Homework 10 (due: Apr 28)

Machine Learning - COSC 4360

Department of Computer Science and Electrical Engineering

Spring 2025

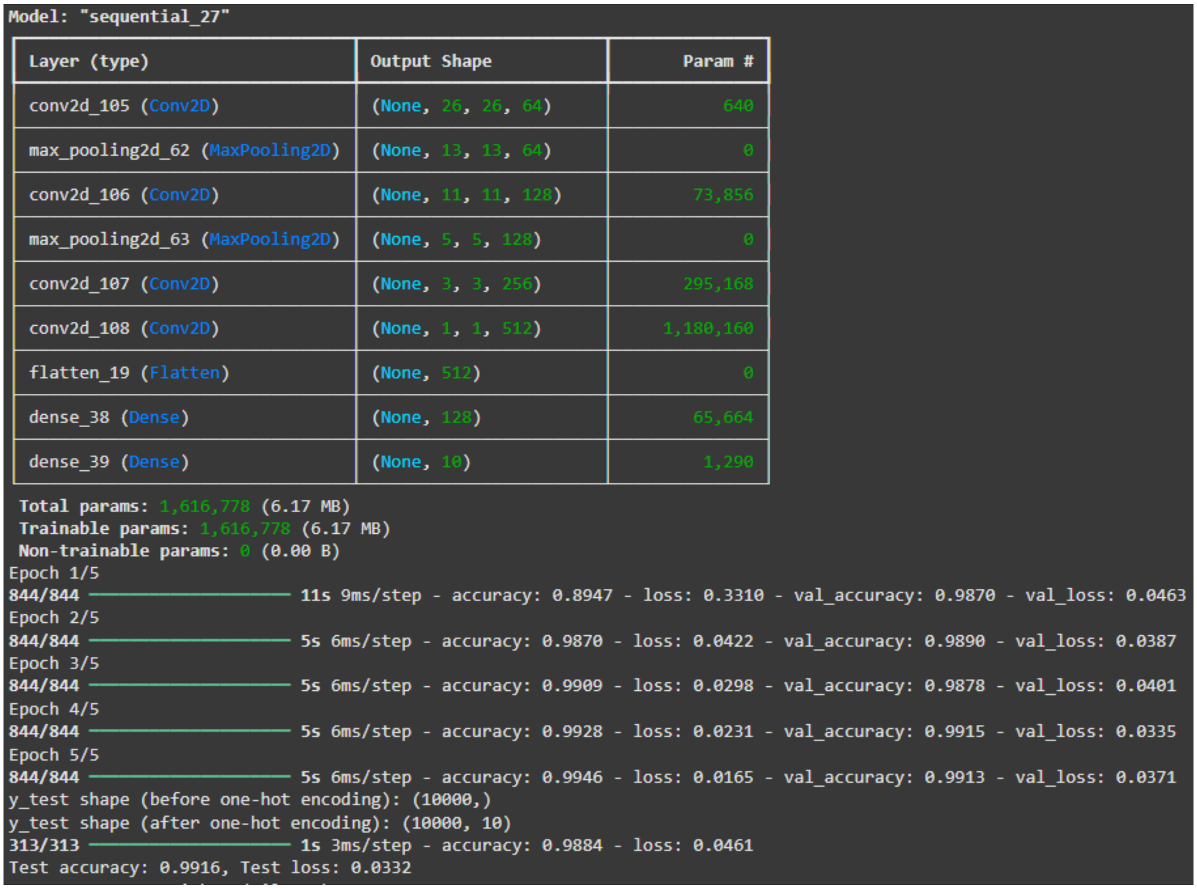
Exercises

Create a **New Project** for every exercise. Take a screenshot of the source code along with its output and place the **source code** and the **screenshot** in a **zipped folder** named **LastNameFirstName_HW10**

Exercise 1

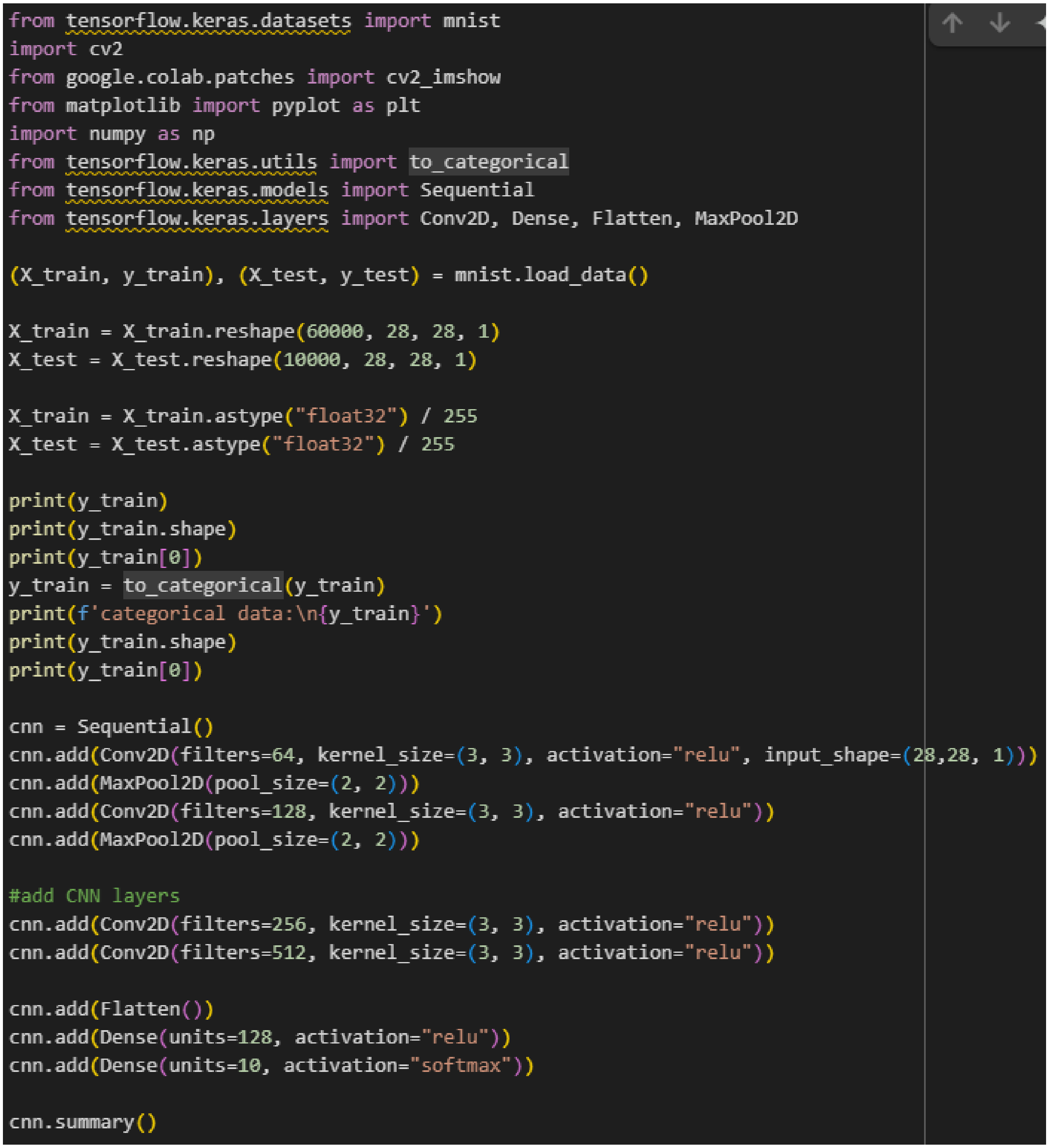
Given the following source code: Lec37.0_CNN_MNIST_withWeights.py, modify the architecture of the CNN by adding CNN layers. Report the new training, validation and test **accuracy**.

