Reyhane Shahrokhian 99521361

HomeWork6 of Computer Vision Course

Dr. Mohammadi

Q1:

1-1:

We should know how to compute parameters and output_shapes:

Conv:

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

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

If *padding=same:* $Output\_width = \frac{Input\_width}{stride}$

$Output\_channel = filters$

Pooling:

$Parameters = 0$

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

$Output\_channel = Input\_channel$

Fully connected:

$Parameters = M \times N + N$

Layers	Output_shape	Parameters
Input(shape=(512, 512, 3))	(512, 512, 3)	0
Conv2D(32,(9,9),stride=2,padding='same', activation='relu')	(256, 256, 32)	$32 \times (9 \times 9 \times 3 + 1) = 7808$
MaxPooling2D((4,4),stride=4)	(64, 64, 32)	0
Conv2D(64,(5,5),stride=1)	(60, 60, 64)	$64 \times (5 \times 5 \times 32 + 1) = 51264$
AveragePooling2D((2,2),stride=2)	(30, 30, 64)	0
Conv2D(128,(3,3),stride=1,padding='valid', activation='relu')	(28, 28, 128)	$128 \times (3 \times 3 \times 64 + 1) = 73856$
Conv2D(128,(3,3),stride=1,padding='same', activation='relu')	(28, 28, 128)	$128 \times (3 \times 3 \times 128 + 1) = 147584$
MaxPooling2D((2,2),stride=2)	(14, 14, 128)	0
Conv2D(512,(3,3),stride=1,padding='valid', activation='relu')	(12, 12, 512)	$512 \times (3 \times 3 \times 128 + 1) = 590336$
GlobalAveragePooling2D()	512	0
Dense(1024)	1024	$512 \times 1024 + 1024 = 525312$
Dense(10)	10	$1024 \times 10 + 10 = 10250$

1-2:

We should know how to compute number of multiplications and additions:

Conv:

$multiplications = H_{out} \times W_{out} \times C_{out} \times C_{in} \times k_h \times k_w$

$additions = H_{out} \times W_{out} \times C_{out} \times (C_{in} \times k_h \times k_w - 1)$

Pooling:

$$multiplications = 0$$

$$additions = H_{out} \times W_{out} \times C_{out} \times (P \times P - 1)$$

Fully connected:

$$multiplications = n_{out} \times n_{in}$$

$$additions = n_{out} \times n_{in} + n_{out}$$

Layers	**Number of multiplications**	**Number of additions**
Input(shape=(512, 512, 3))	-	-
Conv2D(32,(9,9),stride=2,padding='same', activation='relu')	$256 \times 256 \times 32 \times 3 \times 9 \times 9$	$256 \times 256 \times 32 \times (3 \times 9 \times 9 - 1)$
MaxPooling2D((4,4),stride=4)	0	$64 \times 64 \times 32 \times (4 \times 4 - 1)$
Conv2D(64,(5,5),stride=1)	$60 \times 60 \times 64 \times 32 \times 5 \times 5$	$60 \times 60 \times 64 \times (32 \times 5 \times 5 - 1)$
AveragePooling2D((2,2),stride=2)	0	$30 \times 30 \times 64 \times (2 \times 2 - 1)$
Conv2D(128,(3,3),stride=1,padding='valid', activation='relu')	$28 \times 28 \times 128 \times 64 \times 3 \times 3$	$28 \times 28 \times 128 \times (64 \times 3 \times 3 - 1)$
Conv2D(128,(3,3),stride=1,padding='same', activation='relu')	$28 \times 28 \times 128 \times 128 \times 3 \times 3$	$28 \times 28 \times 128 \times (128 \times 3 \times 3 - 1)$
MaxPooling2D((2,2),stride=2)	0	$14 \times 14 \times 128 \times (4 \times 4 - 1)$
Conv2D(512,(3,3),stride=1,padding='valid', activation='relu')	$12 \times 12 \times 512 \times 128 \times 3 \times 3$	$12 \times 12 \times 512 \times (128 \times 3 \times 3 - 1)$
GlobalAveragePooling2D()	0	$512 \times (12 \times 12 - 1)$
Dense(1024)	512×1024	$512 \times 1024 + 1024$
Dense(10)	1024×10	$1024 \times 10 + 10$

1-3:

The last three layers with Flatten:

Flatten()	73728	0
Dense(1024)	1024	$73728 \times 1024 + 1024 = 75498496$
Dense(10)	10	$1024 \times 10 + 10 = 10250$

Total parameters with Flatten: 76379594

Total parameter with GAP: 1406410

So: $\frac{76379594}{1406410} = 54.31$

Q2:

First, we should know the gradient of L with respect to X:

$\frac{dL}{dX} = 2X - 10$

So the gradient with X = 20 is 30.

