Real Time Image Classification Method Using Computer Vision Techniques and Customized CNN Model

Cansu IŞIK (1906326), Ömer Faruk KÜÇÜK (1900195)

Bahçeşehir University

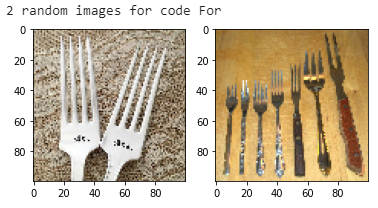
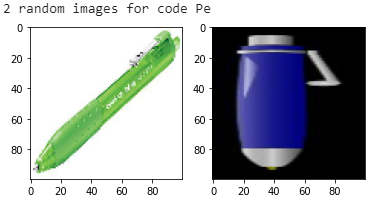
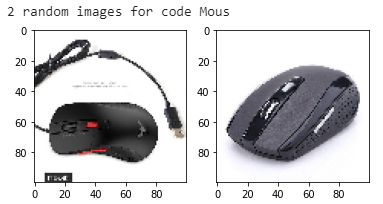
Abstract

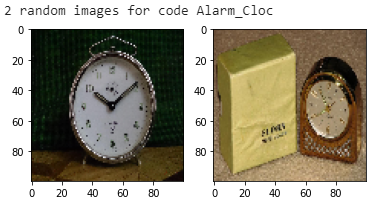
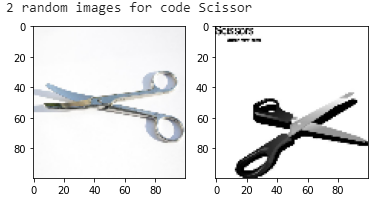
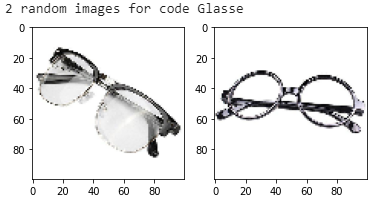
The project examines practices of image classification. The aim of the project is to create a CNN model that is made more efficient with the help of some Computer Vision techniques and compare the efficiency of our model with the MLP and SVM models. After having an accurate result, we created a real time application with the help of a pre-trained object detection algorithm, after the object detection is made, our image classification model classifies the image and gives us the label as a result.

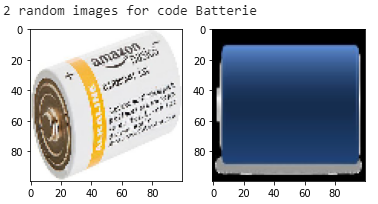
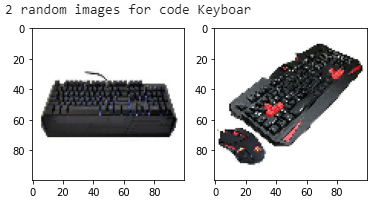
Keywords: CNN, MLP, SVM, Image Processing

1. Introduction

Image classification is a complex process that may be affected by many factors. This project examines practices of image classification. The emphasis is placed on the summarization of major advanced classification approaches and the techniques used for improving classification accuracy. Image classification is a image processing method which to distinguish between different categories of objectives according to the different features of images. It is widely used in pattern recognition and computer vision. Support Vector Machine is a machine learning method based on statistical learning theory. We designed an image classification algorithm based on SVM with RBF as kernel function. Image classification involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. In this project we focus on optimizing the image classification model. However boundary box is also implemented. We tried several models and increase their accuracies with proper methods. Then we used Caffe (MobileNet – SSD detection network) which is a deep learning framework made with expression, speed, and modularity in mind. Multilayer Perceptron used to be applied in computer vision, however MLP is now seems insufficient for modern computer vision tasks. Has the characteristic of fully connected layers, where each perceptron is connected with every other perceptron. Disadvantage is that the number of total parameters can grow to very high. This is inefficient because there is redundancy in such high dimensions. Another disadvantage is that it disregards spatial information. Nowadays, Convolutional Neural Network is favorite of computer vision algorithms. Each filter is panned around the entire image according to certain size, allows the filter to match patterns no matter where the pattern is located. The weights are smaller, and shared, easier to train than MLP. Layers are sparsely connected rather than fully connected. It takes matrices as well as vectors as inputs. The layers are sparsely connected or partially connected rather than fully connected. Because of the advantages of CNN over MLP network, we decided to use CNN as our main model.







1. Dataset Description

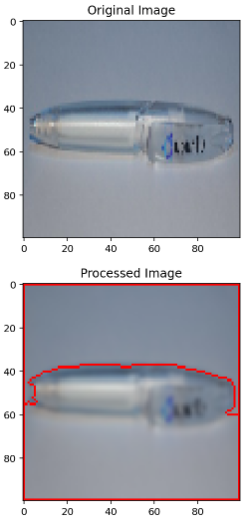
Our dataset consists of 1866 images that belongs to 9 different object types. Distribution of each class;

* Scissors: 273
* Pen: 253
* Mug: 250
* Mouse: 229
* Keyboard: 255
* Glasses: 173
* Fork: 133
* Batteries: 173
* Alarm Clock: 127

The most populated data is Scissors, the least populated data is Alarm Clock.

1. Data Preprocessing

In our dataset we have an original image and a marked image, also we are preprocessing the labels. The operations that we applied on the images are;

* **Resizing:** We are resizing each image to 100x100x3
* **Denoising:**
  + **Gaussian Blur:** We are applying Gaussian Blur to smooth our image and remove unwanted noise.
* **Segmentation and Morphology:**
  + **Graying and Tresholding:** We are segmenting the image, seperating background from foreground objects.
  + **Morphology:** To further noise removal, we are applying following techniques.
  + **Dilating:** To find a sure background area, we are using dilating.
  + **Distance Transform and Tresholding:** To find a sure foreground area, we are applying distance transform and tresholding again.
  + **Uint8 and Substracting:** Converting 0 to 1 image data to the 0 to 255 image data with uint8, then substracting background from foreground to find the unknown area.
  + **Connected Components:** We are labeling the markers, to seperate different objects in the image, we are doing the following.
  + **Other Operations:** Adding 1 to all labels to change the background form 0 to 1. Then we are marking the unknown areas with 0.
  + **Watersheding:** To mark the objects, we are watersheding.
* **Scaling:** Scaling the image array between 0 and 1.

The operation that we applied to the labels;

* **Binarizing:** Since our label data is categorical, we are binarizing the labels at the size of our total categories. When our model predicts a value, iw predicts a score for each label binary between 0 and 1. Biggest number is the strongest prediction.

1. Methods

Data Search and Downloading Application:

* Gathering the input data was a hard process. We had to be sure each image that belongs to a category should be evident and clear. At first we tried to run an algorithm to fetch images from the Google Images service. Then we saw that, there were too many not related images, so that we decided to find another source. There is a platform called ImageNet, this platform is a volunteer labeled image platform supported by Stanford and Princeton Universities and has more than 14 million labeled images. We wrote an algorithm that fetches images from ImageNet and downloads them to the specified folder with an ordered name. We controlled each image with eye to see if it is usable. At the end with the help of the ImageNet, Google and some other platforms we gathered an input data that has 1866 images in 9 categories.

Fetching the Data:

* To fetch the dataset, we wrote a python function that takes a base folder path. Then finds the subfolders and the files in that subfolders. This function is fetching the images, and their folder names. Then creates 2 images and 2 labels for each image at the input folder. For each image that fetched in the function loop, there is a normally processed image array and there is a processed image array that is both appended to the images array, besides of that the function takes the name of the folder that contains the image file and appends it to the labels array 2 times per images. Our "get\_image\_list\_from\_folder" function is compatible to our image downloading algorithm. After this operation we have a twin for each file, that is processed. At the end of the fetching phase, our dataset is two times bigger than before.

Image Augmentation:

* Image Augmentation is a technique for artificially expanding the size of the dataset. It creates modified versions of the images in the dataset. Showing a shape with different directions while training the model improves the training performance. Our modifications over the augmented data is 25 degree randomly rotation, 0.1 shifted pixels both width and height, 0.2 shear degree and zoom angle and for the last horizonally flipping.

Histogram of Oriented Gradients Transformation Method:

* The HOG descriptor focuses on the shape of an object. HOG is able to provide the edge direction. This is done by extracting the gradient and orientation of the edges. Additionally, these orientations are calculated in ‘localized’ portions. The complete image is broken down into smaller regions and for each region, the gradients and orientation are calculated. HOG would generate a Histogram for each of these regions separately. The histograms are created using the gradients and orientations of the pixel values.

Simple SVM Model:

* Support vector machines are supervised learning models with associated learning algorithms that we used to analyze our image data for classification. Firstly we apply preprocess to prepare data for model which includes reading images from different directories, resizing them to (100,100),reducing dimension, and converting them to array. We also convert labels to binary representations. Then we split data to train and test part with 30% test size. We apply GridSearchCV with different cost, kernel and gamma parameters. We checked the best score which was around 50%. We also checked the precision, recall and f1 score of our classes. According to the precision score we can say that pen is predicted more accurately then other classes since its precision is 82%. In the results from recall we saw that keyboard is has the best ratio. This is because our data has more keyboard images. According to the F1 score pen is predicted better than other objects. However our accuracy value for test dataset was too few therefore we realized that we need to apply more advanced techniques.

