Reyhane Shahrokhian 99521361

HomeWork3 of DeepLearning Course

Dr. DavoodAbadi

Question 1:

1-1:

High learning rates can cause the model parameters to update too quickly, leading to overshooting the optimal values and skipping the global minimum. This can result in the optimization process diverging instead of converging to a minimum. If the training loss becomes extremely large or shows oscillatory behavior rather than stabilizing or decreasing, it may indicate that the learning rate is too high.

1-2:

A very low learning rate can result in slow convergence, meaning that the optimization process takes a long time to reach a minimum. If the training loss decreases very slowly, and the model parameters take a long time to update, it may suggest that the learning rate is too low.
It also may cause the optimization algorithm to get stuck in local minima, preventing it from exploring and finding the global minimum. In that situation, the model might converge to a solution that is suboptimal, and evaluating the performance on a validation set may reveal that the model is not achieving its full potential.

1-3:

A saddle point is a point in the parameter space of a machine learning model where the gradient of the loss function is zero, but it is not an optimum. Instead, it is a point where the surface of the loss function resembles a saddle.

SGD can have difficulties escaping saddle points due to the symmetric update rule, where the updates are proportional to the gradient But, Adam's adaptive learning rates and momentum can help it navigate saddle points more efficiently than SGD. SGD may perform better in cases where saddle points are not prevalent, and the optimization landscape is relatively simple. On the other hand, Adam is often more robust in complex optimization landscapes with saddle points and can converge faster.

1-4:

The graph on the left is a batch, while the graph on the right is a mini-batch.

In batch gradient descent, the entire training set is used to update the model's parameters in each iteration. This can be computationally expensive and slow to converge, especially for large training sets. In mini-batch gradient descent, a subset of the training set, called a mini-batch, is used to update the model's parameters in each iteration. This is much faster than batch gradient descent, and it also tends to converge better.

The image shows that the loss function for mini-batch gradient descent is noisier than the loss function for batch gradient descent. This is because mini-batch gradient descent is only using a small subset of the training data to calculate the gradients. However, mini-batch gradient descent still converges to a good solution, and it is much faster and more efficient than batch gradient descent for large datasets.

Question 2:

First the output of the convolutional layer is computed and the result is as follow:

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22} = 1$$

$O_{12} = X_{12}F_{11} + X_{13}F_{12} + X_{22}F_{21} + X_{23}F_{22} = 2$

$O_{21} = X_{21}F_{11} + X_{22}F_{12} + X_{31}F_{21} + X_{32}F_{22} = -10$

$O_{22} = X_{22}F_{11} + X_{23}F_{12} + X_{32}F_{21} + X_{33}F_{22} = 8$

1	2
-10	8

Now the Global average Pooling is applied on it:

$y = (O_{11} + O_{12} + O_{21} + O_{22}) \div 4 = 0.25$

BackPropagation computations:

$\frac{\partial y}{\partial O_{11}} = 0.25$, $\frac{\partial y}{\partial O_{12}} = 0.25$, $\frac{\partial y}{\partial O_{21}} = 0.25$, $\frac{\partial y}{\partial O_{22}} = 0.25$

$\frac{\partial O_{11}}{\partial F_{11}} = X_{11}$, $\frac{\partial O_{11}}{\partial F_{12}} = X_{12}$, $\frac{\partial O_{11}}{\partial F_{21}} = X_{21}$, $\frac{\partial O_{11}}{\partial F_{22}} = X_{22}$

$\frac{\partial O_{12}}{\partial F_{11}} = X_{12}$, $\frac{\partial O_{12}}{\partial F_{12}} = X_{13}$, $\frac{\partial O_{12}}{\partial F_{21}} = X_{22}$, $\frac{\partial O_{12}}{\partial F_{22}} = X_{23}$

$\frac{\partial O_{21}}{\partial F_{11}} = X_{21}$, $\frac{\partial O_{21}}{\partial F_{12}} = X_{22}$, $\frac{\partial O_{21}}{\partial F_{21}} = X_{31}$, $\frac{\partial O_{21}}{\partial F_{22}} = X_{32}$

$\frac{\partial O_{22}}{\partial F_{11}} = X_{22}$, $\frac{\partial O_{22}}{\partial F_{12}} = X_{23}$, $\frac{\partial O_{22}}{\partial F_{21}} = X_{32}$, $\frac{\partial O_{22}}{\partial F_{22}} = X_{33}$

$\frac{\partial O_{11}}{\partial X_{11}} = F_{11}$, $\frac{\partial O_{11}}{\partial X_{12}} = F_{12}$, $\frac{\partial O_{11}}{\partial X_{21}} = F_{21}$, $\frac{\partial O_{11}}{\partial X_{22}} = F_{22}$

$\frac{\partial O_{12}}{\partial X_{12}} = F_{11}$, $\frac{\partial O_{12}}{\partial X_{13}} = F_{12}$, $\frac{\partial O_{12}}{\partial X_{22}} = F_{21}$, $\frac{\partial O_{12}}{\partial X_{23}} = F_{22}$

$$\frac{\partial O_{21}}{\partial X_{21}} = F_{11}, \ \frac{\partial O_{21}}{\partial X_{22}} = F_{12}, \ \frac{\partial O_{21}}{\partial X_{31}} = F_{21}, \ \frac{\partial O_{21}}{\partial X_{32}} = F_{22}$$

$$\frac{\partial O_{22}}{\partial X_{22}} = F_{11}, \ \frac{\partial O_{22}}{\partial X_{23}} = F_{12}, \ \frac{\partial O_{22}}{\partial X_{32}} = F_{21}, \ \frac{\partial O_{22}}{\partial X_{33}} = F_{22}$$

$$\frac{\partial l}{\partial X_{11}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial X_{11}} = \frac{\partial l}{\partial y} \times 0.25 \times 2 = 0.5\frac{\partial l}{\partial y}$$

$$\frac{\partial l}{\partial X_{12}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial X_{12}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{12}} \times \frac{\partial O_{12}}{\partial X_{12}} = \frac{\partial l}{\partial y} \times 0.25 \times 0 + \frac{\partial l}{\partial y} \times 0.25 \times 2 = 0.5\frac{\partial l}{\partial y}$$

$$\frac{\partial l}{\partial X_{13}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{12}} \times \frac{\partial O_{12}}{\partial X_{13}} = \frac{\partial l}{\partial y} \times 0.25 \times 0 = 0$$

$$\frac{\partial l}{\partial X_{21}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial X_{21}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial X_{21}} = \frac{\partial l}{\partial y} \times 0.25 \times -3 + \frac{\partial l}{\partial y} \times 0.25 \times 2 = -0.25\frac{\partial l}{\partial y}$$

$$\frac{\partial l}{\partial X_{22}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial X_{22}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{12}} \times \frac{\partial O_{12}}{\partial X_{22}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial X_{22}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial X_{22}}$$

$$= \frac{\partial l}{\partial y} \times 0.25 \times 1 + \frac{\partial l}{\partial y} \times 0.25 \times -3 + \frac{\partial l}{\partial y} \times 0.25 \times 0 + \frac{\partial l}{\partial y} \times 0.25 \times 2 = 0$$

$$\frac{\partial l}{\partial X_{23}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{12}} \times \frac{\partial O_{12}}{\partial X_{23}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial X_{23}} = \frac{\partial l}{\partial y} \times 0.25 \times 1 + \frac{\partial l}{\partial y} \times 0.25 \times 0 = 0.25\frac{\partial l}{\partial y}$$

$$\frac{\partial l}{\partial X_{31}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial X_{31}} = \frac{\partial l}{\partial y} \times 0.25 \times -3 = -0.75\frac{\partial l}{\partial y}$$

$$\frac{\partial l}{\partial X_{32}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial X_{32}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial X_{32}} = \frac{\partial l}{\partial y} \times 0.25 \times 1 + \frac{\partial l}{\partial y} \times 0.25 \times -3 = -0.5\frac{\partial l}{\partial y}$$

$$\frac{\partial l}{\partial X_{33}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial X_{33}} = \frac{\partial l}{\partial y} \times 0.25 \times 1 = 0.25\frac{\partial l}{\partial y}$$

$$\frac{\partial l}{\partial F_{11}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{12}} \times \frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial F_{11}}$$

$$= \frac{\partial l}{\partial y} \times 0.25 \times 3 + \frac{\partial l}{\partial y} \times 0.25 \times 4 + \frac{\partial l}{\partial y} \times 0.25 \times 2 + \frac{\partial l}{\partial y} \times 0.25 \times 1 = 2.5\frac{\partial l}{\partial y}$$

$$\frac{\partial l}{\partial F_{12}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{12}} \times \frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial F_{12}}$$

$$= \frac{\partial l}{\partial y} \times 0.25 \times 4 + \frac{\partial l}{\partial y} \times 0.25 \times 5 + \frac{\partial l}{\partial y} \times 0.25 \times 1 + \frac{\partial l}{\partial y} \times 0.25 \times - 3 = 1.75\frac{\partial l}{\partial y}$$

$$\frac{\partial l}{\partial F_{21}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial F_{21}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{12}} \times \frac{\partial O_{12}}{\partial F_{21}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial F_{21}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial F_{21}}$$

$$= \frac{\partial l}{\partial y} \times 0.25 \times 2 + \frac{\partial l}{\partial y} \times 0.25 \times 1 + \frac{\partial l}{\partial y} \times 0.25 \times 4 + \frac{\partial l}{\partial y} \times 0.25 \times - 2 = 1.25\frac{\partial l}{\partial y}$$

$$\frac{\partial l}{\partial F_{22}} = \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial F_{22}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{12}} \times \frac{\partial O_{12}}{\partial F_{22}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial F_{22}} + \frac{\partial l}{\partial y} \times \frac{\partial y}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial F_{22}}$$

$$= \frac{\partial l}{\partial y} \times 0.25 \times 1 + \frac{\partial l}{\partial y} \times 0.25 \times - 3 + \frac{\partial l}{\partial y} \times 0.25 \times - 2 + \frac{\partial l}{\partial y} \times 0.25 \times 0 = - \frac{\partial l}{\partial y}$$

