Image Classification using Machine learning

A COURSE PROJECT REPORT

By

KHUSHI ARORA (RA1911003010198)

HEENA RUNGTA (RA1911003010207)

ARYAN MALHOTRA (RA1911003010242)

TUSHAR SOLANKI (RA1911003010249)

Under the guidance of

MS. P. NITHYAKANI

In partial fulfilment for the Course

Of

18CSC305J-ARTIFICIAL INTELLIGENCE

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chenpalpattu District

APRIL 2022

INTRODUCTION

The report provides the information about IMAGE CLASSIFICATION USING MACHINE LEARNING. It describes the content of an image using properly formed English sentences. The description captures objects contained in an image and expresses how these objects relate to each other and the activities they are involved in. The classification problem is to categorize all the pixels of a digital image into one of the defined classes.

The main aim of image classification in my project is to classify a given image into one of the categories using Machine and Deep Learning and understand the difference in classification. The task of image classification is to teach the computer to recognize images and classify them into one of the trained categories. To do so, we first need to teach the computer how a cat, a dog, a bird, etc. look like before it being able to recognize a new object. The more cats the computer sees, the better it gets in recognizing cats. This is known as supervised learning. We can carry this task by inputting labelled images, the computer will start recognizing patterns present in cat pictures that are absent from other ones and will start building its own cognition. We make use of Python to write the program.

Deep Learning: It is a subset of Machine Learning Algorithms that is very good at recognizing patterns but typically requires a large number of data. Deep learning excels in recognizing objects in images as its implemented using 3 or more layers of artificial neural networks where each layer is responsible for extracting one or more feature of the image.

MOTIVATION

Computer vision is actually able to do really complex tasks that have changed (and will change) the way we live. Nonetheless, in the 21st century, sometimes we are able to make complex things really easy, and there is no need to write tons of code lines to have acceptable results. The main motive behind doing this project was to make it more accurate than the older ones. Along with making it more simple and easy to use. We have tried to make it user friendly for beginners.

NOVELTY

1. **Accuracy**: Erroneous values that deviate from the expected. The causes for inaccurate data can vary, but include:
   - Human/computer errors during data entry and transmission
   - Users deliberately submitting incorrect values (called disguised missing data)
   - Incorrect formats for input field.
   - Duplication of training examples
2. **Completeness**: Lacking attribute/feature values or values of interest. The data set might be incomplete due to:
   a. Unavailability of data
   b. Deletion of inconsistent data
   c. Deletion of data deemed irrelevant initially

3. **Consistency**: Aggregation of data is inconsistent.

Techniques used in the project

DATA PRE – PROCESSING

In any Machine Learning process, Data Pre-processing is that step in which the data gets transformed, or Encoded to bring it to such a state that now the machine can easily parse it. In other words, the features of the data can now be easily interpreted by the algorithm.

Pre-processing is required to clean image data for model input. For example, fully connected layers in convolutional neural networks required that all images are the same sized arrays.

Image pre-processing may also decrease model training time and increase model inference speed. If input images are particularly large, reducing the size of these images will dramatically improve model training time without significantly reducing model performance. For example, the standard size of images on iPhone 11 are 3024×4032. The machine learning model Apple uses to create masks and apply Portrait Mode performs on images half this size before its output is rescaled back to full size. Image pre-processing are the steps taken to format images before they are used by model training and inference. This includes, but is not limited to, resizing, orienting, and colour corrections.

Image augmentation creates new training examples out of existing training data. It's impossible to truly capture an image that accounts for every real world scenario a model may encompass. Adjusting existing training data to generalize to other situations allows the model to learn from a wider array of situations.

This is particularly important when collected datasets may be small. A deep learning model will (over)fit to the examples shown in training, so creating variation in the input images enables generation of new, useful training examples.

FEATURE EXTRACTION

The technique of extracting the features is useful when you have a large data set and need to reduce the number of resources without losing any important or relevant information. Feature extraction helps to reduce the amount of redundant data from the data set.

In the end, the reduction of the data helps to build the model with less machine's efforts and also increase the speed of learning and generalization steps in the machinelearning process.

In this domain basically you will start playing with your images in order to understand them. So here we use many techniques which includes feature extraction as well and algorithms to detect features such as shaped, edges, or motion in a digital image or video to process them.

Machines see any images in the form of a matrix of numbers. The size of this matrix actually depends on the number of pixels of the input image. The Pixel Values for each of the pixels stands for or describe how bright that pixel is, and what colour it should be. So In the simplest case of the binary images, the pixel value is a 1-bit number indicating either foreground or background. So pixels are the numbers, or the pixel values which denote the intensity or brightness of the pixel.

ML MODE

A machine learning model is a file that has been trained to recognize certain types of patterns. You train a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data.

Once we have trained the model, we can use it to reason over data that it hasn't seen before, and make predictions about those data. For example, let's say we want to build an application that can recognize a user's emotions based on their facial expressions. We can train a model by providing it with images of faces that are each tagged with a certain emotion, and then we can use that model in an application that can recognize any user's emotion.

Good machine learning scenarios often have the following common properties:

- They involve a repeated decision or evaluation which we want to automate and need consistent results.
- It is difficult or impossible to explicitly describe the solution or criteria behind a decision.

- We have labelled data, or existing examples where we can describe the situation and map it to the correct result.

HOG

Hog in machine learning: HOG, or Histogram of Oriented Gradients, is a feature descriptor that is often used to extract features from image data. It is widely used in computer vision tasks for object detection. The technique counts occurrences of gradient orientation in localized portions of an image or region of interest.

The HOG features are widely used for object detection. HOG decomposes an image into small squared cells, computes a histogram of oriented gradients in each cell, normalizes the result using a block-wise pattern, and return a descriptor for each cell.

PIPELINE MODEL

One definition of an ML pipeline is a means of automating the machine learning workflow by enabling data to be transformed and correlated into a model that can then be analysed to achieve outputs. This type of ML pipeline makes the process of inputting data into the ML model fully automated.

Another type of ML pipeline is the art of splitting up your machine learning workflows into independent, reusable, modular parts that can then be pipelined together to create models. This type of ML pipeline makes building models more efficient and simplified, cutting out redundant work

This goes hand-in-hand with the recent push for micro services architectures, branching off the main idea that by splitting your application into basic and soloed parts that we can build more powerful software over time. Operating systems like Linux and UNIX are also founded on this principle. Basic functions like ‘grep’ and ‘cat’ can create impressive functions when they are pipelined together.

