Face Recognition and Emotion identification

A Project

Presented to

Department of Computer Science

State University of New York Polytechnic Institute

At Utica/Rome

Utica, New York

In partial fulfillment

Of the requirements of the

Master of Science Degree

By

Ms. Akshara Avadhut Lagwankar

May 2021

DECLARATION

I declare that this project is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Ms. Akshara Avadhut Lagwankar

SUNY POLYTECHNIC INSTITUTE

DEPARTMENT OF COMPUTER SCIENCE

Approved and recommended for acceptance as a project in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Ms. Akshara Avadhut Lagwankar.

Date: May 13, 2021.

_____

Dr. Bruno R. Andriamanalimanana

Advisor

_____

Dr. Scott Spetka

_____

Dr. Chen-Fu Chiang

ABSTRACT

While face recognition has been around in one form or another since the 1960s, recent technological developments have led to a wide proliferation of this technology. This technology is no longer seen as something out of science fiction movies like Minority Report. With the release of the iPhone X, millions of people now literally have face recognition technology in the palms of their hands, protecting their data and personal information. While mobile phone access control might be the most recognizable way face recognition is being used, it is being employed for a wide range of use cases including preventing crime, protecting events and making air travel more convenient.

This project focuses on various advanced Python libraries to improve the face recognition accuracy such as OpenCV, Sklearn, face_recognition. The project understands the data and model, train it for further usage. The real time videos are considered for evaluating the results. Further the project glances the emotion recognition algorithms using CV2, Seaborn. The areas of the human faces are highlighted according to different emotions. The large data sets (fer2013, Olivetti faces) are used for training and testing the data sets. PCA, leave one out cross validation, grid search CV, machine learning pipelines, CNN models are used to estimate and increase the accuracy. The project is executed in Anaconda environment Jupyter Notebook. As the data sets are huge Google Collaboratory is used for execution.

ACKNOWLEDGEMENTS

I take this opportunity with much pleasure to thank all the people who have helped me through the course of our journey towards this report.

I sincerely thank my project advisor Dr. Bruno Andriamanalimanana for his guidance, help and motivation. Apart from the subject of my project, I learned a lot from him, I am sure it will be useful to me in different stages of my life. I would like to express my gratitude to him for providing necessary information regarding the project and for his constant guidance, supervision, kind co-operation and invaluable support in all aspects.

I would like to express my sincere thanks to Dr. Scott Spetka and Dr. Chen-fu Chiang for being on the project review committee. I have always enjoyed your classes.

Finally, I would like to express my heartfelt thanks to the God, my parents and colleagues for their blessings and constant support.

Ms. Akshara Avadhut Lagwankar.

Table of Contents

1.Introduction

Project focuses on various advanced Python libraries to improve the face recognition accuracy such as OpenCV, Sklearn, face_recognition. The real time videos are considered for evaluating the results. Further the project glances the emotion recognition algorithms using CV2, Seaborn. The areas of the human faces are highlighted according to different emotions. PCA, leave one out cross validation, grid search CV, machine learning pipelines, CNN models are used to estimate and increase the accuracy.

The references use different types of models under different circumstances. I have tried to collaborate different approaches from models in attempt to increase the accuracy. The weights and biases are adjusted to increase the learning. The PCA model implementation, the parameters passed to face recognition function for directories try to highlight the increased accuracy. Besides the training and testing the database, the attempt is also made to lively capture the face through web camera. The face is identified through web camera with 0.6-0.7 accuracy. There is scope to increase the speed and performance too.

1.1 Computer Vision

First, to understand the project we must understand computer vision. Face tracking and detection features in sequence is an important and fundamental problem in computer vision. This area of research has a lot of applications in face identification systems, model-based coding, gaze detection, human computer, interaction, teleconferencing, etc. human-computer interaction, teleconferencing, etc. [12]

1.2 Open CV

OpenCV means Intel® Open-Source Computer Vision Library. It is a collection of C functions and a few C++ classes that implement some popular image processing and computer vision algorithms. OpenCV has cross-platform middle-to-high level API that consists of a few hundred C functions. It does not rely on external libraries, though it can use some when it is possible. OpenCV is free for both non-commercial and commercial use. OpenCV provides transparent interface to Intel Integrated

Performance Primitives (IPP). That is, it loads automatically IPP libraries optimized for specific processor at runtime, if they are available. [12]

1.2.1 What is new in Open CV?

- (Linux) Support for more cameras has been added in highgui. Different versions of libdc1394 can now be used. (thanks to Frederic Devernay for the new versions of cvcap_dc1394.cpp, cvcap_v4l.cpp and patches for configure script, thanks to Sfuncia Fabio for the patch for cvcap_v4l.cpp)
- (Linux) More types of video files can now be read by using libavformat and libavcodec from ffmpeg-0.4.9pre1 (thanks to Frederic Devernay for new version of cvcap.cpp and patches for configure script)
- (Linux) Python wrappers for OpenCV have been created by Olivier Bornet and Mark Asbach using SWIG. See OpenCV/interfaces/Python and OpenCV/samples/Python. While the wrappers should be OS-independent, so far, they have been built on Linux only.
- OpenCV now builds and runs on 64-bit platforms: EM64T (a.k.a. AMD64) and IA64 (Itanium). Extra configurations have been added to project files for MsDevStudio 6.0.
- • Performance tests for cxcore and part of cv have been created. the output format is plain csv and is like the one used in IPP. run "cxcoretest -t" and "cvtest -t".
- □ Script for creating custom dynamic library for a subset of IPP, used by OpenCV, has been created. Look at OpenCV/interfaces/ipp.
- New checkerboard detection algorithm (based on Vladimir Vezhnevets' code) is now used.

1.3 Image Processing

Some of the many algorithms used in image processing include convolution (on which many others are based), FFT, DCT, thinning (or skeletonization), edge detection and contrast enhancement. These are usually implemented in software but may also use special purpose hardware for

speed. Image processing contrasts with computer graphics, which is usually more concerned with the generation of artificial images, and visualization, which attempts to understand (real world) data by displaying it as an artificial image (e.g., a graph). Image processing is used in image recognition and computer vision. Silicon Graphics manufacture workstations which are often used for image processing. There are a few programming languages designed for image processing, e.g., CELIP, VPL, C++.

1.4 Face Tracking

Face detection and tracking are important in video content analysis since the most important objects in most video are human beings. Research on face tracking and animation techniques has been improved due to its wide range of applications in security, entertainment industry, gaming, psychological facial expression analysis and human computer interaction. Recent advances in face video processing and compression have made face-to-face communication practical in real world applications. However, higher bandwidth is still highly demanded due to the increasing intensive communication. Model based low bit rate transmission with high quality video offers a great potential to mitigate the problem raised by limited communication resources. However, after a decade's effort, robust and realistic real time face tracking and generation still pose a big challenge. The difficulty lies in several issues including the real time face feature tracking under a variety of imaging conditions such as lighting variation, pose change, self-occlusion and multiple non-rigid features deformation and the real time realistic face modeling using an extremely limited number of feature parameters. Traditionally, the head motion is modeled as a 3D rigid motion with the local skin deformation, the linear motion tracking method cannot represent the rapid head motion and dramatic expression change accurately.

The appearance-driven approach requires a significant number of training data to enumerate all the possible appearances of features. The model-based approach assumes the knowledge of a specific object is available, meanwhile the requirement of frontal facial views and constant illumination limited its application. All above tracking methods have shown certain limitations for accurate face feature tracking under complex imaging conditions. Different types of facial features, like skin color,

edges, feature points, motion, have been used for face tracking. Skin color is tried for tracking face motion in X, Y direction and out-of-plane rotation in. It is often too simple to encode structural knowledge of face, it is thus good for coarse face tracking. An optical flow field has been adopted for face tracking. Dense motion information makes face tracking easier. A major constraint is that optical flow estimation is subject to the aperture effect and usually does not allow big movement. Salient facial feature points are better choice for accurate face tracking. The main shortcoming is that tracking of point features is easily impaired by noise and often the face appearing in video should be large enough to facilitate tracking. Face tracking can serve as a front end to further analysis modules, such as face recognition, face expression analysis, gaze tracking, and lip reading. Face tracking is also a core component to enable the computer to see the computer user in a Human Computer Interface system.[12]

2 Background

The first study on automatic facial recognition systems was performed by Bledsoe between 1964 and 1966. This study was semi-automatic. The feature points on the face are determined manually and placed in the table called RAND. Then, a computer would perform the recognition process by classifying these points. However, a fully functional facial recognition application was performed in 1977 by Kanade. A feature-based approach was proposed in the study. After this date, two-dimensional (2D) face recognition has been studied intensively. Three-dimensional (3D) face studies were started to be made after the 2000s.

3D facial recognition approaches developed in different way than 2D facial recognition approaches. Therefore, it will be more accurate to categorize in 2D and 3D when discussing face recognition approaches.