Question 3:

3-1:

We should know how to compute parameters and output_shapes:

Conv1D:

$Parameters = filters \times (kernel\_size \times imput\_channel\_size + 1)$

$Output\_width = \frac{Input\_width - kernel\_size + 2\times Padding}{stride} + 1$

$Output\_channel = filters$

MaxPool1D:

$Parameters = 0$

$Output\_width = \frac{Input\_width - Pool\_size}{stride} + 1$

$Output\_channel$ = stay the same

Flatten:

$Parameters = 0$

$Output\_width \;=\;$ It converts multidimensional input to 1 dimension.

Dense:

$Parameters = (input\_size \;+\; 1) \times units$

$Output\_width \;=\; units$

Layers	Output_shape	Parameters
Conv1D(16, kernel_size = 3)	$(500 - 3 + 1,\ 16) = (498, 16)$	$16 \times (3 \times 7 + 1) = 352$
MaxPool1D()	$(\frac{498-2}{2} + 1, 16) = (249, 16)$	0
Conv1D(32, kernel_size = 5)	$(249 - 5 + 1,\ 32) = (245, 32)$	$32 \times (5 \times 16 + 1) = 2592$
MaxPool1D()	$(\frac{245-2}{2} + 1, 32) = (122, 32)$	0
Conv1D(64, kernel_size = 5)	$(122 - 5 + 1,\ 64) = (118, 64)$	$64 \times (5 \times 32 + 1) = 10304$
MaxPool1D()	$(\frac{118-2}{2} + 1, 64) = (59, 64)$	0
Flatten()	$(59 \times 64) = (3776)$	0
Dense(units = 128)	(128)	$(3776 + 1) \times 128 = 483456$
Dense(units = 5)	(5)	$(128 + 1) \times 5 = 645$

3-2:

Conv2D operates on two-dimensional input data, typically used for processing images with spatial dimensions (height and width). It processes input data with shape (height, width, channels).

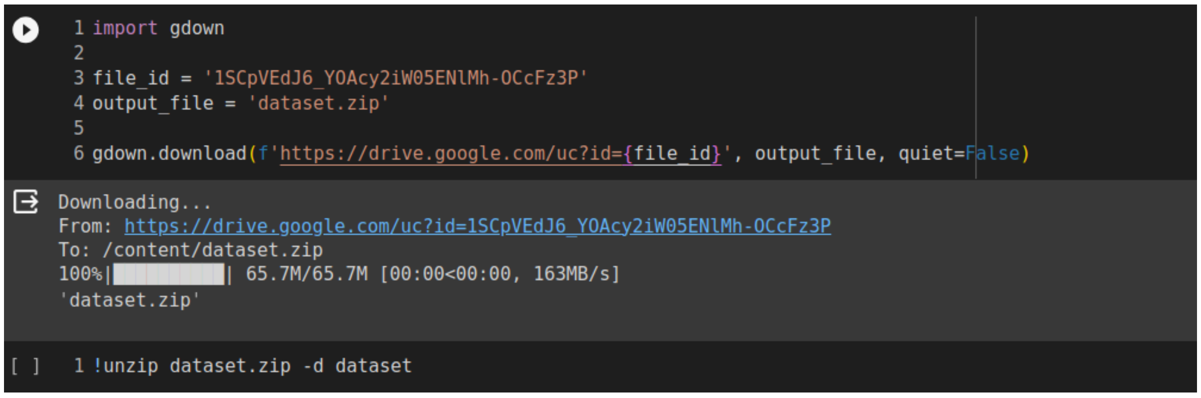
But, Conv3D operates on three-dimensional input data, suitable for processing volumetric data with additional depth (such as videos or 3D medical images). It processes input data with shape (depth, height, width, channels).

Usage of Conv3D: Conv3D is commonly used in applications related to video understanding, action recognition, and video classification. It can capture both spatial features (such as shapes and objects) and temporal features (such as motion and changes over time). It's also suitable for any application where the input data has a three-dimensional structure, such as CT scans, MRI volumes, or any other 3D representation of information.

Question 4:

As instructions, first the dataset should be downloaded and splitted to train, validation, and test datasets. In reading images from the directory, the label_mode is set as int because it's a binary classification task(yes or no which are class_names). I also set seed in order to shuffle the data. And as we're setting both validation and training dataset, the subset is equal to both.

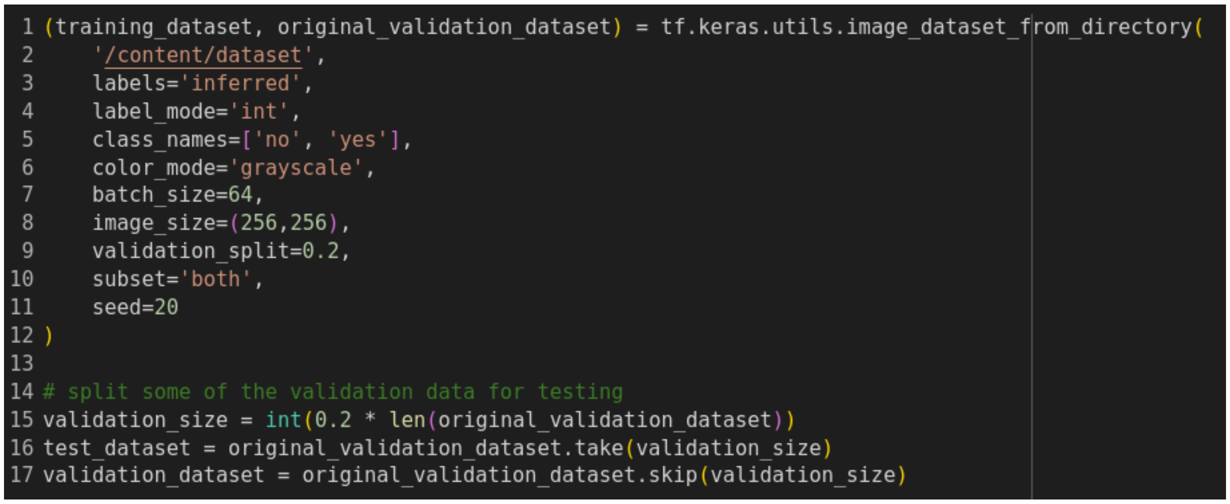
After that, I split some of the validation data for testing.



```
import gdown

file_id = '1SCpVEdJ6_YOAcy2iW05ENlMh-OCcFz3P'
output_file = 'dataset.zip'

gdown.download(f'https://drive.google.com/uc?id={file_id}', output_file, quiet=False)
```

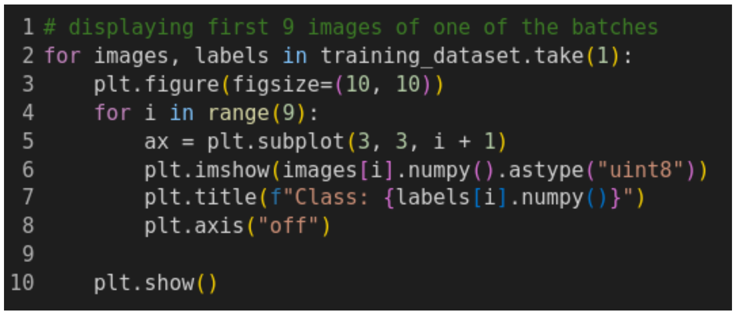
```
Downloading...
From: https://drive.google.com/uc?id=1SCpVEdJ6_YOAcy2iW05ENlMh-OCcFz3P
To: /content/dataset.zip
100%|██████████| 65.7M/65.7M [00:00<00:00, 163MB/s]
'dataset.zip'
```

```
!unzip dataset.zip -d dataset
```