New training, validation, and test accuracy:

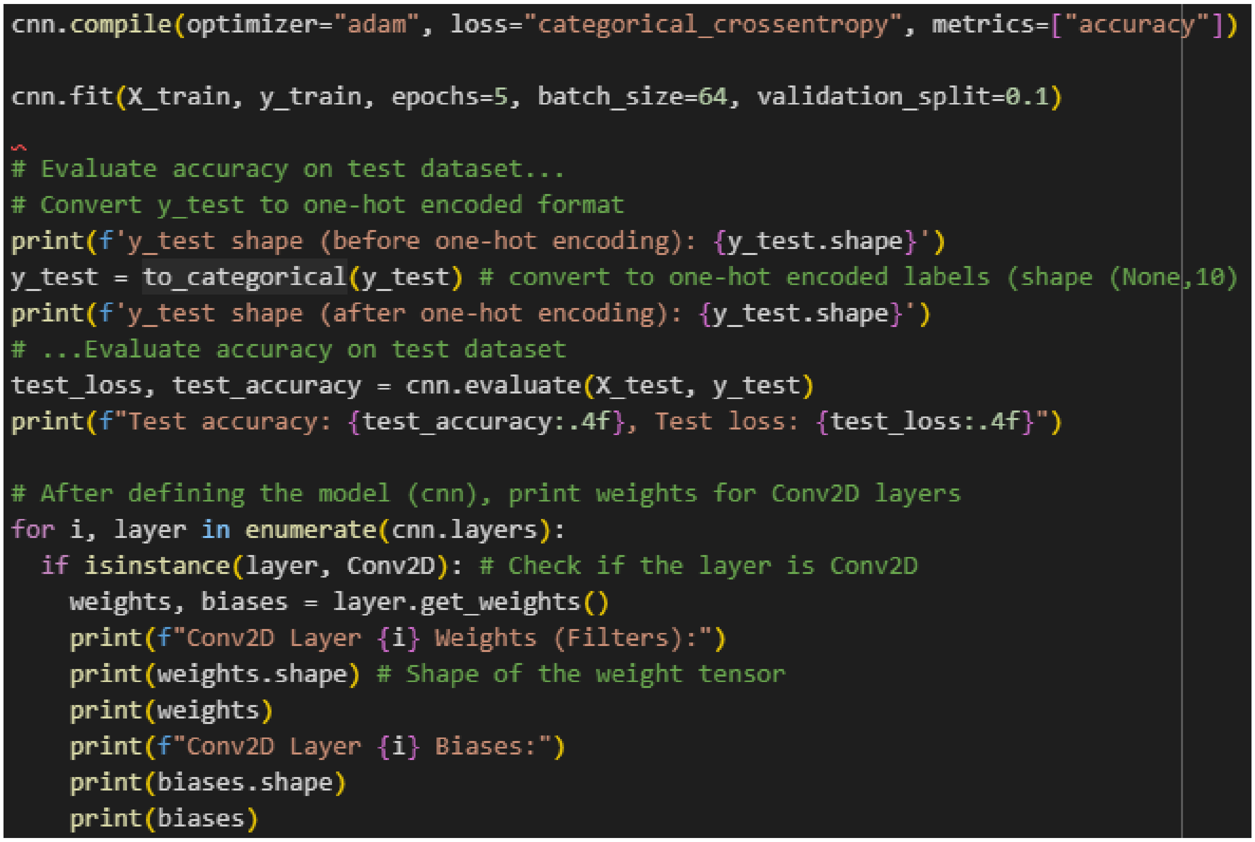


```
Model: "sequential_27"
┌───────────────────────────────┬──────────────────────────┬───────────────┐
│ Layer (type)                  │ Output Shape             │       Param # │
├───────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_105 (Conv2D)           │ (None, 26, 26, 64)       │           640 │
├───────────────────────────────┼──────────────────────────┼───────────────┤
│ max_pooling2d_62 (MaxPooling2D)│ (None, 13, 13, 64)      │             0 │
├───────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_106 (Conv2D)           │ (None, 11, 11, 128)      │        73,856 │
├───────────────────────────────┼──────────────────────────┼───────────────┤
│ max_pooling2d_63 (MaxPooling2D)│ (None, 5, 5, 128)       │             0 │
├───────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_107 (Conv2D)           │ (None, 3, 3, 256)        │       295,168 │
├───────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_108 (Conv2D)           │ (None, 1, 1, 512)        │     1,180,160 │
├───────────────────────────────┼──────────────────────────┼───────────────┤
│ flatten_19 (Flatten)          │ (None, 512)              │             0 │
├───────────────────────────────┼──────────────────────────┼───────────────┤
│ dense_38 (Dense)              │ (None, 128)              │        65,664 │
├───────────────────────────────┼──────────────────────────┼───────────────┤
│ dense_39 (Dense)              │ (None, 10)               │         1,290 │
└───────────────────────────────┴──────────────────────────┴───────────────┘
 Total params: 1,616,778 (6.17 MB)
 Trainable params: 1,616,778 (6.17 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 11s 9ms/step - accuracy: 0.8947 - loss: 0.3310 - val_accuracy: 0.9870 - val_loss: 0.0463
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9870 - loss: 0.0422 - val_accuracy: 0.9890 - val_loss: 0.0387
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9909 - loss: 0.0298 - val_accuracy: 0.9878 - val_loss: 0.0401
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9928 - loss: 0.0231 - val_accuracy: 0.9915 - val_loss: 0.0335
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9946 - loss: 0.0165 - val_accuracy: 0.9913 - val_loss: 0.0371
y_test shape (before one-hot encoding): (10000,)
y_test shape (after one-hot encoding): (10000, 10)
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9884 - loss: 0.0461
Test accuracy: 0.9916, Test loss: 0.0332
```

