

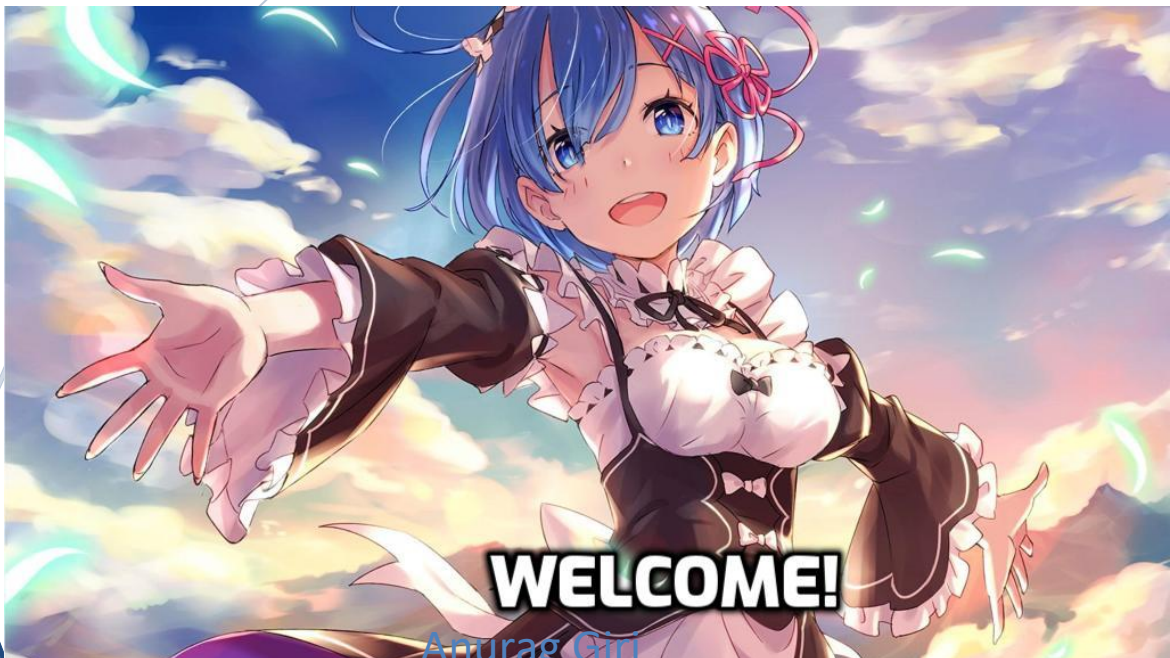
Project

ANIME

CHARACTER

RECOGNITION

(USING CONVOLUTIONAL NEURAL
NETWORKS)



COMPUTER SCIENCE UNDERGRAD

Abstract

The aim of this project is to create an image classifier for animated characters. As animation differs vastly due to different art styles, so here I have chosen the most popular ones amongst them - the Japanese anime characters.

Even though the features of these characters are similar to humans (like eyes, lips, brows, etc.) but due to difference in the way they are drawn and face complexion the human image detectors and classifiers does not perform very well on these characters. Due to lack of availability of dataset, I had to create my own. So, for starters I have taken 16 characters (1,280 images) but it can be easily expanded for many more in future.