We can classify the face recognition research carried out with 2D approach in three categories: analytical (feature-based, local), global (appearance) and hybrid methods. While analytical approaches want to recognize by comparing the properties of the facial components, global approaches try to achieve a recognition with data derived from all the face. Hybrid approaches, together with local and global approaches, try to obtain data that expresses the face more accurately.Face recognition performed in this kernel can assessed under global face recognition approaches.

In analytical approaches, the distance of the determined feature points and the angles between them, the shape of the facial features or the variables containing the regional features are obtained from the face image are used in face recognition. Analytical methods examine the face images in two different ways according to the pattern and geometrical properties. In these methods, the face image is represented by smaller size data, so the big data size problem that increases the computation cost in face recognition is solved.

Global-based methods are applied to face recognition by researchers because they perform facial recognition without feature extraction which is troublesome in feature-based methods. Globally based methods have been used in face recognition since the 1990s, since they significantly improve facial recognition efficiency. Kirby and Sirovich (1990) first developed a method known as Eigenface, which is used in facial representation and recognition based on Principal Component Analysis. With this method, Turk and Pentland transformed the entire face image into vectors and computed eigenfaces with a set of samples. PCA was able to obtain data representing.

the face at the optimum level with the data obtained from the image. The different facial and illumination levels of the same person were evaluated as the weakness point of PCA.

2.1 Principal Component Analysis

Machine learning methods are divided into two: supervised learning and unsupervised learning. In supervised learning, the data set is divided into two main parts: 'data' and 'output'. The data holds the values of the sample in the data set, while the 'output' holds the class (for classification) or the target value (for regression). In unsupervised learning, the data set consists of only the data section.

Non-supervised learning is generally divided into two: data transformation and clustering. In this study, the transformation of the data will be carried out using unsupervised learning. Unsupervised transformation methods allow for easier interpretation of data by computers and people.

The most common unsupervised transformation applications are to reduce data size. In the size reduction process, the dimension of the data are reduced.

Principal Component Analysis (PCA) is a method that allows data to be represented in a lesser size. According to this method, the data is transformed to new components and the size of the data is reduced by selecting the most important components.

Large datasets are increasingly common and are often difficult to interpret. Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance. Finding such new variables, the principal components, reduces to solving an eigenvalue/eigenvector problem, and the new variables are defined by the dataset at hand, not *a priori*, hence making PCA an adaptive data analysis technique. It is adaptive in another sense too, since variants of the technique have been developed that are tailored to various data types and structures.[14]

2.2 Leave One Out Cross Validation

Cross validation is a model evaluation method that is better than residuals. The problem with residual evaluations is that they do not give an indication of how well.

the learner will do when it is asked to make new predictions for data it has not already seen. One way to overcome this problem is to not use the entire data set when training a learner. Some of the data is removed before training begins. Then when training is done, the data that was removed can be used to test the performance of the learned model on ``new'' data. This is the basic idea for a whole class of model evaluation methods called *cross validation*.

Leave-one-out cross validation is K-fold cross validation taken to its logical extreme, with K equal to N, the number of data points in the set. That means that N separate times, the function approximator is trained on all the data except for one point and a prediction is made for that point. As before the average error is computed and used to evaluate the model. The evaluation given by leave-one-out cross validation error (LOO-XVE) is good, but at first pass it seems awfully expensive to compute. Fortunately, locally weighted learners can make LOO predictions just as easily as they make regular predictions. That means computing the LOO-XVE takes no more time than computing the residual error and it is a much better way to evaluate models.[15]

2.3 Machine Learning Automated Workflow: Pipeline

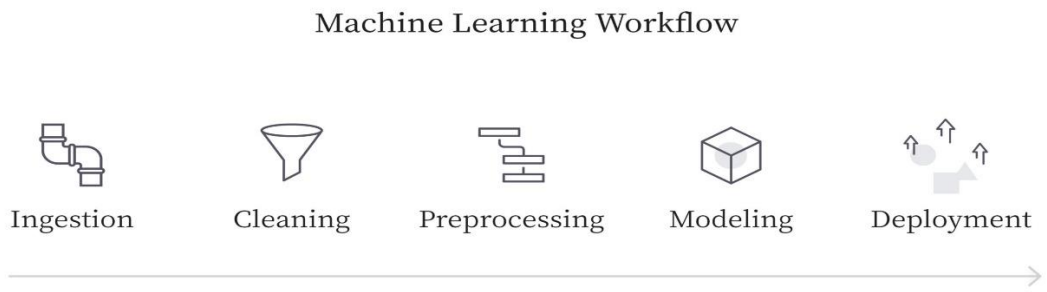
One definition of an ML pipeline is a means of automating the machine learning workflow by enabling data to be transformed and correlated into a model that can then be analyzed to achieve outputs. This type of ML pipeline makes the process of inputting data into the ML model fully automated.

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

This goes together with the recent push for microservices architectures, branching off the main idea by splitting your application into basic and siloed parts you can build more powerful software over time. Operating systems like Linux and Unix are also founded on this principle. Basic functions like ‘grep’ and ‘cat’ can create impressive functions when they are pipelined together.

With the ML pipeline, each part of your workflow is abstracted into an independent service. Then, each time you design a new workflow, you can pick and choose which

elements you need and use them where you need them, while any changes made to that service will be made on a higher level.



Ref [1]

A pipelining architecture solves the problems that arise at scale:

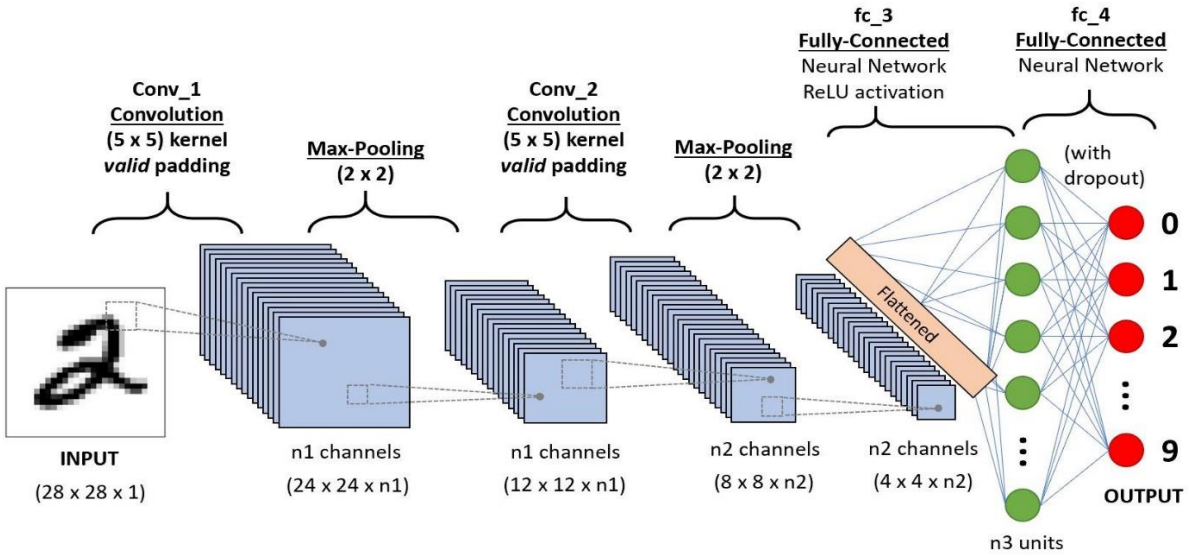
- Volume: only call parts of the workflow when you need them, and cache or store results that you plan on reusing.
- Variety: when you expand your model portfolio, you can use pieces of the beginning stages of the workflow by simply pipelining them into the new models without replicating them.
- Versioning: when services are stored in a central location and pipelined together into various models, there is only one copy of each piece to update. All instances of that code will update when you update the original.[10]

2.4 CNN Model

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 can learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the

2.4 CNN Model

Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.



Ref [2]

- Provide input image into convolution layer.
- Choose parameters, apply filters with strides, padding if requires. Perform convolution on the image and apply ReLU activation to the matrix.
- Perform pooling to reduce dimensionality size.
- Add as many convolutional layers as possible until satisfied.
- Flatten the output and feed into a fully connected layer (FC Layer).
- Output the class using an activation function (Logistic Regression with cost functions) and classifies images. [11]

3 Design

3.1 Overview

In this project three Jupyter notebooks are created for creating different models, training and testing the data set.

