Facial Recognition using Deep Learning

A Project Report

submitted for the partial fulfillment for the award of the degree of

B.Tech (CS)

Submitted by:

Shivangi Dixit

Under the supervision of :

Dr. Pinaki Roy Chowdhury
Mrs. Monika

External Guide
Dr. Pinaki Roy Chowdhury
Scientist - F
DTRL Lab, DRDO, Delhi

Internal Guide
Mrs. Monika
Assistant Professor
Banasthali University

Apaji Institute of Mathematics and Applied Computer Technology
Banasthali Vidyapith

Banasthali - 304022

Session: 2014-18

<u>CERTIFICATE</u>

Certified that **Shivangi Dixit** has carried out the thesis work titled **“Facial Recognition using Deep Learning“** from **03/07/2017** to **13/12/2017** for the award of the degree of **B.Tech** from **Banasthali Vidyapith** under my supervision. The thesis embodies result of original work and studies carried out by student herself and the contents of the thesis do not form the basis for the award of any other degree to the candidate or to anybody else.

Dr. Pinaki Roy Chowdhury

Scientist-F

DRDO (DTRL Lab), Delhi

Date :

CERTIFICATE

This is to certify that **Ms. Shivangi Dixit**, Roll No. **8638** a student of **B.Tech (CS),VII** semester of **Banasthali Vidyapith** has completed the research project titled **"Facial Recognition using Deep Learning"** of 6 months duration in **DRDO**, in partial fulfillment of **B.Tech**.

Mrs. Monika

Asst. Professor

Banasthali Vidyapith

Date :

ABSTRACT

Convolutional Neural Network (CNN), one of the Deep Learning technique has achieved promising results in face recognition recently. Here, we have used public face databases of various universities to perform our facial recognition task.

We have used different architectures of CNN on datasets. To reduce the overfitting of the models, we have utilized techniques like dropout. We have also presented the visualization of different layers of a network to show what features of a face can be learned by CNN models.

We have used CNN architecture that is capable of handling facial images that contain faces of different poses, facial expressions and varying illumination. CNN is also used in order to extract relevant facial features. These features allow to compare faces in an efficient way. The system is trained to recognize a set of people. We have also tested our model on faces of new people.

ACKNOWLEDGEMENT

I am highly thankful to my mentor **Dr. Pinaki Roy Chowdhury** Sir for providing me an opportunity to do my project work in DTRL Lab of DRDO.
I sincerely thank him for the guidance, constant supervision and for his support in completing the project.

I would like to express my special gratitude to my internal guide **Mrs. Monika** Ma’am for her co-operation which helped me to complete this project.

I sincerely thank my department AIM & ACT for providing me an opportunity to do my internship in DRDO.

Also, I would like to thank my parents who constantly motivated me to work on this project.

Table of Contents

CHAPTER 1 INTRODUCTION

Facial Recognition is one of the areas in Computer Vision that has always drawn more interest for long.

There are several factors because of which facial recognition is a challenging problem:

i) due to varying poses of the facial image. Certain poses cause facial features such as eyes or nose to be partially occluded.

ii) due to presence or absence of components such as beard, moustaches, with or without glasses in the image. These components have large variability including shape, colour and size.

iii) Illumination : It means change of light ambient due to skin reflectance properties and camera control that may cast shadow on some part of the face.

iv) Occlusion : It means object covering part of the face, such as a scarf, turban, etc.

v) Facial expressions like smile, laugh, anger, sad, surprise, disgust, and fright.

To perform facial recognition tasks, Neural networks perform quite well. It has the ability to learn from experience. Inspired by the human brain, a neural network or multilayer perceptron (MLP) works as a powerful classifier. But, MLP requires the input image to undergo several image processing tasks such as pre-processing, feature extraction in order to provide good performance. MLP suffers from the existence of free parameters i.e. redundant information in its architecture. These free parameters are formed by full connections between the input layer and the feature maps of the following layer.

Therefore, a variant of MLP that overcomes the limitations mentioned above is known as Convolutional Neural Network (CNN).

CNN applies three concepts in its architecture: shared weights, local receptive field and subsampling.

Shared weights are formed by convolution filter in order to reduce the number of free parameters. These convolution kernels are randomly initialized and used as edge detectors for the feature maps at the respective layer.

Local receptive field is applied by both convolution and subsampling filters by taking a group of pixels for further processing before passing the result to the following CNN layers.

Subsampling process reduces the feature map size at the respective layer. At this point, the exact locations of the features of the image are not important anymore.

CNN is a multi tasking algorithm. It performs feature extraction, face detection and classification in one processing module with minimal pre-processing tasks on the input image. It is quite useful for tasks such as face detection, face recognition, gender recognition, object recognition, character recognition, texture recognition and so on. This is completely in contrast with the conventional facial recognition tasks in which to perform all the above tasks, different algorithms are needed.

Our goal is to create a Facial Recognition system, capable of working with any kind of images which will help to better recognise people in less time.

CHAPTER 2 LITERATURE REVIEW

The research papers which have been referred are:

i) *When Face Recognition Meets with Deep Learning: an Evaluation of Convolutional Neural Networks for Face Recognition* [6]

Convolutional Neural Network (CNN), has achieved very promising results in face recognition recently. Most of the existing studies tend to focus on CNN architectures that work well for face recognition rather than to investigate the reason that why they work well.

In this paper, authors conducted an evaluation of CNN based face recognition system (CNNFRS) to make the work easily reproducible. They have used public database LFW (Labeled Faces in theWild) to train CNNs unlike the existing studies which trains CNNs on private databases.

They proposed three CNN architectures which are the first reported architectures trained using LFW data. This paper compares the architectures of CNNs and evaluates the effect of different implementation techniques.

ii) *Progress in Supervised Neural Networks* [1]

In this paper, theoretical results concerning the capabilities and limitations of various neural network models are summarized and some of their future extensions are discussed.

The network models considered are divided into two basic categories: static networks and dynamic networks. Unlike static networks, dynamic networks have memory.

They fall into three groups: networks with feedforward dynamics, networks with output feedback, and networks with state feedback, which are emphasized in this paper. Most of the networks discussed are trained using supervised learning.

iii) *Face Recognition: features versus templates* [2]

In this paper, two new algorithms for computer recognition of human faces, one based on the computation of a set of geometrical features, such as nose width and length, mouth position, and chin shape, and the second based on almost-gray-level template matching are presented.

The results obtained from the testing sets show about 90% correct recognition using geometrical features and perfect recognition using template matching.

iv) *Trainable Convolution Filters and their application to Face Recognition* [3]

In this paper, authors presented a novel image classification system that is built around a core of trainable filter ensembles that is called Volterra kernel classifiers. Their system treats images as a collection of possibly overlapping patches and is composed of three components:

1) A scheme for a single patch classification that seeks a smooth, possibly nonlinear, functional mapping of the patches into a range space, where patches of the same class are close to one another, while patches from different classes are far apart in the L2 sense.

This mapping is accomplished using trainable convolution filters (or Volterra kernels) where the convolution kernel can be of any shape or order.

2) Given a corpus of Volterra classifiers with various kernel orders and shapes for each patch, a boosting scheme is used for automatically selecting the best weighted combination of the classifiers to achieve higher per-patch classification rate.

3) A scheme for aggregating the classification information obtained for each patch via voting for the parent image classification.

They demonstrated the effectiveness of the proposed technique using face recognition as an application area and did extensive experiments on the Yale, CMU PIE, Extended Yale B, Multi-PIE, and MERL Dome benchmark face data sets.

They called the Volterra kernel classifiers applied to face recognition Volterrafaces. They said that their technique, which falls into the broad class of embedding based face image discrimination methods, consistently outperforms various state-of-the-art methods in the same category.

v) *A Convolutional Neural Network Cascade for Face Detection* [4]

In real-world face detection, large visual variations, such as those due to pose, expression, and lighting, demand an advanced discriminative model to accurately differentiate faces from the backgrounds. Consequently, effective models for the problem tend to be computationally prohibitive.

To address these two conflicting challenges, this paper proposed a cascade architecture built on convolutional neural networks (CNNs) with very powerful discriminative capability, while maintaining high performance.

The proposed CNN cascade operates at multiple resolutions, quickly rejects the background regions in the fast low resolution stages, and carefully evaluates a small number of challenging candidates in the last high resolution stage.

To improve localization effectiveness, and reduce the number of candidates at later stages, authors introduced a CNN-based calibration stage after each of the detection stages in the cascade.

The output of each calibration stage is used to adjust the detection window position for input to the subsequent stage. The proposed method runs at 14 FPS on a single CPU core for VGA-resolution images and 100 FPS using a GPU, and achieved state-of-the-art detection performance on two public face detection benchmarks.

vi) *Convolutional neural network for face recognition with pose and illumination variation* [5]

Face recognition remains a challenging problem till today. The main challenge is how to improve the recognition performance when affected by the variability of non-linear effects that include illumination variances, poses, facial expressions, occlusions, etc.

In this paper, a robust 4-layer Convolutional Neural Network (CNN) architecture is proposed for the face recognition problem, with a solution that is capable of handling facial images that contain occlusions, poses, facial expressions and varying illumination.

Experimental results depicted that the proposed CNN solution outperforms existing works, achieving 99.5% recognition accuracy on AR database.

The test on the 35-subjects of FERET database achieved an accuracy of 85.13%, which is in the similar range of performance as the best result of previous works. More significantly, proposed system completes the facial recognition process in less than 0.01 seconds.

vii) *Deep Face Recognition* [7]

The goal of this paper is to perform face recognition from either a single photograph or from a set of faces tracked in a video. Recent progress in this area has been due to two factors: (i) end to end learning for the task using a convolutional neural network (CNN), and (ii) the availability of very large scale training datasets.

They made two contributions:

First, they demonstrated how a very large scale dataset (2.6M images, over 2.6K people) can be assembled by a combination of automation and human in the loop, and discussed the trade off between data purity and time.

Second, they traversed through the complexities of deep network training and face recognition to present methods and procedures to achieve comparable state of the art results on the standard LFW and YTF face benchmarks.

viii) *Visualizing and Understanding Convolutional Networks* [13]

There is no clear understanding of why large convolutional network models perform so well, or how they might be improved. In this paper authors addressed both issues.

They introduced a novel visualization technique that gives insight into the function of intermediate feature layers and the operation of the classifier. Used in a diagnostic role, these visualizations allowed to find model architectures that outperformed ImageNet classification benchmark.

They also performed an ablation study to discover the performance contribution from different model layers. They demonstrated that ImageNet model generalizes well to other datasets: when the softmax classifier is retrained, it convincingly beats the current state-of-the-art results on Caltech-101 and Caltech-256 datasets.

CHAPTER 3 METHODOLOGY

3.1. DATASETS

To perform the facial recognition task, datasets have been selected from various sources.

Following 10 face datasets are used for experiment :

i) ORL face database

This dataset consists of 400 face images of 40 distinct individuals. There are 10 different images of each individual. The files are in PGM format which are then converted into JPEG format.

The images consist of different facial expressions i.e. open or closed eyes, smiling or non-smiling and different facial details i.e. with glasses or no glasses.

Out of 400 images, 200 images are used for training purpose(5 images/individual) and 200 images for test purpose (5 images/individual).

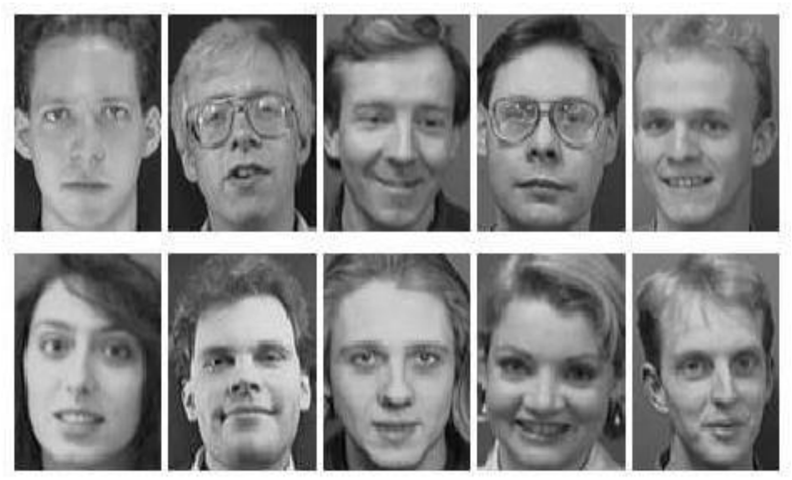


Figure 1: Face samples from the ORL database

ii) JAFFE database

This database is Japanese Female Facial Expression database. It contains 213 images having 7 facial expressions : 6 basic facial expressions and 1 neutral expression posed by 10 Japanese female models.

Out of 213 images, 109 images are used for training purpose and 104 images for test purpose.

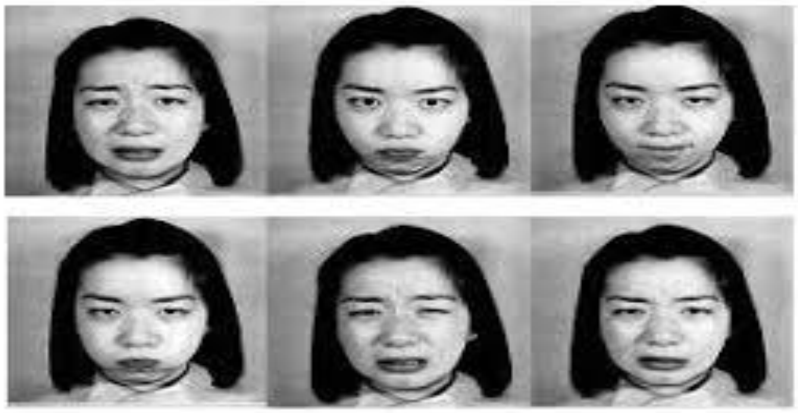


Figure 2: Face samples from the Jaffe database

iii) Extended Yale Face Database

The extended Yale B face database contains 16128 images of 28 individuals under 9 poses and 64 illumination conditions.

Out of 16128 images, 8204 images are used for training purpose and 8204 images for test purpose.

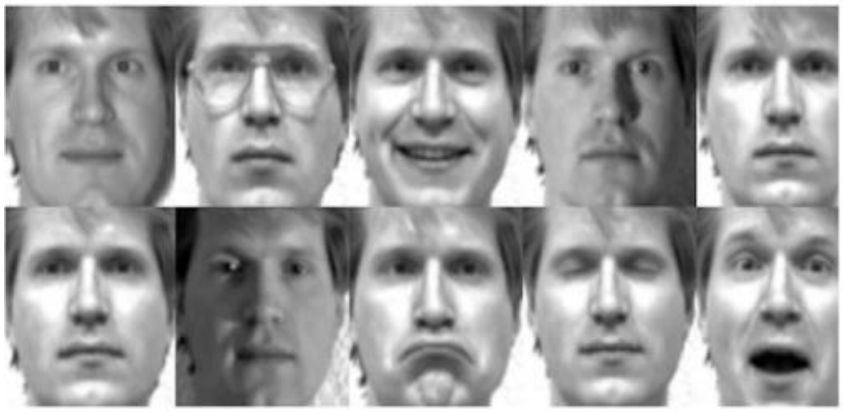


Figure 3: Face samples from the Extended Yale-B face database

iv) Georgia Tech Face Database

The database contains images of 50 people and is stored in JPEG format. For each individual, there are 15 different images. The images were taken in two different sessions to get the variations in illumination conditions, facial expression, and appearance. Also, the faces were captured at different scales and orientations.

Out of 750 images, 350 images are used for training purpose and 400 images for test purpose.

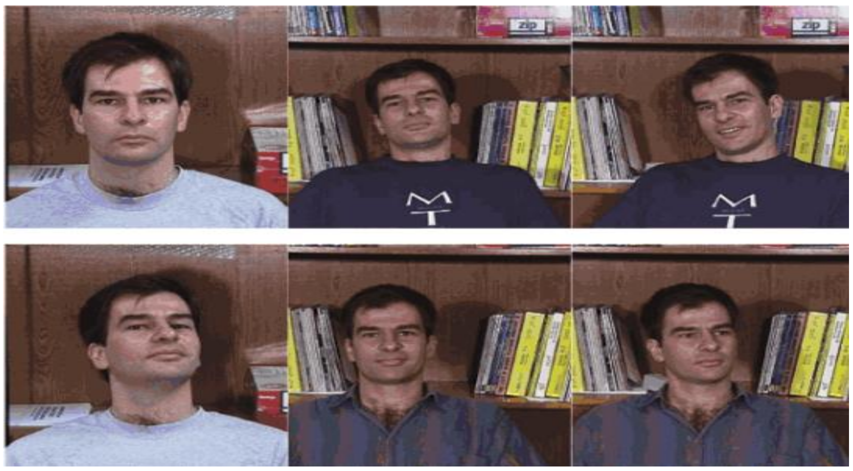


Figure 4: Face samples from the Georgia Tech face database

v) Georgia Tech Face Database (Cropped)

These are the cropped images of Georgia Tech Face Database.

Out of 750 images, 350 images are used for training purpose and 400 images for test purpose.

Figure 5: Cropped face samples from the Georgia Tech face database

vi) UMIST-C Face database

UMIST Cropped Face Database consists of 564 images of 20 individuals (mixed race/gender/appearance). The files are all in PGM format, approximately 220 x 220 pixels with 256-bit grey-scale.

Out of 564 images, 274 images are used for training purpose and 290 images for test purpose.

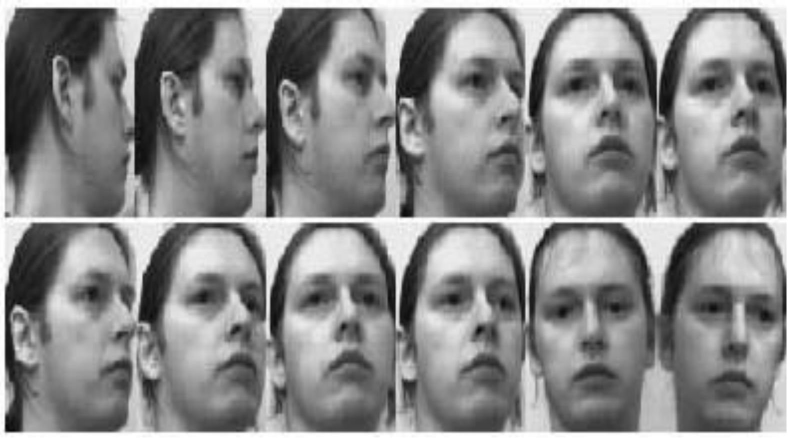


Figure 6: Cropped face samples from the UMIST face database

vii) UMIST – F Face database

The UMIST Face Database consists of images of size 220 x 220 of 20 individuals. No. of pictures per person varies from 19 to 36 under different conditions from various angles.

Out of 1012 images, 500 images are used for training purpose and 512 images for test purpose.

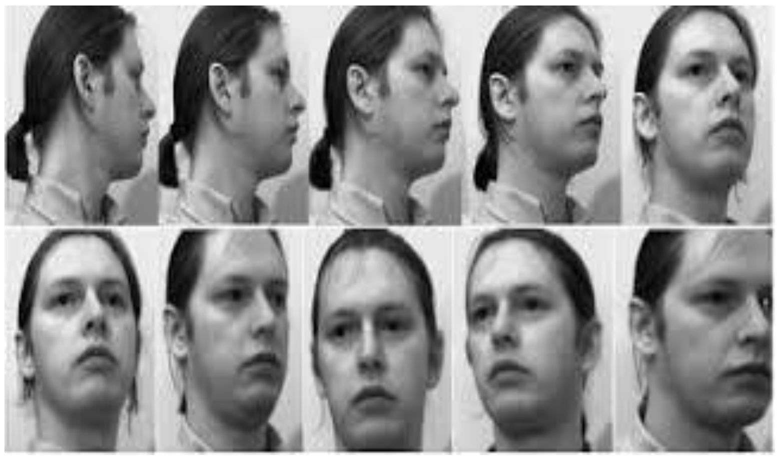


Figure 7: Face samples from the UMIST face database

viii) MIT-CBCL Face database

The MIT-CBCL face recognition database contains face images of 10 subjects. It provides two training sets:

1. High resolution pictures, including frontal, half-profile and profile view.

2. Synthetic images (324/subject) rendered from 3D head models of the 10 subjects. The test set consists of 200 images per subject.