Source Code:



```
from tensorflow.keras.datasets import mnist
import cv2
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPool2D

(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)

X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

print(y_train)
print(y_train.shape)
print(y_train[0])
y_train = to_categorical(y_train)
print(f'categorical data:\n{y_train}')
print(y_train.shape)
print(y_train[0])

cnn = Sequential()
cnn.add(Conv2D(filters=64, kernel_size=(3, 3), activation="relu", input_shape=(28,28, 1)))
cnn.add(MaxPool2D(pool_size=(2, 2)))
cnn.add(Conv2D(filters=128, kernel_size=(3, 3), activation="relu"))
cnn.add(MaxPool2D(pool_size=(2, 2)))

#add CNN layers
cnn.add(Conv2D(filters=256, kernel_size=(3, 3), activation="relu"))
cnn.add(Conv2D(filters=512, kernel_size=(3, 3), activation="relu"))

cnn.add(Flatten())
cnn.add(Dense(units=128, activation="relu"))
cnn.add(Dense(units=10, activation="softmax"))

cnn.summary()
```

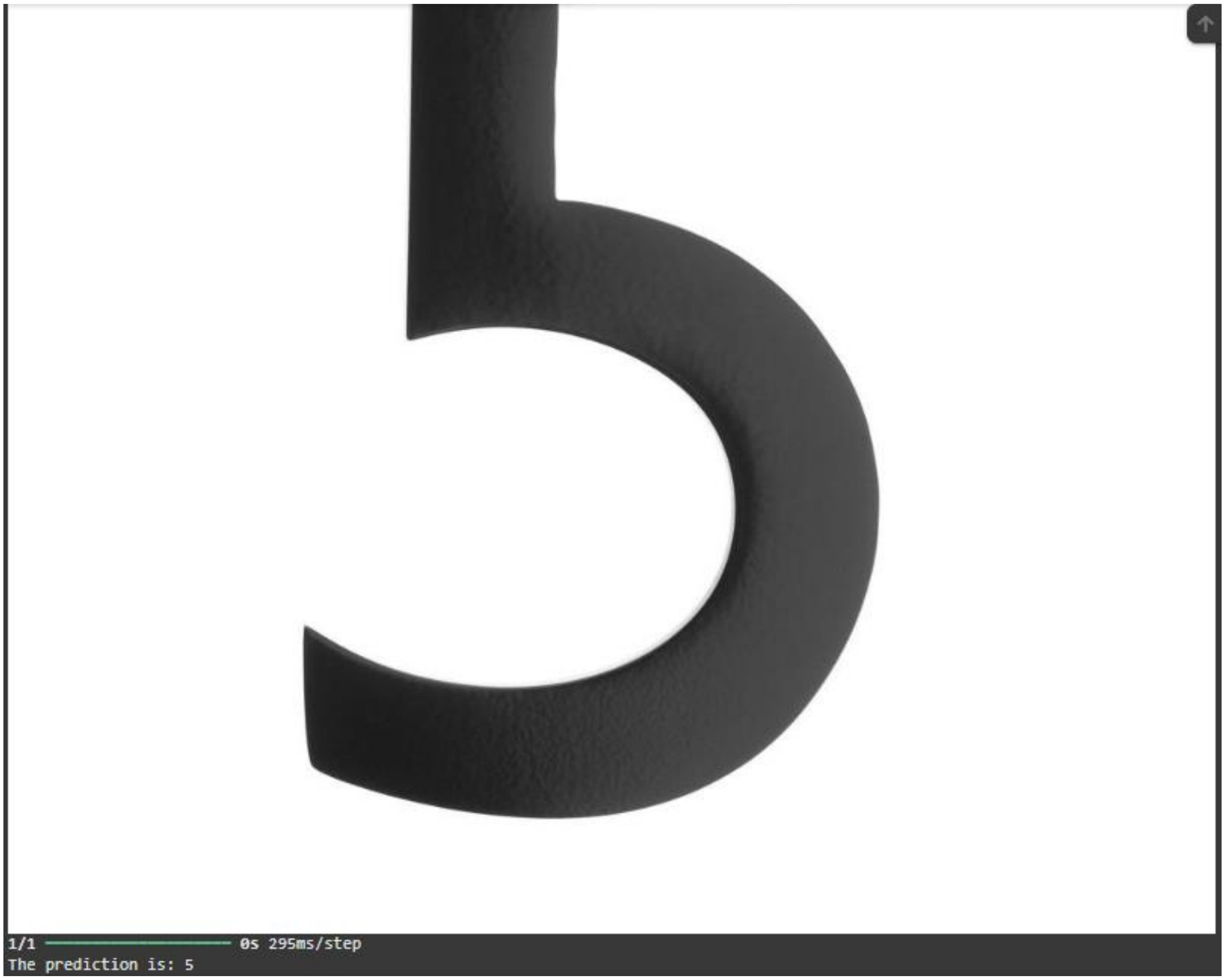


```
cnn.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

cnn.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.1)

# Evaluate accuracy on test dataset...
# Convert y_test to one-hot encoded format
print(f'y_test shape (before one-hot encoding): {y_test.shape}')
y_test = to_categorical(y_test) # convert to one-hot encoded labels (shape (None,10)
print(f'y_test shape (after one-hot encoding): {y_test.shape}')
# ...Evaluate accuracy on test dataset
test_loss, test_accuracy = cnn.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}, Test loss: {test_loss:.4f}")

# After defining the model (cnn), print weights for Conv2D layers
for i, layer in enumerate(cnn.layers):
  if isinstance(layer, Conv2D): # Check if the layer is Conv2D
    weights, biases = layer.get_weights()
    print(f"Conv2D Layer {i} Weights (Filters):")
    print(weights.shape) # Shape of the weight tensor
    print(weights)
    print(f"Conv2D Layer {i} Biases:")
    print(biases.shape)
    print(biases)
```

