Image Classification

Image recognition using deep learning

Nadav Moyal - 208514265
Tzach Aker - 315339069

Please note: when it says picture 1-8, it means that the pictures shown at the end of the .summary describe what is written

Abstract:
The project of **image classification** uses CIFAR-10 dataset and TensorFlow 2. This program uses a pre-trained image-classification model to make predictions on a set of images.

The purpose of this project is to utilize TensorFlow 2 and the CIFAR-10 dataset in Python to develop and train a deep learning model for image classification. The model will be trained on a set of 60,000 32x32 color images, each belonging to one of 10 classes, and will be evaluated on a separate set of 10,000 images. The goal is to achieve a high level of accuracy in classifying the images, demonstrating the effectiveness of TensorFlow 2 and deep learning techniques in image recognition tasks.

It consists of two scripts:

1. modelTensor2.py - it trains and saves a CNN model.
2. main.py - it uses the pre-trained model to classify new images by reading them from a folder, making predictions, and displaying the results.

It saves a CNN model using the CIFAR-10 dataset and uses the pre-trained model to classify new images by reading them from a folder, making predictions, and displaying the results. This program compiles and trains the model, evaluates the model, prints the loss and accuracy of the model, and saves the model to a file.
it loads the pre-trained model, reads image files from a "images" folder, makes predictions on them using the pre-trained model, and displays the predicted class and the image using matplotlib. It uses Adam optimizer, sparse categorical cross-entropy loss function, and accuracy as a metric for the model.

The project's experiment:
The modelTensor2 script is responsible for defining, training, and evaluating the model. It

starts by loading the CIFAR-10 dataset.
Next, it defines a convolutional neural network (CNN) model using the Sequential API in TensorFlow's Keras library.
The model consists of several layers, including convolutional layers, max pooling layers, dropout layers, and dense layers.
Finally, it trains the model using the training images and labels for 15 epochs, evaluate the model using the testing images and labels, print the loss and accuracy of the model, and saves the model to a file named 'image_classifier_tensor_2.model'.

The main script is responsible for using the pre-trained model to classify new images. It reads image files from a "images" folder, makes predictions on them using the pre-trained model, and displays the predicted class and the image using matplotlib.

Links to other experiments on CIFAR-10 dataset or using the CNN model:
1. https://www.kaggle.com/code/csmohamedayman/cifar-10-object-recognition-in-images
2. https://www.kaggle.com/code/ayushnitb/cifar10-robust4layer-convnet-pytorch-85-acc

Description:

To run this program, you will need Python 3.x, OpenCV (cv2), Numpy (numpy), Matplotlib (matplotlib), and TensorFlow (tensorflow 2).
Train the model by running modelTraining.py. This will create the image_classifier_tensor_2.model file.
Use the trained model to make predictions by running main.py.
Note: The script imports the TensorFlow 2.0 API.

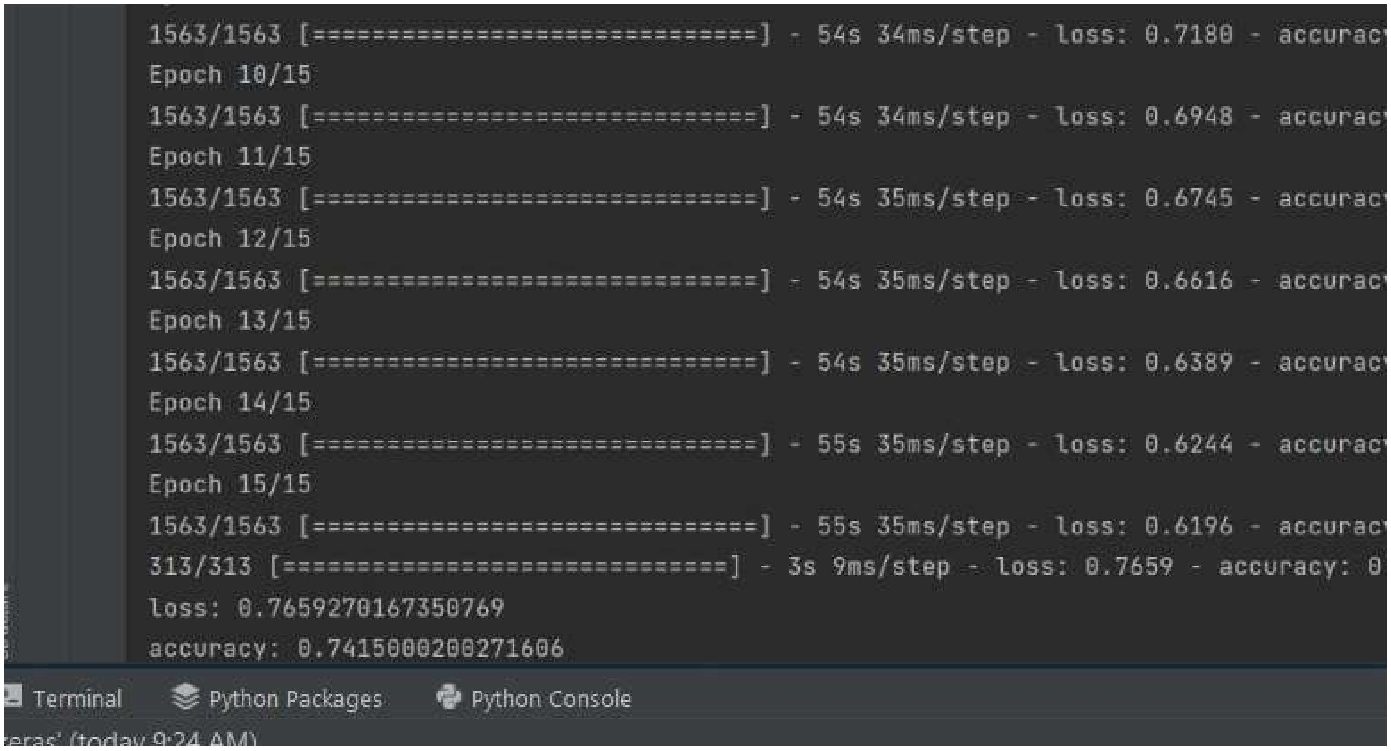
modelTensor2.py:
This script defines functions that load and prepare the CIFAR-10 dataset, define a model using the TensorFlow 2 Keras API, and train and evaluate the model. The trained model is saved to a file.
.script is run when executed (**pictures 1-2**).

main.py:
This script defines functions that load and preprocess data, load a pre-trained model, make predictions on images, and display the results. The script is run when executed. (**pictures 3-4**).

outputs (pictures 5-8).
more complex tests (pictures 9-13).