Out of 3240 images, 1620 images are used for training purpose and 1620 images for test purpose.

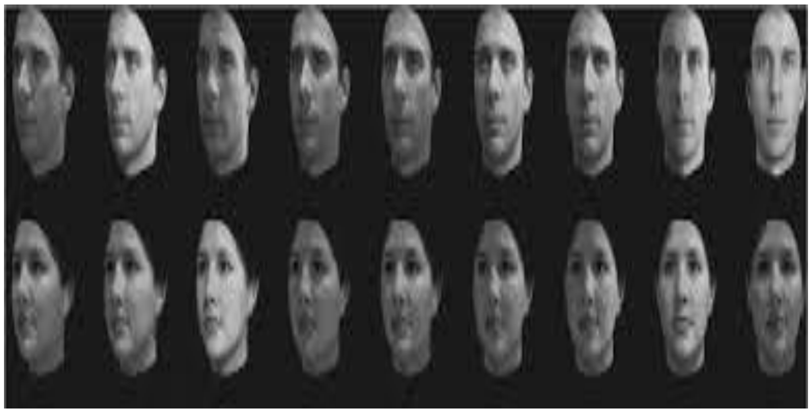


Figure 8: Face samples from the MIT-CBCL face database

ix) Caltech Face Database

It consists of 450 images of 896 x 592 pixels size in JPEG format. There are 27 or so unique individuals under different lighting/expressions/backgrounds.

Out of 450 images, 209 images are used for training purpose and 109 images for test purpose.

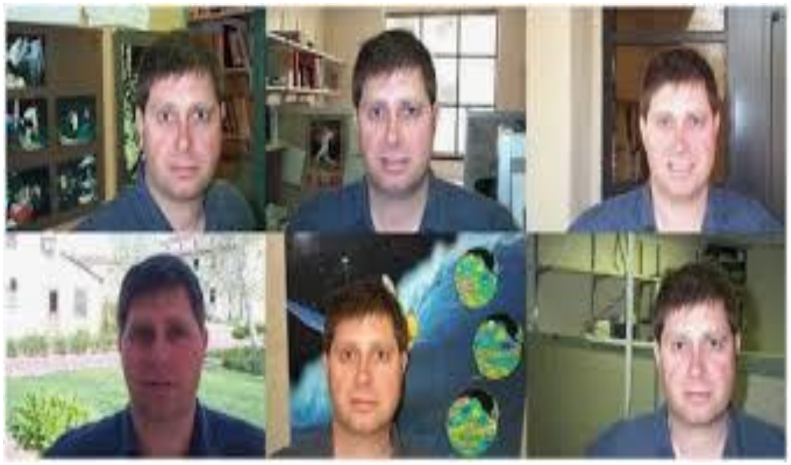


Figure 9: Face samples from the Caltech face database

x) Caltech Face Database (Cropped)

It consists of cropped images of Caltech Face Database.

Out of 450 images, 209 images are used for training purpose and 109 images for test purpose.

Figure 10: Cropped face samples from the Caltech face database

3.2. Data Preprocessing and data augmentation

To get the better results, we have augmented the images via some random transformations so that our model should never see twice the same image. This helps to prevent overfitting and helps the model to generalize better.

Transformations that we have used are :

i) **rescale :** Original images are in RGB coefficients in the range of 0-255 pixels, but such values would be too high for model to process given a typical learning rate, so we should target values between 0 and 1 instead, by scaling with a 1/255 factor.

ii) **shear_range :** It is for applying shape transformations. This is done so that our model generalizes well for different shapes of the images.

iii) **zoom_range :** It is for randomly zooming inside pictures. This is done so that our model would be easily able to extract the features.

iv) **rotation_range :** Range within which to randomly rotate the pictures so that model could see different views of the same image.

3.3 DEEP LEARNING

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

The main characteristic of Deep Learning is that it is capable of making abstractions by building complex concepts from simpler ones. Given an image, it is capable of learning concepts such as cars, cats or humans by combining sets of basic features, such as corners or edges. This process is done through successive “layers” that increase the complexity of the learned concepts.

The idea of depth in Deep Learning comes precisely from these abstraction levels. Each layer gets the input as the output of the previous one and it uses to learn higher-level features. In some cases it is the own network that uses these features to produce an output, and sometimes it simply generates them for other methods to use. See the figure below :

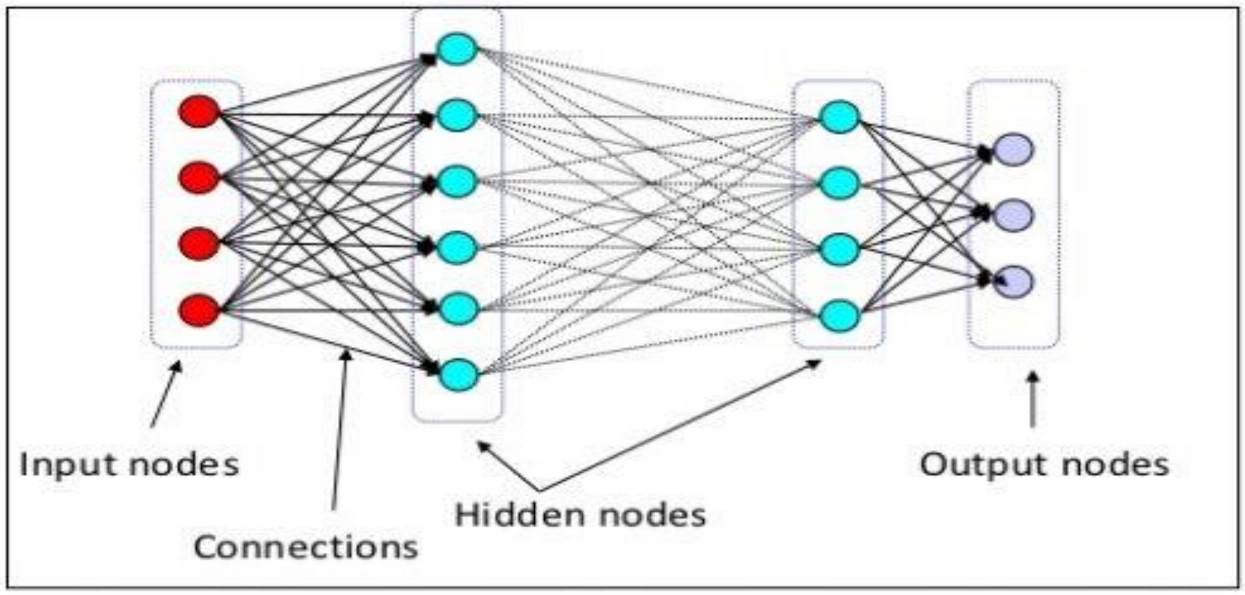


Figure 11: Layers of neural network. Source [31]

3.3.1 Convolutional Neural Networks

Convolutional Neural Networks (**ConvNets** or **CNNs**) are a category of Neural Networks that are very effective in areas such as image recognition and classification. ConvNets have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self driving cars.

CNN Architecture :

There are four main operations in the CNNs :

1. Convolution
2. Non Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification (Fully Connected Layer)

These operations are the basic building blocks of every Convolutional Neural Network.

An Image is a matrix of pixel values.

Essentially, every image can be represented as a matrix of pixel values.

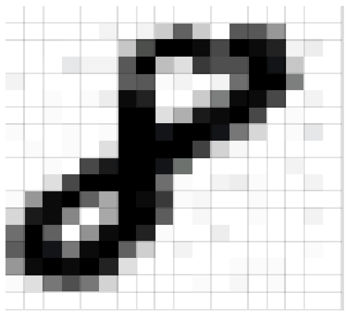


Figure 12: Every image is a matrix of pixel values. Source [32]

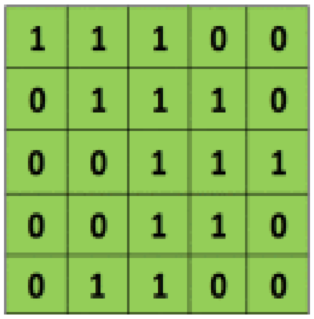
Channel is a conventional term used to refer to a certain component of an image. An image from a standard digital camera will have three channels : red, green and blue. They can be imagined as three 2d-matrices stacked over each other (one for each color), each having pixel values in the range 0 to 255.

A grayscale image has just one channel. The value of each pixel in the matrix will range from 0 to 255 : zero indicating black and 255 indicating white.

ConvNets derive their name from the “convolution” operator. The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

Example :

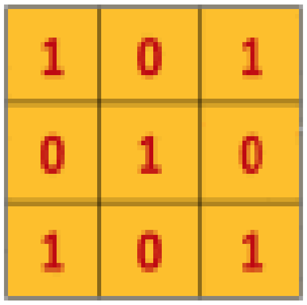
Consider a 5 x 5 image whose pixel values are only 0 and 1 (for a grayscale image, pixel values range from 0 to 255).



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Figure 13: 5x5 image. Source [32]

Consider another 3 x 3 matrix as shown below:



1	0	1
0	1	0
1	0	1

Figure 14: 3x3 matrix. Source [32]

Then, the Convolution of the 5 x 5 image and the 3 x 3 matrix can be computed as shown :

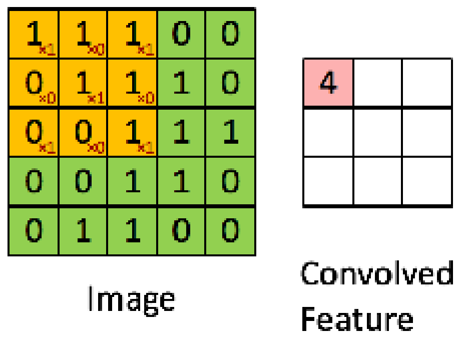


Figure 15 : Convolution of 5x5 image and 3x3 matrix. Source [32]

In CNN terminology, the 3×3 matrix is called a '**filter**' or '**kernel**' or '**feature detector'** and the matrix formed by sliding the filter over the image and computing the dot product is called the '**Convolved Feature'** or '**Activation Map'** or the '**Feature Map**'. Filters act as feature detectors from the original input image.

Different values of the filter matrix will produce different feature maps for the same input image.

CNN learns the values of these filters on its own during the training process although we still need to specify parameters such as number of filters, filter size, architecture of the network etc. before the training process. The more number of filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images.

The size of the Feature Map (Convolved Feature) is controlled by three parameters that we need to decide before the convolution step is performed:

- **Depth:** Depth corresponds to the number of filters we use for the convolution operation. In the given figure, we are performing convolution of the original boat image using three distinct filters, thus producing three different feature maps as shown. These features can be seen as three feature maps as stacked 2d matrices, so, the 'depth' of the feature map would be three.

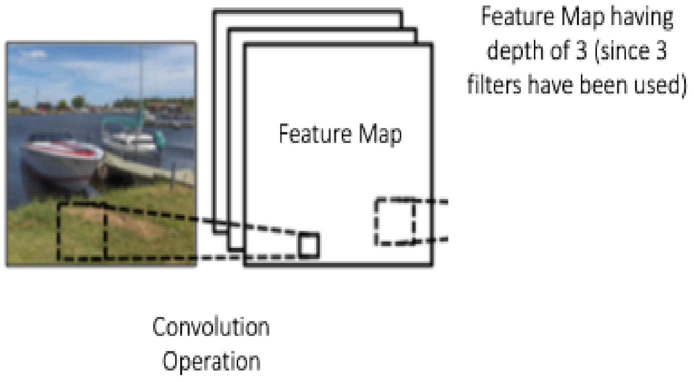
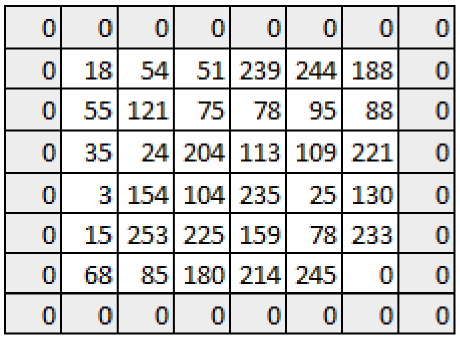

Feature Map having depth of 3 (since 3 filters have been used)
Feature Map
Convolution Operation

Figure 16: Convolution of image using 3 distinct filters. Source [32]

- **Stride:** Stride is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2, then the filters jump 2 pixels at a time as we slide them around. Having a larger stride will produce smaller feature maps.

- **Zero-padding:** Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix. A nice feature of zero padding is that it allows us to control the size of the feature maps. Adding zero-padding is also called wide convolution**,** and not using zero-padding would be a narrow convolution.



0	0	0	0	0	0	0	0
0	18	54	51	239	244	188	0
0	55	121	75	78	95	88	0
0	35	24	204	113	109	221	0
0	3	154	104	235	25	130	0
0	15	253	225	159	78	233	0
0	68	85	180	214	245	0	0
0	0	0	0	0	0	0	0

Figure 17: Image padded with one layer of zeroes. Source [34]

Multiple filters and the activation map

Depth dimension of the weight would be same as the depth dimension of the input image. The weight extends to the entire depth of the input image. Therefore, convolution with a single weight matrix would result into a convolved output with a single depth dimension. In most cases instead of a single filter(weight matrix), we have multiple filters of the same dimensions applied together.

The output from the each filter is stacked together forming the depth dimension of the convolved image. Suppose we have an input image of size 32x32x3. And we apply 10 filters of size 5x5x3 with valid padding. The output would have the dimensions as 28x28x10.

You can visualize it as :

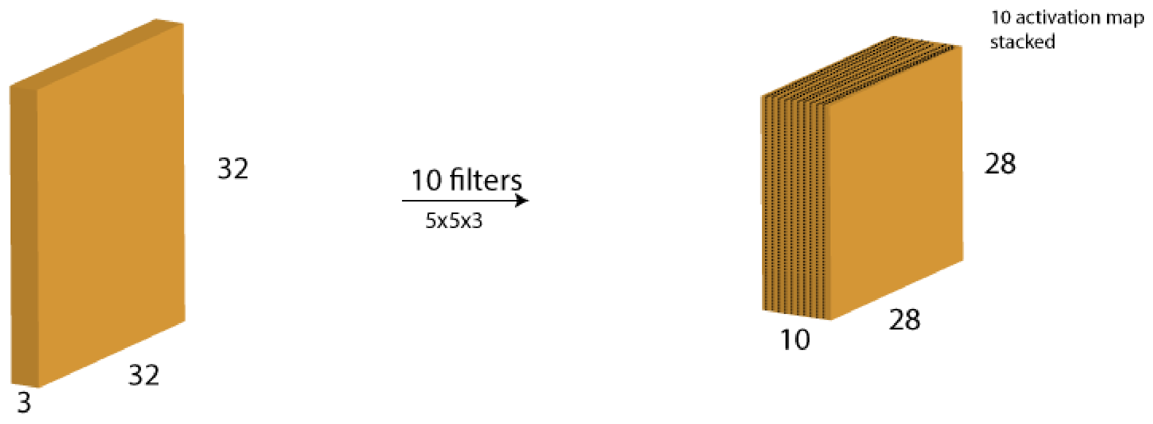


Figure 18: 10 filters of size 5x5x3 convolved on the image of size 32x32x3. Source [34]

This activation map is the output of the convolution layer.

Output dimensions:

We can apply a simple formula to calculate the output dimensions. The spatial size of the output image can be calculated as ([W-F+2P]/S)+1.

Here,

W is the input volume size, F is the size of the filter, P is the number of padding applied and S is the number of strides. Suppose we have an input image of size 32x32x3, we apply 10 filters of size 3x3x3, with single stride and no zero padding.

In figure 18, W=32, F=5, P=0 and S=1. The output depth will be equal to the number of filters applied i.e. 10.

The size of the output volume will be ([32-5+0]/1)+1 = 28. Therefore the output volume will be 28x28x10.

Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by:

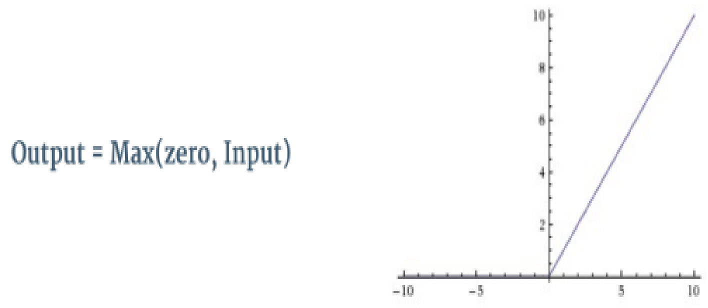


Figure 19: The ReLU operation. Source [32]

ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation – element wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).

In the given picture, ReLU operation is applied to one of the feature maps. The output feature map here is also referred to as the ‘Rectified’ feature map.

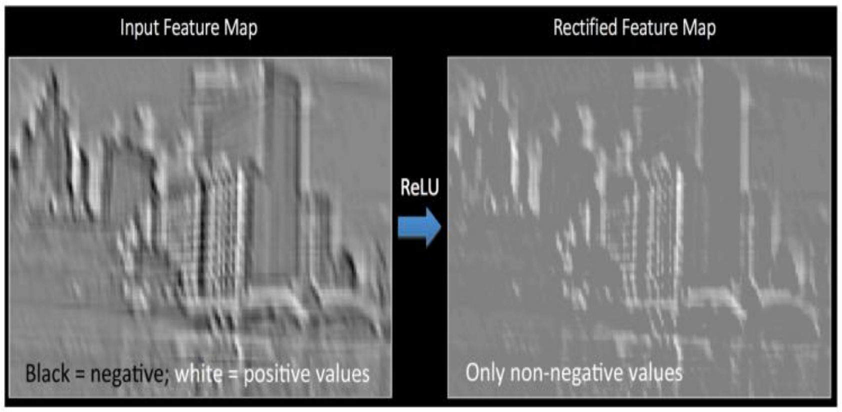


Figure 20: Effect of ReLU on input feature map. Source [32]

Other non linear functions such as **tanh** or **sigmoid** can also be used instead of ReLU, but ReLU has been found to perform better in most situations.

Pooling

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.

In case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.

Given figure shows an example of Max Pooling operation on a Rectified Feature map (obtained after convolution + ReLU operation) by using a 2×2 window.

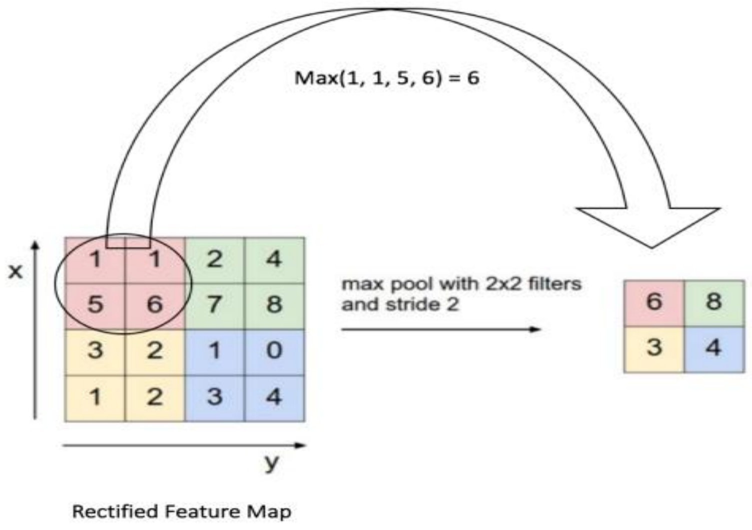


Figure 21: The Pooling operation. Source [32]

We slide our 2 x 2 window by 2 cells (also called 'stride') and take the maximum value in each region. This reduces the dimensionality of our feature map.