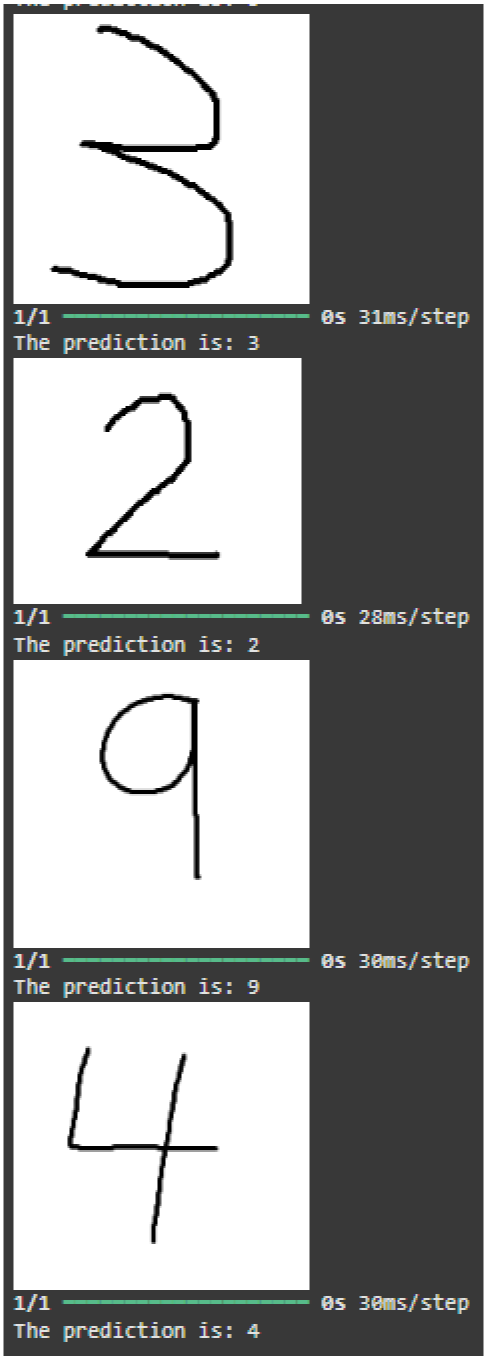
Exercise 2

In continuation of Ex. 1, **predict** the classification outcome of two handwritten digits. Print the images as well as the predictions.

Output:



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 295ms/step
The prediction is: 5
```



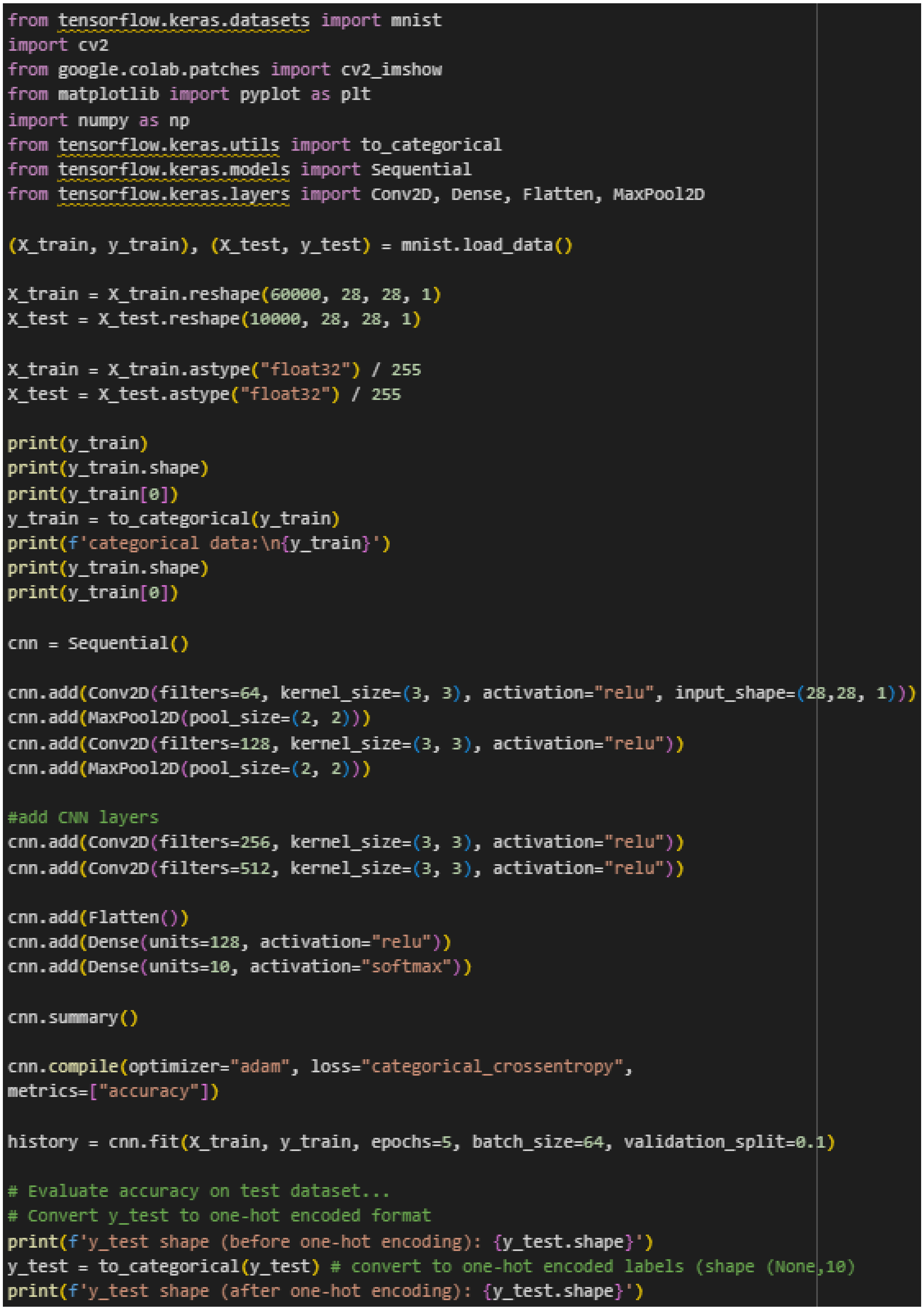
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
The prediction is: 3
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
The prediction is: 2
```

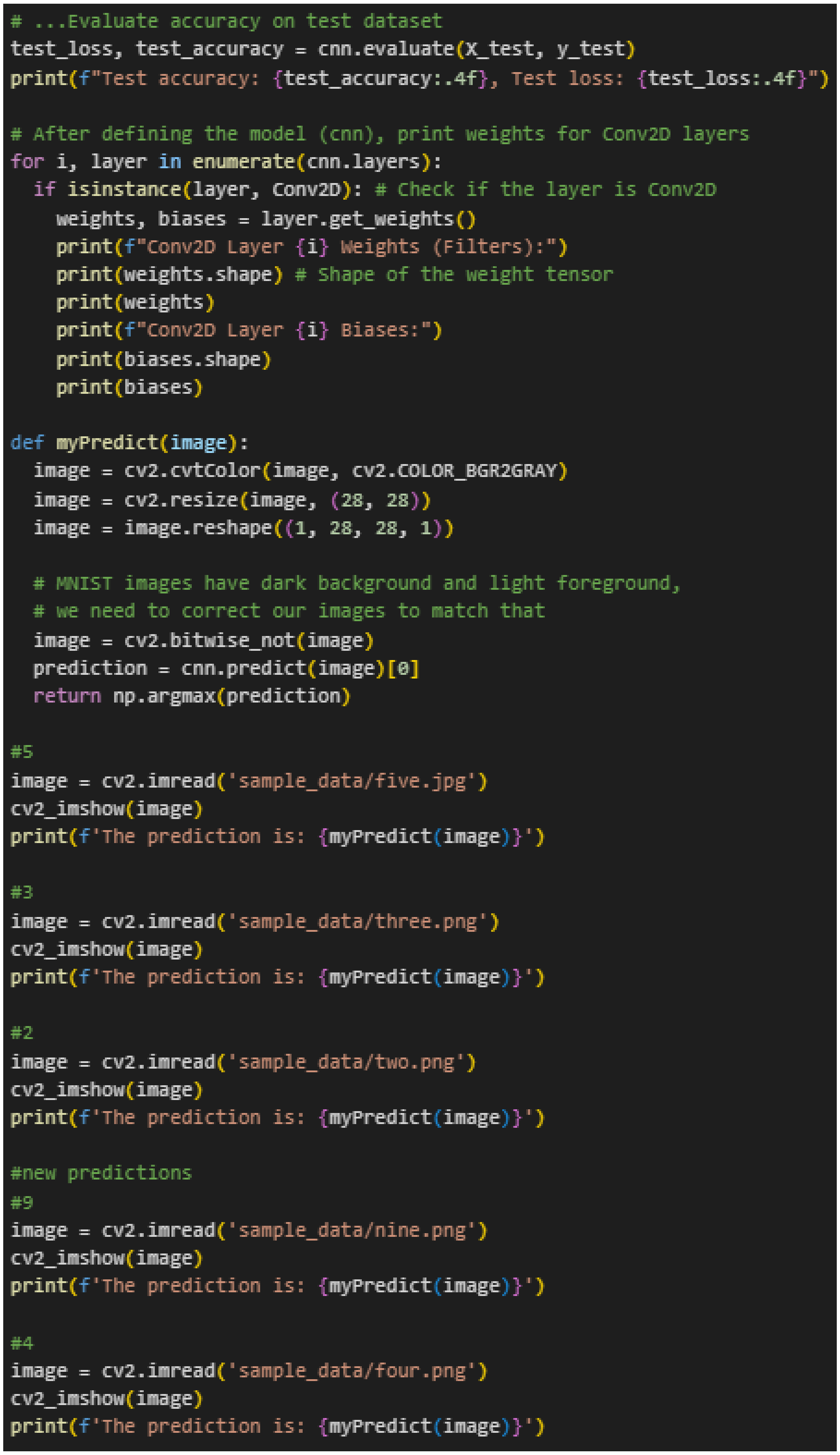
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
The prediction is: 9
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
The prediction is: 4
```

Source Code:



```python
from tensorflow.keras.datasets import mnist
import cv2
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPool2D

(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(60000, 28, 28, 1)
X_test = X_test.reshape(10000, 28, 28, 1)

X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

print(y_train)
print(y_train.shape)
print(y_train[0])
y_train = to_categorical(y_train)
print(f'categorical data:\n{y_train}')
print(y_train.shape)
print(y_train[0])

cnn = Sequential()

cnn.add(Conv2D(filters=64, kernel_size=(3, 3), activation="relu", input_shape=(28,28, 1)))
cnn.add(MaxPool2D(pool_size=(2, 2)))
cnn.add(Conv2D(filters=128, kernel_size=(3, 3), activation="relu"))
cnn.add(MaxPool2D(pool_size=(2, 2)))

#add CNN layers
cnn.add(Conv2D(filters=256, kernel_size=(3, 3), activation="relu"))
cnn.add(Conv2D(filters=512, kernel_size=(3, 3), activation="relu"))

cnn.add(Flatten())
cnn.add(Dense(units=128, activation="relu"))
cnn.add(Dense(units=10, activation="softmax"))

cnn.summary()

cnn.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=["accuracy"])

history = cnn.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.1)

# Evaluate accuracy on test dataset...
# Convert y_test to one-hot encoded format
print(f'y_test shape (before one-hot encoding): {y_test.shape}')
y_test = to_categorical(y_test) # convert to one-hot encoded labels (shape (None,10)
print(f'y_test shape (after one-hot encoding): {y_test.shape}')
```



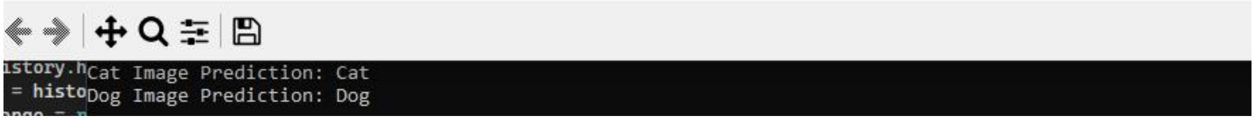
```
# ...Evaluate accuracy on test dataset
test_loss, test_accuracy = cnn.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}, Test loss: {test_loss:.4f}")

# After defining the model (cnn), print weights for Conv2D layers
for i, layer in enumerate(cnn.layers):
  if isinstance(layer, Conv2D): # Check if the layer is Conv2D
    weights, biases = layer.get_weights()
    print(f"Conv2D Layer {i} Weights (Filters):")
    print(weights.shape) # Shape of the weight tensor
    print(weights)
    print(f"Conv2D Layer {i} Biases:")
    print(biases.shape)
    print(biases)

def myPredict(image):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  image = cv2.resize(image, (28, 28))
  image = image.reshape((1, 28, 28, 1))

  # MNIST images have dark background and light foreground,
  # we need to correct our images to match that
  image = cv2.bitwise_not(image)
  prediction = cnn.predict(image)[0]
  return np.argmax(prediction)

#5
image = cv2.imread('sample_data/five.jpg')
cv2_imshow(image)
print(f'The prediction is: {myPredict(image)}')

#3
image = cv2.imread('sample_data/three.png')
cv2_imshow(image)
print(f'The prediction is: {myPredict(image)}')

#2
image = cv2.imread('sample_data/two.png')
cv2_imshow(image)
print(f'The prediction is: {myPredict(image)}')

#new predictions
#9
image = cv2.imread('sample_data/nine.png')
cv2_imshow(image)
print(f'The prediction is: {myPredict(image)}')

