"FACIAL EMOTION RECOGNITION USING NEURAL NETWORKS"

PROJECT REPORT

Submitted for the course:

MACHINE LEARNING (CSE 4020)

By

SANDEEP HARIKRISHNAN	16BCE0120
SHREYASH KAWALKAR	16BCE0221

SLOT: E1(L57+58)

Name of faculty: **PROF CHANDRA MOULLI P.V.S.S.R**

VIT®
UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)

CERTIFICATE

This is to certify that the project work entitled "**FACIAL EMOTION RECOGNITION USING NEURAL NETWORKS**" that is being submitted by "**Sandeep Harikrishnan**" and "**Shreyash Kawalkar**" for the course **MACHINE LEARNING (CSE 4020)** is a record of bonafide work done under my supervision. The contents of this project has been , in full or in parts, not been taken from any other CAL course.

ANBARASI M

Table of Contents

CHAPTER-1

ABSTRACT:

Human emotion recognition plays an important role in the interpersonal relationship. The automatic recognition of emotions has been an active research topic from early eras. Emotion recognition from facial images is a very active research topic in any field of computer science and discipline. However, most of the previous approaches only focus on the frontal or nearly frontal view facial images. In contrast to the frontal/nearly-frontal view images, emotion recognition from non-frontal view or even arbitrary view facial images is much more difficult yet of more practical utility. To handle the emotion recognition problem from arbitrary view facial images, in this report we propose a novel method based on the use of Convolution Neural Networks (CNNs) of facial images. Therefore, there are several advances made in this field. Emotions are reflected from gestures of the body and through facial expressions. Hence extracting and understanding of emotion has a high importance of the interaction between human and machine communication. This report also describes the advances made in this field and the various approaches used for recognition of emotions. The main objective is to propose real time implementation of emotion recognition system.

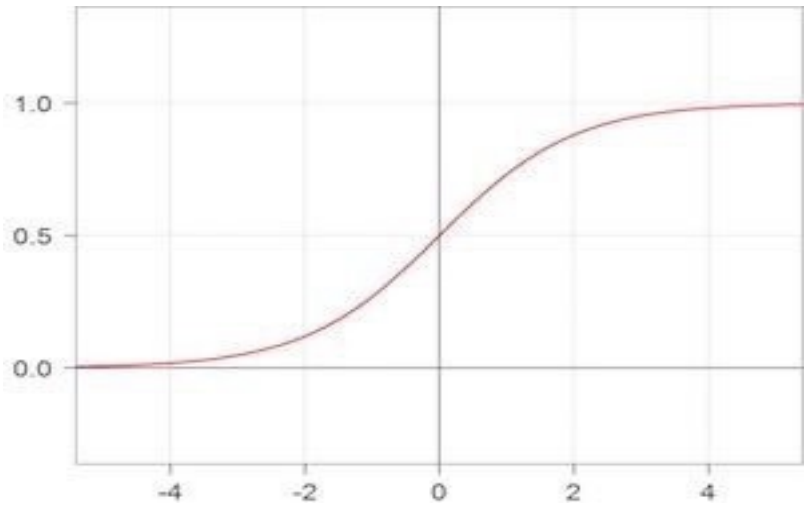
Deep learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks.

Neural networks are the building blocks of today's technological breakthrough in the field of Deep Learning. A neural network can be seen as simple processing unit that is massively parallel, capable to store knowledge and apply this knowledge to make predictions.

A neural network mimics the brain in a way the network acquires knowledge from its environment through a learning process. Then, intervention connection strengths known as synaptic weights are used to store the acquired knowledge. In the learning process, the synaptic weights of the network are modified in an orderly fashion to attain the desired objective.

$$y = \text{sigmoid}(w1*x1 + w2*x2 + w3*x3)$$

$$\text{sigmoid}(x) = 1 / (1 + \exp(-x))$$



CHAPTER-2

INTRODUCTION:

Human beings possess and express emotions in everyday interactions with others.

Emotions are often reflected on the face, in hand and body gestures, in the voice, to

express our feelings or liking.

Recent psychology research has shown that the most expressive way humans 6 display emotions is through facial expressions. Mehrabian [12] indicated that the verbal part of a message contributes only for 7% to the effect of the message as a whole, the vocal part (e.g. voice intonation) for 38%, while facial expressions contributes for 55% to the effect of the speaker's message. In addition to providing information about the affective state, facial expressions also provide information about cognitive state, such as interest, boredom, confusion, and stress, and conversational signals with information about speech emphasis and syntax.