In the network shown in given figure**,** pooling operation is applied separately to each feature map. Due to this, we get three output maps from three input maps.

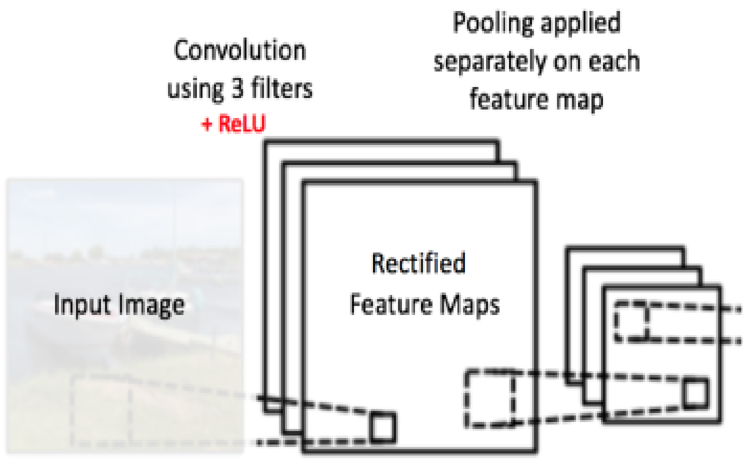


Figure 22: Pooling applied to rectified feature map. Source [32]

Figure shows the effect of Pooling on the Rectified Feature Map received after the ReLU operation.

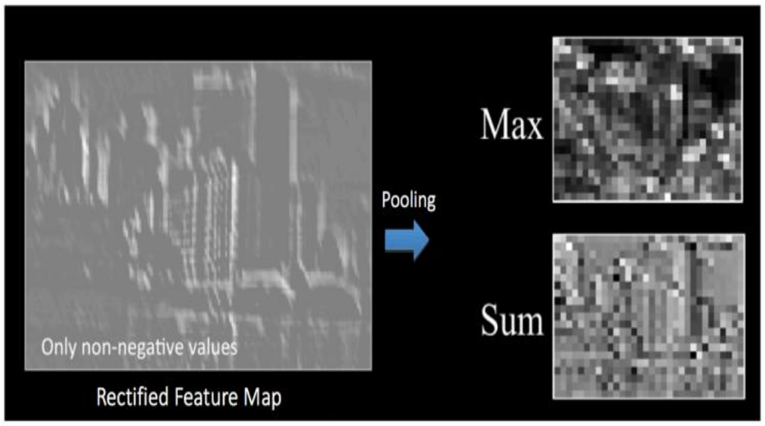


Figure 23: Max and Sum Pooling operations. Source [32]

The function of Pooling is to progressively reduce the spatial size of the input representation. In particular, pooling :

- makes the input representations (feature dimension) smaller and more manageable.
- reduces the number of parameters and computations in the network, therefore, controlling overfitting.
- makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the

output of Pooling – since we take the maximum / average value in a local neighborhood).

- helps us to arrive at an almost scale invariant representation of our image (the exact term is “equivariant”). This is very powerful since we can detect objects in an image no matter where they are located.

Fully Connected Layer

The Fully Connected layer is a Multi Layer Perceptron that uses a softmax or sigmoid activation function in the output layer (other classifiers like SVM can also be used). The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.

The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

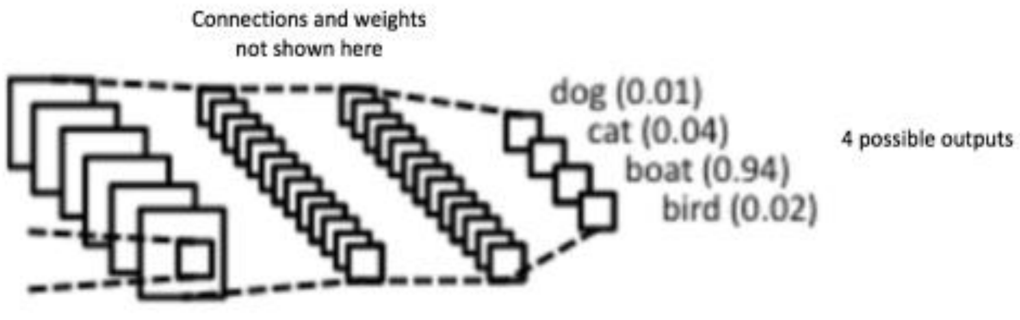


Figure 24: Fully Connected layers- each node is connected to every other node in the adjacent layer. Source [33]

Apart from classification, adding a fully-connected layer is also a cheap way of learning non-linear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better.

The sum of output probabilities from the Fully Connected Layer is 1. This is ensured by using the Softmax as the activation function in the output layer of the Fully Connected Layer. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one.

Weight Initializers

Initializations define the way to set the initial random weights of CNN layers. Some of the initializers that are used to initialize kernel and bias weights are :

1. All zeros

This is generally a bad idea because in this case all the neuron will generate the same output initially and similar gradients would flow back in back-propagation

The results are generally undesirable as network won't train properly.

2. Gaussian Random Variables

The weights can be initialized with random gaussian distribution of 0 mean and small standard deviation (0.1 to 1e-5)

This works for shallow networks, i.e. ~5 hidden layers but not for deep networks.

In case of deep networks, the small weights make the outputs small and as we move towards the end, the values become even smaller. Thus the gradients will also become small resulting in gradient killing at the end.

We will need to play with the standard deviation of the gaussian distribution which works well for your network.

3. Xavier Initialization

It suggests that variance of the gaussian distribution of weights for each neuron should depend on the number of inputs to the layer.

The recommended variance is square root of inputs.

Dropout Layers

They have a very specific function in neural networks. These layers are added to solve the problem of overfitting, where after training, the weights of the network are so tuned to the training examples they are given that the network doesn't perform well when given new examples. The idea of dropout is simplistic in nature.

This layer drops out a random set of activations in that layer by setting them to zero. It forces the network to be redundant. It means the network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out. It makes sure that the network isn't getting too fitted to the training data and thus helps alleviate the overfitting problem.

This layer is only used during training, and not during test time.

Overall Process :

The Convolution + Pooling layers act as Feature Extractors from the input image while Fully Connected layer acts as a classifier.

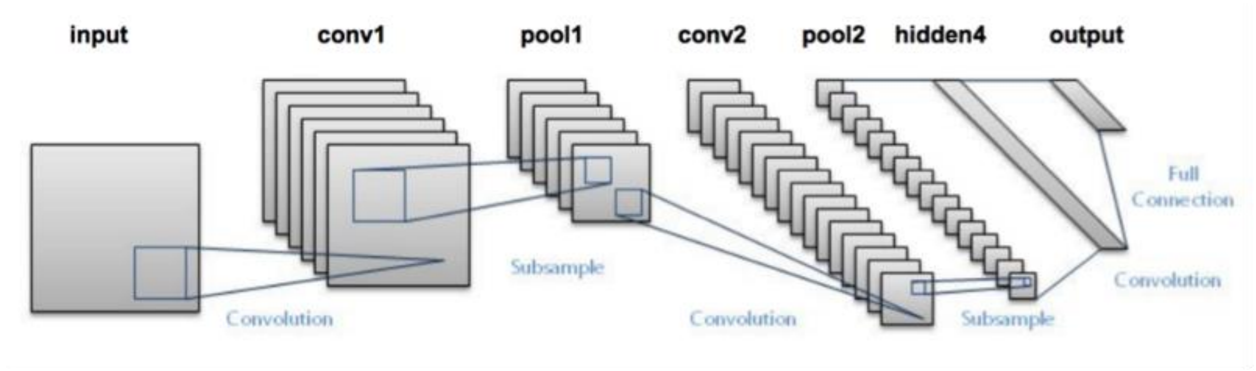


Figure 25: Complete CNN Process. Source [34]

- An input image is passed to the first convolutional layer. The convoluted output is obtained as an activation map. The filters applied in the convolution layer extract relevant features from the input image to pass further.
- Each filter shall give a different feature to aid the correct class prediction. In case we need to retain the size of the image, we use same padding(zero padding), other wise valid padding is used since it helps to reduce the number of features.
- Pooling layers are then added to further reduce the number of parameters
- Several convolution and pooling layers are added before the prediction is made. Convolutional layer help in extracting features. As we go deeper in the network more specific features are extracted as compared to a shallow network where the features extracted are more generic.
- The output layer in a CNN as mentioned previously is a fully connected layer, where the input from the other layers is flattened and sent so as the transform the output into the number of classes as desired by the network.

- The output is then generated through the output layer and is compared to the output layer for error generation. A loss function is defined in the fully connected output layer to compute the mean square loss. The gradient of error is then calculated.
- The error is then backpropagated to update the filter(weights) and bias values.
- One training cycle is completed in a single forward and backward pass.

When a new (unseen) image is input into the ConvNet, the network would go through the forward propagation step and output a probability for each class (for a new image, the output probabilities are calculated using the weights which have been optimized to correctly classify all the previous training examples). If the training set is large enough, the network will (hopefully) generalize well to new images and classify them into correct categories.

Training & Testing of CNN

The way the model is able to adjust its filter values (or weights) is through a training process called **backpropagation**.

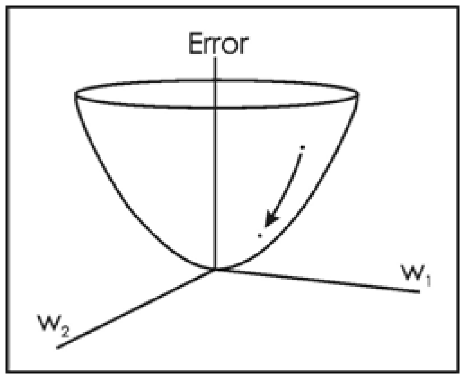
Backpropagation can be separated into 4 distinct sections :

Forward pass : We take a training image and pass it through the whole network. Since all of the weights or filter values were randomly initialized, the output that doesn't give preference to any number in particular. The network, with its current weights, isn't able to look for those low level features or thus isn't able to make any reasonable conclusion about what the classification might be.

Loss function : A loss function can be defined in many different ways but a common one is MSE (mean squared error), which is ½ times (actual - predicted) squared.

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Figure 26: Loss function. Source [35]



One way of visualizing this idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves trying to adjust the weights so that the loss decreases. In visual terms, we want to get to the lowest point in our bowl shaped object. To do this, we have to take a derivative of the loss (visual terms: calculate the slope in every direction) with respect to the weights.

Figure 27: Loss minimization, weights adjustment. Source [35]

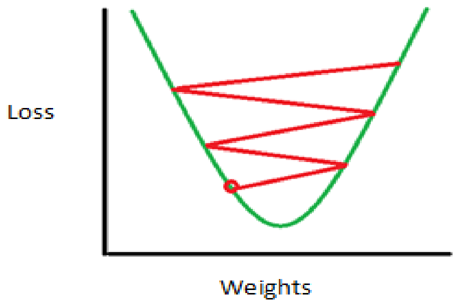
This is the mathematical equivalent of a **dL/dW** where W are the weights at a particular layer.

Backward pass : We determine which weights contributed most to the loss and finding ways to adjust them so that the loss decreases. Once we compute this derivative, we then go to the last step which is the weight update.

Weight update : Here we take all the weights of the filters and update them so that they change in the opposite direction of the gradient.

LEARNING RATE :

It is a parameter that is manually chosen. A high learning rate means that bigger steps are taken in the weight updates and thus, it may take less time for the model to converge on an optimal set of weights. However, a learning rate that is too high could result in jumps that are too large and not precise enough to reach the optimal point.



Consequence of a high learning rate where the jumps are too large and we are not able to minimize the loss.

Figure 28: Effect of high learning rate. Source [35]

The process of forward pass, loss function, backward pass, and weight update is one training iteration. The program will repeat this process for a fixed number of iterations for each set of training images (commonly called a **batch**). Once we finish the parameter update on the last training example,the network should be trained well enough so that the weights of the layers are tuned correctly.

Testing

To see whether or not our CNN works, we have a different set of images and pass the images through the CNN.

Layer Sizing Patterns

I) The input layer that contains the image should be divisible by 2 many times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512.

II) Convolutional layers should be using small filters (e.g. 3x3 or at most 5x5), using a stride of S=1, and crucially, padding the input volume with zeros in such way that the convolutional layer does not alter the spatial dimensions of the input. That is, when F=3, then using P=1 will retain the original size of the input. When F=5,P=2.

It can be seen that P=(F−1)/2P=(F−1)/2 preserves the input size.

III) Pool layers are in charge of downsampling the spatial dimensions of the input. The most common setting is to use max-pooling with 2x2 receptive fields (i.e. F=2), and with a stride of 2 (i.e. S=2). This discards exactly 75% of the activations in an input volume (due to downsampling by 2 in both width and height).

Another slightly less common setting is to use 3x3 receptive fields with a stride of 2. It is very uncommon to see receptive field sizes for max pooling that are larger than 3 because the pooling is then too lossy and aggressive. This usually leads to worse performance.

IV) Smaller strides work better in practice. Stride 1 allows us to leave all spatial downsampling to the POOL layers, with the CONV layers only transforming the input volume depth-wise.

V) If the CONV layers were to not zero-pad the inputs and only perform valid convolutions, then the size of the volumes would reduce by a small amount after each CONV, and the information at the borders would be washed away too quickly.

Other ConvNet Architectures

Convolutional Neural Networks have been around since early 1990s. Some other influential architectures are listed below :

1990s to 2012: In the years from late 1990s to early 2010s convolutional neural network were in incubation. As more and more data and computing power became available, tasks that convolutional neural networks could tackle became more and more interesting.

i) AlexNet (2012) : In 2012, Alex Krizhevsky and others released AlexNet which was a deeper and much wider version of the LeNet and won by a large margin the difficult ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It was a significant breakthrough with respect to the previous approaches and the current widespread application of CNNs can be attributed to this work.

Features :

- Trained the network on ImageNet data, which contained over 15 million annotated images from a total of over 22,000 categories.
- Used ReLU for the nonlinearity functions (Found to decrease training time as ReLUs are several times faster than the conventional tanh function).
- Used data augmentation techniques that consisted of image translations, horizontal reflections, and patch extractions.
- Implemented dropout layers in order to combat the problem of overfitting to the training data.
- Trained the model using batch stochastic gradient descent, with specific values for momentum and weight decay.
- Trained on two GTX 580 GPUs for five to six days.

ii) ZF Net (2013) : The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus. It became known as the ZFNet (short for Zeiler & Fergus Net). It was an improvement on AlexNet by tweaking the architecture hyperparameters.

Features :

- Very similar architecture to AlexNet, except for a few minor modifications.

- AlexNet trained on 15 million images, while ZF Net trained on only 1.3 million images.
- Instead of using 11x11 sized filters in the first layer, ZF Net used filters of size 7x7 and a decreased stride value. The reasoning behind this modification is that a smaller filter size in the first convolutional layer helps to retain a lot of original pixel information in the input volume. A filtering of size 11x11 proved to be skipping a lot of relevant information, especially as this is the first convolutional layer.
- Used ReLUs for their activation functions, cross-entropy loss for the error function, and trained using batch stochastic gradient descent.
- Trained on a GTX 580 GPU for twelve days.
- Developed a visualization technique named Deconvolutional Network, which helps to examine different feature activations and their relation to the input space. It is called 'deconvnet' because it maps features to pixels (the opposite of what a convolutional layer does).

iii) GoogLeNet (2014) : The ILSVRC 2014 winner was a Convolutional Network from Google. Its main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).

Features :

- Used 9 Inception modules in the whole architecture, with over 100 layers in total.
- No use of fully connected layers. They use an average pool instead, to go from a 7x7x1024 volume to a 1x1x1024 volume. This saves a huge number of parameters.
- Uses 12x fewer parameters than AlexNet.
- During testing, multiple crops of the same image were created, fed into the network, and the softmax probabilities were averaged to give us the final solution.
- Utilized concepts from R-CNN for their detection model.
- Trained on a few high-end GPUs within a week.

iv) VGGNet (2014) – The runner-up in ILSVRC 2014 was the network that became known as the VGGNet. Its main contribution was in showing that the depth of the network (number of layers) is a critical component for good performance.

Features :

- The use of only 3x3 sized filters is quite different from AlexNet's 11x11 filters in the first layer and ZF Net's 7x7 filters. The authors' reasoning is that the combination of two 3x3 convolutional layers has an effective receptive field of 5x5. This in turn simulates a larger filter while keeping the benefits of smaller filter sizes. One of the benefits is a decrease in the number of parameters. Also, with two convolutional layers, we're able to use two ReLU layers instead of one.
- 3 convolutional layers back to back have an effective receptive field of 7x7.
- As the spatial size of the input volumes at each layer decreases, the depth of the volumes increases due to the increased number of filters as we go down the network.
- Number of filters doubles after each maxpool layer. This reinforces the idea of shrinking spatial dimensions, but growing depth.
- Worked well on both image classification and localization tasks. The authors used a form of localization as regression.
- Built model with the Caffe toolbox.
- Used scale jittering as one data augmentation technique during training.
- Used ReLU layers after each convolutional layer and trained with batch gradient descent.
- Trained on 4 Nvidia Titan Black GPUs for two to three weeks**.**

v) ResNets (2015) – Residual Network developed by Kaiming He (and others) was the winner of ILSVRC 2015. ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 2016).

Features :

- Ultra deep.
- 152 layers.
- Interesting note that after only the first *2* layers, the spatial size gets compressed from an input volume of 224x224 to a 56x56 volume.
- Authors claim that a naive increase of layers in plain nets result in higher training and test error.
- The group tried a 1202-layer network, but got a lower test accuracy, presumably due to overfitting.
- Trained on an 8 GPU machine for two to three weeks**.**

vi) DenseNet (August 2016) :

Recently published by Gao Huang (and others), the Densely Connected Convolutional Network has each layer directly connected to every other layer in a feed-forward fashion. The DenseNet has been shown to obtain significant improvements over previous state-of-the-art architectures on five highly competitive object recognition benchmark tasks.

3.4 TOOLS

The following tools are used during this research:

i) Anaconda(Python Distribution)

Anaconda is a freemium open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system conda.

ii) Spyder

Spyder is a powerful interactive development environment for the Python language with advanced editing, interactive testing, debugging and introspection features. Additionally, Spyder is a numerical computing environment thanks to the support of IPython and popular Python libraries such as NumPy, SciPy, or matplotlib.

iii) Keras (with Theano Backend)

Keras is a high level neural networks API, written in Python and capable of running on top of Tensorflow, CNTK or Theano. It was develop with a focus on enabling fast experimentation.

Keras is used as a deep learning library that :

- Allows for easy and fast prototyping.
- Supports both Convolutional networks and recurrent networks as well as combination of both.
- Runs seamlessly on CPU and GPU.

Keras is compatible with : Python 2.7-3.6

Theano is an open source symbolic tensor manipulation framework.