#4
image = cv2.imread('sample_data/four.png')
cv2_imshow(image)
print(f'The prediction is: {myPredict(image)}')
```

Exercise 3

Given the following datasets: train.zip and validation.zip, and the accompanying source code: Lec37.1_CNN_cats+dogs.py. This a CNN that classifies images between cats and dogs. Modify the number of **EPOCHS** to a larger value. Report whether the accuracy improves. In addition, **predict** the classification outcome of two images, one **cat** and one **dog**. Print the images as well as the predictions. The two images should not be from the datasets.

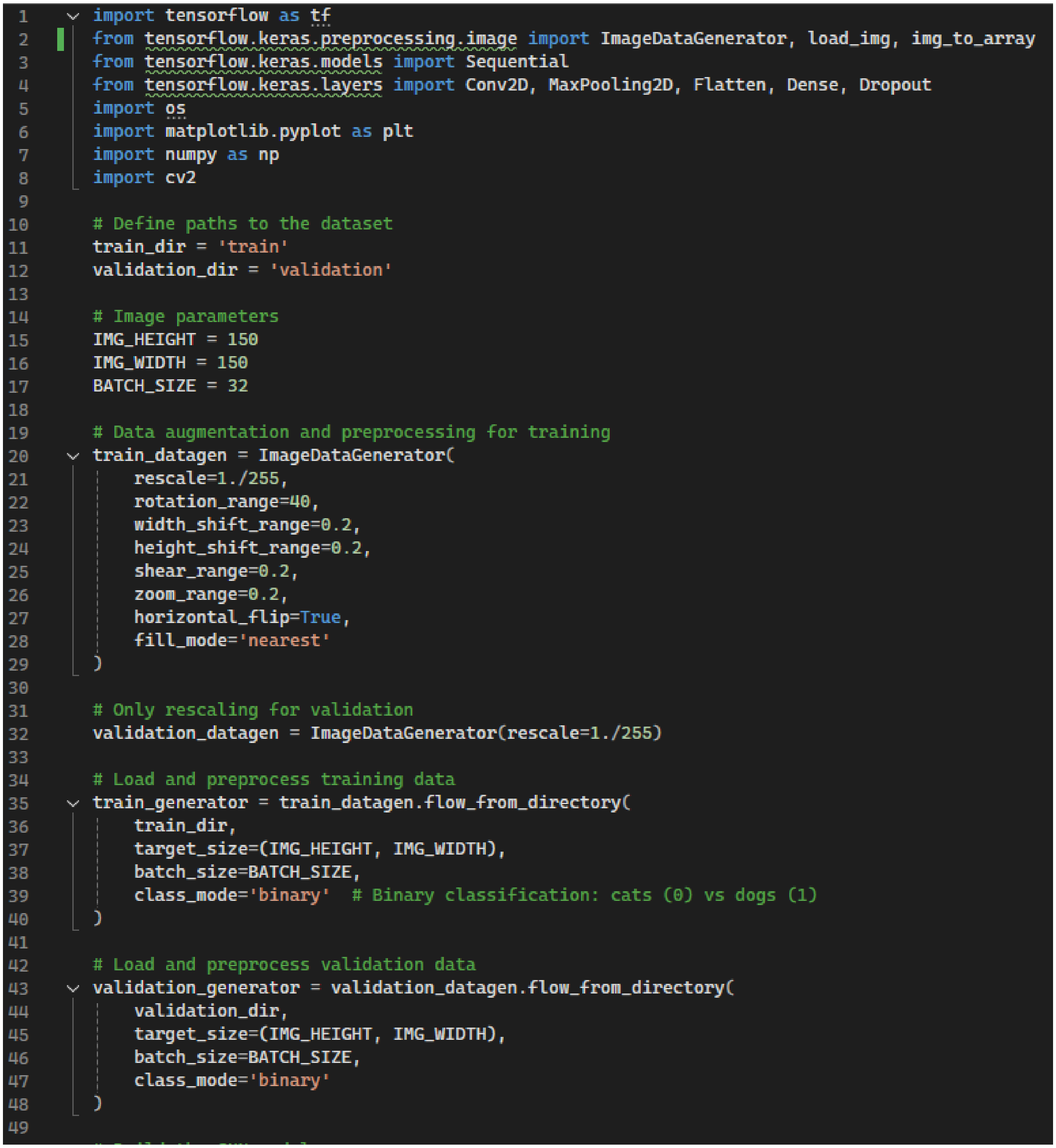
Note: The code should run without any issues on Visual Studio.

Output:

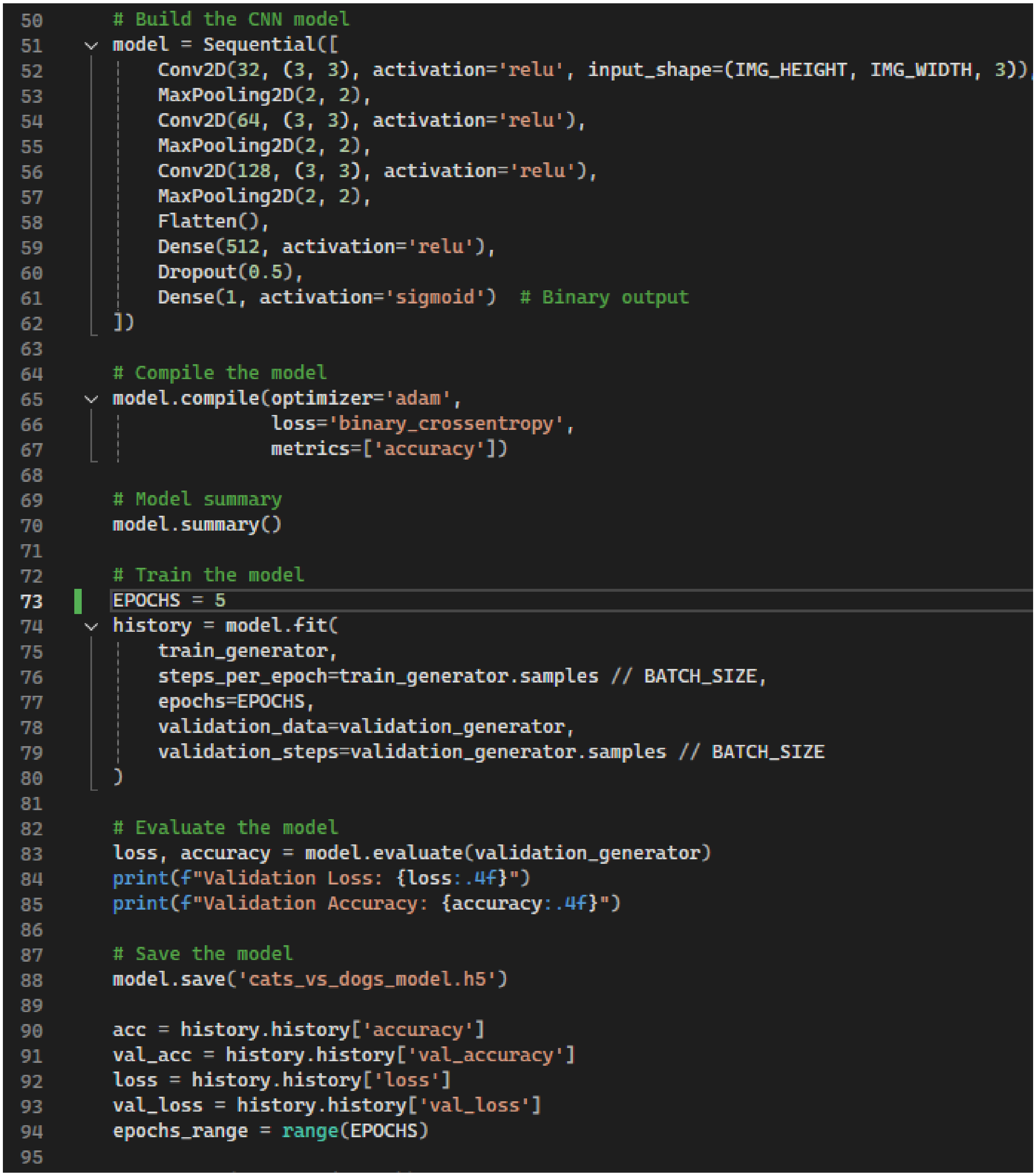