How ML pipelines benefit performance and organization

- Scheduling and runtime optimization
- Language and framework agnosticism
- Broader applicability and fit.

MODELS USED IN THIS PROJECT

SCIKIT-IMAGE

Scikit-image is an open-source image processing library for the Python programming language. It includes algorithms for segmentation, geometric transformations, colour space manipulation, analysis, filtering, morphology, feature detection, and more. It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

NUMPY

NumPy arrays are the basis of all computations performed by the NumPy library. They are simple Python lists with a few additional properties. A NumPy array has the dtype attribute, which specifies the data type of all the elements in the array.

If the array contains elements of different data types, all the elements are cast into the largest type (a process known as upcasting). A NumPy array can also be a copy or a reference to an existing NumPy array.

PANDAS

Pandas (all lowercase) is a popular Python-based data analysis toolkit which can be imported using import pandas as pd. It presents a diverse range of utilities, ranging from parsing multiple file formats to converting an entire data table into a NumPy matrix array. This makes pandas a trusted ally in data science and machine learning.

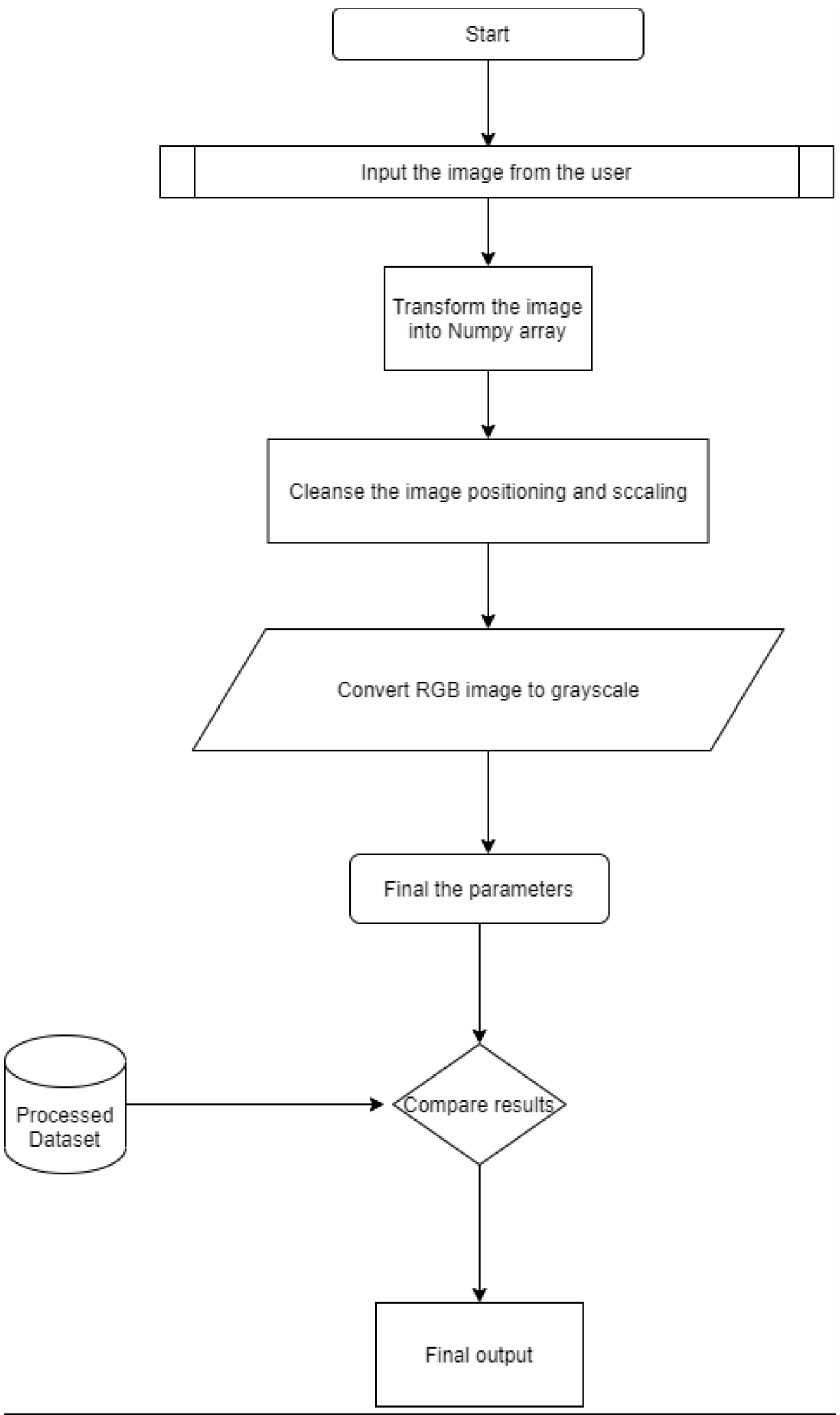
Similar to NumPy, pandas deals primarily with data in 1-D and 2-D arrays; however, pandas handles the two differently.

