EXPLORING THE TRANSFER LEARNING ASPECT OF DEEP NEURAL NETWORKS IN FACIAL INFORMATION PROCESSING

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2015

By
Crefeda Faviola Rodrigues
School of Computer Science

Contents

List of Tables

List of Figures

Abstract

EXPLORING THE TRANSFER LEARNING ASPECT OF DEEP NEURAL NETWORKS IN FACIAL INFORMATION PROCESSING

Crefeda Faviola Rodrigues

A dissertation submitted to the University of Manchester for the degree of Master of Science, 2015

This dissertation presents a complete process of the design, implementation and test of a machine learning model based on deep Convolutional Neural Networks (CNNs) [24], with the aim to investigate its transfer learning aspects for facial information processing.

Human face images convey different types of information ranging from facial identity or morphological information to facial expression information. However, it is challenging to extract a specific information component[1] given that both these types of information are entangled into a universal representation. By investigating the transferability of facial features, it is hoped that the nature and extent to which deep CNN offers an advantage to transfer learning is studied and its utility to disentangle information components is verified.

In the experimental work, various CNN architecture designs were considered for three benchmark facial databases. These designs were narrowed down using model selection and the selected model was subsequently tested on its transfer learning capabilities. Results obtained favour knowledge transfer at lower layers of deep CNNs where features are general while it does not favour knowledge transfer at higher layers where the features are specific to a particular data set or task. This degree of transferability in turn provided evidence of disentangling of facial information components.

[1] Extraction of task-specific representation is called Information Component Analysis (ICA) [8]

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr Ke Chen. I'm very grateful for his expert advice and constant guidance during every stage of this dissertation.

Special thanks to the School of Computer Science at the University of Manchester and to my Professor Graham Riley for permitting me to use the 48-core machine from the MSc labs which was very helpful.

I would also like to thank to my parents, sister Cresida and friends for their constant support and care over the time period of the project. Finally, I am grateful to my dad for proofreading early drafts of my dissertation.

Chapter 1

Introduction

1.1 Overview

Deep learning[1] is an emerging area of research in the field of machine learning. Successful applications have been built using deep learning methods in speech and audio processing, language modelling and natural language processing, information retrieval and computer vision [12]. Some of the reasons which has spiked interest in this field is the availability of large labelled data sets, GPU implementations which makes training of large models possible and better regularization methods to avoid over fitting of these models [23], [44].

In computer vision, the recent success in the ImageNet Challenge [23] shows that such deep architectures are capable of learning low level features such as different oriented edges to higher level abstract features such as object boundaries or contours that are built upon these low level features. Such multi-level representations are considered *good* as it facilitates extraction of useful information for a particular task along with re-use of common features for other tasks.

Transfer Learning is another active research area in machine learning that exploits the re-usability of features learnt by deep models. Here the representations learnt for a given input distribution or task is transferred to a different distribution or task. This technique has been used in acquiring feature representations when there are few or no labels of the class of interest during training [29].

[1] A sub-field within machine learning that is based on algorithms for learning multiple levels of representation in order to model complex relationships among data.

The main motivation behind this project was to use this state-of-the-art technique in representation learning, such as deep CNNs, to verify the hypothesis of transferability of features in deep networks for facial information processing. Two different problems were chosen to test this hypothesis: Face Recognition (FR) and Facial Expression Recognition (FER). The raw pixel images of faces often contain different types of information components like identity, expression, pose, occlusion and illumination which are entangled into a universal representation. Therefore, emphasis was placed on the ability of the model to learn representations which disentangle information components in the input data. If the outcome favours transferability, the improvement in the performance of existing face recognition or facial expression recognition systems can be envisaged to exploit transferability of features.

1.2 Aim

Develop and evaluate a machine learning model based on CNNs, to study the transfer learning aspect (as per the experimental protocols and methods proposed by [43]) in deep CNNs for both face recognition and face expression recognition tasks.

1.3 Objectives

- Learning Objectives
  - Investigate and understand deep architectures: specifically Deep CNNs, together with its layered architecture and properties, methods of training like Stochastic Gradient Descent with Back Propagation.
  - Review how CNNs can be applied to actual Machine Learning related problems such as classification, detection and recognition.
  - Investigate the use of CNNs for facial image processing like face recognition and facial expression recognition.
  - Study and understand the concept of Transfer Learning and review the use of transfer learning methods for machine learning and data mining technologies.
  - Exploration of CNN MATLAB libraries such as MatConvNet toolbox.

- Deliverable Objectives
  - Develop and implement a software program in MATLAB capable of analysing transferability of features at each layer in CNNs when applied to facial images.
  - Develop and implement a visualization technique that facilitates visualization of the learned features by projecting it in raw pixel space.
  - An experiment demonstrating the effectiveness of transfer learning at disentangling information components for the task of face recognition and face expression recognition.
  - Analyse the experimental results and provide a justification of the utility of transfer learning at disentangling facial information components based on the top-1 or top-5 accuracy of the model on test data and the visualization technique.

1.4 Report Outline

The structure of the report is as follows:

- chapter 1: Introduction
  This chapter covers a brief introduction to the problem domain and describes the aims and objectives of the project.

- chapter 2: Literature Review
  This chapter contains a literature review covering topics related to this project. It includes a detailed overview into the research domains and sub-domains involved in the project. The broader context of representation learning has been presented which is followed by a detailed coverage on representation learning by deep CNNs. Concepts from Artificial Neural Networks to deep CNNs is given along with training of such deep architectures. The chapter ends with a another main research area of this project called Transfer Learning. Additionally, this chapter provides an overview on the current research being carried out in deep CNNs in the field of facial image processing, and transfer learning methods used in image classification.

- chapter 3: Problem Statement and Methods:
  In this chapter, the research questions addressed by this project is elaborated

with justification. The research methodology adopted to implement the system is also described. The various architecture design choices and hyper-parameter selection has been explained. Finally, the chapter ends with the transfer learning hypothesis of the project along with two verification methods used to answer the hypothesis.

- chapter 4: Implementation
  The details of the design and implementation of the software solution is exposed in this chapter. The software toolbox and system information is provided. Flow chart representations depict the logical flow of the code developed for the project.

- chapter 5: Experiments
  This chapter covers all the experimental work carried out for the project. Sections are based on three benchmark datasets that were tested for transfer learning characteristics. The results of the experiments are presented with accuracy plots and visualization of feature maps.

- chapter 6: Discussion
  This chapter provides a discussion of the project with a personal reflection on the limitations and challenges of the project. It also contains a discussion of possible directions of future work based on these limitations.

- chapter 7: Conclusions and future work
  This chapter summarizes the entire project while highlighting the objectives fulfilled in the process.

Chapter 2

Literature Review

This chapter is concerned with the background material for the project. The sections are divided based on the research domains and sub-domains involved in this project. It begins with the broader context of Representation Learning within which this project is nestled. The second and third sections describe deep CNNs which is the main technical component of the project to learn hierarchical representations. Training of deep CNNs, that is the process of finding values for the parameters which lead to meaningful representations of the data has been elaborated. Finally, the section ends with the related work in facial recognition using CNNs. The final section covers another key research area of this project, which is Transfer Learning. It explains the concept and need for transfer learning. A brief overview is provided regarding recent research done in the field of transfer learning.

2.1 Representation Learning

Learning good internal representations of the data, can have a huge impact on the performance of machine learning algorithms [5]. A good representation contains features that are invariant to distortions (or *irrelevant features*) of the input, and are capable of disentangling explanatory factors of variation (or *relevant features*) in the input. For facial images the distortions include translation, scaling and rotations of a face within an image and the factors of variations include identity, expression, lighting, pose and occlusions. These factors or information components are intertwined into a single raw pixel representation.

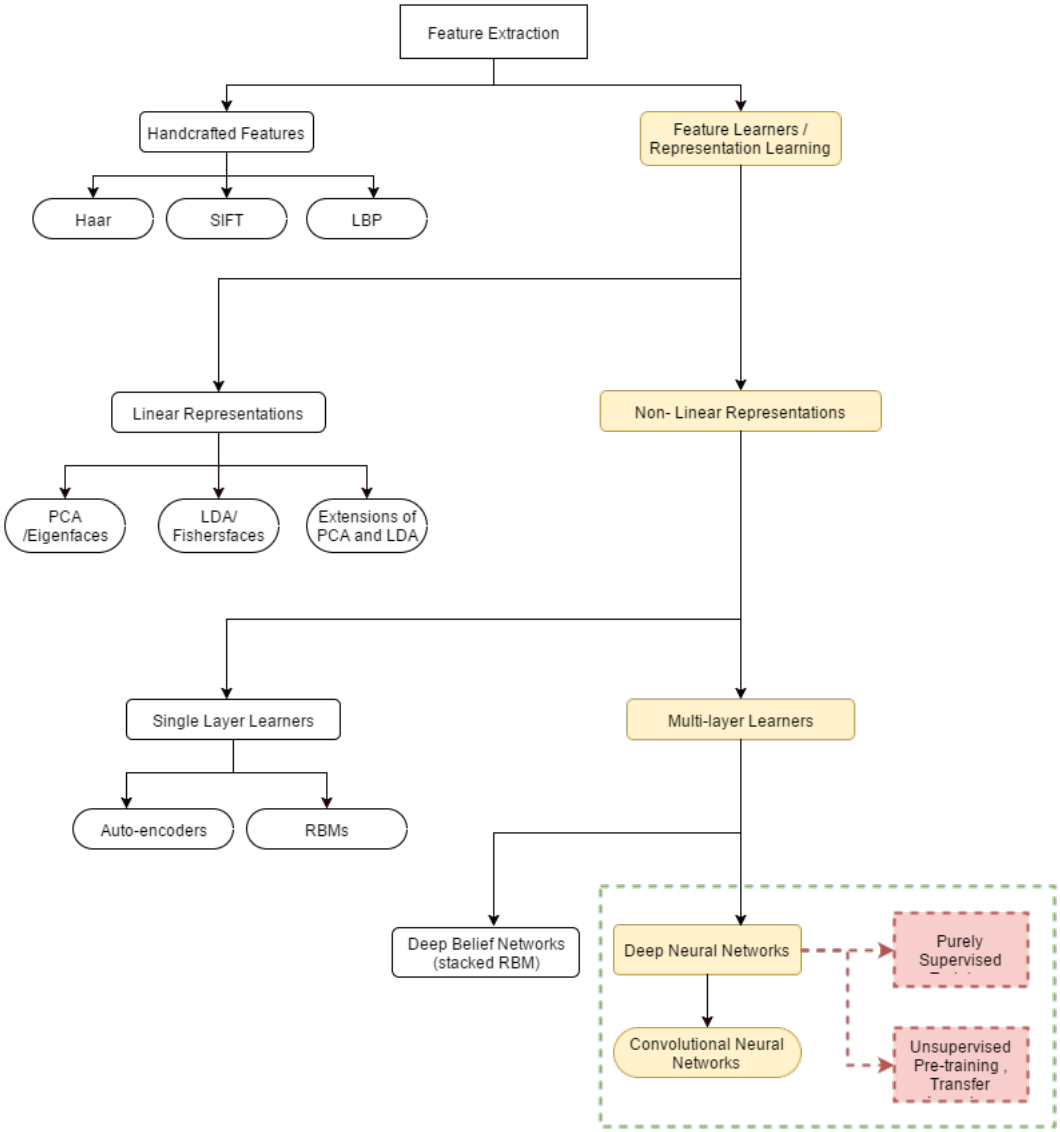


Figure 2.1: Flowchart highlighting the hierarchy of research domains and sub-domains

Images acquired from unconstrained environmental conditions like a CCTV cameras, contain a combination of one or more of these distortions and variation (like pose, lighting or occlusions). which make it difficult to extract relevant features.

Figure 2.1, depicts the various methods to extract features from data. Some of the traditional methods in computer vision to extract useful information rely on *hand-crafting* feature extractors. Methods such as SIFT, LBP and HAAR [5], exploited expert knowledge to design feature extractors while many of the established facial

image processing systems focus on *representation learning* to automatically learn abstract representations from data, upon which a classifier is built. An example of such a system is shown in Figure 2.2, which consists of different steps: Face Detection, Face Normalization, Feature Extraction and Classification [26]. Here the goal of the feature extraction step is to extract suitable information (either identity for FR or expression for FER) from the pre-processed raw pixel representations.

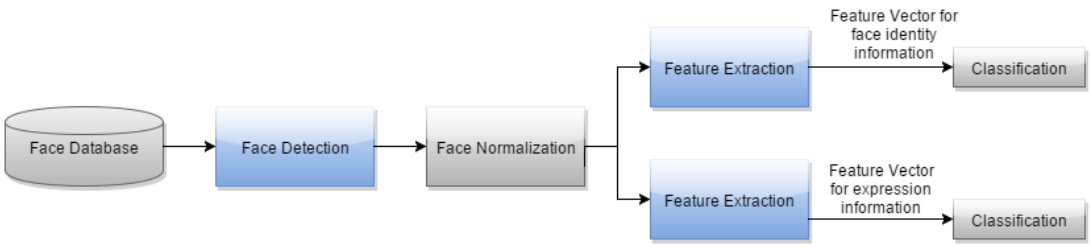


Figure 2.2: Facial Image Processing Pipeline

Earlier methods in representation learning focussed on *linear* methods [2] such as Eigenface (based on PCA), Fisherface (based on LDA) and their extensions. Here each pixel in an image is considered as a coordinate in high dimensional space called the image space. These methods use the technique of dimensionality reduction where the data is projected linearly from the image space to a low-dimensional subspace called face subspace. In particular, Eigenfaces, chooses a projection that maximizes total scatter across all facial images. This projection matrix is composed of principal components or eigenfaces to form a compact low dimensional feature vector and a linear combination of these eigenfaces is used to reconstruct the given face. In Fisherface, a discriminative subspace is created which approximates inter- and intra-personal face variations by using two scatter matrices (within class S_w and between class S_b matrices) and finds the projection directions to maximize the ratio between them.

These systems work well under the assumption that the relationship between the independent underlying factors and the observed high-dimensional data can be expressed by a linear transformation [5]. However, in real world these underlying factors vary with each other in complex ways.

Recently, much of the success of representation learning for Artificial Intelligence (AI)

tasks is accredited to algorithms capable of extracting *non-linear features*. These non-linear representations help in abstracting out the complex interaction of factors in data. These representations are learnt through either single layer models like RBMs and Auto-encoders [5] or they can be stacked into deep models with a hierarchical organization of explanatory factors such as Deep Belief Networks (DBNs) or Deep Neural Networks. These deep models can either be trained layer-wise through unsupervised or supervised pre-training or trained as single unit in a purely supervised setting [5].

As highlighted in Figure 2.1 this project falls in the broader research domain of representation learning and narrows down to the *deep learning* approaches of representation learning with the use of deep CNNs [25]. Another key research area is to use such multi-level representations for *transfer learning*, which is explained section 2.4. These architectures have been used successfully in recent literature on computer vision tasks like object recognition on the ImageNet database [23] and the Facial Expression Recognition Challenge- 2013 [16], with the top three teams using deep CNN architectures.