3.1.1 Face Recognition:

1. Olivetti Face Dataset.
    1.1. Show 48 Distinct People in the Olivetti Dataset.
    1.2. Show 10 Face Images of Selected Target.
2. Machine Learning Model for Face Recognition.
    2.1. Split data and target into Random train and test Subsets.
    2.2. Principle Component Analysis.
    2.3. PCA Projection of Defined Number of Target.
    2.4. Finding Optimum Number of Principle Component.
    2.5. Show Average Face.
    2.6. Show Eigen Faces.
    2.7. Classification Result.
    2.8. More Results.
    2.9. Validated Results.
    2.10. More Validated Results: Leave One Out cross-validation.
    2.11. Hyperparameter Tunning: GridSearcCV.
    2.12. Precision-Recall-ROC Curves.
3. Linear Discriminant Analysis İle Boyut Azaltma.
4. Machine Learning Automated Workflow: Pipeline. [3]

3.1.2: Face recognition using webcam:

3.1.2: Face recognition using webcam:

1. Load libraries.
2. Load training images.
3. Capture and set frames.
4. Use video capture feature to capture video from webcam.
5. Check the results. [4]

3.1.3: Emotion recognition:

1. Load Libraries.
2. Load Data.
    - 2.1 Helping Function.
    - 2.2 Data Preparation.
    - 2.3 Displaying samples from image dataset.
3. CNN Model.
    - 3.1 Training.
    - 3.2 Save model.
4. Evaluating the model
    - 4.1 Generate test prediction and calculating accuracy.
    - 4.2 Confusion Matrix
    - 4.3 Classification Report
    - 4.4 Class Activation Maps [5]

3.2 Databases:

3.2.1. Olivetti database:

- Face images taken between April 1992 and April 1994.
- There are ten different images of each of 40 distinct people.
- There are 400 face images in the dataset.
- Face images were taken at different times, varying lighting, facial express and facial details.
- All face images have black background.
- The images are gray level.
- Size of each image is 64x64.
- Image pixel values were scaled to [0, 1] interval.
- Names of 40 people were encoded to an integer from 0 to 39. [2]

3.2.2. Fer 2013 database:

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is centered and occupies about the same amount of space in each image.

The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 examples and the public test set consists of 3,589 examples. [1]

3.3 Packages:

3.3.1 OpenCV:

Command: pip install opencv-python

Version: 4.5.1.48

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

3.3 Packages:

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies.

If the import fails on Windows, we should have Visual C++ redistributable 2015 installed. If we are using older Windows version than Windows 10 and latest system updates are not installed, Universal C Runtime might be also required. Windows N and KN editions do not include Media Feature Pack which is required by OpenCV. If you are using Windows N or KN edition, please install also Windows Media Feature Pack. If we have Windows Server 2012+, media DLLs are probably missing too; please install the Feature called "Media Foundation" in the Server Manager. Beware, some posts advise to install "Windows Server Essentials Media Pack", but this one requires the "Windows Server Essentials Experience" role, and this role will deeply affect your Windows Server configuration (by enforcing active directory integration etc.); so just installing the "Media Foundation" should be a safer choice. [7]

3.3.2 sklearn:

Command: conda install -c conda-forge scikit-learn or pip install -U scikit-learn

Version: 0.22+

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. It is built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib!

The functionality that scikit-learn provides include:

- Regression, including Linear and Logistic Regression

3.3 Packages:

- Classification, including K-Nearest Neighbors
- Clustering, including K-Means and K-Means++
- Model selection
- Preprocessing, including Min-Max Normalization [6]

3.3.3 seaborn:

Command: conda install seaborn or pip install seaborn

Version: 0.11.1

The main idea of Seaborn is that it provides high-level commands to create a variety of plot types useful for statistical data exploration, and even some statistical model fitting.

Data visualization is easily performed in Seaborn:

- Data from various sources: The data that is needed to perform visualizations and analytics can come into the architecture from a variety of sources, such as a local storage unit, server, cloud structure, etc.
- Data visualization: This is where the data is transformed from its number-state into an aesthetically pleasing visual counterpart. Seaborn plays the main role here.
- Data Analytics: The result of data visualization is to look at the data in a way you have not done before. Analysis helps doing just this to reveal insights and trends that could not have been spotted otherwise.

This workflow is important as it is the chain of events that helps in driving a variety of businesses and their requirements to their goals.

Features:

- Lots of themes to work with Matplotlib-style graphics.
- Ability to visualize both univariate and multivariate data.
- Support for visualizing varieties of regression model data.
- Easy plotting of statistical data for time-series analytics.

- **3.3 Packages:**

- Seamless performance with Pandas, NumPy, and other Python libraries.[13]

3.3.4 keras:

Command: conda install keras or pip install keras

Version: 2.4+

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

Up until version 2.3 Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML. As of version 2.4, only TensorFlow is supported. 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), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. [8]

3.3.5 TensorFlow:

Command: pip install tensorflow

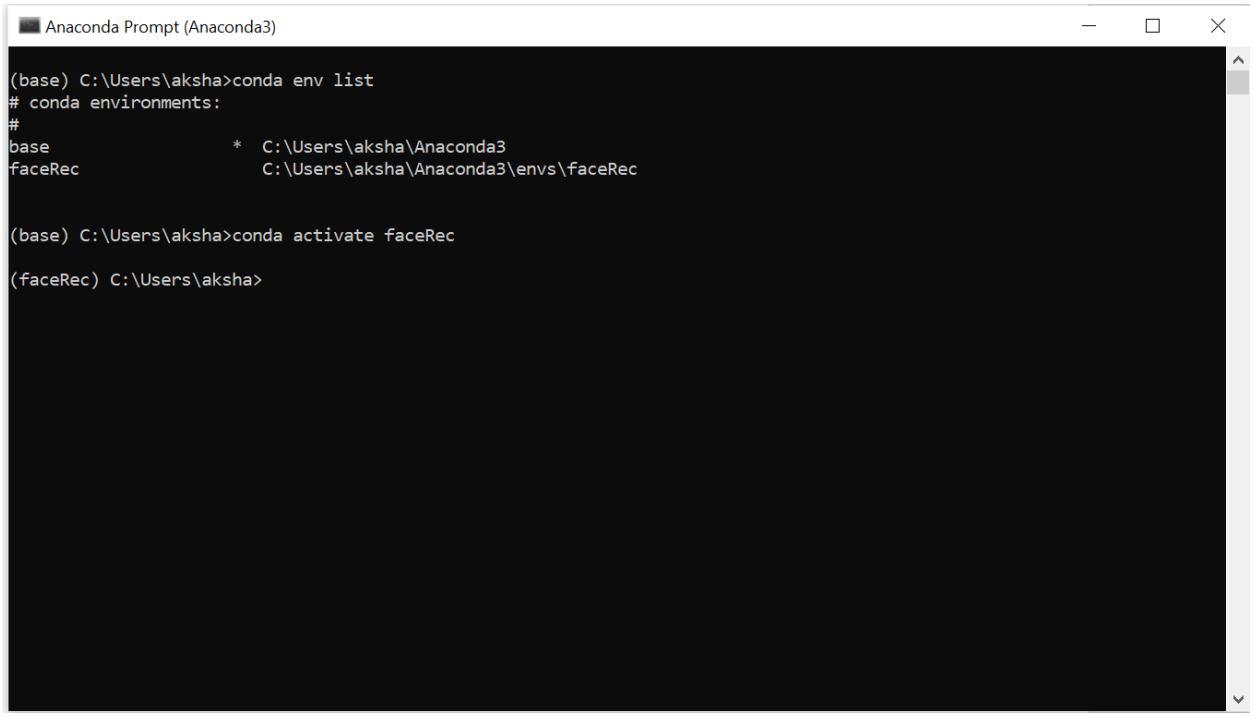
Version: 1.15

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. [9]