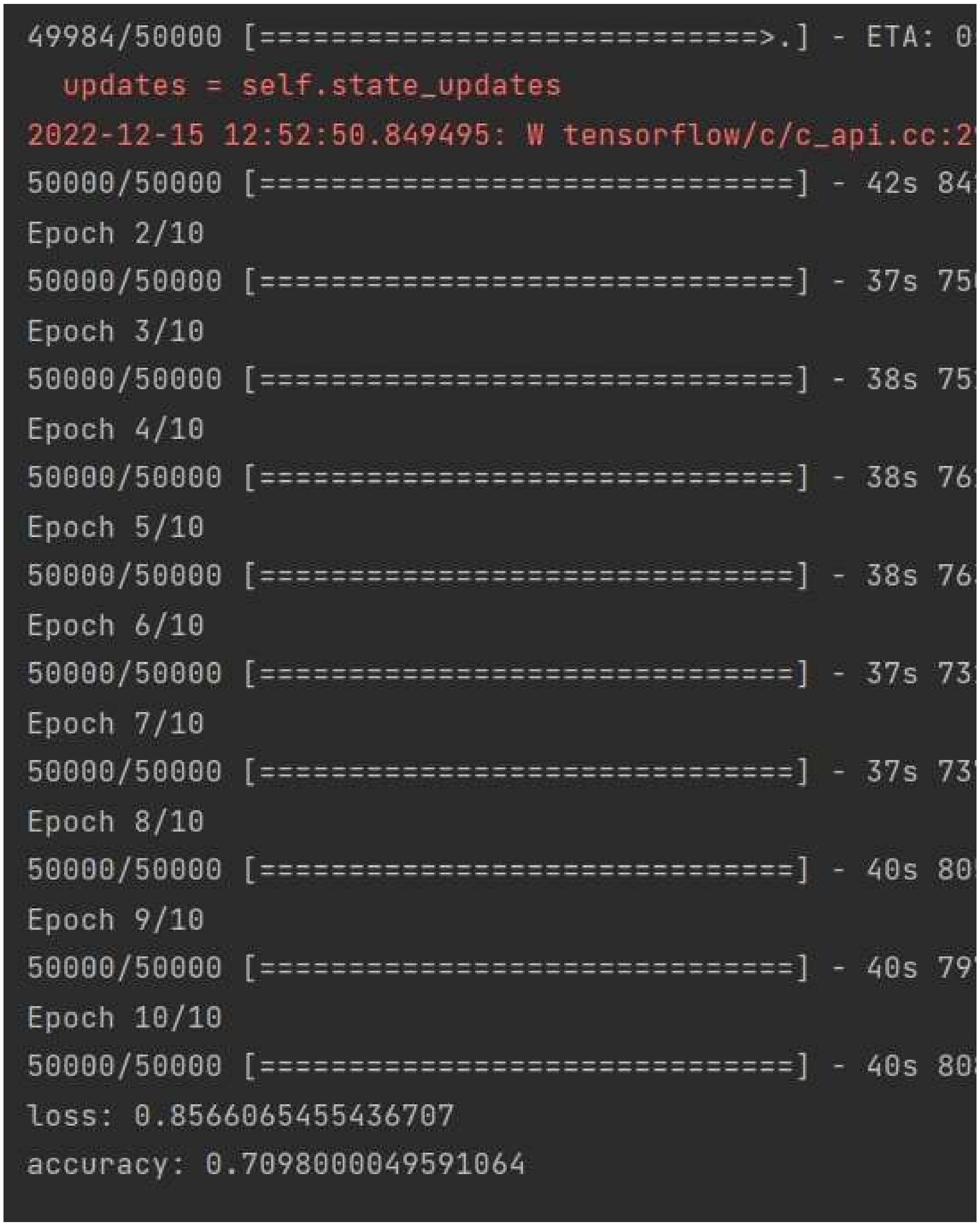
Comparison between the previous model (assignment 2) and the current model (assignment 3):

<u>results of the training with softmax - Tensorflow 2:</u>
<u>loss:</u> 0.765
<u>accuracy:</u> 0.741



```
1563/1563 [==============================] - 54s 34ms/step - loss: 0.7180 - accurac
Epoch 10/15
1563/1563 [==============================] - 54s 34ms/step - loss: 0.6948 - accurac
Epoch 11/15
1563/1563 [==============================] - 54s 35ms/step - loss: 0.6745 - accurac
Epoch 12/15
1563/1563 [==============================] - 54s 35ms/step - loss: 0.6616 - accurac
Epoch 13/15
1563/1563 [==============================] - 54s 35ms/step - loss: 0.6389 - accurac
Epoch 14/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.6244 - accurac
Epoch 15/15
1563/1563 [==============================] - 55s 35ms/step - loss: 0.6196 - accurac
313/313 [==============================] - 3s 9ms/step - loss: 0.7659 - accuracy: 0
loss: 0.7659270167350769
accuracy: 0.7415000200271606
```

<u>results of the training with softmax - Tensorflow 1:</u>
<u>loss:</u> 0.856
<u>accuracy:</u> 0.709