2.2 Convolutional Neural Network (CNN)

CNNs are similar to Regular Neural Networks[1] that are also composed of computational units (nodes) called *neurons* [20], as shown in Figure 2.3. This artificial neuron takes as input $x_1, x_2, \ldots x_d$ where d is dimension of the input and bias b and outputs the *prediction*:

$$h_{w,b}(x) = f(b + \sum_{i=1}^{d} W_i x_i) \tag{2.1}$$

where $f : R \rightarrow R$ is the *activation function*. W_i are the connection weights and bias b are the parameters that fit the data.

[1]Neural Networks are inspired from biological Neurons that simulate the functions of dendrites, axon, synapse and cell body. They communicate through generation of action potential or electrical impulses. This action potential is model through the activation function and bias.

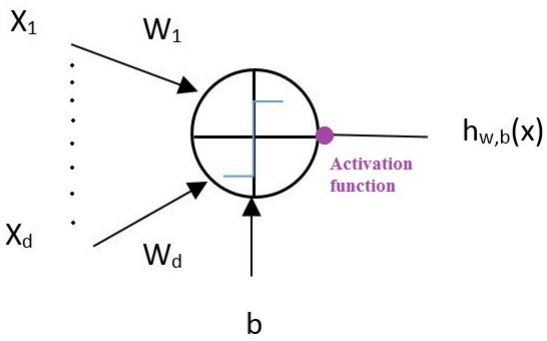


Figure 2.3: Simple Neuron Model

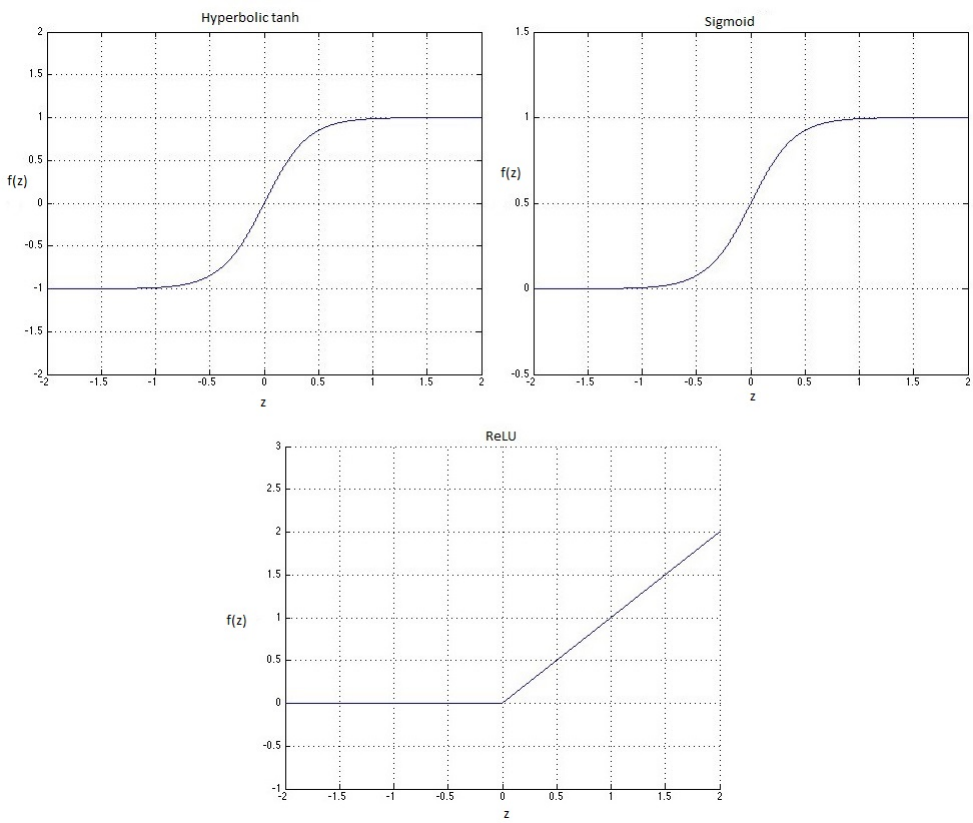


Figure 2.4: Non-Linear activation functions

The output is a linear transformation of the input X (pre-activation) followed by a non-linearity f (output activation). There are a number of activation functions that are available in literature to introduce non-linearity in the computation of the output of a

neuron like the linear function, sigmoid function, hyperbolic tangent function (tanh) and rectified linear (ReLU) functions [39], as shown in Figure 2.4.

Recent published literature on deep neural networks has favoured the use of non-saturating non-linearities such as ReLUs to model the neurons output as these perform faster than saturating non-linearities like tanh, sigmoid [23], [18].
The ReLU activation function is given by:

$$f(z) = max(0, z) \tag{2.2}$$

It has a gradient 0 when $z \leq 0$ and 1 otherwise.

These neurons can be connected together to build neural networks that contain a *single hidden* layer or *multiple hidden* layers.

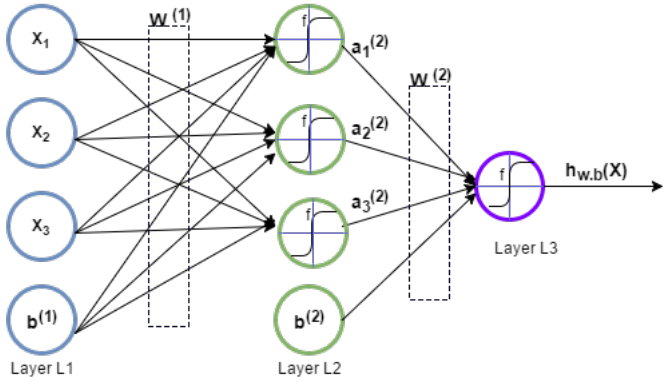


Figure 2.5: Neural Network model

Shown in Figure 2.5, is an example of a neural network with single hidden layer. It has a total of L layers where layer L1 of the network is called *input* layer, layer L3 with a single node is called *output* layer, and layer L2 of nodes is called *hidden* layer. The computations made by this network in compact form is given as follows:

$$z^{(2)} = W^{(1)}x + b^{(1)} \tag{2.3}$$

$$a^{(2)} = f(z^{(2)}) \tag{2.4}$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} \tag{2.5}$$

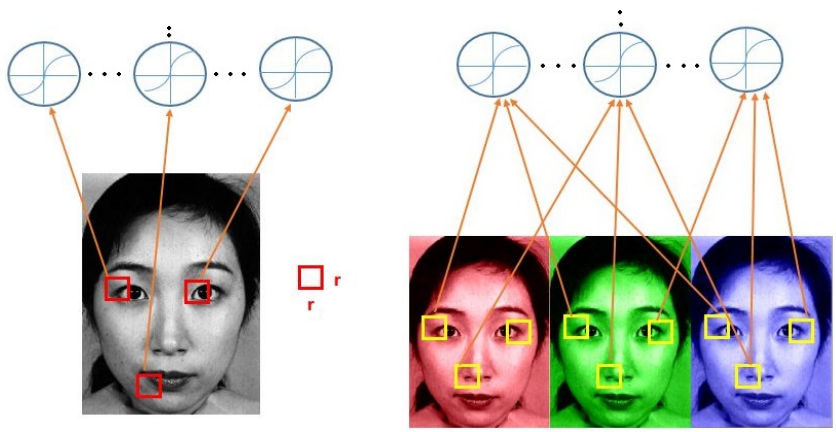
$$h_{w,b}(x) = a^{(3)} = f(z^{(3)}) \tag{2.6}$$

This is called the *forward propagation* step and this network is called a *feed forward neural network*.

2.2.1 Properties of CNNs

CNNs are multi-layered neural networks that are specifically adapted to computer vision problems where images are extremely high dimensional inputs. A gray scale image of 150 X 150 pixels has 22500 different input elements or an RGB image has 3 X 22500 pixels (3 times more) input elements to manipulate. This implies that there will be unmanageable number of parameters to handle, over fitting problems and increased computational time. CNNs are characterized by three different properties that reduce the number of parameters :

- Local Connectivity: For high dimensional inputs it is impractical to connect neurons to every neuron in the previous layer. Each unit in the hidden layer is connected to a subset (patch) of the input image as shown in Figure 2.6a. The hidden unit will also be connected to all the channels or entirely along the depth of the input volume (RGB has a depth of 3 and gray scale has a depth of 1) as shown in Figure 2.6b. This patch is known as the **receptive field** of the neuron with dimensions r x r, which is equal to the size of the kernel that is applied to the image.



(a) Local connectivity and receptive fields

(b) Local Connectivity across channels

Figure 2.6: Local Connectivity

- Parameter Sharing: It segments the units in a hidden layer into different **feature maps**. The hidden units within a feature map share the same parameters and cover different portions of the image as shown in Figure 2.7. Here, W_{ij} connects the ith channel to the jth feature map. Within a feature map receptive fields occur at each possible position within the image. This reduces the number of parameters further and allows us to extract same features at every position. The

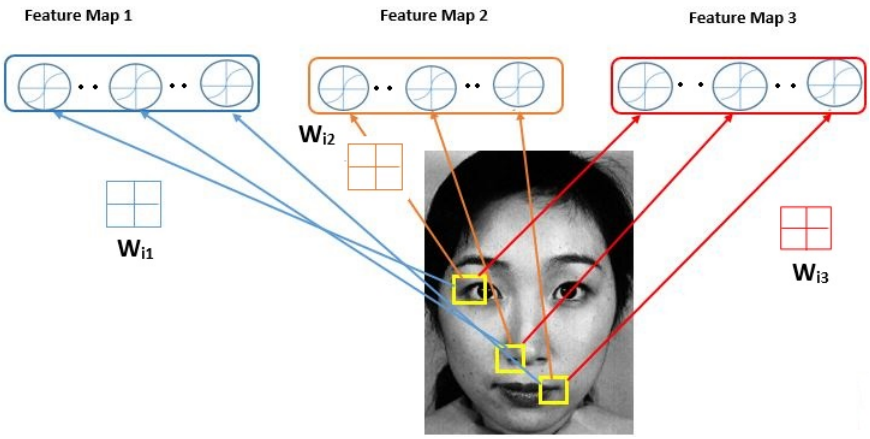


Figure 2.7: Parameter sharing

feature maps corresponds to transformation that is *equivariant*, which implies if a translation is applied to the input image, the feature map will visually contain the same transformation [20].

- Pooling and Sub-Sampling: It aggregates the activations of a set of hidden units within a neighbourhood in a given feature map. The *Max* pooling operator computes the maximum value of the hidden units within the local neighbourhood that summarizes the characteristics of all the units. The pooling is performed on non-overlapping neighbourhoods and results in a smaller feature map (sub-sampling). In Equation 2.7, $x_{i,j,k}$ is the value of ith feature map at position $\{j,k\}$, p is the vertical index in the local neighbourhood, q is the horizontal index in the local neighbourhood and y_{ijk} is the pooled and sub-sampled layer.

$$y_{ijk} = \max_{p,q} x_{i,j+p,k+q} \tag{2.7}$$

The pooling operation reduces the number of hidden units in the next layer. Therefore, computing the subsequent convolution layer is more efficient. The

max operator introduces invariance with local translations of the object.

Discrete Convolution

Instead of performing a linear summation of the input and filter weights as given in Equation 2.1, a discrete convolution operation is used for computing the forward propagation in a CNN. Hidden units in each layer will be segmented into feature maps as described in previous section and for each feature map, a discrete convolution given in Equation 2.8, will be performed to compute the pre-activations of all hidden units within the feature map.

$$h_{w,b}(x) = f(b + \sum_{i=1}^{d} k_i * x_i) \tag{2.8}$$

where f(.) is the ReLU non-linearity. The convolution of an image x with a kernel k (r x r) is computed as in Equation 2.9:

$$(x * k)_{ij} = \sum_{p,q} x_{i+p,j+q} k_{r-p,r-q} \tag{2.9}$$

where p and q are offsets and varies $\{0, ..., r-1\}$. For simplicity, the connection matrix $\widetilde{W_{ij}}$,with the rows and columns flipped shown in Figure 2.8 will be equal to k. The non-

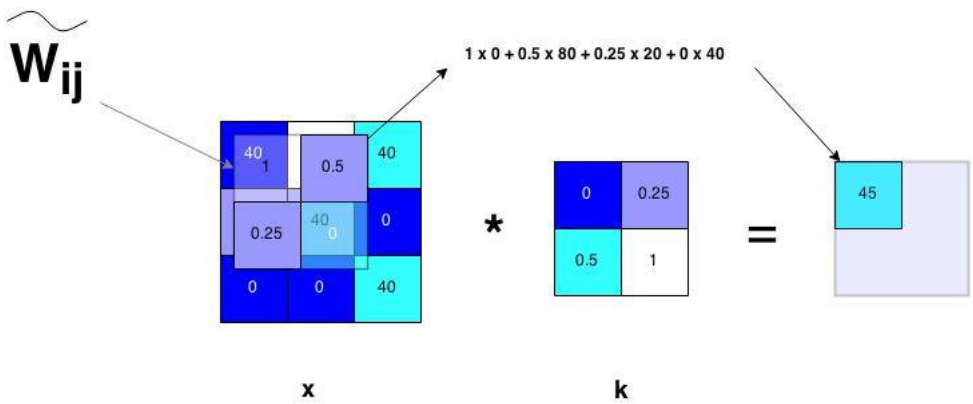


Figure 2.8: Discrete Convolution

linearity helps to emphasize when there is a correspondence with a learned filter and a particular region in an image that contains a feature such as an edge.

2.2.2 Full ConvNet Architecture

A fully connected CNN (Full ConvNet) is built by stacking *Convolutional layers (CONV)*, *Pooling layers (POOL)* and at the output a *Fully-Connected layer (FC)*.

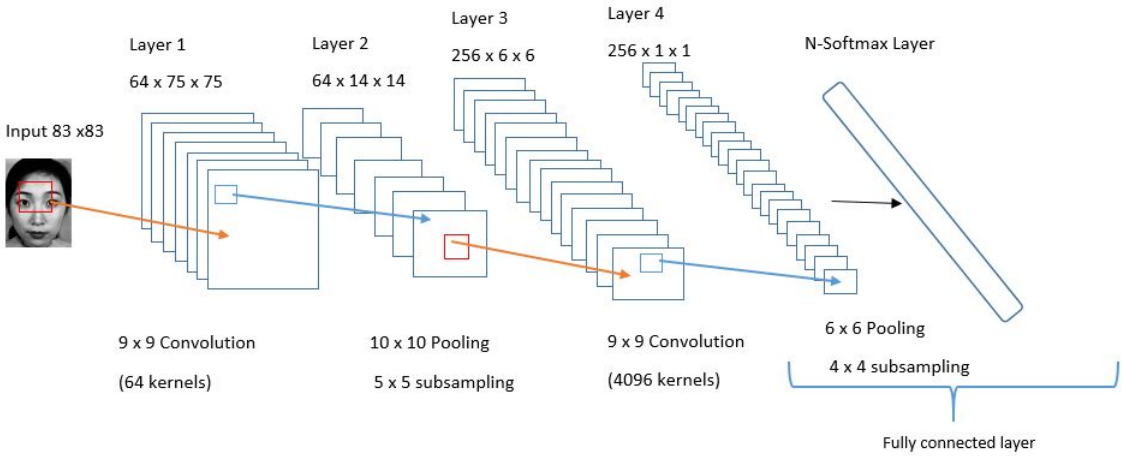


Figure 2.9: Full ConvNet Architecture

- The CONV layer consist of a set of learnable filters or kernels. Each filter slides across the input image and compute the discrete convolution between the filter and the receptive field of the input. If the input has multiple channels or depth (For example an RGB image), then the convolution outputs are summed together, followed by the non-linearity.