To update the value of X according to gradient descent algorithm we can use this formula:

$X_{new} = X_{old} - \eta \frac{dL}{dX}$

Now we can check the given options:

1. $X_{new} = 14 \Rightarrow 14 = 20 - \eta \times 30 \Rightarrow \eta = 0.2$
2. $X_{new} = 19.4 \Rightarrow 19.4 = 20 - \eta \times 30 \Rightarrow \eta = 0.02$
3. $X_{new} = 19.94 \Rightarrow 19.94 = 20 - \eta \times 30 \Rightarrow \eta = 0.002$

We know that the larger learning rate leads to a faster decrease of the loss value. So the first one corresponds to the c, the second one corresponds to the b, and the third one corresponds to the a.

Q3:

I considered all of the blocks with the same padding.

Conv 1×1 ⇒ (12, 12, 64)

Conv 1×1 and Conv 3×3 ⇒ (12, 12, 32)

Conv 1×1 and Conv 5×5 ⇒ (12, 12, 128)

max-pool 3×3 and Conv 1×1 ⇒ (12, 12, 64)

Output after concatenating them: (12, 12, 288)

If we change the chosen convolution block filter, the output would not be different because after that block we have another convolution block causing the output the same.

Q4:

4-1:

The dataset contains t samples which are divided into b batches. So the updates count is:

$\frac{t}{b} \times e$

4-2:

It's related to the mini-batch gradient descent as in this method we are having updates on each batch and the noises on the graph are showing us exactly this.

4-3:

In curve A, the validation loss is much more than the training loss which indicates the probability of overfitting. So we can use the first and third options to reduce the overfitting.

In curve B, the validation loss and training loss are near to each other meaning that there is no overfitting, but the training loss value can get less by making the model more complex. So we can use the second option here.

Q5:

5-1:

For each non-zero pixel we should consider its 8-connectivity neighbors. Then, each neighbor should be compared to the central pixel to check if it is larger or equal to it and change it to 1 or it is smaller so convert it to 0. Then we would have a binary code from a clockwise order.

(1, 1): 00000000

(1, 2): 00000001

(1, 3): 00000111

(2, 1): 01100100

(2, 2): 11000011

(2, 3): 10000001

(3, 1): 00000000

(3, 2): 11110001

(3, 3): 11000000

5-2:

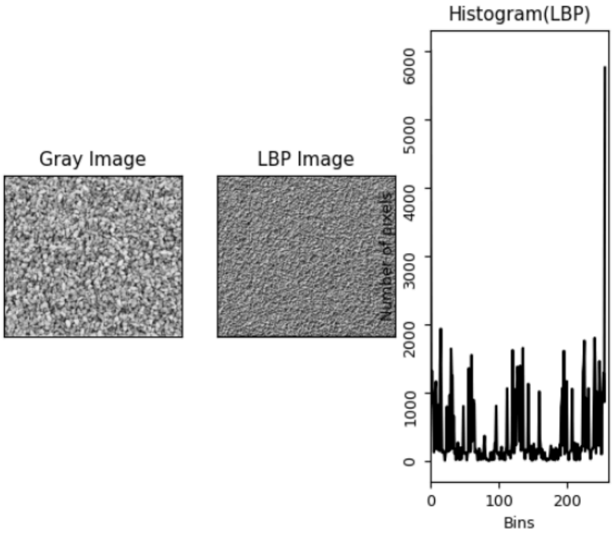
As C is greater than 0, there would be no difference if we add or multiply it to all values.

