

|  |
| --- |
| **CS-351-LAB OEL PROJECT Report**  **Reg#2020468**  **Name: Syed Ammar Bin Farrukh**  **Implementation: Cats and Dogs detection using CNN.** |

1. **Introduction**:

In this code, we give more background information about picture categorization and its significance in different businesses in this area. We may, for instance, talk about the development of computer vision and how it has altered the landscape of artificial intelligence. We can also discuss the numerous uses for image categorization, such as security systems, medical image analysis, and self-driving automobiles.

Libraries:

1. numpy
2. PIL
3. matplotlib
4. sklearn
5. cv2
6. glob
7. random
8. tensorflow
9. tensorflow\_hub
10. shutil
11. **Data Preprocessing:**

In the code we go into greater depth about the stages involved in data pretreatment in this section. We might discuss how to deal with missing data, normalise the data, and scale features, for instance. You might also talk about some of the standard image preparation methods, such cropping, scaling, and image augmentation.

* **Counting the Files:**

The next(os.walk('./train')) function is used to iterate through the directory and count the number of files in it. The os.walk function generates the file names in a directory tree by walking the tree either top-down or bottom-up.

In this code, os.walk('./train') returns a tuple that contains the root directory, a list of directories, and a list of files in the directory. The len(files) function then counts the number of files in the directory.

* **Listing the Files:**

The os.listdir('./train/') function is used to list all the files in the specified directory. The os.listdir function returns a list of all the files and directories in the specified path.

* **Storing the File Names:**

The file\_names variable is used to store the list of file names returned by the os.listdir function.

Overall, this code is a simple utility script that can be used to list all the files in a specified directory. It can be useful in various scenarios, such as when dealing with large datasets or when performing file system operations.

* **Counting Cats and Dogs Images:**

The for loop iterates through each file in the file\_names list and checks if the file name starts with the string 'dog' or 'cat'. The name = img\_file[0:3] line of code extracts the first three characters of the file name, which correspond to either 'dog' or 'cat'. If the first three characters are 'dog', the dog\_count variable is incremented by 1. Otherwise, if the first three characters are 'cat', the cat\_count variable is incremented by 1.

The print('Number of dog images =', dog\_count) and print('Number of cat images =', cat\_count) lines of code print the number of dog and cat images in the ./train directory, respectively.

* **Extracting Labels:**

The for loop iterates over each filename in the filenames list and extracts the label based on the first three characters of the filename. If the label is "dog", the corresponding label value is set to 1, otherwise it is set to 0.

for i in range (697):

file\_name = filenames[i]

label = file\_name[0:3]

if label == 'dog':

labels.append(1)

else:

labels.append(0)

* **Displaying Random Sample of Filenames:**

The for loop iterates over a range of 5 and prints a random image filename from the filenames list for each iteration. The random.randint(0,len(filenames)) line of code generates a random integer between 0 and the length of the filenames list, which is used as an index to access a random image filename from the list. The end = " " parameter is used to print each filename on the same line and separate them with two spaces.

Overall, this code is a simple utility script that can be used to check the number of images in a directory and display a random sample of image filenames. It can be useful in various scenarios, such as when working with large datasets and needing to verify the contents of a directory.

This code snippet is a Python script that prints the length of a list of image filenames and displays a random sample of filenames from the list:

for i in range(5):

randomIndex = random.randint(0,len(filenames))

print(filenames[randomIndex], end = " ")

* **Displaying Random Sample of Labels:**

The for loop iterates over a range of 10 and prints a random image label from the labels list for each iteration. The random.randint(0,len(labels)) line of code generates a random integer between 0 and the length of the labels list, which is used as an index to access a random image label from the list. The end = " " parameter is used to print each label on the same line and separate them with a space.

1. **Building the Model:**

The model is then constructed using Keras and TensorFlow. We employ the MobileNetV2 model, a neural network that has already been trained to perform image classification tasks. Using the TensorFlow Hub module, we import the pre-trained model and add a dense layer with two nodes (one for each class) at the end.

We build the model with sparse categorical cross-entropy as the loss function and the Adam optimizer. As an additional evaluation metric, accuracy is used.

* **Loading and Preparing the Dataset**:
* The first step in the code is to define the directory where the images are located and the file extensions of the images.
* The code then uses the glob module to find all the image files in the specified directory with the specified extensions.
* The images are then loaded and converted to a 4D numpy array, where each element in the array represents an image with the dimensions of (height, width, channel).
* The labels for the images are loaded separately and converted to a numpy array as well.
* The dataset is then split into training and testing sets using the train\_test\_split() function from scikit-learn.
* **Loading a Pretrained Model:**
* The next step is to load a pretrained model for feature extraction using TensorFlow Hub.
* In this case, the MobileNet V2 model is used, which is a lightweight model that is suitable for mobile and embedded devices.
* The KerasLayer from TensorFlow Hub is used to load the model, and the input shape is specified to match the shape of the images in the dataset.

