Face Recognition using Convolutional Neural Networks

Design Oriented Project

EEE F376

Prof. In-Charge: Dr. Surekha Bhanot

Sagnik Majumder 2014A8PS464P

Siddharth K 2014A3PS219P

Birla Institute of Technology and Science Pilani, Pilani Campus

Jan-May 2017

Contents

1. Introduction

The problem of Face recognition in unconstrained environments has been tackled effectively by various research works with the advent of the concepts of Deep Convolutional Networks, which have been supported by the development of numerous deep learning frameworks like Caffe that provide a highly optimized implementation of these networks. With the accuracies approaching human level performance, deep networks have demonstrated their remarkable learning capabilities. In order to cater to the massive data requirements of these networks, research groups across the world have come up with a rich collection of data sets to supplement the training of these deep networks. Thus, evidently there is a growing need for Face recognition in real time as it has wide-ranging applications in surveillance and security systems, face verification, social networking and in recent times the emerging fields of Augmented and Virtual reality have been the driving factors behind the development of fast and highly accurate face recognition systems. In keeping with this, the following project was undertaken to create a system which employs minimal preprocessing of images and is yet able to classify them accurately.

A Brief Overview of Deep Learning

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. It refers to artificial neural networks that are composed of many layers. It is just another name for artificial neural networks (ANN for short) but in a more refined and easier avatar. They have been existing for more than 40 years. In 2000's, NVIDIA accelerated ANN by bringing their chips for scientific computing and from then onwards, use of neural networks has picked up.

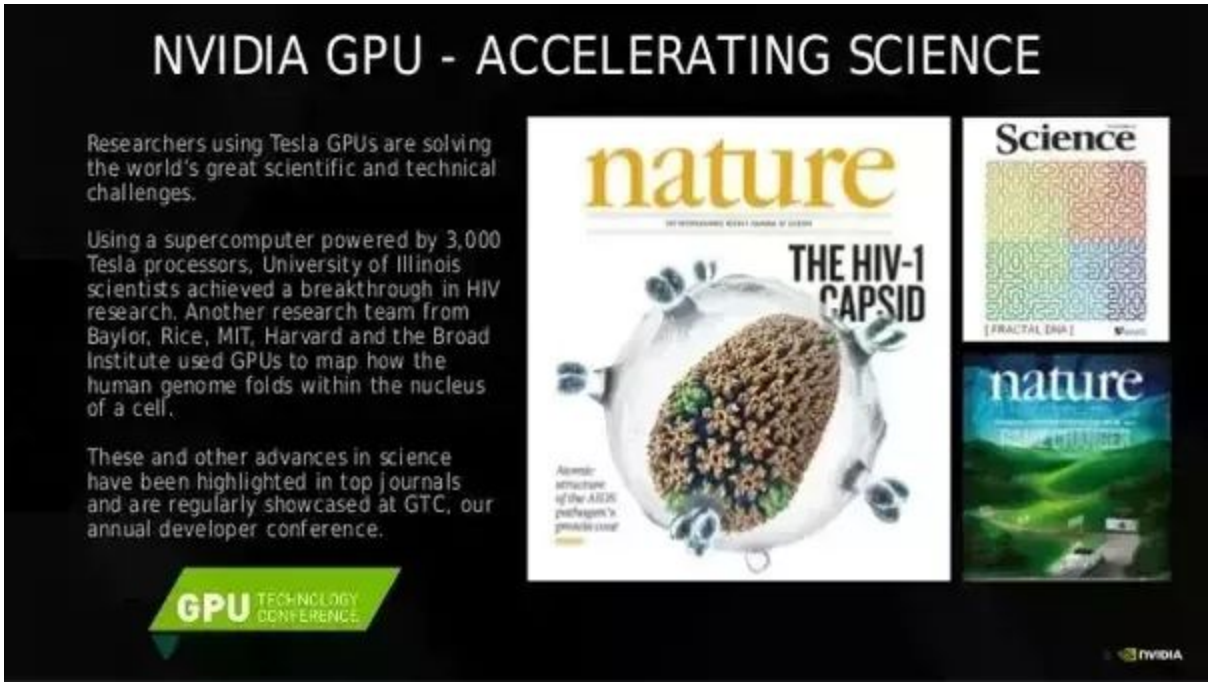

Figure 1: GPUs Have Helped Deep Learning Advance

It's a growing trend in ML due to some favorable results in applications where the target function is very complex and the datasets are large. For example in Hinton et al. (2012), Hinton and his students managed to beat the status quo prediction systems on five well known datasets: Reuters, TIMIT, MNIST, CIFAR and ImageNet. This covers speech, text and image classification - and these are quite mature datasets, so a win on any of these gets some attention. A win on all of them gets a lot of attention.

In machine learning and cognitive science, artificial neural networks (ANNs) are a family of models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.

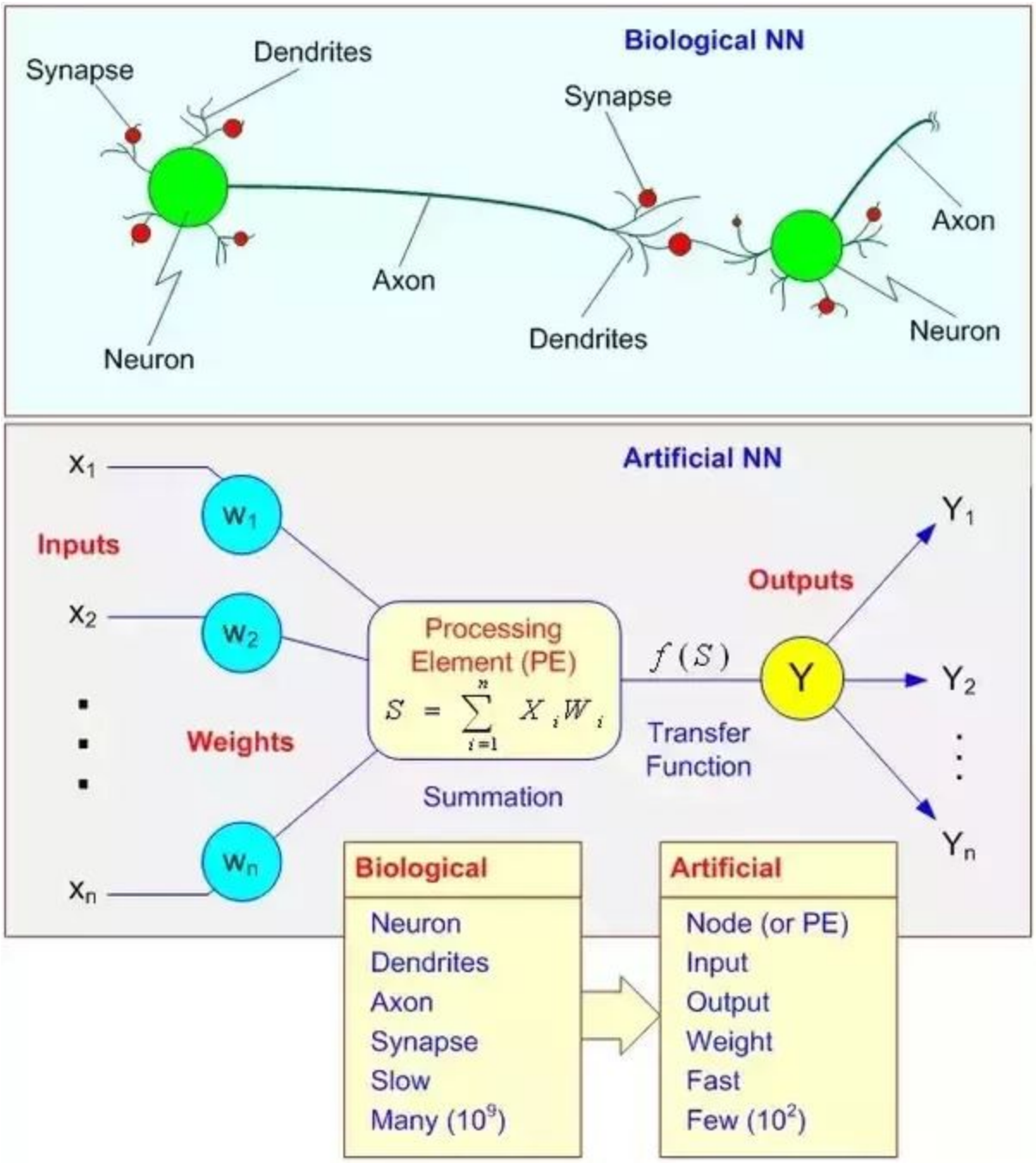

Figure 2: A Comparison Between Biological Neurons and an ANN in Deep Learning

Why 'Deep Learning' is called deep? It is because of the structure of ANNs. Earlier 40 years back, neural networks were only 2 layers deep as it was not computationally feasible to build larger networks. Now it is common to have neural networks with 10+ layers and even 100+ layer ANNs are being tried upon.

One can essentially stack layers of neurons on top of each other. The lowest layer takes the raw data like images, text, sound, etc. and then each neuron stores some information about the

data they encounter. Each neuron in the layer sends information up to the next layers of neurons which learn a more abstract version of the data below it. So the higher one goes up, the more abstract features the layers learn. One can see in the picture below has 5 layers in which 3 are hidden layers.