Object Based SVM &SGDC Models:

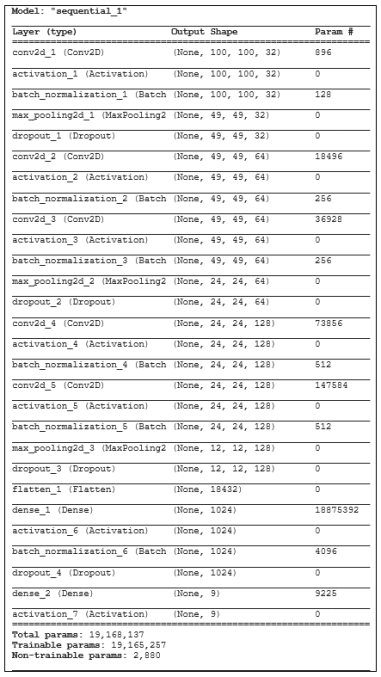
* We appled histogram of oriented gradients and Support Vector Machines method for lowering complexity of object based classification while maintaining variations. To calculate a histogram of oriented gradients, an image is divided into blocks. For each of these blocks the magnitude of the gradient in a given number of directions is calculated. This method reduced data points. We performed transformation to dataset with transformers. We converted images to grayscale and apply scaling to calculate histogram of oriented gradients. Then we set to preprocess our RGB images to scaled histogram of oriented gradient features. We applied Stochastic Gradient Descent classifier, because it works reasonably well. To test the trained SGD classifier, we used test set. First, we transform test set using the same transformers. We made a predictions for our test set. Model accuracy was 79% which is better than Method 1. We set up a pipeline that preprocesses the data. The pipeline fit method takes input data and transforms it in steps by sequentially. The data is passed from output to input until it reaches the end. Then we applied GridSearchCV to improve accuracy. Our parameter grid consists of two dictionaries. In the first improved the histogram of oriented gradient transformer. In the second we test SVM. Lastly our model accuracy is increased to 82%.

Multilayer Perceptron Model:

* We applied Multilayer Perceptron model to our dataset, after we applied preprocess like reading images, converting them to array and reshaping images. In addition we used label binarizer for our labels and we used ImageDataGenerator. When we build 2 hidden layer we checked the validation loss and training loss per epoch. We applied GridSearchCV with different learning rates, activation functions and dropout rate. We created a confusion matrix to visualize results. It was clear that model is underfiting. To solve this problem we should add new photos to per category but we decided to try more complex model. We add more layers and L1 regularization parameter.Then we applied HOG transform to our dataset. After the grid search we apply best parameters:model activation-relu and optimizer-Adam. We plot the losses and accuracy changes per epoch. We checked the accuracy that is 70%. Then we plot the confusion matrix. Missclassification rate in test set is 30%. Still our model is not good enough even we modelled the HOG features.

Convolutional Neural Networks:

* We applied preprocesses like label binarizer to our labels and scaled the pixels. We split the dataset to 3 parts which are train,test and validation. Then we used data augmentation. We initialize the model then used gridsearch. We used the same gridsearch with MLP but this time we changed our model. We build simple CNN with two Convolution layers with 100 and 50 units, Flatten, two Dropout and output layer with 9 units. In the output layer we have fully connected layers which have Softmax as activation and Loss is categorical cross entropy We trained model and applied gridsearch with different batch size. We checked the validation accuracy and loss per epoch with plot. We want to change model the complexity of this model since we need more true classification results for object detection on video. Therefore we add more convolution layers. In addition we add batch normalization which improves performance and stability of artificial neural network. We also add MaxPooling layer which performs downsampling operation and reduces the number of parameters. After we trained the model and cross validation we found very high accuracy which is 95% on test set. Then we saved the model in HDF5 file format. We tried for single image case randomly. Then we draw confusion matrix plots. According to the results on test set Alarm-Clock predicted with 100% accuracy. Fork class is looks like also has very accurate predictions. Scissors has the maximum misclassification rate which is 6%.

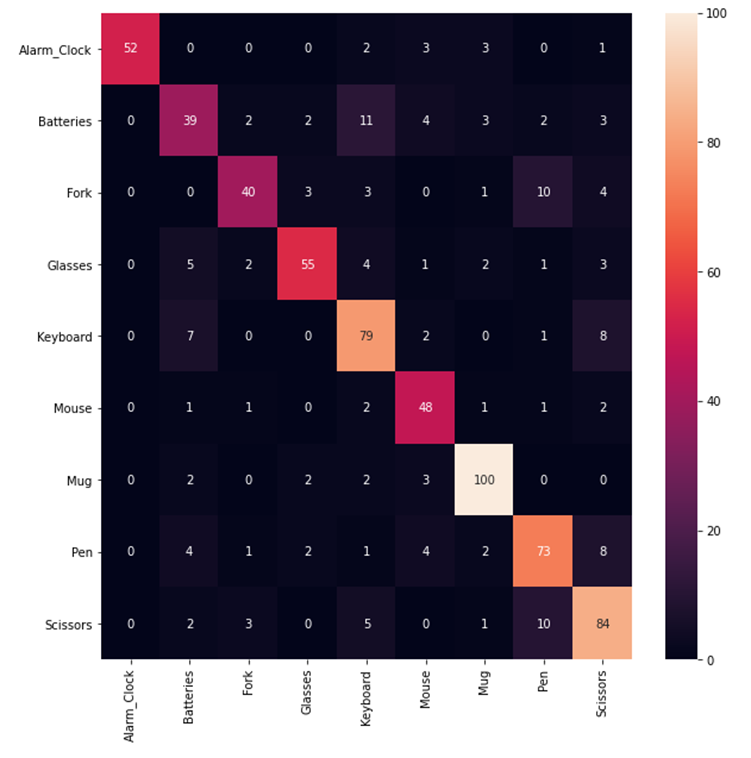


Live Image Classification Program:

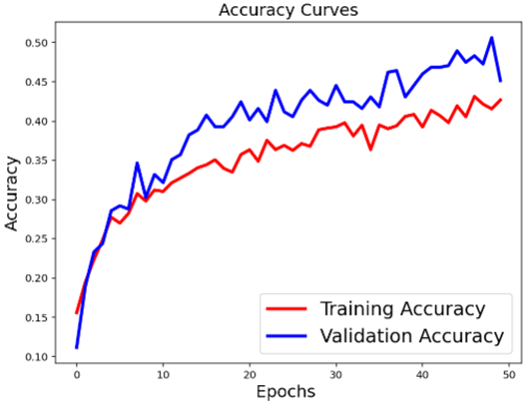
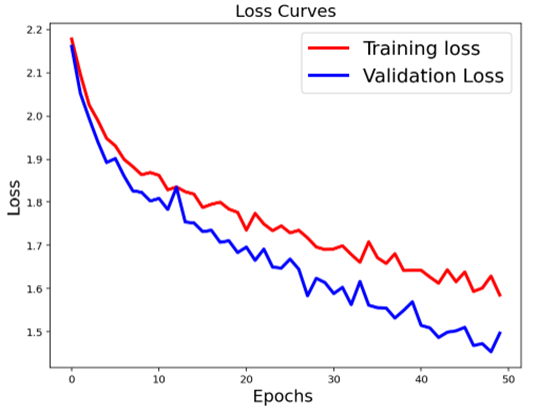
We created a real-time image classification algorithm. Firstly, to detect if there is an object on the live video stream, we are using a MobileNet-SSD detection network. After getting the result of a shape with confidence over 0.5, we are running the image data that is captured from the stream in our model, after our model returns a category number and a score, we are thresholding the score if it's over 0.3,after we are drawing a rectangle with a category color and adding the name of the category over it.

1. Experimental Results

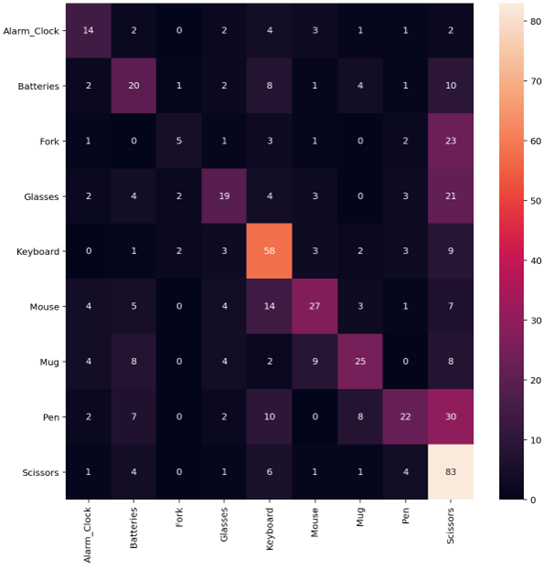
Object based SVM Model test results;

We create a confusing matrix with the confusion\_matrix function from sklearn.metrics. The confusion matrix for the SGD test is a 9x9 matrix. Correct predictions appear on the main diagonal, whereas all off-diagonal values correspond to incorrect classifications. According to the this graph we can say that all Alarm-Clock’s are predicted as Alarm-Clock. However, some Keyboard, Mouse, Mug and Scissors are also predicted as Alarm-Clock. Misclassification rate is 23%.

Simple Multilayer Perceptron Model training results;

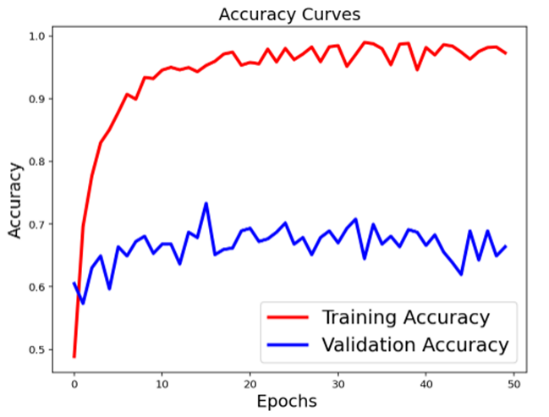
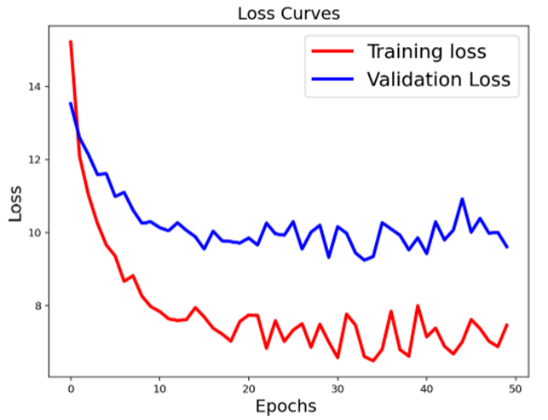


According to the results our validation set might have easier features.