DATASET

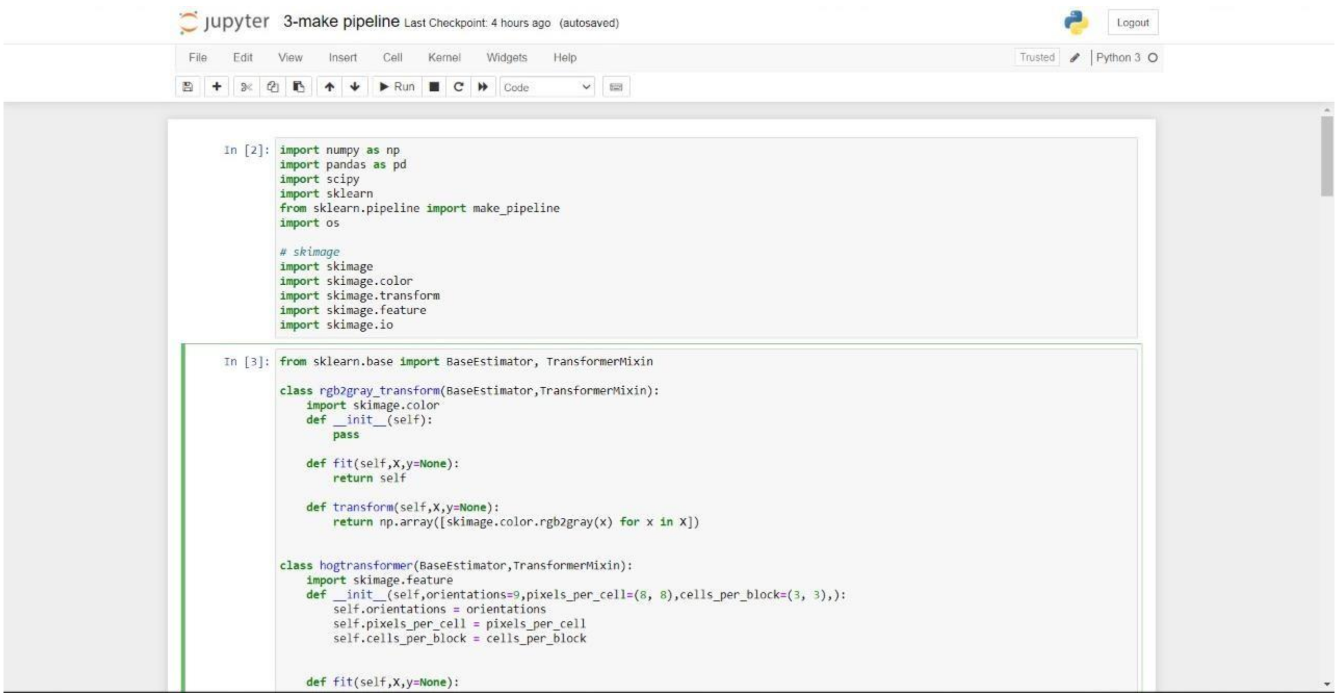
A data set (or dataset) is a collection of data. In the case of tabular data, a data set corresponds to one or more database tables, where every column of a table represents a particular variable, and each row corresponds to a given record of the data set in question. The data set lists values for each of the variables, such as height and weight of an object, for each member of the data set. Each value is known as a datum. Data sets can also consist of a collection of documents or files.

We have used pre-defined datasets which consists of 2000+ images which were tested and sample collected.

Flowchart

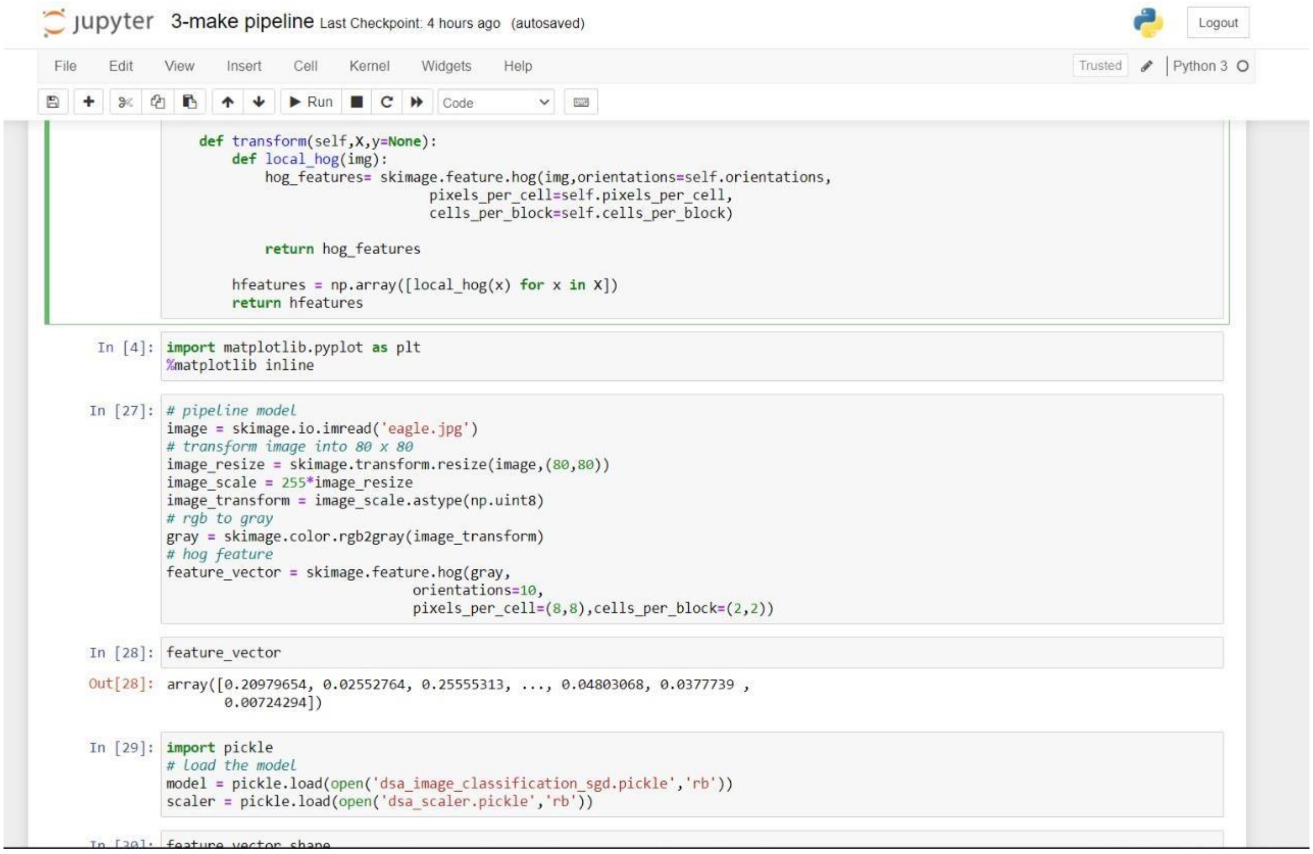

Start
Input the image from the user
Transform the image into Numpy array
Cleanse the image positioning and sccaling
Convert RGB image to grayscale
Final the parameters
Processed Dataset
Compare results
Final output