- The POOL layer involves a pooling operation such as max operator followed by a sub-sampling operation that down samples the features maps using the max value.

Figure 2.9, shows alternately stacked CONV and POOL layers. The input image is convolved with 64 kernels each of 9 x 9 to form 64 feature maps. Each feature map at layer 1 is of 75 x 75 due to the convolution operation. If *zero padding* is used for the input image then the feature map size would be the same size as the input image. This operation is followed by the pooling operation and sub-sampling operation that uses a fixed function such as a max operator to obtain 64 feature maps of reduced dimensions 14 x 14. At layer 3, all of the 64 feature maps at layer 2 are now the input channels for each feature map at layer 3. Therefore, each feature map will have a kernel size

of 64 x 9 x 9, which is a large amount of computation. In practice, random subsets of feature maps at layer 2 are assigned to each feature map at layer 3 to reduce the number of parameters and computation. The penultimate layer is then fully connected to an output layer that can be a classifier such as a multi-class linear support vector machine (SVM) or a n-softmax classifier [38].

2.2.3 Related Work

This subsection covers a brief review of CNNs used in facial image processing and Table 2.1 gives details of the architectures published that have achieved state of the art facial recognition rates.

[46], focussed on learning deep CNNs, to recover faces in arbitrary poses and illumination to frontal faces (canonical view) with normal illumination. The authors proposed a new face representation with *Facial Identity Preserving (FIP)* features by combining the feature extraction layers with reconstruction layer. The reconstruction layer used the canonical view of the original input image as the supervision criteria instead of a class label. This gave the deep structure more resistance to over fitting. These features were robust to intra-personal variation (such as pose, expressions and others) whilst maintaining high discriminative power between identities. Three locally connected layers and two pooling layers are stacked alternately to encode the input into FIP features. The transformed or reconstructed faces were then used to perform facial recognition.

[14], derives a face representation called *DeepFace* from a nine layer deep neural network. The three-fold contributions of this paper were, a learned representation coupled with 3D face alignment trained on a large facial database containing faces in unconstrained environments. The first three layers were conventional convolution-pooling-convolution layers. The subsequent three layers are locally connected, followed by 2 fully connected layers. Only a single max pooling layer was used to maintain a balance between making learned features robust to local translations and preserving local texture details.

[36], proposes a face representation, referred to as *Deep hidden IDentity feature (DeepID)*. DeepID achieved an accuracy of 97.45 % by using only identification information to supervise the CNN training. These features are learnt my combining a small collection

of CNNs (patch or network fusion) instead of using one deep CNN architecture. One small network consists of 4 convolutional layers, 3 max pooling layers and 2 fully connected layers. Data augmentation was also used to expand the data set, where the input to a CNN consisted of a particular face patch. The outputs of all CNNs were combined to form one over-complete representation. All the identities were classified from the training set simultaneously which helped learn features that were highly discriminative and compact with good generalization ability.

Further improving the performance of DeepID, [33] used both verification and identification as supervision signals to form *DeepID2* feature representation. It was argued that using either of the signals as standalone is not as effective and should be employed together. The aim was to reduce intra-personal variation by imposing the constraint that every two DeepID2 feature vectors extracted from the same identity are close to each other while those extracted from different identities are kept away and increase the inter-personal variation by classifying different identities simultaneously. Joint Bayesian model was utilized to model the verification task.

Table 2.1: Overview of published CNNs

	Input Image	**Architecture**	**No. of Parameters**	**Patch Fusion**	**Feature Length**	**Training Set**	**Output layer**
FIP	96X96X1	L1:32X96X96, M2 L3:32X48X48, M4 L5: 32X24X24 F6	1M	No	-	754,204 images 337 subjects	No classifier, Reconstruction layer used
DeepFace	152X152X3	C1:32X11X11,M2, C3:16X9X9, L4: 16X9X9, L5:16X7X7, L6:16X5X5, F7, F8	120M+	No	4096	120M+ images 4K+ subjects	n-softmax
DeepID	39X31 X{3,1} 31X31 X{3,1}	C1:20X4X4, M2, C3:40X3X3, M4, C5:60X3X3, M6, C7:80X2X2, F8, F9	101M+	Yes	19200	202K+ images 10K+ subjects	n-softmax
DeepID2	55X47 X{3,1}	C1:20X4X4, M2, C3:40X3X3, M4, C5:60X3X3, M6, C7:80X2X2, F8, F9	101M+	Yes	180	202K+ images 8192 subjects	n-softmax for identification and Joint Bayesian for verification

2.3 Supervised CNN

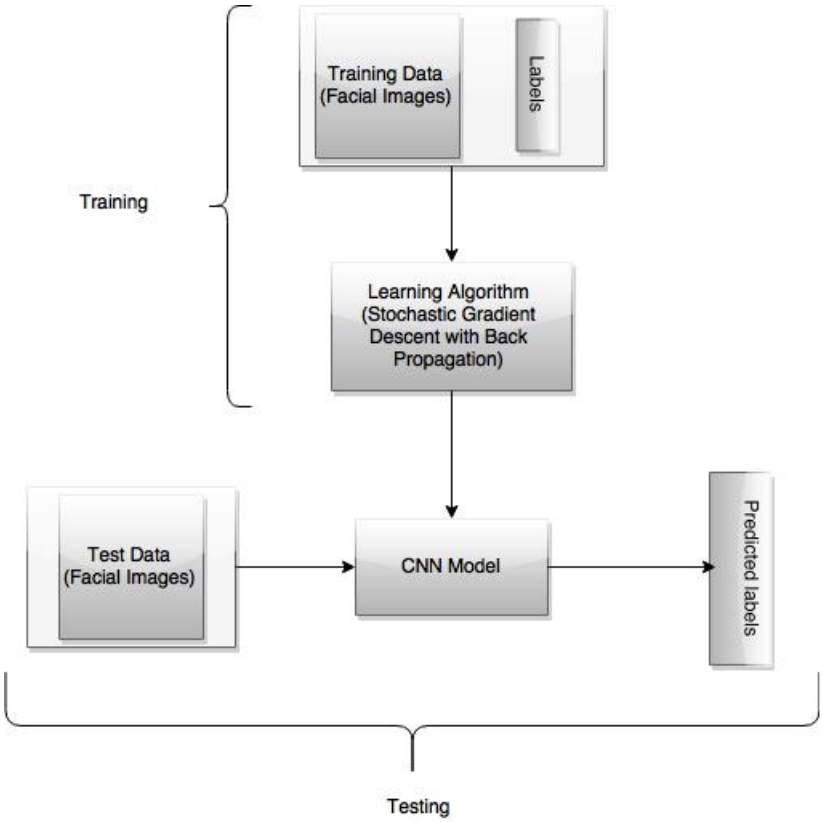


Figure 2.10: Supervised Learning Pipeline

The supervised learning system has been employed in this project to build a CNN model and evaluate it on unseen data, as shown in Figure 2.10. It consist of two stages *training* where the model is trained using a training algorithm and *testing* where the trained model is now tested on examples that are unseen during the training stage. The parameters of the CNN model such as the filter weights and biases are learnt by training the CNN model using **Stochastic Gradient Descent** with **Back propagation** [32] on labelled facial images.

The *training error rate* is calculated during training phase which determines how well the model has fit the training data. The *validation error* helps determine the optimum hyper parameters of the model and the *testing error rate* of the model measures the generalization capability of the model to new data. One of the drawbacks of this model is that if new training examples arrive then the old model has to be re-trained

from scratch to include these new examples as well.

2.3.1 Training CNN architectures

Training a CNN involves finding the right set of weights and biases that will activate when the CNN observes certain features within an image. The training of CNN done in this project is *off-line*, where the training set is presented to the CNN as a single unit, or in batches and the errors are accumulated. These accumulated errors are then used to update the weights and biases.

The training of CNNs can also be viewed as solving an *optimization* problem, to find the optimum values for W and b which minimizes the *optimizer* given in Equation 2.10.

$$\arg\min_{W,b} = \frac{1}{T}\sum_{t} J(f(x^t;W,b),y^t) + \lambda\Omega(W,b) \tag{2.10}$$

where x is the input of the tth sample and y is the corresponding label. $J(f(x^t;W,b),y^t)$ is the loss function which compares the prediction $f(x^t;W,b)$ with actual label y^t, averaged over T samples. $\Omega(W,b)$ is the *regularizer*, that is used to penalize the certain values of W and b, and λ is the regularization term that controls the balance between the optimization of the loss function and the regularizer. There are two types of regularizer that can be employed in the above optimizer equation: L2 regularization penalizes the square of the weight value $\lambda\sum_i W_i^2$ while L1 regularization penalizes the absolute value of the weight $\lambda\sum_i |W_i|$. However, in general the purpose of regularization is to ensure the model learns to generalize for unseen examples in the test set.

Since the classification problem in this project is a multi-class classification problem, the loss function used is a n-softmax function as described in subsection 2.3.2. The cross entropy loss for a n-softmax Equation 2.15 is given by:

$$J(W) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{n} 1\left\{y^{(i)} = k\right\} log\frac{exp(W^{(k)T}x_i)}{\sum_{j=1}^{n} exp(W^{(j)T}x_i)}\right] \tag{2.11}$$

where $y^{(i)} \in 1,....,n$ where n is the nth class label, m is number of labelled examples and $J(W)$ is summed over all n different possibilities of a class label.

Stochastic Gradient Descent (SGD)

SGD is a technique in machine learning that is used to solve the optimization problem given in Equation 2.10. To decrease the error, the parameters should be updated in the direction of the negative gradient shown in Figure 2.11. Unlike the convex function (shown to the left) in this figure, $J(W)$ is a non-convex function (shown to the right) that is susceptible to local minima. However, in practice gradient descent is capable of converging to an optimum solution. First, the weights and biases are initialized to random values from a normal distribution near 0. The purpose of *random initialization* is *symmetry breaking* [39]. Second, for each training example the gradients are computed with respect to the parameters using Back Propagation and using these gradients the parameters are updated iteratively, given by algorithm 1. Algorithm 2, describes

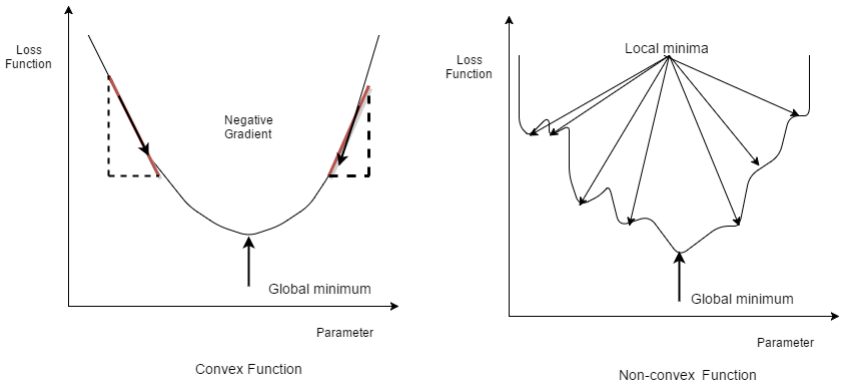


Figure 2.11: Error Function Versus Parameter

the steps for *Mini-Batch Gradient Descent* which is faster than SGD. The main difference lies in the number of training examples considered for a single iteration. In SGD, one example is used per iteration where as in mini-batch gradient descent a smaller set of B examples are used. Mini-Batch Gradient Descent offers the advantage of lesser variance in parameter updates as it sees a group of examples prior to its update, leading to a stable convergence. In this project, the mini-batch gradient is chosen where the optimal batch size B is chosen based on the given architecture. Finally, α is the learning rate that defines how large or small the update can be. In practice, the value for α is chosen to be a small and constant value for the initial epochs and later it is annealed over time[4]. Additionally, it is good practice to randomly shuffle the data prior to each epoch as this prevents bias of gradients which can lead to poor convergence [39]. Momentum method is an extension to SGD that is used to increase the speed

Algorithm 1 Stochastic Gradient Descent

Input: data $x^{(t)}$
Initialize $W^{(1)}$, $b^{(1)}$,....,$W^{(L)}$, $b^{(L)}$
for epoch=1 **to** N **do**
  for each training example $x^{(t)}, y^{(t)}$ **do**
    Use back propagation to compute gradients $\nabla_{W(l)} J(W,b;x,y)$ and $\nabla_{b(l)} J(W,b;x,y)$
    Set $\Delta W^{(l)} \leftarrow \Delta W^{(l)} + \nabla_{W(l)} J(W,b;x,y)$
    Set $\Delta b^{(l)} \leftarrow \Delta b^{(l)} + \nabla_{b(l)} J(W,b;x,y)$
    Update $W^{(l)} \leftarrow W^{(l)} - (\alpha \Delta W^{(l)} + \lambda W^{(l)})$
    Update $b^{(l)} \leftarrow b^{(l)} - (\alpha \Delta b^{(l)})$
  end for
end for

Algorithm 2 Mini-Batch Gradient Descent

Input: data $x^{(t)}$
Initialize $W^{(1)}$, $b^{(1)}$,....,$W^{(L)}$, $b^{(L)}$
for epoch=1 **to** N **do**
  for each i=1 **to** B **do**
    Use back propagation to compute gradients $\nabla_{W(l)} J(W,b;x,y)$ and $\nabla_{b(l)} J(W,b;x,y)$
    Set $\Delta W^{(l)} \leftarrow \Delta W^{(l)} + \nabla_{W(l)} J(W,b;x,y)$
    Set $\Delta b^{(l)} \leftarrow \Delta b^{(l)} + \nabla_{b(l)} J(W,b;x,y)$
    Update $W^{(l)} \leftarrow W^{(l)} - (\alpha[\frac{1}{B}\Delta W^{(l)}] + \lambda W^{(l)})$
    Update $b^{(l)} \leftarrow b^{(l)} - (\alpha[\frac{1}{B}\Delta b^{(l)}])$
  end for
end for

of learning when the loss function contains long, narrow local minima. This method adds a fraction β of the previous weight update to the current update [39] which helps smooth out the noise and oscillations that gradient descent has in directions of high curvature of the loss function [4].

