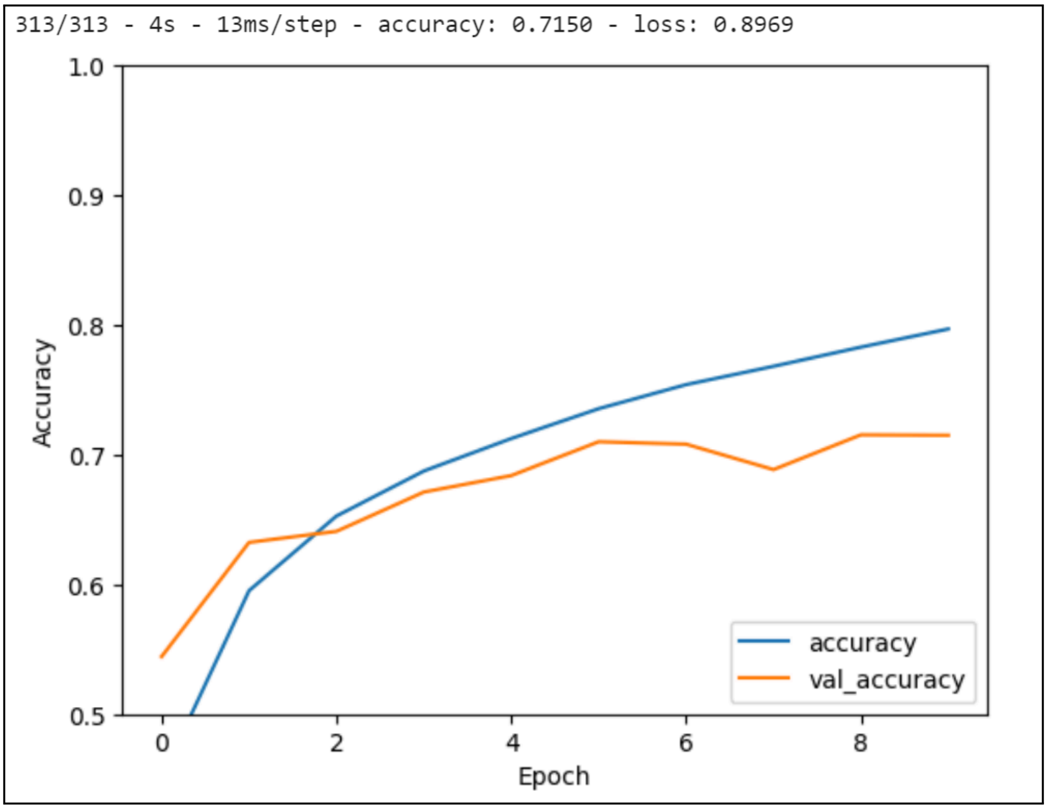
Base Model summary



Compile and train model

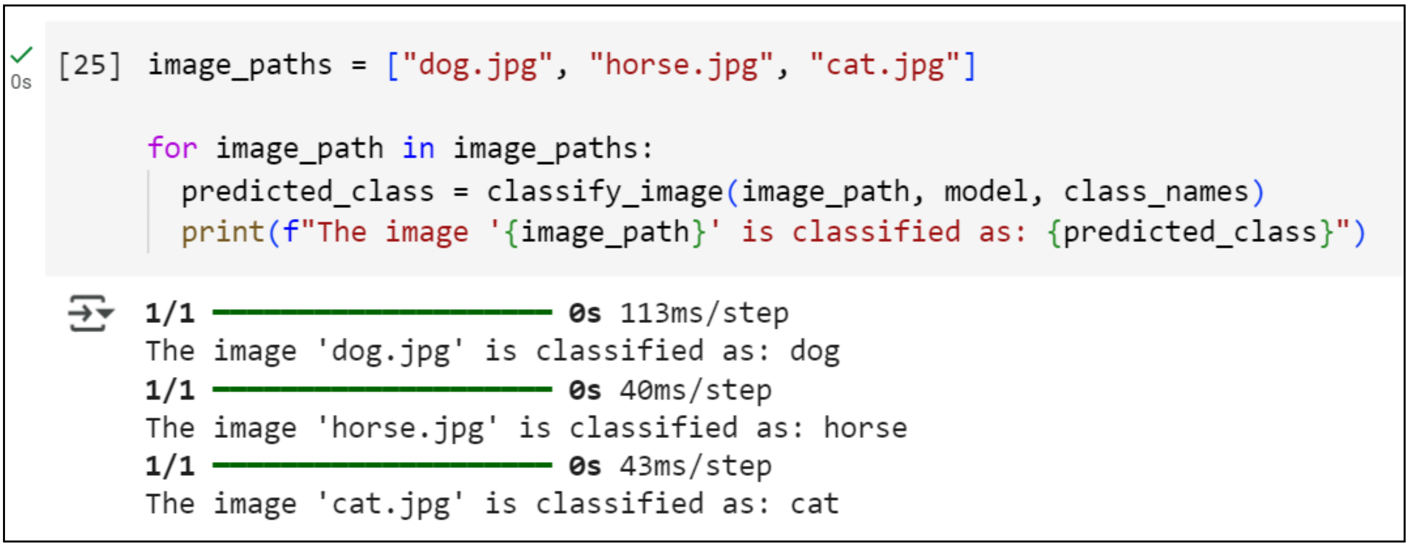
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy
              (from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