IMPLEMENTATION AND CODING



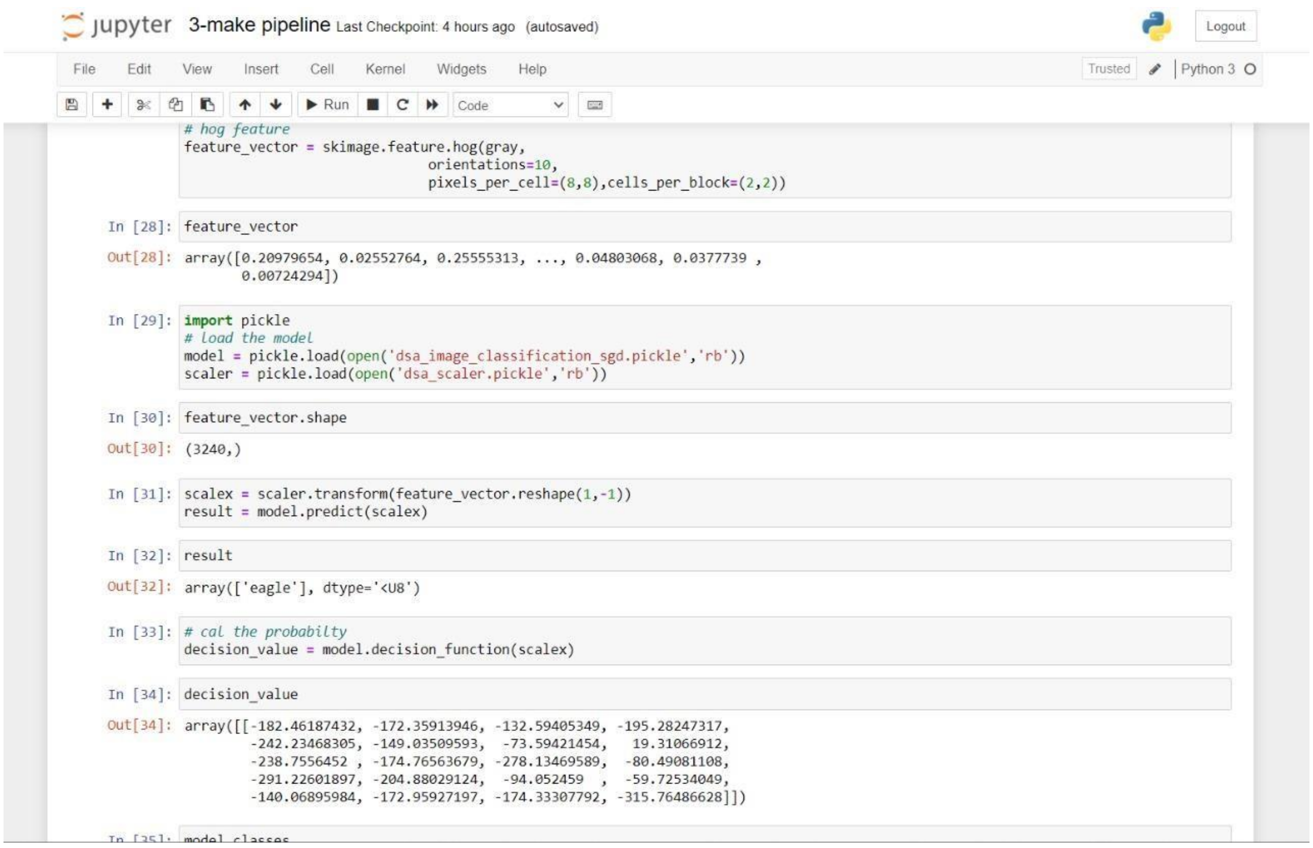
jupyter 3-make pipeline Last Checkpoint: 4 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [2]: import numpy as np
        import pandas as pd
        import scipy
        import sklearn
        from sklearn.pipeline import make_pipeline
        import os

        # skimage
        import skimage
        import skimage.color
        import skimage.transform
        import skimage.feature
        import skimage.io

In [3]: from sklearn.base import BaseEstimator, TransformerMixin

        class rgb2gray_transform(BaseEstimator,TransformerMixin):
            import skimage.color
            def __init__(self):
                pass

            def fit(self,X,y=None):
                return self

            def transform(self,X,y=None):
                return np.array([skimage.color.rgb2gray(x) for x in X])


        class hogtransformer(BaseEstimator,TransformerMixin):
            import skimage.feature
            def __init__(self,orientations=9,pixels_per_cell=(8, 8),cells_per_block=(3, 3),):
                self.orientations = orientations
                self.pixels_per_cell = pixels_per_cell
                self.cells_per_block = cells_per_block