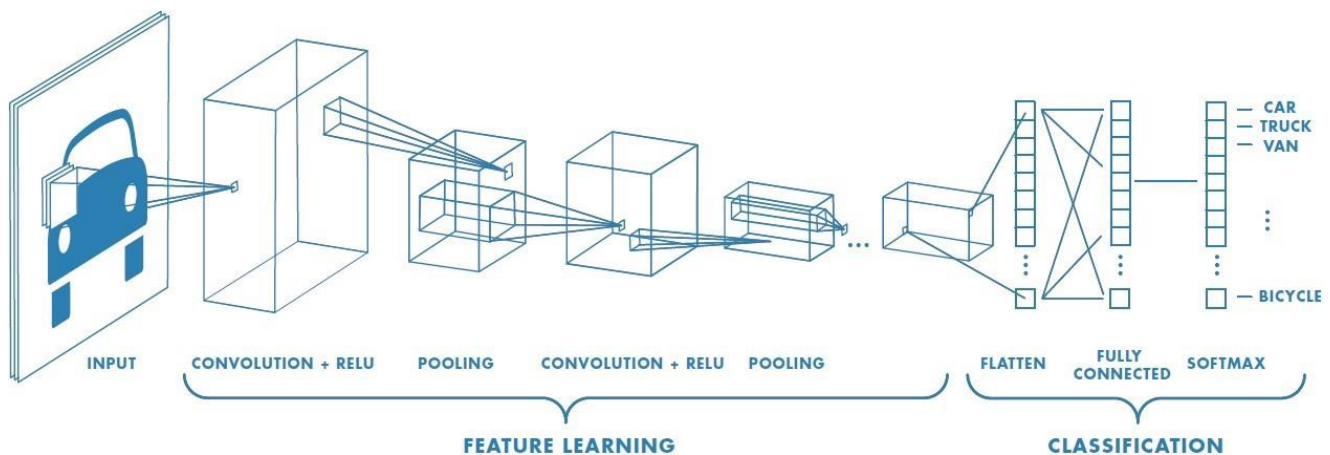
Table of Contents

ABSTRACT.....	2
INTRODUCTION	4
PREPARATION	
I. Technologies Used	5
II. Transfer Learning	5
III. Scrape and clean up the data	6
ANALYZE THE PROBLEM	
I. Face Detection (Introduction)	7
II. Face Detection (Human Face)	8
III. 2D Face is not equal to 3D Face	9
IV. Face Detection: Train New Model.....	10
TRAIN & VALIDATE THE MODEL	
I. Structure of the Model.....	11
II. Building the Model.....	12
III. Model Description.....	13
IV. Model Training	14
V. Training & Validation Accuracy	15
DEMO	
I. Prediction Code	16
II. Examples	17-18

Introduction

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

CNNs allow us to extract a wide range of features from images. Turns out, we can use this idea for face recognition too!



The approach I am going to use for face recognition is fairly straight forward. The key here is to get a deep neural network to produce a bunch of numbers that describe a face (known as face encodings). When you pass in two different images of the same character, the network should return similar outputs (i.e. closer numbers) for both images, whereas when you pass in images of two different characters, the network should return very different outputs for the two images. This means that the neural network needs to be trained to automatically identify different features of faces and calculate numbers based on that. The output of the neural network can be thought of as an identifier for a particular character's face — if you pass in different images of the same person, the output of the neural network will be very similar/close, whereas if you pass in images of a different character, the output will be very different.

STEP 1. PREPARATION

Technologies Used

OpenCV

OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. The library is cross-platform and free for use under the open-source BSD license.

Keras

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

Tensorflow

Library for Machine Learning algorithms and functions for implement Neural Networks

Python

Used as a glue language

Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems. Transfer Learning turns out to be very useful when the dataset available for the second model is comparatively smaller than the first model.

Why Transfer Learning is not a better choice here?

Transfer Learning is not a good strategy for a model build for recognition of animated images because models most frequently used for transfer learning (like VGG-16, Inception-V3 etc.) are all trained on Imagenet dataset which mostly contains images of 3D objects and 2D anime character's facial features are somewhat different from 3D images.

The models build using transfer learning yield almost no accuracy for this project.

Scrape and clean up the data

This project only uses 80 images per character for learning, which is a very low number for image recognition learning. However, this number is chosen since the majority of characters have a limited number of arts.

- Utilize lbpcascade_animeface to recognize character face from each image
- Resize each image to 150 x 150 pixels
- Split images into training & validation before creating the final model

```
dim = (150, 150)

def detect(filename, cascade_file = "C:/Users/Anurag Giri/Desktop/lbpcascade_animeface.xml"):
    cascade = cv2.CascadeClassifier(cascade_file)
    image = cv2.imread(filename, cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.equalizeHist(gray)

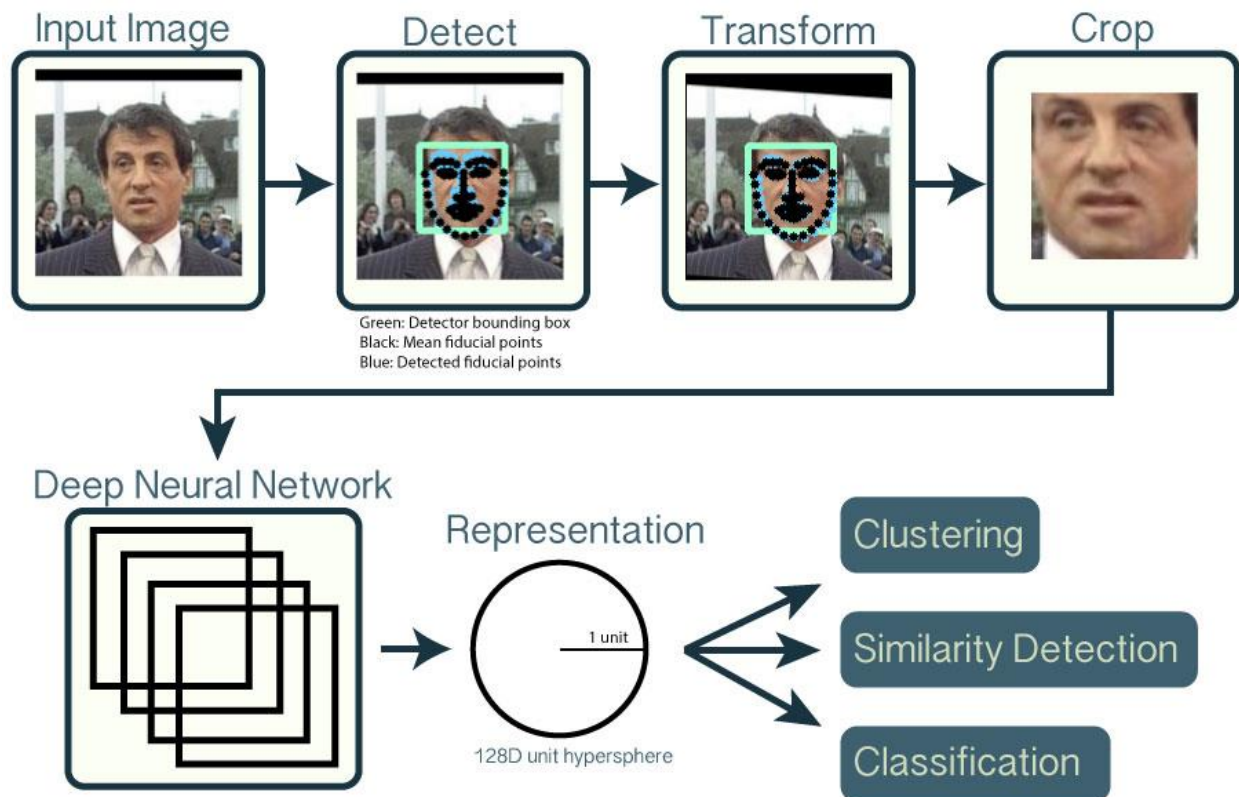
    faces = cascade.detectMultiScale(gray, scaleFactor = 1.1, minNeighbors = 5, minSize = (24, 24))
    for (x, y, w, h) in faces:
        val = max(w,h)
        offset = int(val/12)
        new_img = image[y-(2*offset):y+val,x-offset:x+val+offset]
        new_img = cv2.resize(new_img, dim, interpolation = cv2.INTER_AREA)
        cv2.imwrite(filename.split(".")[0]+"_uwu"+str(x)+"."+filename.split(".")[1],new_img)
```