Back Propagation Algorithm

It is a procedure to compute the parameters gradients with respect to parameters W and b. Given in algorithm 3, for each training example (x,y), the forward pass is computed as described in section 2.2, which computes all the activations of the network [11]. Then, for each node i in the output layer n, the error δ_i^n is calculated that measures the

Algorithm 3 Back Propagation Algorithm

Perform a feed forward pass, computing activations for layer L_2, L_3,...L_n
for each node i in the output layer n **do**
$\delta_i^n = \frac{\partial}{\partial z_i^{(n)}} \frac{1}{2} \left\| \left(y - h_{W,b}(x) \right) \right\|^2 = -(y_i - a_i^{(n)}).f'(z_i^{(n)})$
end for
for $l = n-1, n-2, ..., 2$ **do**
for each node i in the output layer n **do**
$\delta_i^l = (\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}).f'(z_i^{(l)})$
end for
end for
Compute the partial derivatives
$\nabla_{W^{(l)}} J(W,b;x,y) = \delta^{(l+1)}(a^{(l)})^T$
$\nabla_{b^{(l)}} J(W,b;x,y) = \delta^{(l+1)}$

error contribution of that node to the output. For hidden units, the error δ_i^l is computed based on a weighted average of the error terms of the nodes that uses $a_i^{(l)}$ as an input. In addition, for CNN architectures the gradients have to be passed through the convolution and pooling operation [20]. Let l be the loss function of a given training example and $y_j = x_i * k_{ij}$ where y_j is the pre-activation of the jth feature map of the convolution operation, then the gradient for x_i of the ith input channel is given by,

$$\nabla_{x_i} l = \sum_j (\nabla_{y_j} l) * (W_{ij}) \tag{2.12}$$

where ∇_{y_j} is the gradient of the loss of the full feature map.
The gradient for W_{ij} is

$$\nabla_{W_{ij}} l = (\nabla_{y_j} l) * (\widetilde{x_i}) \tag{2.13}$$

where $\widetilde{x_i}$ is the row flipped version of x_i. For max pooling operation given in Equation 2.7, the gradient for x_{ijk}:
$\nabla_{x_{ijk}} l = 0$ except for $\nabla_{x_{i,j+p',k+q'}} l = \nabla_{y_{ijk}} l$ where $p', q' = \arg\max x_{i,j+p,k+q}$.

2.3.2 Softmax Classifier

In classification tasks using deep learning techniques it is standard to use a *Softmax function* for multi-class classification [38]. Here, the activation of the output layer of

the CNN is forced to represent a probability distribution where the conditional probability of the input belongs to class $k \in 1, ...n$ given input x: $p(y^{(i)} = k|x)$. For example, a face recognition task that involves 1:1 *verification*, the classifier simply outputs a binary value of 1 (*yes*) or 0 (*no*) and a task that involves 1:N such as *identification* where the output is given in terms of a confidence measure of the identity of the person when compared with the enrolled faces in the database. For a facial expression recognition task the classifier outputs a confidence measure for each of the expressions.

The n-softmax classifier receives some input a_i accumulated from the layer below, as given in Equation 2.14, and outputs $p(y^{(i)} = k|x^{(i)})$ that not only depends on its own a_i but depends on all other a_j, this relationship is given by Equation 2.15. W is the weight connecting the penultimate layer to the softmax layer.

$$a_i = W^{(k)T} x^{(i)} \tag{2.14}$$

$$p(y^{(i)} = k|x^{(i)}) = \left(\frac{exp(a_i)}{\sum_j^n exp(a_i)} \right) \tag{2.15}$$

The predicted class $\hat{i}$ would be the one with highest estimated probability:

$$\hat{i} = \arg\max_i p(y^{(i)} = k|x^{(i)}) \tag{2.16}$$

2.4 Transfer Learning

The right internal representation of data can have a huge impact on the performance of traditional machine learning algorithms as described in section 2.1. Here, the underlying assumptions is that the training and test data come from the same distribution or feature space $D_s = D_t$, where D_s is the source domain and D_t is the target domain [29]. However, if this distribution changes $D_s \neq D_t$, the machine learning model will have to be re-built using the newly labelled training data, which is expensive. Hence, in such scenarios extracting knowledge from one or more source tasks and transferring it to a target task proves beneficial [3]. Figure 2.12, highlights this difference between traditional machine learning algorithms and transfer learning techniques.

The formal definition of transfer learning is given in [29], which states that given a source domain D_s and learning task T_s, a target domain D_t and learning task T_t , transfer

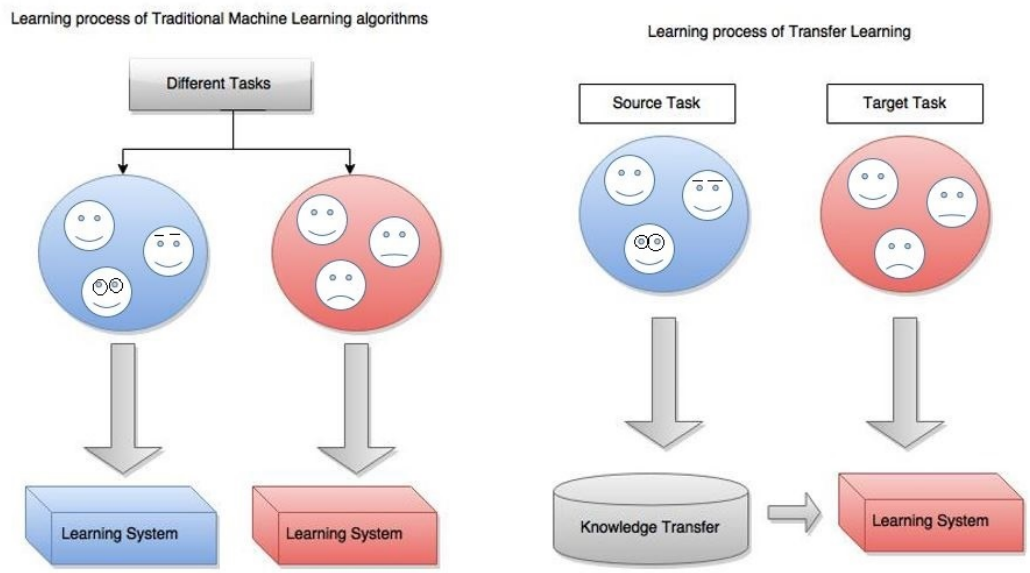


Figure 2.12: Difference between Traditional Machine Learning process and Transfer Learning

learning aims to help improve the learning of the target predictive function $f_T(.)$ in D_T using the knowledge in D_s and T_s, where $D_S \neq D_T$, or $T_S \neq T_T$. Here, $D = \{X, P(X)\}$ is the *domain* consisting of two components feature space X and marginal probability distribution $P(X)$, where the input vectors $X = x_1, ..., x_n \in X$ and *task* $T = \{Y, f(.)\}$ consists of two components a label space y and a predictive function $f(.)$.

There are different categorizations of transfer learning based on different situations of source and target domains and tasks [29]. This project falls in the category of *inductive transfer learning*, where the target task is different from the source task, and the source and target domains are the same.

2.4.1 Related Work

This section covers a brief review of recent research being done in the field of transfer learning.

In [43], the author successfully investigated the transfer learning aspect of deep neural networks for the task of object recognition. The contributions made by the author were three-fold which addressed the generality and specificity of neurons in each layer for a deep CNN architecture used in the ImageNet challenge [23]. The author also

proved the decrease in transferability of features when the base task and target task are different by using two dissimilar datasets (Man-made versus Natural objects) and finally, the author showed that transfer of features from any layer is likely to increase generalization even after fine-tuning to the target dataset.

The authors of [30], introduced a novel approach of transfer learning called *Self-taught learning*, to improve performance on supervised learning tasks when there is a scarcity of labelled data. Unlike the experiment in this project where labelled data is available to perform supervised learning, the approach in this paper falls in the category of inductive transfer learning *without* labelled data. The authors used an algorithm on sparse coding to learn abstract representations of images in terms of low level features like edges from images downloaded from the internet and applied this learned representation or knowledge to labelled data and observed improved performance.

2.5 Chapter Summary

This chapter provides a comprehensive description of all the background material gathered during the literature review phase of the project. It describes in detail various representation learning algorithms available in literature. It then focusses, on representation learning algorithms based on deep learning techniques called deep CNNs. Finally, this chapter concludes on the utility of multi-level representation for transfer learning.

Chapter 3

Problem Statement and Methods

This chapter provides a description of the research problems and the research methodology of the project. It integrates the ideas developed in the background study into a single comprehensive transfer learning system architecture. In addition, it highlights any decisions made regarding the selection of parameters for the CNNs. Finally, it concludes with the transfer learning hypothesis and verification methods used in the project.

3.1 Problem Statement and Motivation

One of the major challenges in computer vision is to learn representations that disentangle factors of variations found in images like pose, identity and others. In particular, the performance improvements in facial expression recognition systems has received a setback due to the dominance of a certain factors of variation such as identity in the representation of images in raw pixel space [31]. This is due to the fact that images of two identities are well separated in pixel space while the images of the same identities with two different expression are found close together.

There are a number of approaches in literature that have focussed exclusively to disentangle specific information suited for a given problem. For the task of FER, the authors of [13] proposed a solution based on semi-supervised learning. In the first stage, a set of feature maps are learnt using a multi-scale contractive convolutional network which are pooled successively to learn translation invariant representations. In the second stage, a unsupervised algorithm based on Contractive auto-encoder is used to train two

blocks of features: one sensitive to emotion labels while the other insensitive to emotion label. These two blocks are jointly trained such that they vary orthogonally to each other. Finally, the block sensitive to emotion label is fed to a linear SVM to perform classification.

Unlike this previous approach that develops a system to extract information suited to the task of FER, this study aims to experimentally test the hypothesis of an implicit transfer learning process in deep neural networks to disentangle different facial information components for FR and FER. If this hypothesis holds true, systems can be developed that utilize this disentangled representation to improve the performance of existing systems mentioned in section 2.1.

The motivation for this approach relies on the hierarchical nature of feature representations learnt by deep CNNs. This hierarchy of features transitions from common features at lower levels to specific features at higher levels. As the features become more specific, the transfer of knowledge suffers. This drop in transfer learning characteristics at higher levels in the hierarchy should imply that representations learnt at this level have successfully disentangled the factors of variation in data.

3.2 Research Methodology

This section elaborates the different phases of the research methodology used in the project, as illustrated in Figure 3.1.

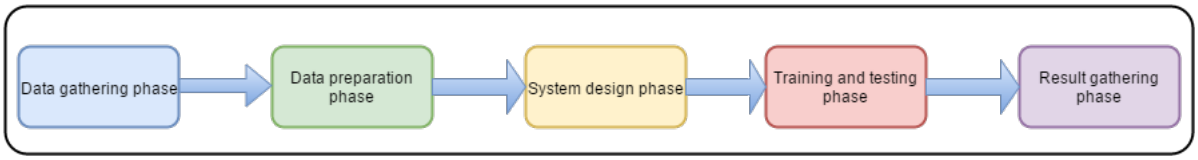


Figure 3.1: Different phases involved in the research

- **Data Gathering Phase:**
  All machine learning algorithms are data driven and acquiring a suitable data set is crucial for the later stages of the process. This phase was the most time consuming as it involved downloading publicly available datasets or placing request

for ones that are privately owned. The data set was selected keeping the objectives of the experiments in mind. There are large number of available data sets, however the difficulty faced at this stage was the non-uniform format of the image files across all available data sets. This includes variation in file extensions like (.jpg, .raw, .png) or some databases are also stored in excel files. Other difficulties include no standard naming conventions for folders or files. Therefore, extracting these images into MATLAB required customizing the input function to suit the database. Once the database was extracted, it underwent the Data preparation phase.

- **Data Preparation Phase:**
  In this phase the data was conditioned into the right format by mathematically modifying the raw pixel values as per the pre-processing pipeline shown in Figure 3.2. The two tasks FER and FR were prepared by manually annotating the data based on expression and identity respectively. After extracting the dataset

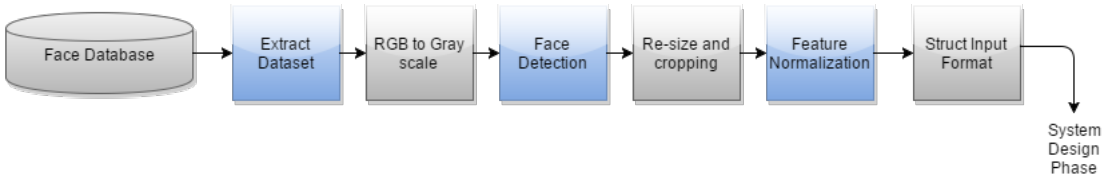


Figure 3.2: Pre-Processing Pipeline

in the previous phase, some or all of the pre-processing steps were executed depending on the characteristics of a particular data set. According to [18], no significant improvement is gained by including color images, and to reduce processing time the image files were first converted from RGB to grey. Next, the images were passed through a face detector based on a pre-defined MATLAB routine using Viola Face Detection Algorithm [42]. This was followed by resizing and cropping the images to 48 x 48 pixels. A feature normalization step was performed to standardize the range off features, as given in Equation 3.1.

$$\hat{x} = \frac{x - \mu_x}{\sigma_x} \tag{3.1}$$

where x $\in$ R and $\hat{x} \in$ R are original and normalised feature vectors respectively. μ_x and σ_x are the mean and standard deviation of x respectively.

The dataset was split into 10 folds and a separate test set. This is because all experiments carried out in this project used **10-fold cross validation**, to ensure results collected from the experiments were statistically equivalent. Finally, all folds were converted into a Struct data structure with 4 fields, which is the standard input format for MatconvNet toolbox, shown in Figure 3.3 . The four fields included an *id* field that provided every image a unique identifier, a *data* field consisting of all images stacked in the third dimension, a *label* field consisting of class labels and a *set* field indicating whether an image belongs to the train, validation or test set.

Field	Value
id	1x213 double
data	48x48x213 single
label	1x213 double
set	1x213 double

Figure 3.3: Struct input format for MatConvNet

- **System Design Phase:**
  This phase involves designing CNN architectures for FR and FER. A set of architecture configurations available in literature were chosen for both tasks and narrowed down using model selection by measuring the validation error and standard deviation of the model. The model with a low validation error and lowest standard deviation is considered to be a better choice than the model that gives lowest validation error but a higher standard deviation. This choice takes into account the *accuracy* of the model along with *model stability* across all folds of the input data. The selected model was then chosen to design the networks to be used in the transfer learning system.

- **Training and Testing Phase:**
  This phase involved training and testing the networks in the transfer learning system described in section 3.4.

- **Result Gathering Phase:**
  The final phase involves analysing the results from training and testing phase with the help of suitable metrics, plots and tables, and providing a negative or positive answer to the research questions on transfer learning aspect in facial information processing.

3.3 Model Selection

Deep learning algorithms have a large number of hyper-parameters that have to be adjusted and selection of these hyper-parameters is known as *Model Selection*. The following section is divided based on the common architectural design choices made and hyper-parameters values selected for the models.

3.3.1 Architecture Design Choices

Designing a 'good' CNN architecture remains an open problem [18], hence the CNN architectures proposed were based on the literature survey conducted on the state of art CNN architectures in subsection 2.2.3, and keeping in mind the size of training data. Performance of CNN architectures are heavily dependent on the size of training set used. The smaller the dataset, the use of larger and complex CNN architectures is prohibitive. The number of hyper parameters to be trained in a CNN architecture increases with increase in architecture size, which is unsuitable to capture the variations in a small dataset. Another reason for the use of a smaller network for less data is that a smaller network (fewer filters and layers) helps avoid over fitting [18].