3.5. IMPLEMENTATION

We have used following CNN architectures for our 10 datasets :

1. Jaffe database

Total images : 213

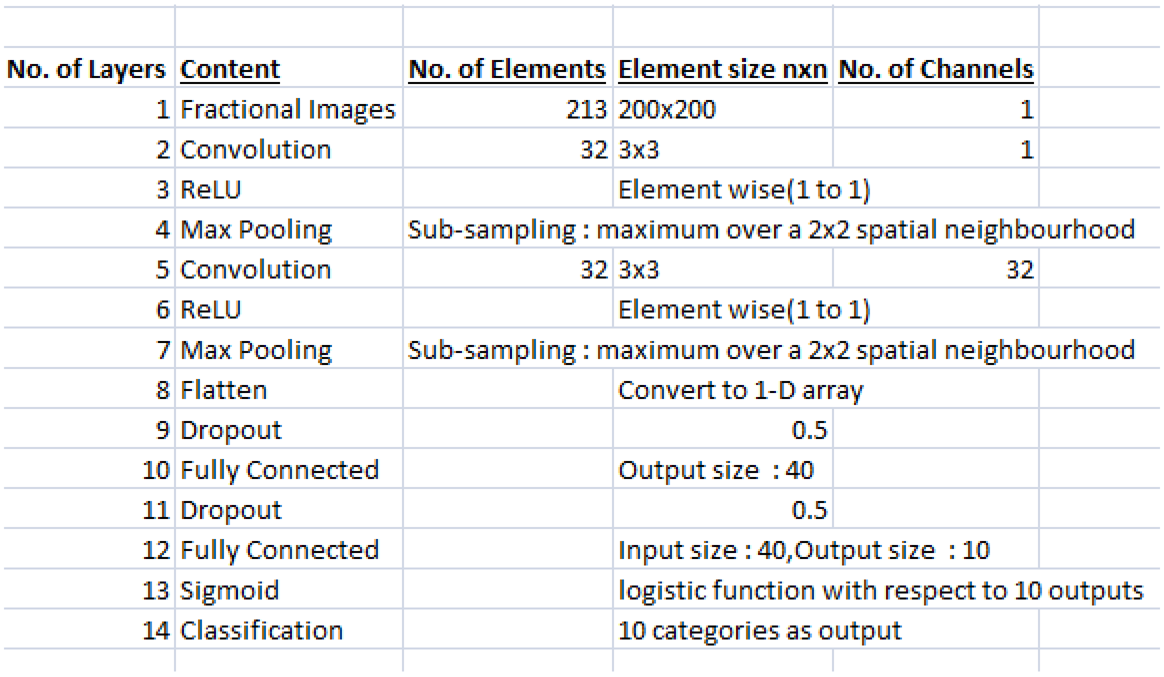
batch size : 5

epochs : 100

image size : 200x200

model : Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<- Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<-Flatten<-Dropout 0.5<-Fully Connected layer with output 40<-Dropout 0.5<-Fully Connected with output 10

In tabular format :



No. of Layers	Content	No. of Elements	Element size nxn	No. of Channels
1	Fractional Images	213	200x200	1
2	Convolution	32	3x3	1
3	ReLU		Element wise(1 to 1)	
4	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
5	Convolution	32	3x3	32
6	ReLU		Element wise(1 to 1)	
7	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
8	Flatten		Convert to 1-D array	
9	Dropout		0.5	
10	Fully Connected		Output size : 40	
11	Dropout		0.5	
12	Fully Connected		Input size : 40,Output size : 10	
13	Sigmoid		logistic function with respect to 10 outputs	
14	Classification		10 categories as output	

Table 1: CNN architecture for Jaffe face database

2. MIT-CBCL Face database

Total images : 3240

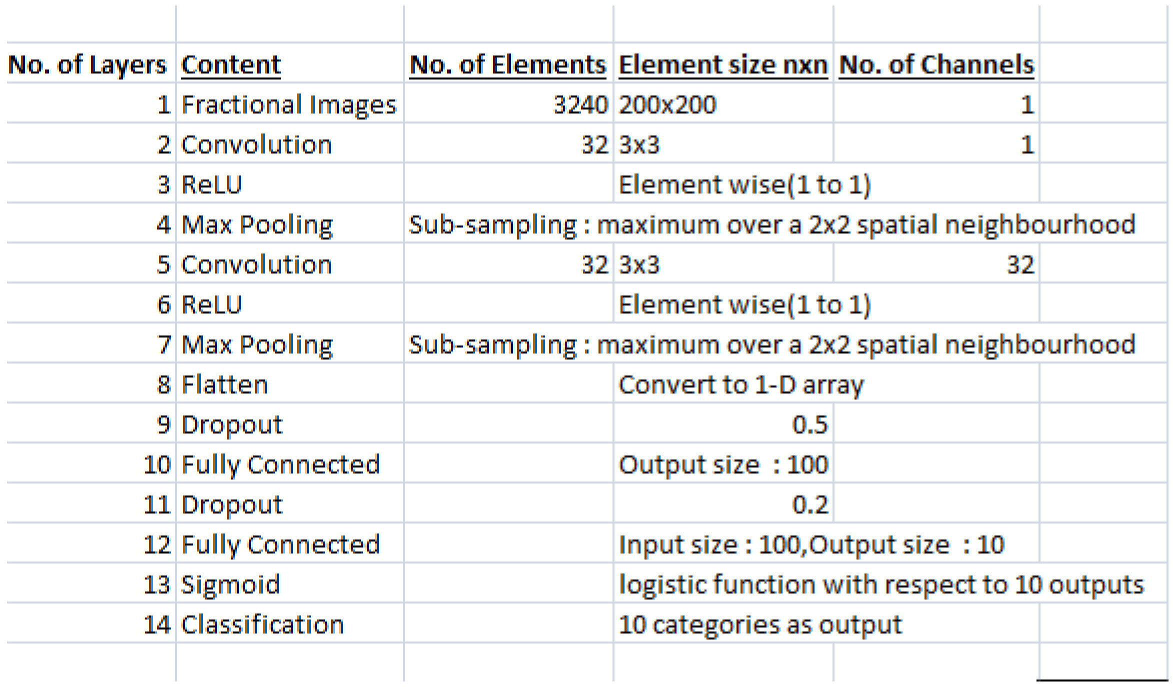
batch size : 15

epochs : 100

image size : 200x200

model : Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<- Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<-Flatten<-Dropout 0.5<-Fully Connected layer with output 100<-Dropout 0.2<-Fully Connected with output 10

In tabular format :



No. of Layers	Content	No. of Elements	Element size nxn	No. of Channels
1	Fractional Images	3240	200x200	1
2	Convolution	32	3x3	1
3	ReLU		Element wise(1 to 1)	
4	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
5	Convolution	32	3x3	32
6	ReLU		Element wise(1 to 1)	
7	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
8	Flatten		Convert to 1-D array	
9	Dropout		0.5	
10	Fully Connected		Output size : 100	
11	Dropout		0.2	
12	Fully Connected		Input size : 100,Output size : 10	
13	Sigmoid		logistic function with respect to 10 outputs	
14	Classification		10 categories as output	

Table 2: CNN architecture for MIT-CBCL face database

3. UMIST – F Face database

Total images : 1012

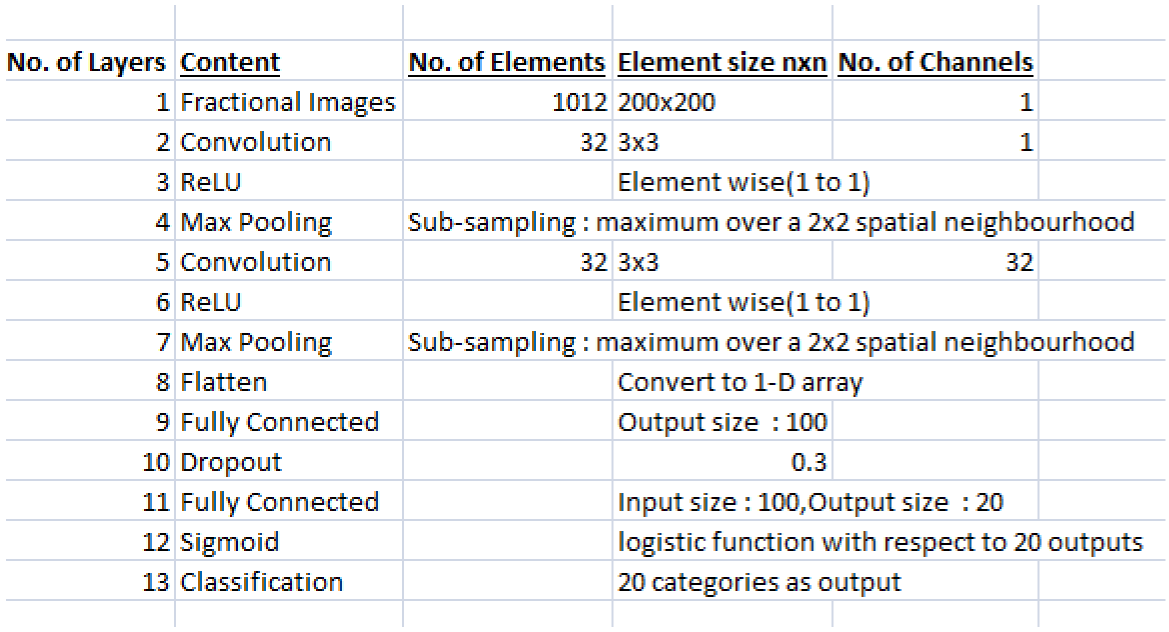
batch size : 5

epochs : 100

image size : 200x200

model : Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<- Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<-Flatten <-Fully Connected layer with output 100<-Dropout 0.3<-Fully Connected with output 20

In tabular format :



No. of Layers	Content	No. of Elements	Element size nxn	No. of Channels
1	Fractional Images	1012	200x200	1
2	Convolution	32	3x3	1
3	ReLU		Element wise(1 to 1)	
4	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
5	Convolution	32	3x3	32
6	ReLU		Element wise(1 to 1)	
7	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
8	Flatten		Convert to 1-D array	
9	Fully Connected		Output size : 100	
10	Dropout		0.3	
11	Fully Connected		Input size : 100,Output size : 20	
12	Sigmoid		logistic function with respect to 20 outputs	
13	Classification		20 categories as output	

Table 3: CNN architecture for UMIST face database

4. **UMIST – Cropped Face database**

Total images : 575

batch size : 5

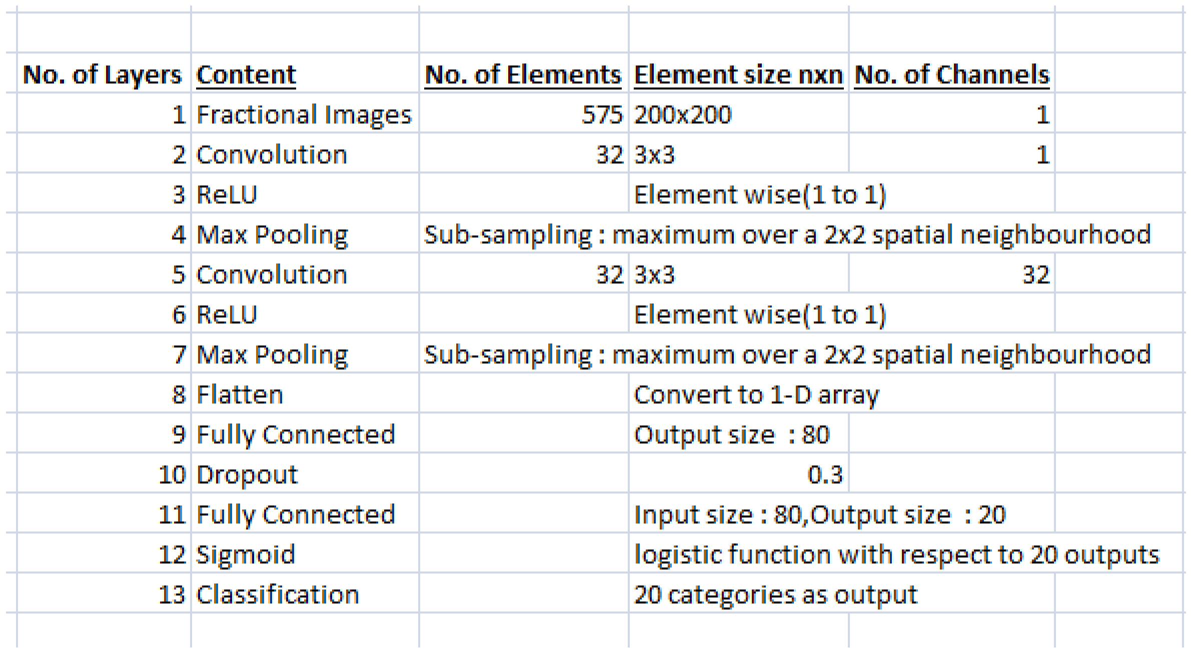
epochs : 100

image size : 200x200

model : Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<- Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with

2x2 filter, stride 2<-Flatten <-Fully Connected layer with output 80<-Dropout 0.3<-Fully Connected with output 20

In tabular format :



No. of Layers	Content	No. of Elements	Element size nxn	No. of Channels
1	Fractional Images	575	200x200	1
2	Convolution	32	3x3	1
3	ReLU		Element wise(1 to 1)	
4	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
5	Convolution	32	3x3	32
6	ReLU		Element wise(1 to 1)	
7	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
8	Flatten		Convert to 1-D array	
9	Fully Connected		Output size : 80	
10	Dropout		0.3	
11	Fully Connected		Input size : 80,Output size : 20	
12	Sigmoid		logistic function with respect to 20 outputs	
13	Classification		20 categories as output	

Table 4: CNN architecture for UMIST-Cropped face database

5. **ORL face database**

Total images : 400

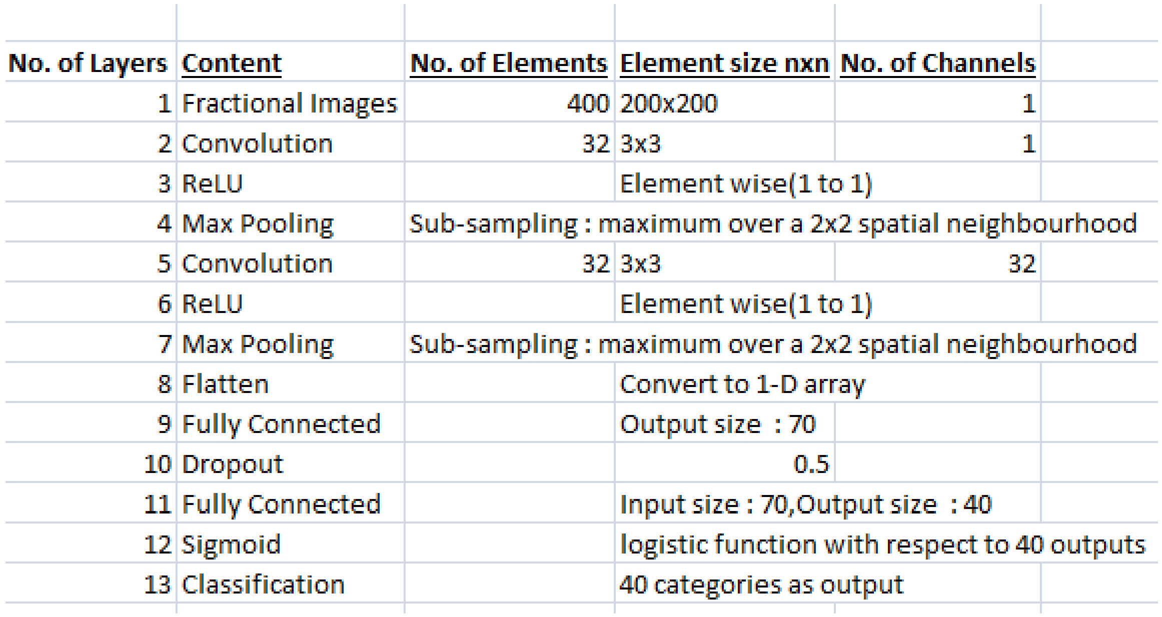
batch size : 2

epochs : 100

image size : 200x200

model : Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<- Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<-Flatten <-Fully Connected layer with output 70<-Dropout 0.5<-Fully Connected with output 40

In tabular format :



No. of Layers	Content	No. of Elements	Element size nxn	No. of Channels
1	Fractional Images	400	200x200	1
2	Convolution	32	3x3	1
3	ReLU		Element wise(1 to 1)	
4	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
5	Convolution	32	3x3	32
6	ReLU		Element wise(1 to 1)	
7	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
8	Flatten		Convert to 1-D array	
9	Fully Connected		Output size : 70	
10	Dropout		0.5	
11	Fully Connected		Input size : 70,Output size : 40	
12	Sigmoid		logistic function with respect to 40 outputs	
13	Classification		40 categories as output	

Table 5: CNN architecture for ORL face database

6. **Caltech Face Database**

Total images : 408

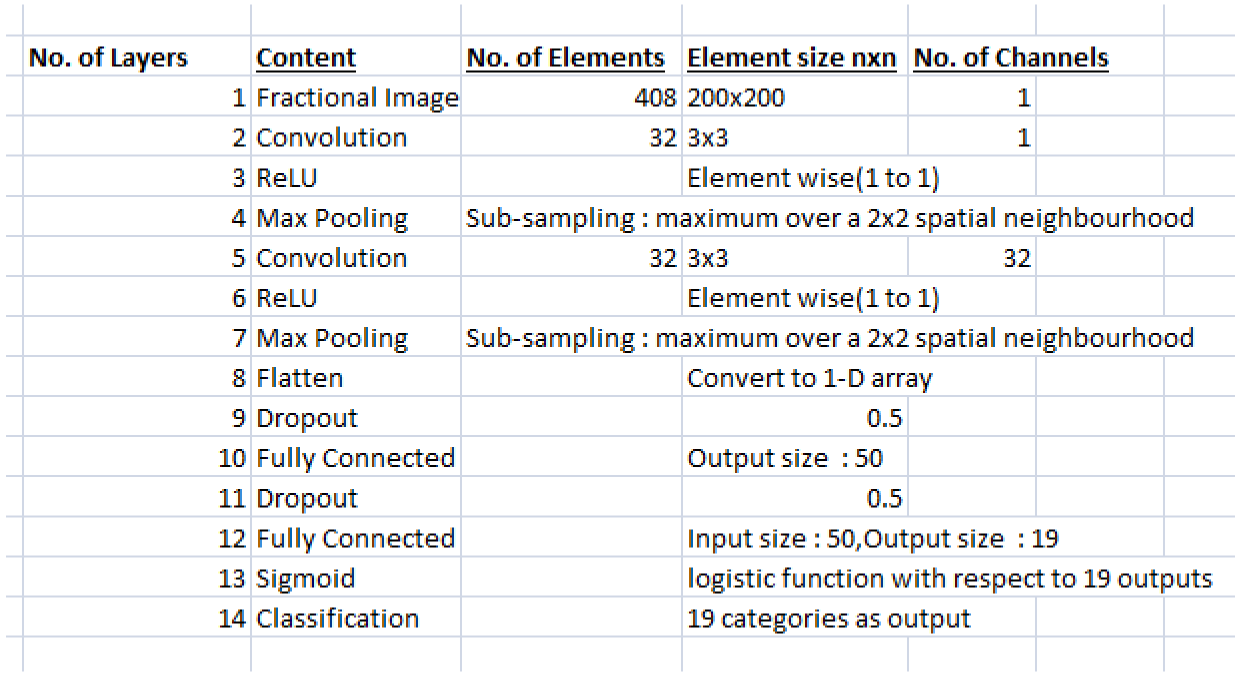
batch size : 10

epochs : 100

image size : 200x200

model : Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<- Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<-Flatten<-Dropout 0.5<-Fully Connected layer with output 50<-Dropout 0.5<-Fully Connected with output 19