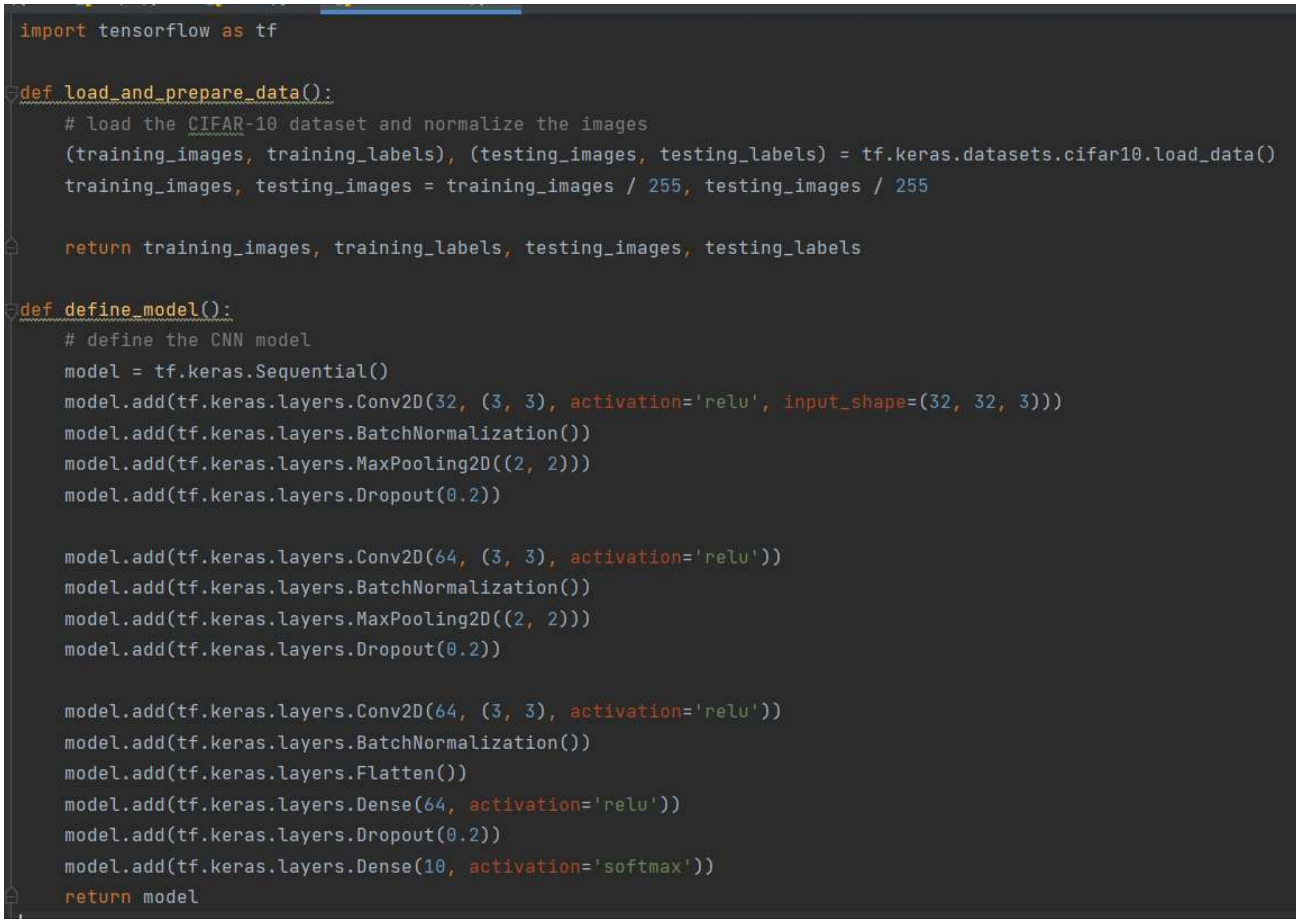


```
49984/50000 [============================>.] - ETA: 0
  updates = self.state_updates
2022-12-15 12:52:50.849495: W tensorflow/c/c_api.cc:2
50000/50000 [==============================] - 42s 84
Epoch 2/10
50000/50000 [==============================] - 37s 75
Epoch 3/10
50000/50000 [==============================] - 38s 75
Epoch 4/10
50000/50000 [==============================] - 38s 76
Epoch 5/10
50000/50000 [==============================] - 38s 76
Epoch 6/10
50000/50000 [==============================] - 37s 73
Epoch 7/10
50000/50000 [==============================] - 37s 73
Epoch 8/10
50000/50000 [==============================] - 40s 80
Epoch 9/10
50000/50000 [==============================] - 40s 79
Epoch 10/10
50000/50000 [==============================] - 40s 80
loss: 0.8566065455436707
accuracy: 0.7098000049591064
```

In conclusion, it seems that with TensorFlow 2 the model is much more accurate and efficient.

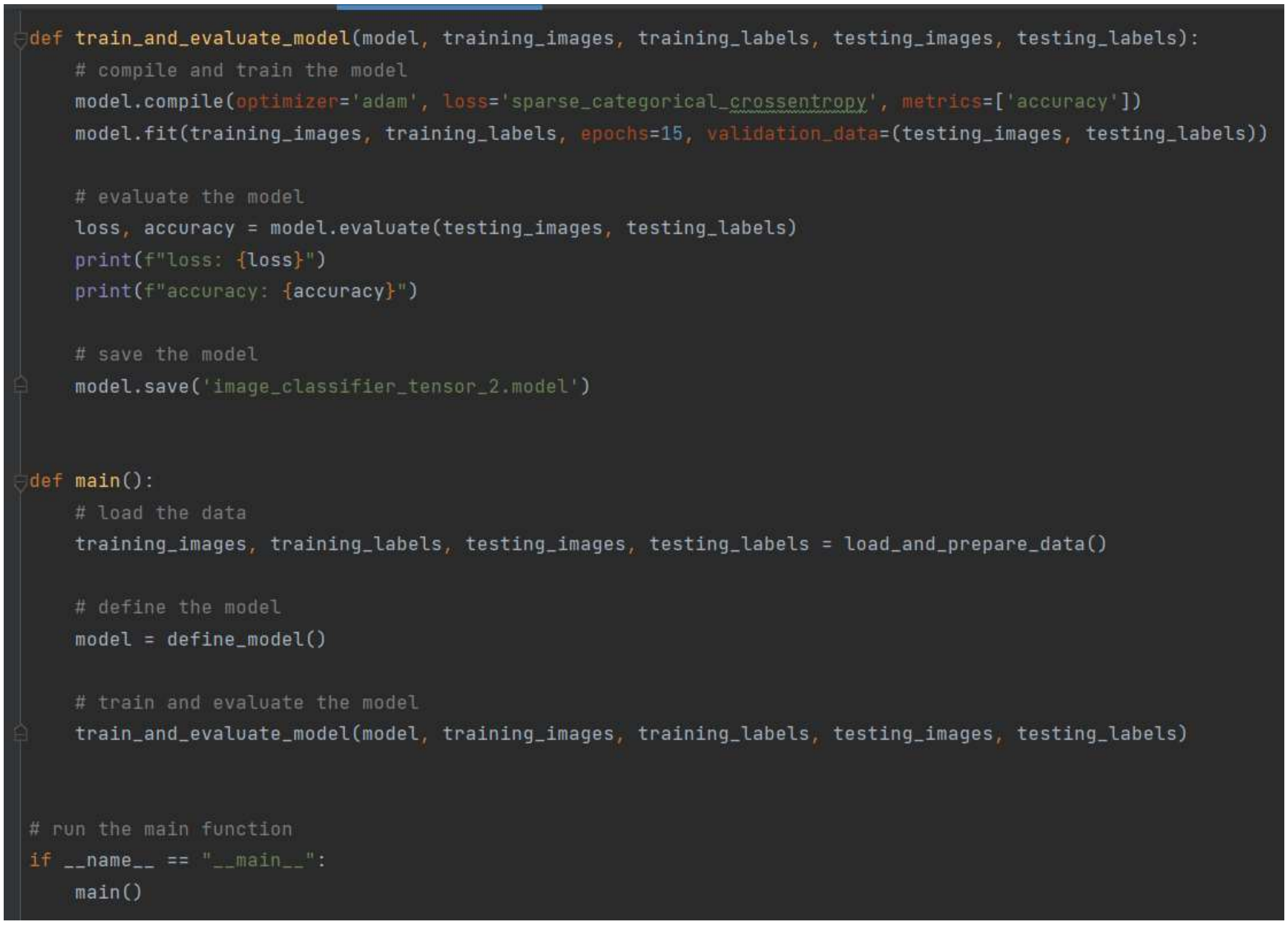
pictures:

picture 1: (modelTensor2.py)



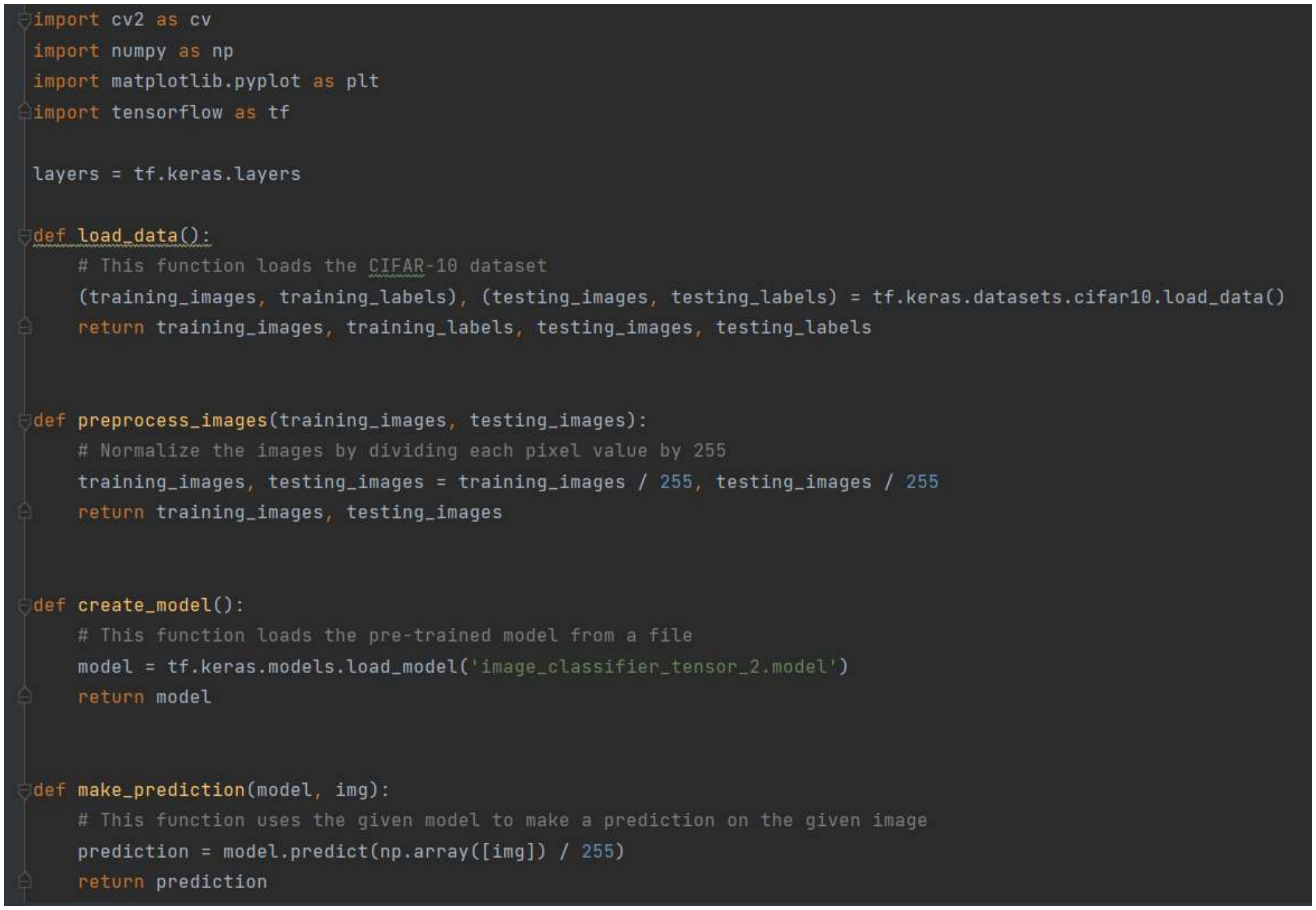
```
import tensorflow as tf

def load_and_prepare_data():
    # load the CIFAR-10 dataset and normalize the images
    (training_images, training_labels), (testing_images, testing_labels) = tf.keras.datasets.cifar10.load_data()
    training_images, testing_images = training_images / 255, testing_images / 255

    return training_images, training_labels, testing_images, testing_labels

def define_model():
    # define the CNN model
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.MaxPooling2D((2, 2)))
    model.add(tf.keras.layers.Dropout(0.2))

    model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.MaxPooling2D((2, 2)))
    model.add(tf.keras.layers.Dropout(0.2))

    model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.2))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))
    return model
```

picture 2:



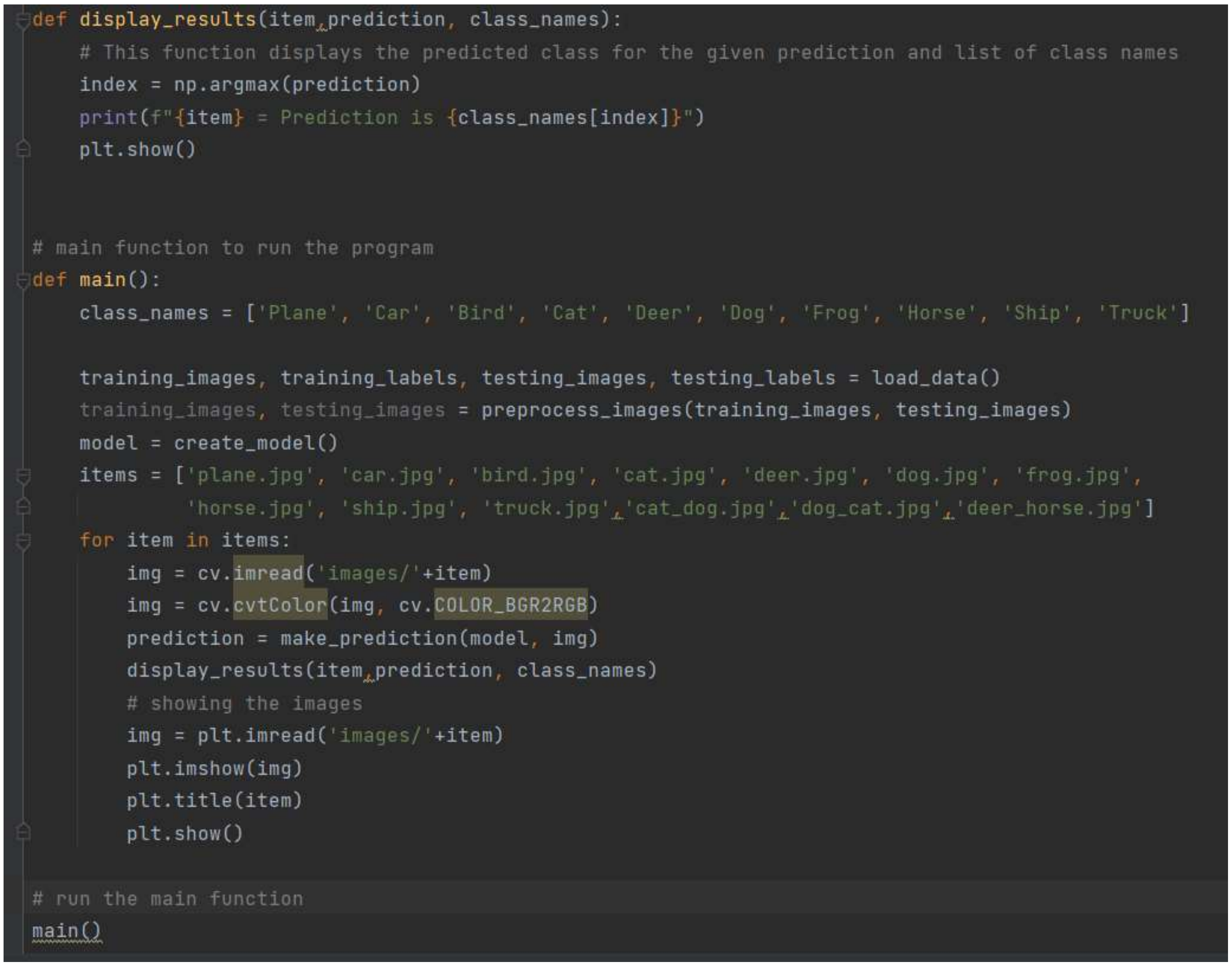
```python
def train_and_evaluate_model(model, training_images, training_labels, testing_images, testing_labels):
    # compile and train the model
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    model.fit(training_images, training_labels, epochs=15, validation_data=(testing_images, testing_labels))

    # evaluate the model
    loss, accuracy = model.evaluate(testing_images, testing_labels)
    print(f"loss: {loss}")
    print(f"accuracy: {accuracy}")

    # save the model
    model.save('image_classifier_tensor_2.model')


def main():
    # load the data
    training_images, training_labels, testing_images, testing_labels = load_and_prepare_data()

    # define the model
    model = define_model()

    # train and evaluate the model
    train_and_evaluate_model(model, training_images, training_labels, testing_images, testing_labels)


# run the main function
if __name__ == "__main__":
    main()
```

picture 3: (main.py)



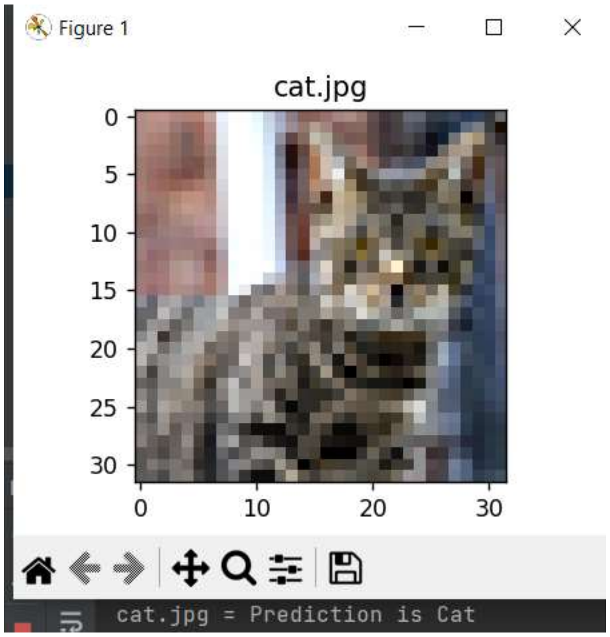
```python
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

layers = tf.keras.layers

def load_data():
    # This function loads the CIFAR-10 dataset
    (training_images, training_labels), (testing_images, testing_labels) = tf.keras.datasets.cifar10.load_data()
    return training_images, training_labels, testing_images, testing_labels


def preprocess_images(training_images, testing_images):
    # Normalize the images by dividing each pixel value by 255
    training_images, testing_images = training_images / 255, testing_images / 255
    return training_images, testing_images


def create_model():
    # This function loads the pre-trained model from a file
    model = tf.keras.models.load_model('image_classifier_tensor_2.model')
    return model


def make_prediction(model, img):
    # This function uses the given model to make a prediction on the given image
    prediction = model.predict(np.array([img]) / 255)
    return prediction
```

picture 4:



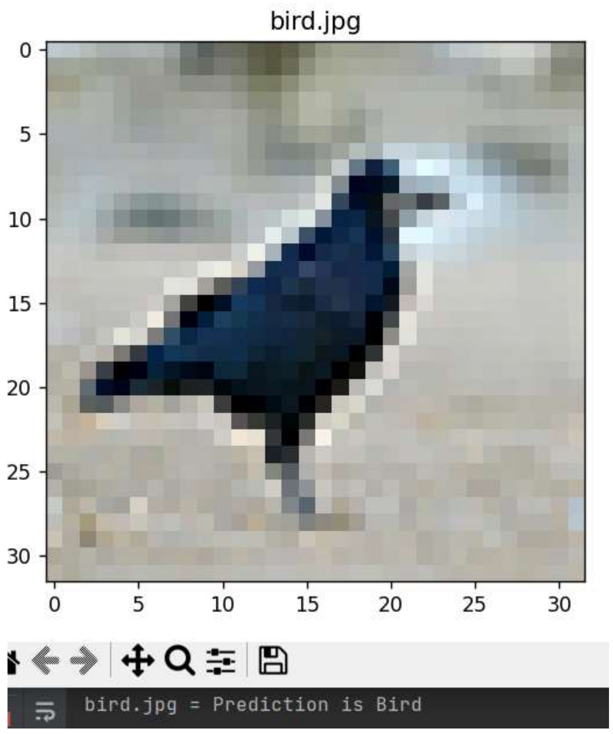
```python
def display_results(item,prediction, class_names):
    # This function displays the predicted class for the given prediction and list of class names
    index = np.argmax(prediction)
    print(f"{item} = Prediction is {class_names[index]}")
    plt.show()


# main function to run the program
def main():
    class_names = ['Plane', 'Car', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

    training_images, training_labels, testing_images, testing_labels = load_data()
    training_images, testing_images = preprocess_images(training_images, testing_images)
    model = create_model()
    items = ['plane.jpg', 'car.jpg', 'bird.jpg', 'cat.jpg', 'deer.jpg', 'dog.jpg', 'frog.jpg',
             'horse.jpg', 'ship.jpg', 'truck.jpg','cat_dog.jpg','dog_cat.jpg','deer_horse.jpg']
    for item in items:
        img = cv.imread('images/'+item)
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        prediction = make_prediction(model, img)
        display_results(item,prediction, class_names)
        # showing the images
        img = plt.imread('images/'+item)
        plt.imshow(img)
        plt.title(item)
        plt.show()

# run the main function
main()
```