Model training



```
313/313 - 4s - 13ms/step - accuracy: 0.7150 - loss: 0.8969
```

Model evaluation



```
[25] image_paths = ["dog.jpg", "horse.jpg", "cat.jpg"]

     for image_path in image_paths:
       predicted_class = classify_image(image_path, model, class_names)
       print(f"The image '{image_path}' is classified as: {predicted_class}")
```

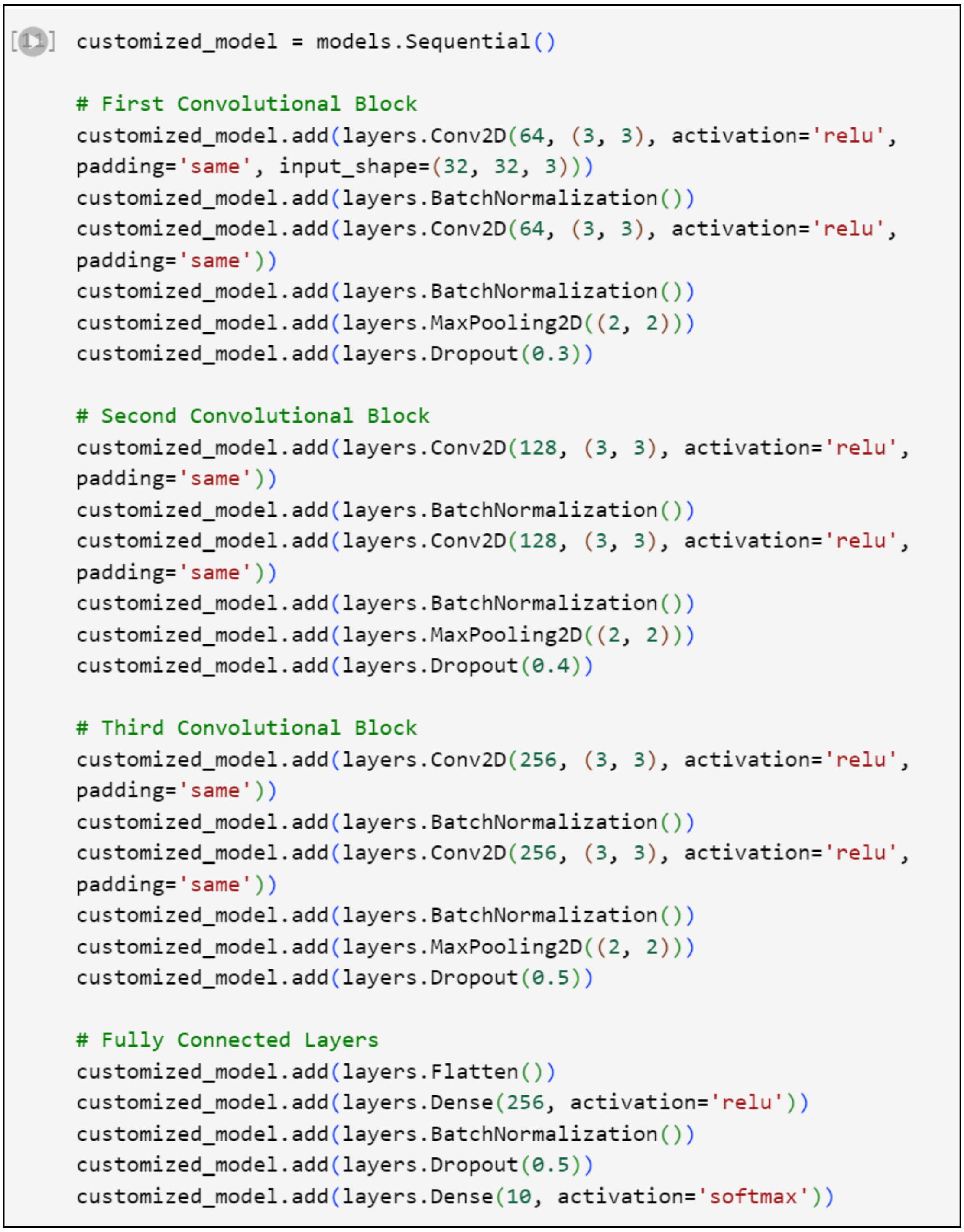
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step
The image 'dog.jpg' is classified as: dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
The image 'horse.jpg' is classified as: horse