In tabular format :



No. of Layers	Content	No. of Elements	Element size nxn	No. of Channels
1	Fractional Image	408	200x200	1
2	Convolution	32	3x3	1
3	ReLU		Element wise(1 to 1)	
4	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
5	Convolution	32	3x3	32
6	ReLU		Element wise(1 to 1)	
7	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
8	Flatten		Convert to 1-D array	
9	Dropout		0.5	
10	Fully Connected		Output size : 50	
11	Dropout		0.5	
12	Fully Connected		Input size : 50,Output size : 19	
13	Sigmoid		logistic function with respect to 19 outputs	
14	Classification		19 categories as output	

Table 6: CNN architecture for Caltech face database

7. **Caltech Face Database (Cropped)**

Total images : 408

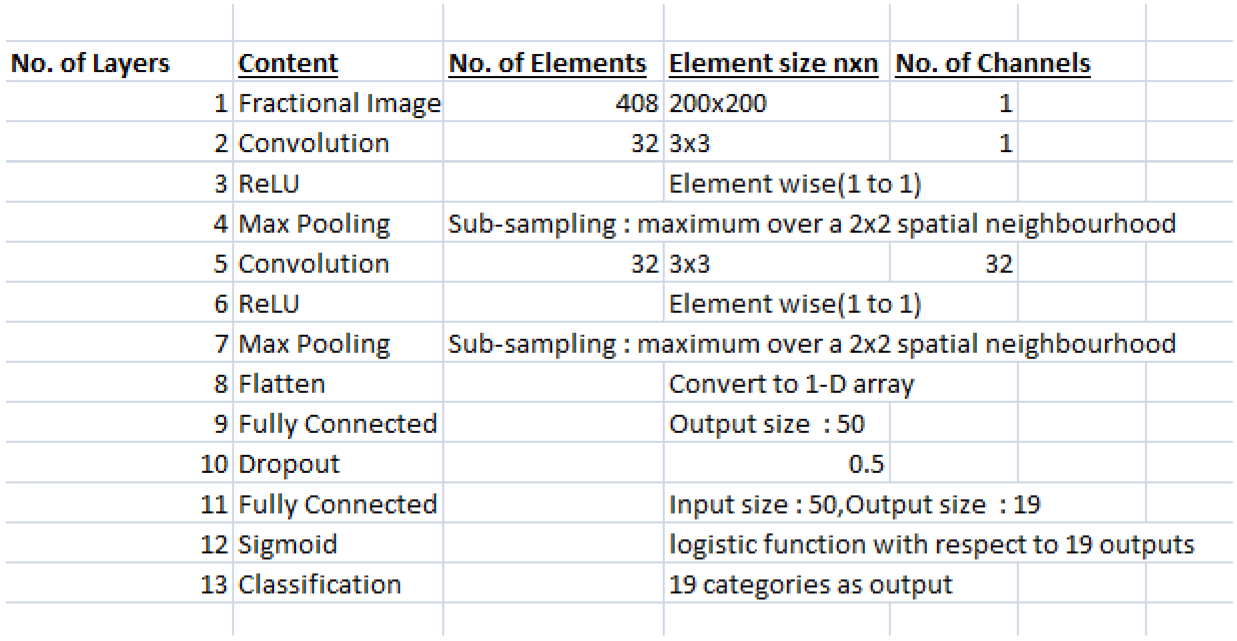
batch size : 10

epochs : 100

image size : 200x200

model : Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<- Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<-Flatten<-Fully Connected layer with output 50<-Dropout 0.5<-Fully Connected with output 19

In tabular format :



No. of Layers	Content	No. of Elements	Element size nxn	No. of Channels
1	Fractional Image	408	200x200	1
2	Convolution	32	3x3	1
3	ReLU		Element wise(1 to 1)	
4	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
5	Convolution	32	3x3	32
6	ReLU		Element wise(1 to 1)	
7	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
8	Flatten		Convert to 1-D array	
9	Fully Connected		Output size : 50	
10	Dropout		0.5	
11	Fully Connected		Input size : 50,Output size : 19	
12	Sigmoid		logistic function with respect to 19 outputs	
13	Classification		19 categories as output	

Table 7: CNN architecture for Caltech Cropped face database

8. Georgia Tech Face Database

Total images : 750

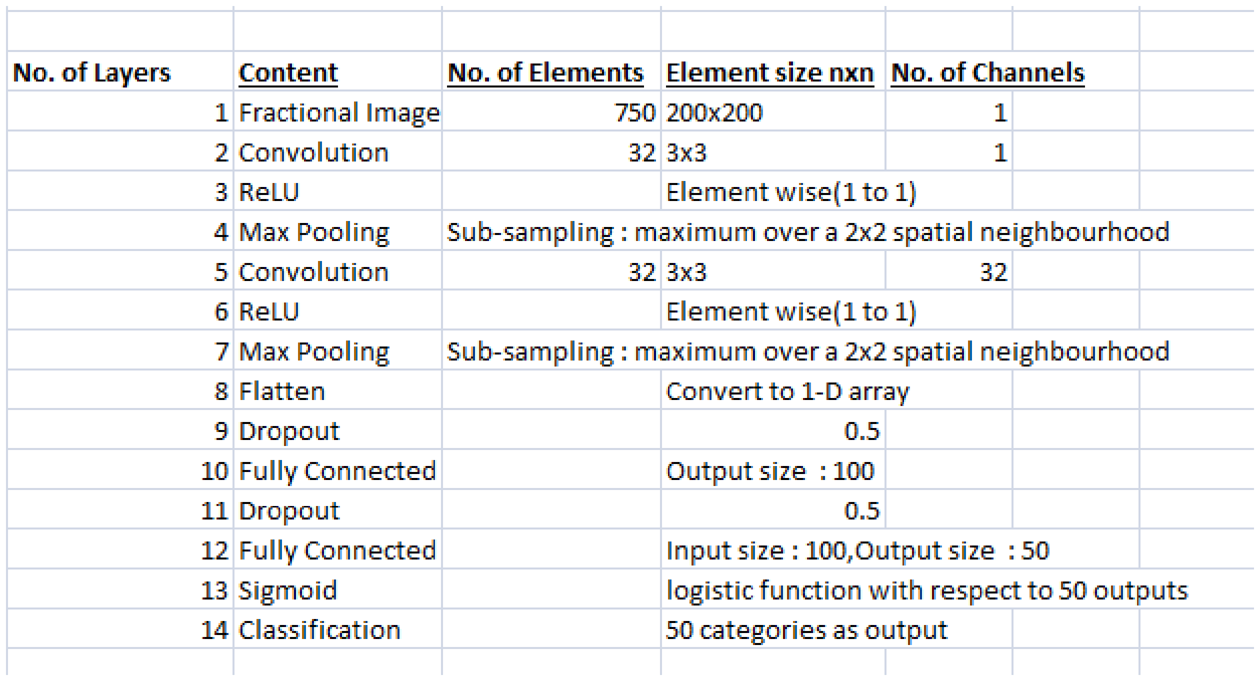
batch size : 5

epochs : 100

image size : 200x200

model : Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<- Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<-Flatten<-Dropout 0.5<-Fully Connected layer with output 100<-Dropout 0.5<-Fully Connected with output 50

In tabular format :



No. of Layers	Content	No. of Elements	Element size nxn	No. of Channels
1	Fractional Image	750	200x200	1
2	Convolution	32	3x3	1
3	ReLU		Element wise(1 to 1)	
4	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
5	Convolution	32	3x3	32
6	ReLU		Element wise(1 to 1)	
7	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
8	Flatten		Convert to 1-D array	
9	Dropout		0.5	
10	Fully Connected		Output size : 100	
11	Dropout		0.5	
12	Fully Connected		Input size : 100,Output size : 50	
13	Sigmoid		logistic function with respect to 50 outputs	
14	Classification		50 categories as output	

Table 6: CNN architecture for Georgia Tech face database

9. Georgia Tech Face Database (Cropped)

Total images : 750

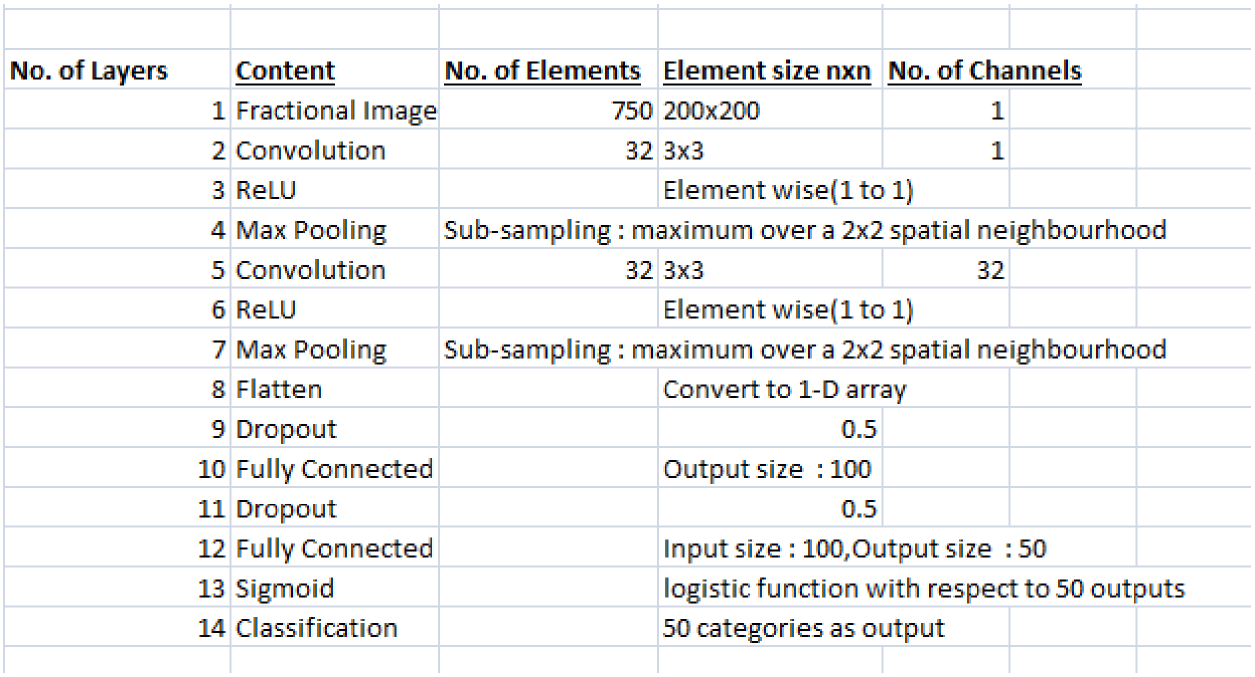
batch size : 5

epochs : 100

image size : 200x200

model : Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<- Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<-Flatten<-Dropout 0.5<-Fully Connected layer with output 50<- Dropout 0.5<-Fully Connected with output 19

In tabular format :



No. of Layers	Content	No. of Elements	Element size nxn	No. of Channels
1	Fractional Image	750	200x200	1
2	Convolution	32	3x3	1
3	ReLU		Element wise(1 to 1)	
4	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
5	Convolution	32	3x3	32
6	ReLU		Element wise(1 to 1)	
7	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
8	Flatten		Convert to 1-D array	
9	Dropout		0.5	
10	Fully Connected		Output size : 100	
11	Dropout		0.5	
12	Fully Connected		Input size : 100,Output size : 50	
13	Sigmoid		logistic function with respect to 50 outputs	
14	Classification		50 categories as output	

Table 9: CNN architecture for Georgia Tech Cropped face database

10. Extended Yale Face Database

Total images : 16378

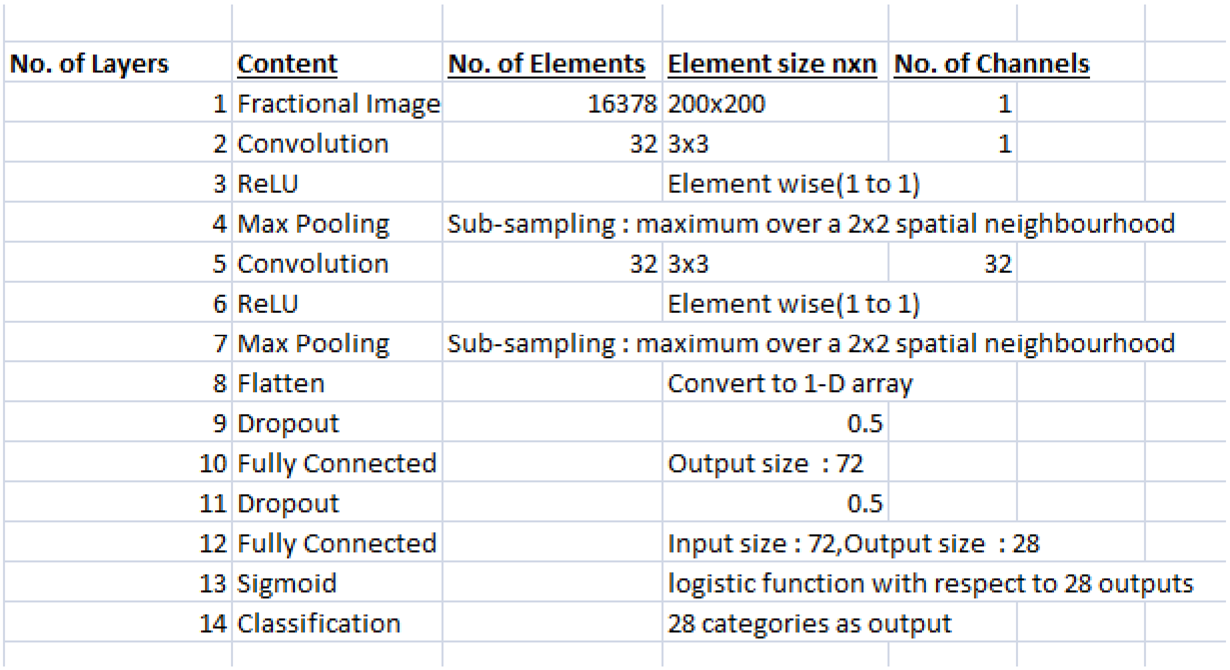
batch size : 64

epochs : 50

image size : 200x200

model : Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<- Conv layer of 32 filters, 3x3 size kernel <-ReLU <-Max pooling layer with 2x2 filter, stride 2<-Flatten<-Dropout 0.5<-Fully Connected layer with output 72<-Dropout 0.5<-Fully Connected with output 28

In tabular format :

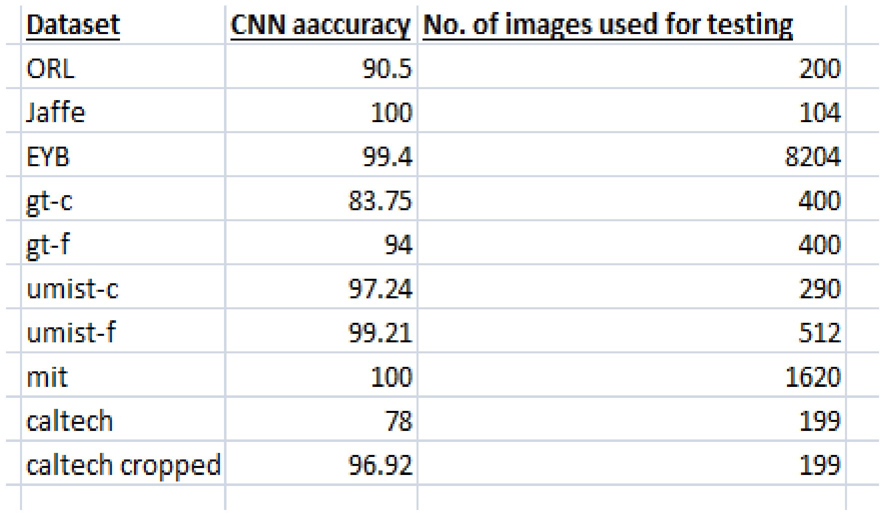


No. of Layers	Content	No. of Elements	Element size nxn	No. of Channels
1	Fractional Image	16378	200x200	1
2	Convolution	32	3x3	1
3	ReLU		Element wise(1 to 1)	
4	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
5	Convolution	32	3x3	32
6	ReLU		Element wise(1 to 1)	
7	Max Pooling	Sub-sampling : maximum over a 2x2 spatial neighbourhood		
8	Flatten		Convert to 1-D array	
9	Dropout		0.5	
10	Fully Connected		Output size : 72	
11	Dropout		0.5	
12	Fully Connected		Input size : 72,Output size : 28	
13	Sigmoid		logistic function with respect to 28 outputs	
14	Classification		28 categories as output	

Table 10: CNN architecture for Extended Yale B face database

CHAPTER 4 RESULTS & DISCUSSION

1. **Accuracy comparison of different face datasets** :



Dataset	CNN aaccuracy	No. of images used for testing
ORL	90.5	200
Jaffe	100	104
EYB	99.4	8204
gt-c	83.75	400
gt-f	94	400
umist-c	97.24	290
umist-f	99.21	512
mit	100	1620
caltech	78	199
caltech cropped	96.92	199

Table 11: Accuracy of different face databases

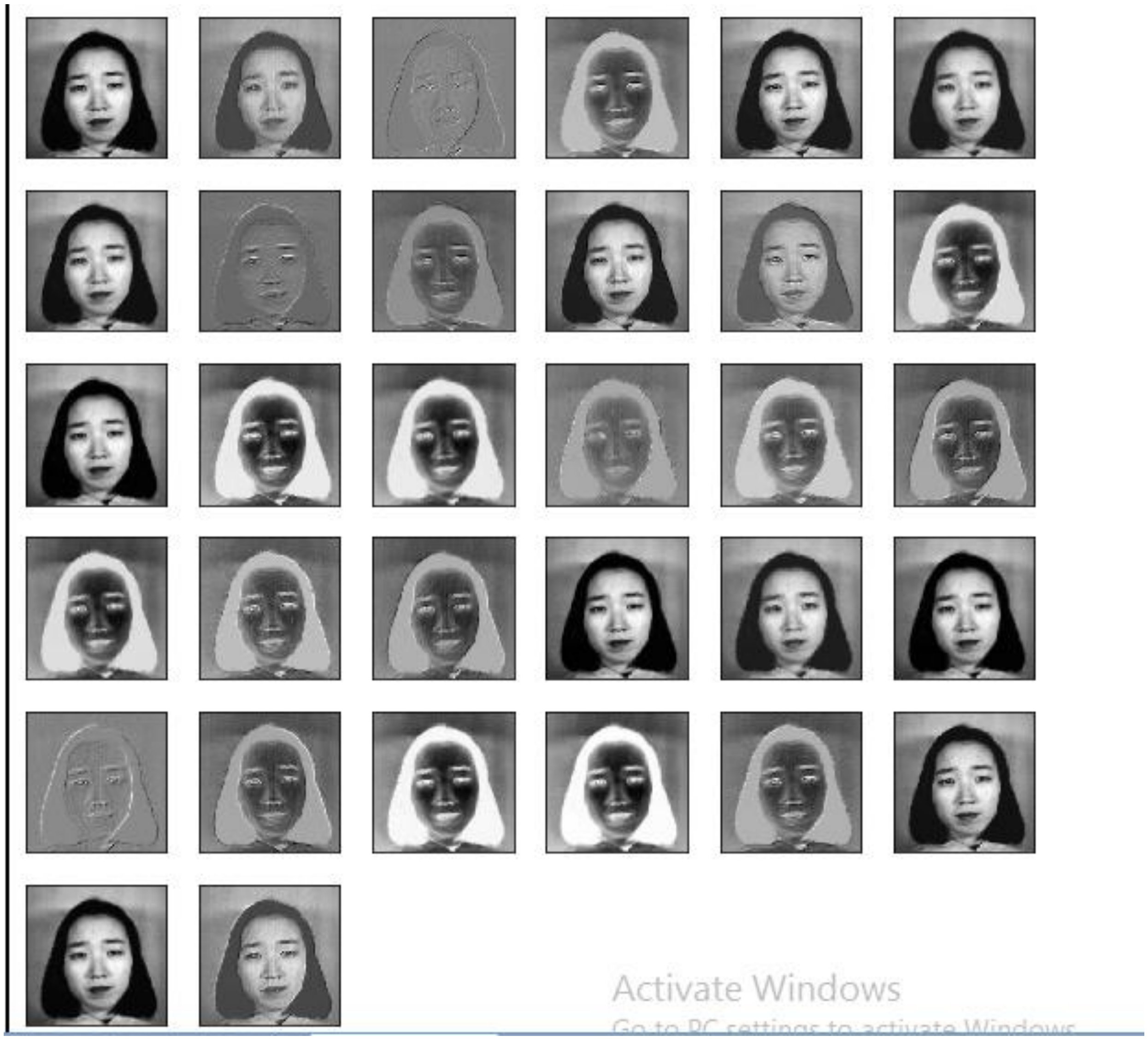
From the above table, we can say that we have achieved more than 90% accuracy in 8 datasets namely : ORL, Extended Yale B, Georgia Tech, UMIST and Caltech face databases.