STEP 2. ANALYZE THE PROBLEM

Face Detection: Introduction

First of all, the face recognition system needs to find a face in the image and highlight this area. For this, the software can use a variety of algorithms: for example, determining the similarity of proportions and skin color, the selection of contours in the image and their comparison with the contours of faces, the selection of symmetries using neural networks. The most effective is the **Viola-Jones method**, which can be used in real time. With it, the system recognizes faces even when rotated 30 degrees.

The most important measurements for face recognition programs are the distance between the eyes, the width of the nostrils, the length of the nose, the height and shape of the cheekbones, the width of the chin, the height of the forehead and other parameters.



Face Detection (Human Face)

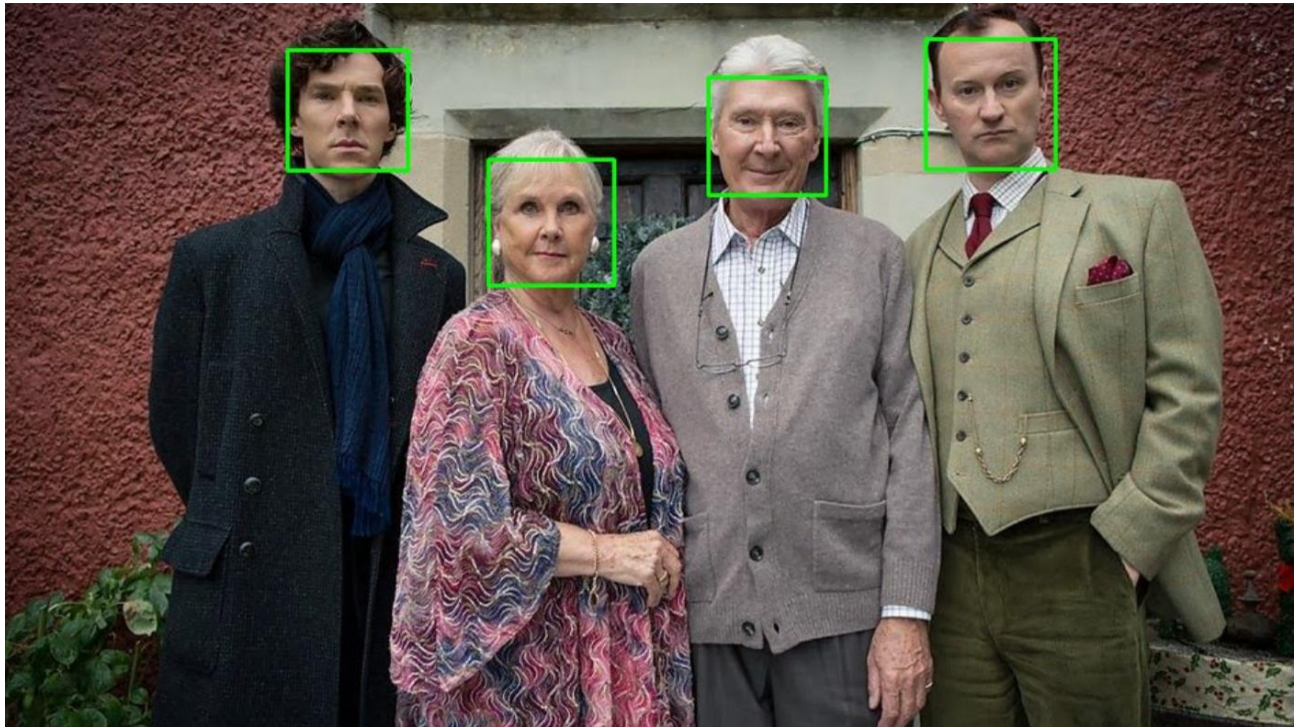
Adapted from: <https://github.com/shantnu/FaceDetect>