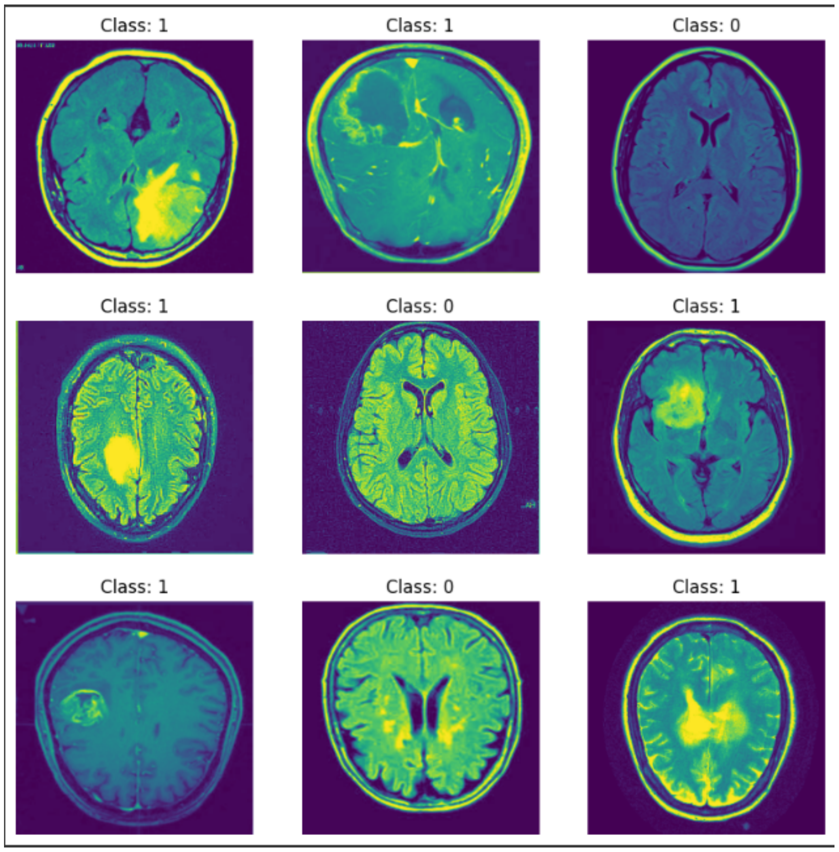


```python
(training_dataset, original_validation_dataset) = tf.keras.utils.image_dataset_from_directory(
    '/content/dataset',
    labels='inferred',
    label_mode='int',
    class_names=['no', 'yes'],
    color_mode='grayscale',
    batch_size=64,
    image_size=(256,256),
    validation_split=0.2,
    subset='both',
    seed=20
)

# split some of the validation data for testing
validation_size = int(0.2 * len(original_validation_dataset))
test_dataset = original_validation_dataset.take(validation_size)
validation_dataset = original_validation_dataset.skip(validation_size)
```

Now,I just display 9 images of one of the batches:

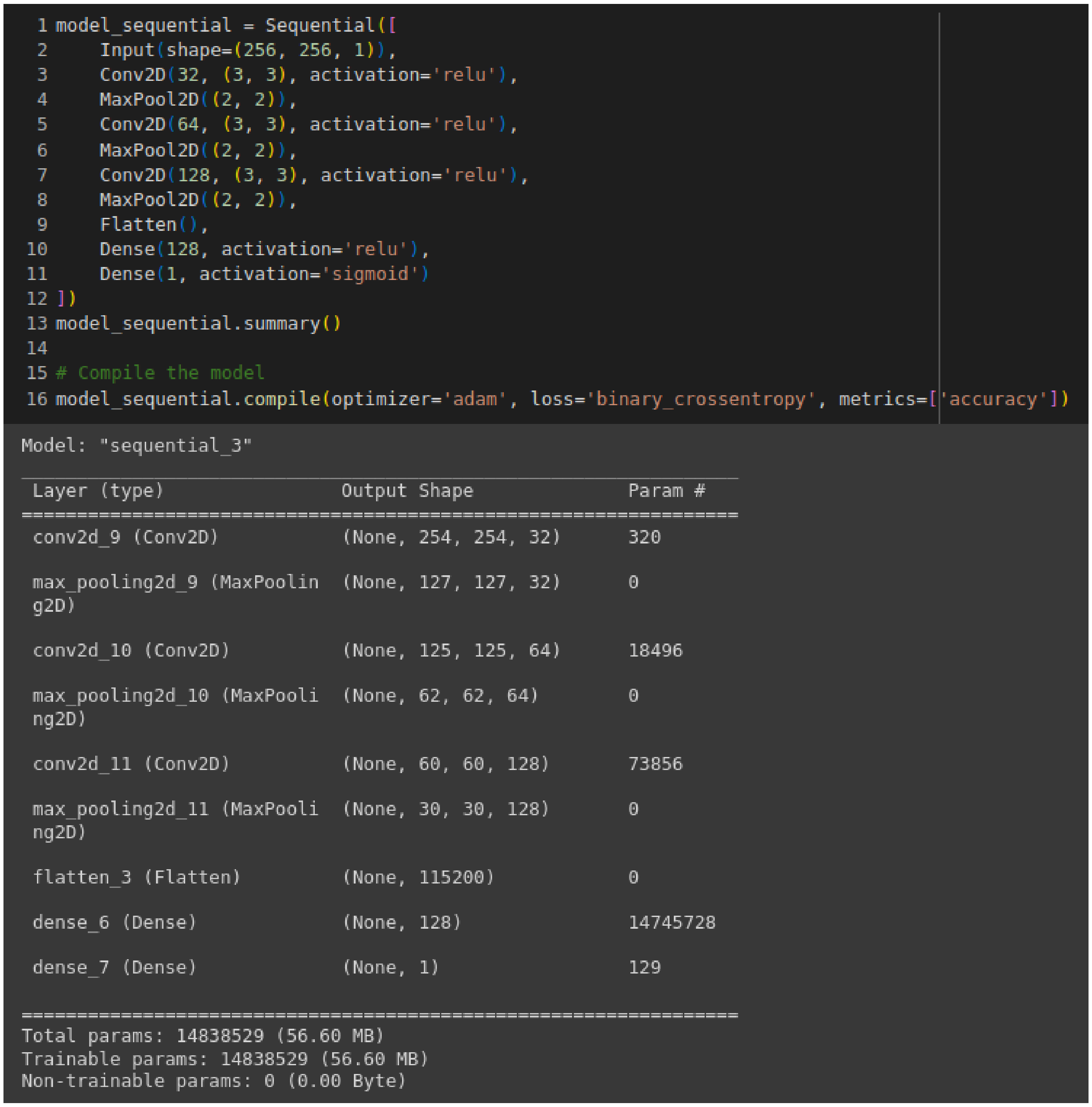


```python
# displaying first 9 images of one of the batches
for images, labels in training_dataset.take(1):
    plt.figure(figsize=(10, 10))
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(f"Class: {labels[i].numpy()}")
        plt.axis("off")

    plt.show()
```



Now the model should be built in both ways, sequential and functional.

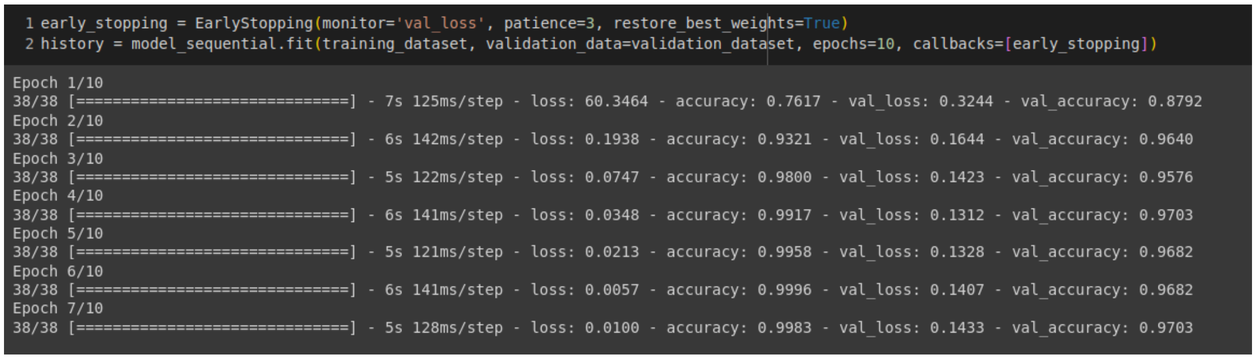
Sequential:



```
model_sequential = Sequential([
    Input(shape=(256, 256, 1)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPool2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPool2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPool2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')
])
model_sequential.summary()

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

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 254, 254, 32)      320

 max_pooling2d_9 (MaxPoolin  (None, 127, 127, 32)      0
 g2D)

 conv2d_10 (Conv2D)          (None, 125, 125, 64)      18496

 max_pooling2d_10 (MaxPooli  (None, 62, 62, 64)        0
 ng2D)

 conv2d_11 (Conv2D)          (None, 60, 60, 128)       73856

 max_pooling2d_11 (MaxPooli  (None, 30, 30, 128)       0
 ng2D)

 flatten_3 (Flatten)         (None, 115200)            0

 dense_6 (Dense)             (None, 128)               14745728

 dense_7 (Dense)             (None, 1)                 129

=================================================================
Total params: 14838529 (56.60 MB)
Trainable params: 14838529 (56.60 MB)
Non-trainable params: 0 (0.00 Byte)
```

