**MINI PROJECT – II**

**(2019-2020)**

# Pokemon Classification

# 

**SYNOPSIS**



**Institute of Engineering & Technology**

**Team Members**

Ayush Kumar Singh

171500074

Govind Sharma

171500115

Pranav Agarwal

171500224

## Supervised By

**Mr. Akash Kumar Chaudhary**

**Technical Trainer**

**Department of Computer Engineering & Applications**

**Name/Title of the project**

Pokemon Classification

**Problem Statement**

In todays era there are lot many games which are based on machine learning and attain maximum efficiency. In this we have also a method to classify different species which are used in famous games Pokemon go and keeping in mind to avoid over fitting.

**Reason for selecting the topic**

Due to increase of mobile games which are totally depend upon Machine Learning. Today’s youth is just crazy which connect them to real life and has minimum error and with such a huge amount of data they do not occur error.

**Objectives of the project**

The project aims at using state of the art machine learning algorithms for classifying pictures of Pokemon. The pictures belong to three specific type of Pokemon (Pikachu, Butterfree, Ditto) The project also uses a neural netwok to avoid overfitting.

**Literature Survey/Feasibility Study**

Young children may benefit from the incorporation of devices and game applications with use of their inhalers. Feedback from children and parents suggests potential benefits to children may include learning correct use, reducing distress and anxiety and improving overall adherence.

**Future Scope**

This will help a lot more in society in many different ways:

1. Various pictures recognition.
2. Game development.

**Methodology**

We are creating different files which consists many images of Pikachu, Butterfree, Ditto at different angles and different size so that the data content adequate amount of photographs so that it efficiency can be up to maximum

**Hardware and Software to be used**

1. *Hardware****:***

NO hardware required.

1. *Software*:

* Jupyter Ide

**Testing technologies to be used**

* *Smartphone*

**What contribution would the project make and where?**

The project will contribute in the field of Machine learning in recognition of different pictures and developing different mobile games.

**Scope for extension into a major project**

*Things that can be added to extend this project into a major one:*

1. Various pictures recognition.
2. Game development.

Code:

|  |
| --- |
|  |
| import numpy as np |
|  | import matplotlib.pyplot as plt |
|  | import pandas as pd |
|  | import cv2 |
|  |  |
|  | img = cv2.imread('C:/Users/Rahul/Desktop/data/Pikachu/b.jpg', cv2.IMREAD\_UNCHANGED) |
|  |  |
|  | print('Original Dimensions : ',img.shape) |
|  |  |
|  | scale\_percent = 60 # percent of original size |
|  | width = int(img.shape[1] \* scale\_percent / 100) |
|  | height = int(img.shape[0] \* scale\_percent / 100) |
|  | dim = (width, height) |
|  | # resize image |
|  | resized = cv2.resize(img, dim, interpolation = cv2.INTER\_AREA) |
|  |  |
|  | print('Resized Dimensions : ',resized.shape) |
|  |  |
|  | cv2.imshow("Resized image", resized) |
|  | cv2.waitKey(0) |
|  | cv2.destroyAllWindows() |
|  |  |
|  | cv2.imshow("Original image", img) |
|  | cv2.waitKey(0) |
|  | cv2.destroyAllWindows() |
|  |  |
|  |  |
|  | ################## Resizing all images ################## |
|  |  |
|  |  |
|  | desired\_size = 368 |
|  | im\_pth = "C:/Users/Rahul/Desktop/data/Pikachu/a.jpg" |
|  |  |
|  | im = cv2.imread(im\_pth) |
|  | old\_size = im.shape[:2] # old\_size is in (height, width) format |
|  |  |
|  | ratio = float(desired\_size)/max(old\_size) |
|  | new\_size = tuple([int(x\*ratio) for x in old\_size]) |
|  |  |
|  | # new\_size should be in (width, height) format |
|  |  |
|  | im = cv2.resize(im, (new\_size[1], new\_size[0])) |
|  |  |
|  | delta\_w = desired\_size - new\_size[1] |
|  | delta\_h = desired\_size - new\_size[0] |
|  | top, bottom = delta\_h//2, delta\_h-(delta\_h//2) |
|  | left, right = delta\_w//2, delta\_w-(delta\_w//2) |
|  |  |
|  | color = [0, 0, 0] |
|  | new\_im = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER\_CONSTANT, |
|  | value=color) |
|  |  |
|  | cv2.imshow("image", new\_im) |
|  | cv2.waitKey(0) |
|  | cv2.destroyAllWindows() |
|  |  |
|  | cv2.imwrite('C:/Users/Rahul/Desktop/a.jpg', new\_im) |
|  |  |
|  |  |
|  | ########### Reading multiple images at once |
|  |  |
|  | import glob |
|  |  |
|  | images = [cv2.imread(file) for file in glob.glob("C:/Users/Rahul/Desktop/data/Pikachu/\*.jpg")] |
|  |  |
|  | images\_1 = [cv2.imread(file) for file in glob.glob("C:/Users/Rahul/Desktop/data/Butterfree/\*.jpg")] |
|  |  |
|  | images\_2 = [cv2.imread(file) for file in glob.glob("C:/Users/Rahul/Desktop/data/Ditto/\*.jpg")] |
|  |  |
|  |  |
|  |  |
|  | ############################################################### |
|  |  |
|  | mera\_dat = [] |
|  |  |
|  | for i in range(199): |
|  | desired\_size = 368 |
|  |  |
|  | im = images[i] |
|  | old\_size = im.shape[:2] # old\_size is in (height, width) format |
|  |  |
|  | ratio = float(desired\_size)/max(old\_size) |
|  | new\_size = tuple([int(x\*ratio) for x in old\_size]) |
|  |  |
|  | # new\_size should be in (width, height) format |
|  |  |
|  | im = cv2.resize(im, (new\_size[1], new\_size[0])) |
|  |  |
|  | delta\_w = desired\_size - new\_size[1] |
|  | delta\_h = desired\_size - new\_size[0] |
|  | top, bottom = delta\_h//2, delta\_h-(delta\_h//2) |
|  | left, right = delta\_w//2, delta\_w-(delta\_w//2) |
|  |  |
|  | color = [0, 0, 0] |
|  | new\_im = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER\_CONSTANT, |
|  | value=color) |
|  |  |
|  | # cv2.imshow("image", new\_im) |
|  | # cv2.waitKey(0) |
|  | # cv2.destroyAllWindows() |
|  | # |
|  | # cv2.imwrite('C:/Users/Rahul/Desktop/a.jpg'.format(i), new\_im) |
|  | mera\_dat.append(new\_im) |
|  |  |
|  |  |
|  | cv2.imshow("", mera\_dat[120]) |
|  |  |
|  |  |
|  | ############# Butterfree |
|  |  |
|  | mera\_dat\_1 = [] |
|  |  |
|  | for i in range(66): |
|  | desired\_size = 368 |
|  |  |
|  | im = images\_1[i] |
|  | old\_size = im.shape[:2] # old\_size is in (height, width) format |
|  |  |
|  | ratio = float(desired\_size)/max(old\_size) |
|  | new\_size = tuple([int(x\*ratio) for x in old\_size]) |
|  |  |
|  | # new\_size should be in (width, height) format |
|  |  |
|  | im = cv2.resize(im, (new\_size[1], new\_size[0])) |
|  |  |
|  | delta\_w = desired\_size - new\_size[1] |