While a precise, generally agreed definition of emotion does not exist, it is undeniable that emotions are an integral part of our existence, as one smiles to show greeting, frowns when confused, or raises one's voice when enraged. The fact that we understand emotions and know how to react to other people's expressions greatly enriches the interaction. There is a growing amount of evidence showing that emotional skills are part of what is called "intelligence". Computers today, on the other hand, are still quite "emotionally challenged." They neither recognize the user's emotions nor possess emotions of their own

An important problem in the emotion recognition field is the lack of agreed upon benchmark database and methods for compare different methods' performance.

There are several approaches taken in the literature for learning classifiers for emotion recognition

1. The static approach.

Here the classifier classifies each frame in the video to one of the facial expression categories based on the tracking results of that frame. Bayesian network classifiers were commonly used in this approach. While Naive-Bayes classifiers were often successful in practice, they use a very strict and often unrealistic assumption that the features are independent given the class. Therefore, another approach using Gaussian TAN classifiers have the advantage of modeling dependencies between the features without much added complexity compared to the Naive-Bayes classifiers. TAN classifiers have an additional advantage in that the dependencies between the features, modeled as a tree structure, are efficiently learned from data and the resultant tree structure is assured to maximize the likelihood function.

2. The dynamic approach.

These classifiers take into account the temporal pattern in displaying facial expression. Hidden Markov model (HMM) based classifiers for facial expression recognition has been previously used in recent works. Cohen and Sebe [1] further advanced this line of research and proposed a multi-level HMM classifier, combining the temporal information, which allowed not only to perform the classification of a video segment to the corresponding facial expression, as in the previous works on HMM based classifiers, but also to automatically segment an arbitrary long video 8 sequence to the different expressions segments without sorting to empirical methods of segmentation

CHAPTER-3

1. Convolutional Neural Networks (CNNs / ConvNets)

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

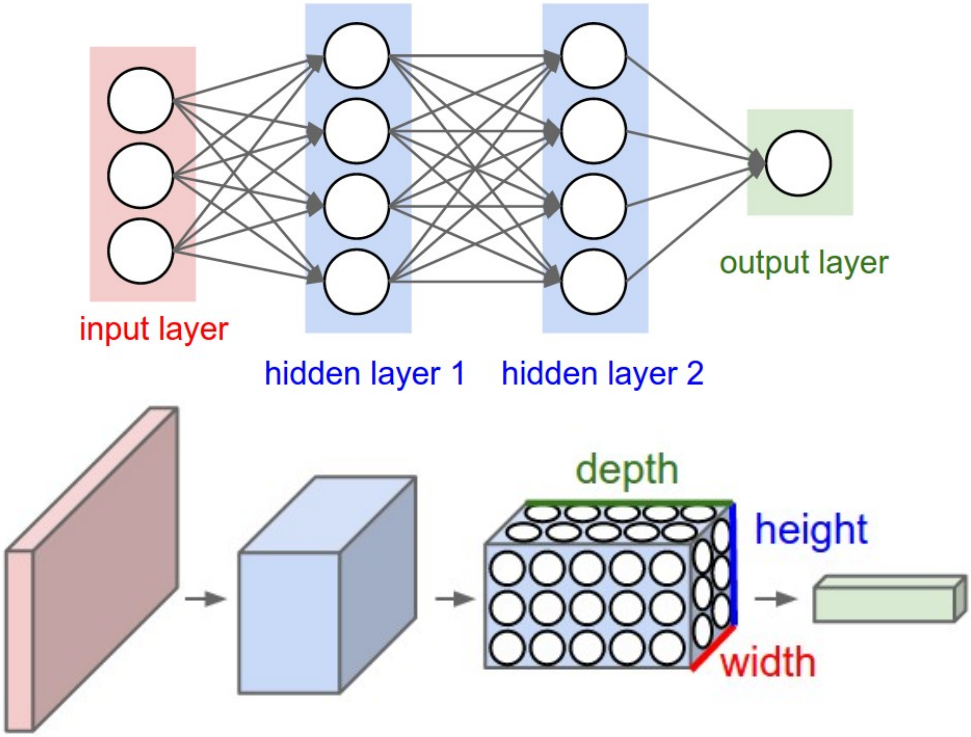
So what does change? ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

Architecture Overview

Recall: Regular Neural Nets. As we saw in the previous chapter, Neural Networks receive an input (a single vector), and transform it through a series of *hidden layers*. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the "output layer" and in classification settings it represents the class scores.

Regular Neural Nets don't scale well to full images. Images are only of size 32x32x3 (32 wide, 32 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights. Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

3D volumes of neurons. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: **width, height, depth**. (Note that the word *depth* here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) For example, the input images in CIFAR-10 are an input volume of activations, and the volume has dimensions 32x32x3 (width, height, depth respectively). As we will soon see, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions 1x1x10, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension. Here is a visualization:



A ConvNet is made up of Layers. Every Layer has a simple API: It transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters.