picture 5: Prediction is Cat



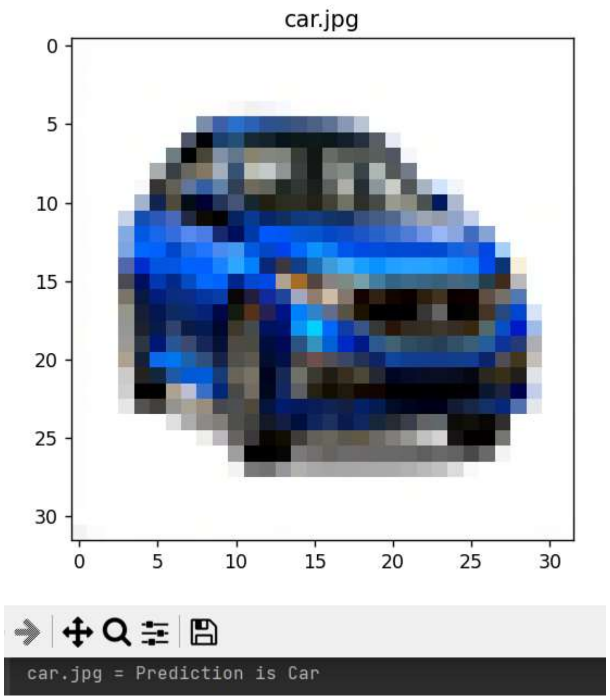
```
cat.jpg = Prediction is Cat
```

picture 6: Prediction is bird



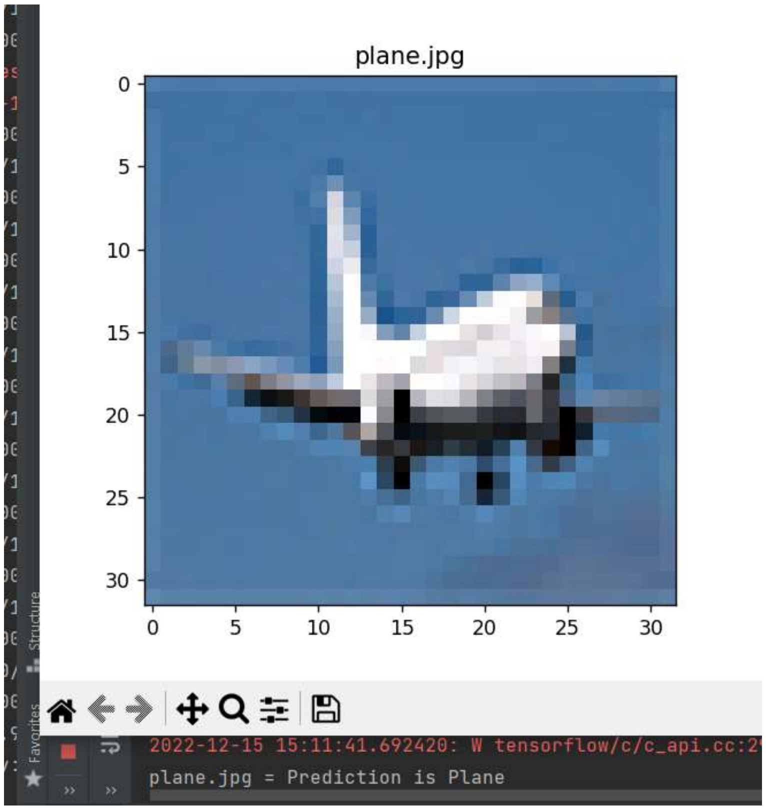
```
bird.jpg = Prediction is Bird
```

picture 7: Prediction is car



```
car.jpg = Prediction is Car
```

picture 8: Prediction is plane

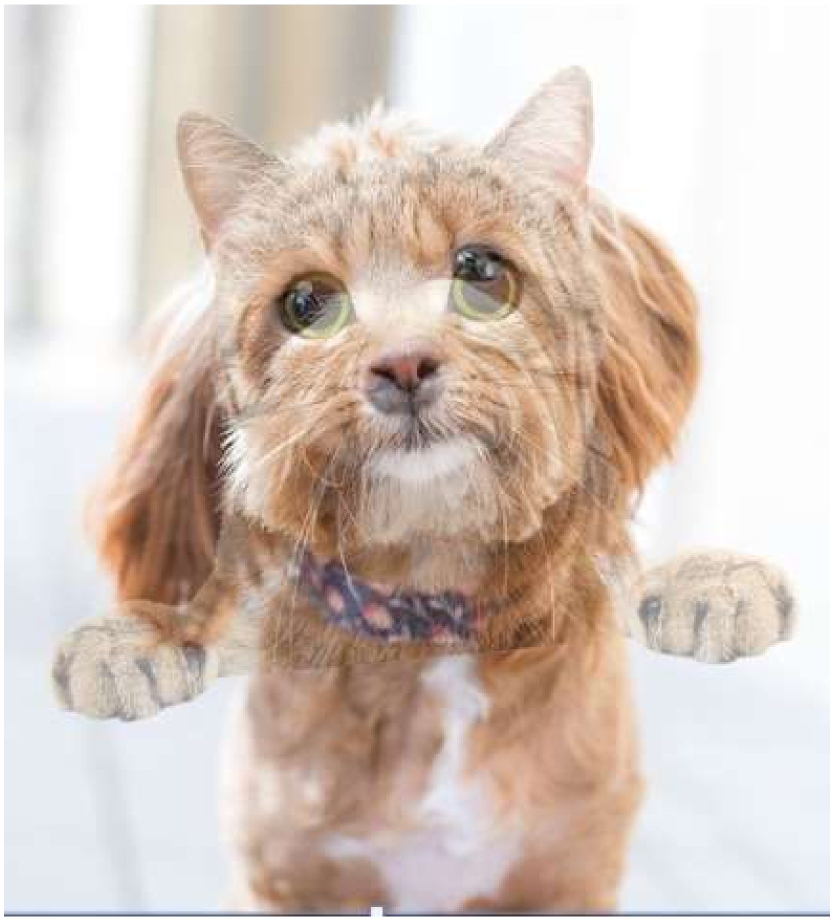


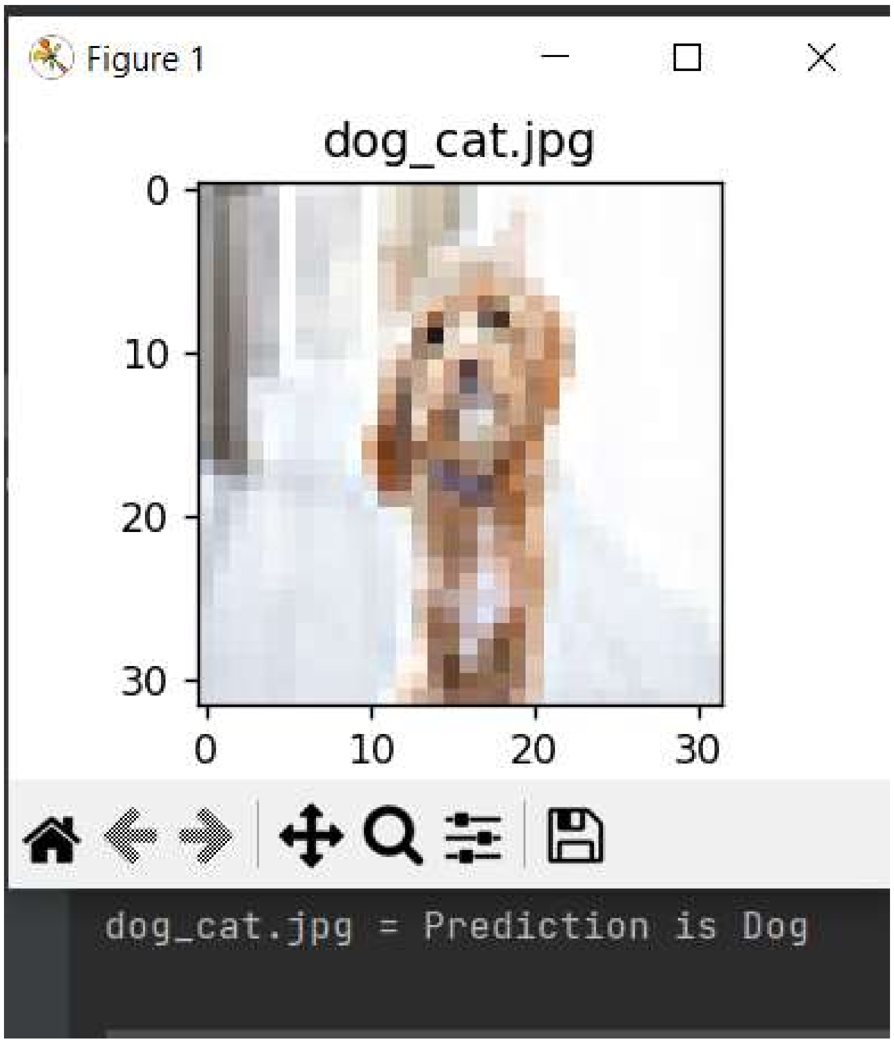
```
2022-12-15 15:11:41.692420: W tensorflow/c/c_api.cc:2
plane.jpg = Prediction is Plane
```

picture 9-13 : more complex tests:

We merged 2 images, and let one of them be more dominant, and checked if the model recognizes who is dominant.