|  | delta\_h = desired\_size - new\_size[0] |
|  | top, bottom = delta\_h//2, delta\_h-(delta\_h//2) |
|  | left, right = delta\_w//2, delta\_w-(delta\_w//2) |
|  |  |
|  | color = [0, 0, 0] |
|  | new\_im = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER\_CONSTANT, |
|  | value=color) |
|  |  |
|  | # cv2.imshow("image", new\_im) |
|  | # cv2.waitKey(0) |
|  | # cv2.destroyAllWindows() |
|  | # |
|  | # cv2.imwrite('C:/Users/Rahul/Desktop/a.jpg'.format(i), new\_im) |
|  | mera\_dat\_1.append(new\_im) |
|  |  |
|  |  |
|  | ########### Ditto |
|  |  |
|  |  |
|  | mera\_dat\_2 = [] |
|  |  |
|  | for i in range(56): |
|  | desired\_size = 368 |
|  |  |
|  | im = images\_2[i] |
|  | old\_size = im.shape[:2] # old\_size is in (height, width) format |
|  |  |
|  | ratio = float(desired\_size)/max(old\_size) |
|  | new\_size = tuple([int(x\*ratio) for x in old\_size]) |
|  |  |
|  | # new\_size should be in (width, height) format |
|  |  |
|  | im = cv2.resize(im, (new\_size[1], new\_size[0])) |
|  |  |
|  | delta\_w = desired\_size - new\_size[1] |
|  | delta\_h = desired\_size - new\_size[0] |
|  | top, bottom = delta\_h//2, delta\_h-(delta\_h//2) |
|  | left, right = delta\_w//2, delta\_w-(delta\_w//2) |
|  |  |
|  | color = [0, 0, 0] |
|  | new\_im = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER\_CONSTANT, |
|  | value=color) |
|  |  |
|  | # cv2.imshow("image", new\_im) |
|  | # cv2.waitKey(0) |
|  | # cv2.destroyAllWindows() |
|  | # |
|  | # cv2.imwrite('C:/Users/Rahul/Desktop/a.jpg'.format(i), new\_im) |
|  | mera\_dat\_2.append(new\_im) |
|  |  |
|  |  |
|  | arr = np.array(mera\_dat) |
|  | arr = arr.reshape((199, 406272)) |
|  |  |
|  | ar1 = np.array(mera\_dat\_1) |
|  | ar1 = ar1.reshape((66, 406272)) |
|  |  |
|  | ar2 = np.array(mera\_dat\_2) |
|  | ar2 = ar2.reshape((56, 406272)) |
|  |  |
|  | arr = arr / 255 |
|  | ar1 = ar1 / 255 |
|  | ar2 = ar2 / 255 |
|  |  |
|  | dataset = pd.DataFrame(arr) |
|  | dataset['label'] = np.ones(199) |
|  |  |
|  | dataset.iloc[:, -1] |
|  |  |
|  | dataset\_1 = pd.DataFrame(ar1) |
|  | dataset\_1['label'] = np.zeros(66) |
|  |  |
|  | dataset\_1.iloc[:, -1] |
|  |  |
|  | dataset\_2 = pd.DataFrame(ar2) |
|  | dataset\_2['label'] = np.array(np.ones(56) + np.ones(56)) |
|  |  |
|  | dataset\_2.iloc[:, -1] |
|  |  |
|  | dataset\_master = pd.concat([dataset, dataset\_1, dataset\_2]) |
|  |  |
|  | dataset\_master.iloc[:, 406272] |
|  |  |
|  | X = dataset\_master.iloc[:, 0:406272].values |
|  | y = dataset\_master.iloc[:, -1].values |
|  |  |
|  |  |
|  | from sklearn.tree import DecisionTreeClassifier |
|  | dtf = DecisionTreeClassifier(max\_depth = 3) |
|  | dtf.fit(X, y) |
|  |  |
|  | dtf.score(X, y) |
|  |  |
|  | from sklearn.naive\_bayes import GaussianNB |
|  | nb = GaussianNB() |
|  | nb.fit(X, y) |
|  |  |
|  | nb.score(X, y) |
|  |  |
|  | from sklearn.linear\_model import LogisticRegression |
|  | log\_reg = LogisticRegression() |
|  | log\_reg.fit(X, y) |
|  |  |
|  | log\_reg.score(X, y) |
|  |  |
|  | from sklearn.svm import SVC |
|  | svm = SVC() |
|  | svm.fit(X, y) |
|  |  |
|  | svm.score(X, y) |
|  |  |
|  | from sklearn.cluster import KMeans |
|  |  |
|  | wcv = [] |
|  |  |
|  | for i in range(1, 8): |
|  | km = KMeans(n\_clusters = i) |
|  | km.fit(X) |
|  | wcv.append(km.inertia\_) |
|  |  |
|  | plt.plot(range(1, 8), wcv) |
|  | plt.show() |
|  |  |
|  |  |
|  | import tensorflow as tf |
|  | from tensorflow import keras |
|  |  |
|  | model = keras.models.Sequential() |
|  | model.add(keras.layers.Dense(256, activation = 'relu')) |
|  | model.add(keras.layers.Dense(128, activation = 'relu')) |
|  | model.add(keras.layers.Dense(3, activation = 'softmax')) |
|  |  |
|  | model.compile(loss = 'sparse\_categorical\_crossentropy', optimizer = 'adam', metrics = ['accuracy']) |
|  |  |
|  | history = model.fit(X, y, epochs = 5) |
|  |  |
|  | pd.DataFrame(history.history).plot(figsize = (8, 5)) |
|  | plt.grid(True) |
|  | plt.gca().set\_ylim(0, 1) |
|  | plt.show() |
|  |  |