The architectures followed a pyramidal scheme which is known to make effective use of computational resources [10]. Based on the literature survey conducted in subsection 2.2.3 for FR, CNN architectures can either be trained as a single deep CNN or can be combined with multiple CNN architectures. Each of the CNNs in this combined network focusses on a single patch of the image and are later combined to form a dense feature vector. However, for the sake of simplicity only single deep CNN architectures were considered.

3.3.2 Hyper-Parameter Selection

The selection of hyper-parameters was categorized into two types; optimizer and model hyper-parameters. For the purpose of this study, focus was given to select the hyper-parameters of only the model via model selection.

Hyper-Parameters of the optimizer

- Initial Learning Rate α_0:
  Typical range of values for input that was standardized in (0,1) interval (or feature normalized), is less than 1 and greater than 10^{-6}. Since the architectures considered followed similar designs in [18], the value of α_0 was fixed to 0.001 as reported in this paper.

- Number of iterations or epochs:
  To fix the number of epochs for training, the method of **Early Stopping** was used. Training was stopped either when validation error became zero or validation error reached its minimum peak while training error continued to decrease. This ensured that the network never over-fitted.

- Mini-batch size B:
  Typical values range from 1 to 100's. This size of B affects training time rather than performance. Considering the data set size, B was fixed to 50.

- Momentum co-efficient β:
  The default value of momentum is 1 (no momentum). However, the implementation in MatConvNet uses α and β in its weight update equation and is fixed to 0.9.

Hyper-Parameters of the Model

- Number of hidden units:
  The number of hidden units was selected through the model selection experiment in chapter 5 by varying number of feature maps, filter sizes and fully connected layers. Other settings included no zero-padding and a stride of 1 for convolutional operator and a stride of 2 for the pooling operator.

- Weight decay regularization co-efficient λ:
  The implementation of the training function in MatConvNet, uses L2 regularization to penalize the weights towards zero (see subsection 2.3.1).

- Neuron non-linearity:
  The ReLU non-linearity was used to model the neuron's output as it is successfully used in literature [23].

- Weight initialization:
  Biases were initialized to zero while weights were initialized to random values near 0 to ensure symmetry breaking (see subsection 2.3.1).

3.4 Transfer Learning Hypothesis and Verification Methods

Transfer Learning Hypothesis:
Given a task of FR or FER, our hypothesis is that there is some shared explanatory factors of variation between both tasks such that it facilitates transfer of knowledge between the two task. This knowledge transfer will benefit both tasks until the point that these explanatory factors disentangle into representations that are specific to either task.

This implies that through transfer learning in deep architectures like CNNs, it is hoped the degree to which a layer is general or specific can be quantified. Generic or common features between two tasks implies that shared factors of variation should exists between the two tasks while specific features implies the representation learnt has no shared factors of variation and has disentangled to its individual factors.

To conduct the transfer learning experiment in this project, the question of *"What"* is to be transferred, is addressed here. From subsection 2.2.2, CNNs consists of alternating CONV and POOL layers that form a hierarchy of feature representations. In CNNs the specificity and generality are dependent on the kind of filters that are learnt in the CONV layers of the CNN architecture. The first level filters tend to represent *Gabor* edge like filters and color blobs. These low level filters are learned are common across all natural image datasets and can be used for knowledge transfer in CNNs. Filters higher up in the hierarchy tend to be more specific to a particular task and are less suited for transfer learning. In the experiments carried out in this project, the knowledge are the *filters* that are learned by the CNNs for both FR and FER.

Finally, for this project two verification methods were used to verify the transfer learning hypothesis in facial information processing. Method 1, described in subsection 3.4.1, was chosen to explicitly addresses the following questions in transfer learning for facial images:

1. To quantify the degree to which a layer is specific or general.

2. To analyse whether this transition occurs at a specific layer or is spread across several layers.

3. To estimate where this transition takes place: near the first, middle, or last layer of the network.

Method 2 described in subsection 3.4.2, was chosen to provide a visual representation in terms of transfer learning.

3.4.1 Verification by Transfer Learning Experiment

The first method chosen to verify the hypothesis was based on the experimental protocols by authors in [43]. The transfer learning system along with the various phases in the project has been highlighted in Figure 3.4, and is described in the following steps:

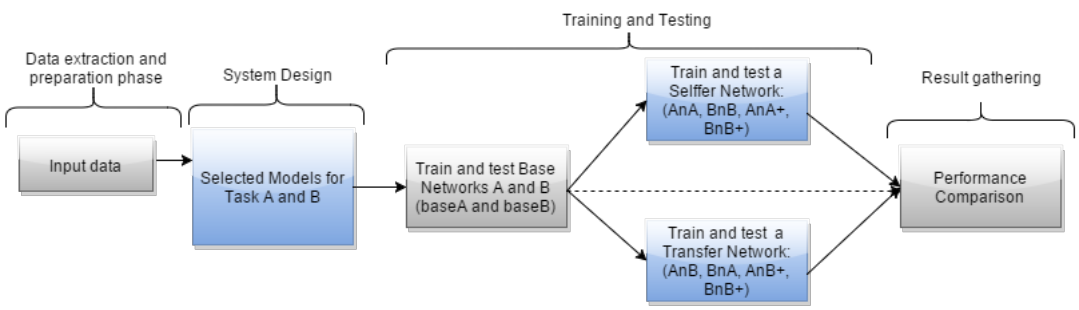


Figure 3.4: Block diagram of the Transfer learning system

- Input Data
  The input to the transfer learning system was divided based on the two tasks: task A (or FR) and task B (or FER). Additionally, the input consisted of same facial images of a cohort of the fixed subjects in the two tasks to verify the transferability on the exactly same condition of variation in identity and expression information only.

- Selected Models for Task A and B
  The models selected to test the transfer learning hypothesis, were narrowed down using model selection described in chapter 5.

- Train and Test Base Networks A and B
  The chosen models were first tested for face recognition (task A) and facial expression recognition (task B) without transfer learning called the base network *baseA* and *baseB*.

- Transfer Learning Experiment
  The architectures were then modified to incorporate transfer learning. For a given task B, a layer n $\{1, 2..n\}$ was chosen from these trained networks to create two types of networks:-

  [1] *Selffer network BnB*: Selffer network BnB or BnB+: the first n layers were copied from baseB and frozen (BnB) or fine-tuned (BnB+). The next higher layers were initialized randomly and trained on dataset B.

  [2] *Transfer network AnB*: tthe first n layers were copied from baseA and frozen (AnB) or fine-tuned (AnB+). The next higher layers were initialized randomly and trained toward dataset B.
  This was repeated for all n and in both directions *AnB* and *BnA*.

- Performance Comparison
  A performance comparison was made of the selffer network, transfer network and base network to study the effects of transferability. Focussed was given to measure performance gain or drop by measuring top-1 or top-5 accuracy.

3.4.2 Verification by Visualization

Another verification method chosen was the visualization technique introduced in [44] to identify the input stimuli that activates a particular feature map at a particular layer. Much of the work carried out using CNNs given in subsection 2.2.3 offers very little insight about the operation and behaviour of these models. One has to rely solely on the metric of accuracy as a performance measure for these models.

This technique maps feature activations at all layers to the input pixel space. This mapping is based on [45], where a Deconvolutional network (deConvNet) is used to reverse all the operations carried out in the Convolutional network (ConvNet) in the same order as shown in Figure 3.5. To visualize a particular activation of a feature map at a given layer, all other activations at that layer are set to zero. These feature maps are then passed through a deConvNet to be unpooled, rectified and deconvolved

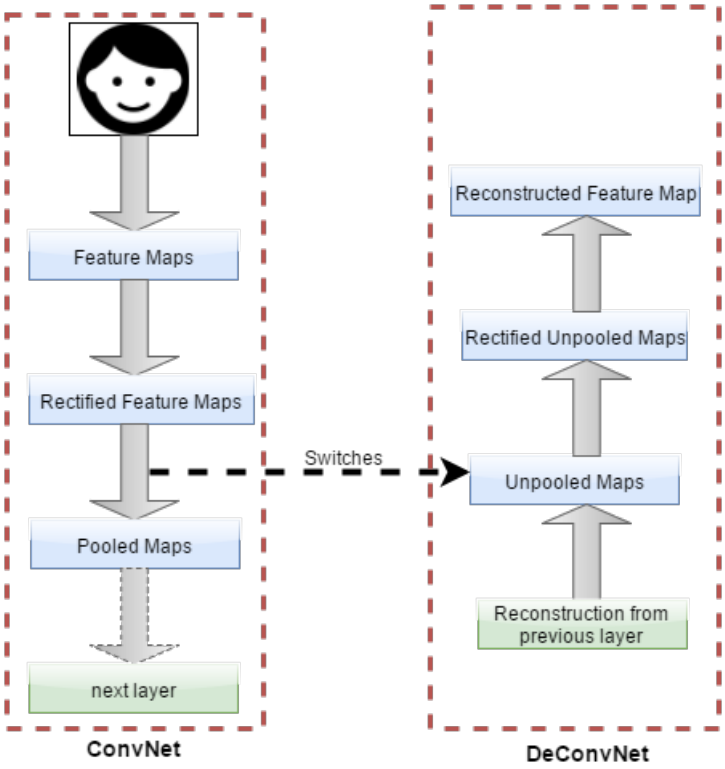


Figure 3.5: Illustration of the reconstruction of maximally activated feature maps

until the input space is reached, as described in the original paper [44].

Modified method of unpooling:

To carry out the reconstruction process of the maximally activated feature maps, the most efficient way is to keep track of the location of max activations via *switch variables* during the forward propagation step. Figure 3.6a, provides an illustration of the unpooling process. It shows the process of 2D Max pooling where each similarly coloured region in the Feature Maps z, depicts a local neighbourhood region. The max values of these local regions are recorded in a Pooled Map p and the location of this max value is recorded in a switch variable s. It also shows the process of unpooling into Unpooled Feature Maps $\hat{z}$ where the max value has been place in its original position in the feature map with the help of the switches.

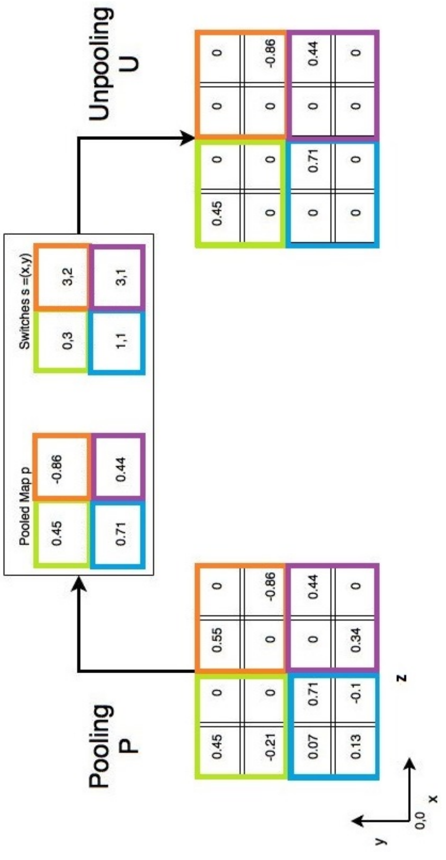
However, MatConvNet does not allow modification of its pooling function to extract these switches. To bypass this, a method was developed to approximately obtain the

switch values as shown Figure 3.6b. This involved comparing the feature maps of the conv layer and its corresponding pooling layer and performing a search in the conv layer feature map (record the positions in the switch variable) for every value present in pool layer feature map. The search was performed in a 2 x 2 local region in the conv layer since a max pooling of 2 was used. A reference to the code for calculating the switch variable and unpooling is provided in Appendix A.

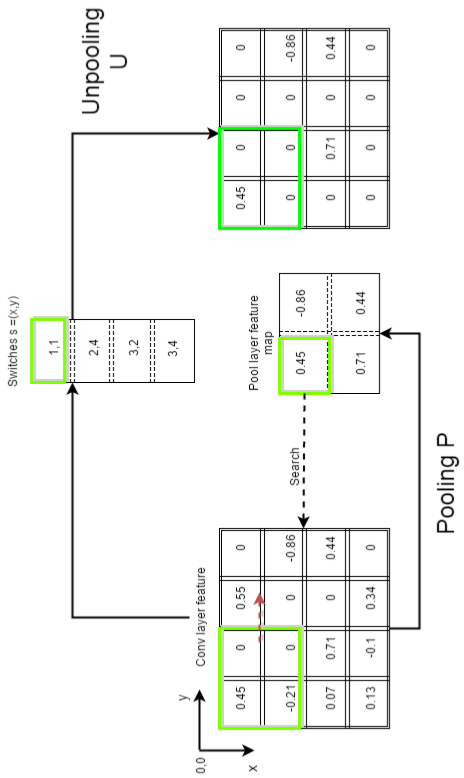
The drawback of this method is the inability to find the index of max activation when there are zero values present in a local region. Such a case arises when a reLU layer after a conv layer which rounds negative values to zero. In this case, the location is fixed to the first location in the local grid i.e, (1,1).

3.5 Chapter Summary

This chapter describes the research problem this project addresses along with the motivation for the approach adopted to tackle this problem. A detailed account has been provided regarding the research methodology while highlighting general issues faced at each phase. The design choices for the architecture and parameters has been stated. Finally, a description of the transfer learning hypothesis along with verification methods has been described.



(a) Unpooling Operation



(b) Modified Unpooling operation

Figure 3.6: Unpooling operation

Chapter 4

Implementation

This chapter provides a brief discussion of the technologies used in the experiments carried out during this project. It explains in brief about the Open-Source MATLAB Toolbox used for designing the CNNs. In addition, it provides simple flow chart representation of the software code written for this project.

4.1 MatConvNet Toolbox

The CNNs architectures used in the experiments in chapter 5 were developed using a publicly made available MATLAB toolbox called MATCONVNET [40]. This toolbox can be used for fast prototyping of CNN architectures where the implementation of all the building blocks of CNN are provided with simple MATLAB functions. This toolbox was used to develop small, medium and large scale CNN architectures in [18]. Some key features of this toolbox is its efficient support for GPU computation based on NVIDIA CUDA and MATLAB built-in CUDA capabilities, its flexible implementation in MATLAB which can be easily modified along with minimum set up and system requirements during installation.

Current version used for this project uses the Convolutional block (vl_nnconv(x, f, b)), that computes the convolution of the input map X with a bank of K multi-dimensional filters f and biases b, Convolution transpose block (vl_nnconvt), that implements the transpose operator of the convolution, Pooling block (vl_nnpool), which implements max and sum pooling, Activation functions ReLU (vl_nnrelu). and Softmax (vl_nnsoftmax), for the n-softmax loss. The description for the aforementioned functions can be found in [41].

4.2 System Information

All software developed for this project was develop on a system with the following specifications:

Table 4.1: System Information

System Information	
Operating system	Microsoft Windows 8.1
Processor	Intel(R) Core(TM) i5 @ 1.70 GHz, Dual Core
Memory - RAM	4GB
Graphics Card	Nvidia GeForce
MATLAB Version	2014b
MatConvNet Toolbox Version	1.0-beta13
C++ Compiler	Visual Studio 2012

4.3 Implementation

This section deals with the flow chart diagrams of all the software code developed for the verification methods described in section 3.4. It takes the concepts developed in chapter 2, applies the methodology in chapter 3 and uses the available library functions in MatConvNet toolbox described in section 4.1 to develop a system that can be used for experiments in chapter 5.