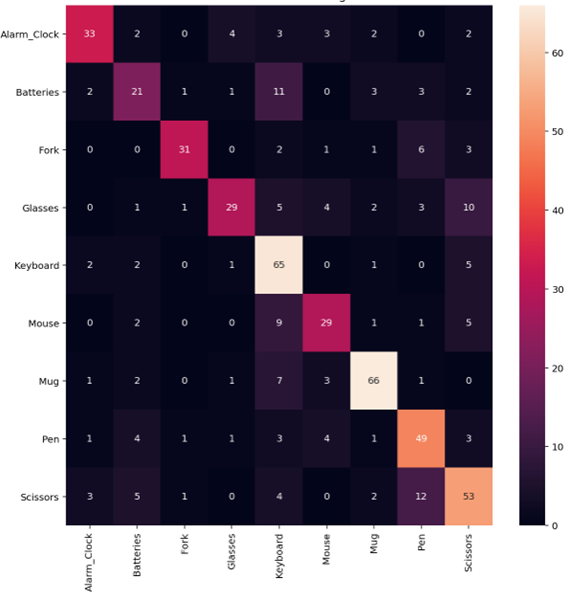


According to this table we can say that our model is not accurate enough. Scissors, Pen and Keyboard are not predicted very well. Misclassification rates of best predicted object is around 1% which is Fork. We concluded that we should to apply more advance techniques.

Multilayer Perceptron Model with Hog Transformed dataset training results;

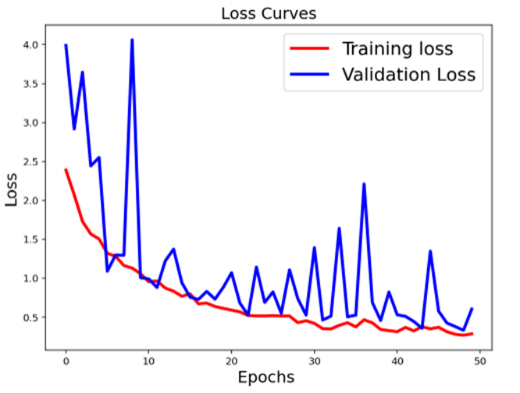


According to the results of this graph we can conclude that this model is underfit.

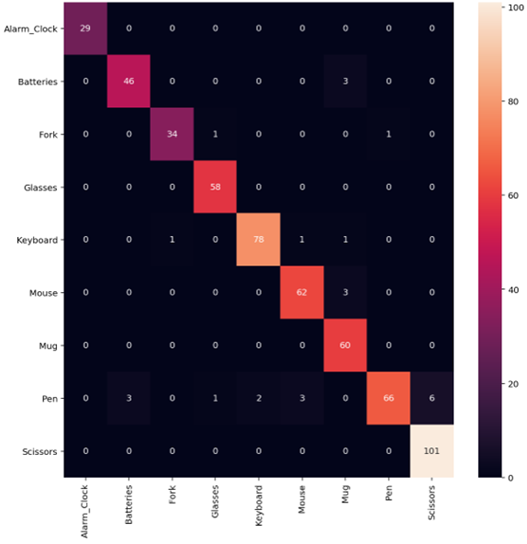


According to the results Fork is predicted more accurate however Scissors and Keyboard has the higher misclassification rates than others.

CNN Model results;



According to the results we might consider to change parameters for convergence, or randomly selected classed inbalanced validation set might cause this situation.



According to the results Alarm Clock predicted with 100% accuracy on the test dataset.Our model is now looking like well understood of almost every features.

1. Conclusions

We experimented with several different models and we understand that if we increase the number of images in every label that could increase our accuracy so that we could probably do this with less complex model.

In every model we used cross validation. Computer Vision approaches like histogram of oriented gradient, image segmentation, removing noise etc. helped for pre-processing part of dataset. Data augmentation helped also in this process. Because of time and memory complexity we preferred to use 100x100 as pixel size. However, we saw that increasing the pixel size could improve the model. We compared the classification performance with ANN and SVM since SVM gives good results. We get best predictions on test dataset with our last CNN model. Most of the time Fork has better accuracy than others with these models. We also used this models weights for real time detection of our objects with the help of Caffe. It has sometimes bad predictions, boundaries are not very perfect but still we are gladsome about it. For object localizing and bounding box could be optimized.

1. Appendix

|  |
| --- |
| **ImageNet Data Fetcher:** |
| import os  import wget  import urllib.request  #################################################################  #Please specify the WorldNet ID and the folder name that you want  worldnet\_id='n07930864'  desired\_foldername='cup'  #################################################################  base\_url = 'http://image-net.org/api/text/imagenet.synset.geturls?wnid='  complete\_url=base\_url+worldnet\_id  fetched\_file\_open = urllib.request.urlopen(complete\_url)  fetched\_file\_bytearray=fetched\_file\_open.read()  fetched\_file=fetched\_file\_bytearray.decode("utf8")  fetched\_file\_open.close()  found\_urls=[]  found\_urls=fetched\_file.splitlines()  try:      os.mkdir(desired\_foldername)      print("Folder Created:",desired\_foldername)  except:      print("There is a folder with the same name.")      exit()  print(str(len(found\_urls)),"images will be downloaded.")  filename\_counter=0  total\_count=0  for url in found\_urls:      try:          filename= os.getcwd()+"\\"+desired\_foldername+"\\"+str(filename\_counter)+".jpg"          total\_count=total\_count+1          print("("+str(total\_count)+"/"+str(len(found\_urls))+")")          local\_image\_filename = wget.download(url,filename)          filename\_counter=filename\_counter+1      except:          print("Bad Request")  print(str(filename\_counter),"of the",str(len(found\_urls)),"images downloaded succesfully.") |
| **Real-Time Object Detector and Classifier** |
| import numpy as np  import argparse  import imutils  import time  import cv2  import tensorflow as tf  import wget  import os  from imutils.video import VideoStream  from keras.preprocessing.image import img\_to\_array  from keras.models import load\_model  net = cv2.dnn.readNetFromCaffe("MobileNetSSD\_deploy.prototxt.txt", "MobileNetSSD\_deploy.caffemodel")  vs = VideoStream(src=0).start()  time.sleep(2.0)  model = tf.keras.models.load\_model("model\_dataset\_2\_marked\_100epoch\_changed\_model\_full\_patience\_20.h5")  classify=['Alarm\_Clock','Batteries','Fork','Glasses','Keyboard','Mouse','Mug','Pen','Scissors']  while True:      frame = vs.read()      frame = imutils.resize(frame, width=1200)      (h, w) = frame.shape[:2]      blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), scalefactor=0.007843, size=(300, 300), mean=127.5)      net.setInput(blob)      detections = net.forward()      for i in range(0, detections.shape[2]):          confidence = detections[0, 0, i, 2]          if confidence < 0.1:              continue          box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])          (startX, startY, endX, endY) = box.astype("int")          try:              image = cv2.resize(frame[startY:endY, startX:endX], (300, 300))              image = cv2.resize(image, (100, 100))              image = img\_to\_array(image)              image = np.array(image, dtype="float") / 255.0              image = np.expand\_dims(image, axis=0)              a = model.predict(image)              idx\_int = np.argmax(a)              proba = model.predict(image)[0]              idx = np.argmax(proba)              texts = "NONE"              treshold=0.3              if  proba[idx]>treshold and idx == 0:                  text = classify[0]                  y = startY - 10 if startY - 10 > 10 else startY + 10                  cv2.rectangle(frame, (startX, startY), (endX, endY), (0, 0, 255), 2)                  cv2.putText(frame, text, (startX, y),                  cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (0, 0, 255), 2)              elif  proba[idx]>treshold and idx == 1:                  text = classify[1]                  y = startY - 10 if startY - 10 > 10 else startY + 10                  cv2.rectangle(frame, (startX, startY), (endX, endY), (0, 255, 0), 2)                  cv2.putText(frame, text, (startX, y),                  cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (0, 255, 0), 2)              elif proba[idx]>treshold and idx == 2:                  text = classify[2]                  y = startY - 10 if startY - 10 > 10 else startY + 10                  cv2.rectangle(frame, (startX, startY), (endX, endY), (255, 0, 255), 2)                  cv2.putText(frame, text, (startX, y),                  cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (255, 0, 255), 2)              elif proba[idx]>treshold and idx == 3:                  text = classify[3]                  y = startY - 10 if startY - 10 > 10 else startY + 10                  cv2.rectangle(frame, (startX, startY), (endX, endY), (255, 0, 0), 2)                  cv2.putText(frame, text, (startX, y),                  cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (255, 0, 0), 2)                elif proba[idx]>treshold and idx == 4:                  text = classify[4]                  y = startY - 10 if startY - 10 > 10 else startY + 10                  cv2.rectangle(frame, (startX, startY), (endX, endY), (0, 255, 0), 2)                  cv2.putText(frame, text, (startX, y),                  cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (0, 255, 0), 2)              elif proba[idx]>treshold and idx == 5:                  text = classify[5]                  y = startY - 10 if startY - 10 > 10 else startY + 10                  cv2.rectangle(frame, (startX, startY), (endX, endY), (0, 255, 255), 2)                  cv2.putText(frame, text, (startX, y),                  cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (0, 255, 255), 2)              elif proba[idx]>treshold and idx == 6:                  text = classify[6]                  y = startY - 10 if startY - 10 > 10 else startY + 10                  cv2.rectangle(frame, (startX, startY), (endX, endY), (125, 255, 0), 2)                  cv2.putText(frame, text, (startX, y),                  cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (125, 255, 0), 2)              elif proba[idx]>treshold and idx == 7:                  text = classify[7]                  y = startY - 10 if startY - 10 > 10 else startY + 10                  cv2.rectangle(frame, (startX, startY), (endX, endY), (255, 125, 0), 2)                  cv2.putText(frame, text, (startX, y),                  cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (255, 125, 0), 2)              elif proba[idx]>treshold and idx == 8:                  text = classify[8]                  y = startY - 10 if startY - 10 > 10 else startY + 10                  cv2.rectangle(frame, (startX, startY), (endX, endY), (255, 0, 125), 2)                  cv2.putText(frame, text, (startX, y),                  cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (255, 0, 125), 2)              elif proba[idx]<=treshold:                  text = "UNKNOWN OBJECT"                  y = startY - 10 if startY - 10 > 10 else startY + 10                  cv2.rectangle(frame, (startX, startY), (endX, endY), (0, 0, 0), 2)                  cv2.putText(frame, text, (startX, y),                  cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (0, 0, 0), 2)          except:              pass      cv2.imshow("Frame", frame)      key = cv2.waitKey(1) & 0xFF      if key == ord("q"):          raise SystemExit          break  cv2.destroyAllWindows()  vs.stop() |