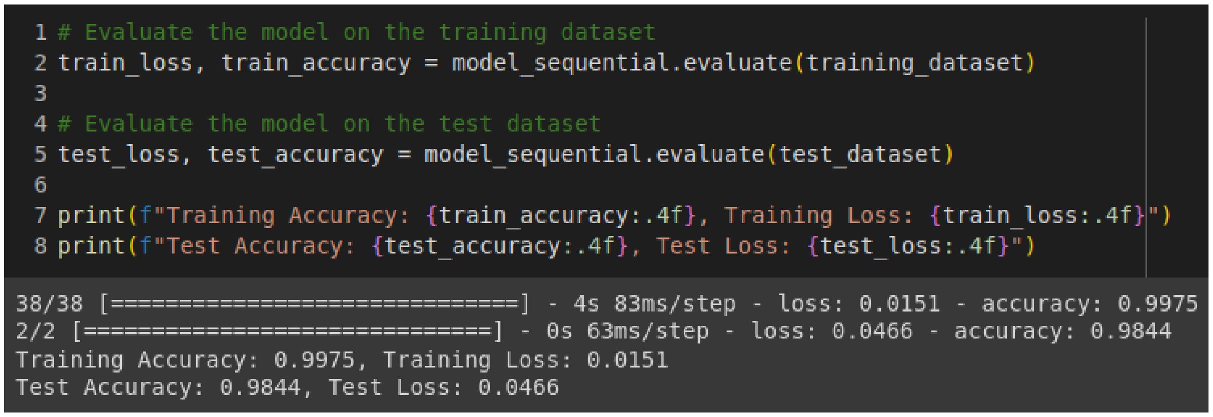
Before training the model, I set an early_stopping in order to check and monitor the validation_loss which will stop the training automatically before ending epochs if overfitting is occurring.



```
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
history = model_sequential.fit(training_dataset, validation_data=validation_dataset, epochs=10, callbacks=[early_stopping])
```

```
Epoch 1/10
38/38 [==============================] - 7s 125ms/step - loss: 60.3464 - accuracy: 0.7617 - val_loss: 0.3244 - val_accuracy: 0.8792
Epoch 2/10
38/38 [==============================] - 6s 142ms/step - loss: 0.1938 - accuracy: 0.9321 - val_loss: 0.1644 - val_accuracy: 0.9640
Epoch 3/10
38/38 [==============================] - 5s 122ms/step - loss: 0.0747 - accuracy: 0.9800 - val_loss: 0.1423 - val_accuracy: 0.9576
Epoch 4/10
38/38 [==============================] - 6s 141ms/step - loss: 0.0348 - accuracy: 0.9917 - val_loss: 0.1312 - val_accuracy: 0.9703
Epoch 5/10
38/38 [==============================] - 5s 121ms/step - loss: 0.0213 - accuracy: 0.9958 - val_loss: 0.1328 - val_accuracy: 0.9682
Epoch 6/10
38/38 [==============================] - 6s 141ms/step - loss: 0.0057 - accuracy: 0.9996 - val_loss: 0.1407 - val_accuracy: 0.9682
Epoch 7/10
38/38 [==============================] - 5s 128ms/step - loss: 0.0100 - accuracy: 0.9983 - val_loss: 0.1433 - val_accuracy: 0.9703
```

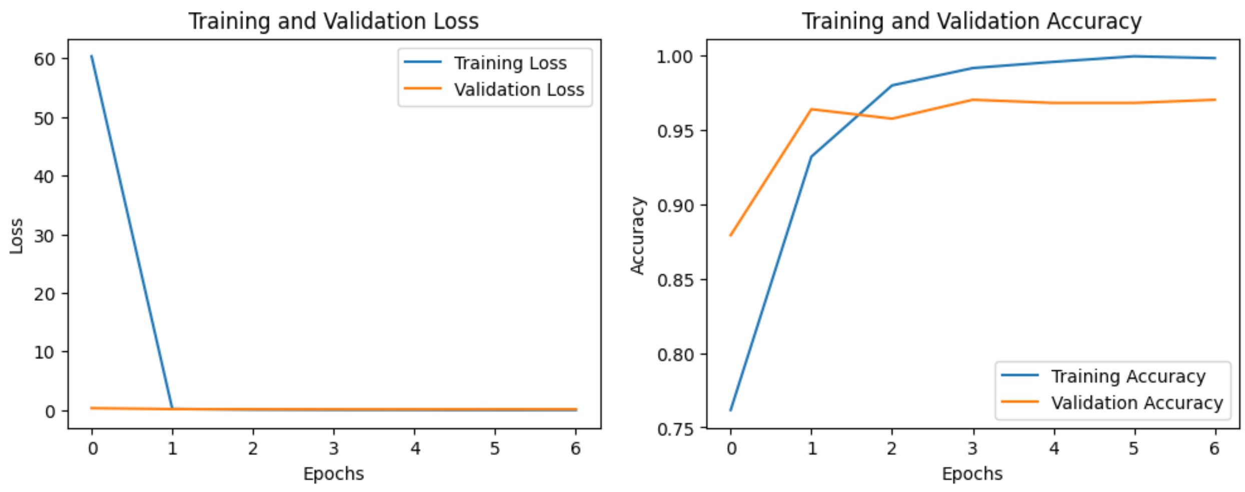
After that, the loss and accuracy of the model is checked with testing_data.



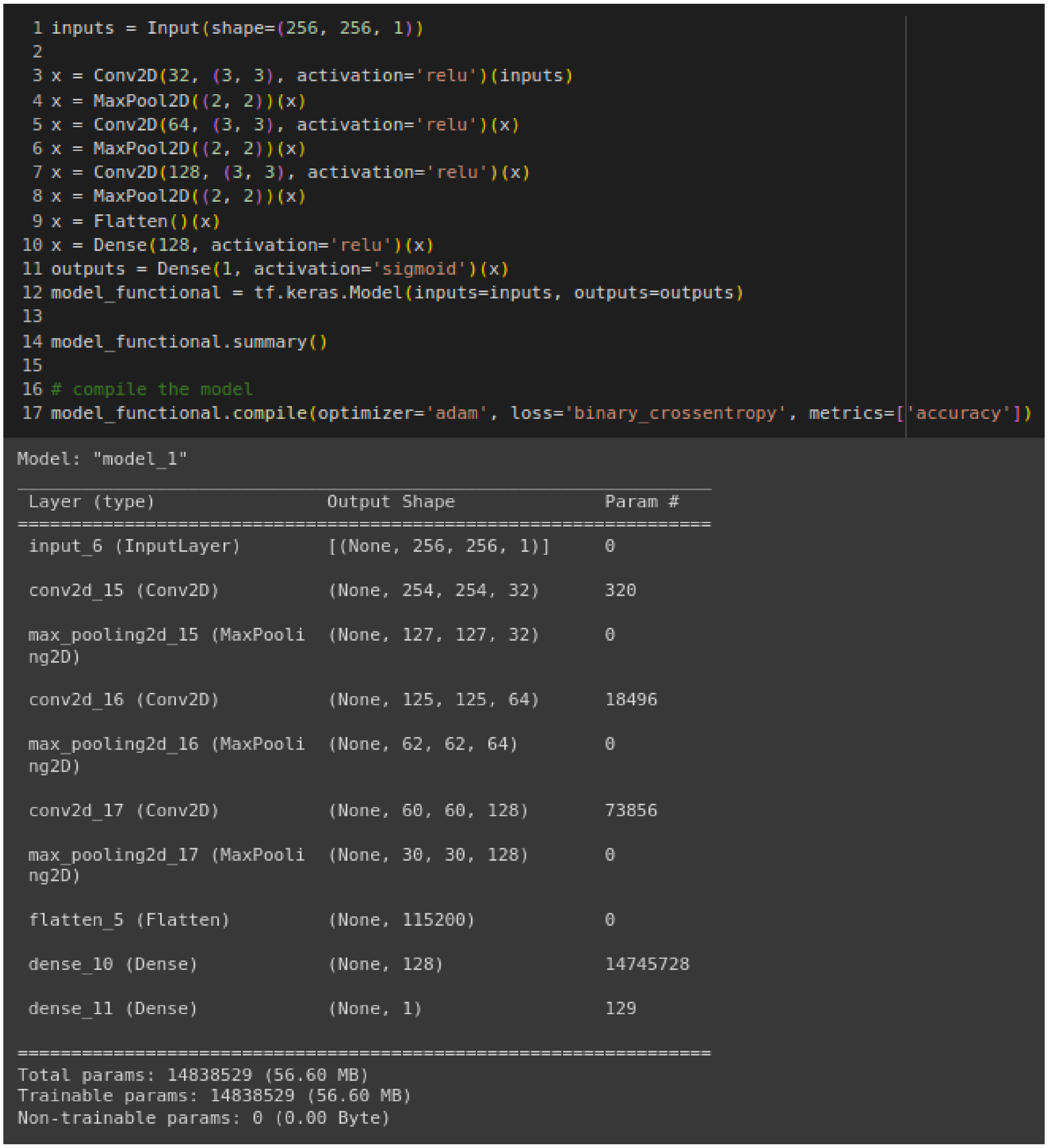
```
# Evaluate the model on the training dataset
train_loss, train_accuracy = model_sequential.evaluate(training_dataset)

# Evaluate the model on the test dataset
test_loss, test_accuracy = model_sequential.evaluate(test_dataset)

print(f"Training Accuracy: {train_accuracy:.4f}, Training Loss: {train_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}, Test Loss: {test_loss:.4f}")
```

```
38/38 [==============================] - 4s 83ms/step - loss: 0.0151 - accuracy: 0.9975
2/2 [==============================] - 0s 63ms/step - loss: 0.0466 - accuracy: 0.9844
Training Accuracy: 0.9975, Training Loss: 0.0151
Test Accuracy: 0.9844, Test Loss: 0.0466
```