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
The image 'cat.jpg' is classified as: cat
```

Test classifying with 3 images

2. Customized CNN model

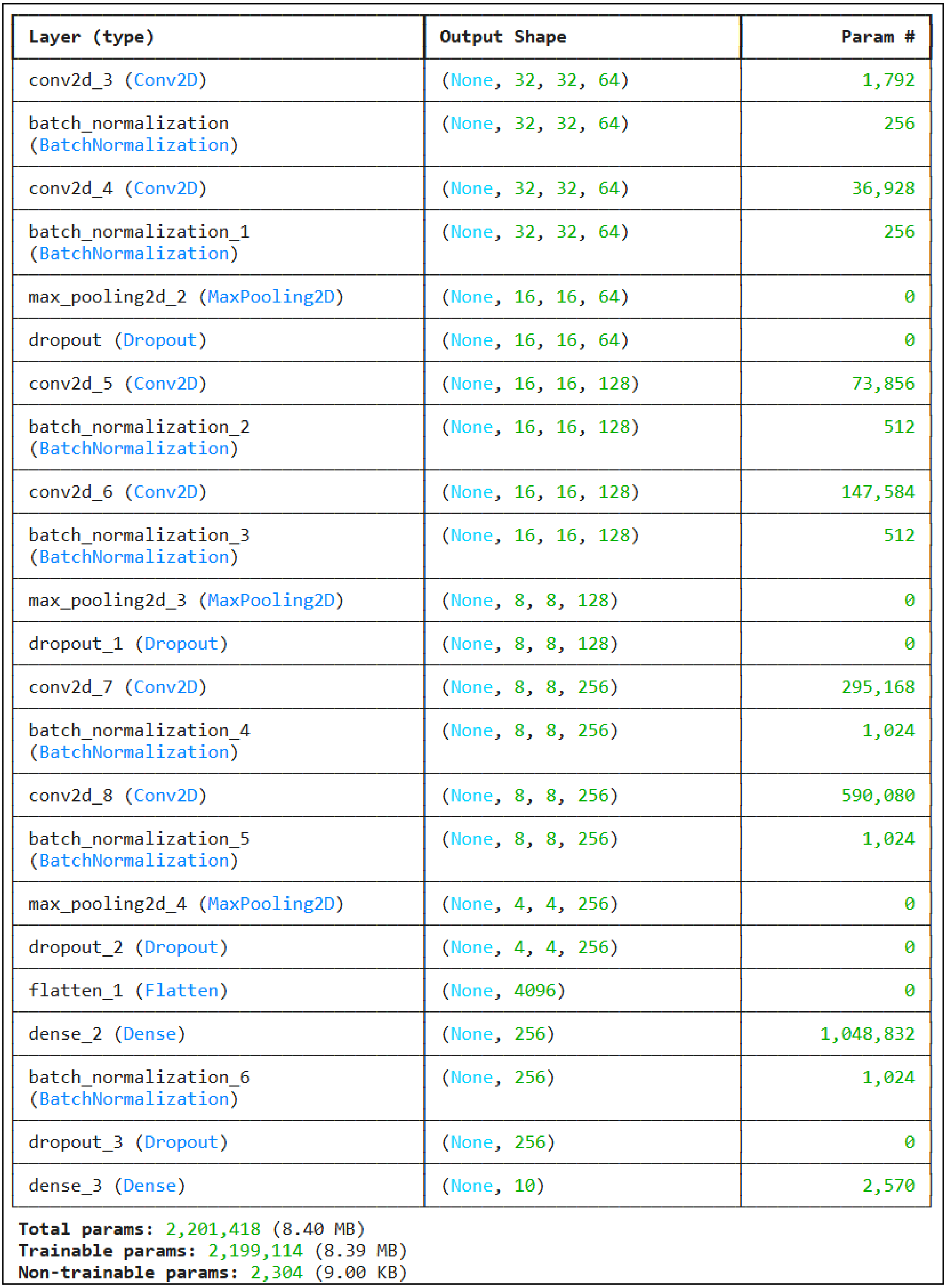
In this step, I tried to modify the base CNN model, the changes include:

- **Increase Model Depth**: Add more convolutional layers to force the model to learn more.
- **Batch Normalization**: Normalize data during training
- **Dropout**: Prevents overfitting.
- **Global Average Pooling**: Reduces parameters and improves generalization.
- **Use Softmax for Output**: Better layer for classification