|  |
| --- |
| **Our CNN Model** |
| **Imports.** |
| import PIL  import matplotlib  matplotlib.use("Agg")  import matplotlib.pyplot as plt  import numpy as np  import argparse  import random  import pickle  import cv2  import os  import glob  import numpy as np  import pandas as pd  import xgboost as xgb  import lightgbm as lgb  import tensorflow as tf  import numpy as np  import seaborn as sns  from sklearn.metrics import confusion\_matrix  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import BatchNormalization  from tensorflow.keras.layers import Conv2D  from tensorflow.keras.layers import MaxPooling2D  from tensorflow.keras.layers import Activation  from tensorflow.keras.layers import Flatten  from tensorflow.keras.layers import Dropout  from tensorflow.keras.layers import Dense  from tensorflow.keras import backend as K  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.optimizers import Adam  from tensorflow.keras.preprocessing.image import img\_to\_array  from sklearn.preprocessing import LabelBinarizer  from sklearn.model\_selection import train\_test\_split  from imutils import paths  from PIL import Image  from sklearn.model\_selection import GridSearchCV, train\_test\_split  from sklearn.datasets import load\_boston, load\_breast\_cancer  from sklearn.ensemble import RandomForestClassifier  from keras.datasets import mnist  from keras.utils.np\_utils import to\_categorical  from keras.models import Sequential  from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Activation  from keras.optimizers import Adam  from keras.wrappers.scikit\_learn import KerasClassifier  from keras.models import Sequential  from keras.layers.normalization import BatchNormalization  from keras.layers.convolutional import Conv2D  from keras.layers.convolutional import MaxPooling2D  from keras.layers.core import Activation  from keras.layers.core import Flatten  from keras.layers.core import Dropout  from keras.layers.core import Dense  from keras import backend as K  from keras.models import load\_model  from IPython.display import clear\_output, display  from keras.utils.vis\_utils import plot\_model |
| **The Function that gets the images, and does the preprocessing.** |
| def get\_image\_list\_from\_folder(imageFolderPath, dim1, dim2):    IMAGE\_DIMS = (dim1, dim2, 3)    data = []    labels = []    counter=0    imagePaths=glob.glob(imageFolderPath+'\*\*/\*.jpg', recursive=True)    # loop over the input images    for imagePath in imagePaths:      # load the image, pre-process it, and store it in the data list      try:        counter=counter+1        clear\_output(wait=True)        print("Try:"+str(counter)+"/"+str(len(imagePaths)))        image = cv2.imread(imagePath)        image\_norm = cv2.resize(image, (dim2, dim1))        image\_norm = img\_to\_array(image\_norm)        #Preproccessing        temp\_img= cv2.resize(image, (dim2, dim1))        temp\_img= cv2.GaussianBlur(temp\_img, (5, 5), 0)        gray = cv2.cvtColor(temp\_img, cv2.COLOR\_RGB2GRAY)        ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)        kernel = np.ones((3, 3), np.uint8)        opening = cv2.morphologyEx(thresh, cv2.MORPH\_OPEN, kernel, iterations=2)        sure\_bg = cv2.dilate(opening, kernel, iterations=3)        dist\_transform = cv2.distanceTransform(opening, cv2.DIST\_L2, 5)        ret, sure\_fg = cv2.threshold(dist\_transform, 0.7 \* dist\_transform.max(), 255, 0)        sure\_fg = np.uint8(sure\_fg)        unknown = cv2.subtract(sure\_bg, sure\_fg)        ret, markers = cv2.connectedComponents(sure\_fg)        markers = markers + 1        markers[unknown == 255] = 0        markers = cv2.watershed(temp\_img, markers)        temp\_img[markers == -1] = [255, 0, 0]        data.append(temp\_img)        data.append(image\_norm)        label = imagePath.split(os.path.sep)[-2]        labels.append(label)        labels.append(label)      except:        print("A corrupted file.")    return data, labels |

|  |
| --- |
| **Our pipeline function to do the Grid Search.** |
| def algorithm\_pipeline(X\_train\_data, X\_test\_data, y\_train\_data, y\_test\_data,                         model, param\_grid, cv=10, scoring\_fit='neg\_mean\_squared\_error',                         do\_probabilities = False):      gs = GridSearchCV(          estimator=model,          param\_grid=param\_grid,          cv=cv,          n\_jobs=-1,          scoring=scoring\_fit,          verbose=2      )      fitted\_model = gs.fit(X\_train\_data, y\_train\_data)      if do\_probabilities:        pred = fitted\_model.predict\_proba(X\_test\_data)      else:        pred = fitted\_model.predict(X\_test\_data)      return fitted\_model, pred |
| **Our Model Function.** |
| def build\_CNN(width, height, depth, classes, activation = 'relu', dropout\_rate = 0.2, learning\_rate=1e-3, epochs=100):    model = Sequential()    inputShape = (height, width, depth)    chanDim = -1    if K.image\_data\_format() == "channels\_first":      inputShape = (depth, height, width)      chanDim = 1    model.add(Conv2D(32, (3, 3), padding="same", input\_shape=inputShape))    model.add(Activation(activation))    model.add(BatchNormalization(axis=chanDim))    model.add(MaxPooling2D(pool\_size=(3, 3), strides=2))    model.add(Dropout(dropout\_rate))    model.add(Conv2D(64, (3, 3), padding="same"))    model.add(Activation(activation))    model.add(BatchNormalization(axis=chanDim))    model.add(Conv2D(64, (3, 3), padding="same"))    model.add(Activation(activation))    model.add(BatchNormalization(axis=chanDim))    model.add(MaxPooling2D(pool\_size=(2, 2), strides=2))    model.add(Dropout(dropout\_rate))    model.add(Conv2D(128, (3, 3), padding="same"))    model.add(Activation(activation))    model.add(BatchNormalization(axis=chanDim))    model.add(Conv2D(128, (3, 3), padding="same"))    model.add(Activation(activation))    model.add(BatchNormalization(axis=chanDim))    model.add(MaxPooling2D(pool\_size=(2, 2), strides=2))    model.add(Dropout(dropout\_rate))    model.add(Flatten())    model.add(Dense(1024))    model.add(Activation(activation))    model.add(BatchNormalization())    model.add(Dropout(dropout\_rate))    model.add(Dense(classes))    model.add(Activation("softmax"))    model.compile(loss="categorical\_crossentropy", optimizer='Adam', metrics=["accuracy"])    return model |

|  |
| --- |
| **Getting the dataset and the labels and the preprocessed images.** |
| base\_folder\_address=os.getcwd()+"/drive/My Drive/project/Project-Files/Dataset\_2/"  #Specify the desired dimensions for the images  IMAGE\_DIMS = (100, 100, 3)  #Specify the folder that contains your images  data, labels=get\_image\_list\_from\_folder(base\_folder\_address, IMAGE\_DIMS[1], IMAGE\_DIMS[0])  print("Total image count with the processed ones:",len(data)) |
| **Scaling the total dataset.** |
| #Scaling between 0 and 1  data = np.array(data, dtype="float") / 255.0  labels = np.array(labels)  print("Your data matrix has the size of {:.2f} MB without the augmented images.".format(  data.nbytes / (1024 \* 1000.0))) |
| **Binarizing the labels.** |
| lb = LabelBinarizer()  labels = lb.fit\_transform(labels)  print("You have",len(lb.classes\_),"classes at your data.") |
| **Training, Validation and Test Split.** |
| (trainX, testX, trainY, testY) = train\_test\_split(data, labels, test\_size=0.15, random\_state=42)  (trainX, validationX, trainY, validationY) = train\_test\_split(trainX, trainY, test\_size=0.15, random\_state=42)  print("Your Train Data size is:",len(trainX))  print("Your Validation Data size is:",len(validationX))  print("Your Test Data size is:",len(testX)) |
| **Data Generator arrangement for Data Augmentation.** |
| aug = ImageDataGenerator(rotation\_range=25, width\_shift\_range=0.1, height\_shift\_range=0.1, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True, fill\_mode="nearest") |
| **Grid Search.** |
| param\_grid = {                'width': [IMAGE\_DIMS[1]],                'height': [IMAGE\_DIMS[0]],                'depth': [IMAGE\_DIMS[2]],                'classes': [len(lb.classes\_)],                'epochs':[10],                'dropout\_rate' :[0.2],                'activation' :['relu']               }  model = KerasClassifier(build\_fn = build\_CNN, verbose=0)  model, pred = algorithm\_pipeline(trainX, testX, trainY, testY, model, param\_grid, cv=3, scoring\_fit='neg\_log\_loss')  print("Best Score:",model.best\_score\_)  print("Best Parameters:",model.best\_params\_) |

|  |
| --- |
| **Normal Running Process of the Model.** |
| EPOCHS = 50  INIT\_LR = 1e-3  BS = 32  #Compiling the model  model = build\_CNN(width=IMAGE\_DIMS[1], height=IMAGE\_DIMS[0],depth=IMAGE\_DIMS[2], classes=len(lb.classes\_))  #Training the Model  print("Training the model:")  H = model.fit( x=aug.flow(trainX, trainY, batch\_size=BS), validation\_data=(validationX, validationY), steps\_per\_epoch=len(trainX) // BS, epochs=EPOCHS, verbose=1) |
| **Getting the Loss and Accuracy Plots.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  history=H  #Plot the Loss Curves  plt.figure(figsize=[8,6])  plt.plot(history.history['loss'],'r',linewidth=3.0)  plt.plot(history.history['val\_loss'],'b',linewidth=3.0)  plt.legend(['Training loss', 'Validation Loss'],fontsize=18)  plt.xlabel('Epochs ',fontsize=16)  plt.ylabel('Loss',fontsize=16)  plt.title('Loss Curves',fontsize=16)  #Plot the Accuracy Curves  plt.figure(figsize=[8,6])  plt.plot(history.history['accuracy'],'r',linewidth=3.0)  plt.plot(history.history['val\_accuracy'],'b',linewidth=3.0)  plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)  plt.xlabel('Epochs ',fontsize=16)  plt.ylabel('Accuracy',fontsize=16)  plt.title('Accuracy Curves',fontsize=16) |
| **Evaluation with the Test Data.** |
| results = model.evaluate(testX, testY, batch\_size=32)  print("Loss",results[0])  print("Accuracy:",results[1]) |
| **Saving a Model.** |
| model\_tmp=model  try:    model\_tmp.save(os.getcwd()+"/drive/My Drive/project/Project-Files/"+"model\_dataset\_2\_marked\_100epoch\_changed\_model\_full\_new.h5")    print("Saved model to disk")  except:    print("Problem while saving the model.") |
| **Loading a Model.** |
| model = tf.keras.models.load\_model(os.getcwd()+"/drive/My Drive/project/Project-Files/"+"model\_dataset\_2\_marked\_100epoch\_changed\_model\_full\_patience\_20.h5") |
| **Visualizing the Network.** |
| plot\_model(model, to\_file='model\_plot.png', rankdir="TB",show\_shapes=True, show\_layer\_names=True) |