And at the end the loss and accuracy should be plotted:

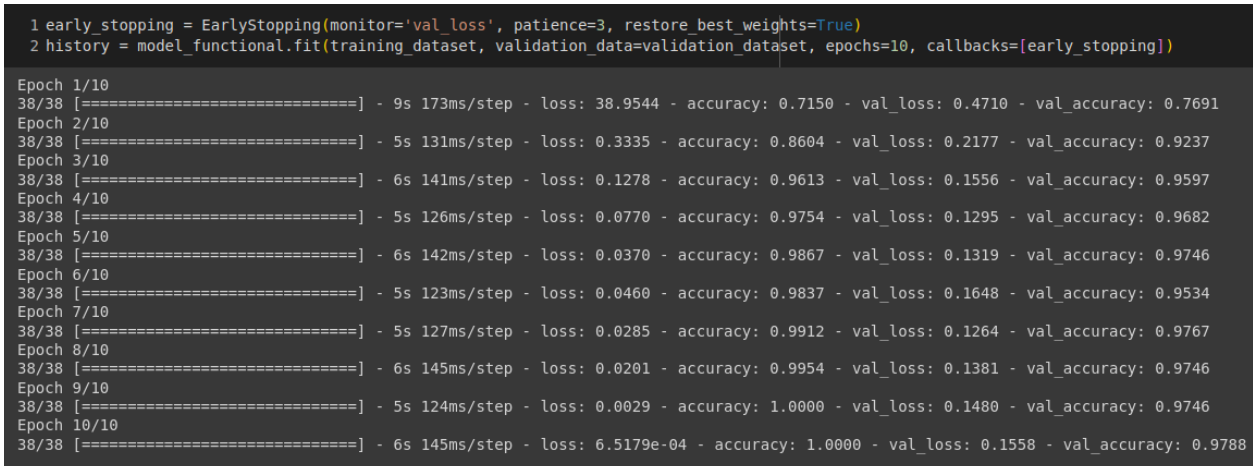


All these steps should be done for functional implementation.



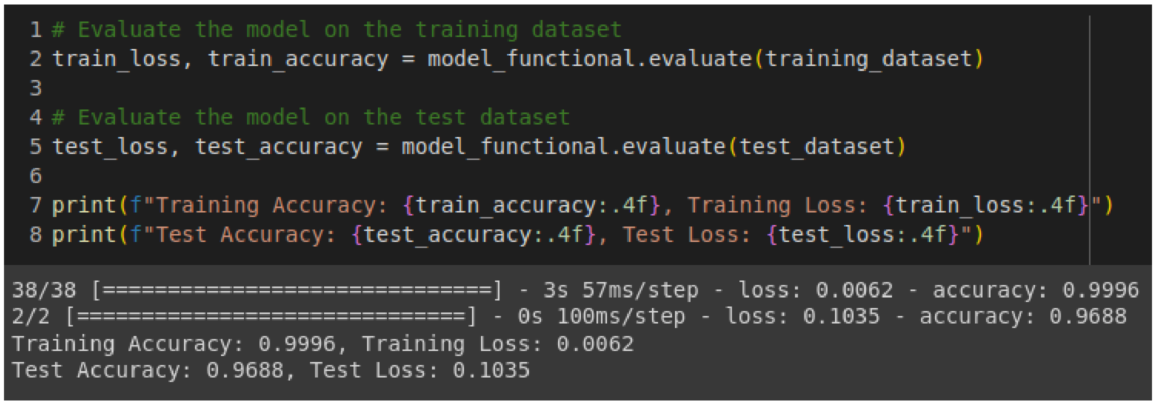
```python
inputs = Input(shape=(256, 256, 1))

x = Conv2D(32, (3, 3), activation='relu')(inputs)
x = MaxPool2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = MaxPool2D((2, 2))(x)
x = Conv2D(128, (3, 3), activation='relu')(x)
x = MaxPool2D((2, 2))(x)
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
outputs = Dense(1, activation='sigmoid')(x)
model_functional = tf.keras.Model(inputs=inputs, outputs=outputs)

model_functional.summary()

# compile the model
model_functional.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 256, 256, 1)]     0

 conv2d_15 (Conv2D)          (None, 254, 254, 32)      320

 max_pooling2d_15 (MaxPooli  (None, 127, 127, 32)      0
 ng2D)

 conv2d_16 (Conv2D)          (None, 125, 125, 64)      18496

 max_pooling2d_16 (MaxPooli  (None, 62, 62, 64)        0
 ng2D)

 conv2d_17 (Conv2D)          (None, 60, 60, 128)       73856

 max_pooling2d_17 (MaxPooli  (None, 30, 30, 128)       0
 ng2D)

 flatten_5 (Flatten)         (None, 115200)            0

 dense_10 (Dense)            (None, 128)               14745728

 dense_11 (Dense)            (None, 1)                 129

=================================================================
Total params: 14838529 (56.60 MB)
Trainable params: 14838529 (56.60 MB)
Non-trainable params: 0 (0.00 Byte)
```



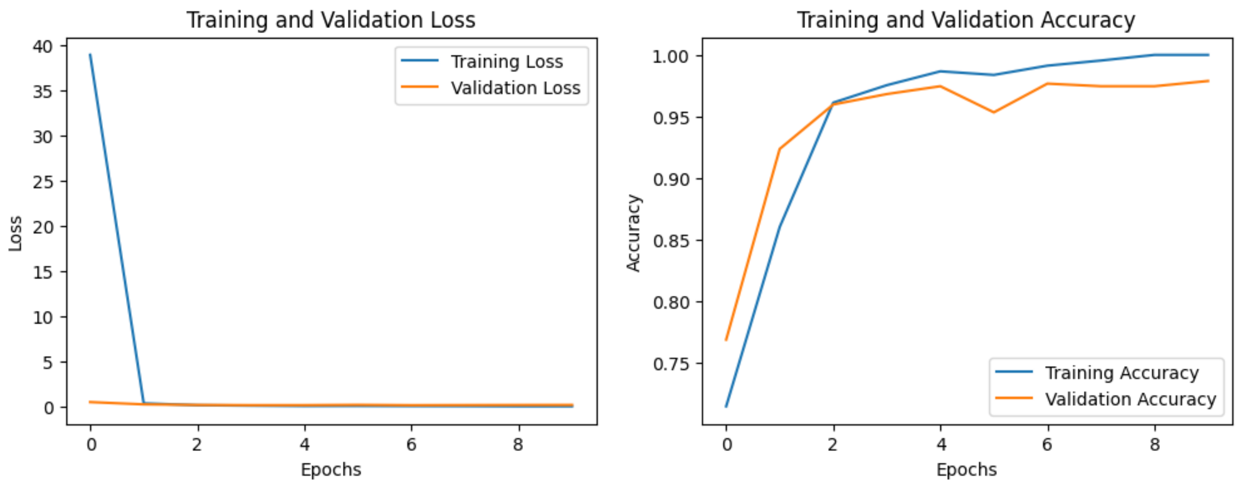
```python
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
history = model_functional.fit(training_dataset, validation_data=validation_dataset, epochs=10, callbacks=[early_stopping])
```

```
Epoch 1/10
38/38 [==============================] - 9s 173ms/step - loss: 38.9544 - accuracy: 0.7150 - val_loss: 0.4710 - val_accuracy: 0.7691
Epoch 2/10
38/38 [==============================] - 5s 131ms/step - loss: 0.3335 - accuracy: 0.8604 - val_loss: 0.2177 - val_accuracy: 0.9237
Epoch 3/10
38/38 [==============================] - 6s 141ms/step - loss: 0.1278 - accuracy: 0.9613 - val_loss: 0.1556 - val_accuracy: 0.9597
Epoch 4/10
38/38 [==============================] - 5s 126ms/step - loss: 0.0770 - accuracy: 0.9754 - val_loss: 0.1295 - val_accuracy: 0.9682
Epoch 5/10
38/38 [==============================] - 6s 142ms/step - loss: 0.0370 - accuracy: 0.9867 - val_loss: 0.1319 - val_accuracy: 0.9746
Epoch 6/10
38/38 [==============================] - 5s 123ms/step - loss: 0.0460 - accuracy: 0.9837 - val_loss: 0.1648 - val_accuracy: 0.9534
Epoch 7/10
38/38 [==============================] - 5s 127ms/step - loss: 0.0285 - accuracy: 0.9912 - val_loss: 0.1264 - val_accuracy: 0.9767
Epoch 8/10
38/38 [==============================] - 6s 145ms/step - loss: 0.0201 - accuracy: 0.9954 - val_loss: 0.1381 - val_accuracy: 0.9746
Epoch 9/10
38/38 [==============================] - 5s 124ms/step - loss: 0.0029 - accuracy: 1.0000 - val_loss: 0.1480 - val_accuracy: 0.9746
Epoch 10/10
38/38 [==============================] - 6s 145ms/step - loss: 6.5179e-04 - accuracy: 1.0000 - val_loss: 0.1558 - val_accuracy: 0.9788
```



```python
# Evaluate the model on the training dataset
train_loss, train_accuracy = model_functional.evaluate(training_dataset)

# Evaluate the model on the test dataset
test_loss, test_accuracy = model_functional.evaluate(test_dataset)

print(f"Training Accuracy: {train_accuracy:.4f}, Training Loss: {train_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}, Test Loss: {test_loss:.4f}")
```

```
38/38 [==============================] - 3s 57ms/step - loss: 0.0062 - accuracy: 0.9996
2/2 [==============================] - 0s 100ms/step - loss: 0.1035 - accuracy: 0.9688
Training Accuracy: 0.9996, Training Loss: 0.0062
Test Accuracy: 0.9688, Test Loss: 0.1035
```