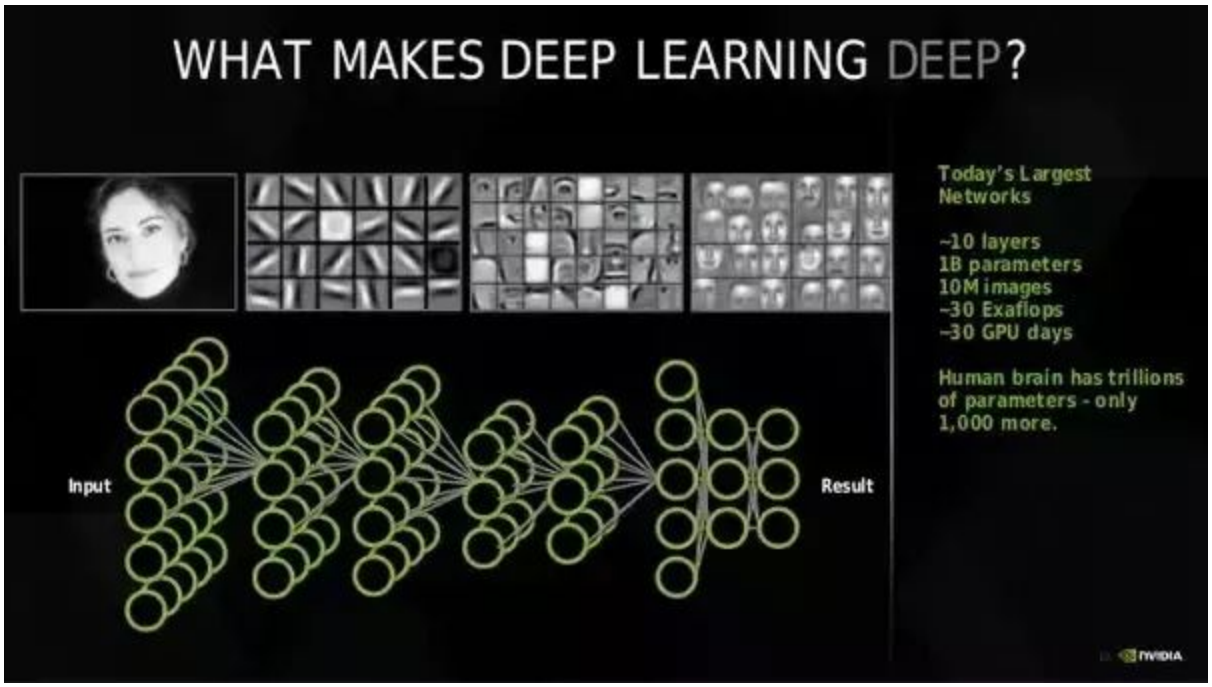

Figure 3: A Representative Deep Neural Network

Why feature engineering may turn obsolete?

Today as a data scientist, one may be learning feature engineering as a part of machine learning skill. In that, one need to transform one's data to the computer in a form that it can understand. For that one may use R or Python or spreadsheet software to translate the data. One may be converting one's data into a large spreadsheet of numbers containing rows and columns of instances and features. Then one feeds this data into the machine learning algorithm and it tries to learn from this data. As engineering the features is a time consuming task, we need to extract only the relevant features that improve our model. But as one is generally unaware of the usefulness of these features until one trains and tests the model, one is caught into the vicious cycle of developing new features, rebuilding the model, measure results, and repeat until one is satisfied with the results. This is very time consuming task and takes lot of your time.

How deep learning may save one's time?

In deep learning, ANNs are automatically extracting features instead of manual extraction in feature engineering. One can take the example of image as input. Instead of a user taking an image and hand compute features like distribution of colors, image histograms, distinct color count, etc., the user just have to feed the raw images in ANN. ANNs have already proved their

worth in handling images, but now they are being applied to all kinds of other datasets like raw text, numbers etc. This helps the data scientist to concentrate more on building deep learning algorithms.

Big Data is required for deep learning

Soon, feature engineering may turn obsolete but deep learning algorithm will require massive data for feeding into our models. Fortunately, we now have big data sources not available two decades back – facebook, twitter, Wikipedia, project Guttenberg etc. However, the bottleneck remains in cleaning and processing these data into required format for powering the machine learning models. More and more big data will be made available for public consumption in near future.

Where can one get help in deep learning?

Open sourcing is predominant in deep learning, Tensorflow, Torch, Keras, Big Sur hardware, DIGITS and Caffe are some of the massive deep learning projects. In academic research, there are lots of papers with algorithm source code along with their findings. arXiv.org (a Cornell University Library) has open access to over 1 million papers in deep learning.

Does one need a computer expertise to learn deep learning?

One does not need a rigorous computer expertise to learn deep learning. The user's domain knowledge of the discipline will help him in building deep learning models. If one has learnt any data science language like 'R' or 'Python', spreadsheets like 'Excel' or any other basic programming and studied 'STEM', then one is proficient to delve into deep learning.

What are deep learning advantages?

One advantage that has already been explained is that the user doesn't have to figure out the features ahead of time. Another advantage is that one can use the same neural net approach for many different problems like Support Vector Machines, Linear Classifier, Regression, Bayesian, Decision Trees, Clustering and Association Rules. It is fault tolerant and scales well. Many perceptual tasks have been performed with help of CNNs. Some case studies are given below:

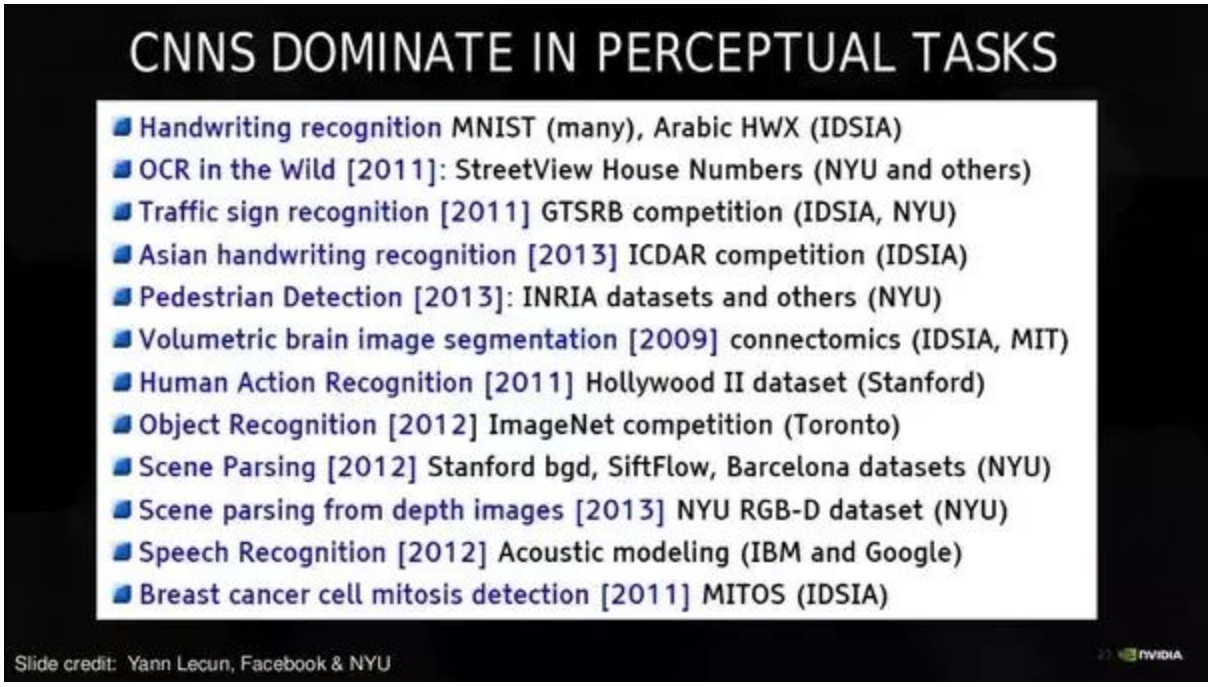

Figure 4: A Few Case Studies Showing the Vast Prospects of Deep Learning

How classical ML using feature engineering can be compared with deep learning using CNN? Well it can be seen with the help of these pictures.