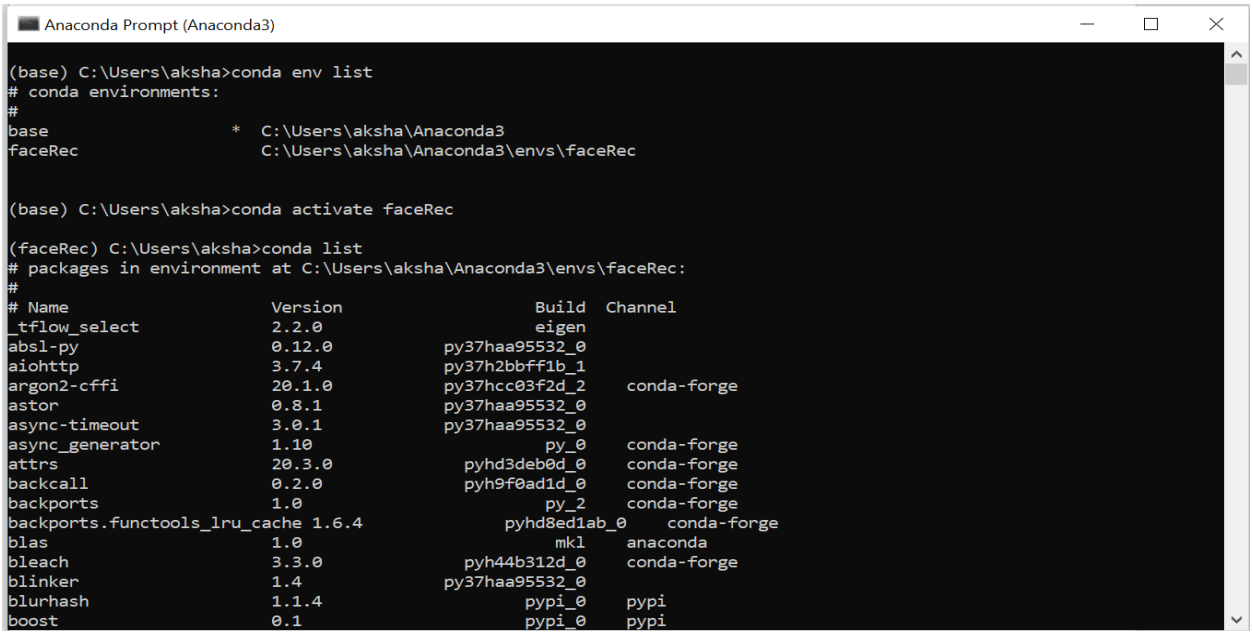
4 Implementation

4.1 Environment (faceRec):

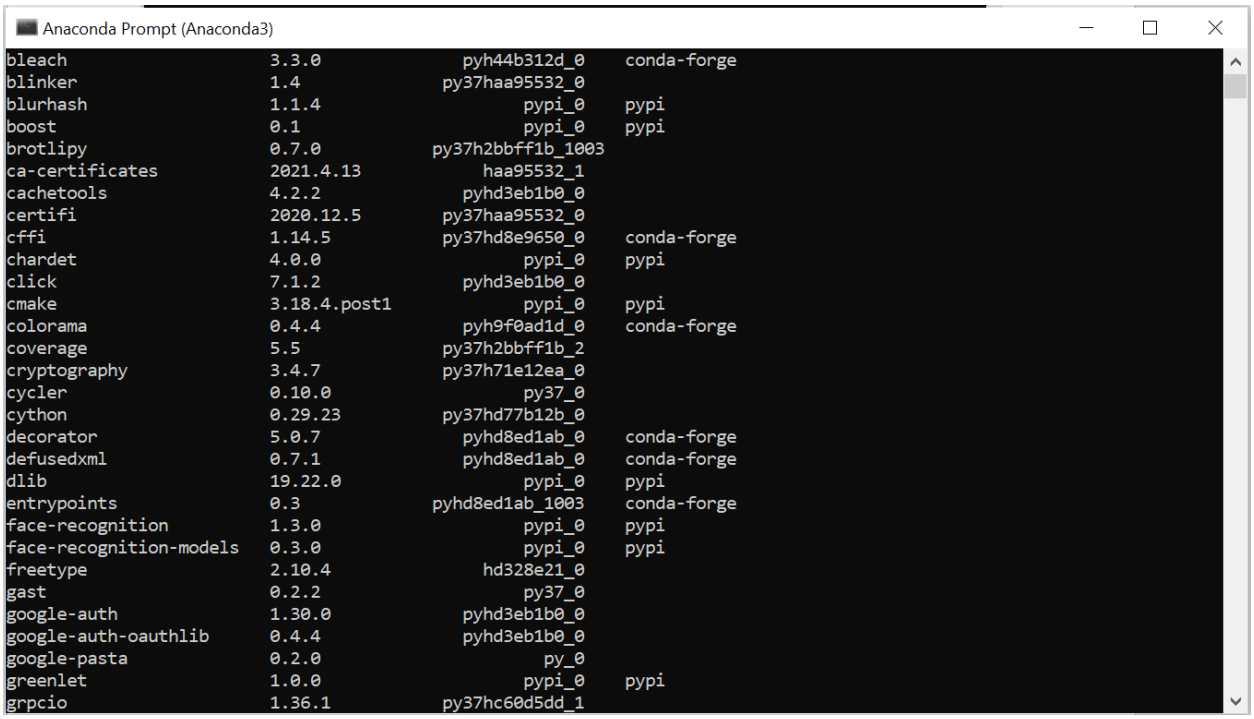
Create and activate the environment:

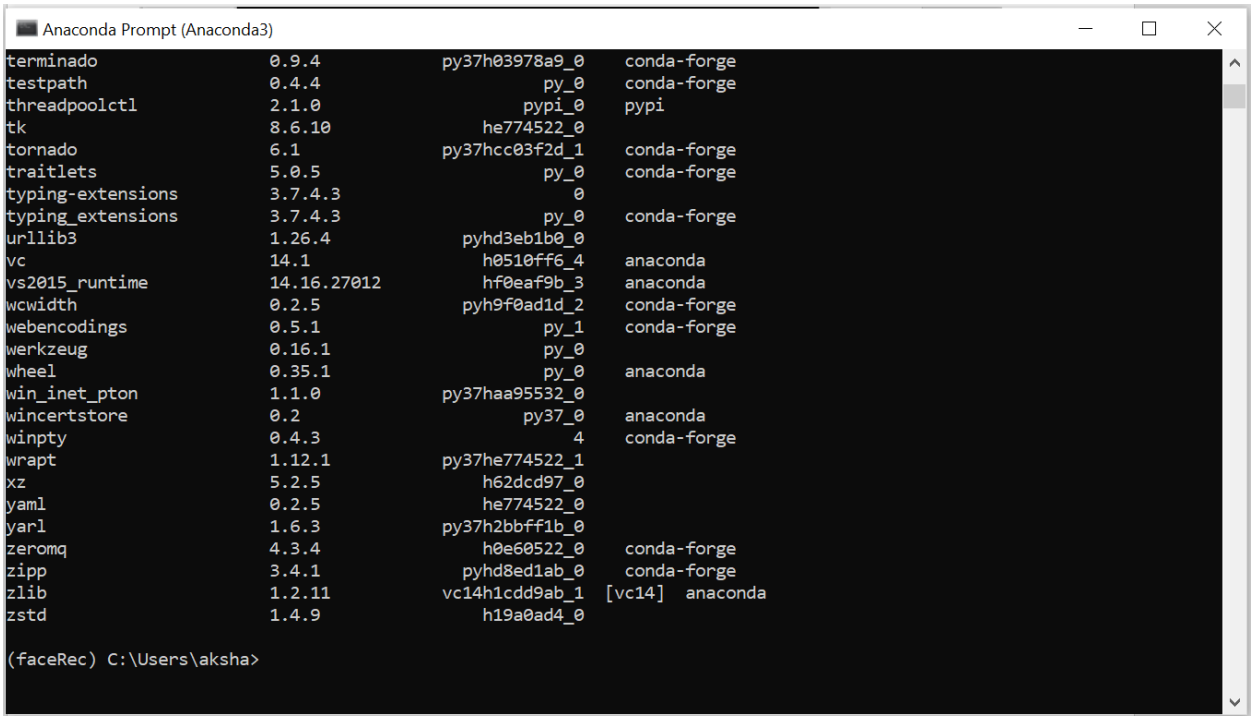


Install all required packages:



4. Implementation



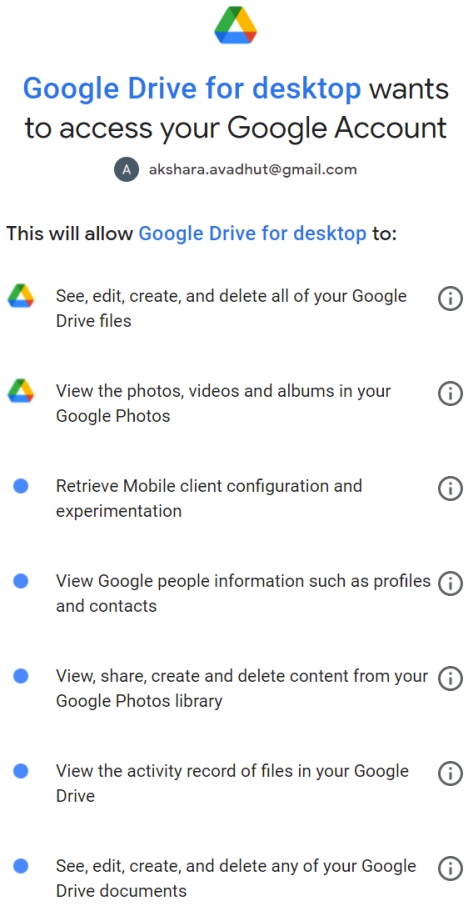


4.2 Uploading data set to drive for colab accessing data:

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuser

Enter your authorization code:
```

Allow access through the link.



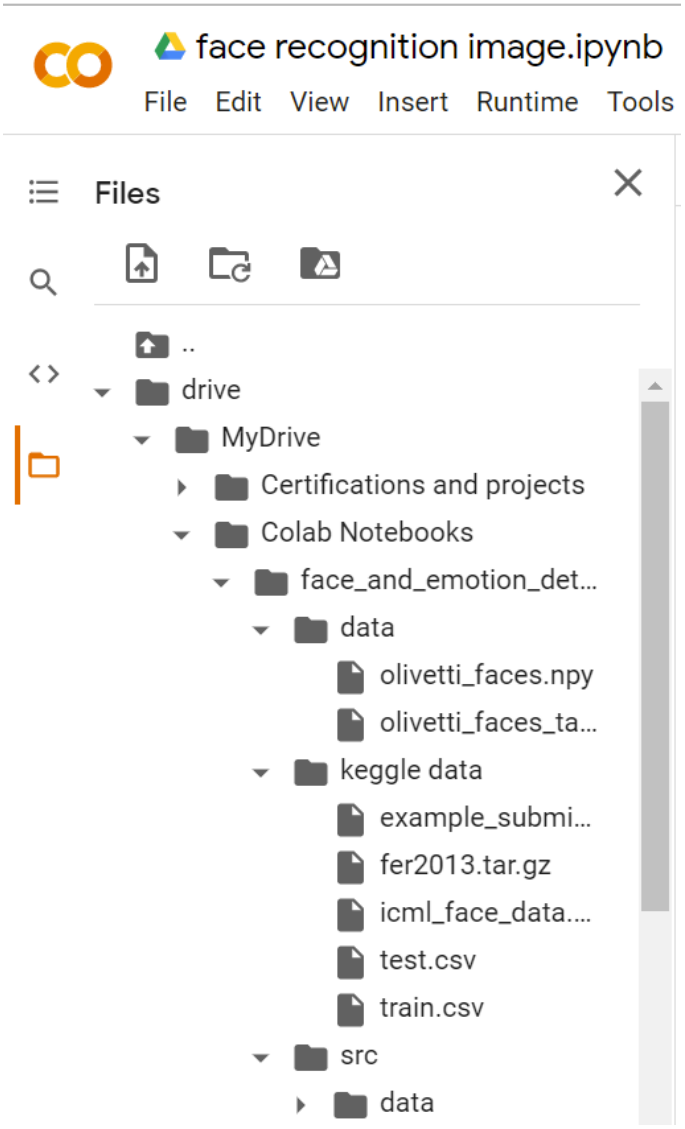
Drive mounted for data access.

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

4. Implementation

Files are accessible.



4. Implementation

4.3 CNN model:

In [14]:

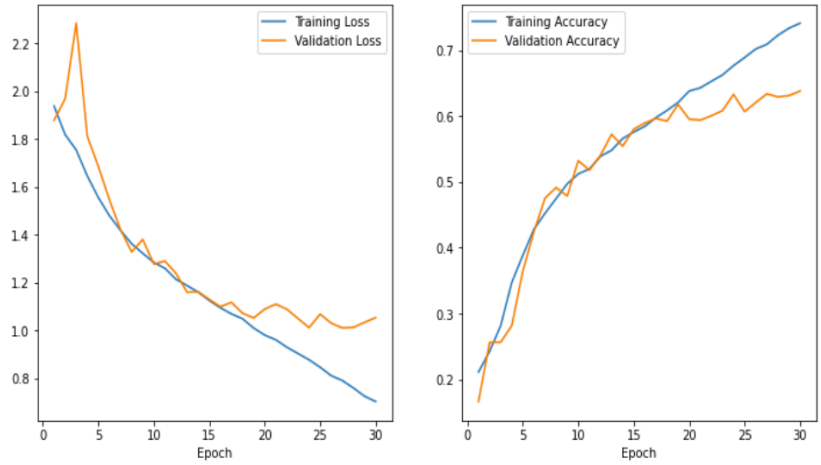
```
%%time

cnn = Sequential()

cnn.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same', input_shape=(48,48,1)))
cnn.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
cnn.add(Dropout(0.25))
cnn.add(BatchNormalization())

cnn.add(Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same'))
cnn.add(Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
cnn.add(Dropout(0.25))
cnn.add(BatchNormalization())

cnn.add(Conv2D(512, kernel_size=(3, 3), activation='relu', padding='same'))
cnn.add(Conv2D(512, kernel_size=(3, 3), activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
cnn.add(Dropout(0.5))
cnn.add(BatchNormalization())

cnn.add(Flatten())

cnn.add(Dense(512, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(512, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(512, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(7, activation='softmax'))

cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 48, 48, 128)       1280
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 48, 48, 128)       147584
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 24, 24, 128)       0
_________________________________________________________________
dropout (Dropout)            (None, 24, 24, 128)       0
_________________________________________________________________
batch_normalization (BatchNo (None, 24, 24, 128)       512
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 256)       295168
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 256)       590080
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 256)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 256)       0
_________________________________________________________________
batch_normalization_1 (Batch (None, 12, 12, 256)       1024
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 512)       1180160
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 512)       2359808
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 512)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 512)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 6, 6, 512)         2048
_________________________________________________________________
flatten (Flatten)            (None, 18432)             0
_________________________________________________________________
dense (Dense)                (None, 512)               9437696
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 3591
=================================================================
Total params: 14,544,263
Trainable params: 14,542,471
Non-trainable params: 1,792
_________________________________________________________________
CPU times: user 589 ms, sys: 266 ms, total: 854 ms
```

4. Implementation

Training and epochs:

In [17]:
```
vis_training([h1])
```



In [18]:
```
%%time
keras.backend.set_value(cnn.optimizer.learning_rate, 0.0001)

h2 = cnn.fit(train_images, train_labels, batch_size=256, epochs=30, verbose=1,
                    validation_data =(valid_images, valid_labels))
```

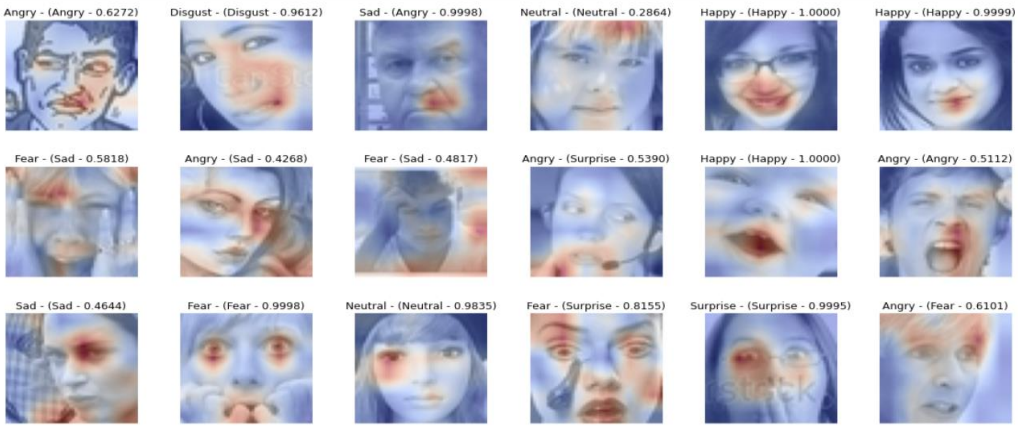
```
Epoch 1/30
90/90 [==============================] - 32s 351ms/step - loss: 0.6070 - accuracy: 0.7763 - val_loss: 0.9846 - val_accuracy:
0.6573
Epoch 2/30
90/90 [==============================] - 32s 353ms/step - loss: 0.5589 - accuracy: 0.7952 - val_loss: 0.9879 - val_accuracy:
0.6602
Epoch 3/30
90/90 [==============================] - 32s 351ms/step - loss: 0.5350 - accuracy: 0.8039 - val_loss: 0.9962 - val_accuracy:
0.6595
Epoch 4/30
90/90 [==============================] - 32s 352ms/step - loss: 0.5190 - accuracy: 0.8079 - val_loss: 0.9951 - val_accuracy:
0.6614
Epoch 5/30
90/90 [==============================] - 32s 352ms/step - loss: 0.5000 - accuracy: 0.8189 - val_loss: 1.0144 - val_accuracy:
0.6630
Epoch 6/30
90/90 [==============================] - 32s 352ms/step - loss: 0.4783 - accuracy: 0.8267 - val_loss: 1.0203 - val_accuracy:
0.6616
```

4. Implementation

4.4 Image processing:

In [29]:

```
plt.figure(figsize=[16,16])
for i in range(36):
    img = test_images[i,:,:,0]
    p_dist = cnn.predict(img.reshape(1,48,48,1))
    k = np.argmax(p_dist)
    p = np.max(p_dist)

    cam = GradCAM(cnn, k)
    heatmap = cam.compute_heatmap(img.reshape(1,48,48,1))

    plt.subplot(6,6,i+1)
    plt.imshow(img, cmap='binary_r')
    plt.imshow(heatmap, alpha=0.5, cmap='coolwarm')
    plt.title(f'{emotions[test_labels[i]]} - ({emotions[k]} - {p:.4f})')
    plt.axis('off')
plt.tight_layout()
plt.show()
```



In [12]:

```
#We reshape images for machine learnig  model
X=data.reshape((data.shape[0],data.shape[1]*data.shape[2]))
print("X shape:",X.shape)
```

```
X shape: (400, 4096)
```

In [13]:

```
X_train, X_test, y_train, y_test=train_test_split(X, target, test_size=0.3, stratify=target, random_state=0)
print("X_train shape:",X_train.shape)
print("y_train shape:{}".format(y_train.shape))
```

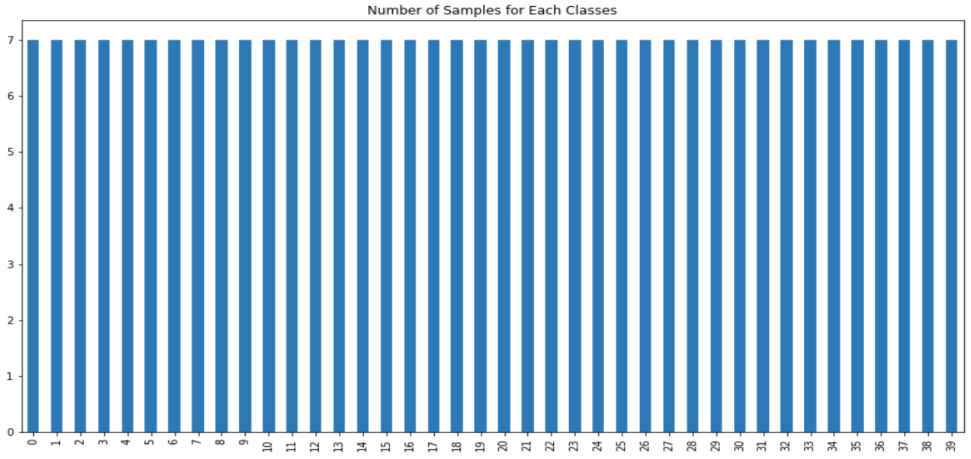
```
X_train shape: (280, 4096)
y_train shape:(280,)
```

In [14]:

```
y_frame=pd.DataFrame()
y_frame['subject ids']=y_train
y_frame.groupby(['subject ids']).size().plot.bar(figsize=(15,8),title="Number of Samples for Each Classes")
```

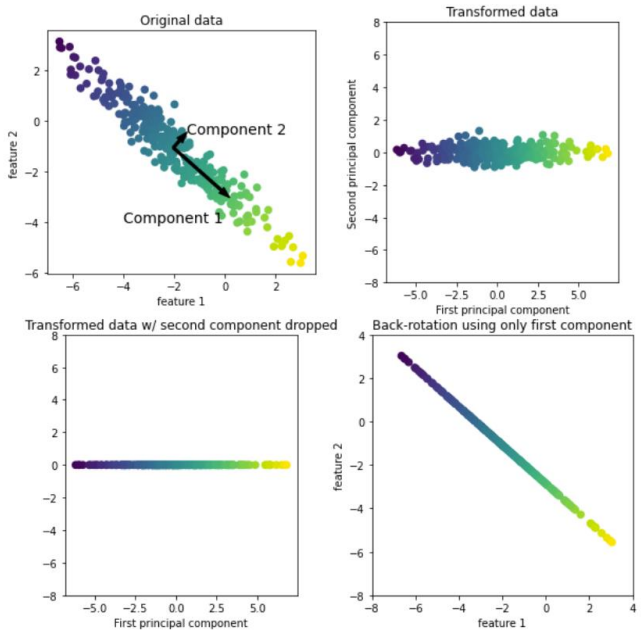
Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff9809b3410>
```



4. Implementation

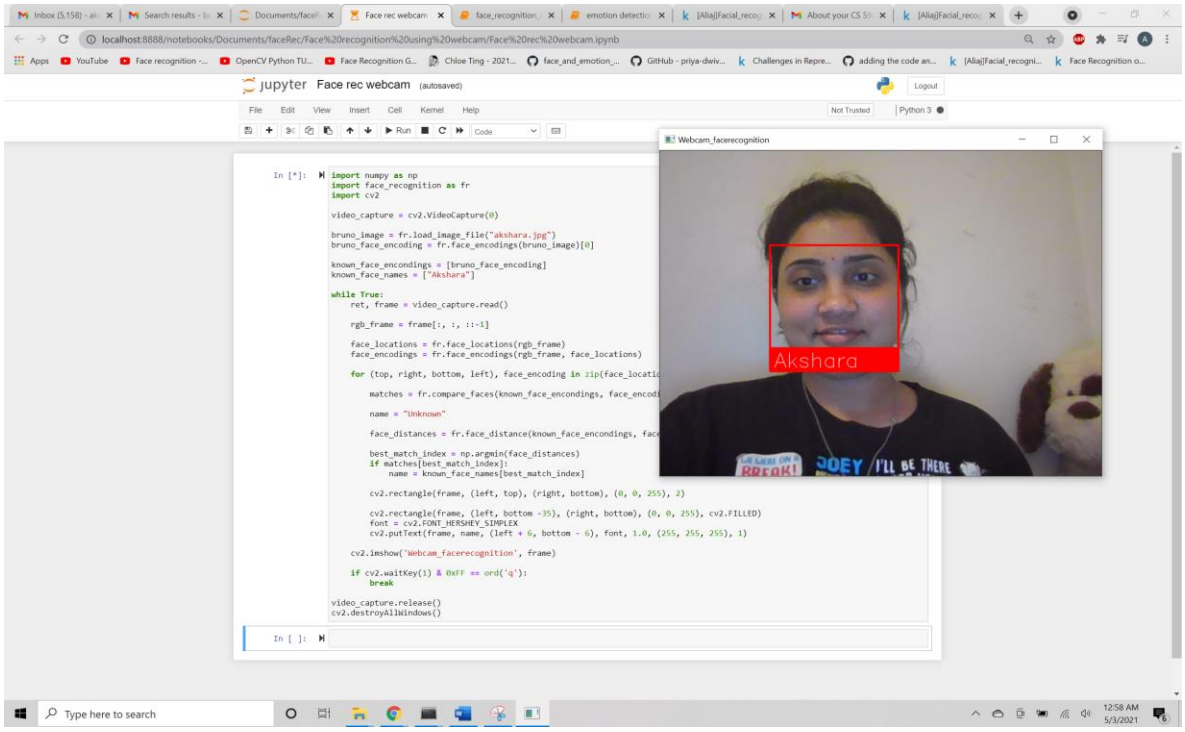
In [18]:
```
mglearn.plots.plot_pca_illustration()
```