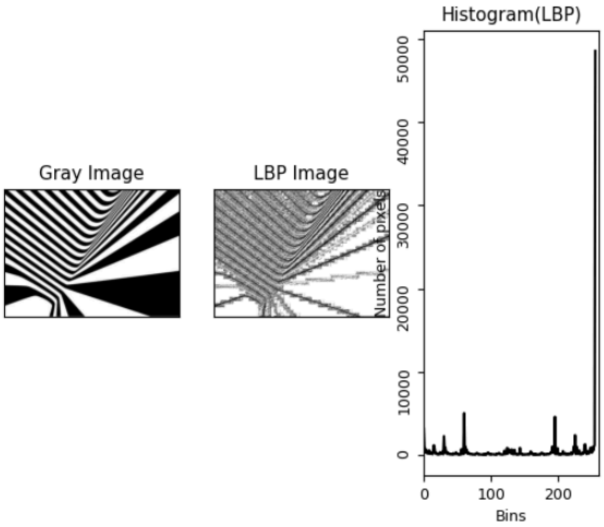
5-3:

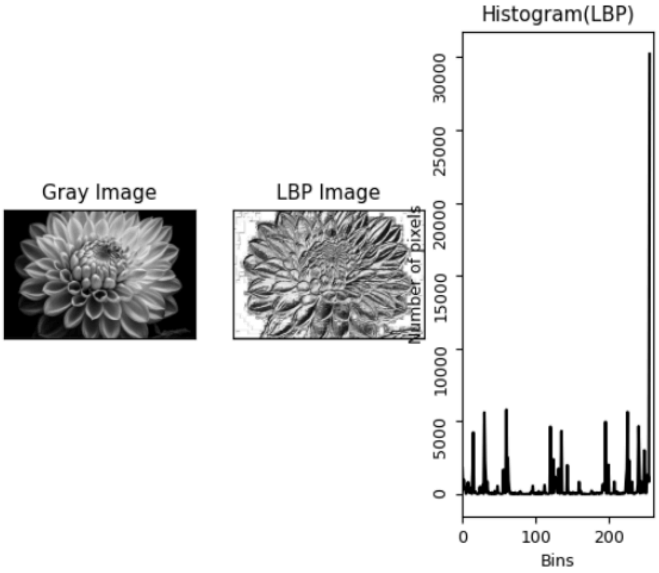
The A LBP histogram corresponds to the third image as it consists of many black and white and gray pixels causing the more uniform histogram.

The B LBP histogram corresponds to the first image because if we want to convert the image to LBP, it would consist of many white pixels(the black ones in the image) and many gray pixels. The C LBP histogram corresponds to the second image as the LBP image would consist of many white pixels and a little black or gray ones showing the line which colors had a change.

I also check that with a python code:



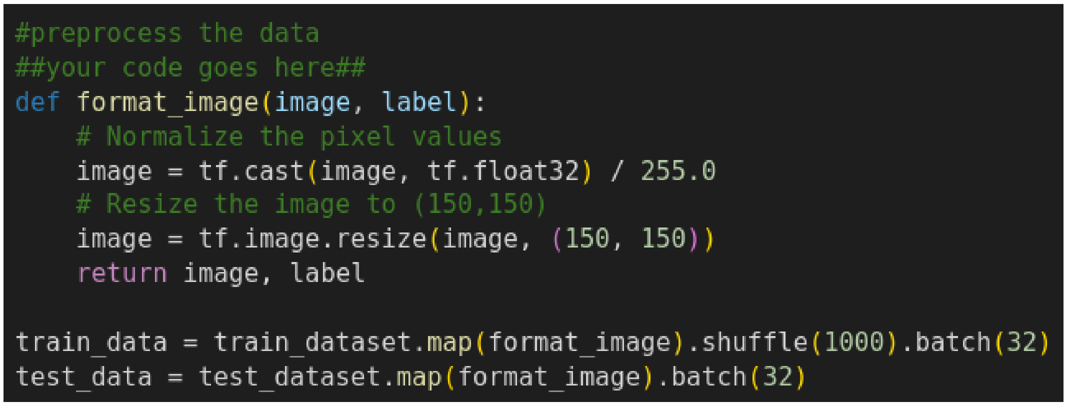




Q6:

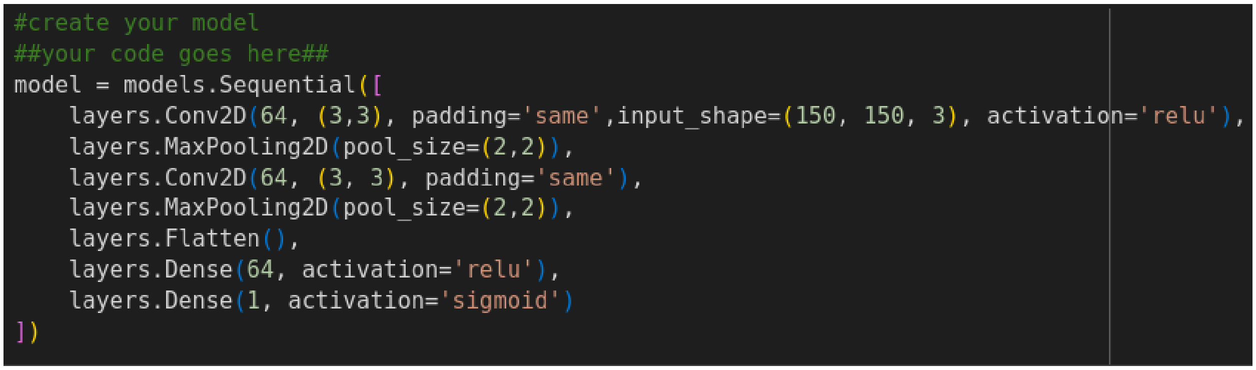
6-1:

After loading the dataset, I preprocess the images by normalizing and resizing them to (150, 150).



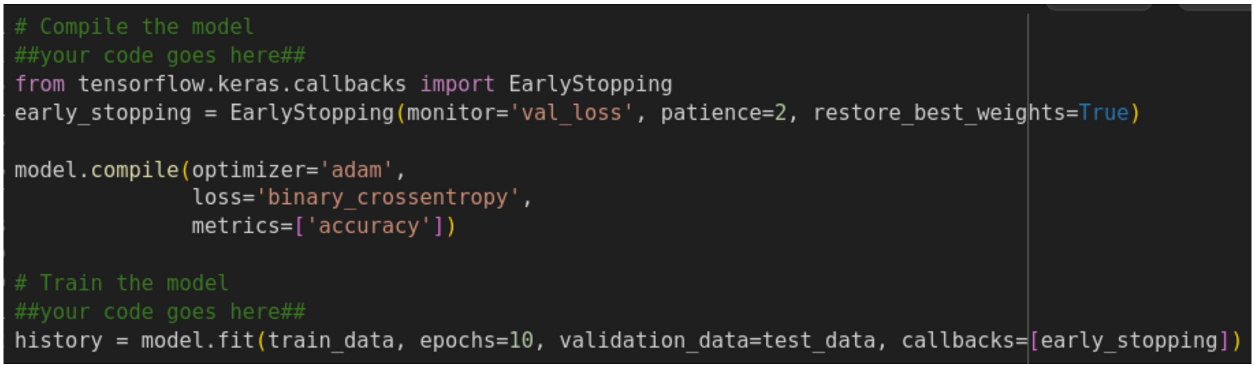
```
#preprocess the data
##your code goes here##
def format_image(image, label):
    # Normalize the pixel values
    image = tf.cast(image, tf.float32) / 255.0
    # Resize the image to (150,150)
    image = tf.image.resize(image, (150, 150))
    return image, label

train_data = train_dataset.map(format_image).shuffle(1000).batch(32)
test_data = test_dataset.map(format_image).batch(32)
```

Then, I create the model with 2 convolutional layers and using max pooling between them and at the end, a flatten layer is added to make it ready for dense layers. The last dense layer shows that the output is just 1 neuron(0 or 1).