Code:

```
import cv2

faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
image = cv2.imread(imagePath)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = faceCascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))
```

➤ Model's response to Human Face



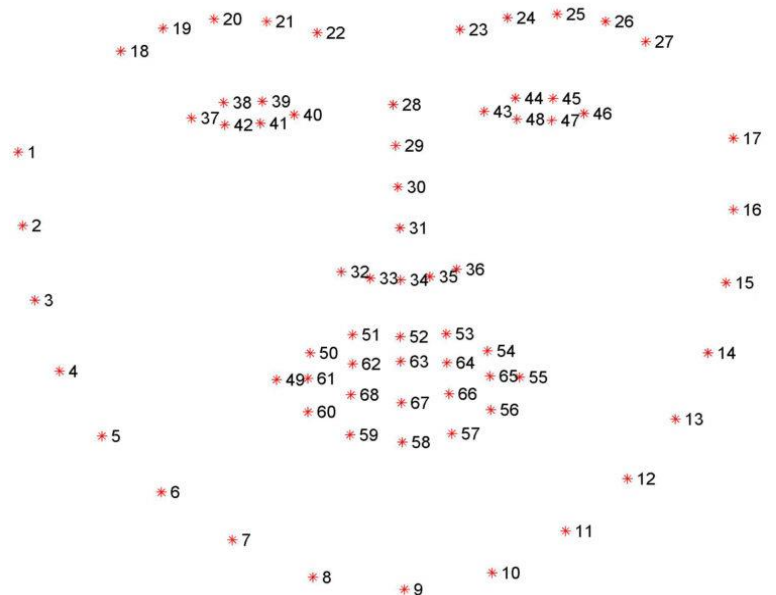
➤ Same Model on Anime Face



2D face is not equal to 3D face!

My initial naive idea was considering facial features of 2D anime character are equal to 3D human. That was proven wrong later on.

Facial features are different!
e.g.: 2D has no nose



Face Detection: Train New Model!

Fortunately, [nagadomi](#), the creator of waifu2x, has created a face detector for anime characters using OpenCV which is based on LBP cascade.

Adapted from : https://github.com/nagadomi/lbpcascade_animeface

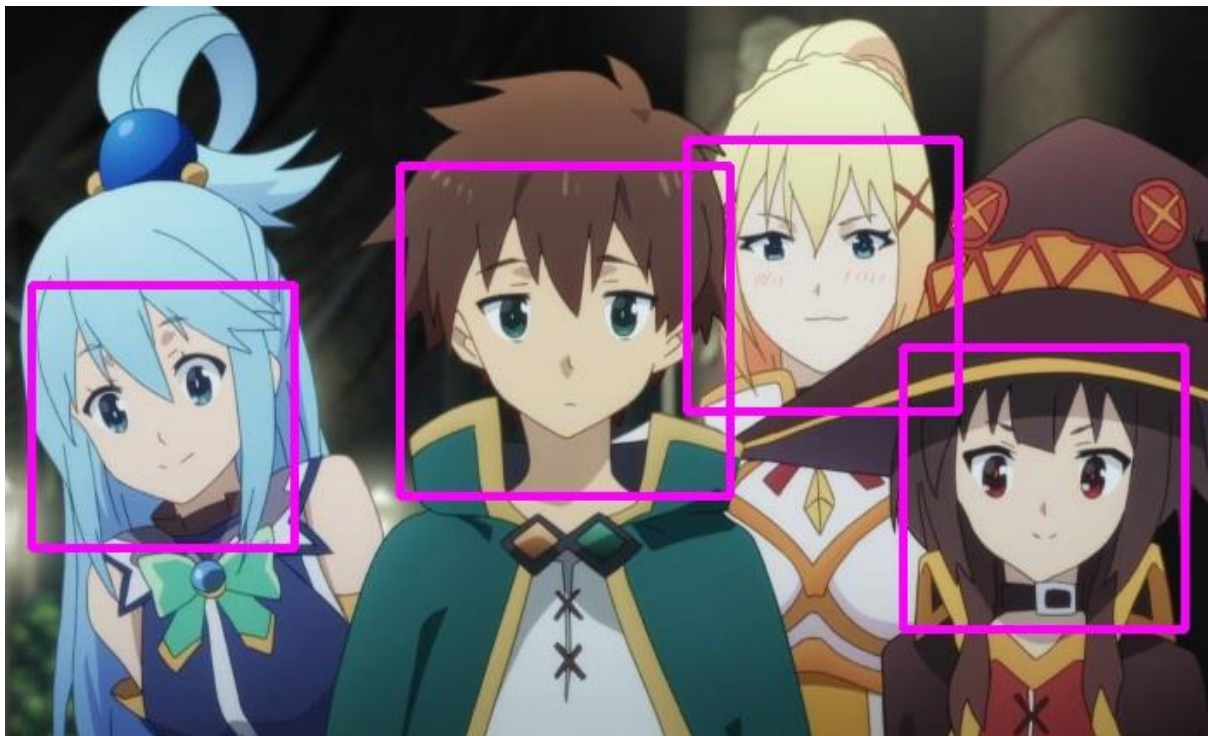
lbpcascade_animeface can detect character faces with an accuracy of around **83%**.