|  |
| --- |
| **Training Confusion Matrix & Training Normalised Confusion Matrix.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  predY=model.predict(trainX)  real\_testY=lb.inverse\_transform(trainY)  real\_predY=lb.inverse\_transform(predY)  cm=confusion\_matrix(real\_testY,real\_predY)  plt.figure(figsize=(10,10))  plt.title("Confusion Matrix for Training Data")  sns.heatmap(cm, annot=True,fmt='d', xticklabels=lb.classes\_, yticklabels=lb.classes\_)  plt.figure()  plt.figure(figsize=(10,10))  plt.title("Normalized Confusion Matrix for Training Data")  sns.heatmap(cm/np.sum(cm), annot=True, xticklabels=lb.classes\_, yticklabels=lb.classes\_, fmt='.2%', cmap='Blues') |
| **Validation Confusion Matrix & Validation Normalised Confusion Matrix.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  predY=model.predict(validationX)  real\_testY=lb.inverse\_transform(validationY)  real\_predY=lb.inverse\_transform(predY)  cm=confusion\_matrix(real\_testY,real\_predY)  plt.figure(figsize=(10,10))  plt.title("Confusion Matrix for Validation Data")  sns.heatmap(cm, annot=True,fmt='d', xticklabels=lb.classes\_, yticklabels=lb.classes\_)  plt.figure()  plt.figure(figsize=(10,10))  plt.title("Normalized Confusion Matrix for Validation Data")  sns.heatmap(cm/np.sum(cm), annot=True, xticklabels=lb.classes\_, yticklabels=lb.classes\_, fmt='.2%', cmap='Blues') |
| **Test Confusion Matrix & Test Normalised Confusion Matrix.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  predY=model.predict(testX)  real\_testY=lb.inverse\_transform(testY)  real\_predY=lb.inverse\_transform(predY)  cm=confusion\_matrix(real\_testY,real\_predY)  plt.figure(figsize=(10,10))  plt.title("Confusion Matrix for Test Data")  sns.heatmap(cm, annot=True,fmt='d', xticklabels=lb.classes\_, yticklabels=lb.classes\_)  plt.figure()  plt.figure(figsize=(10,10))  plt.title("Normalized Confusion Matrix for Test Data")  sns.heatmap(cm/np.sum(cm), annot=True, xticklabels=lb.classes\_, yticklabels=lb.classes\_, fmt='.2%', cmap='Blues') |
| **Getting the Summary of the Model.** |
| model.summary() |

|  |
| --- |
| **MLP Model** |
| **Imports.** |
| import PIL  import matplotlib  matplotlib.use("Agg")  import matplotlib.pyplot as plt  import numpy as np  import argparse  import random  import pickle  import cv2  import os  import glob  import skimage  import numpy as np  import pandas as pd  import xgboost as xgb  import lightgbm as lgb  import tensorflow as tf  import numpy as np  import seaborn as sns  from sklearn.metrics import confusion\_matrix  from PIL import Image  from keras.datasets import mnist  from keras.utils.np\_utils import to\_categorical  from keras.models import Sequential  from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Activation  from keras.optimizers import Adam  from keras.wrappers.scikit\_learn import KerasClassifier  from keras.models import Sequential  from keras.layers.normalization import BatchNormalization  from keras.layers.convolutional import Conv2D  from keras.layers.convolutional import MaxPooling2D  from keras.layers.core import Activation  from keras.layers.core import Flatten  from keras.layers.core import Dropout  from keras.layers.core import Dense  from keras import backend as K  from keras.models import load\_model  from imutils import paths  from IPython.display import clear\_output, display  from keras.utils.vis\_utils import plot\_model  from sklearn.model\_selection import GridSearchCV, train\_test\_split  from sklearn.datasets import load\_boston, load\_breast\_cancer  from sklearn.ensemble import RandomForestClassifier  from skimage.feature import hog  from skimage.io import imread  from skimage.transform import rescale  from sklearn.base import BaseEstimator, TransformerMixin  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.optimizers import Adam  from tensorflow.keras.preprocessing.image import img\_to\_array  from sklearn.preprocessing import LabelBinarizer  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from tensorflow.keras import regularizers  from keras.regularizers import l1  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import BatchNormalization  from tensorflow.keras.layers import Conv2D  from tensorflow.keras.layers import MaxPooling2D  from tensorflow.keras.layers import Activation  from tensorflow.keras.layers import Flatten  from tensorflow.keras.layers import Dropout  from tensorflow.keras.layers import Dense  from tensorflow.keras import backend as K |

|  |
| --- |
| **The Function that gets the images, and does the preprocessing.** |
| def get\_image\_list\_from\_folder(imageFolderPath, dim1, dim2):    IMAGE\_DIMS = (dim1, dim2, 3)    data = []    labels = []    counter=0    imagePaths=glob.glob(imageFolderPath+'\*\*/\*.jpg', recursive=True)    # loop over the input images    for imagePath in imagePaths:      # load the image, pre-process it, and store it in the data list      try:        counter=counter+1        clear\_output(wait=True)        print("Try:"+str(counter)+"/"+str(len(imagePaths)))        image = cv2.imread(imagePath)        image\_norm = cv2.resize(image, (dim2, dim1))        image\_norm = img\_to\_array(image\_norm)        #Preproccessing        temp\_img= cv2.resize(image, (dim2, dim1))        temp\_img= cv2.GaussianBlur(temp\_img, (5, 5), 0)        gray = cv2.cvtColor(temp\_img, cv2.COLOR\_RGB2GRAY)        ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)        kernel = np.ones((3, 3), np.uint8)        opening = cv2.morphologyEx(thresh, cv2.MORPH\_OPEN, kernel, iterations=2)        sure\_bg = cv2.dilate(opening, kernel, iterations=3)        dist\_transform = cv2.distanceTransform(opening, cv2.DIST\_L2, 5)        ret, sure\_fg = cv2.threshold(dist\_transform, 0.7 \* dist\_transform.max(), 255, 0)        sure\_fg = np.uint8(sure\_fg)        unknown = cv2.subtract(sure\_bg, sure\_fg)        ret, markers = cv2.connectedComponents(sure\_fg)        markers = markers + 1        markers[unknown == 255] = 0        markers = cv2.watershed(temp\_img, markers)        temp\_img[markers == -1] = [255, 0, 0]        data.append(temp\_img)        data.append(image\_norm)        label = imagePath.split(os.path.sep)[-2]        labels.append(label)        labels.append(label)      except:        print("A corrupted file.")    return data, labels |

|  |
| --- |
| **Color to Gray Transformer Function.** |
| class RGB2GrayTransformer(BaseEstimator, TransformerMixin):      def \_\_init\_\_(self):          pass      def fit(self, X, y=None):          return self      def transform(self, X, y=None):          return np.array([skimage.color.rgb2gray(img) for img in X]) |
| **Hog Transform Function.** |
| class HogTransformer(BaseEstimator, TransformerMixin):      def \_\_init\_\_(self, y=None, orientations=9,                   pixels\_per\_cell=(8, 8),                   cells\_per\_block=(3, 3), block\_norm='L2-Hys'):          self.y = y          self.orientations = orientations          self.pixels\_per\_cell = pixels\_per\_cell          self.cells\_per\_block = cells\_per\_block          self.block\_norm = block\_norm      def fit(self, X, y=None):          return self      def transform(self, X, y=None):          def local\_hog(X):              return hog(X,                         orientations=self.orientations,                         pixels\_per\_cell=self.pixels\_per\_cell,                         cells\_per\_block=self.cells\_per\_block,                         block\_norm=self.block\_norm)          try:              return np.array([local\_hog(img) for img in X])          except:              return np.array([local\_hog(img) for img in X]) |
| **Pipeline function to do Grid Search.** |
| def algorithm\_pipeline(X\_train\_data, X\_test\_data, y\_train\_data, y\_test\_data,                         model, param\_grid, cv=10, scoring\_fit='neg\_mean\_squared\_error',                         do\_probabilities = False):      gs = GridSearchCV(          estimator=model,          param\_grid=param\_grid,          cv=cv,          n\_jobs=-1,          scoring=scoring\_fit,          verbose=2      )      fitted\_model = gs.fit(X\_train\_data, y\_train\_data)      if do\_probabilities:        pred = fitted\_model.predict\_proba(X\_test\_data)      else:        pred = fitted\_model.predict(X\_test\_data)      return fitted\_model, pred |