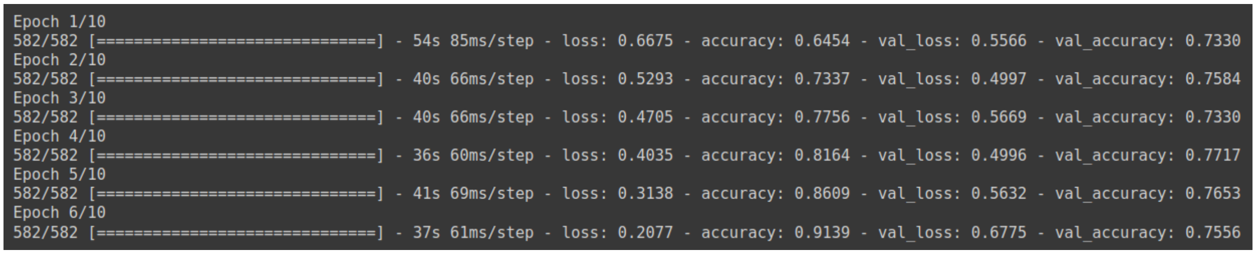


```
#create your model
##your code goes here##
model = models.Sequential([
    layers.Conv2D(64, (3,3), padding='same',input_shape=(150, 150, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Conv2D(64, (3, 3), padding='same'),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

After that, the model is compiled and then trained. I also use an early stop method which stops the training whenever the validation loss starts to decrease meaning that the model is overfitted.

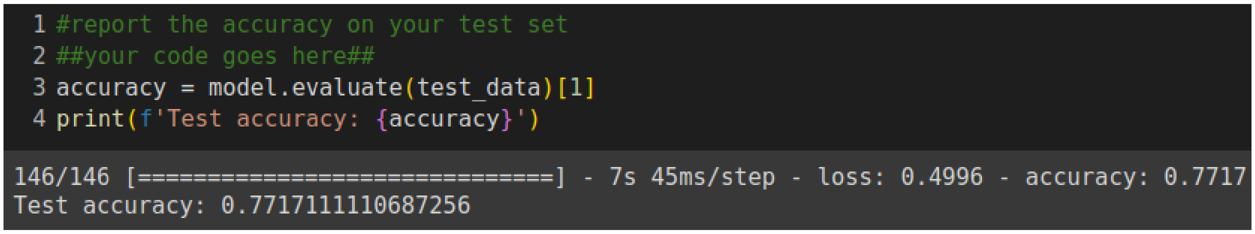


```
# Compile the model
##your code goes here##
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)

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

# Train the model
##your code goes here##
history = model.fit(train_data, epochs=10, validation_data=test_data, callbacks=[early_stopping])
```



```
Epoch 1/10
582/582 [==============================] - 54s 85ms/step - loss: 0.6675 - accuracy: 0.6454 - val_loss: 0.5566 - val_accuracy: 0.7330
Epoch 2/10
582/582 [==============================] - 40s 66ms/step - loss: 0.5293 - accuracy: 0.7337 - val_loss: 0.4997 - val_accuracy: 0.7584
Epoch 3/10
582/582 [==============================] - 40s 66ms/step - loss: 0.4705 - accuracy: 0.7756 - val_loss: 0.5669 - val_accuracy: 0.7330
Epoch 4/10
582/582 [==============================] - 36s 60ms/step - loss: 0.4035 - accuracy: 0.8164 - val_loss: 0.4996 - val_accuracy: 0.7717
Epoch 5/10
582/582 [==============================] - 41s 69ms/step - loss: 0.3138 - accuracy: 0.8609 - val_loss: 0.5632 - val_accuracy: 0.7653
Epoch 6/10
582/582 [==============================] - 37s 61ms/step - loss: 0.2077 - accuracy: 0.9139 - val_loss: 0.6775 - val_accuracy: 0.7556
```

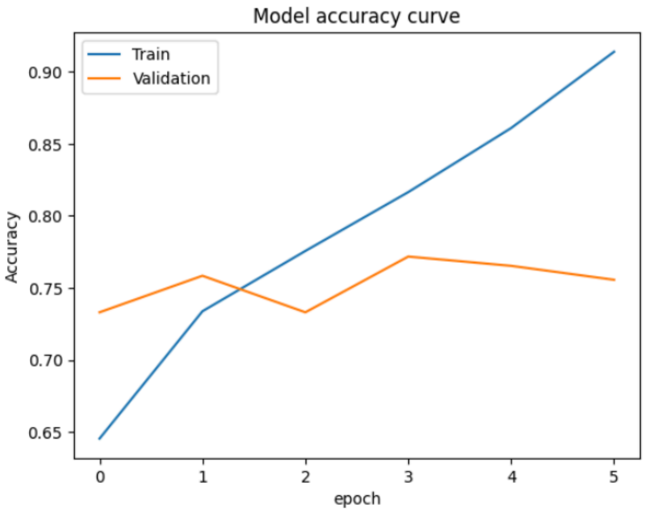
The accuracy on the test set:

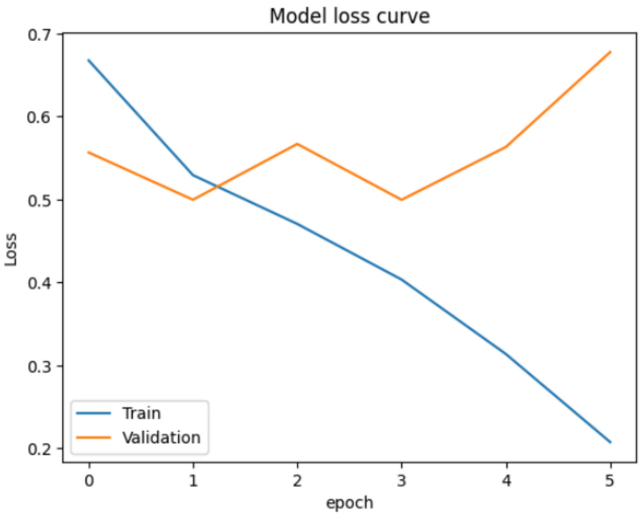