Code:

```
import cv2

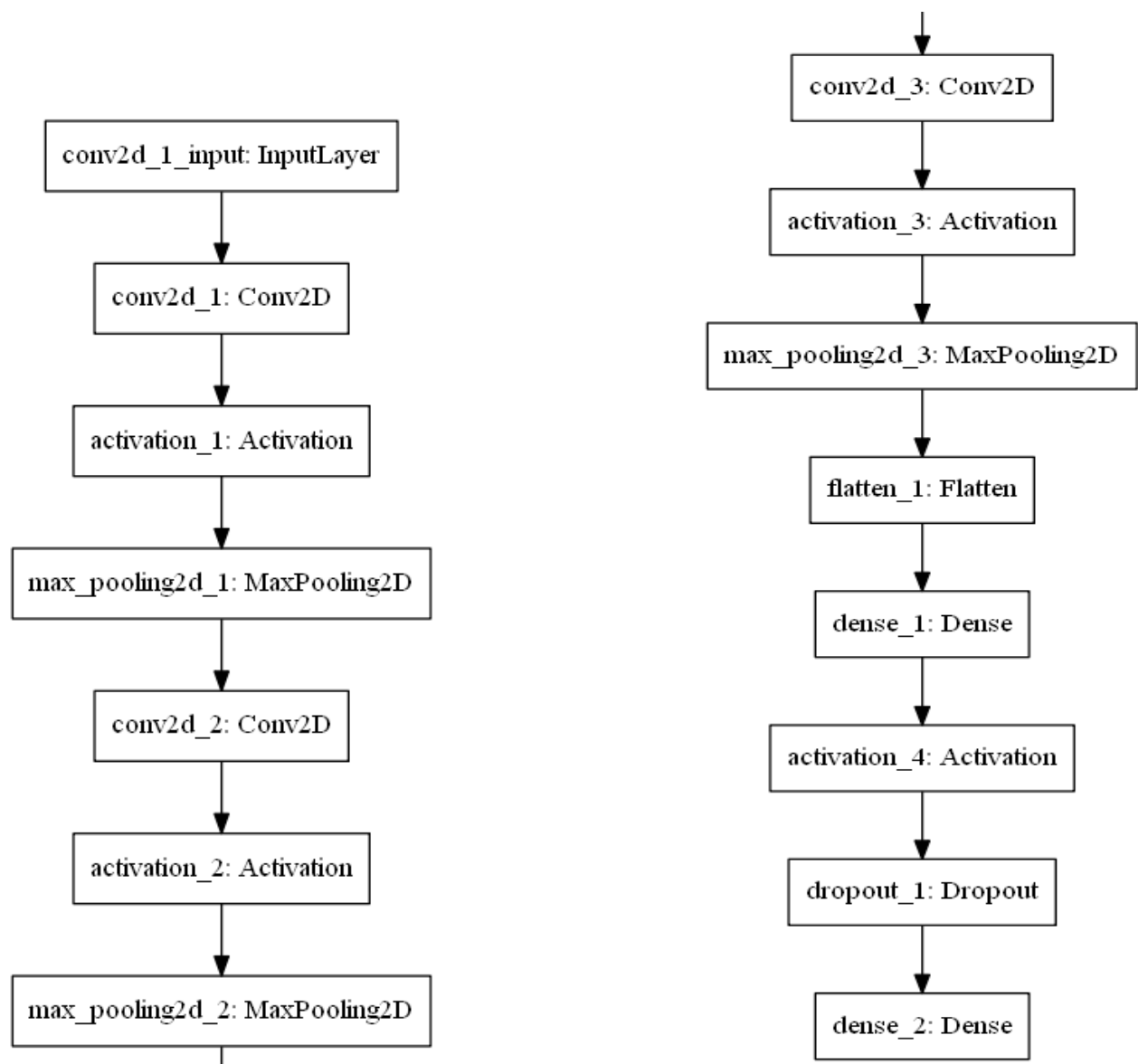
cascade = cv2.CascadeClassifier("lbpcascade_animeface.xml")
image = cv2.imread(imagePath)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = cascade.detectMultiScale(gray, scaleFactor=1.1,
                                minNeighbors=5, minSize=(24, 24))
```