4.3.1 Method 1: Transfer Learning Experiment

The system shown in Figure 4.1, assumes that the model selection experiment has been performed and the required base networks for A and B have been trained. The number of cases 'N' is a variable that stores the number of selffer and transfer networks (inclusive of both frozen and fine-tuning cases) to be trained. The experiment is performed on 10-folds. With each fold there are 'N' cases of selffer and transfer networks that are trained. While loading the CNN architecture, the epoch at which the filters are to be transferred is passed as an argument to the function. In the main function the hyper parameters like learning rate α, Number of epochs, β and B are set. Next, the training function from MatConvNet is called to train the given network. After training, the results are stored and the next network to be trained is loaded. Finally, the process ends when all networks are trained with all folds of the data.

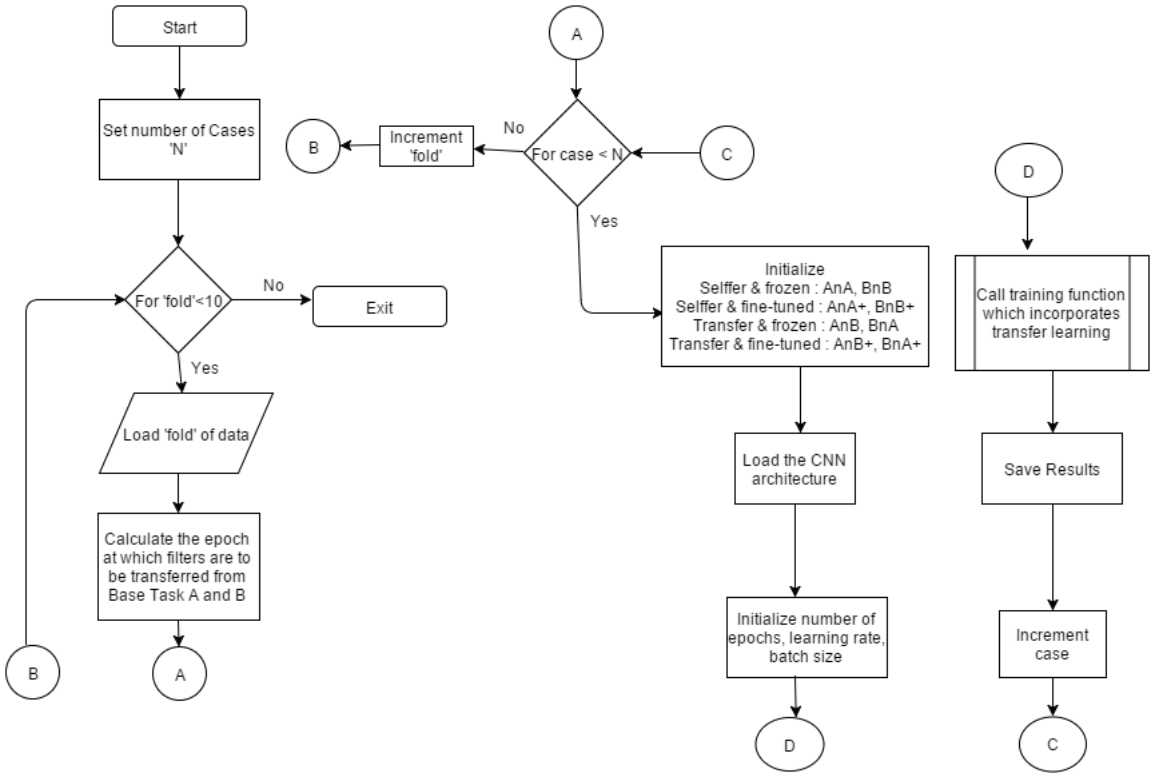


Figure 4.1: Flow chart of the Transfer Learning System

4.3.2 Method 2: Visualization of CNNs

The flowchart shown in Figure 4.2, assumes the given CNN network is fully trained and that learned filters 'net' at each layer is available. The process is explained in two phases. The first phases mainly focusses on passing all the test images through the network and ranking the test images in the order from maximum activation to minimum activation for a given feature map. After the test images are sorted, a function is called where the feature map to visualize at a specific layer is specified. The chosen feature map is then activated by setting all other feature maps to zero and is passed to another function to perform the reconstruction process and as shown in Figure 3.5. Finally, the process exits upon fully reconstructing the feature map in the raw pixel space. The second phase involves the actual reconstruction process of the feature map based on subsection 3.4.2. The set of feature maps is passed through a deConvNet and recursively unpooled, rectified and deconvolved until the input or raw pixel representation is reached.

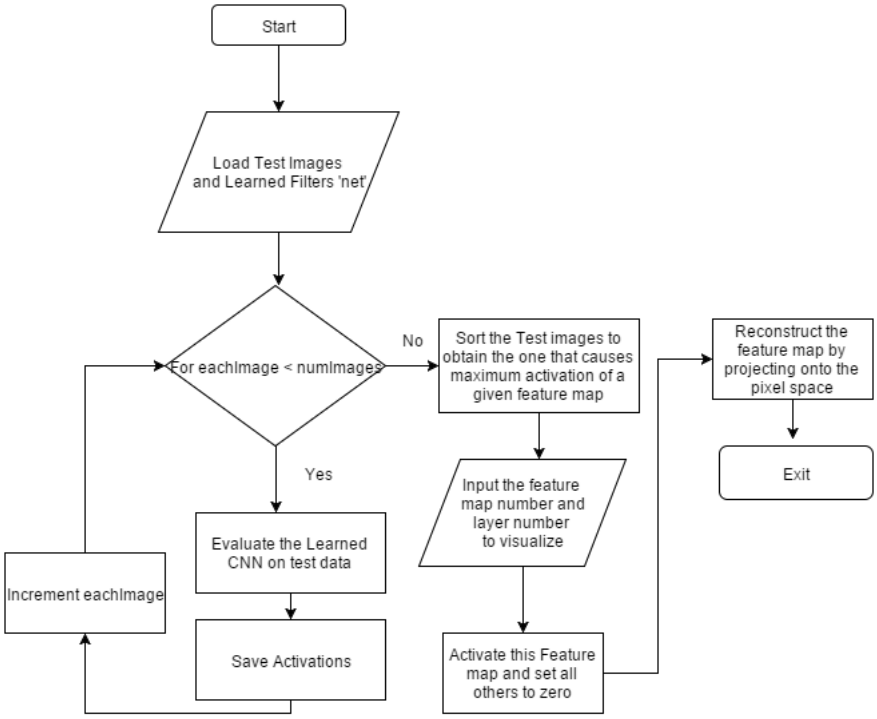


Figure 4.2: Flowchart for the Visualization of CNNs

4.4 Chapter Summary

This chapter provides a brief overview on the MatConvNet toolbox used to develop the software code for this project. Details of the CPU system in which the software code was developed is mentioned. The logical flow of the software code developed for this project is explained through simple flow chart representations.

Chapter 5

Experiments

This chapter deals with the experiments carried out during the course of this project. The sections are divided based on three benchmark databases given in Table 5.1. These databases were chosen based on requirements of the two classification tasks FR and FER. These databases allowed the testing of the transfer learning hypothesis with two sources of variation: identity and expression in a constrained environmental condition.

Table 5.1: Description of different databases

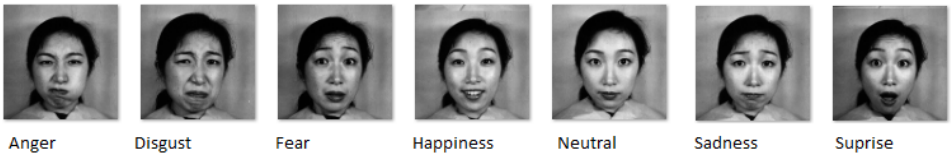
Datasets	Number of images	Image Resolution	Image colour	Number of identities	Images per person	Number of Expressions
JAFFE	213	256 x 256	Grayscale	10	20-23	7 Facial Expressions
AR database	4000	768 x 576	RGB	126	30	4 Expressions
Taiwanese Facial Expression Image Database	300	480 x 600	RGB	41	7	7 Facial Expressions

Differences exist in the acquisition conditions for each of the databases as well its size, which ranges from small to medium size. This ensured that the results of the experiments carried out in this project are consistent with increase in size of the input data as well across different image acquisition conditions. Each section describes in three major steps, the procedure involved: Preparation of Input Data, Model Selection via Cross Validation and the Transfer Learning experiment.

5.1 The Japanese Female Facial Expression (JAFFE) Database

The first dataset chosen for the experiments in this project is the JAFFE database [28]. It consists of 213 grayscale images of 10 identities with 7 different facial expressions

(a) 10 Identities



(b) 7 Facial Expressions

Figure 5.1: Samples from JAFFE Database

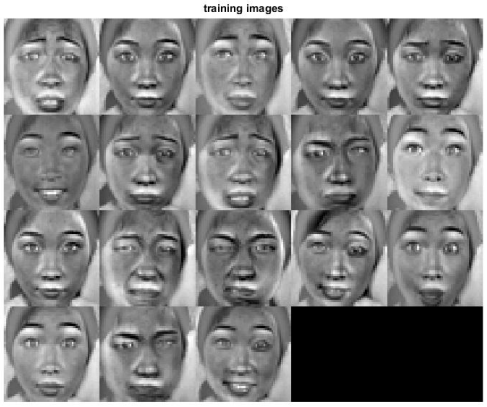
(30 angry, 29 disgust, 33 fear, 30 happy, 30 neutral, 31 sad and 30 surprise), as shown in Figure 5.1. The facial images were taken under strict controlled conditions of similar lighting and no occlusion such as hair or glasses. All images are frontal with a resolution of 256 X 256 pixels. According to [15], the accuracy achieved on the JAFFE database using a single CNN architecture for the task of face recognition was 98.6% and the task of facial expression recognition was 68.5%

5.1.1 Preparation of Input Data

The preparation of input data, involved downloading the publicly made available JAFFE database, pre-processing these images into a suitable format for the proposed CNN architectures in subsection 5.1.2. Since the images were obtained in a constrained environment, limited pre-processing steps were performed on the images. The images were first manually annotated for identity and expression, as shown in Figure 5.2a. These images were passed through a face detector to discard any background information, cropped and resized to 48 x 48 using predefined MATLAB functions. All images were standardised to ensure homogeneity by feature normalization, as shown in Figure 5.2b.

emotion	identities	pixels																	
1	1	114	85	59	27	8	4	3	3	2	2	2	3	2	2	4	3	2	2
1	1	3	3	2	4	3	2	3	2	2	3	4	2	2	3	3	3	3	2
1	1	2	3	3	3	3	3	2	2	3	3	2	3	2	2	2	3	4	2
2	1	3	3	3	2	2	3	2	2	2	2	2	1	2	2	3	2	3	3
2	1	3	2	2	2	3	3	2	2	2	2	2	2	2	3	2	2	2	3
2	1	2	2	2	1	2	3	2	2	2	2	2	2	3	3	2	3	2	2
3	1	115	107	83	51	16	2	1	1	1	1	1	1	1	1	1	1	1	1
3	1	2	2	1	2	3	2	2	2	1	2	1	1	2	1	2	2	1	2
3	1	46	9	1	1	1	2	1	2	2	2	1	1	1	2	2	2	1	1
3	1	105	110	116	99	66	36	22	9	1	1	1	1	2	1	1	1	1	1
4	1	134	136	141	140	139	146	149	150	149	134	108	95	69	34	21	37	41	40
4	1	143	146	148	147	144	138	123	99	80	53	20	2	3	5	4	5	5	16
4	1	143	146	145	143	146	150	156	162	157	150	128	101	85	94	86	75	65	77
4	1	126	127	126	125	126	130	132	132	133	130	128	125	118	108	99	101	81	90
5	1	152	139	113	89	69	54	26	3	4	4	3	4	3	4	4	4	9	7
5	1	155	149	138	114	100	91	58	25	3	4	4	4	4	9	10	10	22	18
5	1	146	146	126	106	92	77	54	41	10	3	3	3	4	4	4	4	7	21
6	1	137	96	52	15	3	3	3	4	3	3	3	3	3	4	4	3	3	4
6	1	64	16	3	4	4	4	3	3	3	3	4	3	2	4	4	3	2	4
6	1	154	135	94	46	12	4	3	2	3	4	3	4	3	3	3	4	4	4
7	1	139	144	143	143	120	86	51	6	3	4	4	4	4	4	3	4	4	4
7	1	154	153	151	145	154	163	164	156	153	141	110	82	59	48	53	57	52	53

(a) Manually annotated raw data



(b) Feature Normalized JAFFE images

Figure 5.2: Pre-processing steps

5.1.2 Model Selection

The candidate architectures chosen for this dataset were based on the architectures used in [14], [18] [15] described in subsection 2.2.3. In [15], the authors use the JAFFE database for both face recognition and facial expression recognition with two conv layers and two pooling layers and a fully connected Multi-Layered Perceptron (MLP) layer. Due to the small size of the JAFFE dataset the CNNs designed for this database, were similarly restricted to two alternate conv and pooling layers.

Table 5.2: Different architecture configurations for JAFFE FR and FER tasks

Architecture					**FR**	**FER**
Trials	*Layer 1*	*Layer 2*	*Layer 3*	*FC*	*Validation error*	*Validation error*
1	C:5x5x6 ReLU P: max,2	4x4x12 ReLU P: max,2	-	1x1x5 ReLU	33.01 ± 8%	27.37 ± 9%
2	C:5x5x6 ReLU P: max,2	4x4x12 ReLU P: max,2	-	1x1x10 ReLU	48.48 ± 7%	23.88 ± 9%
3	C:5x5x8 ReLU P: max,2	4x4x8 ReLU P: max,2	-	1x1x5 ReLU	21.34 ± 3%	30.58 ± 9%
4	C:5x5x8 ReLU P: max,2	4x4x8 ReLU P: max,2	-	1x1x10 ReLU	40.17 ± 4%	23.54 ± 9%
5	C:5x5x8 ReLU P: max,2	4x4x16 ReLU P: max,2	-	1x1x10 ReLU	32.45 ± 4%	22.88 ± 7%
6	C:5x5x12 ReLU P: max,2	4x4x24 ReLU P: max,2	-	1x1x10 ReLU	26.90 ± 2%	20.77 ± 3%
7	C:7x7x12 ReLU P: max,2	5x5x24 ReLU P:max,2	-	1x1x10 ReLU	31.87 ± 5%	25.35 ± 9%

The configurations in Table 5.2[1] were tested based on the following:

1) Number of feature maps per layer:

   a) Gradual increase in number of feature maps at each layer, which included variations such as {6-12},{8-16} and {12-24}[18].

   b) Same number of feature maps at all layers was tested for {6-6}, {8-8}. Having equal number of feature maps at each layer, reported a higher error

[1]The notations used in table: C → CONV layer with r x r (filter sizes) x number of feature maps; P → POOL layer with {type, size} of the pooling operator

rate when compared to pyramidal CNNs such as {12-24}. Only trial 3 and 4 were reported as it was lower in error rate.