            def fit(self,X,y=None):
```



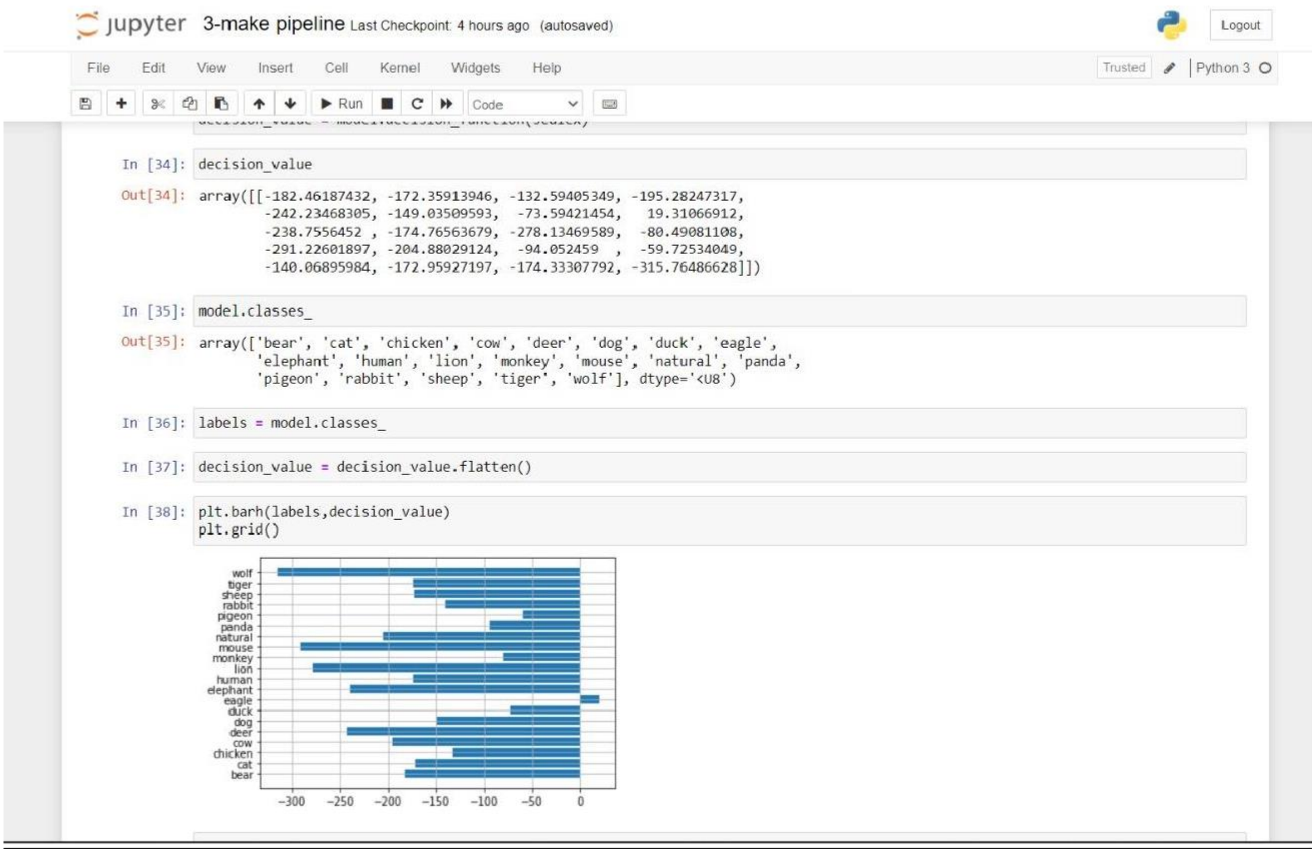
jupyter 3-make pipeline Last Checkpoint: 4 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
            def transform(self,X,y=None):
                def local_hog(img):
                    hog_features= skimage.feature.hog(img,orientations=self.orientations,
                                        pixels_per_cell=self.pixels_per_cell,
                                        cells_per_block=self.cells_per_block)

                    return hog_features

                hfeatures = np.array([local_hog(x) for x in X])
                return hfeatures

In [4]: import matplotlib.pyplot as plt
        %matplotlib inline

In [27]: # pipeline model
         image = skimage.io.imread('eagle.jpg')
         # transform image into 80 x 80
         image_resize = skimage.transform.resize(image,(80,80))
         image_scale = 255*image_resize
         image_transform = image_scale.astype(np.uint8)
         # rgb to gray
         gray = skimage.color.rgb2gray(image_transform)
         # hog feature
         feature_vector = skimage.feature.hog(gray,
                                       orientations=10,
                                       pixels_per_cell=(8,8),cells_per_block=(2,2))

In [28]: feature_vector

Out[28]: array([0.20979654, 0.02552764, 0.25555313, ..., 0.04803068, 0.0377739 ,
                0.00724294])

In [29]: import pickle
         # load the model
         model = pickle.load(open('dsa_image_classification_sgd.pickle','rb'))
         scaler = pickle.load(open('dsa_scaler.pickle','rb'))

In [30]: feature_vector.shape
```

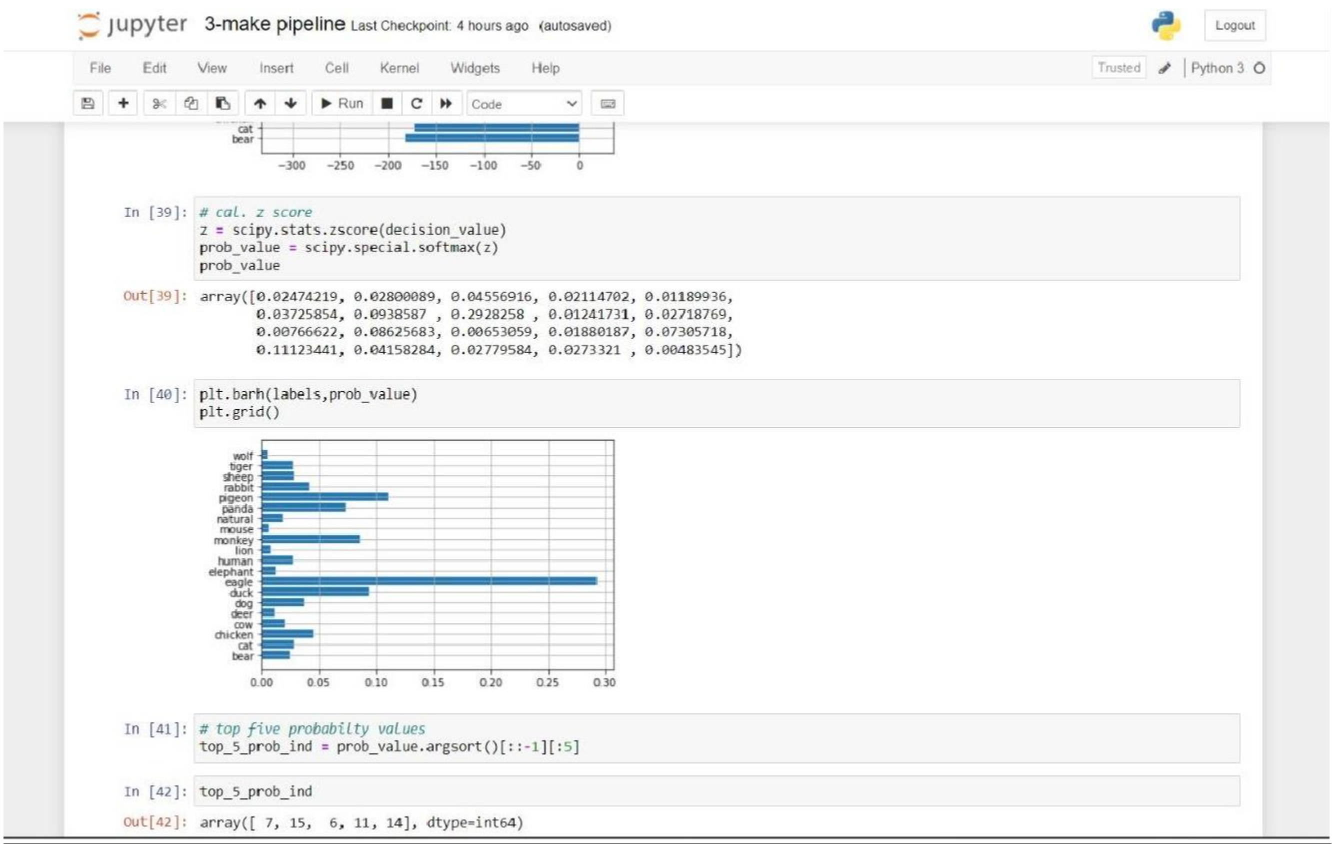


```
              # hog feature
              feature_vector = skimage.feature.hog(gray,
                                            orientations=10,
                                            pixels_per_cell=(8,8),cells_per_block=(2,2))

     In [28]: feature_vector

    Out[28]: array([0.20979654, 0.02552764, 0.25555313, ..., 0.04803068, 0.0377739 ,
                    0.00724294])

     In [29]: import pickle
              # load the model
              model = pickle.load(open('dsa_image_classification_sgd.pickle','rb'))
              scaler = pickle.load(open('dsa_scaler.pickle','rb'))

     In [30]: feature_vector.shape

    Out[30]: (3240,)

     In [31]: scalex = scaler.transform(feature_vector.reshape(1,-1))
              result = model.predict(scalex)

     In [32]: result

    Out[32]: array(['eagle'], dtype='<U8')

     In [33]: # cal the probabilty
              decision_value = model.decision_function(scalex)

     In [34]: decision_value

    Out[34]: array([[-182.46187432, -172.35913946, -132.59405349, -195.28247317,
                     -242.23468305, -149.03509593,  -73.59421454,   19.31066912,
                     -238.7556452 , -174.76563679, -278.13469589,  -80.49081108,
                     -291.22601897, -204.88029124,  -94.052459  ,  -59.72534049,
                     -140.06895984, -172.95927197, -174.33307792, -315.76486628]])

     In [35]: model.classes
```