```
#report the accuracy on your test set
##your code goes here##
accuracy = model.evaluate(test_data)[1]
print(f'Test accuracy: {accuracy}')
```

```
146/146 [==============================] - 7s 45ms/step - loss: 0.4996 - accuracy: 0.7717
Test accuracy: 0.7717111110687256
```

Plots:

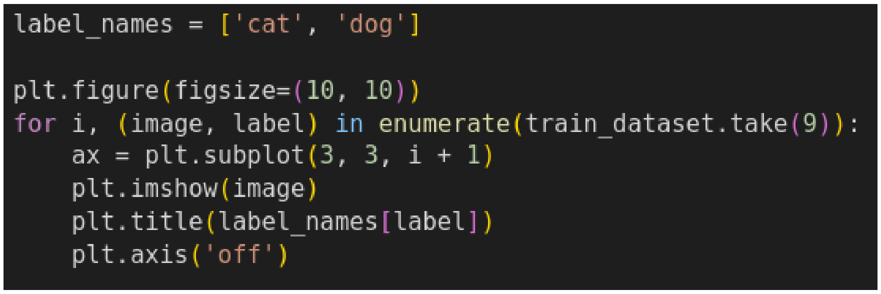




6-2:

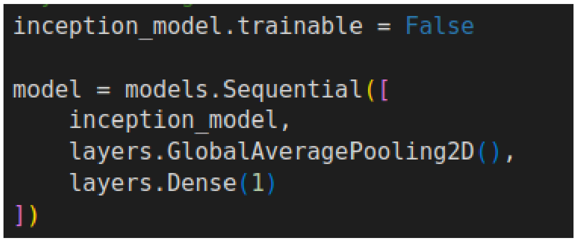
Preprocessing the data is exactly the same as the previous part.

Showing some of images with their labels:



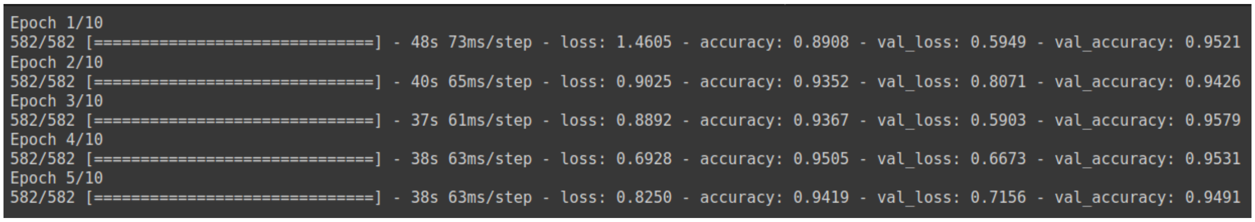
```
label_names = ['cat', 'dog']

plt.figure(figsize=(10, 10))
for i, (image, label) in enumerate(train_dataset.take(9)):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(image)
    plt.title(label_names[label])
    plt.axis('off')
```

To use the inception model as pretrained part, its parameters should be freezed. Then in the model after the inception part, I added a GAP and then a dense layer with 1 neurun for the output.



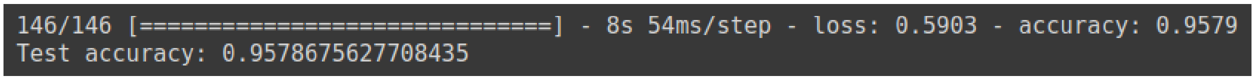
```
inception_model.trainable = False

model = models.Sequential([
    inception_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(1)
])
```

The rest is again like the previous section.