*Importing the libraries*

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **pandas** **as** **pd**

**import** **cv2**

In [2]:

*# Using glob to read all pokemon images at once*

*# Don't forget to change the path when copying this project*

**import** **glob**

images = [cv2.imread(file) **for** file **in** glob.glob("C:/Users/Rahul/Desktop/data/Pikachu/\*.jpg")]

images\_1 = [cv2.imread(file) **for** file **in** glob.glob("C:/Users/Rahul/Desktop/data/Butterfree/\*.jpg")]

images\_2 = [cv2.imread(file) **for** file **in** glob.glob("C:/Users/Rahul/Desktop/data/Ditto/\*.jpg")]

In [3]:

images, images\_1, images\_2

In [4]:

*# Scaling and resizing all the Pikachu images and saving the result in a new list called mera\_dat*

mera\_dat = []

**for** i **in** range(199):

desired\_size = 368

im = images[i]

old\_size = im.shape[:2] *# old\_size is in (height, width) format*

ratio = float(desired\_size)/max(old\_size)

new\_size = tuple([int(x\*ratio) **for** x **in** old\_size])

*# new\_size should be in (width, height) format*

im = cv2.resize(im, (new\_size[1], new\_size[0]))

delta\_w = desired\_size - new\_size[1]

delta\_h = desired\_size - new\_size[0]

top, bottom = delta\_h//2, delta\_h-(delta\_h//2)

left, right = delta\_w//2, delta\_w-(delta\_w//2)

color = [0, 0, 0]

new\_im = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER\_CONSTANT,

value=color)

*# cv2.imshow("image", new\_im)*

*# cv2.waitKey(0)*

*# cv2.destroyAllWindows()*

*#*

*# cv2.imwrite('C:/Users/Rahul/Desktop/a.jpg'.format(i), new\_im)*

mera\_dat.append(new\_im)

In [5]:

*# Scaling and resizing all the Butterfree images and saving the result in a new list called mera\_dat*

mera\_dat\_1 = []

**for** i **in** range(66):

desired\_size = 368

im = images\_1[i]

old\_size = im.shape[:2] *# old\_size is in (height, width) format*

ratio = float(desired\_size)/max(old\_size)

new\_size = tuple([int(x\*ratio) **for** x **in** old\_size])

*# new\_size should be in (width, height) format*

im = cv2.resize(im, (new\_size[1], new\_size[0]))

delta\_w = desired\_size - new\_size[1]

delta\_h = desired\_size - new\_size[0]

top, bottom = delta\_h//2, delta\_h-(delta\_h//2)

left, right = delta\_w//2, delta\_w-(delta\_w//2)

color = [0, 0, 0]

new\_im = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER\_CONSTANT,

value=color)

*# cv2.imshow("image", new\_im)*

*# cv2.waitKey(0)*

*# cv2.destroyAllWindows()*

*#*

*# cv2.imwrite('C:/Users/Rahul/Desktop/a.jpg'.format(i), new\_im)*

mera\_dat\_1.append(new\_im)

In [6]:

*# Scaling and resizing all the Ditto images and saving the result in a new list called mera\_dat\_2*

mera\_dat\_2 = []

**for** i **in** range(56):

desired\_size = 368

im = images\_2[i]

old\_size = im.shape[:2] *# old\_size is in (height, width) format*

ratio = float(desired\_size)/max(old\_size)

new\_size = tuple([int(x\*ratio) **for** x **in** old\_size])