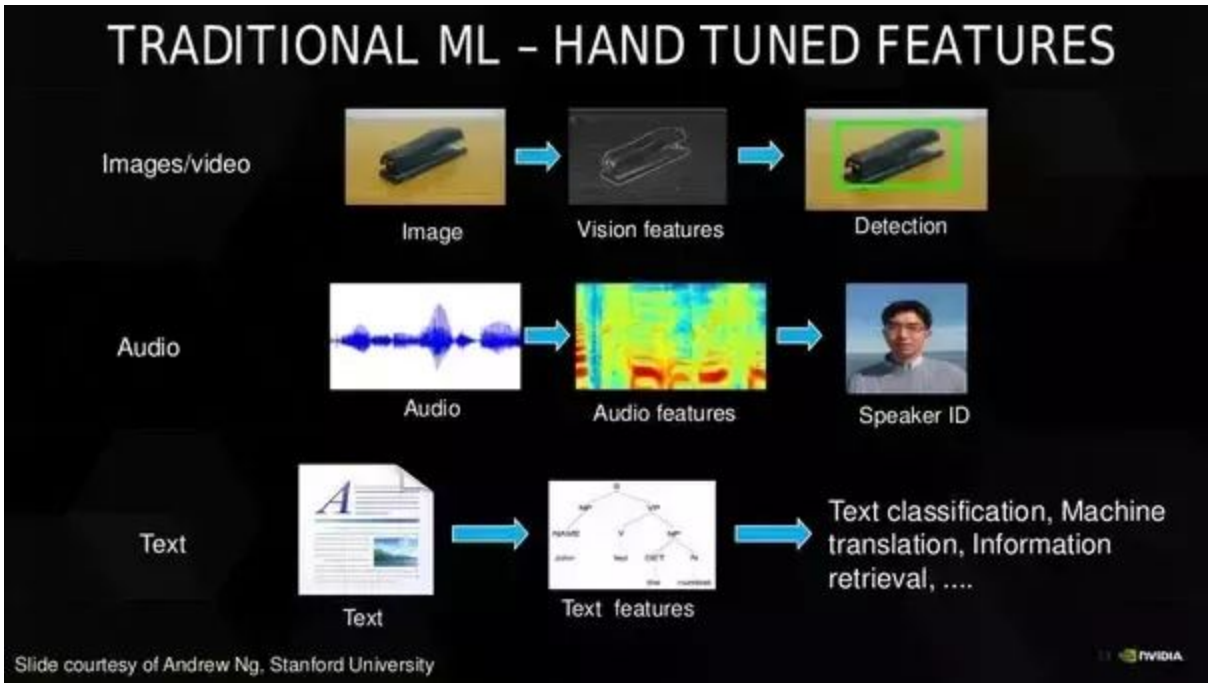

Figure 5 : Traditional ML

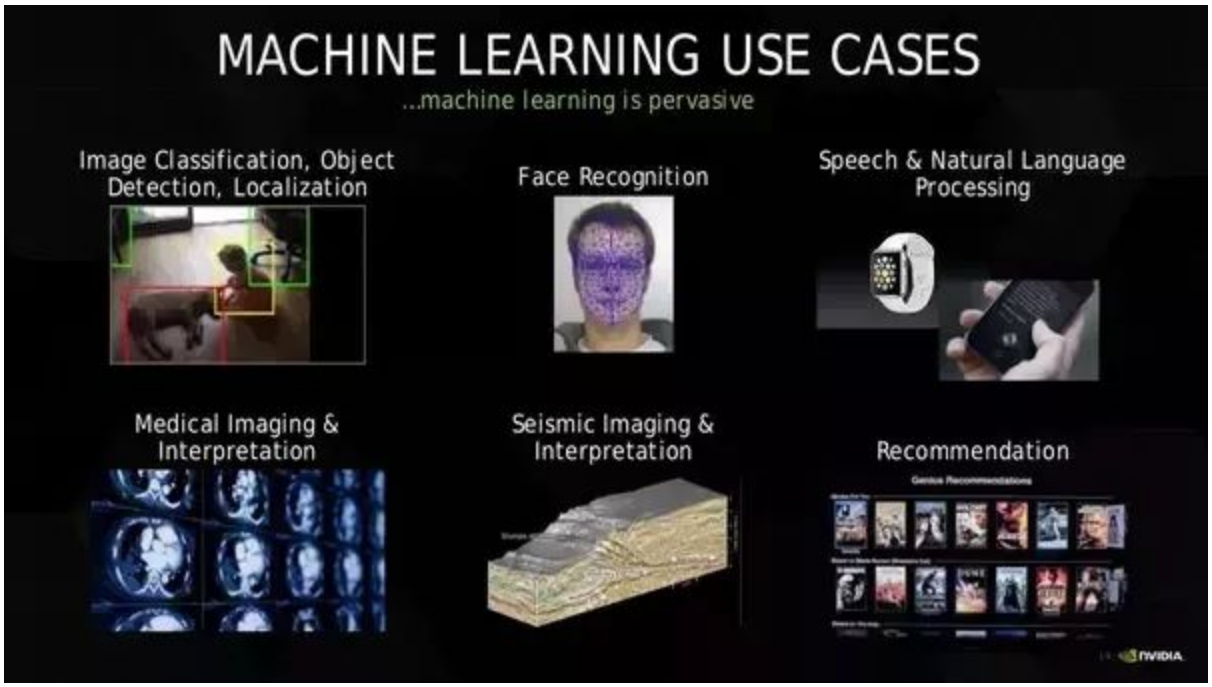

Figure 6 : Deep Learning in ML

From these pictures, one can see the wide applicability of deep learning in all aspects of life.

2. Literature Review

The problem of Face Recognition has always been a challenging one for researchers as it is characterized by a great degree of obscurity and modelling faces is not a simple task due to the wide range of variations in ethnicity, age, orientation, lighting, pose, expression, background clutter, facial variations like hairstyle, facial hair, spectacles, which are hard to capture by hand-crafted features. This necessitated the use of data driven approaches, which seem to be a relatively easier approach. Some of the pre-existing methods are described below.

2.1. EigenFaces [20]

In this approach by Turk and Pentland the face images are projected onto a set of principal axe that have been generated from the training data. The components of the test image are compared with that of training images that are already labelled. The closest match is said to be the class of the test image. Although, an accuracy of 96% over a set of 200 individuals and 3000 images was obtained, the generalization ability of the model was under question as the images of individuals used for training and testing were quite similar in many aspects.

2.2. Geometrical Features

This approach uses hand-crafted features mentioned above to model faces. Some of the features used were chin shape, nose width and length, distance between eyes, position of mouth. Accuracies of upto 90% on a dataset of 47 people were reported by Brunelli and Poggio[4].But it is quite a cumbersome process to locate these features and extract them. Complex algorithms are involved which makes the method inefficient for real time recognition. Moreover other approaches reported 100% accuracy on the same dataset.

2.3. Template Matching

Template Matching works by performing a correlation operation on different segments of images. But the drawback is that it works only on images of same scale, which have similar orientation and lighting.

2.4. Graph Matching

Images are represented as dynamic link graphs and elastic graph matching is used to compare the closest graph among all the stored image graphs. The drawback of this approach is that it is computationally intensive, although it has given good accuracy in literature.

2.5. Convolutional Neural Network

In 1997 Lawrence et al adopted a Convolutional Neural Network approach for tackling the problem of Face recognition. Using five images per person on the ORL [15] database they report an error percentage of 3.8%, which was a very impressive result on the ORL database in comparison to EigenFaces or other prevalent method. But their approach used extensive preprocessing in the form of Self Organizing Maps[9] and KL transform before feeding the image to a LeNet architecture. This preprocessing of images has now been done away with,after the advent Convolutional Networks nearly two decades after Yann LeCun introduced the first architecture of LeNet [16]. The deep architectures in vogue today are capable of extracting very high level features from raw images.

2.6. State-of-the-art DeepFace [19]

The current state-of-the-art is a 9 layer deep network by researchers in Facebook AI Research and University of Tel Aviv, Israel, called DeepFace [19]. With a staggering accuracy on LFW [7] dataset of 97.35%, DeepFace [19] has set the benchmark ever so high on a challenging unconditioned database like LFW [7]. The distinct feature of DeepFace [19] is that the last few convolutional layers do not use the concept of parameter sharing unlike most convnets. The authors argue that as the net progresses into deeper layers the statistical distribution of image data is not the same spatially and it is therefore reasonable to learn different filters for these regions. Hence the network essentially learns a much larger number of parameters as

compared to a conventional convnet for the same number of layers.

3. Description of VGGNet [17] Architecture

CONVNET CONFIGURATIONS - To measure the improvement brought by the increased ConvNet depth in a fair setting, all popular ConvNet layer configurations are designed using the same principles, inspired by Ciresan et al. (2011)[6]; Krizhevsky et al. (2012)[10]. A generic layout of the ConvNet Configurations (VGGNet) is described in Sect. 3.1 and then the specific configurations are detailed in Sect. 3.2. The different design choices for VGGNet are
then discussed and compared to the prior art in Sect. 3.3

3.1. Architecture

During training, the input to our ConvNets is a fixed-size 224 x 224 RGB image. The only pre-processing that needs to be done is subtracting the mean RGB value, computed on the training set, from each pixel. The image is passed through a stack of convolutional (conv.) layers, where we use filters with a very small receptive field: 3 x 3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations we also utilize 1 x 1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3 x 3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2 x 2 pixel window, with stride 2. A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC[14] classification and thus contains 1000 channels (one for each class). The final layer is the softmax layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with the rectification (ReLU (Krizhevsky et al., 2012[10])) non-linearity. It is to be noted that none of the VGGNets (except for one) contain Local Response Normalisation (LRN) normalization (Krizhevsky et al., 2012[10]):, such normalisation does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time. Where applicable, the parameters for the LRN layer are those of (Krizhevsky et al., 2012).

3.2. Configurations

The ConvNet configurations, different configurations of the VGGNet, are outlined in Table 1, one per column. In the following we will refer to the nets by their names (AE). All configurations follow the generic design presented Sect. 3.1 and differ only in the depth: from 11 weight layers in the network A (8 conv. and 3 FC layers) to 19 weight layers in the network E (16 conv. and 3 FC layers). The width of conv. layers (the number of channels) is rather small, starting from 64 in the first layer and then increasing by a factor of 2 after each max-pooling layer, until it

reaches 512.
In Table 2 we report the number of parameters for each configuration. In spite of a large depth, the number of weights in our nets is not greater than the number of weights in a more shallow net with larger conv. layer widths and receptive fields (144M weights in (Sermanet et al., 2014)).

3.3. Discussion

Our ConvNet configurations are quite different from the ones used in the top-performing entries of the ILSVRC-2012 (Krizhevsky et al., 2012) and ILSVRC- 2013 competitions (Zeiler and Fergus[22], 2013; Sermanet et al., 2014[16]). Rather than using relatively large receptive fields in the first conv. layers (e.g. 11 x 11 with stride 4 in (Krizhevsky et al., 2012), or 7 x 7 with stride 2 in (Zeiler & Fergus, 2013; Sermanet et al., 2014)), we use very small 3 x 3 receptive fields throughout the whole net, which are convolved with the input at every pixel (with stride 1). It is easy to see that a stack of two 3 x 3 conv. layers (without spatial pooling in between) has an effective receptive field of 5 x 5; three such layers have a 7 x 7 effective receptive field. So the advantages of using, for instance, a stack of three 3 x 3 conv. layers instead of a single 7 x 7 layer, are two-fold:

1. First, three non-linear rectification layers are incorporated instead of a single one, which makes the decision function more discriminative.

2. Second, we decrease the number of parameters: assuming that both the input and the output of a three-layer 3 x 3 convolution stack has C channels, the stack is parametrized by 3 (3^2 C^2) = $27C^2$; at the same time, a single 7 x 7 conv. layer would require 7^2C^2 = $49C^2$ parameters i.e. 81% more. This can be seen as imposing regularization on the 7 x 7 conv filters, forcing them to have decomposition through the 3 x 3 filters (with nonlinearity injected in between).