➤ New Model's response to Anime Face



STEP 3. TRAIN & VALIDATE THE MODEL

Structure of the Model



Building the model

```
49 model = Sequential()
50 model.add(Conv2D(32, (3, 3), input_shape=input_shape))
51 model.add(Activation('relu'))
52 model.add(MaxPooling2D(pool_size=(2,2)))
53
54 model.add(Conv2D(32, (3, 3), input_shape=input_shape))
55 model.add(Activation('relu'))
56 model.add(MaxPooling2D(pool_size=(2,2)))
57
58 model.add(Conv2D(64, (3, 3), input_shape=input_shape))
59 model.add(Activation('relu'))
60 model.add(MaxPooling2D(pool_size=(2,2)))
61
62 model.add(Flatten())
63 model.add(Dense(64))
64 model.add(Activation('relu'))
65 model.add(Dropout(0.3))
66 model.add(Dense(16, activation='softmax'))
67
68 optimizer = keras.optimizers.SGD(lr=0.001, decay=1e-6, momentum=0.09, nesterov=False)
69
70 model.compile(loss='categorical_crossentropy',
71               optimizer=optimizer,
72               metrics=['categorical_accuracy'])
73
74 history = model.fit_generator(
75     train_generator,
76     steps_per_epoch=nb_train_samples,
77     epochs=epochs,
78     validation_data=validation_generator,
79     validation_steps=nb_validation_samples)
```

CNN Layers

- **Convolutional:** Convolution is a mathematical operation to merge two sets of information. In our case the convolution is applied on the input data using a *convolution filter* to produce a *feature map*.
- **Pooling:** The Pooling layer is responsible for reducing the spatial size of the Convolved Feature.
- **Dropout:** Dropping out units to prevent overfitting.
- **Fully Connected:** Extracting global features, every node in the layer is connected to the preceding layer.
- **Softmax:** Squashing final layer to make a prediction, which sums up to 1. For example, if we have 2 classes and class A has the value of 0.95, then class B will have the value of 0.05.

Model Description

- Model - Sequential
- Optimizer - SGD
- Neurons in 1st Convolutional Layer - 32
- Neurons in 2nd Convolutional Layer - 32
- Neurons in 3rd Convolutional Layer - 64
- Dropout Value - 0.3

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
=====		
conv2d_1 (Conv2D)	(None, 148, 148, 32)	896
activation_1 (Activation)	(None, 148, 148, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_2 (Conv2D)	(None, 72, 72, 32)	9248
activation_2 (Activation)	(None, 72, 72, 32)	0
max_pooling2d_2 (MaxPooling2D)	(None, 36, 36, 32)	0
conv2d_3 (Conv2D)	(None, 34, 34, 64)	18496
activation_3 (Activation)	(None, 34, 34, 64)	0
max_pooling2d_3 (MaxPooling2D)	(None, 17, 17, 64)	0
flatten_1 (Flatten)	(None, 18496)	0
dense_1 (Dense)	(None, 64)	1183808
activation_4 (Activation)	(None, 64)	0
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 11)	715
=====		
Total params: 1,213,163		
Trainable params: 1,213,163		
Non-trainable params: 0		

Model Training

```
Epoch 1/100
2020-05-07 01:52:42.425831: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cublas64_10.dll
2020-05-07 01:52:44.703256: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudnn64_7.dll
2020-05-07 01:52:52.588118: W tensorflow/stream_executor/gpu/redzone_allocator.cc:312] Internal: Invoking GPU asm compilation is supported on Cuda non-Windows
platforms only
Relying on driver to perform ptx compilation. This message will be only logged once.
200/200 [=====] - 27s 134ms/step - loss: 2.7624 - categorical_accuracy: 0.0775 - val_loss: 2.7649 - val_categorical_accuracy: 0.0750
Epoch 2/100
200/200 [=====] - 12s 58ms/step - loss: 2.7321 - categorical_accuracy: 0.1055 - val_loss: 2.7381 - val_categorical_accuracy: 0.0700
Epoch 3/100
200/200 [=====] - 13s 65ms/step - loss: 2.6848 - categorical_accuracy: 0.1190 - val_loss: 2.5510 - val_categorical_accuracy: 0.0900
Epoch 4/100
200/200 [=====] - 15s 73ms/step - loss: 2.6057 - categorical_accuracy: 0.1425 - val_loss: 2.6095 - val_categorical_accuracy: 0.1600
Epoch 5/100
200/200 [=====] - 14s 68ms/step - loss: 2.4825 - categorical_accuracy: 0.1885 - val_loss: 2.5525 - val_categorical_accuracy: 0.2350
Epoch 6/100
200/200 [=====] - 13s 66ms/step - loss: 2.3436 - categorical_accuracy: 0.2420 - val_loss: 2.4452 - val_categorical_accuracy: 0.3450
Epoch 7/100
200/200 [=====] - 10s 52ms/step - loss: 2.1719 - categorical_accuracy: 0.2930 - val_loss: 1.8015 - val_categorical_accuracy: 0.3200
Epoch 8/100
200/200 [=====] - 12s 62ms/step - loss: 2.0821 - categorical_accuracy: 0.3110 - val_loss: 1.7000 - val_categorical_accuracy: 0.5500
Epoch 9/100
200/200 [=====] - 10s 51ms/step - loss: 1.8993 - categorical_accuracy: 0.3810 - val_loss: 1.8847 - val_categorical_accuracy: 0.5200
Epoch 10/100
200/200 [=====] - 11s 57ms/step - loss: 1.7657 - categorical_accuracy: 0.4045 - val_loss: 1.2225 - val_categorical_accuracy: 0.5850
Epoch 11/100
200/200 [=====] - 12s 58ms/step - loss: 1.6396 - categorical_accuracy: 0.4580 - val_loss: 1.1126 - val_categorical_accuracy: 0.6150
Epoch 12/100
200/200 [=====] - 14s 72ms/step - loss: 1.5347 - categorical_accuracy: 0.4900 - val_loss: 1.0696 - val_categorical_accuracy: 0.6450
```