Layers used to build ConvNets

As we described above, a simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer** (exactly as seen in regular Neural Networks). We will stack these layers to form a full

In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

2.TENSORFLOW

TensorFlow™ is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

TensorFlow provides a Python API[17] as well as C++,[18] Haskell,[19] Java,[20] Go[21] and Rust[22] APIs. Third party packages are available for C#,[23] Julia,[24] R,[25] Scala[26] and OCaml.

3.KERAS

Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or MXNet.[1] Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System),[2] and its primary author and maintainer is François Chollet, a Google engineer.

In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Chollet explained that Keras was conceived to be an interface rather than a standalone machine-learning framework. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Microsoft added a CNTK backend to Keras as well.

4. NUMPY

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,[15] and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

5. PANDAS

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.[2] The name is derived from the term "panel data", an econometrics term for data sets that include both time-series and cross-sectional data.

The library features included in it are:

- DataFrame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and subsetting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation[4] and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.

6. OPEN CV

OpenCV (*Open Source Computer Vision*) is a library of programming functions mainly aimed at real-time computer vision.

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

- Boosting
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- k-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Support vector machine (SVM)
- Deep neural networks (DNN)

7. TARFILE:

In computing, **tar** is a computer software utility for collecting many files into one archive file, often referred to as a **tarball**, for distribution or backup purposes. The name was derived from *(t)ape (ar)chive*, as it was originally developed to write data to sequential I/O devices with no file system of their own. The archive data sets created by tar contain various file system parameters, such as name, time stamps, ownership, file access permissions, and directory organization.

8. CAMERA MODULE:

-- 30 FPS

-- 3 MPixels

9. CPU SYSTEM:

-- Processor 1.6 GHz base frequency.

-- 2 GB RAM

-- Quad Core

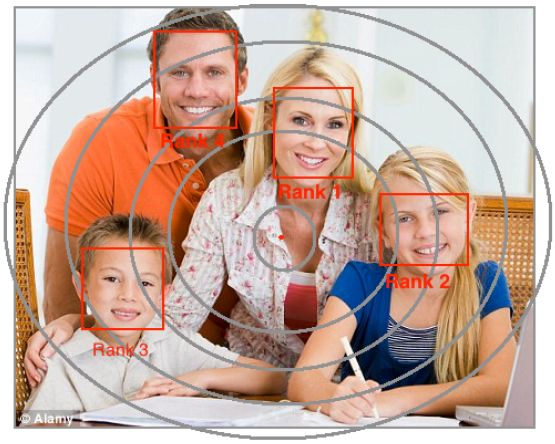
-- 32 GB Memory

Minimum of above should be met for trained model to be executed.

CHAPTER-4

Face Detection and Ranking:

The captured images using camera may contain one or more faces at a time. The whole frame is investigated using Cascades for face detection from front view to profile view. The face – rectangles then cropped are reshaped into 48 x 48 squares.



The face - rectangles are of different sizes and located at different location across the frame based upon the position of human in front of camera in 3D. These faces are assigned ranks as follows:

-- Ranking based upon size of face-rectangle (measured diagonally) in descending order i.e., larger rectangle is ranked prior to smaller ones. (seems to be near and more interactive)

-- Ranking based upon displacement form center. Person at center should be prioritized.

Preprocessing :

Pre - Processing:

The face frame of 48 X 48 is then converted into grayscale from RGB form. It is then normalized and edges are sharpened by tuning Opencv inbuilt function createCLAHE.

This is performed so that it reduces the runtime during skin texture detection, and positioning of facial landmarks.

The model shows no requirement of any heavy or additional preprocessing and face alignments, it works deals with the challenges more efficiently. (discussed in Masking). Preprocessing: original image-->gray image-->clahe image

Algorithm:

- The data is fed into training_pixels, training_labels, testing_pixels, testing_labels, respectively.
- The original network starts with an input layer of 48 by 48, matching the size of the input data.
- Then the processing starts with 2 layered convolutions followed by an intermediate maxpolling and dropout
- Further in network it undergoes other 2 layered convolutions followed by an intermediate maxpolling and dropout
- Finally in the network it is flattened then densed and dropout is executed.
- Further the data array undergoes final dense, activated by Softmax and then compiled
- The network is further validated with test data for 16 epoches.
- Check points for best results are committed in chkPt1.h5

CHAPTER-5

CODE:

In [1]:
```
#dependencies
import numpy as np
import cv2
import pandas as pd
import matplotlib.pyplot as plt
from keras.utils import np_utils
from keras.models import Sequential, Model, model_from_json
from keras.layers import Conv2D, Activation, MaxPool2D, Dropout, Dense, BatchNormalization, Flatten
from keras.callbacks import ModelCheckpoint
import tarfile
Using TensorFlow backend.
```

```
/Users/shreyashkawalkar/anaconda3/lib/python3.6/importlib/_bootstrap.py:205: RuntimeWarning: compiletime version 3.5
of module 'tensorflow.python.framework.fast_tensor_util' does not match runtime version 3.6
  return f(*args, **kwds)
```

In [3]:

```
#dataset
data_comp = tarfile.open("fer2013.tar")
ds = pd.read_csv(data_comp.extractfile("fer2013/fer2013.csv"))
ds.head()
#0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral
```

Out[3]:

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

In [4]:

```
ds["Usage"].value_counts()
```

Out[4]:

```
Training       28709
PrivateTest     3589
PublicTest      3589
Name: Usage, dtype: int64
```

In [5]:

```
train = ds[["emotion", "pixels"]][ds["Usage"] == "Training"]
train['pixels'] = train['pixels'].apply(lambda x: np.fromstring(x, sep=' '))
train_pix = np.vstack(train['pixels'].values)
test = ds[["emotion", "pixels"]][ds["Usage"] == "PublicTest"]
test['pixels'] = test['pixels'].apply(lambda x: np.fromstring(x, sep=' '))
test_pix = np.vstack(test['pixels'].values)
train_pix.shape, test_pix.shape
```

Out[5]:

```
((28709, 2304), (3589, 2304))
```

In [6]:

```
#0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral
#0=angry, 1=disgust+fear+surprise, 3=Happy, 4=sad, 6= neutral
#apending 2 and 5 to 1; 5 to 2; 6 to 4
#::: set append = false to avoid changes
def e_ind(x):
    if(x==2 or x==5):
        return 1
    elif(x==5):
        return 2
    elif(x==6):
        return 4
    else:
        return x
F_S_D = True
if(F_S_D):
    train['emotion'] = train['emotion'].apply(lambda x: e_ind(x))
    test['emotion'] = test['emotion'].apply(lambda x: e_ind(x))
train_ind = np.array(train["emotion"])
test_ind = np.array(test["emotion"])
```

```
train_ind.shape, test_ind.shape
```

Out[6]:

```
((28709,), (3589,))
```

In [7]:

```
train_pix = train_pix.reshape(-1,48,48,1)
train_ind = np_utils.to_categorical(train_ind)
test_pix = test_pix.reshape(-1,48,48,1)
test_ind = np_utils.to_categorical(test_ind)
train_ind.shape, test_ind.shape
#train_pix.shape, test_pix.shape
```

Out[7]:

```
((28709, 5), (3589, 5))
```

In [17]:

```
model = Sequential()
model.add(Conv2D(64, 5, data_format="channels_last", kernel_initializer="he_normal",
                 input_shape=(48, 48, 1)))
model.add(BatchNormalization())
model.add(Activation("relu"))

model.add(Conv2D(64, 4))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))
model.add(Dropout(0.5))

model.add(Conv2D(32, 3))
model.add(BatchNormalization())
model.add(Activation("relu"))

model.add(Conv2D(32, 3))
model.add(BatchNormalization())
model.add(Activation("relu"))

model.add(MaxPool2D(pool_size=(2, 2), strides=2))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(0.2))

if(F_S_D):
   model.add(Dense(5))
else:
   model.add(Dense(7))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_29 (Conv2D)           (None, 44, 44, 64)        1664
_________________________________________________________________
batch_normalization_22 (Batc (None, 44, 44, 64)        256
_________________________________________________________________
activation_43 (Activation)   (None, 44, 44, 64)        0
_________________________________________________________________
```

```
conv2d_30 (Conv2D)           (None, 41, 41, 64)        65600
_________________________________________________________________
batch_normalization_23 (Batc (None, 41, 41, 64)        256
_________________________________________________________________
activation_44 (Activation)   (None, 41, 41, 64)        0
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 20, 20, 64)        0
_________________________________________________________________
dropout_13 (Dropout)         (None, 20, 20, 64)        0
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 18, 18, 32)        18464
_________________________________________________________________
batch_normalization_24 (Batc (None, 18, 18, 32)        128
_________________________________________________________________
activation_45 (Activation)   (None, 18, 18, 32)        0
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 16, 16, 32)        9248
_________________________________________________________________
batch_normalization_25 (Batc (None, 16, 16, 32)        128
_________________________________________________________________
activation_46 (Activation)   (None, 16, 16, 32)        0
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 8, 8, 32)          0
_________________________________________________________________
dropout_14 (Dropout)         (None, 8, 8, 32)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               262272
_________________________________________________________________
batch_normalization_26 (Batc (None, 128)               512
_________________________________________________________________
activation_47 (Activation)   (None, 128)               0
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_16 (Dense)             (None, 5)                 645
_________________________________________________________________
activation_48 (Activation)   (None, 5)                 0
=================================================================
Total params: 359,173
Trainable params: 358,533
Non-trainable params: 640
_________________________________________________________________
```