The incorporation of 1 x 1 conv. layers (configuration C, Table 1) is a way to increase the nonlinearity of the decision function without affecting the receptive fields of the conv. layers. Even though in our case the 1 x 1 convolution is essentially a linear projection onto the space of the same dimensionality (the number of input and output channels is the same), an additional nonlinearity is introduced by the rectification function. It should be noted that 1 x 1 conv layers have recently been utilized in the Network in Network[12] architecture of Lin et al. (2014).

Small-size convolution filters have been previously used by Ciresan et al. (2011), but their nets are significantly less deep than VGGNets, and they did not evaluate on the large-scale ILSVRC dataset. Goodfellow et al. (2014) applied deep ConvNets (11 weight layers) to the task of street number recognition, and showed that the increased depth led to better performance. GoogLeNet[18] (Szegedy et al., 2014), a top-performing entry of the ILSVRC- 2014 classification task, was developed independently of the work on VGGNets, but is similar in that it is based on very deep ConvNets (22 weight layers) and small convolution filters (apart from 3 x 3, they also use 1 x 1 and 5 x 5 convolutions).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 7: Table 1: ConvNet configurations (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as conv receptive field size - number of channels. The ReLU activation function is not shown for brevity. ([10])

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Figure 8: Table 2: Number of parameters (in millions). ([10])

Their network topology is, however, more complex than that of VGGNets, and the spatial resolution of the feature maps is reduced more aggressively in the first layers to decrease the amount of computation. The VGGNet model outperforms that of Szegedy et al. (2014) in terms of the single-network classification accuracy.

4. Classification Framework

4.1. Training of VGGNets

The ConvNet training procedure generally follows Krizhevsky et al. (2012) (except for sampling the input crops from multi-scale training images, as explained later). Namely, the training is carried out by optimizing the multinomial logistic regression objective using mini-batch gradient descent (based on backpropagation (LeCun et al., 1989[11])) with momentum. The batch size was set to 256, momentum to 0.9. The training was regularized by weight decay (the L2 penalty multiplier set to 5.10-4) and dropout regularization for the first two fully-connected layers (dropout ratio set to 0.5).

The learning rate was initially set to 10^{-2}, and then decreased by a factor of 10 when the validation set accuracy stopped improving. In total, the learning rate was decreased 3 times, and the learning was stopped after 370K iterations (74 epochs). A conjecture can be drawn that in spite of the larger number of parameters and the greater depth of our nets compared to (Krizhevsky et al., 2012), the nets required less epochs to converge due to

1. Implicit regularization imposed by greater depth and smaller conv filter sizes;

2. Pre-initialization of certain layers.

The initialization of the network weights is important, since bad initialization can stall learning due to the instability of gradient in deep nets. To circumvent this problem, training the configuration A (Table 1) was started, shallow enough to be trained with random initialization. Then, when training deeper architectures, the first four convolutional layers and the last three fully-connected layers were initialized with the layers of net A (the intermediate layers were initialized randomly). The learning rate for the pre-initialized layers was not decreased, allowing them to change during learning. For random initialization (where applicable), the weights were sampled from a normal distribution with the zero mean and 10-2 variance.

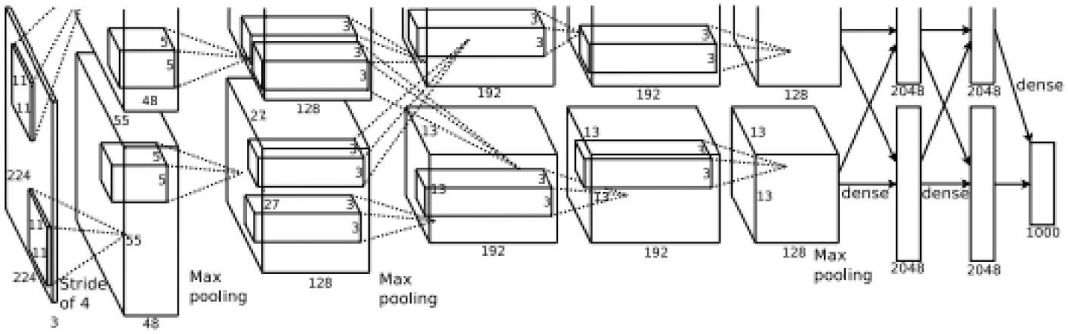

Figure 9: An image of a generic CNN showing the convolutions and pooling operations ([10])

The biases were initialized with zero. It was worth noting that after the paper submission of VGGNets it was found that it was possible to initialize the weights without pre-training by using the random initialization procedure of Glorot & Bengio (2010).

To obtain the fixed-size 224x224 ConvNet input images, they were randomly cropped from rescaled training images (one crop per image per SGD iteration). To further augment the training set, the crops underwent random horizontal flipping and random RGB color shift (Krizhevsky et al., 2012). Training image rescaling is explained below.

Training image size. Let S be the smallest side of an isotropically-rescaled training image, from which the ConvNet input is cropped (we also refer to S as the training scale). While the crop size is fixed to 224 x 224, in principle S can take on any value not less than 224: for S = 224 the crop will capture whole image statistics, completely spanning the smallest side of a training image; for S 224 the crop will correspond to a small part of the image, containing a small object or an object part.

Two approaches were considered for setting the training scale S. The first is to fix S, which corresponds to single-scale training (note that image content within the sampled crops can still represent multi-scale image statistics). In experiments on VGGNets,models trained at two fixed scales: S = 256 (which has been widely used in the prior art (Krizhevsky et al., 2012; Zeiler & Fergus, 2013; Sermanet et al., 2014)) and S = 384 were evaluated. Given a ConvNet configuration, the network was first trained using S = 256. To speed-up training of the S = 384 network, it was initialized with the weights pre-trained with S = 256, and a smaller initial learning rate of 10^{-3} was used.

The second approach to setting S is multi-scale training, where each training image is individually rescaled by randomly sampling S from a certain range [Smin, Smax] (Smin = 256 and Smax= 512 were used). Since objects in images can be of different size, it is beneficial to take this into account during training. This can also be seen as training set augmentation by scale jittering, where a single model is trained to recognize objects over a wide range of scales. For speed reasons, multi-scale models were trained by fine-tuning all layers of a single-scale model with the same configuration, pre-trained with fixed S = 384.

4.2. Testing

At test time, given a trained ConvNet and an input image, it is classified in the following way. First, it is isotropically rescaled to a predefined smallest image side, denoted as Q (we also refer to it as the test scale). We note that Q is not necessarily equal to the training scale S (using several values of Q for each S leads to improved performance). Then, the network is applied densely over the rescaled test image in a way similar to (Sermanet et al., 2014). Namely, the fully-connected layers are first converted to convolutional layers (the first FC layer to a 7 x 7 conv. layer, the last two FC layers to 1 x 1 conv. layers).
The resulting fully-convolutional net is then applied to the whole (uncropped) image. The result is a class score map with the number of channels equal to the number of classes, and a variable spatial resolution, dependent on the input image size. Finally, to obtain a fixed-size vector of class scores for the image, the class score map is spatially averaged (sum-pooled). We also augment the test set by horizontal flipping of the images; the soft-max class posteriors of the original and flipped images are averaged to obtain the final scores for the image.

Since the fully-convolutional network is applied over the whole image, there is no need to sample multiple crops at test time (Krizhevsky et al., 2012), which is less efficient as it requires network re-computation for each crop. At the same time, using a large set of crops, as done by Szegedy et al. (2014), can lead to improved accuracy, as it results in a finer sampling of the input image compared to the fully-convolutional net. Also, multi-crop evaluation is complementary to dense evaluation due to different convolution boundary conditions: when applying a ConvNet to a crop, the convolved feature maps are padded with zeros, while in the case of dense evaluation the padding for the same crop naturally comes from the neighbouring parts of an image (due to both the convolutions and spatial pooling), which substantially increases the overall network receptive field, so more context is captured. While we believe that I practice the increased computation time of multiple crops does not justify the potential gains in accuracy, for reference we also evaluate our networks using 50 crops per scale (5 x 5 regular grid with 2 flips), for a total of 150 crops over 3 scales, which is comparable to 144 crops over 4 scales used by Szegedy et al. (2014).

5. Datasets

5.1. Color Feret [13]

A standard database of face imagery was essential to the success of the FERET program, both to supply standard imagery to the algorithm developers and to supply a sufficient number of images to allow testing of these algorithms. Before the start of the FERET program, there was no way to accurately evaluate or compare facial recognition algorithms. Various researchers collected their own databases for the problems they were investigating. Most of the databases were small and consisted of images of less than 50 individuals. Notable exceptions were databases collected by three primary researchers:

1. Alex Pentland of the Massachusetts Institute of Technology (MIT) assembled a database of 7500 images that had been collected in a highly controlled environment with controlled illumination; all images had the eyes in a registered location, and all images were full frontal face views.

2. Joseph Wilder of Rutgers University assembled a database of 250 individuals collected under similarly controlled conditions.

3. Christoph von der Malsburg of the University of Southern California (USC) and colleagues used a database of 100 images that were of controlled size and illumination but did include some head rotation.

The FERET program set out to establish a large database of facial images that was gathered independently from the algorithm developers. Dr. Harry Wechsler at George Mason University was selected to direct the collection of this database. The database collection was a collaborative effort between Dr. Wechsler and Dr. Phillips. The images were collected in a semi-controlled environment. To maintain a degree of consistency throughout the database, the same physical setup was used in each photography session. Because the equipment had to be reassembled for each session, there was some minor variation in images collected on different dates. The FERET database was collected in 15 sessions between August 1993 and July 1996. The database contains 1564 sets of images for a total of 14,126 images that includes 1199 individuals and 365 duplicate sets of images. A duplicate set is a second set of images of a person already in the database and was usually taken on a different day. For some individuals, over two years had elapsed between their first and last sittings, with some subjects being photographed multiple times. This time-lapse was important because it enabled researchers to study, for the first time, changes in a subject's appearance that occur over a year.