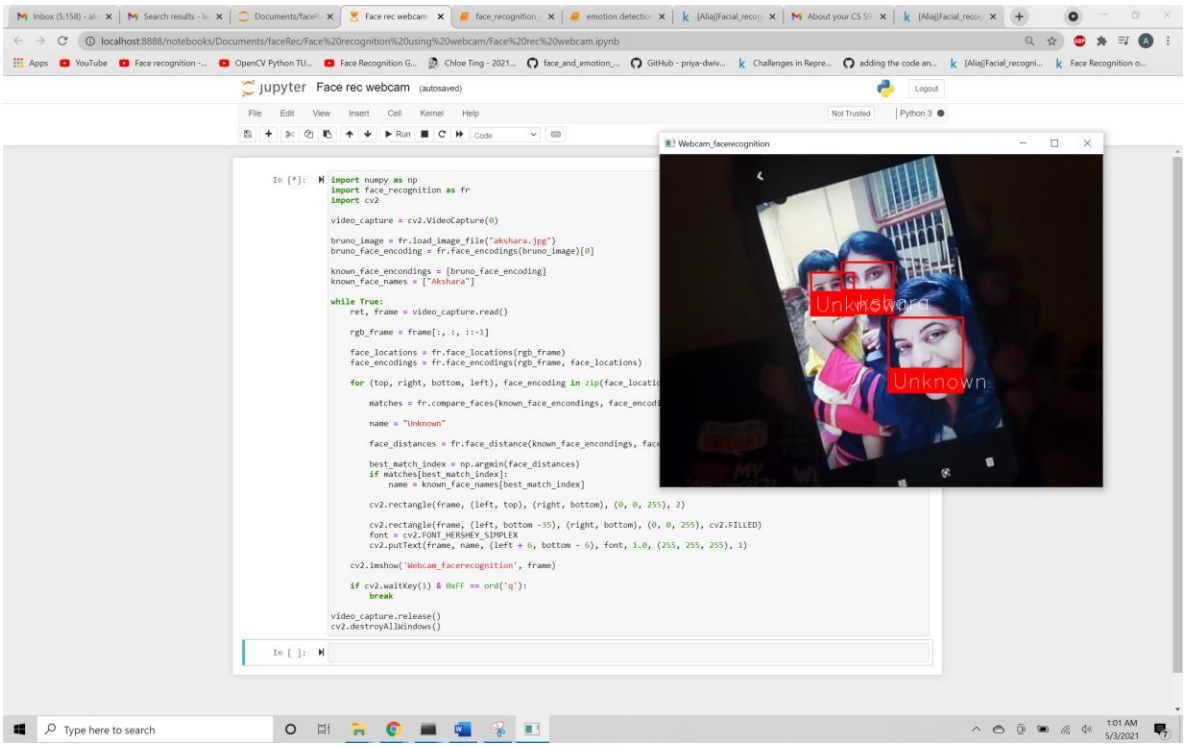


In [56]:
```
print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pred)))
```

```
Accuracy score:0.95
Classification Results:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       1.00      1.00      1.00         3
           2       0.60      1.00      0.75         3
           3       1.00      1.00      1.00         3
           4       1.00      1.00      1.00         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         3
           7       1.00      0.67      0.80         3
           8       1.00      1.00      1.00         3
           9       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         3
          12       1.00      0.67      0.80         3
          13       1.00      1.00      1.00         3
          14       0.75      1.00      0.86         3
          15       1.00      1.00      1.00         3
          16       1.00      1.00      1.00         3
          17       1.00      1.00      1.00         3
          18       1.00      1.00      1.00         3
          19       1.00      1.00      1.00         3
          20       1.00      1.00      1.00         3
          21       1.00      0.67      0.80         3
          22       1.00      1.00      1.00         3
          23       1.00      1.00      1.00         3
          24       1.00      1.00      1.00         3
          25       1.00      0.67      0.80         3
          26       1.00      1.00      1.00         3
          27       1.00      1.00      1.00         3
          28       1.00      1.00      1.00         3
          29       1.00      1.00      1.00         3
          30       0.60      1.00      0.75         3
          31       1.00      1.00      1.00         3
          32       1.00      1.00      1.00         3
          33       1.00      1.00      1.00         3
          34       1.00      1.00      1.00         3
          35       1.00      1.00      1.00         3
          36       1.00      1.00      1.00         3
          37       1.00      1.00      1.00         3
          38       0.75      1.00      0.86         3
          39       1.00      1.00      1.00         3

    accuracy                           0.95       120
   macro avg       0.97      0.95      0.95       120
weighted avg       0.97      0.95      0.95       120
```

4. Implementation





5. Conclusion

5.1 Accomplishments and lesson learned:

As part of this project research, I developed a deep understanding of the convolution neural networks and requirements for creating an accurate model for natural face and emotion identification. The implementation of a bidirectional model is much more recommended than the traditional sequential unidirectional models.

As part of the code implementation, Google Colab platform provided by Google is a great help to developers that allowed me to train the CNN model at higher epochs and on GPU as well as TPU. Python frameworks like Keras and openCV enabled the pre-processing apart from just training the model (Keras).

The references use different types of models under different circumstances. I have tried to collaborate different approaches from models in attempt to increase the accuracy. The weights and biases are adjusted to increase the learning. The PCA model implementation, the parameters passed to face recognition function for directories try to highlight the increased accuracy. Besides the training and testing the database, the attempt is also made to lively capture the face through web camera. The face is identified through web camera with 0.6-0.7 accuracy. There is scope to increase the speed and performance too. Understanding the code from references and then attempting for variance was bit of a task.

I have learned about image processing and a lot of the things that comes with face detection and face tracking. Not only have I learned things about my project I have also learned different techniques about my project. I have learned the importance of face tracking and face detection in the real world. I learned that there are a lot of face tracking and face detection programs and software but the challenge for programmers today is how to make it more robust and how to modify it to fit each individual specific need. I have also learned things that will benefit me in my future education in school. I have learned things such as programming, teamwork adapting to new environments, and oral presentations.

References

[1] https://www.kaggle.com/msambare/fer2013

[2] https://www.kaggle.com/serkanpeldek/face-recognition-on-olivetti-dataset/data

[3] https://www.kaggle.com/serkanpeldek/face-recognition-on-olivetti-dataset

[4] https://www.youtube.com/watch?v=lC_y8wD7F3Y

[5] https://www.kaggle.com/drbeanesp21/aliaj-facial-recognition-v01

[6] https://en.wikipedia.org/wiki/Scikit-learn

[7] https://pypi.org/project/opencv-Python/

[8] https://pypi.org/project/keras/

[9] https://www.tensorflow.org/

[10] https://algorithmia.com/blog/ml-pipeline

[11] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53#:~:text=A%20Convolutional%20Neural%20Network%20(ConvNet,differentiate%20one%20from%20the%20other.

[12] https://web.stevens.edu/wireless/reu/2005/final-reports/coney-dorsey-final-report.pdf

[13]https://seaborn.pydata.org/#:~:text=Seaborn%20is%20a%20Python%20data,can%20read%20the%20introductory%20notes.

[14] https://builtin.com/data-science/step-step-explanation-principal-component-analysis

[15] https://www.cs.cmu.edu/~schneide/tut5/node42.html

Image References:

[1] https://algorithmia.com/blog/ml-pipeline

[2] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53#:~:text=A%20Convolutional%20Neural%20Network%20(ConvNet,differentiate%20one%20from%20the%20other.

Appendix

Code for face recognition through webcam

```
import numpy as np
import face_recognition as fr
import cv2
video_capture = cv2.VideoCapture(0)
akshara_image = fr.load_image_file("akshara.jpg")
akshar_face_encoding = fr.face_encodings(aksharaimage)[0]

known_face_encondings = [akshara_face_encoding]
known_face_names = ["Akshara"]

while True:
    ret, frame = video_capture.read()

    rgb_frame = frame[:, :, ::-1]

    face_locations = fr.face_locations(rgb_frame)
    face_encodings = fr.face_encodings(rgb_frame, face_locations)
    for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
        matches = fr.compare_faces(known_face_encondings, face_encoding
        name = "Unknown
```

```
        face_distances = fr.face_distance(known_face_encondings, face_encoding)
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]:
            name = known_face_names[best_match_index]

        cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)
        cv2.rectangle(frame, (left, bottom -35), (right, bottom), (0, 0, 255), cv2.FILLED)
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1
    cv2.imshow('Webcam_facerecognition', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
video_capture.release()
cv2.destroyAllWindows()
```

Face Detection using Olivetti's data:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

#Visualization
import matplotlib.pyplot as plt

#Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn import metrics

#System
import os
print(os.listdir("/content/drive/MyDrive/Colab Notebooks/face_and_emotion_detection-master/data"))

data=np.load("/content/drive/MyDrive/Colab Notebooks/face_and_emotion_detection-master/data/olivetti_faces.npy")
target=np.load("/content/drive/MyDrive/Colab Notebooks/face_and_emotion_detection-master/data/olivetti_faces_target.npy")

print("There are {} images in the dataset".format(len(data)))
print("There are {} unique targets in the dataset".format(len(np.unique(target))))
print("Size of each image is {}x{}".format(data.shape[1],data.shape[2]))
print("Pixel values were scaled to [0,1] interval. e.g:{}".format(data[0][0,:4]))

def show_40_distinct_people(images, unique_ids):
    #Creating 4X10 subplots in  18x9 figure size
    fig, axarr=plt.subplots(nrows=4, ncols=10, figsize=(18, 9))
    #For easy iteration flattened 4X10 subplots matrix to 40 array
    axarr=axarr.flatten()

    #iterating over user ids
    for unique_id in unique_ids:
        image_index=unique_id*10
        axarr[unique_id].imshow(images[image_index], cmap='gray')
        axarr[unique_id].set_xticks([])
        axarr[unique_id].set_yticks([])
        axarr[unique_id].set_title("face id:{}".format(unique_id))
```

```
    plt.suptitle("There are 40 distinct people in the dataset")
show_40_distinct_people(data, np.unique(target))

def show_10_faces_of_n_subject(images, subject_ids):
    cols=10# each subject has 10 distinct face images
    rows=(len(subject_ids)*10)/cols #
    rows=int(rows)
    fig, axarr=plt.subplots(nrows=rows, ncols=cols, figsize=(18,9))
    #axarr=axarr.flatten()

    for i, subject_id in enumerate(subject_ids):
        for j in range(cols):
            image_index=subject_id*10 + j
            axarr[i,j].imshow(images[image_index], cmap="gray")
            axarr[i,j].set_xticks([])
            axarr[i,j].set_yticks([])
            axarr[i,j].set_title("face id:{}".format(subject_id))

show_10_faces_of_n_subject(images=data, subject_ids=[0,5, 21, 24, 36])
X=data.reshape((data.shape[0],data.shape[1]*data.shape[2]))
print("X shape:",X.shape)
X_train, X_test, y_train, y_test=train_test_split(X, target, test_size=0.3, stratify=target,
random_state=0)
print("X_train shape:",X_train.shape)
print("y_train shape:{}".format(y_train.shape))

y_frame=pd.DataFrame()
y_frame['subject ids']=y_train
y_frame.groupby(['subject ids']).size().plot.bar(figsize=(15,8),title="Number
```