In [18]:

```
# checkpoint
checkPoint = ModelCheckpoint(filepath='chkPt1.h5', verbose=1, save_best_only=True)

res = model.fit(train_pix, train_ind, epochs=16,
                shuffle=True,
                batch_size=100,
                validation_data=(test_pix, test_ind),
                callbacks=[checkPoint],
                verbose=2)

# save model to json
model_json = model.to_json()
with open("model1.json", "w") as json_file:
```

```
    json_file.write(model_json)
Train on 28709 samples, validate on 3589 samples
Epoch 1/16

Epoch 00001: val_loss improved from inf to 1.18156, saving model to chkPt1.h5
 - 1189s - loss: 1.3142 - acc: 0.4424 - val_loss: 1.1816 - val_acc: 0.4684
Epoch 2/16

Epoch 00002: val_loss improved from 1.18156 to 1.02509, saving model to chkPt1.h5
 - 1089s - loss: 1.1040 - acc: 0.5325 - val_loss: 1.0251 - val_acc: 0.5837
Epoch 3/16

Epoch 00003: val_loss improved from 1.02509 to 0.95903, saving model to chkPt1.h5
 - 1154s - loss: 1.0263 - acc: 0.5712 - val_loss: 0.9590 - val_acc: 0.6144
Epoch 4/16

Epoch 00004: val_loss did not improve
 - 1119s - loss: 0.9729 - acc: 0.5935 - val_loss: 0.9751 - val_acc: 0.5979
Epoch 5/16

Epoch 00005: val_loss improved from 0.95903 to 0.90675, saving model to chkPt1.h5
 - 2065s - loss: 0.9350 - acc: 0.6130 - val_loss: 0.9067 - val_acc: 0.6364
Epoch 6/16

Epoch 00006: val_loss improved from 0.90675 to 0.89092, saving model to chkPt1.h5
 - 1171s - loss: 0.9102 - acc: 0.6239 - val_loss: 0.8909 - val_acc: 0.6350
Epoch 7/16

Epoch 00007: val_loss improved from 0.89092 to 0.87333, saving model to chkPt1.h5
 - 3102s - loss: 0.8864 - acc: 0.6386 - val_loss: 0.8733 - val_acc: 0.6478
Epoch 8/16

Epoch 00008: val_loss improved from 0.87333 to 0.86625, saving model to chkPt1.h5
 - 1521s - loss: 0.8641 - acc: 0.6496 - val_loss: 0.8663 - val_acc: 0.6542
Epoch 9/16

Epoch 00009: val_loss improved from 0.86625 to 0.84629, saving model to chkPt1.h5
 - 6372s - loss: 0.8430 - acc: 0.6570 - val_loss: 0.8463 - val_acc: 0.6573
Epoch 10/16

Epoch 00010: val_loss did not improve
 - 1214s - loss: 0.8320 - acc: 0.6627 - val_loss: 0.8478 - val_acc: 0.6567
Epoch 11/16

Epoch 00011: val_loss improved from 0.84629 to 0.82408, saving model to chkPt1.h5
 - 4998s - loss: 0.8197 - acc: 0.6693 - val_loss: 0.8241 - val_acc: 0.6734
Epoch 12/16

Epoch 00012: val_loss improved from 0.82408 to 0.82149, saving model to chkPt1.h5
 - 1139s - loss: 0.8007 - acc: 0.6771 - val_loss: 0.8215 - val_acc: 0.6693
Epoch 13/16

Epoch 00013: val_loss improved from 0.82149 to 0.81216, saving model to chkPt1.h5
 - 1898s - loss: 0.7862 - acc: 0.6840 - val_loss: 0.8122 - val_acc: 0.6718
Epoch 14/16

Epoch 00014: val_loss improved from 0.81216 to 0.80925, saving model to chkPt1.h5
 - 1082s - loss: 0.7794 - acc: 0.6882 - val_loss: 0.8093 - val_acc: 0.6732
```

```
Epoch 15/16

Epoch 00015: val_loss improved from 0.80925 to 0.80548, saving model to chkPt1.h5
 - 1124s - loss: 0.7643 - acc: 0.6988 - val_loss: 0.8055 - val_acc: 0.6737
Epoch 16/16

Epoch 00016: val_loss improved from 0.80548 to 0.79962, saving model to chkPt1.h5
 - 1131s - loss: 0.7517 - acc: 0.7006 - val_loss: 0.7996 - val_acc: 0.6868
```