5.2. LFW [7]

Labeled Faces in the Wild, a database of face photographs designed for studying the problem of unconstrained face recognition. The data set contains more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set. The only constraint on these faces is that they were detected by the Viola-Jones face detector. More details can be found in the technical report below. Information:

1. 13233 images
2. 5749 people
3. 1680 people with two or more images

5.3 You Tube Faces (YTF)[21]

The data set has been developed for studying unconstrained face recognition in videos. It contains 3425 videos of 1595 people, which have been downloaded from YouTube. Average

number of videos available per subject is 2.15, with average number of frames per video clip being 181.3. Hence the total number of images is 620,953.

#videos	1	2	3	4	5	6
#people	591	471	307	167	51	8

Table 1: Number of Videos Per Person

6. Caffe Deep Learning Framework [8]

Caffe is a deep learning framework developed by the Berkeley Vision and Learning Center. The main features of Caffe that make it one of the most attractive frameworks for deep learning enthusiasts is listed below:-

1. Expression: The models and optimization parameters in Caffe are defined by plain text documents rather than cumbersome code which makes it very user friendly.

2. Speed: Caffe provides a robust implementation of algorithms which enables fast computation using state-of-the-art models for massive data.

3. Modularity: Flexibility of Caffe allows for easy extension and modification of pre-defined models.

4. Openness: Caffe source code has been open sourced, hence giving users a comprehensive understanding of the back-end.

5. Community: research communities, industries, startups are part of the active Caffe community and have made significant contribution in debugging and adding code to the framework.

Basic working of Caffe: Caffe stores its data in structures known as blobs, which act as a wrapper around the actual data and thereby concealing from the user the overhead of switching of data between CPU and GPU. This is seamless switching between CPU and GPU is a major advantage of using Caffe. The data stored in blobs could be gradient values for optimization, network parameters or even batches of images. Each layer in a network performs three fundamental operations namely:- setup, forward and backward.

The setup operation initializes the network with its connection values. The forward operation computes the output from the input blob and sends it to the output blob. The backward operation computes the gradient with respect to bottom input using the gradients with respect to the top output. The net used by Caffe is a set of layers connected in a computational graph.

This directed cyclic graph is the basic workhorse of Caffe. A function and its gradient are defined by composition and auto-differentiation by Caffe. The composition of outputs of the various layers of Caffe gives the function that computes the final output from input data. The composition of gradients of parameters from output to input computes the gradient of the loss with respect to each layer and hence the network is able to learn the task at hand.

The Solver of Caffe is responsible for coordinating the forward and backward operations in order to optimize the model by performing parameter updates in order to improve the loss. Hence, the Solver and Net modules of Caffe work in synchronization to train the defined model.

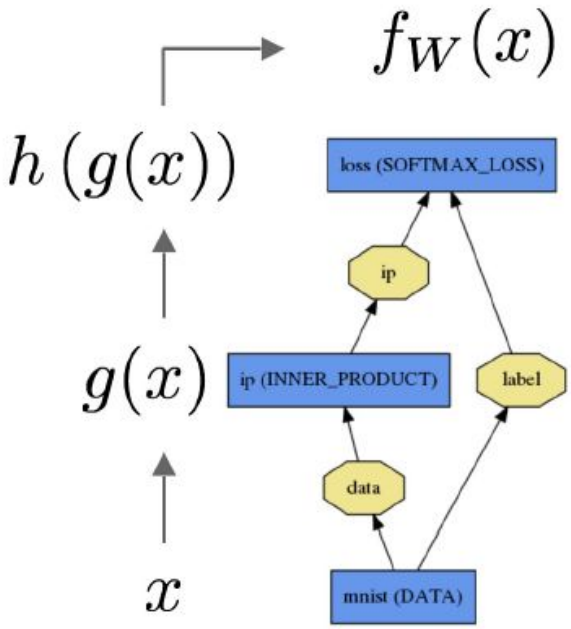

Figure 10: An image showing forward pass ([8])

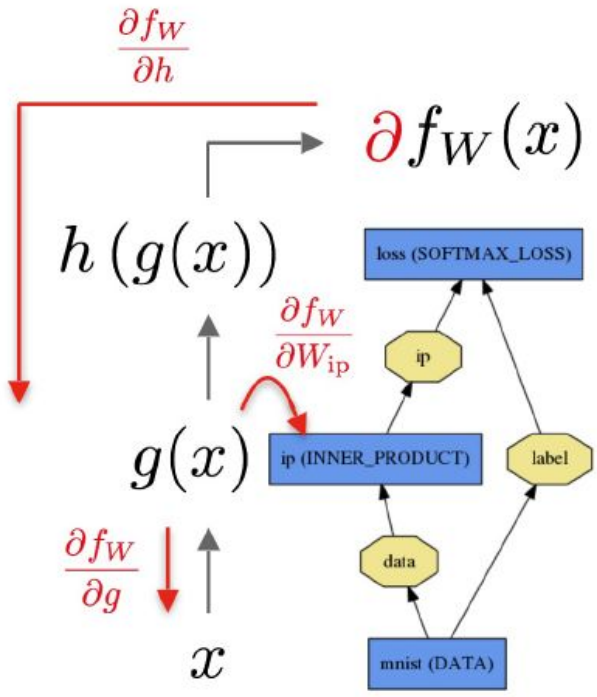

Figure 11: An image showing backward pass ([8])

Caffe provides the implementation of various Solver schemes like Stochastic Gradient Descent, Ada Delta, Adaptive Gradient, Adam, Nesterov's Accelerated Gradient and RMSProp. Description of the parameters of Solver is given below:-

1. **base lr**: sets the initial learning rate of the model
2. **lr policy**: defines how the learning rate should vary during the course of training
3. **gamma**: the factor by which learning rate drops by
4. **step size**: sets the number of iterations after which the learning rate is changed
5. **max iter**: sets an upper cap on maximum number of iterations for training exceeding which training terminates.
6. **momentum**: sets the momentum parameter of training

The models in Caffe are defined in text files that are of the format .prototxt. A sample definition of a layer is given below to demonstrate the simplicity of the framework.