•

•

```
Epoch 89/100
200/200 [=====] - 16s 78ms/step - loss: 0.1604 - categorical_accuracy: 0.9490 - val_loss: 0.0070 - val_categorical_accuracy: 0.9000
Epoch 90/100
200/200 [=====] - 12s 58ms/step - loss: 0.1755 - categorical_accuracy: 0.9405 - val_loss: 0.1964 - val_categorical_accuracy: 0.9000
Epoch 91/100
200/200 [=====] - 15s 76ms/step - loss: 0.1758 - categorical_accuracy: 0.9415 - val_loss: 0.3885 - val_categorical_accuracy: 0.9200
Epoch 92/100
200/200 [=====] - 12s 61ms/step - loss: 0.1509 - categorical_accuracy: 0.9520 - val_loss: 0.3992 - val_categorical_accuracy: 0.8800
Epoch 93/100
200/200 [=====] - 11s 54ms/step - loss: 0.1520 - categorical_accuracy: 0.9520 - val_loss: 0.0137 - val_categorical_accuracy: 0.9100
Epoch 94/100
200/200 [=====] - 17s 86ms/step - loss: 0.1587 - categorical_accuracy: 0.9470 - val_loss: 0.1857 - val_categorical_accuracy: 0.9300
Epoch 95/100
200/200 [=====] - 12s 59ms/step - loss: 0.1494 - categorical_accuracy: 0.9525 - val_loss: 0.1477 - val_categorical_accuracy: 0.9100
Epoch 96/100
200/200 [=====] - 15s 76ms/step - loss: 0.1440 - categorical_accuracy: 0.9565 - val_loss: 0.0036 - val_categorical_accuracy: 0.9000
Epoch 97/100
200/200 [=====] - 12s 58ms/step - loss: 0.1397 - categorical_accuracy: 0.9560 - val_loss: 0.4621 - val_categorical_accuracy: 0.9250
Epoch 98/100
200/200 [=====] - 12s 59ms/step - loss: 0.1516 - categorical_accuracy: 0.9495 - val_loss: 0.0017 - val_categorical_accuracy: 0.9000
Epoch 99/100
200/200 [=====] - 13s 63ms/step - loss: 0.1566 - categorical_accuracy: 0.9445 - val_loss: 0.2868 - val_categorical_accuracy: 0.9150
Epoch 100/100
200/200 [=====] - 11s 54ms/step - loss: 0.1722 - categorical_accuracy: 0.9450 - val_loss: 0.0611 - val_categorical_accuracy: 0.9050
```