OUTPUT

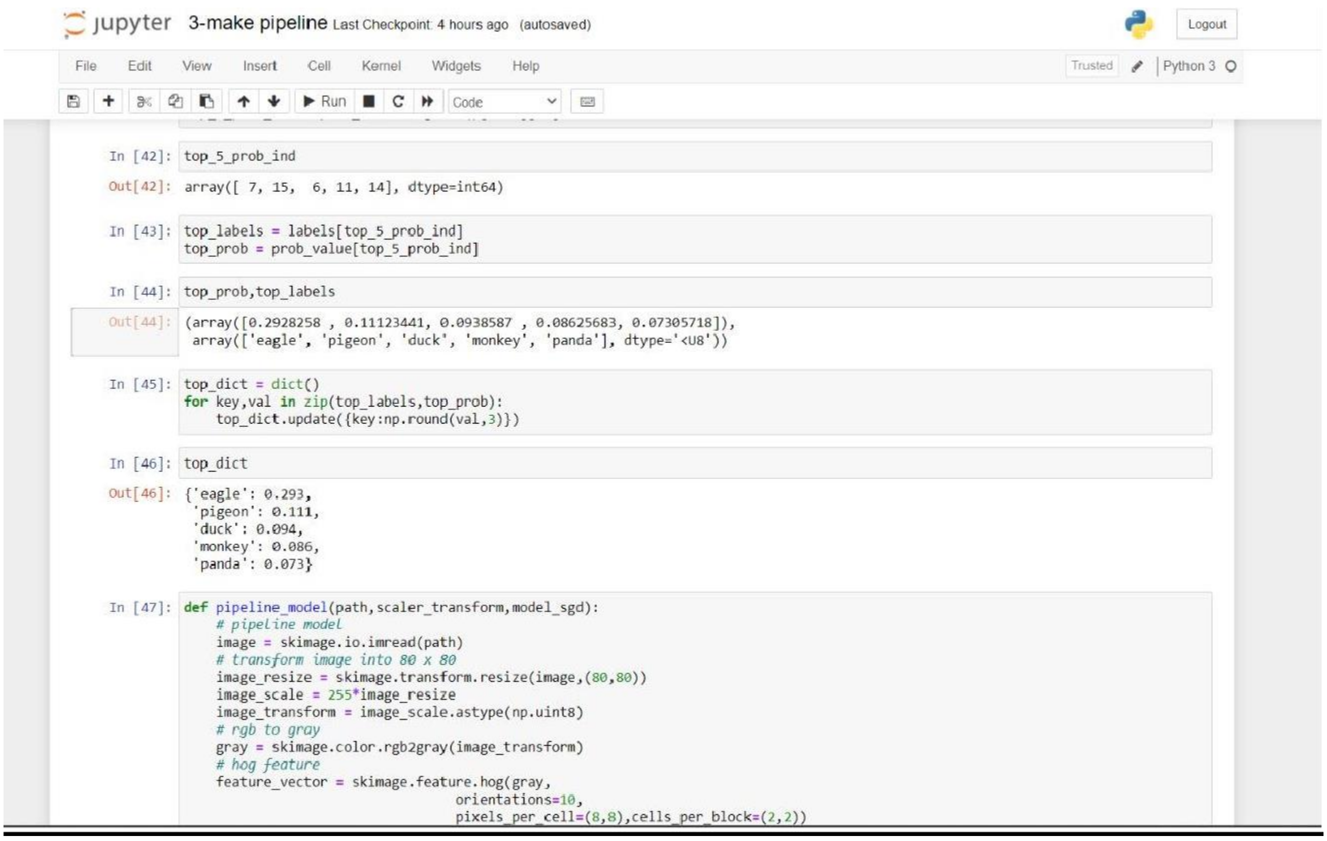

jupyter 3-make pipeline Last Checkpoint: 4 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [34]: decision_value

Out[34]: array([[-182.46187432, -172.35913946, -132.59405349, -195.28247317,
                 -242.23468305, -149.03509593,  -73.59421454,   19.31066912,
                 -238.7556452 , -174.76563679, -278.13469589,  -80.49081108,
                 -291.22601897, -204.88029124,  -94.052459  ,  -59.72534049,
                 -140.06895984, -172.95927197, -174.33307792, -315.76486628]])