```
mglearn.plots.plot_pca_illustration()

from sklearn.decomposition import PCA
pca=PCA(n_components=2)
pca.fit(X)
X_pca=pca.transform(X)
number_of_people=10
index_range=number_of_people*10
fig=plt.figure(figsize=(10,8))
ax=fig.add_subplot(1,1,1)
scatter=ax.scatter(X_pca[:index_range,0],
          X_pca[:index_range,1],
          c=target[:index_range],
          s=10,
         cmap=plt.get_cmap('jet', number_of_people)
        )

ax.set_xlabel("First Principle Component")
ax.set_ylabel("Second Principle Component")
ax.set_title("PCA projection of {} people".format(number_of_people))

fig.colorbar(scatter)
pca=PCA()
pca.fit(X)
plt.figure(1, figsize=(12,8))
plt.plot(pca.explained_variance_, linewidth=2)
plt.xlabel('Components')
plt.ylabel('Explained Variaces')
plt.show()
n_components=90
```

```
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X_train)
fig,ax=plt.subplots(1,1,figsize=(8,8))
ax.imshow(pca.mean_.reshape((64,64)), cmap="gray")
ax.set_xticks([])
ax.set_yticks([])
ax.set_title('Average Face')
number_of_eigenfaces=len(pca.components_)
eigen_faces=pca.components_.reshape((number_of_eigenfaces, data.shape[1], data.shape[2]))

cols=10
rows=int(number_of_eigenfaces/cols)
fig, axarr=plt.subplots(nrows=rows, ncols=cols, figsize=(15,15))
axarr=axarr.flatten()
for i in range(number_of_eigenfaces):
    axarr[i].imshow(eigen_faces[i],cmap="gray")
    axarr[i].set_xticks([])
    axarr[i].set_yticks([])
    axarr[i].set_title("eigen id:{}".format(i))
plt.suptitle("All Eigen Faces".format(10*"=", 10*"="))

X_train_pca=pca.transform(X_train)
X_test_pca=pca.transform(X_test)
clf = SVC()
clf.fit(X_train_pca, y_train)
y_pred = clf.predict(X_test_pca)
print("accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))


import seaborn as sns
```

```
plt.figure(1, figsize=(12,8))
sns.heatmap(metrics.confusion_matrix(y_test, y_pred))

models=[]
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(("LR",LogisticRegression()))
models.append(("NB",GaussianNB()))
models.append(("KNN",KNeighborsClassifier(n_neighbors=5)))
models.append(("DT",DecisionTreeClassifier()))
models.append(("SVM",SVC()))
for name, model in models:
    clf=model
    clf.fit(X_train_pca, y_train)
    y_pred=clf.predict(X_test_pca)
    print(10*"=","{} Result".format(name).upper(),10*"=")
    print("Accuracy score:{:0.2f}".format(metrics.accuracy_score(y_test, y_pred)))
    print()
rom sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X)
X_pca=pca.transform(X)
for name, model in models:
    kfold=KFold(n_splits=5, shuffle=True, random_state=0)
    cv_scores=cross_val_score(model, X_pca, target, cv=kfold)
    print("{} mean cross validations score:{:.2f}".format(name, cv_scores.mean()))
cm=metrics.confusion_matrix(y_test, y_pred)
plt.subplots(1, figsize=(12,12))
sns.heatmap(cm)
```

```
from sklearn.pipeline import Pipeline
work_flows_std = list()
work_flows_std.append(('lda', LinearDiscriminantAnalysis(n_components=n_components)))
work_flows_std.append(('logReg', LogisticRegression(C=1.0, penalty="l2")))
model_std = Pipeline(work_flows_std)
model_std.fit(X_train, y_train)
y_pred=model_std.predict(X_test)

print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pred)))
```

Code for emotion recognition CNN model:

```
def prepare_data(data):
    image_array = np.zeros(shape=(len(data), 48, 48, 1))
    image_label = np.array(list(map(int, data['emotion'])))

    for i, row in enumerate(data.index):
        image = np.fromstring(data.loc[row, 'pixels'], dtype=int, sep=' ')
        image = np.reshape(image, (48, 48))
        image_array[i, :, :, 0] = image / 255

    return image_array, image_label

def vis_training(hlist, start=1):

    loss = np.concatenate([h.history['loss'] for h in hlist])
    val_loss = np.concatenate([h.history['val_loss'] for h in hlist])
    acc = np.concatenate([h.history['accuracy'] for h in hlist])
    val_acc = np.concatenate([h.history['val_accuracy'] for h in hlist])
```

```
    epoch_range = range(1,len(loss)+1)

    plt.figure(figsize=[12,6])
    plt.subplot(1,2,1)
    plt.plot(epoch_range[start-1:], loss[start-1:], label='Training Loss')
    plt.plot(epoch_range[start-1:], val_loss[start-1:], label='Validation Loss')
    plt.xlabel('Epoch')
    plt.legend()
    plt.subplot(1,2,2)
    plt.plot(epoch_range[start-1:], acc[start-1:], label='Training Accuracy')
    plt.plot(epoch_range[start-1:], val_acc[start-1:], label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.legend()

    plt.show()

%%time

cnn = Sequential()

cnn.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=(48,48,1)))
cnn.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
cnn.add(Dropout(0.25))
cnn.add(BatchNormalization())

cnn.add(Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same'))
cnn.add(Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

```
cnn.add(Dropout(0.25))
cnn.add(BatchNormalization())

cnn.add(Conv2D(512, kernel_size=(3, 3), activation='relu', padding='same'))
cnn.add(Conv2D(512, kernel_size=(3, 3), activation='relu', padding='same'))
cnn.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
cnn.add(Dropout(0.5))
cnn.add(BatchNormalization())

cnn.add(Flatten())

cnn.add(Dense(512, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(512, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(512, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(7, activation='softmax'))

cnn.summary()

%%time
h1 = cnn.fit(train_images, train_labels, batch_size=256, epochs=30, verbose=1,
             validation_data =(valid_images, valid_labels))

class GradCAM:
   def __init__(self, model, classIdx, layerName=None):
      self.model = model
      self.classIdx = classIdx
      self.layerName = layerName
```

```
    if self.layerName is None:
        self.layerName = self.find_target_layer()

def find_target_layer(self):
    for layer in reversed(self.model.layers):
        if len(layer.output_shape) == 4:
            return layer.name
    raise ValueError("Could not find 4D layer. Cannot apply GradCAM.")

def compute_heatmap(self, image, eps=1e-8):
    gradModel = Model(
        inputs=[self.model.inputs],
        outputs=[self.model.get_layer(self.layerName).output,self.model.output]
    )

    with tf.GradientTape() as tape:
        inputs = tf.cast(image, tf.float32)
        (convOutputs, predictions) = gradModel(inputs)
        loss = predictions[:, self.classIdx]
        grads = tape.gradient(loss, convOutputs)

        castConvOutputs = tf.cast(convOutputs > 0, "float32")
        castGrads = tf.cast(grads > 0, "float32")
        guidedGrads = castConvOutputs * castGrads * grads
        convOutputs = convOutputs[0]
        guidedGrads = guidedGrads[0]
        weights = tf.reduce_mean(guidedGrads, axis=(0, 1))
        cam = tf.reduce_sum(tf.multiply(weights, convOutputs), axis=-1)

        (w, h) = (image.shape[2], image.shape[1])
```

```
            heatmap = cv2.resize(cam.numpy(), (w, h))
            numer = heatmap - np.min(heatmap)
            denom = (heatmap.max() - heatmap.min()) + eps
            heatmap = numer / denom
            heatmap = (heatmap * 255).astype("uint8")
        return heatmap

    def overlay_heatmap(self, heatmap, image, alpha=0.5,
        colormap = cv2.COLORMAP_VIRIDIS):
        heatmap = cv2.applyColorMap(heatmap, colormap)
        output = cv2.addWeighted(image, alpha, heatmap, 1 - alpha, 0)
        return (heatmap, output)

plt.figure(figsize=[16,16])
for i in range(36):
    img = test_images[i,:,:,0]
    p_dist = cnn.predict(img.reshape(1,48,48,1))
    k = np.argmax(p_dist)
    p = np.max(p_dist)

    cam = GradCAM(cnn, k)
    heatmap = cam.compute_heatmap(img.reshape(1,48,48,1))

    plt.subplot(6,6,i+1)
    plt.imshow(img, cmap='binary_r')
    plt.imshow(heatmap, alpha=0.5, cmap='coolwarm')
    plt.title(f'{emotions[test_labels[i]]} - ({emotions[k]} - {p:.4f})')
    plt.axis('off')
plt.tight_layout()
plt.show()
```