As we can see the output of the both models are mostly the same because just the way of implementation are different and all the layers that are used are the same.

Question 5:

Image classification is a common application where convolutional layers are widely used in deep learning models. Convolutional Neural Networks (CNNs) are particularly effective for tasks like image classification due to their ability to capture spatial hierarchies and local patterns in the input data. For example, in the digit classification task, these local features might correspond to parts of digits like curves and straight lines. Convolutional layers excel at capturing local features in an image. They use small filters (kernels) that slide across the input image, learning to detect simple patterns like edges, corners, and textures. They are also capable of achieving translation invariance, meaning they can recognize patterns regardless of their position in the image. This is crucial for image classification, where the spatial arrangement of features may vary.

While convolutional layers are highly effective for image-related tasks, there are situations where they might have disadvantages or may not be the best choice. One such situation is when dealing with sequential data, such as speech recognition or NLP tasks.

For example, speech signals have intricate temporal dependencies, where the meaning of a sound is not only influenced by the current moment but also by the preceding and succeeding moments. Convolutional layers, designed for spatial hierarchies in images, may not capture long-term dependencies effectively in one-dimensional sequences.

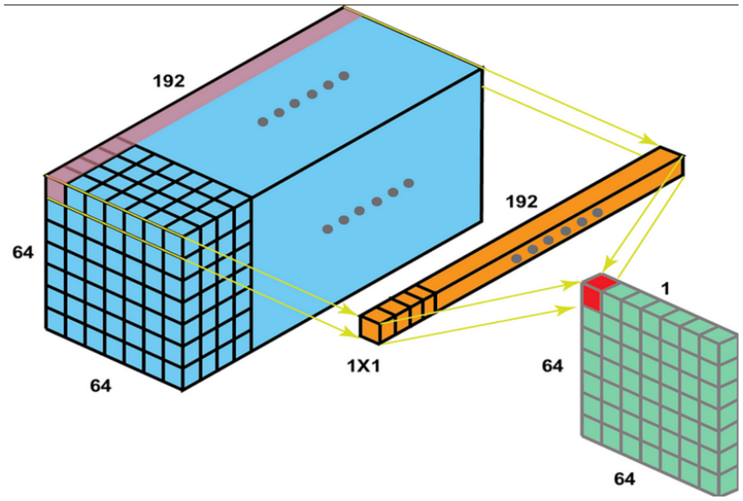
They are not inherently built to understand sequential dependencies in data. NLP tasks often involve understanding the context and relationships between words in a sentence, which is better handled by models designed with sequential processing in mind, such as Recurrent Neural Networks (RNNs) or Transformer models.

CNNs also require fixed-size input, but natural language sequences or speech signals can vary in length.

Question 6:

6-1:

In Convolutional Neural Networks (CNNs), 1x1 filters are used for various purposes, such as Dimensionality Reduction/Augmentation, Building DEEPER Network (“Bottle-Neck” Layer), and Smaller yet Accurate Model (“FIRE-MODULE” Layer). The 1x1 Convolution layer was used for ‘Cross Channel Down sampling’ or Cross Channel Pooling. In other words, 1x1 Conv was used to reduce the number of channels while introducing non-linearity. Now If we want to reduce the depth but keep the $Height \times Width$ of the feature maps the same, then we can choose 1x1 filters to achieve this effect. This effect of cross channel down-sampling is called 'Dimensionality reduction'.



But there is a point that is explained below:

Suppose we have a conv layer which outputs an (N,F,H,W) shaped tensor and the input is fed into a conv layer with F1 1x1 filters, zero padding and stride 1. Then the output of this 1x1 conv layer will have shape (N,F1,H,W). Now If F1>F then we are increasing dimensionality and if F1<F we are decreasing dimensionality, in the filter dimension.

I get help from this site.

6-2:

After using 1x1 filters in a convolutional neural network (CNN), the feature maps generated can convey several important aspects of the learned representations.

Each channel in the feature map produced by the 1x1 convolution corresponds to a different aspect or combination of features. These channels can be thought of as feature detectors or filters capturing specific patterns.

Channels in the feature maps may interact with each other. The relationships between different channels can represent higher-order feature combinations that are useful for the given task. Depending on the specific task (for example image classification), certain channels in the feature maps may be particularly important for distinguishing between different classes. These channels might focus on aspects of the input data that are discriminative for the classification task.

6-3:

The original input image contains spatial information such as pixels, edges, and textures. Each pixel in the image represents a specific location in space. Feature maps, on the other hand, capture abstract features and patterns from the input. Each element in a feature map corresponds to the activation of a filter at a particular spatial location.

Filters with different dimensions capture different types of spatial patterns. Larger filters have a broader receptive field and can capture more global features, while smaller filters focus on local patterns. 1x1 filters, being pointwise, operate at the pixel level. They don't capture spatial patterns like edges or textures on their own but are useful for learning channel-wise combinations and adjusting the depth of the feature map.

6-4:

The 1x1 convolutions have been widely adopted in various CNN architectures due to their effectiveness in adjusting depth, controlling model complexity, and improving computational efficiency. Their ability to capture channel-wise interactions and reduce dimensionality makes them a valuable tool in the design of deep neural networks. One of its usages is in the Inception architecture, introduced by the GoogLeNet model. Inception modules consist of multiple filter sizes (1x1, 3x3, 5x5), and the 1x1 convolutions are employed to reduce dimensionality and adjust the number of channels efficiently. It is also used in MobileNet which is designed for mobile and edge devices with limited computational resources. It employs depth wise separable convolutions, which include 1x1 convolutions for pointwise feature transformation, reducing the number of parameters and computations.

The MobileNet example is with the help of ChatGPt:)

6-5:

While 1x1 filters are widely used in many deep learning architectures, there are situations where they might not provide significant benefits or could even be less useful.

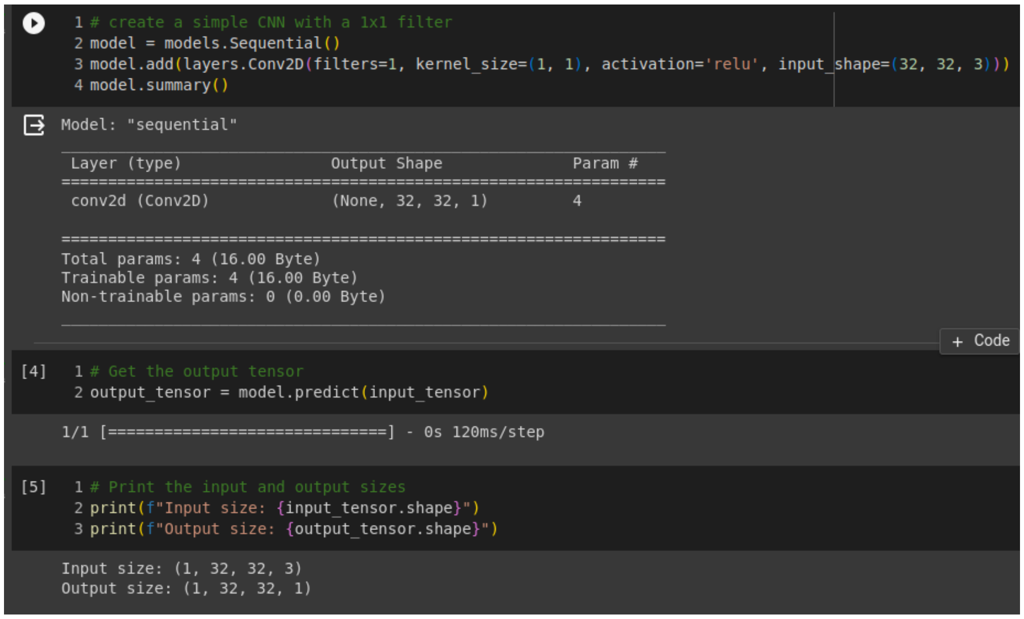
In scenarios where the input data has very small spatial dimensions, the use of 1x1 filters may not be as beneficial.