```
istory.hCat Image Prediction: Cat
 = histoDog Image Prediction: Dog
```

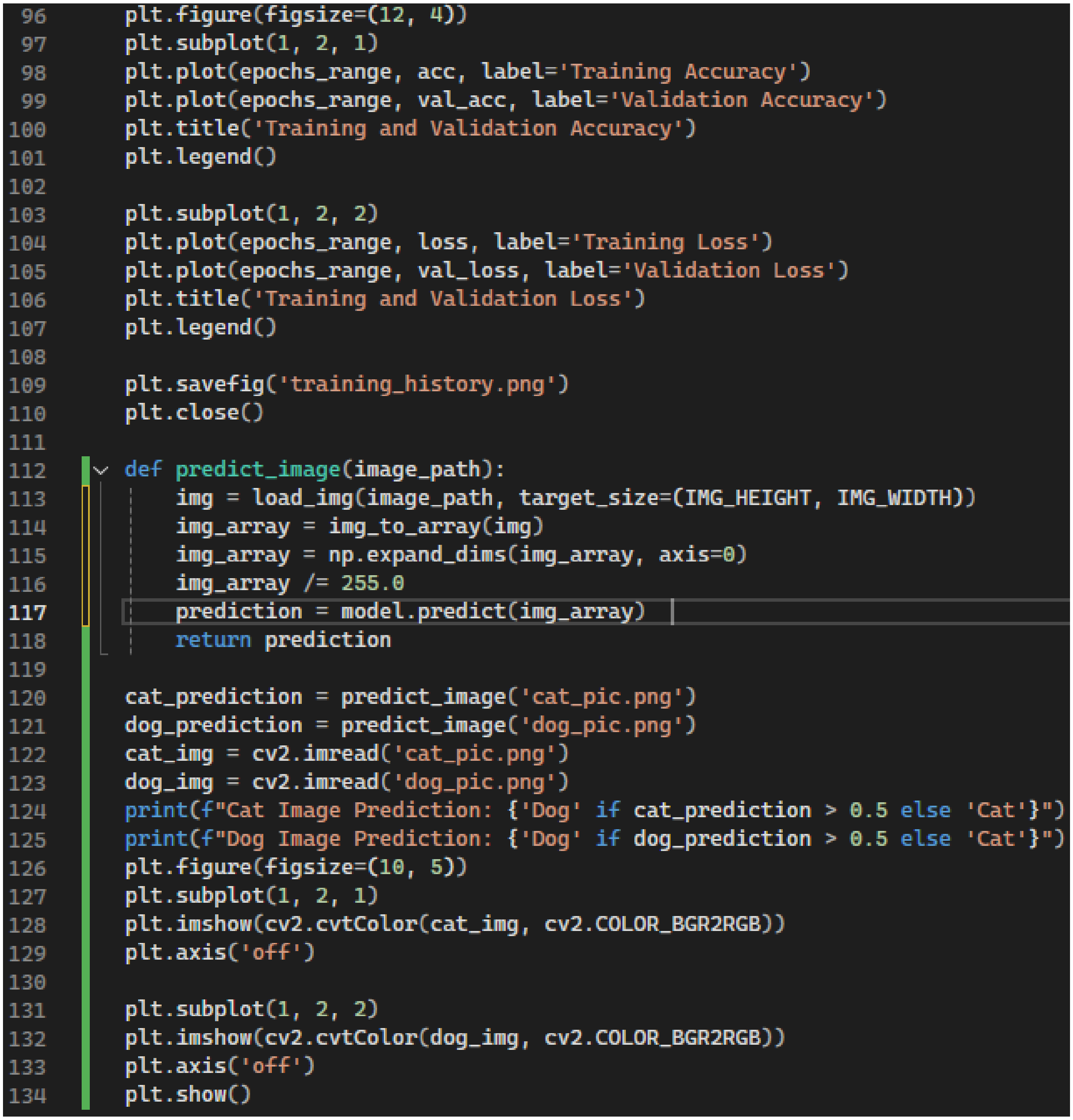
Source Code:



```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import os
import matplotlib.pyplot as plt
import numpy as np
import cv2

# Define paths to the dataset
train_dir = 'train'
validation_dir = 'validation'

# Image parameters
IMG_HEIGHT = 150
IMG_WIDTH = 150
BATCH_SIZE = 32

# Data augmentation and preprocessing for training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Only rescaling for validation
validation_datagen = ImageDataGenerator(rescale=1./255)

# Load and preprocess training data
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary'  # Binary classification: cats (0) vs dogs (1)
)

# Load and preprocess validation data
validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary'
)

# Build the CNN model
```



```python
# Build the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # Binary output
])

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

# Model summary
model.summary()

# Train the model
EPOCHS = 5
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE
)

# Evaluate the model
loss, accuracy = model.evaluate(validation_generator)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy:.4f}")

# Save the model
model.save('cats_vs_dogs_model.h5')

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(EPOCHS)

```



```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.savefig('training_history.png')
plt.close()

def predict_image(image_path):
    img = load_img(image_path, target_size=(IMG_HEIGHT, IMG_WIDTH))
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0
    prediction = model.predict(img_array)
    return prediction

cat_prediction = predict_image('cat_pic.png')
dog_prediction = predict_image('dog_pic.png')
cat_img = cv2.imread('cat_pic.png')
dog_img = cv2.imread('dog_pic.png')
print(f"Cat Image Prediction: {'Dog' if cat_prediction > 0.5 else 'Cat'}")
print(f"Dog Image Prediction: {'Dog' if dog_prediction > 0.5 else 'Cat'}")
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(cat_img, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(dog_img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

Note: Submit through Canvas