|  |
| --- |
| **Our MLP Model Function.** |
| def build\_MLP(classes, activation = 'relu', dropout\_rate = 0.2, learning\_rate=1e-3, epochs=100):    model = Sequential()    model.add(Dense(512, activation='linear', activity\_regularizer=l1(0.001)))    model.add(Activation('relu'))    model.add(Dense(512))    model.add(Activation(activation))    model.add(Dense(512))    model.add(Activation('linear'))    model.add(Dense(512))    model.add(Activation(activation))    model.add(Dropout(dropout\_rate))    model.add(Dense(classes))    model.add(Activation("softmax"))    model.compile(loss="categorical\_crossentropy", optimizer='Adam', metrics=["accuracy"])    return model |
| **Getting the dataset and the labels and the preprocessed images.** |
| base\_folder\_address=os.getcwd()+"/drive/My Drive/project/Project-Files/Dataset\_2/"  #Specify the desired dimensions for the images  IMAGE\_DIMS = (100, 100, 3)  #Specify the folder that contains your images  data, labels=get\_image\_list\_from\_folder(base\_folder\_address, IMAGE\_DIMS[1], IMAGE\_DIMS[0])  print("Total image count with the processed ones:",len(data)) |
| **Scaling the total dataset.** |
| #Scaling between 0 and 1  data = np.array(data, dtype="float") / 255.0  labels = np.array(labels)  print("Your data matrix has the size of {:.2f} MB without the augmented images.".format(  data.nbytes / (1024 \* 1000.0))) |
| **Binarizing the labels.** |
| lb = LabelBinarizer()  labels = lb.fit\_transform(labels)  print("You have",len(lb.classes\_),"classes at your data.") |
| **Training, Validation and Test Split.** |
| (trainX, testX, trainY, testY) = train\_test\_split(data, labels, test\_size=0.15, random\_state=42)  (trainX, validationX, trainY, validationY) = train\_test\_split(trainX, trainY, test\_size=0.15, random\_state=42)  print("Your Train Data size is:",len(trainX))  print("Your Validation Data size is:",len(validationX))  print("Your Test Data size is:",len(testX)) |
| **Data Generator arrangement for Data Augmentation.** |
| aug = ImageDataGenerator(rotation\_range=25, width\_shift\_range=0.1, height\_shift\_range=0.1, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True, fill\_mode="nearest") |

|  |
| --- |
| **Grid Search.** |
| param\_grid = {                'width': [IMAGE\_DIMS[1]],                'height': [IMAGE\_DIMS[0]],                'depth': [IMAGE\_DIMS[2]],                'classes': [len(lb.classes\_)],                'epochs':[10],                'dropout\_rate' :[0.2],                'activation' :['relu']               }  model = KerasClassifier(build\_fn = build\_MLP, verbose=0)  model, pred = algorithm\_pipeline(trainX, testX, trainY, testY, model, param\_grid, cv=3, scoring\_fit='neg\_log\_loss')  print("Best Score:",model.best\_score\_)  print("Best Parameters:",model.best\_params\_) |
| **Normal Running Process of the Model.** |
| EPOCHS = 50  INIT\_LR = 1e-3  BS = 32  #Compiling the model  model = build\_MLP(width=IMAGE\_DIMS[1], height=IMAGE\_DIMS[0],depth=IMAGE\_DIMS[2], classes=len(lb.classes\_))  #Training the Model  print("Training the model:")  H = model.fit( x=aug.flow(trainX, trainY, batch\_size=BS), validation\_data=(validationX, validationY), steps\_per\_epoch=len(trainX) // BS, epochs=EPOCHS, verbose=1) |
| **Getting the Loss and Accuracy Plots.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  history=H  #Plot the Loss Curves  plt.figure(figsize=[8,6])  plt.plot(history.history['loss'],'r',linewidth=3.0)  plt.plot(history.history['val\_loss'],'b',linewidth=3.0)  plt.legend(['Training loss', 'Validation Loss'],fontsize=18)  plt.xlabel('Epochs ',fontsize=16)  plt.ylabel('Loss',fontsize=16)  plt.title('Loss Curves',fontsize=16)  #Plot the Accuracy Curves  plt.figure(figsize=[8,6])  plt.plot(history.history['accuracy'],'r',linewidth=3.0)  plt.plot(history.history['val\_accuracy'],'b',linewidth=3.0)  plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)  plt.xlabel('Epochs ',fontsize=16)  plt.ylabel('Accuracy',fontsize=16)  plt.title('Accuracy Curves',fontsize=16) |
| **Evaluation with the Test Data.** |
| results = model.evaluate(testX, testY, batch\_size=32)  print("Loss",results[0])  print("Accuracy:",results[1]) |

|  |
| --- |
| **Saving a Model.** |
| model\_tmp=model  try:    model\_tmp.save(os.getcwd()+"/drive/My Drive/project/Project-Files/"+"model\_dataset\_2\_marked\_100epoch\_changed\_model\_full\_new.h5")    print("Saved model to disk")  except:    print("Problem while saving the model.") |
| **Loading a Model.** |
| model = tf.keras.models.load\_model(os.getcwd()+"/drive/My Drive/project/Project-Files/"+"model\_dataset\_2\_marked\_100epoch\_changed\_model\_full\_patience\_20.h5") |
| **Visualizing the Network.** |
| plot\_model(model, to\_file='model\_plot.png', rankdir="TB",show\_shapes=True, show\_layer\_names=True) |
| **Training Confusion Matrix & Training Normalised Confusion Matrix.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  classify=['Alarm\_Clock','Batteries','Fork','Glasses','Keyboard','Mouse','Mug','Pen','Scissors']  predY=np.argmax(model.predict(trainX), axis=1)  real\_testY=np.argmax(trainY, axis=1)  real\_predY=lb.classes\_[predY]  cm=confusion\_matrix(real\_testY,real\_predY)  plt.figure(figsize=(10,10))  plt.title("Confusion Matrix for Training Data")  sns.heatmap(cm, annot=True,fmt='d', xticklabels=classify, yticklabels=classify)  plt.figure()  plt.figure(figsize=(10,10))  plt.title("Normalized Confusion Matrix for Training Data")  sns.heatmap(cm/np.sum(cm), annot=True, xticklabels=classify, yticklabels=classify, fmt='.2%', cmap='Blues') |
| **Validation Confusion Matrix & Validation Normalised Confusion Matrix.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  classify=['Alarm\_Clock','Batteries','Fork','Glasses','Keyboard','Mouse','Mug','Pen','Scissors']  predY=np.argmax(model.predict(validationX), axis=1)  real\_testY=np.argmax(validationY, axis=1)  real\_predY=lb.classes\_[predY]  cm=confusion\_matrix(real\_testY,real\_predY)  plt.figure(figsize=(10,10))  plt.title("Confusion Matrix for Validation Data")  sns.heatmap(cm, annot=True,fmt='d', xticklabels=classify, yticklabels=classify)  plt.figure()  plt.figure(figsize=(10,10))  plt.title("Normalized Confusion Matrix for Validation Data")  sns.heatmap(cm/np.sum(cm), annot=True, xticklabels=classify, yticklabels=classify, fmt='.2%', cmap='Blues') |

|  |
| --- |
| **Test Confusion Matrix & Test Normalised Confusion Matrix.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  classify=['Alarm\_Clock','Batteries','Fork','Glasses','Keyboard','Mouse','Mug','Pen','Scissors']  predY=np.argmax(model.predict(testX), axis=1)  real\_testY=np.argmax(testY, axis=1)  real\_predY=lb.classes\_[predY]  cm=confusion\_matrix(real\_testY,real\_predY)  plt.figure(figsize=(10,10))  plt.title("Confusion Matrix for Test Data")  sns.heatmap(cm, annot=True,fmt='d', xticklabels=classify, yticklabels=classify)  plt.figure()  plt.figure(figsize=(10,10))  plt.title("Normalized Confusion Matrix for Test Data")  sns.heatmap(cm/np.sum(cm), annot=True, xticklabels=classify, yticklabels=classify, fmt='.2%', cmap='Blues') |
| **Getting the Summary of the Model.** |
| model.summary() |

|  |
| --- |
| **MLP Model with Hog Transformation** |
| **Imports.** |
| import PIL  import matplotlib  matplotlib.use("Agg")  import matplotlib.pyplot as plt  import numpy as np  import argparse  import random  import pickle  import cv2  import os  import glob  import skimage  import numpy as np  import pandas as pd  import xgboost as xgb  import lightgbm as lgb  import tensorflow as tf  import numpy as np  import seaborn as sns  from sklearn.metrics import confusion\_matrix  from PIL import Image  from keras.datasets import mnist  from keras.utils.np\_utils import to\_categorical  from keras.models import Sequential  from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Activation  from keras.optimizers import Adam  from keras.wrappers.scikit\_learn import KerasClassifier  from keras.models import Sequential  from keras.layers.normalization import BatchNormalization  from keras.layers.convolutional import Conv2D  from keras.layers.convolutional import MaxPooling2D  from keras.layers.core import Activation  from keras.layers.core import Flatten  from keras.layers.core import Dropout  from keras.layers.core import Dense  from keras import backend as K  from keras.models import load\_model  from imutils import paths  from IPython.display import clear\_output, display  from keras.utils.vis\_utils import plot\_model  from sklearn.model\_selection import GridSearchCV, train\_test\_split  from sklearn.datasets import load\_boston, load\_breast\_cancer  from sklearn.ensemble import RandomForestClassifier  from skimage.feature import hog  from skimage.io import imread  from skimage.transform import rescale  from sklearn.base import BaseEstimator, TransformerMixin  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.optimizers import Adam  from tensorflow.keras.preprocessing.image import img\_to\_array  from sklearn.preprocessing import LabelBinarizer  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from tensorflow.keras import regularizers  from keras.regularizers import l1  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import BatchNormalization  from tensorflow.keras.layers import Conv2D  from tensorflow.keras.layers import MaxPooling2D  from tensorflow.keras.layers import Activation  from tensorflow.keras.layers import Flatten  from tensorflow.keras.layers import Dropout  from tensorflow.keras.layers import Dense  from tensorflow.keras import backend as K |
| **The Function that gets the images, and does the preprocessing.** |
| def get\_image\_list\_from\_folder(imageFolderPath, dim1, dim2):    IMAGE\_DIMS = (dim1, dim2, 3)    data = []    labels = []    counter=0    imagePaths=glob.glob(imageFolderPath+'\*\*/\*.jpg', recursive=True)    # loop over the input images    for imagePath in imagePaths:      # load the image, pre-process it, and store it in the data list      try:        counter=counter+1        clear\_output(wait=True)        print("Try:"+str(counter)+"/"+str(len(imagePaths)))        image = cv2.imread(imagePath)        image\_norm = cv2.resize(image, (dim2, dim1))        image\_norm = img\_to\_array(image\_norm)        #Preproccessing        temp\_img= cv2.resize(image, (dim2, dim1))        temp\_img= cv2.GaussianBlur(temp\_img, (5, 5), 0)        gray = cv2.cvtColor(temp\_img, cv2.COLOR\_RGB2GRAY)        ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)        kernel = np.ones((3, 3), np.uint8)        opening = cv2.morphologyEx(thresh, cv2.MORPH\_OPEN, kernel, iterations=2)        sure\_bg = cv2.dilate(opening, kernel, iterations=3)        dist\_transform = cv2.distanceTransform(opening, cv2.DIST\_L2, 5)        ret, sure\_fg = cv2.threshold(dist\_transform, 0.7 \* dist\_transform.max(), 255, 0)        sure\_fg = np.uint8(sure\_fg)        unknown = cv2.subtract(sure\_bg, sure\_fg)        ret, markers = cv2.connectedComponents(sure\_fg)        markers = markers + 1        markers[unknown == 255] = 0        markers = cv2.watershed(temp\_img, markers)        temp\_img[markers == -1] = [255, 0, 0]        data.append(temp\_img)        data.append(image\_norm)        label = imagePath.split(os.path.sep)[-2]        labels.append(label)        labels.append(label)      except:        print("A corrupted file.")    return data, labels |
| **Color to Gray Transformer Function.** |
| class RGB2GrayTransformer(BaseEstimator, TransformerMixin):      def \_\_init\_\_(self):          pass      def fit(self, X, y=None):          return self      def transform(self, X, y=None):          return np.array([skimage.color.rgb2gray(img) for img in X]) |