For relatively simple tasks where the input data has few distinctive features, introducing 1x1 convolutions may add unnecessary complexity to the model. In such cases, a simpler architecture without pointwise convolutions might be sufficient.

In some applications where interpretability is critical, the introduction of 1x1 convolutions can make it more challenging to understand the contribution of individual features. The use of such filters might create more abstract representations that are harder to interpret compared to simpler architectures.

6-6:

Here is a simple CNN model with 1x1 filter:



```
# create a simple CNN with a 1x1 filter
model = models.Sequential()
model.add(layers.Conv2D(filters=1, kernel_size=(1, 1), activation='relu', input_shape=(32, 32, 3)))
model.summary()
```

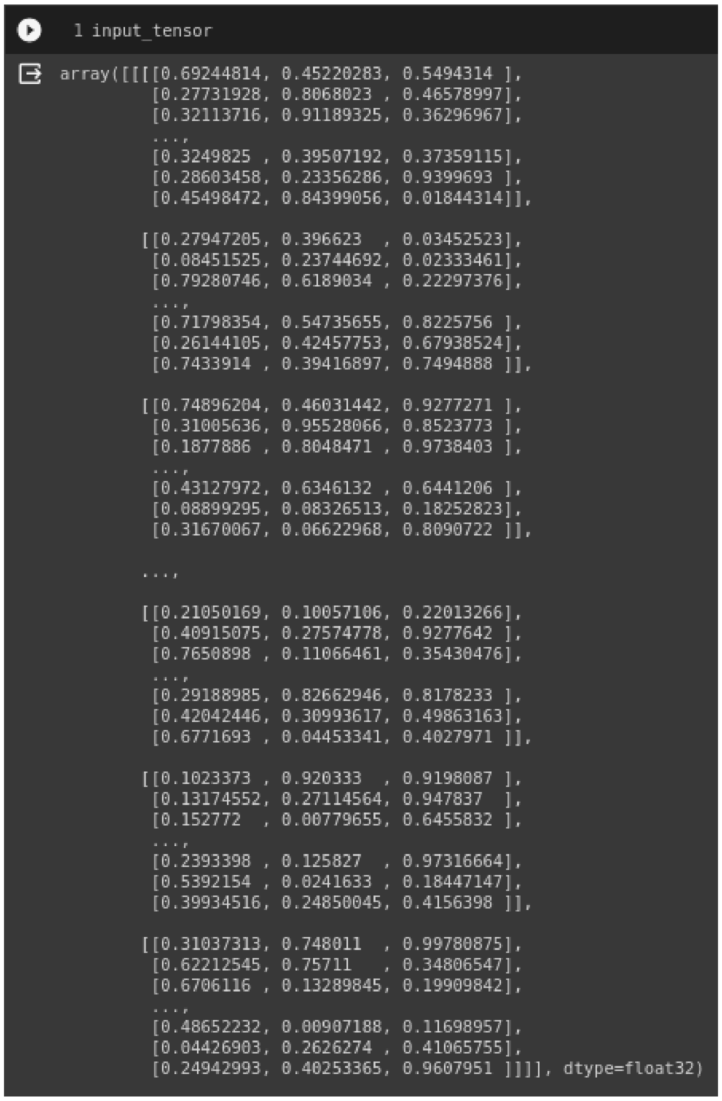
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 1)         4

=================================================================
Total params: 4 (16.00 Byte)
Trainable params: 4 (16.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[4]
```
# Get the output tensor
output_tensor = model.predict(input_tensor)
```

```
1/1 [==============================] - 0s 120ms/step
```

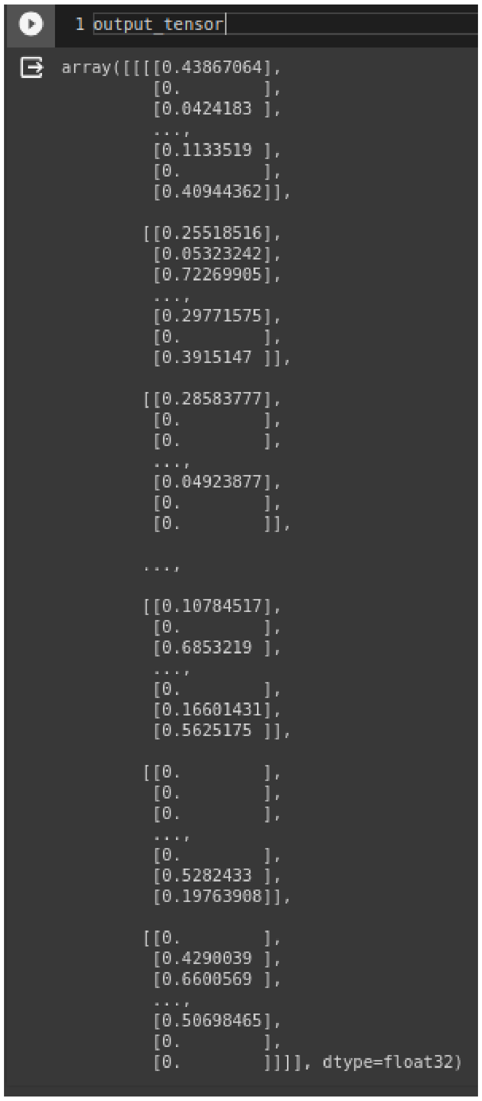
[5]
```
# Print the input and output sizes
print(f"Input size: {input_tensor.shape}")
print(f"Output size: {output_tensor.shape}")
```

```
Input size: (1, 32, 32, 3)
Output size: (1, 32, 32, 1)
```



```
input_tensor
```

```
array([[[[0.69244814, 0.45220283, 0.5494314 ],
         [0.27731928, 0.8068023 , 0.46578997],
         [0.32113716, 0.91189325, 0.36296967],
         ...,
         [0.3249825 , 0.39507192, 0.37359115],
         [0.28603458, 0.23356286, 0.9399693 ],
         [0.45498472, 0.84399056, 0.01844314]],

        [[0.27947205, 0.396623  , 0.03452523],
         [0.08451525, 0.23744692, 0.02333461],
         [0.79280746, 0.6189034 , 0.22297376],
         ...,
         [0.71798354, 0.54735655, 0.8225756 ],
         [0.26144105, 0.42457753, 0.67938524],
         [0.7433914 , 0.39416897, 0.7494888 ]],

        [[0.74896204, 0.46031442, 0.9277271 ],
         [0.31005636, 0.95528066, 0.8523773 ],
         [0.1877886 , 0.8048471 , 0.9738403 ],
         ...,
         [0.43127972, 0.6346132 , 0.6441206 ],
         [0.08899295, 0.08326513, 0.18252823],
         [0.31670067, 0.06622968, 0.8090722 ]],

        ...,

        [[0.21050169, 0.10057106, 0.22013266],
         [0.40915075, 0.27574778, 0.9277642 ],
         [0.7650898 , 0.11066461, 0.35430476],
         ...,
         [0.29188985, 0.82662946, 0.8178233 ],
         [0.42042446, 0.30993617, 0.49863163],
         [0.6771693 , 0.04453341, 0.4027971 ]],

        [[0.1023373 , 0.920333  , 0.9198087 ],
         [0.13174552, 0.27114564, 0.947837  ],
         [0.152772  , 0.00779655, 0.6455832 ],
         ...,
         [0.2393398 , 0.125827  , 0.97316664],
         [0.5392154 , 0.0241633 , 0.18447147],
         [0.39934516, 0.24850045, 0.4156398 ]],

        [[0.31037313, 0.748011  , 0.99780875],
         [0.62212545, 0.75711   , 0.34806547],
         [0.6706116 , 0.13289845, 0.19909842],
         ...,
         [0.48652232, 0.00907188, 0.11698957],
         [0.04426903, 0.2626274 , 0.41065755],
         [0.24942993, 0.40253365, 0.9607951 ]]]], dtype=float32)
```



```
output_tensor
```

```
array([[[[0.43867064],
         [0.        ],
         [0.0424183 ],
         ...,
         [0.1133519 ],
         [0.        ],
         [0.40944362]],

        [[0.25518516],
         [0.05323242],
         [0.72269905],
         ...,
         [0.29771575],
         [0.        ],
         [0.3915147 ]],

        [[0.28583777],
         [0.        ],
         [0.        ],
         ...,
         [0.04923877],
         [0.        ],
         [0.        ]],

        ...,

        [[0.10784517],
         [0.        ],
         [0.6853219 ],
         ...,
         [0.        ],
         [0.16601431],
         [0.5625175 ]],

        [[0.        ],
         [0.        ],
         [0.        ],
         ...,
         [0.        ],
         [0.5282433 ],
         [0.19763908]],

        [[0.        ],
         [0.4290039 ],
         [0.6600569 ],
         ...,
         [0.50698465],
         [0.        ],
         [0.        ]]]], dtype=float32)
```

As it’s clear from the results, the 1x1 filter decreases the dimension of the input and combines the features. So the input with 32x32x3 converts to 32x32x1.