In [35]: model.classes_

Out[35]: array(['bear', 'cat', 'chicken', 'cow', 'deer', 'dog', 'duck', 'eagle',
                'elephant', 'human', 'lion', 'monkey', 'mouse', 'natural', 'panda',
                'pigeon', 'rabbit', 'sheep', 'tiger', 'wolf'], dtype='<U8')

In [36]: labels = model.classes_

In [37]: decision_value = decision_value.flatten()

In [38]: plt.barh(labels,decision_value)
         plt.grid()
```

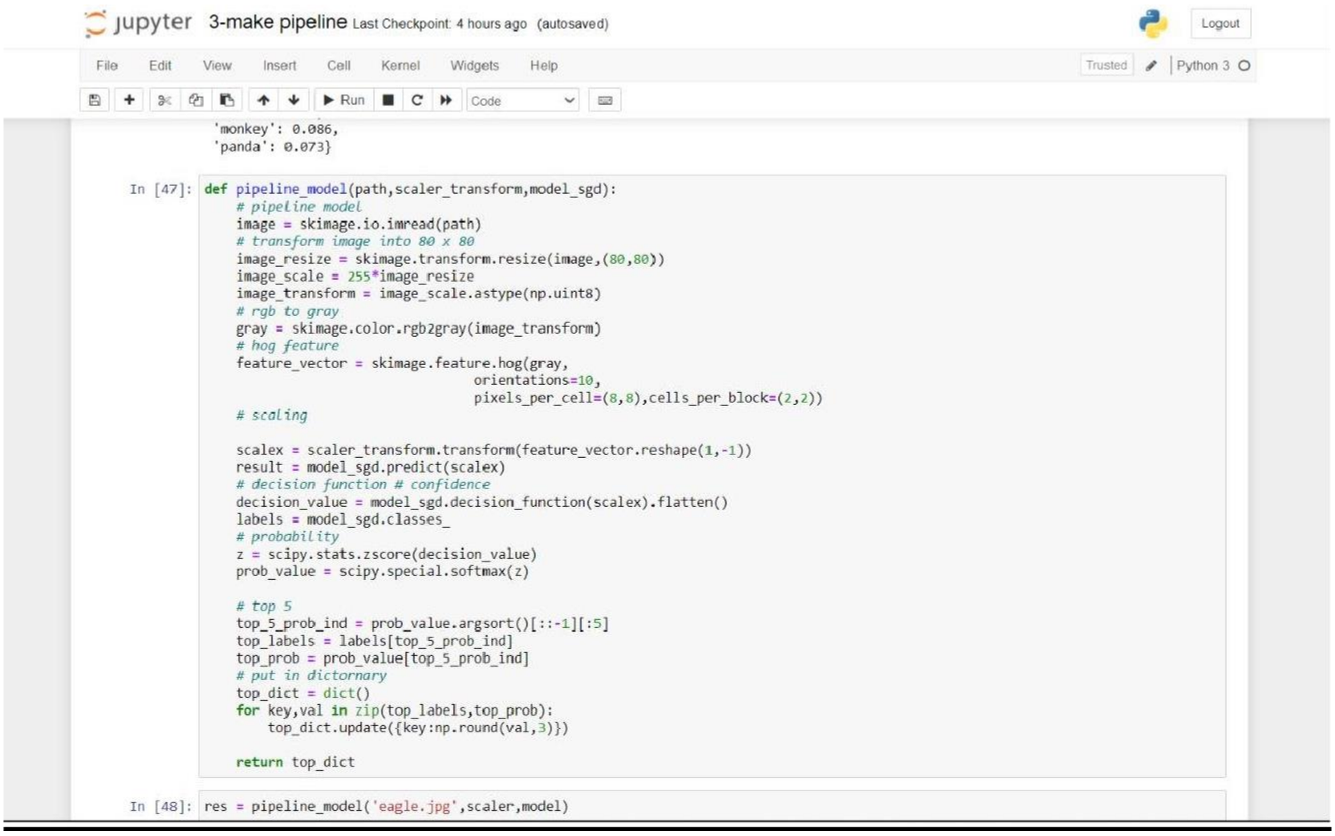

jupyter 3-make pipeline Last Checkpoint: 4 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
cat
bear
−300
−250
−200
−150
−100
−50
0

```
In [39]: # cal. z score
         z = scipy.stats.zscore(decision_value)
         prob_value = scipy.special.softmax(z)
         prob_value

Out[39]: array([0.02474219, 0.02800089, 0.04556916, 0.02114702, 0.01189936,
                0.03725854, 0.0938587 , 0.2928258 , 0.01241731, 0.02718769,
                0.00766622, 0.08625683, 0.00653059, 0.01880187, 0.07305718,
                0.11123441, 0.04158284, 0.02779584, 0.0273321 , 0.00483545])