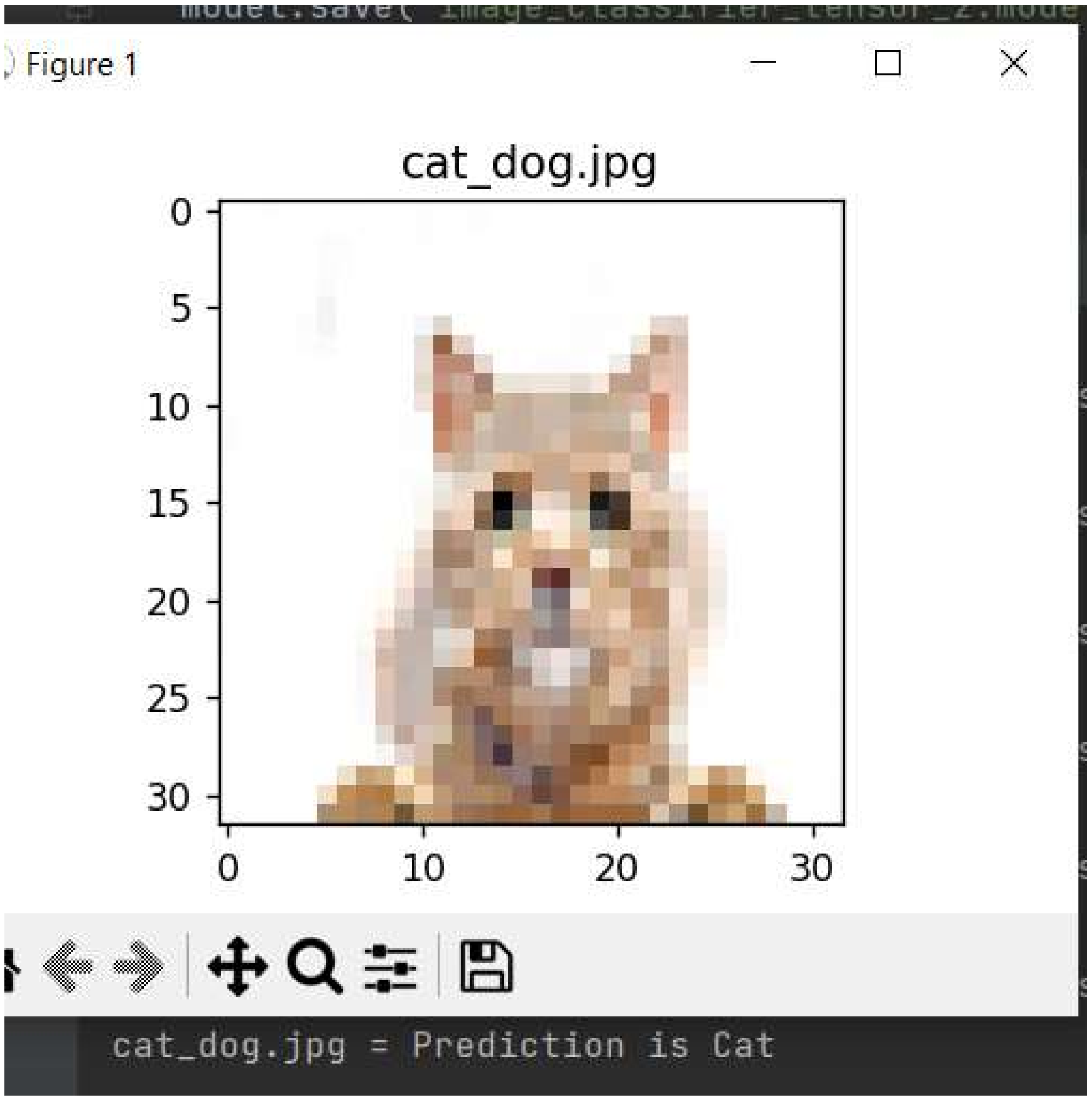
-case 1:
40 % cat + 60 % dog:





```
dog_cat.jpg = Prediction is Dog
```

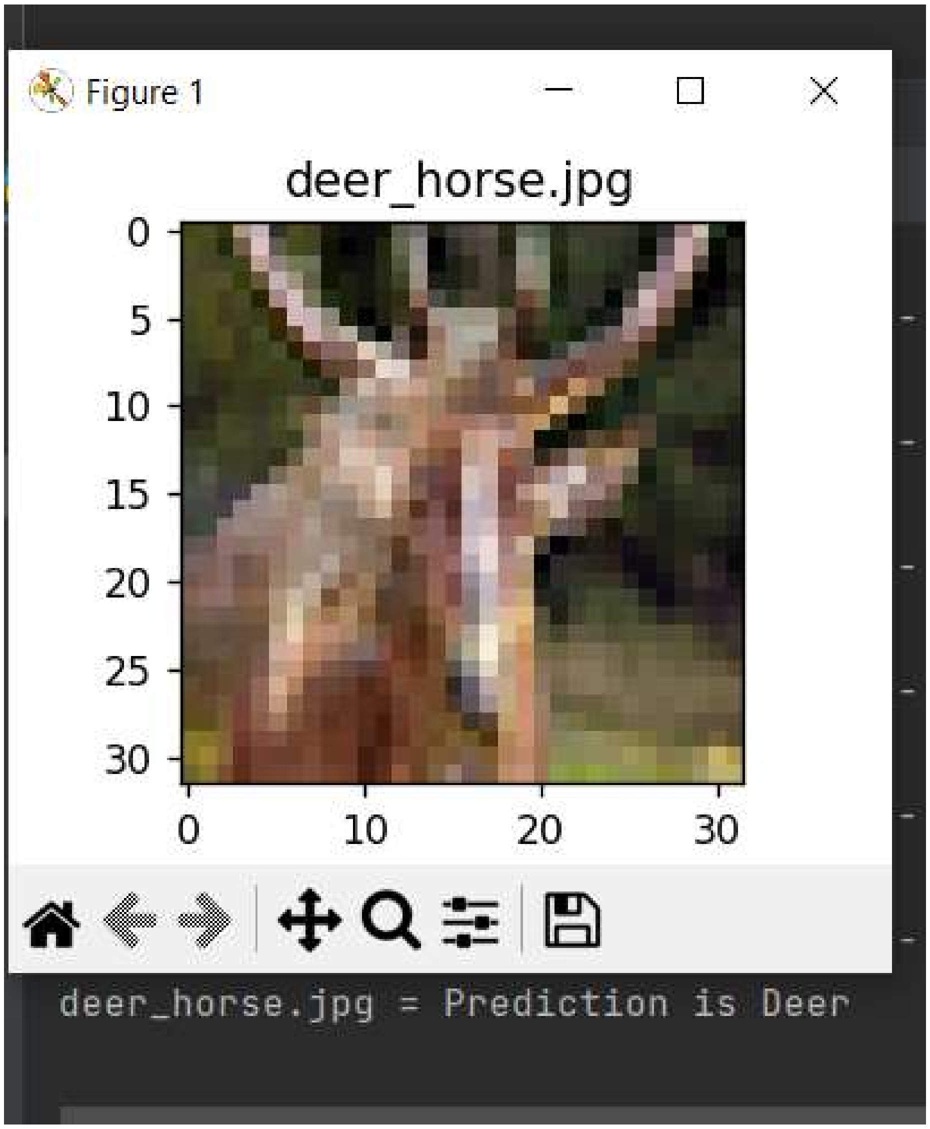
<u>**-case 2:**</u>
60 % cat + 40 % dog:



```
cat_dog.jpg = Prediction is Cat
```

-case 3:

60% deer + 40% horse :



```
deer_horse.jpg = Prediction is Deer
```