2) Different sizes of FC layers:
Having the right sized fully connected layer, determines how robust and discriminative the fully connected feature vector will be. Since, most of the work done in CNNs involved datasets in the scale of million images with 4000 to 10,000 for FR tasks, the feature vector chosen for this scale ranged from 4000 to 19200 [14], [36]. By following a similar approach of using feature vector lengths that were discriminative, the chosen lengths for the FC layer were {1x1x5} and {1x1x10}.

3) Size of filters:
{5x5} and {7x7} were chosen as these filter sizes are the most common filter sizes used [18].

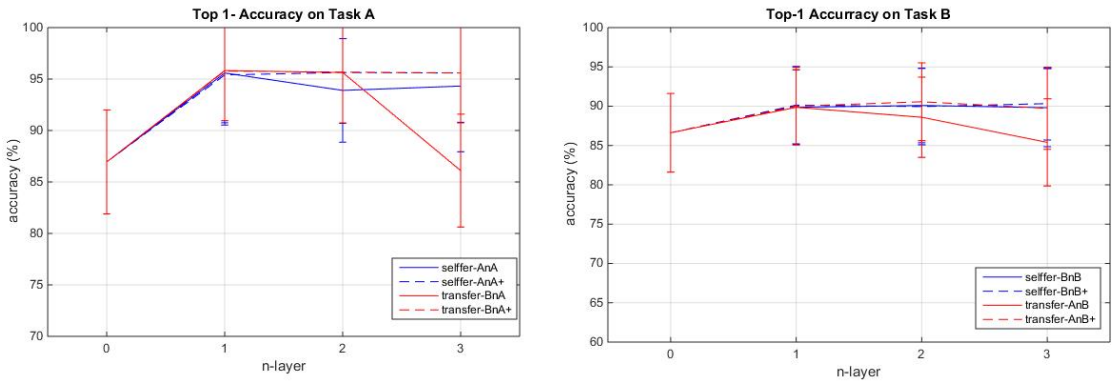
Based on these architecture trials, the top 2 models (highlighted in Table 5.2) were selected with low error along with low standard deviation for the transfer learning experiment.

5.1.3 Transfer Learning Experiment

If the hypothesis hold true regarding the transferability of facial features, then the expected outcome of this experiment, should indicate transferring the first layer filters favours knowledge transfer. The filters at this layer should consists of low level features such as edge-like or blob features which should be common across both FR and FER. However, at higher layers the networks should exhibit a drop in performance. Based on the analysis provided by the authors of [43], the reasons for drop in performance can be attributed to either the inability to learn the co-adaptation[2] among neurons or specificity of neurons.

Selffer networks should exhibit performance drop due to inability of the neurons to co-adapt while transfer networks should show a performance drop due to a combination of the two: specificity and fragile co-adaptation. However, with fine tuning this co-adaptation is re-learned. Moreover, fine-tuned transfer network should exhibit a boost in generalization performance, with the influence of base data set still persistent even after fine-tuning the filter weights.

[2]Complex interactions of neurons



(a) Top-1 test accuracy on task A

(b) Top-1 test accuracy on task B

Figure 5.3: Top-1 test accuracy for trial 5

The results of this experiment is reported in terms of the top-1 accuracy of the model on unseen data and is shown for both selected architectures in Figure 5.3 and Figure 5.4. In these figures, x-axis represents the layer at which filters are transferred, with 0 being the base case (baseA or baseB) or no transfer learning. The y-axis represents the test accuracy in percentage (%). Finally, 4 cases (or networks) are tested. The first two cases are the selffer networks with frozen and fine-tuned weights. Fine-tuned networks are indicated with a '+' sign and the next two cases are the transfer networks with frozen and fine-tuned weights (refer subsection 3.4.1).

Figure 5.3 and Figure 5.4demonstrate that a positive answer is achieved to the question of quantifying the degree to which a layer is general or specific. This is measured in terms of accuracy of the model to unseen data. For layer 1, the test accuracies are similar in all cases, which justifies that the representations learnt at this layer are general. At layer 2, only a slight drop in performance was observed for A2A and A2B and could be a result of neurons not being able to co-adapt with other neurons. Layer 3 being the fully connected layer definitely contained neurons more specific to the target task, hence the graphs shows a large drop in the accuracy.

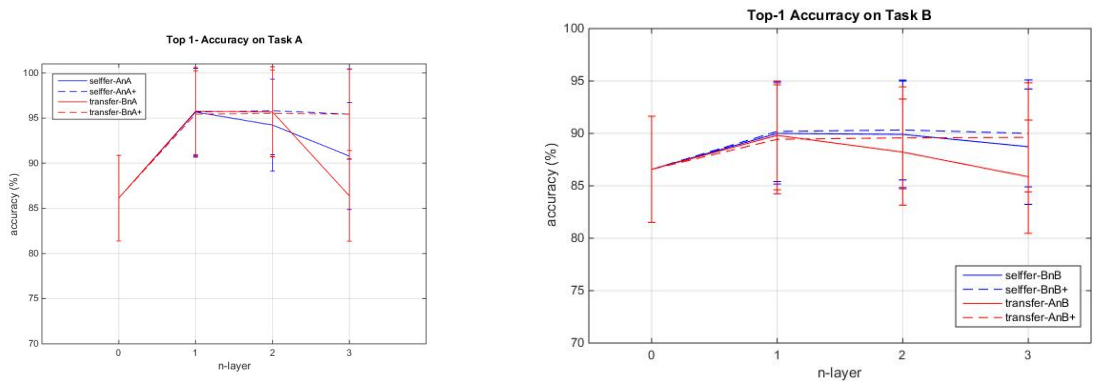
Given a target task B, transferring the filters learnt from the source task A AnB produces a lower test accuracy when compared transferring layers from source task B itself (BnB). These layers are transferred without fine tuning to the target task, hence the neurons at these layers are not allowed to adapt their weights to the target task.

Since the networks consist of only two layers, it is difficult to determine from the

accuracy plots whether the representation learnt at layer 2 was able to disentangle the different components and would benefit from visualization of the projected feature maps.

Due to the small size of the architecture, it is difficult to determine whether this transition from generality to specificity occurred at a single layer or was spread across layers. The inference that can be drawn from this experiment is that for small architectures the second layer marks the beginning of representations that tend to be more specific.

Using fine tuned filters, the test accuracies reported are higher than the base case as the filters now update their weights towards the target task and do not suffer from co-adaptation or neuron specificity. Here, the representations learnt by the target task build upon the representations learnt by the source task, which helps generalization to new data.



(a) Top-1 test accuracy on task A

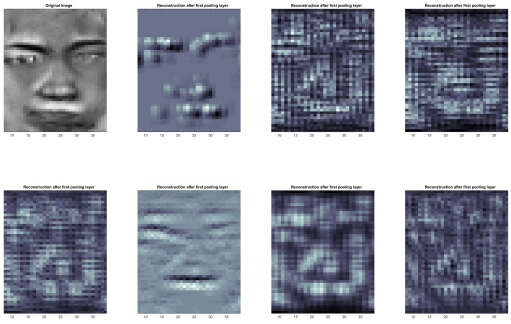
(b) Top-1 test accuracy on task B

Figure 5.4: Top-1 test accuracy for trial 6

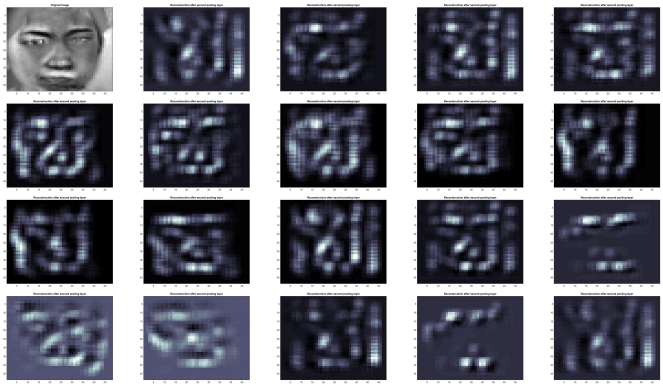
To supplement the results obtained in these transfer learning experiments, Figure 5.5 show the top-1 feature map activations based on subsection 3.4.2. For the JAFFE data set, a given identity was chosen with one facial expression and the image was passed through both the FR and FER networks. This allows us to see clearly differentiate between the representations learnt for both cases. It shows that at the first layer, the CNN network focusses on representations common to both FR and FER where the information about the face structure as whole is captured with some details within the face such as the eyes and mouth being emphasized.

At the second layer, the representations show some evidence of disentangling where

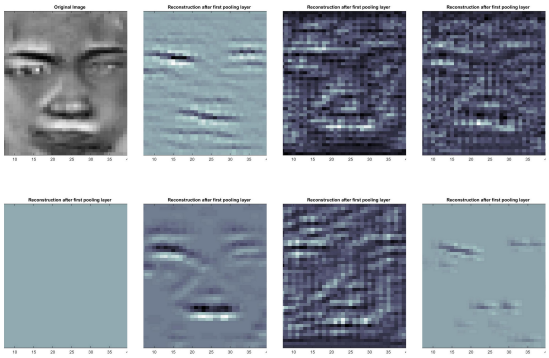
the feature activations in FR consisted of mainly face-like regions while the feature activations in FER, were focussed on specific regions within the face such as eyes, nose and mouth.



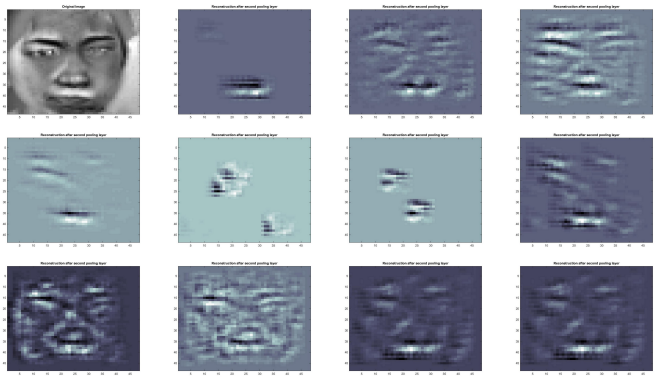
(a) Layer 1 Feature Maps for FR network



(b) Layer 2 Feature Maps for FR network



(c) Layer 1 Feature Maps for FER network



(d) Layer 2 Feature Maps for FER network

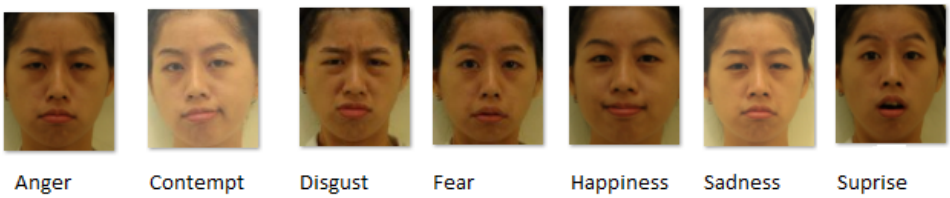
Figure 5.5: JAFFE database- Top-1 Feature Map Activations - identity 1; expression angry ; trial 6

5.2 Taiwanese Facial Expression Image Database (TFEID)

The second dataset chosen for the experiments in this project is TFEID [27]. It consisted of 336 RGB images of 41 identities (male and female) with 7 different facial expressions (anger, contempt, disgust, fear, happiness, sadness and surprise), as shown in Figure 5.6. These images were acquired in different conditions when compared to JAFFE dataset, where each image was obtained under two intensities (high and slight) and was captured by two CCD-cameras simultaneously with different viewing angles (0° and 45°). However, for experiments in this project only images with 0° and slight

intensity variation were considered.

(a) 41 Identities



(b) 7 Facial Expressions

Figure 5.6: Samples from Taiwanese Facial Expression Database

5.2.1 Preparation of Input Data

The preparation of the input for this database involved downloading this publicly accessible database. Images were first manually annotated for both FR and FER. Since the images were frontal with little background information, face detection was not performed. Conversion from RGB to grayscale was done along with cropping, resizing to 48 x 48 and normalizing the features. Additionally, the dataset was artificially aug-

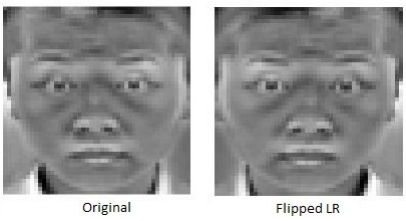


Figure 5.7: Feature Normalized and Augmented TFEID images

mented with left to right flipped versions of each image shown in Figure 5.7. This was

done to increase the training set to twice the original size of the database to allow a deeper network to be trained. Similar pre-processing steps were undertaken as given section 5.1.

5.2.2 Model Selection

Similar to the approach in subsection 5.1.2, the candidate architectures in this section were chosen based on [18]. The only difference is the introduction of deeper nets for this augmented dataset. The architecture configurations was chosen based on the following:

1) Different sizes of FC layers:
   The two selected FC feature vectors for this dataset were $\{1x1x40\}$ and $\{1x1x100\}$. It was observed (Trial 1, 3, 4 and 5) achieved a lower error when compared to a network with a larger FC layer (Trial 2).

2) Number of deep nets:
   Two different deep nets were chosen for this dataset: 3 layered deep net and 4 layered deep net. It was observed that the 3 layered network performed approximately twice as better when compared to the 4 layered network (Trial 4 error $\approx$ 2 x Trial 3 error).

3) Number of Max Pooling layers:
   Since a deeper network was introduced, having a max pooling layer after every conv layer resulted in a higher error. This is due to the loss of information of facial structures as a result of applying multiple pooling operations [14], (Trial 1 error $>$ Trial 3 error).

Based on these architecture trials, the model with low error along with low standard deviation (trial 3), was selected to perform the transfer learning experiments highlighted in the Table 5.3 and Table 5.4.