In [19]:

```
import seaborn as sns
plt.figure(figsize=(14,3))
plt.subplot(1, 2, 1)
plt.suptitle('Optimizer : Adam', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(res.history['loss'], color='b', label='Training Loss')
plt.plot(res.history['val_loss'], color='r', label='Validation Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(res.history['acc'], color='b', label='Training Accuracy')
plt.plot(res.history['val_acc'], color='r', label='Validation Accuracy')
plt.legend(loc='lower right')
plt.show()
```

In [ ]:

In [20]:

```
score = model.evaluate(test_pix, test_ind, verbose=0)
score
```

Out[20]:

```
[0.79961926936605043, 0.68682084146001676]
```

In [3]:

```
def simple_CNN(input_shape, num_classes):

    model = Sequential()
    model.add(Convolution2D(filters=16, kernel_size=(7, 7), padding='same',
                            name='image_array', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=16, kernel_size=(7, 7), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=32, kernel_size=(5, 5), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=32, kernel_size=(5, 5), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=64, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=64, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
```

```
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=128, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=128, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.5))

    model.add(Convolution2D(filters=256, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=num_classes, kernel_size=(3, 3), padding='same'))
    model.add(GlobalAveragePooling2D())
    model.add(Activation('softmax',name='predictions'))
    return model
```

In [1]:

```
from keras.layers import Activation, Convolution2D, Dropout, Conv2D
from keras.layers import AveragePooling2D, BatchNormalization
from keras.layers import GlobalAveragePooling2D
from keras.models import Sequential
from keras.layers import Flatten
from keras.models import Model
from keras.layers import Input
from keras.layers import MaxPooling2D
from keras.layers import SeparableConv2D
from keras.layers import Conv2D, Activation, MaxPool2D, Dropout, Dense, BatchNormalization, Flatten
from keras import layers
from keras.regularizers import l2
import keras
Using TensorFlow backend.
/Users/shreyashkawalkar/anaconda3/lib/python3.6/importlib/_bootstrap.py:205: RuntimeWarning: compiletime version 3.5
of module 'tensorflow.python.framework.fast_tensor_util' does not match runtime version 3.6
  return f(*args, **kwds)
```

In [2]:

```
input_shape = (48, 48, 1)
num_classes = 7
```

In [3]:

```
def simple_CNN(input_shape, num_classes):

    model = Sequential()
    model.add(Conv2D(64, (5, 5), activation='relu', input_shape = input_shape))
    model.add(keras.layers.convolutional.ZeroPadding2D(padding=(2, 2)))
    model.add(keras.layers.pooling.MaxPooling2D(pool_size=(5, 5),strides=(2, 2)))

    model.add(keras.layers.convolutional.ZeroPadding2D(padding=(1, 1)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(keras.layers.convolutional.ZeroPadding2D(padding=(1, 1)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(keras.layers.pooling.AveragePooling2D(pool_size=(3, 3),strides=(2, 2)))

    model.add(keras.layers.convolutional.ZeroPadding2D(padding=(1, 1)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(keras.layers.convolutional.ZeroPadding2D(padding=(1, 1)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(keras.layers.convolutional.ZeroPadding2D(padding=(1, 1)))
```

```
    model.add(keras.layers.pooling.AveragePooling2D(pool_size=(3, 3),strides=(2, 2)))

    model.add(Flatten())

    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.2))

    model.add(Dense(num_classes, activation='softmax'))

    #sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(loss='categorical_crossentropy', optimizer='adadelta', metrics=['accuracy'])
    return model
```

In [4]:

```
model = simple_CNN((48, 48, 1), num_classes)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 44, 44, 64)        1664
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 48, 48, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 22, 22, 64)        36928
_________________________________________________________________
zero_padding2d_3 (ZeroPaddin (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 22, 64)        36928
_________________________________________________________________
average_pooling2d_1 (Average (None, 10, 10, 64)        0
_________________________________________________________________
zero_padding2d_4 (ZeroPaddin (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 128)       73856
_________________________________________________________________
zero_padding2d_5 (ZeroPaddin (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 128)       147584
_________________________________________________________________
zero_padding2d_6 (ZeroPaddin (None, 12, 12, 128)       0
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              3277824
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
```

```
dense_3 (Dense)              (None, 7)                 7175
=================================================================
Total params: 4,631,559
Trainable params: 4,631,559
Non-trainable params: 0
_________________________________________________________________
```