For MIT and JAFFE dataset, we have achieved 100% accuracy.

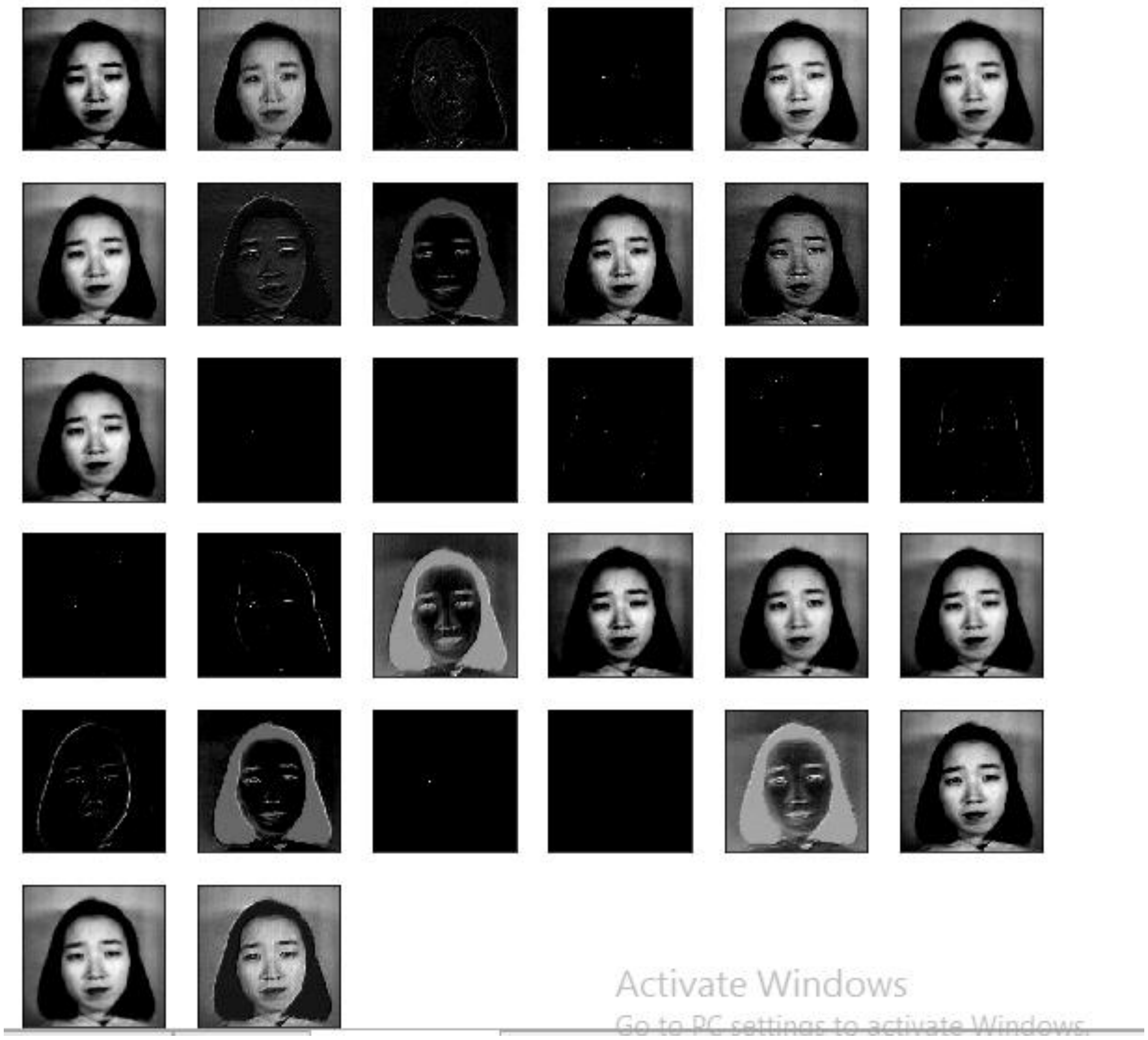
2. **Visualization of CNN layers**

Here we have taken the example of one of the image of Jaffe database to show the changes in the image after each layer of CNN model.

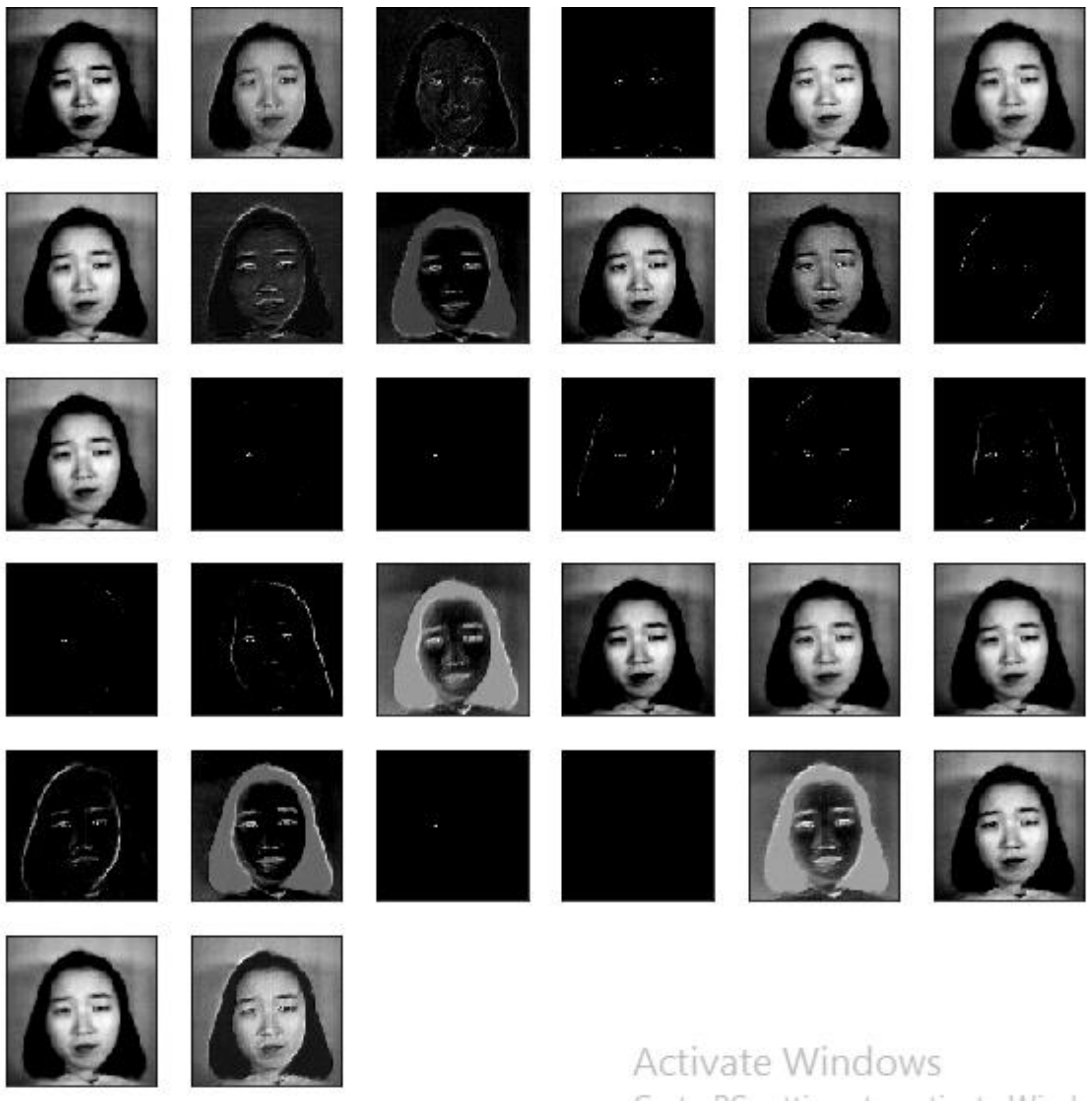
i) Output after 1st Convolutional Layer



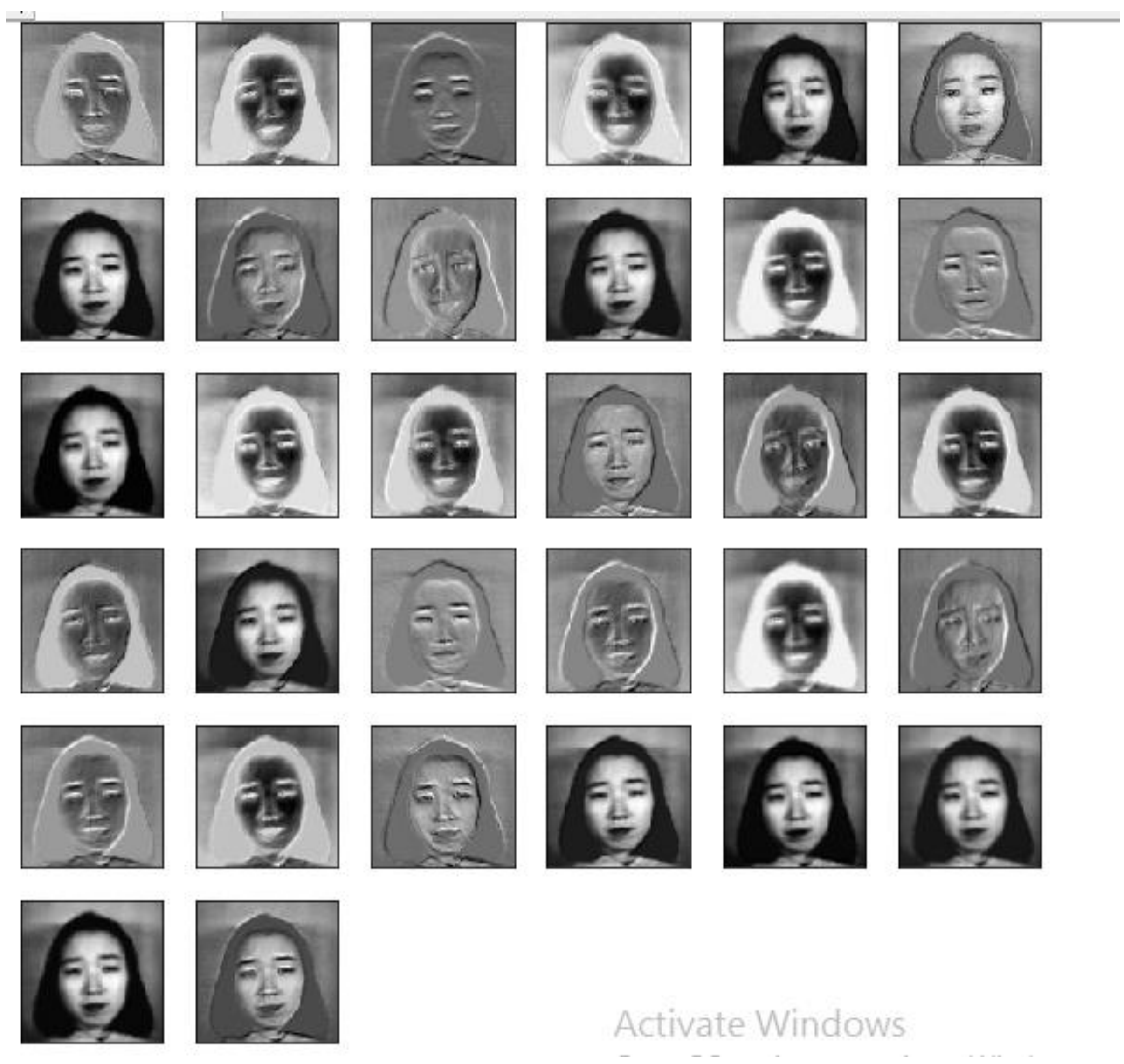
ii) Output after 1st Relu Layer



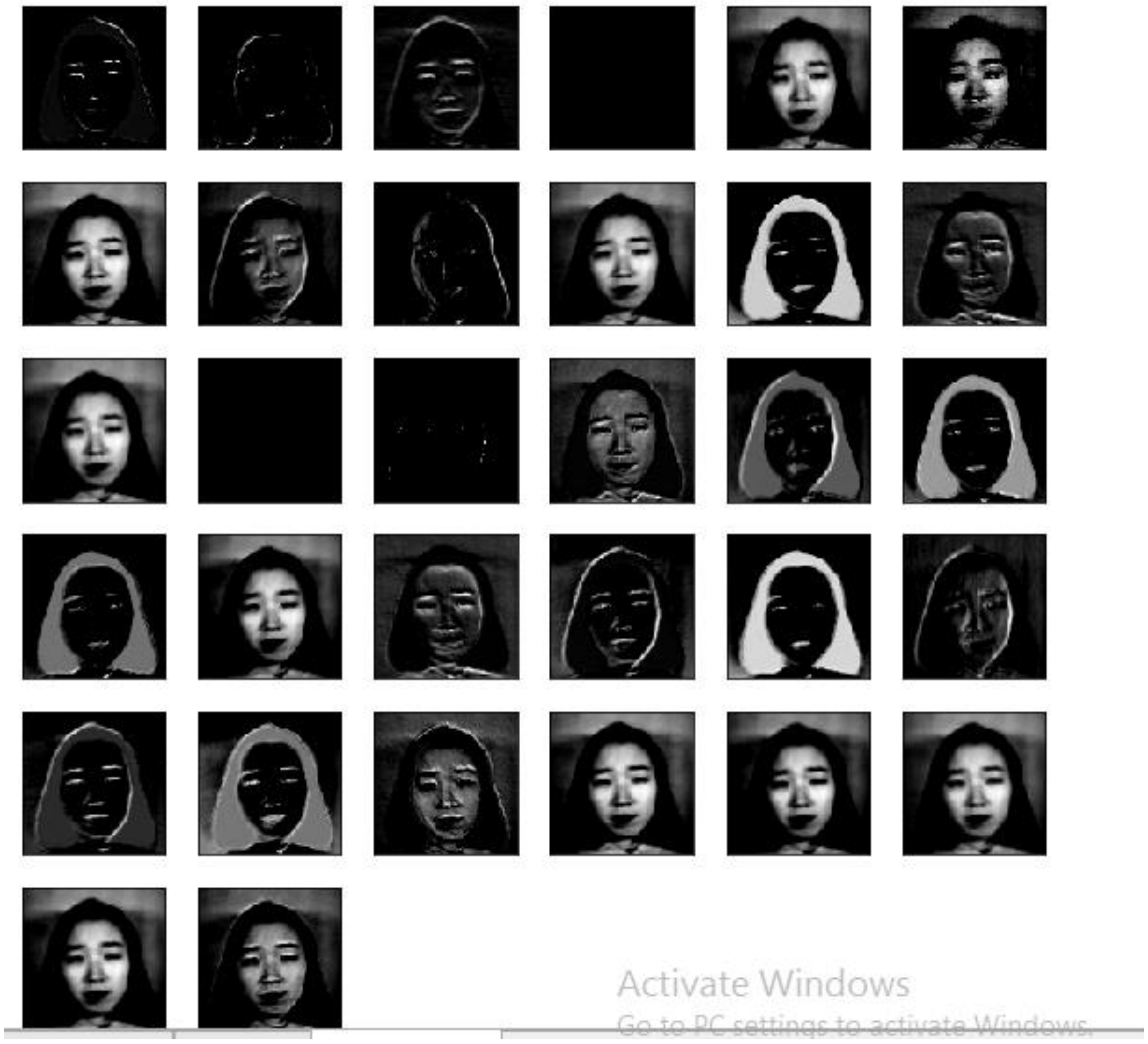
iii) Output after 1st Pooling Layer



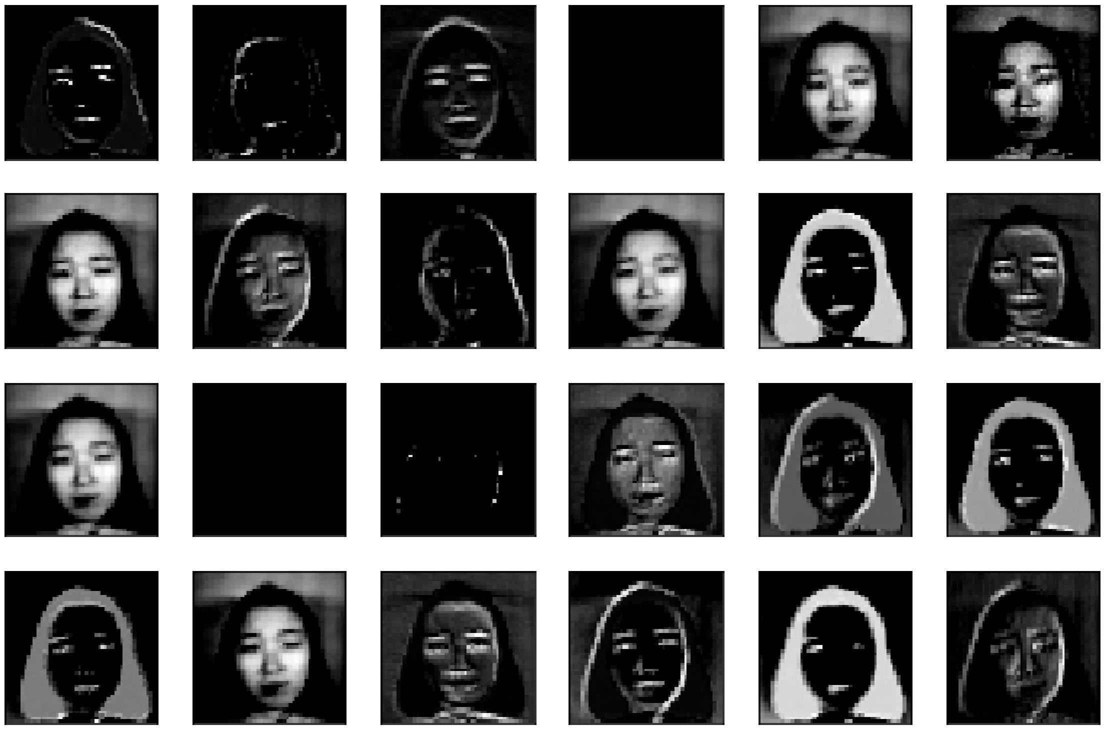
iv) Output after 2nd Convolutional Layer



v) Output after 2nd Relu layer



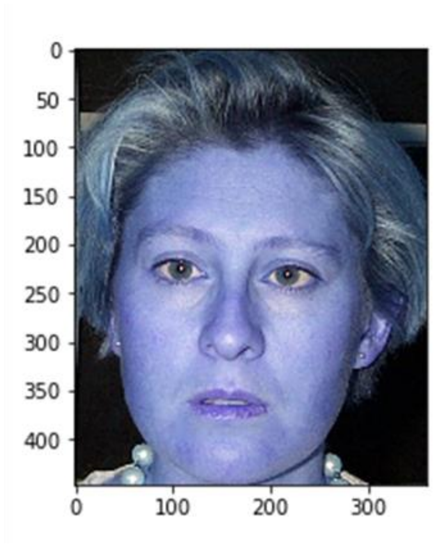
vi) Output after 2nd Pooling layer



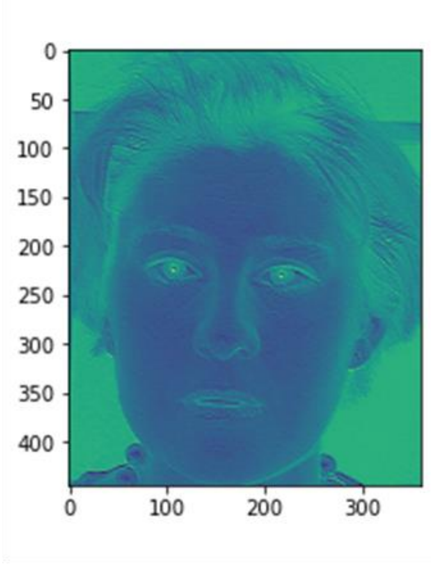
3. **Visualization of combined effect of Convolutional and Pooling layer**

We have taken the example of one of the image of Caltech face database to visualize the combined effect of layers of CNN model.