```
customized_model = models.Sequential()

# First Convolutional Block
customized_model.add(layers.Conv2D(64, (3, 3), activation='relu',
padding='same', input_shape=(32, 32, 3)))
customized_model.add(layers.BatchNormalization())
customized_model.add(layers.Conv2D(64, (3, 3), activation='relu',
padding='same'))
customized_model.add(layers.BatchNormalization())
customized_model.add(layers.MaxPooling2D((2, 2)))
customized_model.add(layers.Dropout(0.3))

# Second Convolutional Block
customized_model.add(layers.Conv2D(128, (3, 3), activation='relu',
padding='same'))
customized_model.add(layers.BatchNormalization())
customized_model.add(layers.Conv2D(128, (3, 3), activation='relu',
padding='same'))
customized_model.add(layers.BatchNormalization())
customized_model.add(layers.MaxPooling2D((2, 2)))
customized_model.add(layers.Dropout(0.4))

# Third Convolutional Block
customized_model.add(layers.Conv2D(256, (3, 3), activation='relu',
padding='same'))
customized_model.add(layers.BatchNormalization())
customized_model.add(layers.Conv2D(256, (3, 3), activation='relu',
padding='same'))
customized_model.add(layers.BatchNormalization())
customized_model.add(layers.MaxPooling2D((2, 2)))
customized_model.add(layers.Dropout(0.5))