|  |
| --- |
| **Hog Transform Function.** |
| class HogTransformer(BaseEstimator, TransformerMixin):      def \_\_init\_\_(self, y=None, orientations=9,                   pixels\_per\_cell=(8, 8),                   cells\_per\_block=(3, 3), block\_norm='L2-Hys'):          self.y = y          self.orientations = orientations          self.pixels\_per\_cell = pixels\_per\_cell          self.cells\_per\_block = cells\_per\_block          self.block\_norm = block\_norm      def fit(self, X, y=None):          return self      def transform(self, X, y=None):          def local\_hog(X):              return hog(X,                         orientations=self.orientations,                         pixels\_per\_cell=self.pixels\_per\_cell,                         cells\_per\_block=self.cells\_per\_block,                         block\_norm=self.block\_norm)          try:              return np.array([local\_hog(img) for img in X])          except:              return np.array([local\_hog(img) for img in X]) |
| **Pipeline function to do Grid Search.** |
| def algorithm\_pipeline(X\_train\_data, X\_test\_data, y\_train\_data, y\_test\_data,                         model, param\_grid, cv=10, scoring\_fit='neg\_mean\_squared\_error',                         do\_probabilities = False):      gs = GridSearchCV(          estimator=model,          param\_grid=param\_grid,          cv=cv,          n\_jobs=-1,          scoring=scoring\_fit,          verbose=2      )      fitted\_model = gs.fit(X\_train\_data, y\_train\_data)      if do\_probabilities:        pred = fitted\_model.predict\_proba(X\_test\_data)      else:        pred = fitted\_model.predict(X\_test\_data)      return fitted\_model, pred |

|  |
| --- |
| **Our MLP Model Function.** |
| def build\_MLP(classes, activation = 'relu', dropout\_rate = 0.2, learning\_rate=1e-3, epochs=100):    model = Sequential()    model.add(Dense(512, activation='linear', activity\_regularizer=l1(0.001)))    model.add(Activation('relu'))    model.add(Dense(512))    model.add(Activation(activation))    model.add(Dense(512))    model.add(Activation('linear'))    model.add(Dense(512))    model.add(Activation(activation))    model.add(Dropout(dropout\_rate))    model.add(Dense(classes))    model.add(Activation("softmax"))    model.compile(loss="categorical\_crossentropy", optimizer='Adam', metrics=["accuracy"])    return model |
| **Getting the dataset and the labels and the preprocessed images.** |
| base\_folder\_address=os.getcwd()+"/drive/My Drive/project/Project-Files/Dataset\_2/"  #Specify the desired dimensions for the images  IMAGE\_DIMS = (100, 100, 3)  #Specify the folder that contains your images  data, labels=get\_image\_list\_from\_folder(base\_folder\_address, IMAGE\_DIMS[1], IMAGE\_DIMS[0])  print("Total image count with the processed ones:",len(data)) |
| **Applying the Hog Transformation and Scaling the Dataset.** |
| data = np.array(data, dtype="float")  grayify = RGB2GrayTransformer()  hogify = HogTransformer(      pixels\_per\_cell=(8, 8),      cells\_per\_block=(2,2),      orientations=9,      block\_norm='L2-Hys'  )  scalify = StandardScaler()    # call fit\_transform on each transform converting X\_train step by step  data = grayify.fit\_transform(data)  data = hogify.fit\_transform(data)  data = scalify.fit\_transform(data)  print("Your data matrix has the size of {:.2f} MB without the augmented images.".format(  data.nbytes / (1024 \* 1000.0))) |
| **Binarizing the labels.** |
| lb = LabelBinarizer()  labels = lb.fit\_transform(labels)  print("You have",len(lb.classes\_),"classes at your data.") |

|  |
| --- |
| **Training, Validation and Test Split.** |
| (trainX, testX, trainY, testY) = train\_test\_split(data, labels, test\_size=0.15, random\_state=42)  (trainX, validationX, trainY, validationY) = train\_test\_split(trainX, trainY, test\_size=0.15, random\_state=42)  print("Your Train Data size is:",len(trainX))  print("Your Validation Data size is:",len(validationX))  print("Your Test Data size is:",len(testX)) |
| **Data Generator arrangement for Data Augmentation.** |
| aug = ImageDataGenerator(rotation\_range=25, width\_shift\_range=0.1, height\_shift\_range=0.1, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True, fill\_mode="nearest") |
| **Grid Search.** |
| param\_grid = {                'width': [IMAGE\_DIMS[1]],                'height': [IMAGE\_DIMS[0]],                'depth': [IMAGE\_DIMS[2]],                'classes': [len(lb.classes\_)],                'epochs':[10],                'dropout\_rate' :[0.2],                'activation' :['relu']               }  model = KerasClassifier(build\_fn = build\_MLP, verbose=0)  model, pred = algorithm\_pipeline(trainX, testX, trainY, testY, model, param\_grid, cv=3, scoring\_fit='neg\_log\_loss')  print("Best Score:",model.best\_score\_)  print("Best Parameters:",model.best\_params\_) |
| **Normal Running Process of the Model.** |
| EPOCHS = 50  INIT\_LR = 1e-6  BS = 32  #Compiling the model  model = build\_MLP(classes=len(lb.classes\_))  #Training the Model  print("Training the model:")  H = model.fit( x=trainX, y=trainY, validation\_data=(validationX, validationY), epochs=EPOCHS, verbose=1) |

|  |
| --- |
| **Getting the Loss and Accuracy Plots.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  history=H  #Plot the Loss Curves  plt.figure(figsize=[8,6])  plt.plot(history.history['loss'],'r',linewidth=3.0)  plt.plot(history.history['val\_loss'],'b',linewidth=3.0)  plt.legend(['Training loss', 'Validation Loss'],fontsize=18)  plt.xlabel('Epochs ',fontsize=16)  plt.ylabel('Loss',fontsize=16)  plt.title('Loss Curves',fontsize=16)  #Plot the Accuracy Curves  plt.figure(figsize=[8,6])  plt.plot(history.history['accuracy'],'r',linewidth=3.0)  plt.plot(history.history['val\_accuracy'],'b',linewidth=3.0)  plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)  plt.xlabel('Epochs ',fontsize=16)  plt.ylabel('Accuracy',fontsize=16)  plt.title('Accuracy Curves',fontsize=16) |
| **Evaluation with the Test Data.** |
| results = model.evaluate(testX, testY, batch\_size=32)  print("Loss",results[0])  print("Accuracy:",results[1]) |
| **Saving a Model.** |
| model\_tmp=model  try:    model\_tmp.save(os.getcwd()+"/drive/My Drive/project/Project-Files/"+"model\_dataset\_2\_marked\_100epoch\_changed\_model\_full\_new.h5")    print("Saved model to disk")  except:    print("Problem while saving the model.") |
| **Loading a Model.** |
| model = tf.keras.models.load\_model(os.getcwd()+"/drive/My Drive/project/Project-Files/"+"model\_dataset\_2\_marked\_100epoch\_changed\_model\_full\_patience\_20.h5") |
| **Visualizing the Network.** |
| plot\_model(model, to\_file='model\_plot.png', rankdir="TB",show\_shapes=True, show\_layer\_names=True) |

|  |
| --- |
| **Training Confusion Matrix.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  classify=['Alarm\_Clock','Batteries','Fork','Glasses','Keyboard','Mouse','Mug','Pen','Scissors']  predicted=model.predict(trainX)  pred\_y=lb.inverse\_transform(predicted)  train\_y=lb.inverse\_transform(trainY)  cm=confusion\_matrix(train\_y,pred\_y)  plt.figure(figsize=(10,10))  plt.title("Confusion Matrix for Training Data")  sns.heatmap(cm, annot=True,fmt='d', xticklabels=classify, yticklabels=classify)  plt.title("Confusion Matrix for Training Data")  sns.heatmap(cm, annot=True,fmt='d', xticklabels=classify, yticklabels=classify)  plt.figure()  plt.figure(figsize=(10,10))  plt.title("Normalized Confusion Matrix for Training Data")  sns.heatmap(cm/np.sum(cm), annot=True, xticklabels=classify, yticklabels=classify, fmt='.2%', cmap='Blues') |
| **Validation Confusion Matrix.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  predicted=model.predict(validationX)  pred\_y=lb.inverse\_transform(predicted)  validation\_y=lb.inverse\_transform(validationY)  classify=['Alarm\_Clock','Batteries','Fork','Glasses','Keyboard','Mouse','Mug','Pen','Scissors']  cm=confusion\_matrix(validation\_y,pred\_y)  plt.figure(figsize=(10,10))  plt.title("Confusion Matrix for Validation Data")  sns.heatmap(cm, annot=True,fmt='d', xticklabels=classify, yticklabels=classify) |
| **Test Confusion Matrix.** |
| %matplotlib inline  %config InlineBackend.figure\_format = 'retina'  predicted=model.predict(testX)  pred\_y=lb.inverse\_transform(predicted)  test\_y=lb.inverse\_transform(testY)  classify=['Alarm\_Clock','Batteries','Fork','Glasses','Keyboard','Mouse','Mug','Pen','Scissors']  cm=confusion\_matrix(test\_y,pred\_y)  plt.figure(figsize=(10,10))  plt.title("Confusion Matrix for Test Data")  sns.heatmap(cm, annot=True,fmt='d', xticklabels=classify, yticklabels=classify) |
| **Getting the Summary of the Model.** |
| model.summary() |