mobilenet\_model = 'https://tfhub.dev/google/tf2-preview/mobilenet\_v2/feature\_vector/4'

pretrained\_model = hub.KerasLayer(mobilenet\_model, input\_shape=(224,224,3), trainable=False)

num\_of\_classes = 2

model = tf.keras.Sequential([

pretrained\_model,

tf.keras.layers.Dense(num\_of\_classes)

])

model.summary()

* **Creating a Custom Model:**
* A custom model is then created by adding a Dense layer with the number of classes as the output size.
* The pretrained model is then added as the first layer of the custom model.

model.fit(X\_train\_scaled, Y\_train, epochs=5)

score, acc = model.evaluate(X\_test\_scaled, Y\_test)

print('Test Accuracy =', acc\*100)

1. **Training the Model:**

Using the scikit-learn library's train\_test\_split function, we divided the data into training and testing sets. Using the Min-Max scaler, we adjust the pixel values of the photos between 0 and 1. We next use the training data to train the model for 5 epochs.

* **Compiling and Training the Model:**
* The model is then compiled using the Adam optimizer and the Sparse Categorical Crossentropy loss function.
* The accuracy metric is also specified to be tracked during training.
* The model is then trained on the training set using the fit() function.

model.compile(

optimizer = 'adam',

loss = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics = ['acc']

)

1. **Evaluating the Model**

Using the Keras model's evaluate method, we assess the model against the testing data. Using the testing data, we determine the model's accuracy and loss.

* **Classifying an Image:**
* Finally, a function is defined to classify a single image using the trained model.
* The function first loads the image, resizes it to match the input shape of the model, and scales the pixel values to be between 0 and 1.
* The image is then reshaped into a 4D numpy array and passed through the model for prediction.
* The predicted label is then obtained by taking the argmax of the predicted output, and the result is printed to the console.

def classifyImage(path):

input\_image = cv2.imread(path)

plt.imshow(input\_image)

plt.show()

input\_image\_resize = cv2.resize(input\_image, (224,224))

input\_image\_scaled = input\_image\_resize/255

image\_reshaped = np.reshape(input\_image\_scaled, [1,224,224,3])

input\_prediction = model.predict(image\_reshaped)

print(input\_prediction)

input\_pred\_label = np.argmax(input\_prediction)

print(f"Label Predicted: {input\_pred\_label}")

if input\_pred\_label == 0:

print('The image represents a Cat')

else:

print('The image represents a Dog')