In [40]: plt.barh(labels,prob_value)
         plt.grid()
```

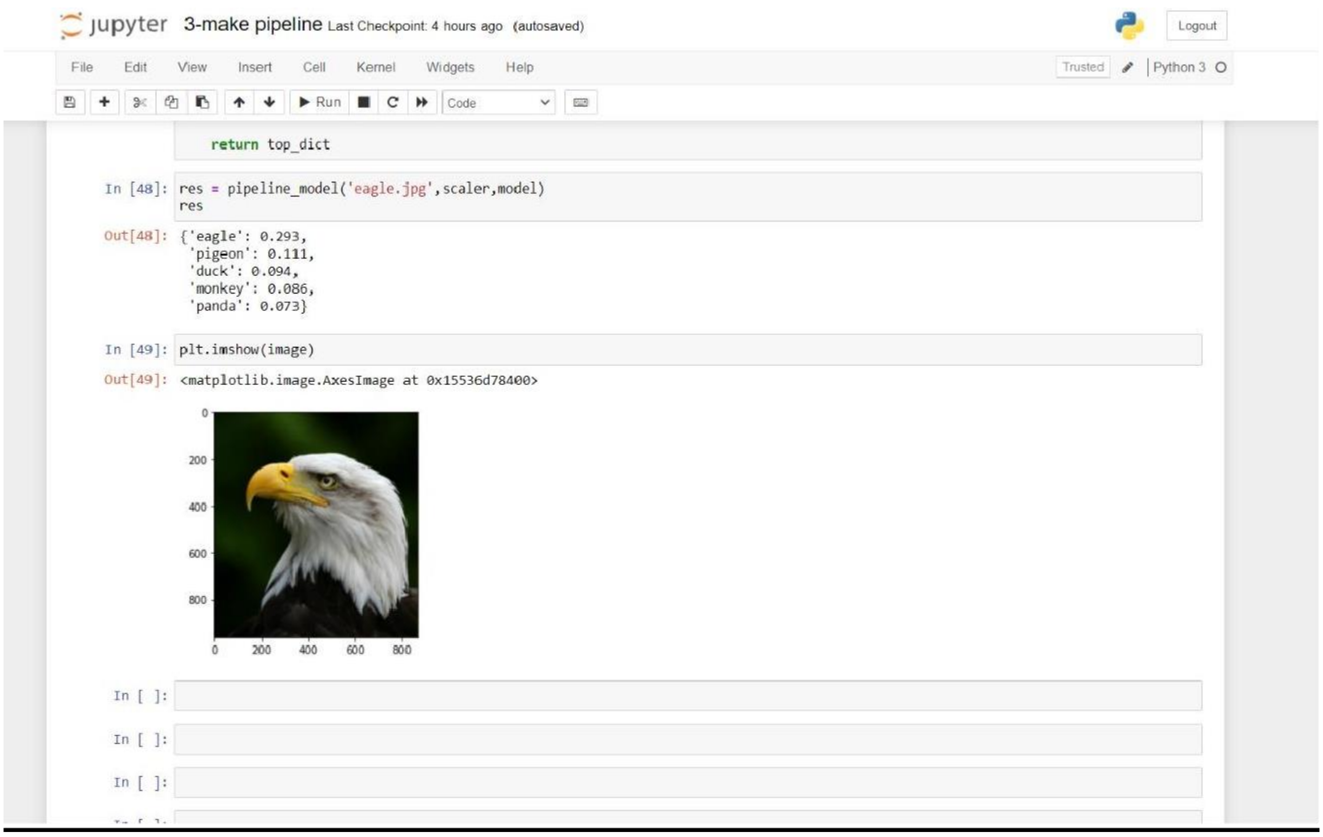
```
In [41]: # top five probabilty values
         top_5_prob_ind = prob_value.argsort()[::-1][:5]

In [42]: top_5_prob_ind

Out[42]: array([ 7, 15,  6, 11, 14], dtype=int64)
```


jupyter 3-make pipeline Last Checkpoint: 4 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [42]: top_5_prob_ind
Out[42]: array([ 7, 15,  6, 11, 14], dtype=int64)

In [43]: top_labels = labels[top_5_prob_ind]
         top_prob = prob_value[top_5_prob_ind]

In [44]: top_prob,top_labels
Out[44]: (array([0.2928258 , 0.11123441, 0.0938587 , 0.08625683, 0.07305718]),
          array(['eagle', 'pigeon', 'duck', 'monkey', 'panda'], dtype='<U8'))

In [45]: top_dict = dict()
         for key,val in zip(top_labels,top_prob):
             top_dict.update({key:np.round(val,3)})

In [46]: top_dict
Out[46]: {'eagle': 0.293,
          'pigeon': 0.111,
          'duck': 0.094,
          'monkey': 0.086,
          'panda': 0.073}

In [47]: def pipeline_model(path,scaler_transform,model_sgd):
             # pipeline model
             image = skimage.io.imread(path)
             # transform image into 80 x 80
             image_resize = skimage.transform.resize(image,(80,80))
             image_scale = 255*image_resize
             image_transform = image_scale.astype(np.uint8)
             # rgb to gray
             gray = skimage.color.rgb2gray(image_transform)
             # hog feature
             feature_vector = skimage.feature.hog(gray,
                                           orientations=10,
                                           pixels_per_cell=(8,8),cells_per_block=(2,2))
```


jupyter 3-make pipeline Last Checkpoint: 4 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
          'monkey': 0.086,
          'panda': 0.073}

In [47]: def pipeline_model(path,scaler_transform,model_sgd):
             # pipeline model
             image = skimage.io.imread(path)
             # transform image into 80 x 80
             image_resize = skimage.transform.resize(image,(80,80))
             image_scale = 255*image_resize
             image_transform = image_scale.astype(np.uint8)
             # rgb to gray
             gray = skimage.color.rgb2gray(image_transform)
             # hog feature
             feature_vector = skimage.feature.hog(gray,
                                           orientations=10,
                                           pixels_per_cell=(8,8),cells_per_block=(2,2))
             # scaling

             scalex = scaler_transform.transform(feature_vector.reshape(1,-1))
             result = model_sgd.predict(scalex)
             # decision function # confidence
             decision_value = model_sgd.decision_function(scalex).flatten()
             labels = model_sgd.classes_
             # probability
             z = scipy.stats.zscore(decision_value)
             prob_value = scipy.special.softmax(z)

             # top 5
             top_5_prob_ind = prob_value.argsort()[::-1][:5]
             top_labels = labels[top_5_prob_ind]
             top_prob = prob_value[top_5_prob_ind]
             # put in dictornary
             top_dict = dict()
             for key,val in zip(top_labels,top_prob):
                 top_dict.update({key:np.round(val,3)})

             return top_dict

In [48]: res = pipeline_model('eagle.jpg',scaler,model)
```


jupyter 3-make pipeline Last Checkpoint: 4 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
        return top_dict

In [48]: res = pipeline_model('eagle.jpg',scaler,model)
         res

Out[48]: {'eagle': 0.293,
          'pigeon': 0.111,
          'duck': 0.094,
          'monkey': 0.086,
          'panda': 0.073}

In [49]: plt.imshow(image)

Out[49]: <matplotlib.image.AxesImage at 0x15536d78400>
```

0
200
400
600
800
0
200
400
600
800

```
In [ ]:

In [ ]:

In [ ]:
```

RESULT ANALYSIS

The results obtained from the classification model yields an accuracy of about 99% which is better than some conventional techniques used in other classification methods. In future, the dataset can be improved by increasing its volume and adding more and more data to it.

CONCLUSION

Hence, we can conclude that our project, Image classification using Machine Learning was successfully done using Jupiter notebook in Python. It described the content of an image using properly formed English sentences. The description captured objects contained in an image and expressed how these objects relate to each other and the activities they were involved in.