# Fully Connected Layers
customized_model.add(layers.Flatten())
customized_model.add(layers.Dense(256, activation='relu'))
customized_model.add(layers.BatchNormalization())
customized_model.add(layers.Dropout(0.5))
customized_model.add(layers.Dense(10, activation='softmax'))
```

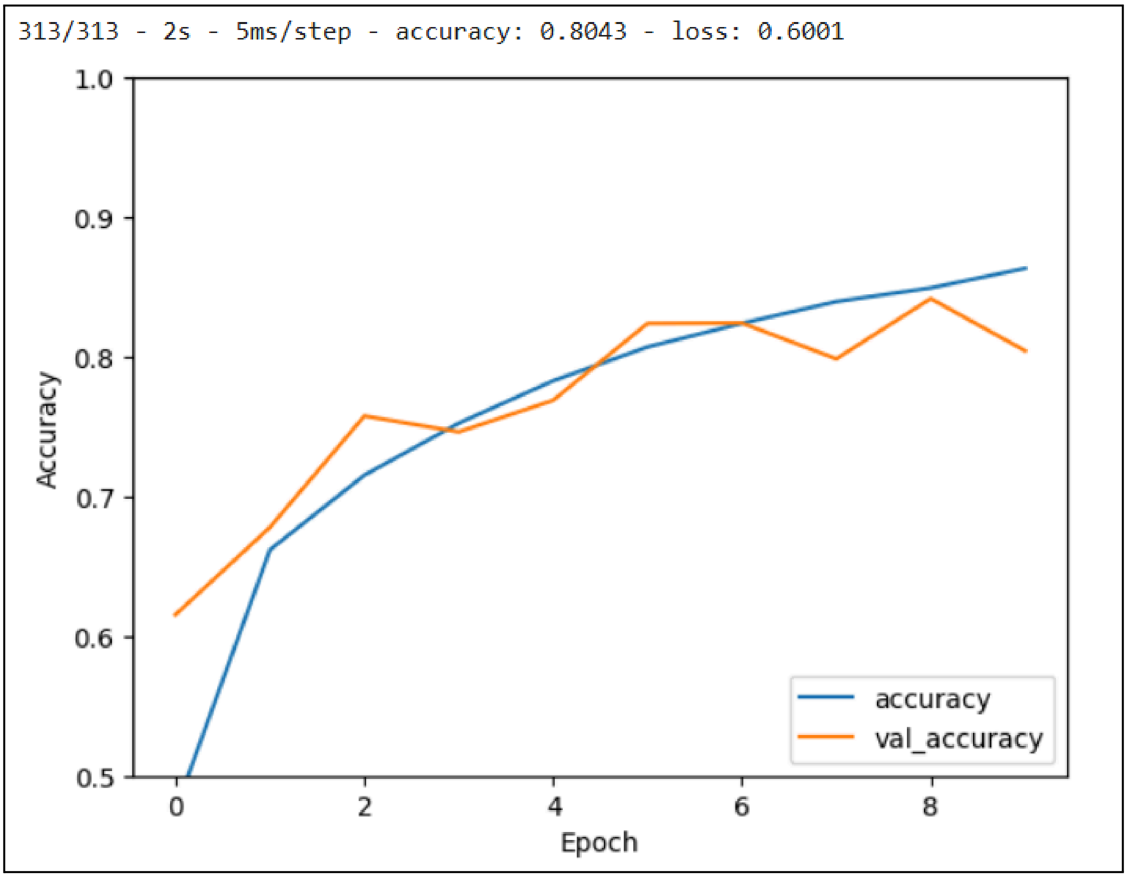
Model construction



Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_4 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_5 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_6 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
conv2d_7 (Conv2D)	(None, 8, 8, 256)	295,168
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_8 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 256)	1,024
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_2 (Dense)	(None, 256)	1,048,832
batch_normalization_6 (BatchNormalization)	(None, 256)	1,024
dropout_3 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 10)	2,570

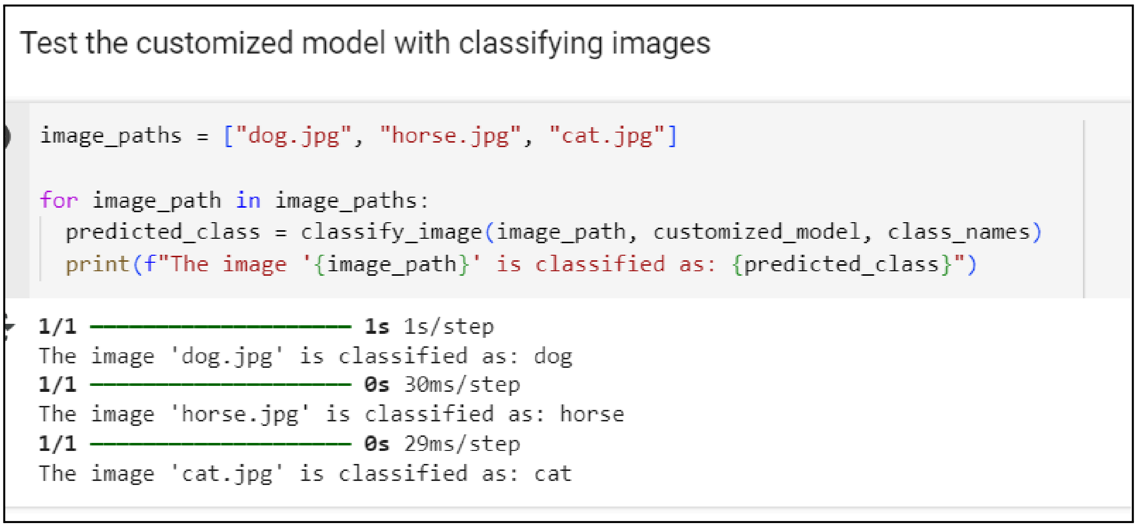
```
Total params: 2,201,418 (8.40 MB)
Trainable params: 2,199,114 (8.39 MB)
Non-trainable params: 2,304 (9.00 KB)
```

Customized Model summary



```
313/313 - 2s - 5ms/step - accuracy: 0.8043 - loss: 0.6001
```

Model evaluation



Test the customized model with classifying images

```
image_paths = ["dog.jpg", "horse.jpg", "cat.jpg"]

for image_path in image_paths:
  predicted_class = classify_image(image_path, customized_model, class_names)
  print(f"The image '{image_path}' is classified as: {predicted_class}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
The image 'dog.jpg' is classified as: dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
The image 'horse.jpg' is classified as: horse
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
The image 'cat.jpg' is classified as: cat
```

Test classifying with 3 images

In this step, I tried to modify the base CNN model, the changes include: As we can see, by adding more layers, the model now produces better classification with overall 80%, meaning 10% improvement compared to base model.