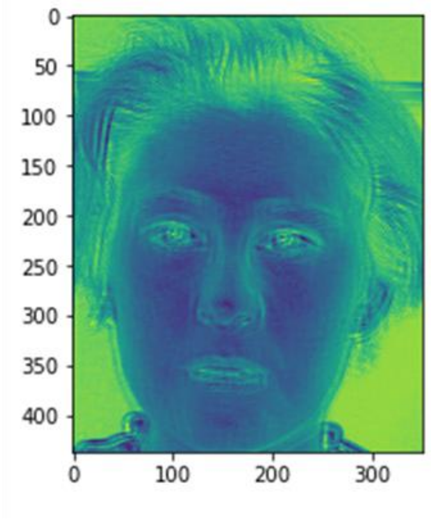
i) Original image



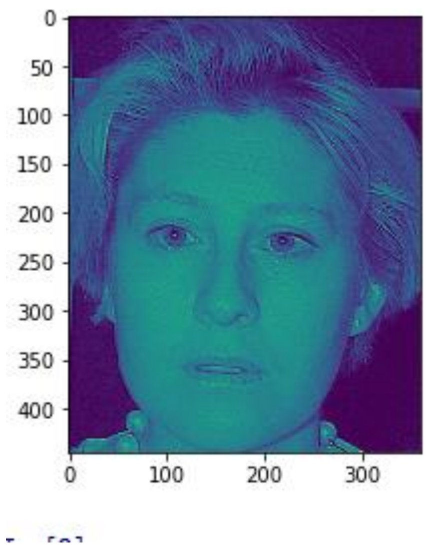
ii) After Convolution layer – 1 filter, filter size : 3x3



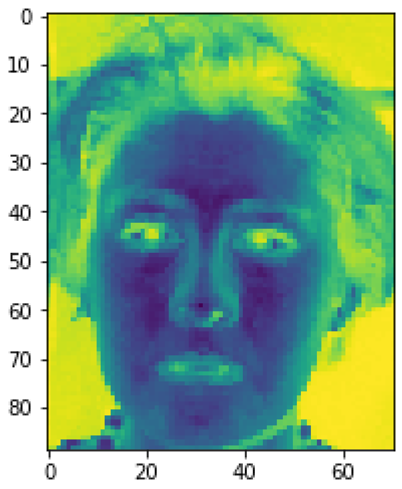
iii) After Convolution layer – 1 filter, filter size : 11x11



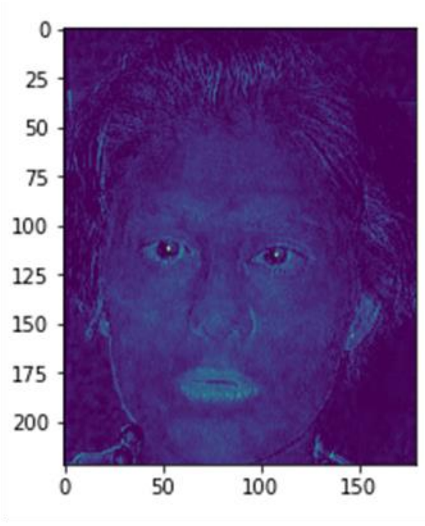
iv) Combined effect of Convolutional and ReLU



v) Combined effect of Convolutional and Pooling



vi) Combined effect of Convolutional, ReLU and Pooling layer


0
25
50
75
100
125
150
175
200
0
50
100
150

CHAPTER 5 CONCLUSION

We presented a evaluation of CNN based face recognition system. Specifically, we evaluated the impact of different architectures and implementation choices of CNNs on different face databases.

The results demonstrated that deep CNNs are capable of learning facial characteristics very well as compared to other existing methods.We developed a Face Recognition system that can work with any kind of images, and is reasonably robust to changes in face expression or orientation, light conditions and other factors.

We extensively tested our system with different datasets and different parameters combinations have been tried. We have obtained results around the 90% of accuracy for most of the datasets and 100% accuracy for 2 datasets. These results are better than the ones we expected, and they allow to use this to solve real life problems.

CHAPTER 6 REFERENCES

1. D.R.Hush, B.G.Horne (1993). Progress in Supervised Neural Networks. IEEE Signal Processing Magazine (Volume: 10, Issue: 1, Jan. 1993). https://doi.org/ 10.1109/79.180705

2. R.Brunelli, T.Poggio (1993). Face Recognition : features versus templates. IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 15, Issue: 10, Oct 1993). https://doi.org/ 10.1109/34.254061

3. Ritwik Kumar, Arunava Banerjee, Baba C. Vemuri, Hanspeter Pfister (2011). TrainableConvolution Filters and their application to Face Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 34, Issue: 7, July 2012). https://doi.org/ 10.1109/TPAMI.2011.225

4. Haoxiang Li, Zhe Lin, Xiaohui Shen, Gang Hua (2015). A Convolutional Neural Network Cascade for Face Detection. Proc. IEEE Conf. on Computer Vision and Pattern Recognition. https://doi.org/ 10.1109/CVPR.2015.7299170

5. Syafeeza, A. R. Khalil-Hani, M. Liew, S. S. Bakhteri, (2014). Convolutional neural network for face recognition with pose and illumination variation. International Journal of Engineering and Technology, 6(1), 44-57.

6. Guosheng Hu, Yongxin Yang, Dong Yi, Josef T Kittler, William J. Christmas, Stan Z. Li, Timothy M. Hospedales (Dec, 2015). When Face Recognition meets with Deep Learning : An Evaluation of Convoluional Neural Networks for Face Recognition. IEEE International Conference on Computer Vision Workshop (ICCVW). https://doi.org/10.1109/ICCVW.2015.58

7. Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman (Jan, 2015). Deep Face Recognition. Published in Conference: British Machine Vision Conference 2015. https://doi.org/10.5244/C.29.41

8. Erhan, D., Bengio, Y., Courville, A., Vincent, P. : Visualizing higher layer features of a deep network. Technical Report, University of Montreal(2009).

9. Donahue, J.Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T. : DeCAF : A deep convolutional activation feature for generic visual recognition. arXiv : 1310.1531

10. Krizhevsky, A., Sutskever, I., Hinton, G. : ImageNet classification with deep convolutional neural networks. In : NIPS(2012).

11. Howard, A.G. : Some improvements on deep convolutional neural network based image classification. arXiv 1312.5402(2013).

12. Hinton, G.E., Osindero, S., Teh, Y. : A fast learning algorithm for deep belief nets. Neural Computation 18,1527-1554(2006).

13. M. D. Zeiler and R. Fergus. Visualizing and understandingconvolutional networks. In Computer Vision–ECCV 2014,pages 818–833. Springer, 2014.

14. Y. Sun, D. Liang, X. Wang, and X. Tang. Deepid3: Face recognition with very deep neural networks. arXiv preprint arXiv:1502.00873, 2015.

15. Y. Sun, X. Wang, and X. Tang. Deep learning face representation from predicting 10,000 classes. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 1891–1898. IEEE, 2014.

16. http://neuralnetworksanddeeplearning.com/

17. http://deeplearning.net/tutorial/

18. http://deeplearning.stanford.edu/tutorial/

19. http://cs231n.github.io/convolutional-networks/

20. http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

21. http://xrds.acm.org/blog/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/

22. https://www.analyticsvidhya.com/blog/2016/04/deep-learning-computer-vision-introduction-convolution-neural-networks/

23. http://www.nptelvideos.in/2012/12/neural-networks-and-applications.html

24. http://deeplearning.net/tutorial/mlp.html

25. https://hackernoon.com/overview-of-artificial-neural-networks-and-its-applications-2525c1addff7

26. http://www.cs.sun.ac.za/~kroon/courses/machine_learning/lecture5/mlp.pdf

27. https://www.coursera.org/learn/neural-networks

28. http://www.deeplearningbook.org/

29. https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

30. https://brilliant.org/wiki/backpropagation/

31. https://www.kdnuggets.com/2016/01/seven-steps-deep-learning.html

32. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

33. http://cs231n.github.io/convolutional-networks/

34. https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

35. https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

CHAPTER 7 APPENDIX 1

1. **Artificial Neural Networks**

An Artificial Neural Network (ANN) is a computational model that is inspired by the way biological neural networks in the human brain process information.

Neuron :

The basic unit of computation in a neural network is the neuron, often called a **node** or **unit**. It receives input from some other nodes, or from an external source and computes an output. Each input has an associated **weight** (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function **f** (defined below) to the weighted sum of its inputs as shown in figure below:

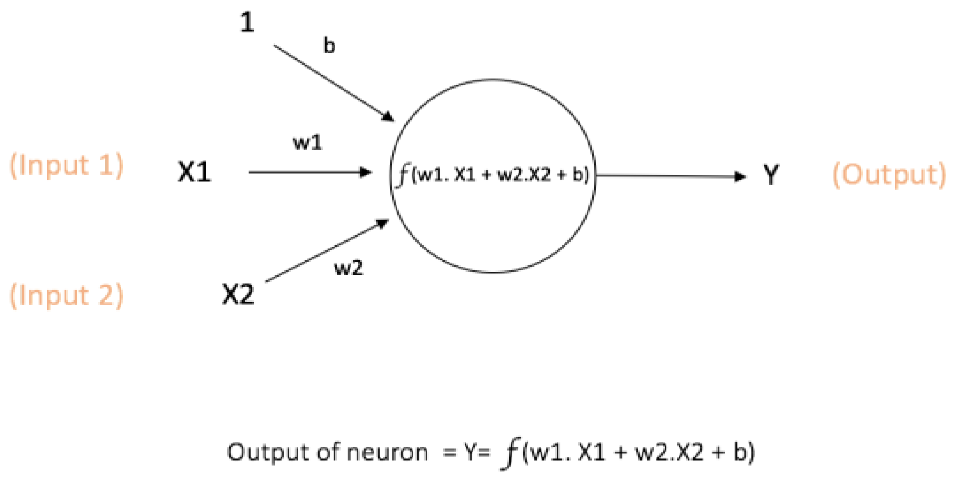


Figure 29: Input and output of neuron

The above network takes numerical inputs **X1** and **X2** and has weights **w1** and **w2** associated with those inputs. Additionally, there is another input **1** with weight **b** (called the **Bias**) associated with it.

The output **Y** from the neuron is computed as shown in the Figure. The function **f** is non-linear and is called the **Activation Function**.

The purpose of the **activation function** is to introduce non-linearity into the output of a neuron. This is important because most real world data is non linear and we want neurons to learn these non linear representations.

Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions we may encounter in practice:

- **Sigmoid:** takes a real-valued input and squashes it to range between 0 and 1

$\sigma(x) = 1 / (1 + \exp(-x))$

- **tanh:** takes a real-valued input and squashes it to the range [-1, 1]

$\tanh(x) = 2\sigma(2x) - 1$

- **ReLU**: ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero (replaces negative values with zero)

$f(x) = \max(0, x)$

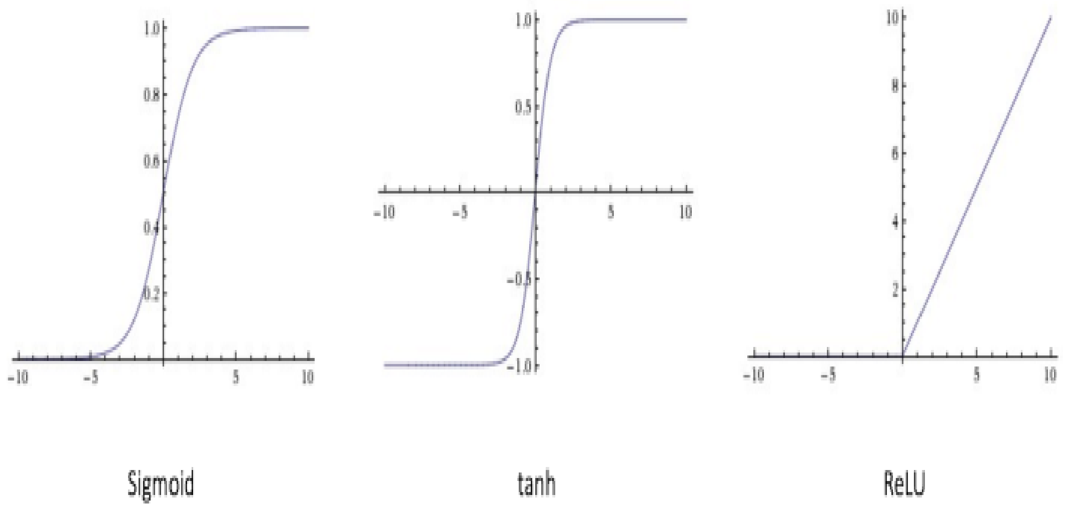


Figure 30: Different activation functions

Importance of Bias: The main function of Bias is to provide every node with a trainable constant value (in addition to the normal inputs that the node receives).

1. **Feedforward Neural Network**

The feedforward neural network was the first and simplest type of artificial neural network devised. It contains multiple neurons (nodes) arranged in **layers**. Nodes from adjacent layers have **connections** or **edges** between them. All these connections have **weights** associated with them.

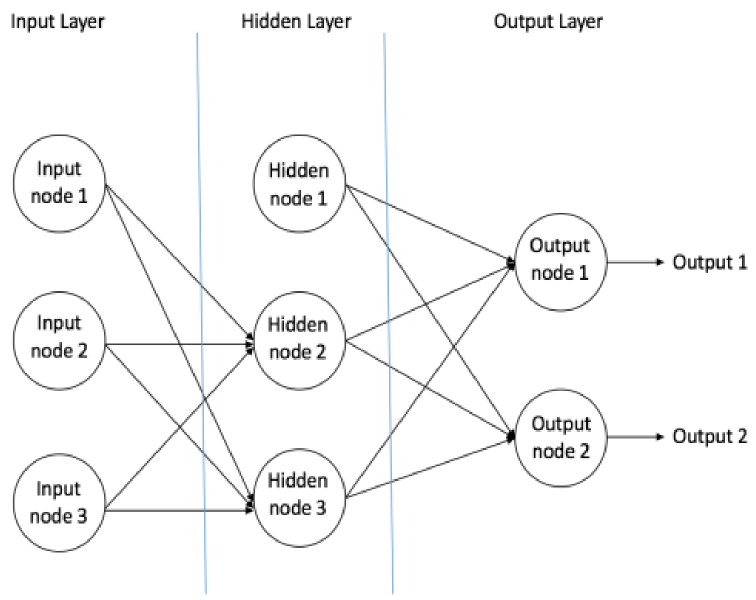


Figure 31: an example of feedforward neural network

A feedforward neural network can consist of three types of nodes:

1. **Input Nodes** – The Input nodes provide information from the outside world to the network and are together referred to as the **Input Layer**. No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes.
2. **Hidden Nodes** – The Hidden nodes have no direct connection with the outside world (hence the name 'hidden'). They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a "Hidden Layer". While a feedforward network will only

have a single input layer and a single output layer, it can have zero or multiple Hidden Layers.

3. **Output Nodes –** The Output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network to the outside world.

In a feedforward network, the information moves in only one direction – forward – from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network (this property of feed forward networks is different from Recurrent Neural Networks in which the connections between the nodes form a cycle).

Two examples of feedforward networks are given below:

i) Single Layer Perceptron – This is the simplest feedforward neural network and does not contain any hidden layer.

ii) Multi Layer Perceptron – A Multi Layer Perceptron has one or more hidden layers.

3. **Multi Layer Perceptron**

A Multi Layer Perceptron (MLP) contains one or more hidden layers (apart from one input and one output layer). While a single layer perceptron can only learn linear functions, a multi layer perceptron can also learn non – linear functions.

Figure 4 shows a multi layer perceptron with a single hidden layer. Note that all connections have weights associated with them, but only three weights (w0, w1, w2) are shown in the figure.

Input Layer: The Input layer has three nodes. The Bias node has a value of 1. The other two nodes take X1 and X2 as external inputs (which are numerical values depending upon the input dataset). No computation is performed in the Input layer, so the outputs from nodes in the Input layer are 1, X1 and X2 respectively, which are fed into the Hidden Layer.

Hidden Layer: The Hidden layer also has three nodes with the Bias node having an output of 1. The output of the other two nodes in the Hidden layer depends on the outputs from the Input layer (1, X1, X2) as well as the weights associated with the connections (edges). Figure 4 shows the output calculation for one of the hidden nodes (highlighted). Similarly, the output from other hidden node can be calculated. **f** refers to the activation function. These outputs are then fed to the nodes in the Output layer.

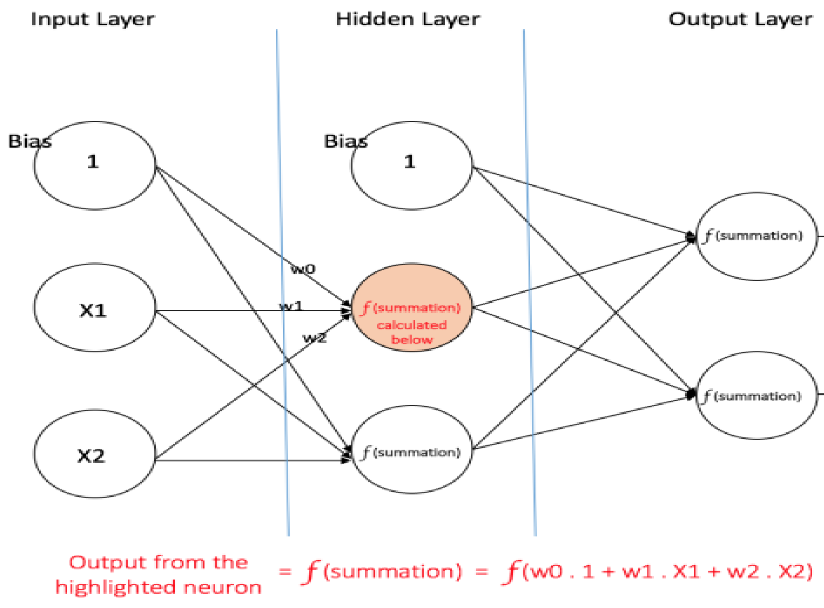

Input Layer
Hidden Layer
Output Layer
Bias
1
Bias
1
X1
X2
w0
w1
w2
f (summation) calculated below
f (summation)
f (summation)
f (summation)
Y1
Y2
Output from the highlighted neuron = f (summation) = f(w0 . 1 + w1 . X1 + w2 . X2)

Figure 32: a multi layer perceptron having one hidden layer

Output Layer: The Output layer has two nodes which take inputs from the hidden layer and perform similar computations as shown for the highlighted hidden node. The values calculated (Y1 and Y2) as a result of these computations act as outputs of the Multi Layer Perceptron.