Question 7:

- The Inception module, also known as GoogLeNet, is designed to address the challenge of finding the optimal filter size for convolutional layers. Instead of relying on a single filter size, Inception uses a combination of filters with different receptive field sizes within the same module. This allows the network to capture features at multiple scales and helps improve its ability to recognize complex patterns in the data. The use of 1x1 convolutions helps in reducing the number of parameters, making the model more parameter-efficient. Also networks with Inception Modules have shown improved performance for image recognition and classification tasks.
- The stride in Convolutional Neural Networks (CNNs) is a hyperparameter that determines the step size the convolutional filter takes as it slides/spatially moves across the input volume. Increasing the stride value reduces the spatial dimensions of the feature maps. A larger stride means the filter skips more pixels when moving across the input, leading to a smaller output volume. It results in a larger receptive field for each neuron in the feature map. The receptive field is the region in the input space that contributes to the computation of a particular neuron in the output. A larger receptive field can be beneficial for capturing larger patterns in the input. Using a stride greater than 1 is a form of downsampling. It reduces the spatial resolution of the feature maps, which can be useful in situations where reducing spatial dimensions is desired. Adjusting the stride is often done in conjunction with adjusting other hyperparameters like zero-padding and filter

size to achieve the desired behavior in the network.

- UpSampling2D increases the spatial dimensions of the input feature maps by replicating values leading to generating finer details in feature maps.

The base model, InceptionV3 in this case, is a pre-trained convolutional neural network that serves as a feature extractor. Transfer learning using a pre-trained model allows the network to leverage knowledge gained from a large dataset (ImageNet in this case). The base model extracts complex hierarchical features from the input images, which can be valuable for various computer vision tasks.

The Flatten layer converts the multi-dimensional feature maps into a one-dimensional vector. It reshapes the output of the convolutional layers into a form that can be processed by dense layers.

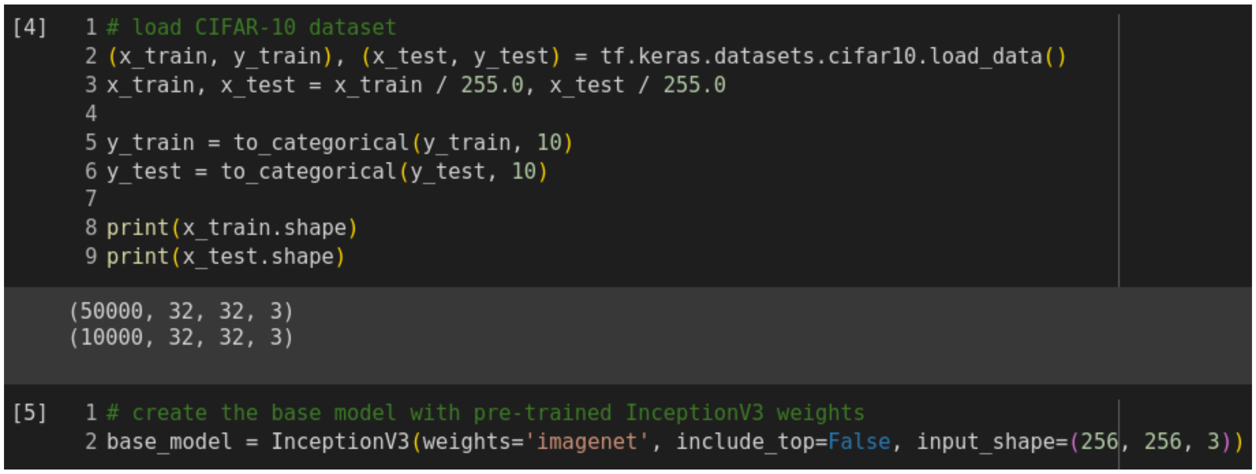
Dense layers are fully connected layers that capture high-level abstract features by combining information from the flattened feature maps. ReLU introduces non-linearity, allowing the model to learn complex relationships between features.

Dropout layers randomly set a fraction of input units to zero during training to prevent overfitting. It is a regularization technique that helps the model to prevent from relying on specific neurons. It encourages the network to learn more robust and generalizable features.

BatchNormalization normalizes the input by adjusting and scaling the activations during training. It helps stabilize and accelerate training.

The Softmax activation function is applied to the final layer, providing normalized class probabilities. It converts the raw model outputs into probabilities, making it suitable for multi-class classification.

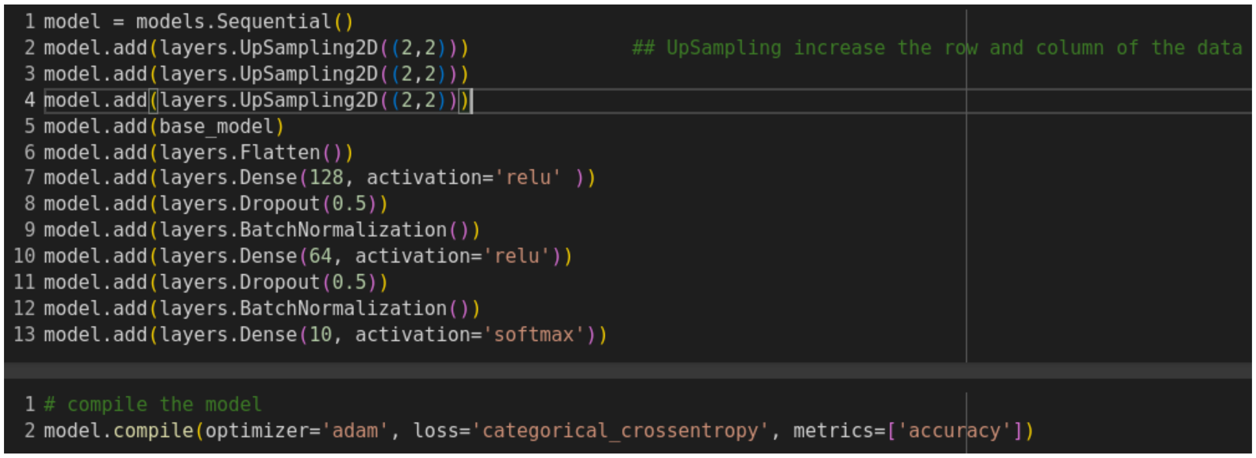
- Here is my implementation for this task:



```
[4]
# load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

print(x_train.shape)
print(x_test.shape)
```

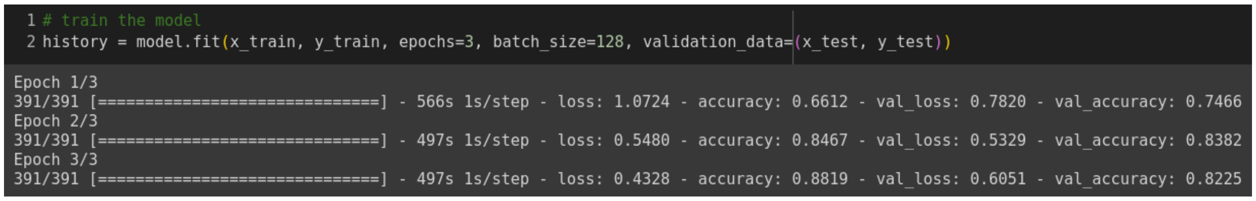
```
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

```
[5]
# create the base model with pre-trained InceptionV3 weights
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(256, 256, 3))
```



```
model = models.Sequential()
model.add(layers.UpSampling2D((2,2)))          ## UpSampling increase the row and column of the data
model.add(layers.UpSampling2D((2,2)))
model.add(layers.UpSampling2D((2,2)))
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu' ))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(10, activation='softmax'))
```

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



```
# train the model
history = model.fit(x_train, y_train, epochs=3, batch_size=128, validation_data=(x_test, y_test))
```

```
Epoch 1/3
391/391 [==============================] - 566s 1s/step - loss: 1.0724 - accuracy: 0.6612 - val_loss: 0.7820 - val_accuracy: 0.7466
Epoch 2/3
391/391 [==============================] - 497s 1s/step - loss: 0.5480 - accuracy: 0.8467 - val_loss: 0.5329 - val_accuracy: 0.8382
Epoch 3/3
391/391 [==============================] - 497s 1s/step - loss: 0.4328 - accuracy: 0.8819 - val_loss: 0.6051 - val_accuracy: 0.8225
```

I get help from [this site.]