Training & Validation Accuracy

➤ Dataset Size

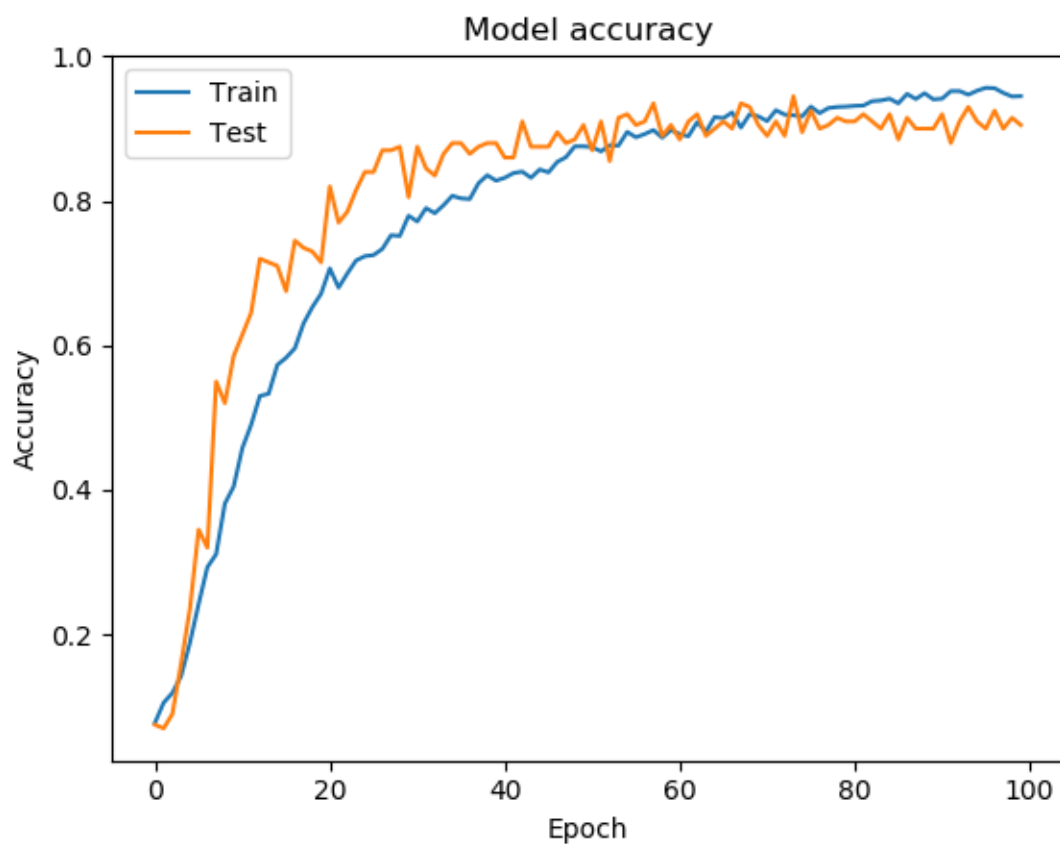
Training: 1120 images belonging to 16 classes

Validation: 160 images belonging to 16 classes

➤ Accuracy (after 100 epochs)

categorical_accuracy: 0.9450

val_categorical_accuracy: 0.9050



STEP 4. DEMO

Prediction Code

```
model = load_model('my_model.h5')
model.load_weights('first_try.h5')

img = input()
np.set_printoptions(precision=5, suppress=True)

cascade = cv2.CascadeClassifier('lbpcascade_animeface.xml')
imag = cv2.imread(img, cv2.IMREAD_COLOR)
gray = cv2.cvtColor(imag, cv2.COLOR_BGR2GRAY)
gray = cv2.equalizeHist(gray)

faces = cascade.detectMultiScale(gray, scaleFactor = 1.1, minNeighbors = 5, minSize = (24, 24))
i = 0
for (x, y, w, h) in faces:
    val = max(w,h)

    new_img = imag[y:y+val,x:x+val]
    new_img = cv2.resize(new_img, dim, interpolation = cv2.INTER_AREA)
    loc = img.split(".")[0]+"_"+str(x)+"_uwu"+".jpg"
    cv2.imwrite(loc,new_img)

    imm = image.load_img(loc, target_size = (150,150))
    file = ImageTk.PhotoImage(imm)
    canvas.create_image(900, 150*i + 100 + 30*i, anchor=NW, image=file)
    img_ref.append(file)
    text = prediction(loc)
    canvas.create_text(1175, 150*i + 190 + 12*i, font=("Purisa", rndfont2), text=text)
    print("\n\n",i,"\n\n")
    i = i+1

root.mainloop()
```

```
def prediction(Loc):
    print(Loc)
    imm = image.load_img(Loc, target_size = (150,150))
    img_pred = image.img_to_array(imm)
    img_pred = np.expand_dims(img_pred, axis=0)

    Data = img_pred.astype('float32')
    Data /=255

    text=""



    array = model.predict_proba(Data)
    result = array[0]

    xx = []
    for i in range(0,16,1):
        x = (result[i], i)
        xx.append(x)




    xx.sort(reverse = True)
    for i in range(0,3,1):
        text = text + classname(xx[i][1]) + " : " + str(round(xx[i][0]*100,2)) + "\n"

    return text
```






Example 1 - Single Character

INPUT	OUTPUT
	 Zero Two : 99.37 Darkness : 0.51 Genos : 0.06



Example 2 - Two Characters

INPUT	OUTPUT
	 Asuna : 99.95 Lucy : 0.04 Darkness : 0.01
	 Lucy : 83.66 Asuna : 16.32 Darkness : 0.01

Example 3 - Multiple Characters

INPUT	OUTPUT
	 <p>Aqua : 99.91 Rem : 0.09 Killua : 0.0</p>
	 <p>Megumin : 99.88 Zero Two : 0.11 Kaguya : 0.0</p>
	 <p>Darkness : 99.53 Zero Two : 0.44 Lucy : 0.01</p>
	 <p>Kazuma : 99.76 Megumin : 0.24 Midoriya : 0.0</p>

Example 4 - Hand-drawn Character

INPUT	OUTPUT
	 <p>Rem : 99.9 Killua : 0.06 Aqua : 0.03</p>