In [5]:

```
model.load_weights("chkPt1.hdf5")
model_json = model.to_json()
with open("model1.json", "w") as json_file:
    json_file.write(model_json)
```

In [6]:

```
from keras.models import model_from_json
import numpy as np

class FacialExpressionModel(object):
    EMOTIONS_LIST = ["ANGRY", "DISGUST", "FEAR", "HAPPY", "SAD", "SURPRISE", "NEUTRAL"];
    def __init__(self, model_json_file, model_weights_file):
        # load model from JSON file
        with open(model_json_file, "r") as json_file:
            loaded_model_json = json_file.read()
            self.loaded_model = model_from_json(loaded_model_json)

        # load weights into the new model
        self.loaded_model.load_weights(model_weights_file)
        print("Model loaded from disk")
        self.loaded_model.summary()

    def predict_emotion(self, img):
        self.preds = self.loaded_model.predict(img)

        return FacialExpressionModel.EMOTIONS_LIST[np.argmax(self.preds)]


if __name__ == '__main__':
    pass
```

In [ ]:

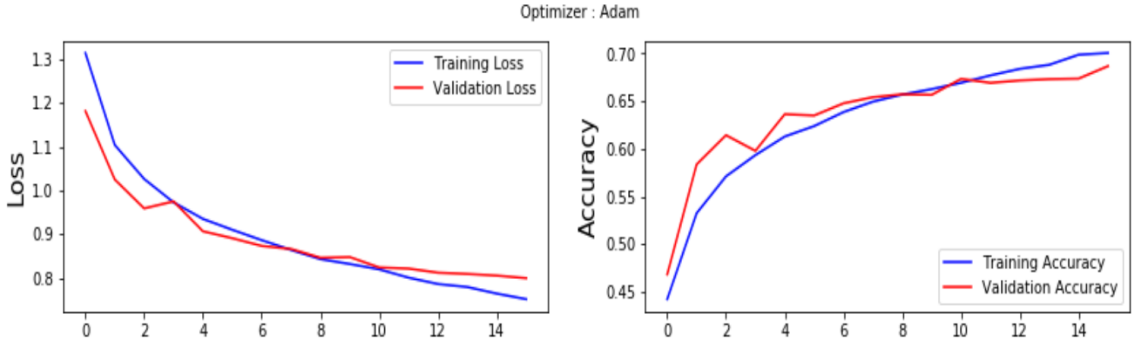
```
import cv2
import numpy as np

rgb = cv2.VideoCapture(0)
facec = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
font = cv2.FONT_HERSHEY_SIMPLEX

def __get_data__():
    _, fr = rgb.read()
    gray = cv2.cvtColor(fr, cv2.COLOR_BGR2GRAY)
    faces = facec.detectMultiScale(gray, 1.3, 5)
    return faces, fr, gray

def start_app(cnn):
    skip_frame = 10
    data = []
    flag = False
    ix = 0
    while True:
        ix += 1
```

```
        faces, fr, gray_fr = __get_data__()
        for (x, y, w, h) in faces:
            fc = gray_fr[y:y+h, x:x+w]

            roi = cv2.resize(fc, (48, 48))
            pred = cnn.predict_emotion(roi[np.newaxis, :, :, np.newaxis])

            cv2.putText(fr, pred, (x, y), font, 1, (255, 255, 0), 2)
            cv2.rectangle(fr,(x,y),(x+w,y+h),(255,0,0),2)

        if cv2.waitKey(1) == 27:
            break
        cv2.imshow('Filter', fr)
    cv2.destroyAllWindows()


if __name__ == '__main__':
    model = FacialExpressionModel("model1.json", "chkPt1.hdf5")
    start_app(model)
Model loaded from disk
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 44, 44, 64)        1664
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 48, 48, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 22, 22, 64)        36928
_________________________________________________________________
zero_padding2d_3 (ZeroPaddin (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 22, 64)        36928
_________________________________________________________________
average_pooling2d_1 (Average (None, 10, 10, 64)        0
_________________________________________________________________
zero_padding2d_4 (ZeroPaddin (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 128)       73856
_________________________________________________________________
zero_padding2d_5 (ZeroPaddin (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 128)       147584
_________________________________________________________________
zero_padding2d_6 (ZeroPaddin (None, 12, 12, 128)       0
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              3277824
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
```

```
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 7175
=================================================================
Total params: 4,631,559
Trainable params: 4,631,559
Non-trainable params: 0
_________________________________________________________________
```

OUTPUT:

We got an accuracy score as 68.68% after 16 epoches.