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
  }
}
```

Figure 12: An image showing a layer in Caffe ([8])

Caffe can read input image data in multiple forms. Images can be read in their raw form, LMBD /LEVELDB format or can even be read directly from memory. In this project we use the LMDB database format to feed input images to the convnet owing to its reliability and read-optimized nature. LMDB stands for Lightning Memory- Mapped Database is a software library that provides a high performance transactional database in the form of key-value pairs. Its reliability comes from the fact that it uses a copy-on-write approach and hence never overwrites data that is currently being used. This prevents the database from getting corrupted in case the application or system crashes.

7. Training Pipeline

No.	Database	Training split (%)	Validation split (%)	Testing split (%)
1	Color Feret	70	0	30
2	Color Feret	70	15	15
3	Color Feret	50	15	35
4	LFW	70	15	15

Table 2: Dataset Split

Parameter	Value
Batch_size_train	10
Batch_size_test	5
base_lr	0.001
lr_policy	step
stepsize	33000
max_iter	100000
momentum	0.9
gamma	0.1
weight_decay	5.10^{-4}
solver_mode	GPU

Table 3: Training Parameters

8. Approaches for Tackling LFW [7] Dataset

Papers on Face recognition have used Face Alignment as a preprocessing step in order to improve accuracy of the algorithm.Two libraries available on GitHub were used to perform the alignment. (The link for which has been provided in the references)

8.1. OpenFace[2] library by TadasBaltrusaitis

It is an open-source computer vision toolbox developed mainly by Tadas Baltrusaitis during his time at the Language Technologies Institute at the Carnegie Mellon University. The toolbox is equipped with functionalities for performing a wide array of tasks on face like Facial Landmark Detection, Facial Landmark and head pose tracking, Gaze tracking, Facial Action Unit Recognition and Feature Extraction (aligned faces and HOG features). In particular the script named 'feature_extraction_demo_img_seq.m' was used to perform the face alignment. The samples of a few aligned images and their corresponding unaligned images have been shown below to demonstrate the robustness of this toolbox.

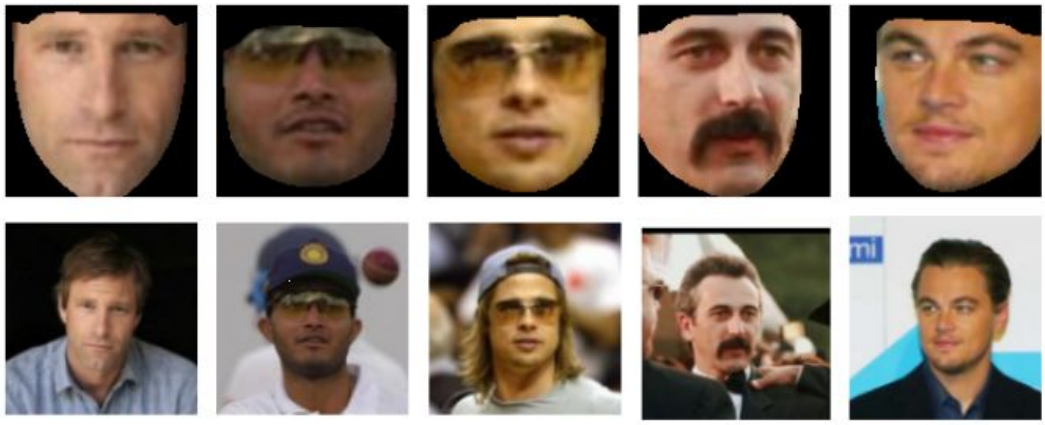

Figure 13: Aligned and scaled images using OpenFace by Baltrusaitis ([2])

As is observed from the above images the toolbox is able to tackle rotation translation and even occlusion to a great degree and produce images of size 112x112. Moreover, the background clutter is completely eliminated and thereby making the task of learning much easier for the network. The drawback of this toolbox was that the output dimension of the aligned images could not be set. Hence we altered the input layer dimensions of our ConvNet from 224x224x3 to 112x112x3. This does not have any impact on the number of weights of the convolutional layers but will alter the number of weights between the max-pooling layer and first fully connected layer.

Result of training: - Unfortunately, the training was unfruitful and the loss function failed to converge to a minimum value. Speculating that the cause of failure might be the alteration in input dimensions we moved over to the subsequent toolbox which did provide the flexibility of altering the dimensions of the output aligned images.

8.2. OpenFace[1] library by cmusatyalab

OpenFace is a free and open source library using deep neural networks, based on the paper FaceNet: A Unified Embedding for Face Recognition and Clustering published in CVPR 2015. It has been implemented in Python and Torch and can be run on CPU as well as CUDA. It uses the following steps to align images. Firstly, the faces are detected using pre-trained models from dlib or OpenCV (which are open source libraries coded in C/C++). Next, real time pose-estimation from dlib and affine transformation from OpenCV are used to align images in such a way that eyes and bottom lip appear on the same location in each image. In particular, the script named align-dlib.py was used to perform the alignment. The samples of a few aligned images and their corresponding unaligned images have been shown below to demonstrate the robustness of this toolbox.