```
Epoch 1/10
582/582 [==============================] - 48s 73ms/step - loss: 1.4605 - accuracy: 0.8908 - val_loss: 0.5949 - val_accuracy: 0.9521
Epoch 2/10
582/582 [==============================] - 40s 65ms/step - loss: 0.9025 - accuracy: 0.9352 - val_loss: 0.8071 - val_accuracy: 0.9426
Epoch 3/10
582/582 [==============================] - 37s 61ms/step - loss: 0.8892 - accuracy: 0.9367 - val_loss: 0.5903 - val_accuracy: 0.9579
Epoch 4/10
582/582 [==============================] - 38s 63ms/step - loss: 0.6928 - accuracy: 0.9505 - val_loss: 0.6673 - val_accuracy: 0.9531
Epoch 5/10
582/582 [==============================] - 38s 63ms/step - loss: 0.8250 - accuracy: 0.9419 - val_loss: 0.7156 - val_accuracy: 0.9491
```

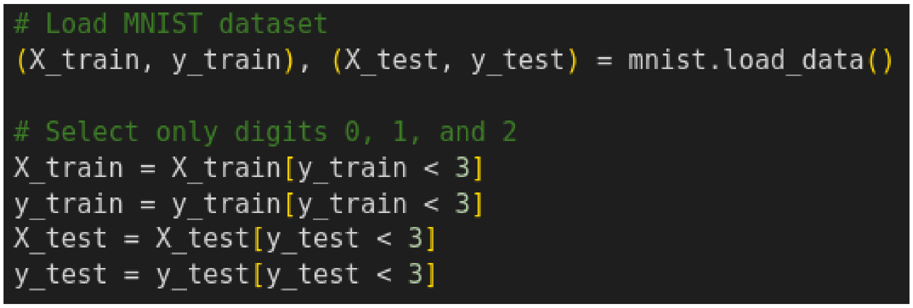
Test accuracy:



```
146/146 [==============================] - 8s 54ms/step - loss: 0.5903 - accuracy: 0.9579
Test accuracy: 0.9578675627708435
```

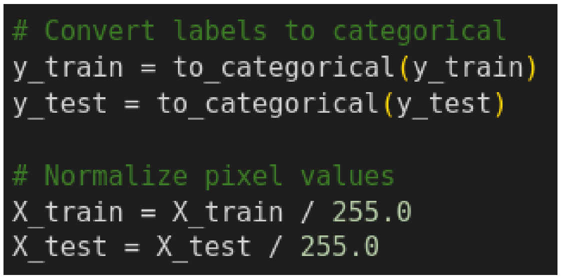
Q7:

The MNIST dataset is loaded using keras.datasets.mnist.load_data() which contains images of handwritten digits from 0 to 9 but I select only the digits 0, 1, and 2 from the dataset by filtering the labels.



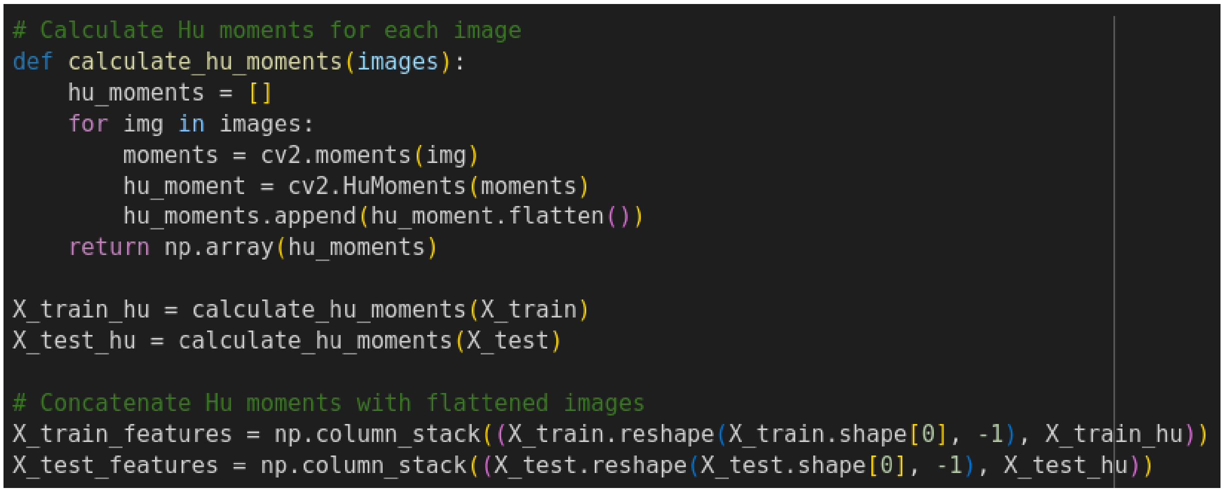
```
# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Select only digits 0, 1, and 2
X_train = X_train[y_train < 3]
y_train = y_train[y_train < 3]
X_test = X_test[y_test < 3]
y_test = y_test[y_test < 3]
```

Then, the labels are converted to categorical format using keras.utils.to_categorical() because it is necessary for the Random Forest Classifier, which expects categorical labels. The pixel values of the images are then normalized to the range by dividing each pixel value by 255.



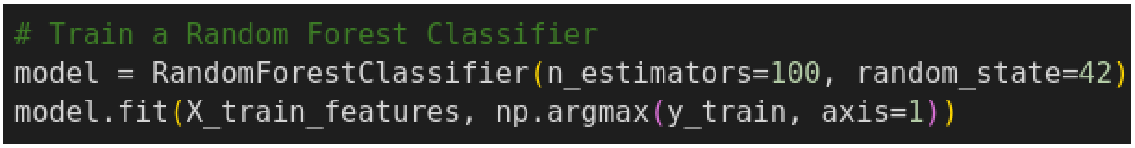
```
# Convert labels to categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Normalize pixel values
X_train = X_train / 255.0
X_test = X_test / 255.0
```

The calculate_hu_moments() function calculates the Hu moments for each image in the dataset. Hu moments are invariant to image transformations such as rotation, scaling, and skewness. This means that even if the image is rotated or scaled, the Hu moments will remain the same. The Hu moments are concatenated with the flattened images to create new feature vectors. This is done to combine the spatial information from the images with the Hu moments.



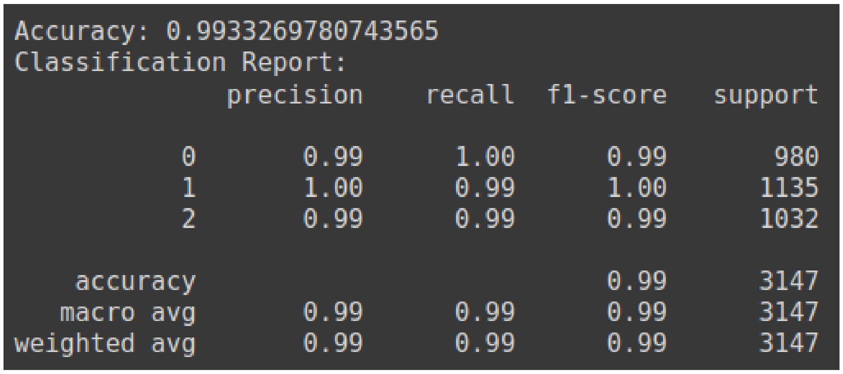
```
# Calculate Hu moments for each image
def calculate_hu_moments(images):
    hu_moments = []
    for img in images:
        moments = cv2.moments(img)
        hu_moment = cv2.HuMoments(moments)
        hu_moments.append(hu_moment.flatten())
    return np.array(hu_moments)

X_train_hu = calculate_hu_moments(X_train)
X_test_hu = calculate_hu_moments(X_test)

# Concatenate Hu moments with flattened images
X_train_features = np.column_stack((X_train.reshape(X_train.shape[0], -1), X_train_hu))
X_test_features = np.column_stack((X_test.reshape(X_test.shape[0], -1), X_test_hu))
```

A Random Forest Classifier is trained using the concatenated feature vectors and the categorical labels. The classifier is configured to use 100 trees and a random state of 42 for reproducibility.



```
# Train a Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_features, np.argmax(y_train, axis=1))
```

The accuracy and classification report:



```
Accuracy: 0.9933269780743565
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       980
           1       1.00      0.99      1.00      1135
           2       0.99      0.99      0.99      1032

    accuracy                           0.99      3147
   macro avg       0.99      0.99      0.99      3147
weighted avg       0.99      0.99      0.99      3147
```

Some random images with their predicted labels:

Predicted: 1
Predicted: 1
Predicted: 2
Predicted: 2
Predicted: 1