Given a set of features **X = (x1, x2, …)** and a target **y**, a Multi Layer Perceptron can learn the relationship between the features and the target, for either classification or regression.

4. Training Neural Networks

Data Preparation

We must first prepare your data for training on a neural network.

Neural networks require the input to be scaled in a consistent way. We can **rescale** it to the range between 0 and 1 called normalization. Another popular technique is to standardize it so that the distribution of each column has the mean of zero and the standard deviation of 1.

Scaling also applies to image pixel data.

Full Batch Gradient Descent and Stochastic Gradient Descent

Both variants of Gradient Descent perform the same work of updating the weights of the MLP by using the same updating algorithm but the difference lies in the number of training samples used to update the weights and biases.

Full Batch Gradient Descent Algorithm as the name implies uses all the training data points to update each of the weights once whereas Stochastic Gradient uses 1 or more(sample) but never the entire training data to update the weights once.

Let us understand this with a simple example of a dataset of 10 data points with two weights **w1** and **w2**.

Full Batch**:** We use 10 data points (entire training data) and calculate the change in w1 (Δw1) and change in w2(Δw2) and update w1 and w2.

Stochasic Gradient Descent**:** We use 1st data point and calculate the change in w1 (Δw1) and change in w2(Δw2) and update w1 and w2. Next, when we use 2nd data point, we will work on the updated weights.

The classical and still preferred training algorithm for neural networks is stochastic gradient descent.

This is where one row of data is exposed to the network at a time as input. The network processes the input upward activating neurons as it goes to finally produce an output value. This is called a **forward pass** on the network. It is the type of pass that is also used after the network is trained in order to make predictions on new data.

The output of the network is compared to the expected output and an error is calculated. This error is then propagated back through the network, one layer at a time, and the weights are updated according to the amount that they contributed to the error. This is called the **backpropagation algorithm**.

The process by which a Multi Layer Perceptron learns is called the Backpropagation algorithm.

Backpropagation algorithm

Backward Propagation of Errors, often abbreviated as BackProp is one of the several ways in which an artificial neural network (ANN) can be trained. It is a supervised training scheme, which means, it learns from labeled training data.

An ANN consists of nodes in different layers; input layer, intermediate hidden layers and the output layer. The connections between nodes of adjacent layers have “weights” associated with them. The goal of learning is to assign correct weights for these edges. Given an input vector, these weights determine what the output vector is.

In supervised learning, the training set is labeled. This means, for some given inputs, we know the desired/expected output.

Algorithm :

Initially all the edge weights are randomly assigned. For every input in the training dataset, the ANN is activated and its output is observed. This output is compared with the desired output that we already know, and the error is propagated back to the previous layer. This error is noted and the weights are adjusted accordingly. This process is repeated until the output error is below a predetermined threshold.

Once the above algorithm terminates, we have a learned ANN which, we consider is ready to work with new inputs. This ANN is said to have learned from several examples (labeled data) and from its mistakes (error propagation)

The process is repeated for all of the examples in our training data. One of updating the network for the entire training dataset is called an **epoch**. A network may be trained for tens, hundreds or many thousands of epochs.

Weight Updates

The weights in the network can be updated from the errors calculated for each training example and this is called **online learning**. It can result in fast but also chaotic changes to the network.Alternatively, the errors can be saved up across all of the training examples and the network can be updated at the end. This is called **batch learning** and is often more stable.

Typically, because datasets are so large and because of computational efficiencies, the size of the batch, the number of examples the network is shown before an update is often reduced to a small number, such as tens or hundreds of examples.

The amount that weights are updated is controlled by a configuration parameters called the **learning rate**. It is also called the step size and controls the step or change made to network weight for a given error. Often small weight sizes are used such as 0.1 or 0.01 or smaller.

Momentum is a term that incorporates the properties from the previous weight update to allow the weights to continue to change in the same direction even when there is less error being calculated.

Learning Rate Decay is used to decrease the learning rate over epochs to allow the network to make large changes to the weights at the beginning and smaller fine tuning changes later in the training schedule.

Prediction

Once a neural network has been trained it can be used to make predictions.

We can make predictions on test or validation data in order to estimate the skill of the model on unseen data. We can also deploy it operationally and use it to make predictions continuously.

The network topology and the final set of weights is all that we need to save from the model. Predictions are made by providing the input to the network and performing a forward-pass allowing it to generate an output that we can use as a prediction.

One round of forward and back propagation iteration is known as one training iteration aka **Epoch**.

APPENDIX 2

Visualizing CNN filters

There is a lot of work being done about visualizing what deep learning networks learned.
This in part is due to criticism saying that it’s hard to understand what these black box networks learned, but this is also very useful to debug them.

Many techniques propagating gradients back to the input image became popular lately, like Google’s deep dream, or even the neural artistic style algorithm.

The idea is quite simple: we want to find an input image that would produce the largest output from one of convolution filters in one of the layers. To do that, we can perform back propagation from the output of the filter we’re interested in, back to an input image. That gives us the gradient of the output of the filter with respect to the input image pixels.

We can use that to perform gradient ascent, searching for the image pixels that maximize the output of the filter. The output of the filter is an image. We need to define a scalar score function for computing the gradient of it with respect to the image.One easy way of doing that, is just taking the average output of that filter.

<u>Visualizing Learned Features in Convolutional Neural Networks</u>

Visualization methods consists of 3 classes. They are:

1. Input Modification Methods

Input Modification methods are visualization technique which modify, as the name suggests, the input and measure the resulting changes in the output or intermediate layers of the network.

2. Deconvolutional Methods

In contrast to Input Modification methods, Deconvolutional methods see the network as a white-box and use the network structure itself for their visualizations.

The common denominator amongst the different methods in this class is the idea to determine the contribution of one pixel of the input image by starting with the activation of interest and iteratively computing the contribution of each unit in the next lower layer to this activation.

In this way, by moving backwards through the network until the input layer is reached, the contribution values for each pixel can be obtained, which together form a visualization of the features that are most relevant to the activation of interest.

While Backpropagation uses the information from the original activations of the lower layer to decide which values to mask out during the pass through a ReLU layer, the Deconvnet approach uses the deconvolved activities from the higher layer.

In other words, while Backpropagation uses information from the lower layers and the input image to determine which of the pixel values were important in computing the activations of the higher layers, the Deconvnet approach uses (gradient) information from the higher layers to determine which of the pixels from the input image had a beneficial effect on the activations of the units of interest.

3. Input Reconstruction Methods

It follows the idea that reconstructing an input, which either maximally activates a unit of interest or leads to the same activations as a natural image prior, reveals which filters are important for the associated for the associated features.

They use gradient descent in the input space to iteratively find an image which maximally activates a unit of choice.

To find the gradient, they backpropagate the difference between the actual activation and the desired activation to the input layer without changing the network itself. The resulting image contains all the features that the associated filters selects for.

APPENDIX 3

Some important notes realted to CNN :

How to choose Stride size :

Stride size depends on : the way we combine layers together (convolution filter and pooling) and size of the convolution filter.

1. Size of the convolution filter : Generally people take filter 11×11 with stride 4, 7×7 with stride 2, 3×3 with stride 1. The size of convolution is ability to hold information. If we use filter 11×11 with stride 1, there is almost no difference between 2 neighbor filter.

2. The way we combine layers together : It is easy to see that a stack of two3×3 conv. layers (without spatial pooling in between) has an effective receptive field than 5×5.

3. Stride 1 is better because large overlaps ensure that the model is translational invariant (object could be anywhere in the image).

4. We use stride larger than 1 to reduce the representational dimension and use stride 1 to keep it same for the next layer. As we go deeper, using larger stride values means more lost in representation since the abstraction level goes up with more layers.

5. According to how much information we want to keep from the image to be used for features extraction. The smaller the stride, the more info we keep from the image.

What information do CNNs destroy as the input goes up the layers of the CNN?

Information loss is motivated primarily by the output we're mapping the CNN to, but the first few layers of the network usually learn small, local features and progressively discriminatory elements as we go deeper.

Max-pooling will lose significant location information -we're simply selecting the strongest activation in a grid and and propagating it.

Deconvolution is important here i.e. reverse each of the convolution and pooling operations to return to pixel space to see what actually happens.

It can be seen that as we go deeper into the network, the activations get more discriminatory in nature - i.e. selecting the most relevant information for classification, whereas the activations in the earlier layers are basic filters. Almost all of the redundant low-frequency information is lost in the deeper layers.

They lose positional information. As a result of multiple convolutions and pooling, the position information where inside the input image a particular feature was found is lost.

Spatial information may be lost if max-pooling is used. In the convolution layer, large kernel sizes and large strides may also lead to loss of spatial details.

Defining only a few number of filters in a convolution layer could lead to suppression of information. On the other hand defining too many could be too computationally expensive, potentially redundant or it could lead to overfitting. A reconstruction of the image from the filter responses would help in measuring the loss of information.

Pooling operation throws away location information.High layer filters supposedly represent more task specific input representation

How are the filters for the convolution layer of a CNN built?

They are learnt through the training process with gradient descent optimization. Generally people initialize these filters from Gaussian random noise, and let the model learn the best filters by back-propagation.

Why does each filter learn different features in a CNN?

Random initialisation of the weights will likely ensure each filter converges to different local minima in the cost function.

If we initialize them randomly, and one weight ends up on one side of the hump and the other on the other side, then they will probably end up finding different local optima. That corresponds to discovering two different features.

In CNN, what are the contents of different filters used in the first layer?

Genearlly ,we instantiate it with random weights (close to 0), and then the back-propagation algorithm will gradually change those weights in a direction which gives a slightly better result.

What is the difference between convolution filter with large stride and max pooling layer as both down sample the input layer, when to use each?

The major difference is a convolution filter is extracting features from the matrix of data, whereas the pooling layer is only downsampling the matrix of data.

This means that if we include a large stride in the convolution filter, we are changing the types of features we extract in the algorithm, whereas if we change it in the pooling layer, we are simply changing how much the data is downsampled.

Can we improve the performance of neural networks just by increasing its depth?

If current model is underfitting (and we have enough data) then Yes otherwise No.

Deep residual learning is proposed for this purpose. Namely, with the residual mechanism, going deeper can gain much more improvement comparing the same network structure but without the residual mechanism.

Increasing number of layers improves performance in cases where data is sparse.

ResNets are needed to propagate gradients otherwise we end up with problems when we go this deep.

How can we decide the kernel size and layers of CNN?

We should initially use fewer filters and gradually increase and monitor the error rate to see how it is varying.

Very small filter sizes will capture very fine details of the image. On the other hand having a bigger filter size will leave out minute details in the image.

Deeper networks is always better, at the cost of more data and increased complexity of learning.

Why do we use shared weights in the convolutional layers of CNN?

Eg - If detecting cars is important for our objective, and we don't share weights, the network will have to independently learn the appearance of cars at every distinct image location (that the network processes via strides etc). This is because if we're not sharing weights, a filter thats allocated for the bottom left part of the image, will only see bottom-left image patches.

By sharing weights, we enable the network to learn a single filter for cars no matter where the car appears in the image (translation invariance).

Because we want to capture the same feature at different places in an image.

This is because the convolutional layers has a property, namely translational invariance

What is the motivation for pooling in convolutional neural networks (CNN)?

One benefit of pooling is that we get rid of a lot of data, which means that your computation is less intensive.

The convolutional layer is a stack of feature maps where we have one feature map for each filter. A complicated dataset with many different object categories will require a large number of filters, each responsible for finding a pattern in the image.

More filters mean a bigger stack which means that the dimensionality of our convolutional layers can get quite large. Higher dimensionality means we'll need to use more parameters which can lead to overfitting. Thus, we need a method for reducing this dimensionality so that we can avoid overfitting and this is the role of pooling layers.

Lets consider **translation invariance** since that is a form of invariance gained by using pooling layers. We will define invariance to mean that the class label of the prediction does not change if the image is translated by a small amount.

Suppose our CNN is predicting if the image is of a cat face, and that the last layer of the network (before the final classification layer) produces a dense pixel-wise feature map of the probability of the cat being present at every pixel location. Suppose that the activation is particularly high at one of the pixels, but low everywhere else in the image.

Consider a max pooling layer with pooling region the entire input feature map. Notice that we could translate the image arbitrarily, so long as the high activation pixel remained inside the input feature map, and the result of maxing over the entire feature map would still be the same high-probability prediction of the cat being present. This is the translation invariance that we want.

To summarize, by pooling over a region it doesn't matter where exactly within the pooling region the high activation is because max pooling will ignore everything except that highest activation input. Other forms of pooling (e.g. average pooling) offer similar invariance but allow the other non-maximum parts of the pooling region to contribute more to the output.

The main motivation is to aggregate multiple low-level features in the neighborhood to gain invariance mainly in object recognition.(in case of aggregate pooling).

What is the difference between a convolutional neural network and a multilayer perceptron?

Convolutional Neural Networks are MLPs with a special structure.

CNNs have repetitive blocks of neurons that are applied across space (for images) or time (for audio signals etc).

For images, these blocks of neurons can be interpreted as 2D convolutional kernels, repeatedly applied over each patch of the image.

For speech, they can be seen as the 1D convolutional kernels applied across time-windows. At training time, the weights for these repeated blocks are 'shared', i.e. the weight gradients learned over various image patches are averaged.

The reason for choosing this special structure, is to exploit spatial or temporal invariance in recognition. For instance, a "dog" or a "car" may appear anywhere in the image. If we were to learn independent weights at each spatial or temporal location, it would take orders of magnitude more training data to train such an MLP.

In over-simplified terms, for an MLP which didn't repeat weights across space, the group of neurons receiving inputs from the lower left corner of the image will have to learn to represent "dog" independently from the group of neurons connected to upper left corner, and correspondingly we will need enough images of dogs such that the network had seen several examples of dogs at each possible image location separately.

How is a convolutional neural network able to learn invariant features?

It is the pooling layer that introduces such invariants.

The pooling regimes make convolution process invariant to translation, rotation and shifting. Most widely used one is max-pooling. We take the highest activation to propagate at the interest region so called receptive field. Even the images are relatively a little shifted, since we are looking for highest activation, we are able to capture commonalities between images.

How does zero-centering step help convolutional neural network?

If we normalize then we are already reducing the effect of scale differences and only remain the content of the image.

What is meant by feature maps in convolutional neural networks?

The feature map is the output of one filter applied to the previous layer. A given filter is drawn across the entire previous layer, moved one pixel at a time. Each position results in an activation of the neuron and the output is collected in the feature map. A hidden layer is segmented into feature maps where each unit in a feature map looks for the same feature but at different positions of the input image

How do researchers initialize convolution matrices/features before training the Convolutional Neural Network?

MSRA and Xavier initialization are used.

While using Convolutional Neural Networks as feature extractor, do we extract a layer's features after ReLU activation function or before?

We usually take features after the non-linearity, by convention.

Is there order among different features maps of Convolutional Neural Networks?

No, they are typically unordered, and fully connected in that dimension.

How do Convolutional Neural Networks develop more complex features?

The CNN discovers the basic shapes from the complex ones.

Let’s say we have an object recognition network. The last layer needs to identify the object category. So it will learn something like “if there are wheels, then the object is a car with high probability”, and so on. Now, the previous layer needs to know if there are wheels in the image, to be able to make that decision. Now to detect a wheel, the previous layer will learn something like “if there is a black circle, with metallic sections inside, then it is a wheel with high probability”. As we go from the output layer to the input layer, we need to detect more and more basic shapes.

This is also the intuition for why backpropagation works.

Also it's because each pixel in the output of a convolution layer corresponds to multiple pixels from the previous layer. Hence, we are combining information from multiple data points into a single value, which is essentially an abstraction.

So, with each layer, we keep abstracting the data further, letting the neural network handle more complex structures. Any part of the output of the later convolution layers is dependent on multiple pixels of the previous, and similarly these pixels are dependent on even more pixels.

How do convolutional neural networks for images achieve rotational-invariance?

CNNs achieve invariance (translational, scale and rotational) via pooling. The scale and rotation invariance isn't perfect though, so often researchers expand their datasets by simply adding in rotated versions of the same image (particularly in character recognition).

How do convolutional neural networks for images achieve color invariance?

For example, if we provide various different samples of apples of red, green and yellow, all labelled as apples, the network's influence on the colour aspect will be lesser compared to its importance to other kinds of features.

So, the weights connected to the red, green and blue channels of the image won't be skewed if it has many samples of the same object in different colours.

However, if we have a good dataset with diverse examples for each class, neural networks learn to recognize images (or perform whatever other task) through a combination of factors, not just color.

How can we make convolutional neural networks invariant to scale?

The most common way would be augment your dataset by taking the pre-existing images, and zooming them in or out to different random scales so that by the end of this process we have a bunch of images of different scales. Use this for training and our CNN will likely be able to accomodate the 'scale-range' that we used to augment your dataset. The downside is, we'd need a bigger CNN with more layers to account for the increased complexity.

We can try using this heuristic:

If we have N images in our dataset and M parameters in our CNN to start with, and our CNN works pretty well apart from scale issues:

1) Decide on a 'scale-range'. For example I want my CNN to accomodate ~10 different zoom settings so I apply those to each of my N images. My dataset now has 10N images.

2) Scale the CNN appropriately. Now I need >10M parameters in my CNN to get it to work well on my augmented data. If it overfits, I regularise, if it underfits - I keep adding layers until it overfits, then I regularise.

What is the difference between equivariance and invariance in Convolution neural networks?

Imagine an image of a cat comes in.

Imagine the same image comes in, but rotated.

If we have the same response for the two, that’s invariance.

If we have a response that notes the rotation angle (or something along the lines of that), that’s equivariance.

Convolution is equivariant (not invariant) with respect to translation.

Why and how are convolutional neural networks translation-invariant?

Convolution + Max pooling ≈ translation invariance .

Convolution: provides equivariance to translation which is actually really simple to explain. Basically if we move the image to the right so does its feature layer produced by the convolution. This is obvious since convolution in a Neural Network is defined as a feature detector at different locations. So if its detecting edges and one edge is moved to the right it won't detect it until it goes to the right and matches it. Thus, the feature layer will have an activation moved to the right, but they are really the same representation (same equivalence class).

Pooling: provides the real translation invariance but only approximately. A pooling layer (like max) returns the largest value in its receptive field (input to the pooling function). If the largest value is moved to the right and its still within the receptive field then the pooling layer still outputs that largest value. It became invariant to moving it to the right (translation invariance). Obviously this only applies as long as the image is not translated too much.

So both operations together provide some type of (approximate) invariance to translation, one keeps detecting things even if they are moved (equivariance/convolution) while pooling tries to keep the representation as consistent as possible (approx. translation invariance/pooling).

What's wrong with convolutional neural networks?

A MLP has lesser number of hyper-parameters, but the number of MLP involved in a single CNN is high, meaning the number of hyper-parameters are vast. And with these hyper-parameters improperly set, the entire network could lead to possible wrong results. Proper setting of the hyper-parameters such as learning rates and regularizing constants is a must for training any network.