Figure 14: Aligned and scaled images using OpenFace by cmusatyalab([1])

Unlike the former toolbox, this toolbox did not eliminate the background clutter. Also, upon rescaling it was observed that the images, although aligned, were blurred as compared to the original. Since the face occupied only one part of the entire original image, it was expected to be blur, when cropped and rescaled. This was a cause of concern prior to training as the images input to the ConvNet were of a much lower resolution.

Result of training: - As expected, the training was unsuccessful and the loss function yet again failed to converge to a minimum value. The graph of loss function and training accuracy vs number of iterations has been plotted below.

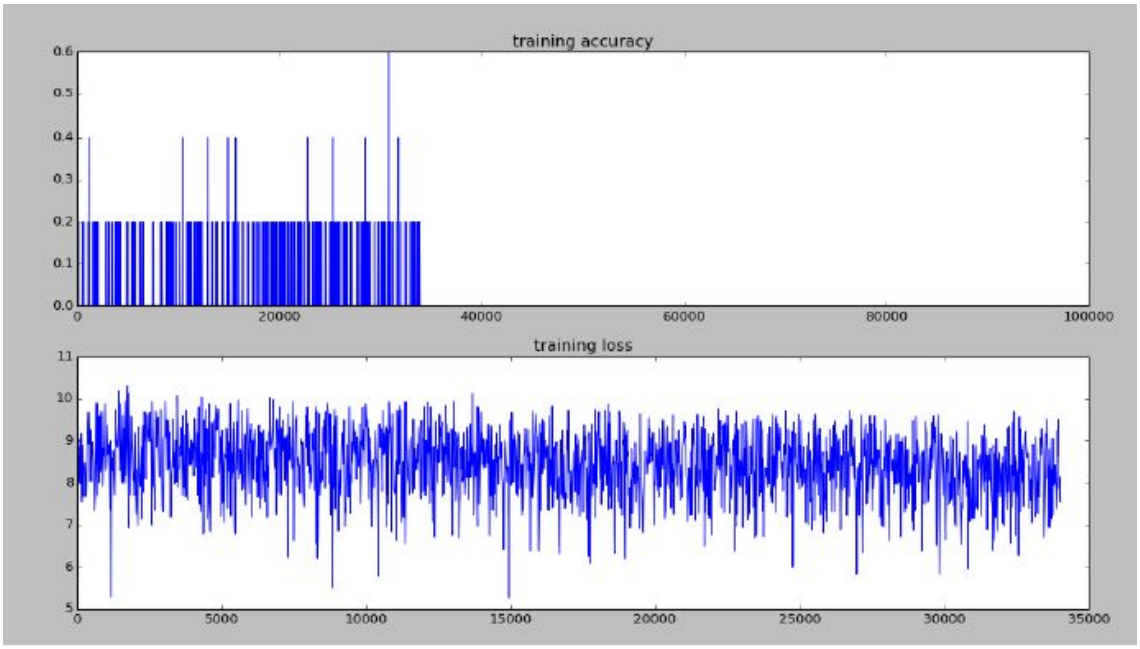

Figure 15: Training Error on Images Aligned by OpenFace ([2])

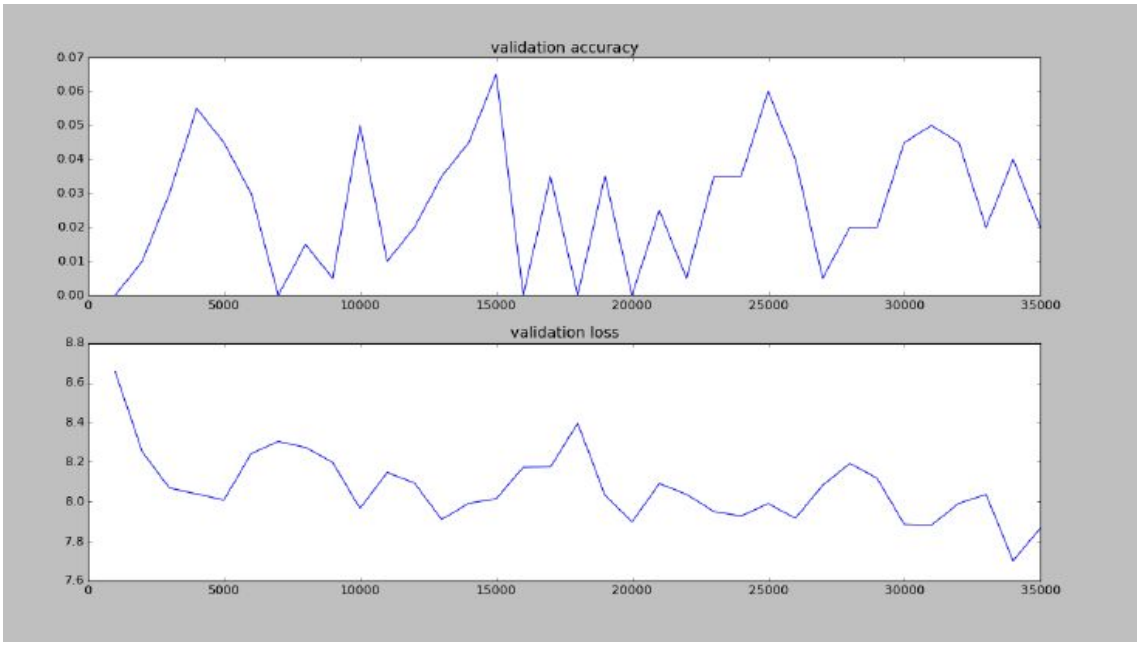

Figure 16: Validation Error on Images Aligned by OpenFace ([2])

9. Transfer Learning

Chen et al argue that a good training data must have width and depth, i.e. the dataset must contain a large number of subject classes for training and each individual class must have a good number of images in order for the network to train effectively. LFW[7] suffers severely from the problem of small depth. Chen et al have resorted to the creation of a new dataset namely, Wide and Deep Reference Net (WDRef) in order to cope with this problem and later hope to transfer the learning acquired by the training of this model to the LFW[7] Dataset. Essentially, the model trained on WDRef learns a high dimensional representation of faces towards the end of the network, which LFW[7] did not allow due to its lack of depth. Training on LFW[7] would hence be more convenient for the network, given that it has already learnt how to represent faces from the previous training.

9.1. Transfer Learning over Color Feret

The first approach that was adopted was to train over the pre-trained Color Feret model which had achieved near 100% accuracy. The learning rates of the first 5 convolutional layers were made 0 as the low level features would be similar to that of Coloreret.

Result of training: - The training was unsuccessful yet again. Hence, we moved over to a different dataset to achieve LFW, namely, YouTube Faces.

9.2. Transfer Learning over YouTube Faces [21]

The YouTube Faces Dataset, which has been described elaborately in the previous section has over 600,000 images. This was sufficient depth for training and additionally some of the classes present LFW were also present in YTF and this gave us an added advantage.

10. Test Results

Training for Color Feret

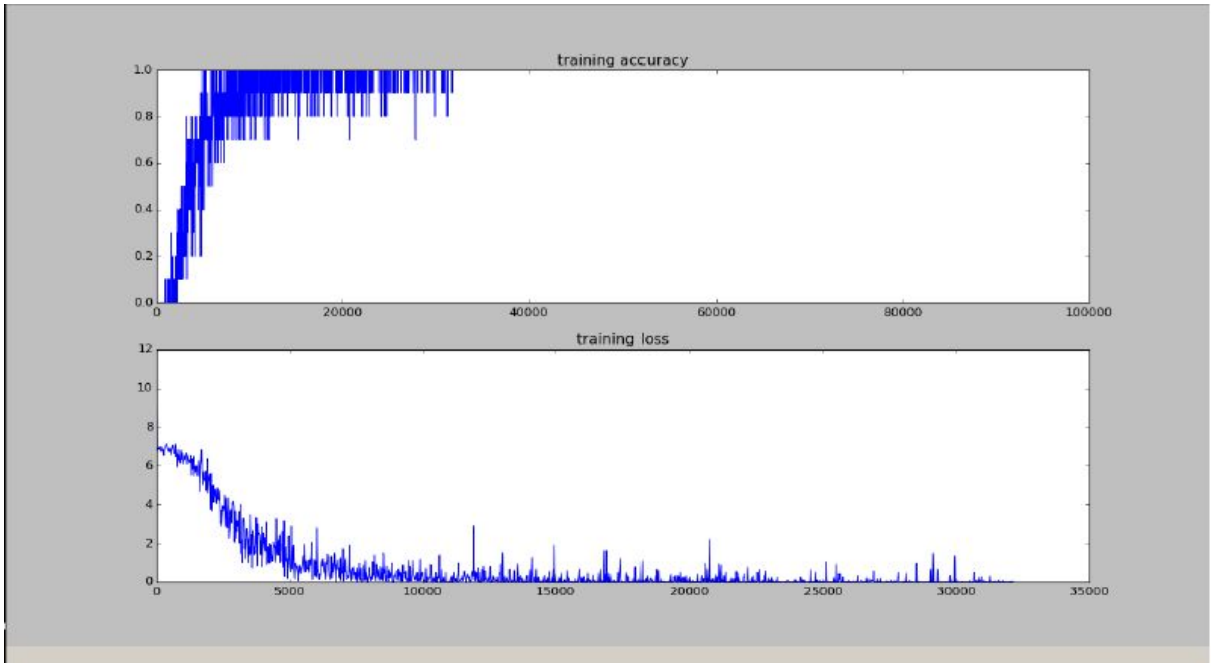

Figure 17: Training Error and Accuracy on Color Feret Images ([13])

Validation for Color Feret

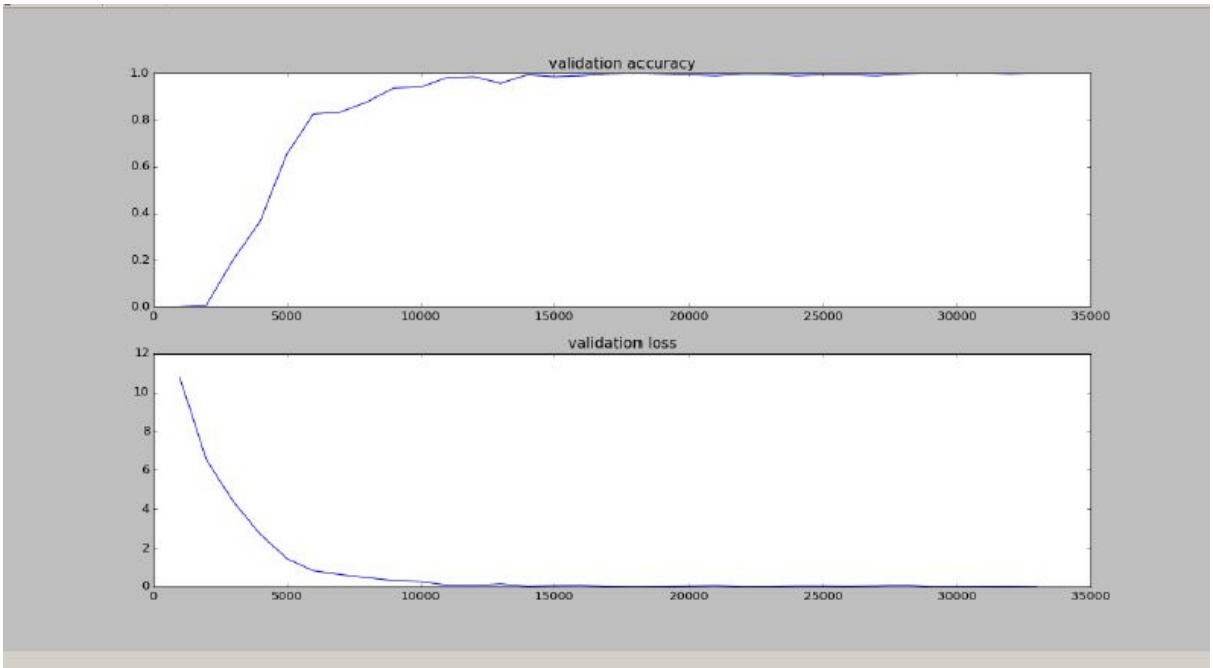

Figure 18: Validation Error and Accuracy on Color Feret Images ([13])

Training for LFW

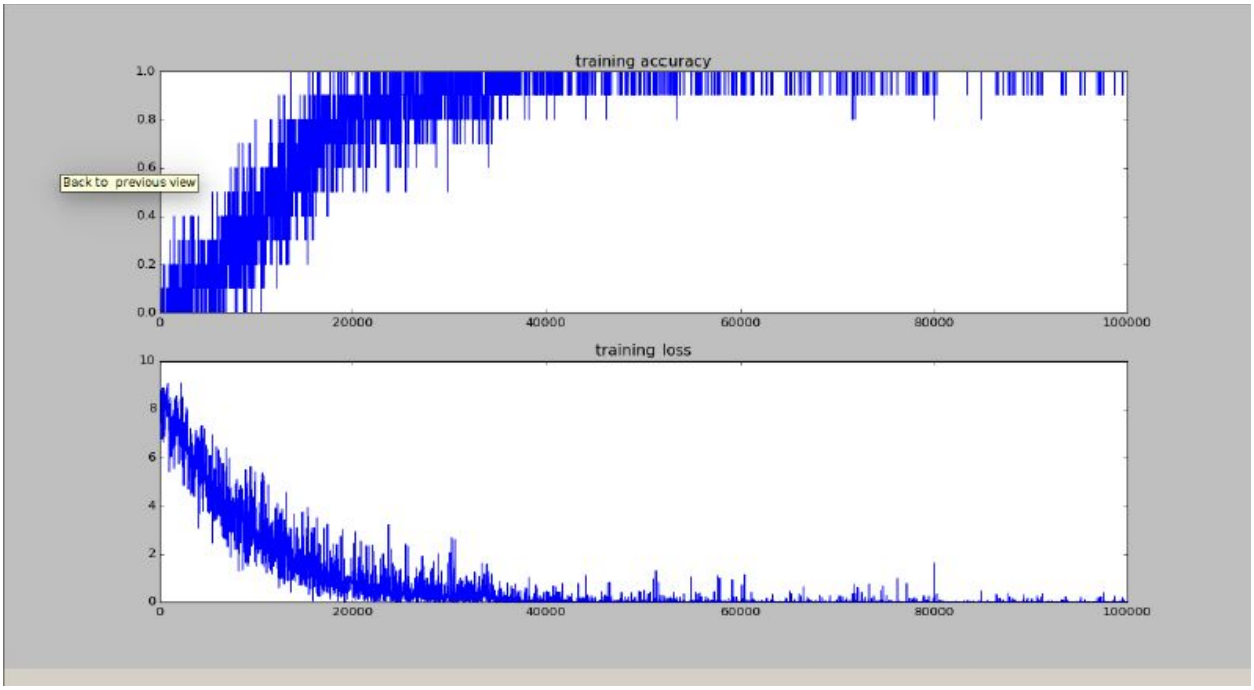

Figure 19: Training Error and Accuracy on LFW Images ([7])

Validation for LFW

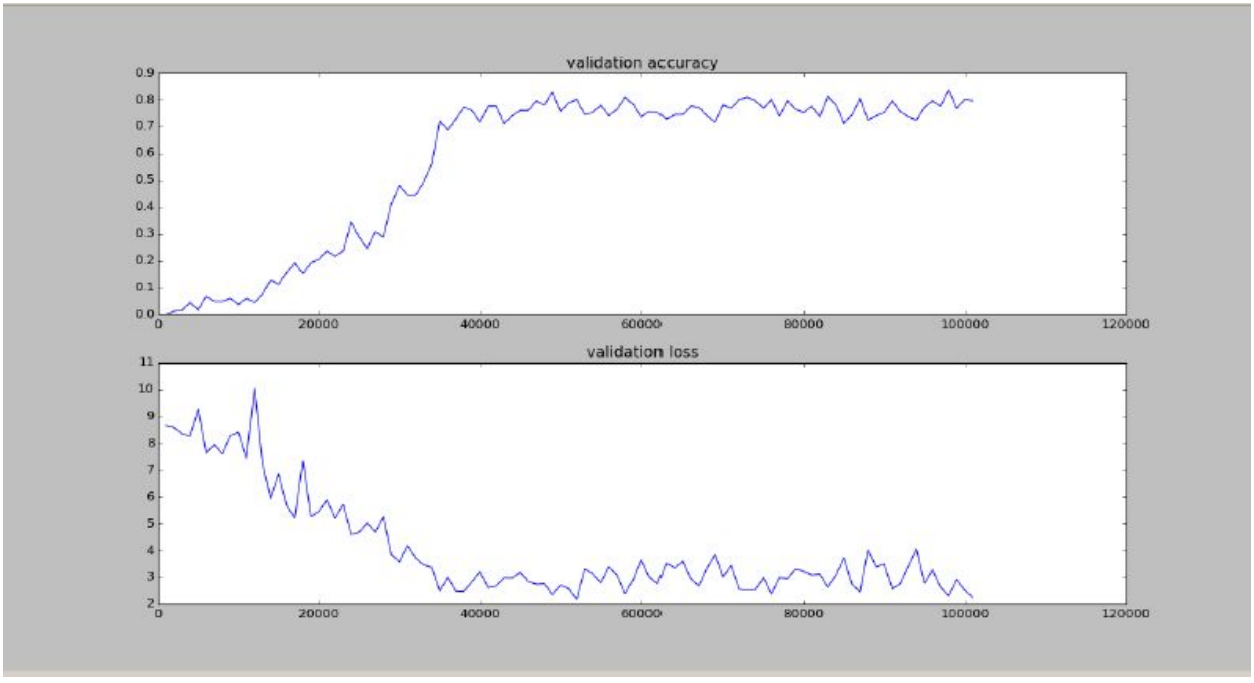

Figure 20: Validation Error and Accuracy on LFW Images ([7])

Training for LFW over YTF

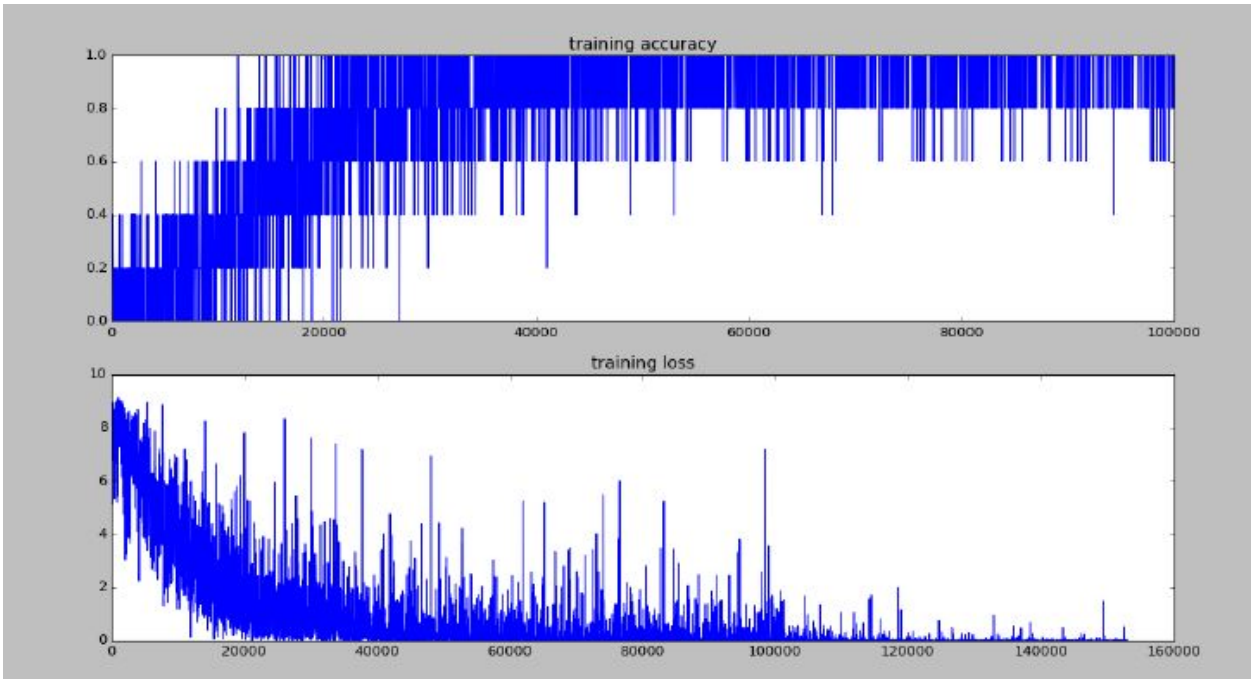

Figure 21: Training Error and Accuracy for LFW over YTF Images ([7],[21])

Validation for LFW over YTF

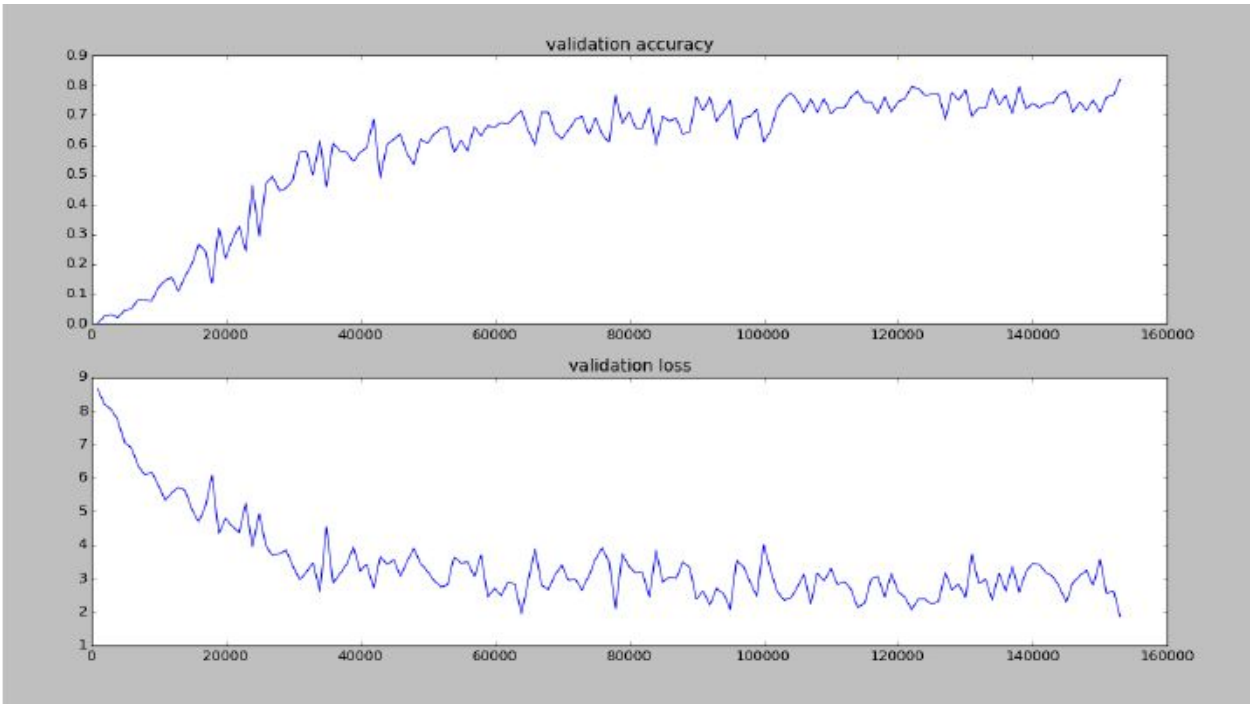

Figure 22: Validation Error and Accuracy on LFW over YTF Images ([7],[21]

Dataset	Training Accuracy	Testing Accuracy
LFW	79.6%	46.7%
Color Feret	100%	96.3%
LFW over YTF	100%	82.2%
YTF	100%	97.4%

Table 4: Test Results

11. Conclusion and Final Discussion