*# new\_size should be in (width, height) format*

im = cv2.resize(im, (new\_size[1], new\_size[0]))

delta\_w = desired\_size - new\_size[1]

delta\_h = desired\_size - new\_size[0]

top, bottom = delta\_h//2, delta\_h-(delta\_h//2)

left, right = delta\_w//2, delta\_w-(delta\_w//2)

color = [0, 0, 0]

new\_im = cv2.copyMakeBorder(im, top, bottom, left, right, cv2.BORDER\_CONSTANT,

value=color)

*# cv2.imshow("image", new\_im)*

*# cv2.waitKey(0)*

*# cv2.destroyAllWindows()*

*#*

*# cv2.imwrite('C:/Users/Rahul/Desktop/a.jpg'.format(i), new\_im)*

mera\_dat\_2.append(new\_im)

In [7]:

*# Converting the preprocessed and resized list into numpy arrays and performing normalization*

arr = np.array(mera\_dat)

arr = arr.reshape((199, 406272))

ar1 = np.array(mera\_dat\_1)

ar1 = ar1.reshape((66, 406272))

ar2 = np.array(mera\_dat\_2)

ar2 = ar2.reshape((56, 406272))

arr = arr / 255

ar1 = ar1 / 255

ar2 = ar2 / 255

In [8]:

*# Scaled Images in numpy ndarray data structure*

arr, ar1, ar2

In [9]:

*# Converting the numpy arrays to pandas dataframe structure*

*# Generating labels for different pokemons*

*# label 1 is for Pikachu*

*# label 0 is for Butterfree*

*# label 2 is for Ditto*

dataset = pd.DataFrame(arr)

dataset['label'] = np.ones(199)

dataset.iloc[:, -1]

dataset\_1 = pd.DataFrame(ar1)

dataset\_1['label'] = np.zeros(66)

dataset\_1.iloc[:, -1]

dataset\_2 = pd.DataFrame(ar2)

dataset\_2['label'] = np.array(np.ones(56) + np.ones(56))

dataset\_2.iloc[:, -1]

*# Concatenating everything into a master dataframe*

dataset\_master = pd.concat([dataset, dataset\_1, dataset\_2])

dataset\_master

In [10]:

*# Splitting the dataset into feature matrix 'X' and vector of predictions 'y'*

X = dataset\_master.iloc[:, 0:406272].values

y = dataset\_master.iloc[:, -1].values

X, y

In [11]:

*# Implementing a simple ANN architecture for the classification problem*

**import** **tensorflow** **as** **tf**

**from** **tensorflow** **import** keras

model = keras.models.Sequential()

model.add(keras.layers.Dense(256, activation = 'relu'))

model.add(keras.layers.Dense(128, activation = 'relu'))

model.add(keras.layers.Dense(3, activation = 'softmax'))

model.compile(loss = 'sparse\_categorical\_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

history = model.fit(X, y, epochs = 5)

In [12]:

*# Visualizing the results*

pd.DataFrame(history.history).plot(figsize = (8, 5))

plt.grid(**True**)

plt.gca().set\_ylim(0, 1)

plt.show()

Result

Train on 321 samples

Epoch 1/5

321/321 [==============================] - 9s 30ms/sample - loss: 163.4301 - accuracy: 0.5109

Epoch 2/5

321/321 [==============================] - 6s 19ms/sample - loss: 86.0971 - accuracy: 0.6106

Epoch 3/5

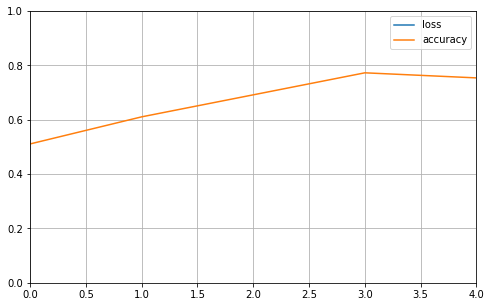
321/321 [==============================] - 5s 16ms/sample - loss: 22.4777 - accuracy: 0.6916

Epoch 4/5

321/321 [==============================] - 5s 16ms/sample - loss: 11.3030 - accuracy: 0.7726

Epoch 5/5

321/321 [==============================] - 5s 16ms/sample - loss: 12.3582 - accuracy: 0.7539



**Bibliography**

1. <https://link.springer.com/article/10.1007/s41030-016-0023>
2. <https://cloud.google.com/machinelearning.com/docs/>
3. <https://nueralnetwork.com/docs>