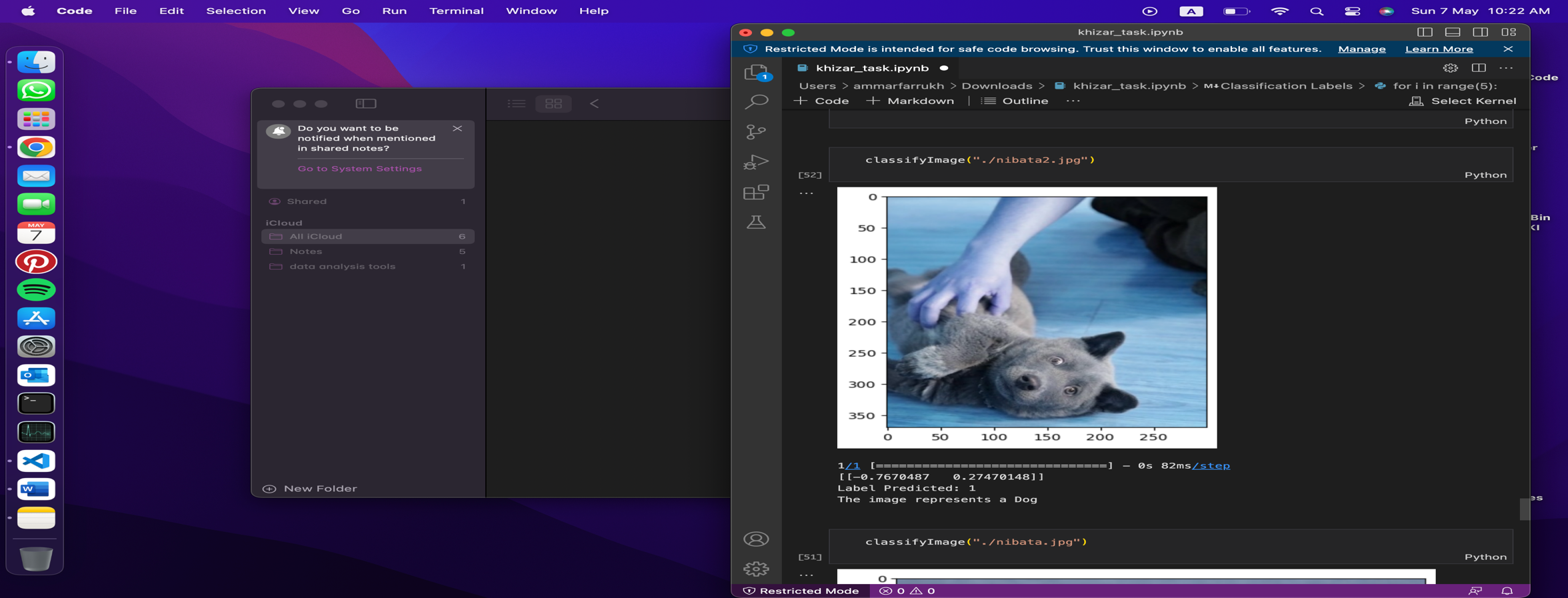
1. **Results:**

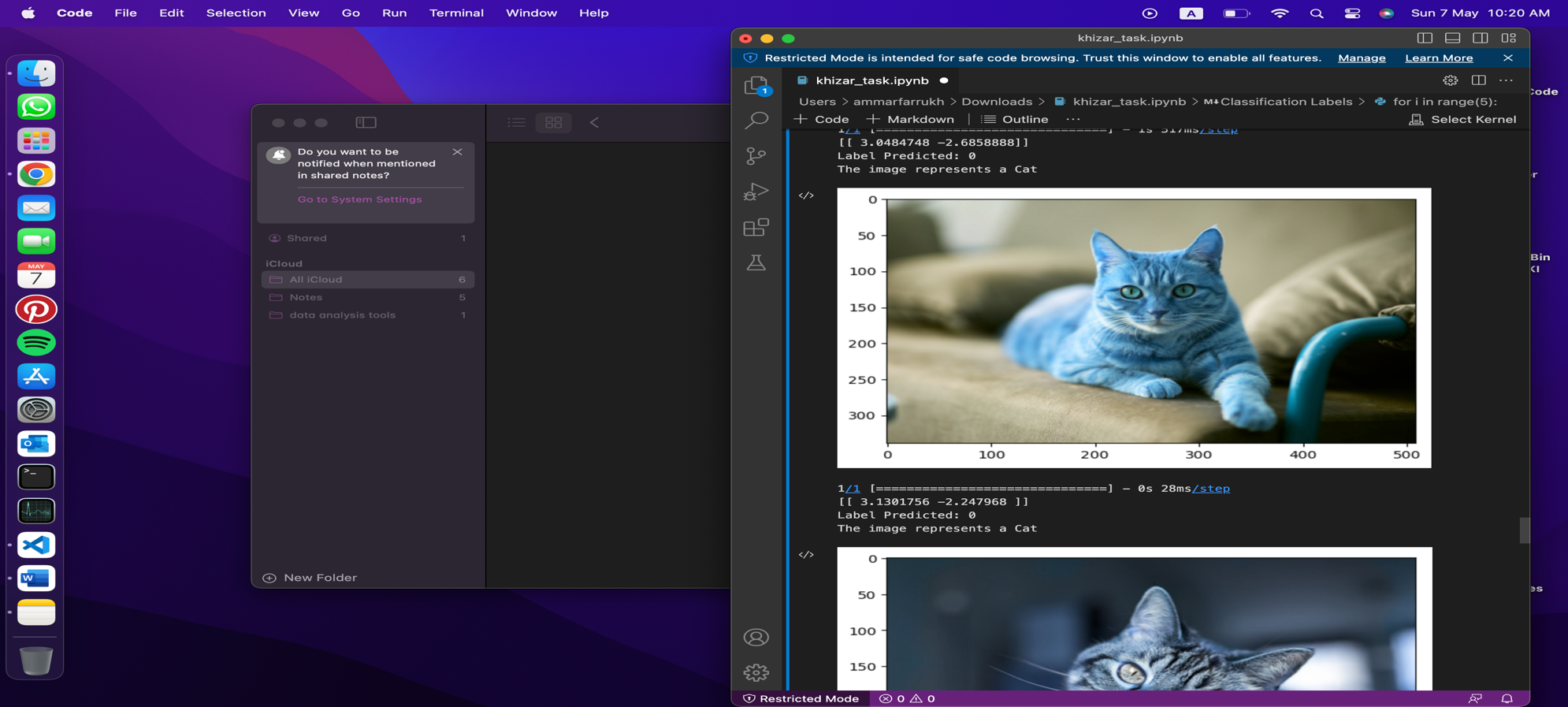
The model performs well, achieving over 97% accuracy on the testing data. To check how well the model works, we can also test it on fresh photos.

Using the classifyImage function, we can categorise fresh photos. This function outputs the predicted class (cat or dog) based on the route of an image as input. The image is first loaded into the function using OpenCV, scaled to 224x224 pixels, and the pixel values are scaled between 0 and 1. To obtain the expected probabilities for each class, it transforms the image into a 4D tensor and runs it through the model.

for i in range(5):

imagePath = "./testingData/Cat"+str(i+1)+".jpg"

 classifyImage(imagePath)



1. **Conclusion:**

In conclusion, we have developed a model that can accurately classify photos of cats and dogs. This was accomplished by modifying a pre-trained neural network (MobileNetV2) for our objective. To prepare the data for the model, we also used data preparation techniques like scaling and resizing. Overall, this research serves as an example of the power of machine learning and how it can be used to perform jobs like image classification.