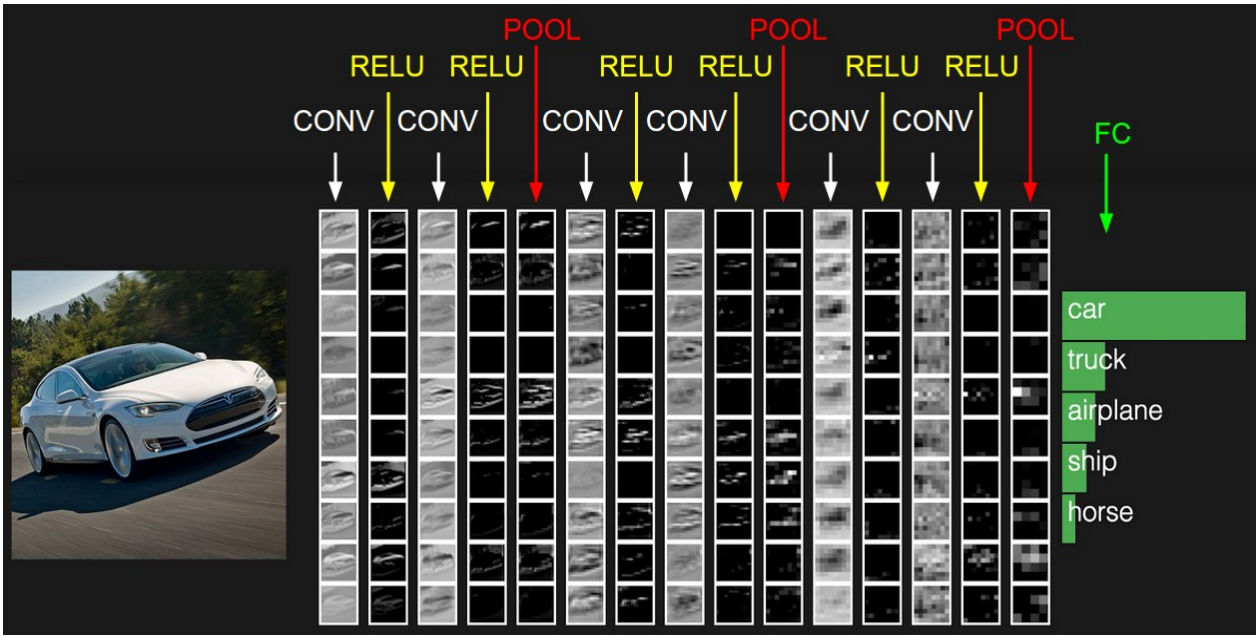
CHAPTER-6

CONCLUSION:

In this project report, we saw how convolution neural network can help us with identifying the facial emotion one possesses.

In summary:

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't)
- Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't)



The activations of an example ConvNet architecture. The initial volume stores the raw image pixels (left) and the last volume stores the class scores (right). Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores, and print the labels of each one. The full web-based demo is shown in the header of our website. The architecture shown here is a tiny VGG Net, which we will discuss later.

We now describe the individual layers and the details of their hyperparameters and their connectivities.

REFERENCES:

1. https://www.kaggle.com/c/challenges-in-representation-learning-facial- expression-recognition-challenge.

2. https://uu.diva-portal.org/smash/get/diva2:952138/FULLTEXT01.pdf

3. https://github.com/aymericdamien/TensorFlow-Examples

4. https://github.com/lazyprogrammer/machine_learning_examples

5. https://www.freetutorials.us/data-science-deep-learning-in-python-1/

6. Smith, R.: The 7 Levels of Change, 2nd edn. Tapeslry Press (2002)

7. Yang, X.: Nature-Inspired Metaheuristic Algorithms. Luniver Press (2008)

8. W. N. Chen, J. Zhang, H. Chung, W. L. Zhong, W. G. Wu, and Y. H. Shi, "A novel set-based particle swarm optimization method for discrete optimization problems," IEEE Trans. Evol. Comput., vol. 14, no. 2, pp. 278-300, April 2010.

9. Y. Shi, "Brain storm optimization algorithm using CNN," in Proc. 2nd Int. Conf. on Swarm Intelligence, 2011, pp. 303-309.

10. F. Peng, K. Tang, G. L. Chen, and X. Yao, "Numpy for numerical optimization," IEEE Trans. Evol. Comput., vol. 14, no. 5, pp. 782-800, Oct. 2010.