|  |
| --- |
| **SVC with Hog Transform** |
| **Import.** |
| import PIL  import matplotlib  matplotlib.use("Agg")  import matplotlib.pyplot as plt  import numpy as np  import argparse  import random  import pickle  import cv2  import os  import glob  import xgboost as xgb  import lightgbm as lgb  import tensorflow as tf  import numpy as np  import numpy as np  import pandas as pd  import skimage  import seaborn as sns  from PIL import Image  from tensorflow.keras.preprocessing.image import ImageDataGenerator  from tensorflow.keras.optimizers import Adam  from tensorflow.keras.preprocessing.image import img\_to\_array  from sklearn.preprocessing import LabelBinarizer  from sklearn.model\_selection import train\_test\_split  from imutils import paths  from sklearn.model\_selection import GridSearchCV, train\_test\_split  from sklearn.datasets import load\_boston, load\_breast\_cancer  from sklearn.ensemble import RandomForestClassifier  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import BatchNormalization  from tensorflow.keras.layers import Conv2D  from tensorflow.keras.layers import MaxPooling2D  from tensorflow.keras.layers import Activation  from tensorflow.keras.layers import Flatten  from tensorflow.keras.layers import Dropout  from tensorflow.keras.layers import Dense  from tensorflow.keras import backend as K  from keras.datasets import mnist  from keras.utils.np\_utils import to\_categorical  from keras.models import Sequential  from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, Activation  from keras.optimizers import Adam  from keras.wrappers.scikit\_learn import KerasClassifier  from keras.models import Sequential  from keras.layers.normalization import BatchNormalization  from keras.layers.convolutional import Conv2D  from keras.layers.convolutional import MaxPooling2D  from keras.layers.core import Activation  from keras.layers.core import Flatten  from keras.layers.core import Dropout  from keras.layers.core import Dense  from keras import backend as K  from keras.models import load\_model  from IPython.display import clear\_output, display  from keras.utils.vis\_utils import plot\_model  from sklearn.linear\_model import SGDClassifier  from sklearn.model\_selection import cross\_val\_predict  from sklearn.preprocessing import StandardScaler |

|  |
| --- |
| **The Function that gets the images, and does the preprocessing.** |
| def get\_image\_list\_from\_folder(imageFolderPath, dim1, dim2):    IMAGE\_DIMS = (dim1, dim2, 3)    data = []    labels = []    counter=0    imagePaths=glob.glob(imageFolderPath+'\*\*/\*.jpg', recursive=True)    # loop over the input images    for imagePath in imagePaths:      # load the image, pre-process it, and store it in the data list      try:        counter=counter+1        clear\_output(wait=True)        print("Try:"+str(counter)+"/"+str(len(imagePaths)))        image = cv2.imread(imagePath)        image\_norm = cv2.resize(image, (dim2, dim1))        image\_norm = img\_to\_array(image\_norm)        #Preproccessing        temp\_img= cv2.resize(image, (dim2, dim1))        temp\_img= cv2.GaussianBlur(temp\_img, (5, 5), 0)        gray = cv2.cvtColor(temp\_img, cv2.COLOR\_RGB2GRAY)        ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)        kernel = np.ones((3, 3), np.uint8)        opening = cv2.morphologyEx(thresh, cv2.MORPH\_OPEN, kernel, iterations=2)        sure\_bg = cv2.dilate(opening, kernel, iterations=3)        dist\_transform = cv2.distanceTransform(opening, cv2.DIST\_L2, 5)        ret, sure\_fg = cv2.threshold(dist\_transform, 0.7 \* dist\_transform.max(), 255, 0)        sure\_fg = np.uint8(sure\_fg)        unknown = cv2.subtract(sure\_bg, sure\_fg)        ret, markers = cv2.connectedComponents(sure\_fg)        markers = markers + 1        markers[unknown == 255] = 0        markers = cv2.watershed(temp\_img, markers)        temp\_img[markers == -1] = [255, 0, 0]        data.append(temp\_img)        data.append(image\_norm)        label = imagePath.split(os.path.sep)[-2]        labels.append(label)        labels.append(label)      except:        print("A corrupted file.")    return data, labels |
| **Color to Gray Transformer Function.** |
| class RGB2GrayTransformer(BaseEstimator, TransformerMixin):      def \_\_init\_\_(self):          pass      def fit(self, X, y=None):          return self      def transform(self, X, y=None):          return np.array([skimage.color.rgb2gray(img) for img in X]) |

|  |
| --- |
| **Hog Transform Function.** |
| class HogTransformer(BaseEstimator, TransformerMixin):      def \_\_init\_\_(self, y=None, orientations=9,                   pixels\_per\_cell=(8, 8),                   cells\_per\_block=(3, 3), block\_norm='L2-Hys'):          self.y = y          self.orientations = orientations          self.pixels\_per\_cell = pixels\_per\_cell          self.cells\_per\_block = cells\_per\_block          self.block\_norm = block\_norm      def fit(self, X, y=None):          return self      def transform(self, X, y=None):          def local\_hog(X):              return hog(X,                         orientations=self.orientations,                         pixels\_per\_cell=self.pixels\_per\_cell,                         cells\_per\_block=self.cells\_per\_block,                         block\_norm=self.block\_norm)          try:              return np.array([local\_hog(img) for img in X])          except:              return np.array([local\_hog(img) for img in X]) |
| **Getting the dataset and the labels and the preprocessed images.** |
| base\_folder\_address=os.getcwd()+"/drive/My Drive/project/Project-Files/Dataset\_2/"  #Specify the desired dimensions for the images  IMAGE\_DIMS = (100, 100, 3)  #Specify the folder that contains your images  data, labels=get\_image\_list\_from\_folder(base\_folder\_address, IMAGE\_DIMS[1], IMAGE\_DIMS[0])  print("Total image count with the processed ones:",len(data)) |
| **Binarizing the labels.** |
| lb = LabelBinarizer()  labels = lb.fit\_transform(labels)  print("You have",len(lb.classes\_),"classes at your data.") |
| **Training and Test Split.** |
| (X\_train, X\_test, y\_train, y\_test) = train\_test\_split(data, labels, test\_size=0.2, random\_state=42) |

|  |
| --- |
| **Applying the Hog Transformation and Scaling the Dataset.** |
| grayify = RGB2GrayTransformer()  hogify = HogTransformer(      pixels\_per\_cell=(8, 8),      cells\_per\_block=(2,2),      orientations=9,      block\_norm='L2-Hys'  )  scalify = StandardScaler()  X\_train\_gray = grayify.fit\_transform(X\_train)  X\_train\_hog = hogify.fit\_transform(X\_train\_gray)  X\_train\_prepared = scalify.fit\_transform(X\_train\_hog)  X\_test\_gray = grayify.transform(x\_test)  X\_test\_hog = hogify.transform(X\_test\_gray)  X\_test\_prepared = scalify.transform(X\_test\_hog)  print(X\_train\_prepared.shape) |
| **SGD Classifier.** |
| ydeneme=lb.inverse\_transform(y\_train)  yson=lb.inverse\_transform(y\_test)  sgd\_clf = SGDClassifier(random\_state=42, max\_iter=1000, tol=1e-3)  sgd\_clf.fit(X\_train\_prepared, ydeneme)  y\_pred = sgd\_clf.predict(X\_test\_prepared) |
| **Creating the Test Confusion Matrix.** |
| % matplotlib inline  cf\_matrix=confusion\_matrix(yson, y\_pred)  labels\_son=['Alarm\_Clock','Batteries','Fork','Glasses','Keyboard','Mouse','Mug','Pen','Scissors']  fig= plt.figure(figsize=(10,10))  sns.heatmap(cf\_matrix,xticklabels=labels\_son,yticklabels=labels\_son, annot=True,fmt='d') |
| **The Hog Pipeline.** |
| from sklearn.pipeline import Pipeline  from sklearn import svm  HOG\_pipeline = Pipeline([      ('grayify', RGB2GrayTransformer()),      ('hogify', HogTransformer(          pixels\_per\_cell=(8, 8),          cells\_per\_block=(2,2),          orientations=9,          block\_norm='L2-Hys')      ),      ('scalify', StandardScaler()),      ('classify', SGDClassifier(random\_state=42, max\_iter=1000, tol=1e-3))  ])  clf = HOG\_pipeline.fit(X\_train, ydeneme)  print('Percentage correct: ', 100\*np.sum(clf.predict(x\_test) == yson)/len(yson)) |

|  |
| --- |
| **Grid Search.** |
| from sklearn.model\_selection import GridSearchCV    param\_grid = [      {'hogify\_\_orientations': [9],      'hogify\_\_cells\_per\_block': [(2, 2)],      'hogify\_\_pixels\_per\_cell': [(7, 7), (8, 8)]},      {'hogify\_\_orientations': [9],       'hogify\_\_cells\_per\_block': [(2, 2)],       'hogify\_\_pixels\_per\_cell': [(7, 7)],       'classify': [           SGDClassifier(random\_state=42, max\_iter=1000, tol=1e-3),           svm.SVC(kernel='linear')]}  ]  grid\_search = GridSearchCV(HOG\_pipeline,                             param\_grid,                             cv=3,                             n\_jobs=-1,                             scoring='accuracy',                             verbose=1,                             return\_train\_score=True)    grid\_res = grid\_search.fit(X\_train, ydeneme) |
| **Getting the Best Parameters.** |
| print(grid\_res.best\_params\_) |