We observed that the transfer learning approach indeed improved the performance accuracy of the network by almost double for the testing set, while a full 100% was achieved on the training set. This further validates the claim of Chen et al that depth of a class is a very important factor in training deep networks efficiently. Moreover, the achieved accuracy is over just half the number of total iterations (150000/300000) that had been set in solver prototxt. Due to lack of memory in the computer the training had to be terminated. Our Future work shall resume training from where it left off and will complete the stipulated number of iterations in the hope for a better accuracy.

In addition to this, some important observations were made. Firstly the images had no preprocessing performed on them whatsoever and the despite this the CNN was able to tackle various obstacles and challenges like variations in pose, lighting conditions, a certain degree of occlusions and variations in expression. Secondly, since the background was constant throughout the video frames in a particular video, one would expect the network to have learnt this as well. But when the network was trained over LFW and presented with images that have a high degree of background clutter, it was still able to identify the face, disregarding the background.

References

[1] Brandon Amos, Bartosz Ludwiczuk, and Mahadev Satyanarayanan. Openface: A general-purpose face recognition library with mobile applications. Technical report, CMU-CS-16-118, CMU School of Computer Science, 2016.

[2] Tadas Baltrusaitis, Peter Robinson, and Louis-Philippe Morency. Openface: an open source facial behavior analysis toolkit. In IEEE Winter Conference on Applications of Computer Vision, 2016.

[3] Oren Barkan, Jonathan Weill, Lior Wolf, and Hagai Aronowitz. Fast high dimensional vector multiplication face recognition. In The IEEE International Conference on Computer Vision (ICCV), December 2013.

[4] Roberto Brunelli. Template Matching Techniques in Computer Vision: Theory and Practice. Wiley Publishing, 2009.

[5] Xudong Cao, David Wipf, Fang Wen, Genquan Duan, and Jian Sun. A practical transfer learning algorithm for face verification. In The IEEE International Conference on Computer Vision (ICCV), December 2013.

[6] Dan C. Ciresan, Ueli Meier, and Jurgen Schmidhuber. Multi-column deep neural networks for image classification. CoRR, abs/1202.2745, 2012.

[7] Gary B. Huang, Marwan Mattar, Honglak Lee, and Erik Learned-Miller. Learning to align from scratch. In NIPS, 2012.

[8] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014.

[9] Teuvo Kohonen. The self-organizing map. Neurocomputing, 21(1):1-6,1998.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097{1105, 2012.

[11] Yann A LeCun, Leon Bottou, Genevieve B Orr, and Klaus-Robert Muller. Efficient backprop. In Neural networks: Tricks of the trade, pages 9{48. Springer, 2012.

[12] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. CoRR, abs/1312.4400, 2013.

[13] P. Jonathon Phillips, Hyeonjoon Moon, Syed A. Rizvi, and Patrick J. Rauss. The feret evaluation methodology for face-recognition algorithms. IEEE Trans. Pattern Anal. Mach.

Intell., 22(10):1090{1104, October 2000.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. CoRR, abs/1409.0575, 2014.

[15] Ferdinando S Samaria and Andy C Harter. Parameterization of a stochastic model for human face identification. In Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on, pages 138{142. IEEE, 1994.

[16] Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. In International Conference on Pattern Recognition (ICPR 2012), 2012.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.

[18] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. CoRR, abs/1409.4842, 2014.

[19] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '14, pages 1701{1708, Washington, DC, USA, 2014. IEEE Computer Society

.

[20] Matthew Turk and Alex Pentland. Eigenfaces for recognition. J. Cognitive Neuroscience, 3(1):71{86, January 1991.

[21] Lior Wolf, Tal Hassner, and Itay Maoz. Face recognition in unconstrained videos with matched background similarity. In Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, pages 529{534. IEEE, 2011.

[22] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. CoRR, abs/1311.2901, 2013.