Table 5.3: Different architecture configurations for TFEID FR task

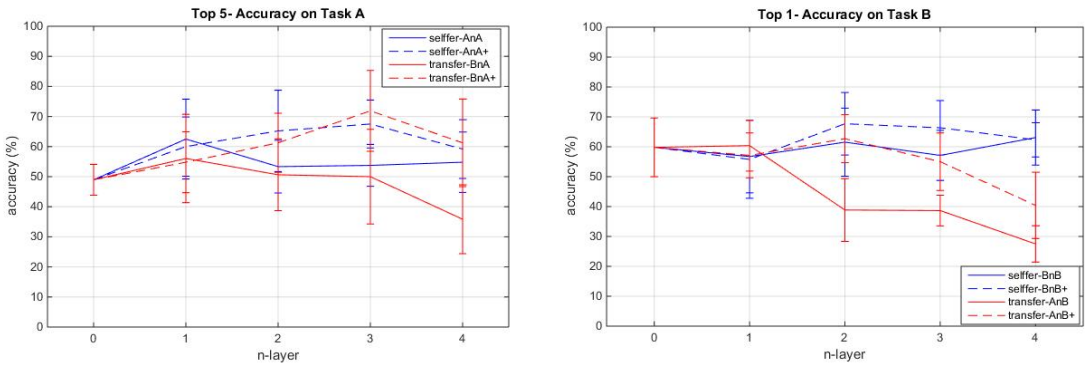
Architecture	**FR**					
Trials	*Layer 1*	*Layer 2*	*Layer 3*	*Layer 4*	*FC*	*Validation error*
1	C:5x5x12 ReLU P: max,2	4x4x24 ReLU P: max,2	3x3x32 ReLU P: max,2		1x1x40 ReLU	26.68 ± 7%
2	C:5x5x12 ReLU P: max,2	4x4x24 ReLU P: max,2	3x3x32 ReLU P: max,2		1x1x100 ReLU	38.44 ± 11%
3	C:5x5x12 ReLU P: max,2	4x4x24 ReLU P: max,2	3x3x32 ReLU		1x1x40 ReLU	9.52 ± 5%
4	C:5x5x12 ReLU P: max,2	4x4x12 ReLU P: max,2	3x3x24 ReLU	3x3x32 ReLU	1x1x40 ReLU	23.14 ± 10%
5	C:5x5x12 ReLU P: max,2	4x4x12 ReLU P: max,2	3x3x24 ReLU	3x3x32 ReLU	1x1x100 ReLU	22.10 ± 7%

Table 5.4: Different architecture configurations for TFEID FER task

Architecture	**FER**					
Trials	*Layer 1*	*Layer 2*	*Layer 3*	*Layer 4*	*FC*	*Validation error*
1	C:5x5x12 ReLU P: max,2	4x4x24 ReLU P: max,2	3x3x32 ReLU P: max,2		1x1x40 ReLU	22.96 ± 7%
3	C:5x5x12 ReLU P: max,2	4x4x24 ReLU P: max,2	3x3x32 ReLU		1x1x40 ReLU	14.53 ± 3%
5	C:5x5x12 ReLU P: max,2	4x4x12 ReLU P: max,2	3x3x24 ReLU	3x3x32 ReLU	1x1x100 ReLU	25.22 ± 9%

5.2.3 Transfer Learning Experiments

The results for this experiment are reported in terms of top-1 accuracy for task B and top-5 accuracy for task A. For FR, the model has to make correct predictions for 1 out of 41 classes and hence its top-1 accuracy is likely to be less accurate.



(a) Top-5 test accuracy on task A

(b) Top-1 test accuracy on task B

Figure 5.8: Top-1 test accuracy for trial 3

Figure 5.8, demonstrates that a positive answer is achieved to the question of quantifying the degree to which a layer is general or specific. Similar to the results achieved on JAFFE data set, for layer 1, the test accuracies are similar in all cases, which justifies that these layers are more general than specific. Layers 2 and 3 exhibit a drop in performance indicating that the representations learnt at these layers have began to disentangle the various information components. At layer 4, there is a drop in performance which indicates the neurons are more specific to the target task.

Figure 5.8, also demonstrates a positive answer is achieved to the question of determining where this transition occurs (first, middle or last layer). The performance of selffer networks at all layers was consistent with a slight improvement in accuracy at layers 3 and 4. Transfer networks exhibited a gradual drop in performance with the addition of layers 2 and 3 and at layer 4 the accuracy drops down by 20%- 30%. This would indicate that the representations are gradually disentangling the factors of variation at the mid layers of the hierarchy. Following the reasoning by [43], at these layers the effects of neuron specificity and fragile co-adaptation should pre-dominate.

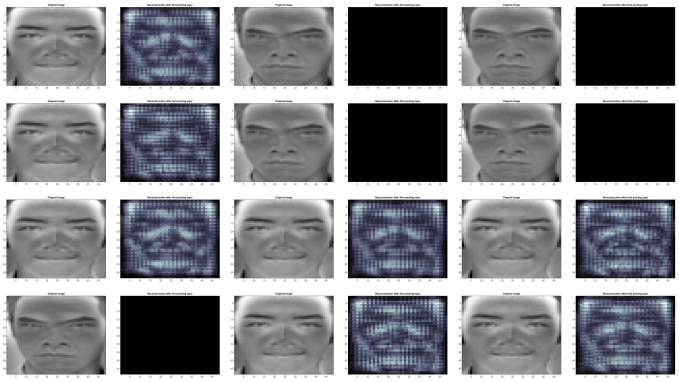
However, with fine tuning of the transferred filters for task A, the test accuracies are higher than the base case as the filters now update their weights towards the target task.

This trend was not observed for A4B+, as the network did not learn features to generalize to new test examples.

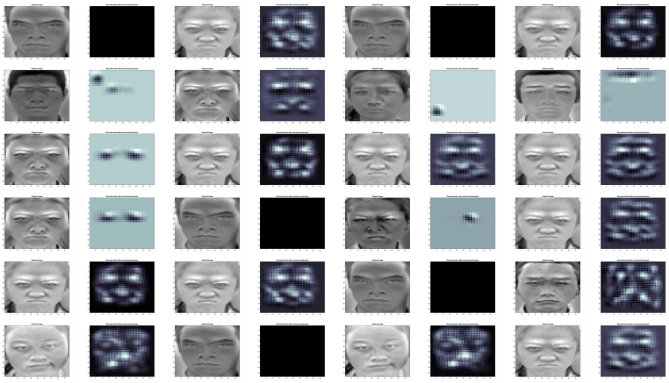
To validate the results obtained above, Figure 5.9 and Figure 5.10 show the top-1 feature map activations based on subsection 3.4.2. Both figures, show that common representations are learnt at the first layer the CNN networks which focusses on the face structure as whole, highlighting the face boundaries and structures within the face.

At the second layer, the feature map activations for FR are similar to that of FER. Both layer 2 feature maps focus on whole face-like regions with few of them emphasizing key features within the face such as the hair, eyes and mouth. This indicates that the representation learnt at this layer is still common for both FR and FER.

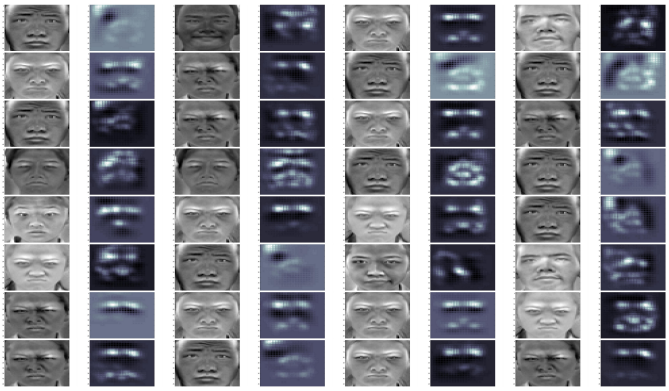
At the third layer, the disentangling of facial information components between FER feature and FR feature maps becomes clear. In FR, the feature maps still focus on key features within the face along with the face boundaries still intact. However, in FER majority of the feature map activations consisted of regions within the face consisting of eyes, nose and mouth which contribute to expression information.



(a) Layer 1 Feature Maps

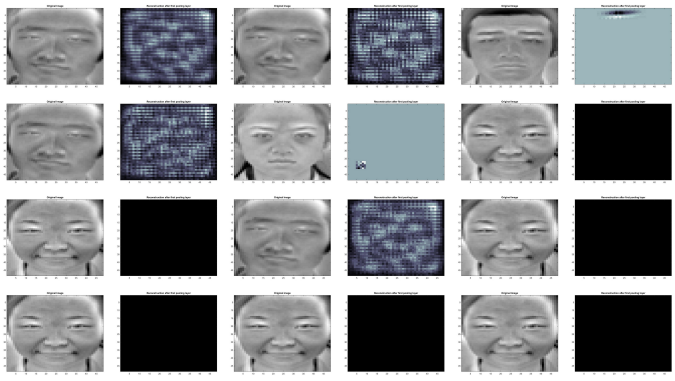


(b) Layer 2 Feature Maps

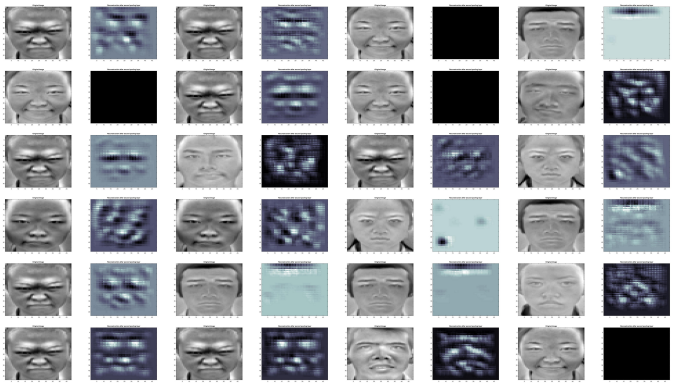


(c) Layer 3 Feature Maps

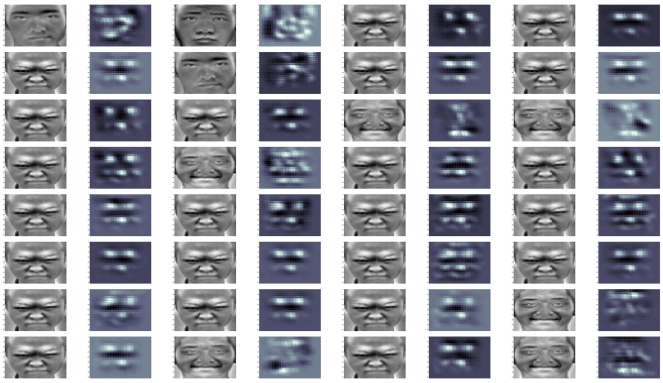
Figure 5.9: TFEID database- Top-1 Feature Map Activations for Task B /FR -trial 3



(a) Layer 1 Feature Maps



(b) Layer 2 Feature Maps



(c) Layer 3 Feature Maps

Figure 5.10: TFEID database- Top-1 Feature Map Activations for Task B /FER - trial 3

5.3 AR Database

The AR database [1] was the final dataset chosen for carrying out the experiments in this project. This database consist of 4000 images with 126 identities (70 men and

56 women). The reason for choosing this dataset was to introduce a deeper network, so as to consolidate the results that were obtained from the first two smaller datasets. All images are frontal with 4 different facial expressions, illumination conditions and occlusions (sun glasses and scarf) as shown in Figure 5.11. The images were acquired under strictly controlled conditions.

Unlike the previous two ”facial expression” databases, this database was designed for the task of face recognition. Hence, there were only 4 classes of expressions, with images having no restriction placed on clothing, hairstyle and others. The reason for choosing a third database was that, the results obtained from the two previous databases justifies the transfer learning hypothesis for smaller CNN architectures and to test the hypothesis on a larger data set with deeper network was required. Due to restricted access to other larger databases such as [22] and [17], it was decided proceed with AR database.

Figure 5.11: Sample images from AR database

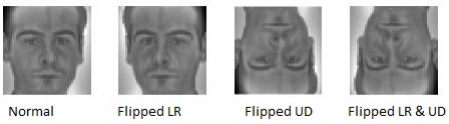


Figure 5.12: Feature Normalized and Augmented AR database images

5.3.1 Preparation of Input Data

The preparation of the input for this database involved obtaining permission to access this database from the owners. The images were available in ’.raw’ format and were

first converted into a '.jpg' format for processing them. A subset of the dataset was extracted consisting of only facial expression variation. The images were frontal with a white background. Hence, face detection was used to extract the face while removing the background information. Conversion from RGB to grayscale was performed along with cropping, resizing to 48 x 48 and normalizing the features. Additionally, the dataset was artificially augmented with flipped left to right (LR), flipped up down (UD) and flipped LR and UD of each image as shown in Figure 5.12. This was done to increase the training set to four times the original size of the database to allow a deeper network to be trained. Similar pre-processing steps were undertaken as given section 5.1.

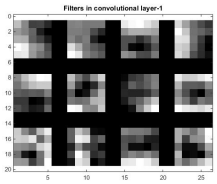
5.3.2 Model Selection

Since the size of the AR database was large enough to accommodate a 5 layered CNN, only three architecture configurations were considered owing to prohibitive time to train such deep networks. The first architecture consisted of three layers and was similar to the architecture proposed for TFEID dataset. Even though the error observed for the three layered network (trial 1) was lower for both FR and FER, the decision to carry out the transfer learning experiment on trial 2 was considered, as the main aim was to test the hypothesis on a deeper network.

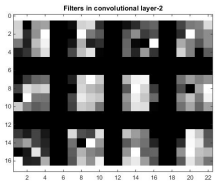
Table 5.5: Different architecture configurations for AR FR and FER tasks

Architecture						**FR**	**FER**
Trials	*Layer 1*	*Layer 2*	*Layer 3*	*Layer 4*	*FC*	*Val. error*	*Val. error*
1	C:5x5x12 ReLU P: max,2	4x4x24 ReLU P: max,2	3x3x32 ReLU	-	1x1x50 ReLU	7.71 ± 3%	24.06 ± 3%
2	C:5x5x12 ReLU P: max,2	4x4x12 ReLU P: max,2	3x3x24 ReLU	3x3x32 ReLU	1x1x50 ReLU	11.78 ± 2%	28.31 ± 4%
3	C:9x9x12 ReLU P: max,2	7x7x12 ReLU	7x7x24 ReLU	5x5x32 ReLU	1x1x50 ReLU	29.81 ± 6%	25.00 ± 3%

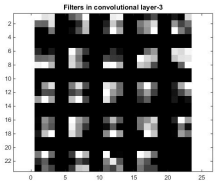
The convolutional filters for trial 2 and trial 3 CNN architectures are shown in Figure 5.13 and Figure 5.14 respectively. The first layer filters were observed to be edge-like and blob filters which indicates a nicely converged network. At higher layers these filters are not visually interpretable but are expected to be smooth and absent of noisy patterns [11].



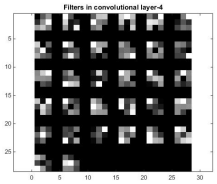
(a) Layer 1 Filters



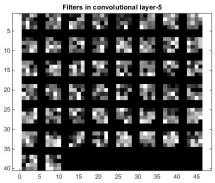
(b) Layer 2 Filters



(c) Layer 3 Filters

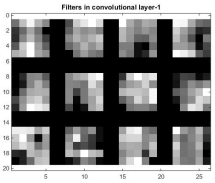


(d) Layer 4 Filters

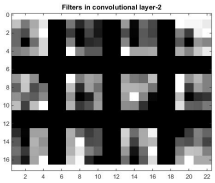


(e) Layer 5 Filters

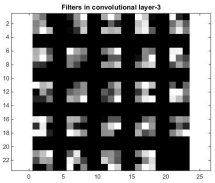
Figure 5.13: AR database- Convolutional Filters learnt for FR - trial 2



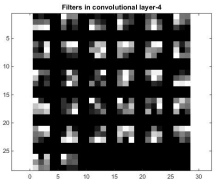
(a) Layer 1 Filters



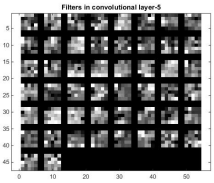
(b) Layer 2 Filters



(c) Layer 3 Filters

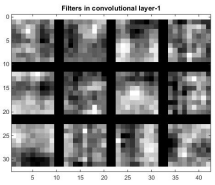


(d) Layer 4 Filters

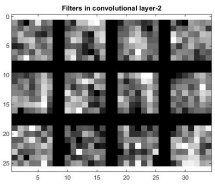


(e) Layer 5 Filters

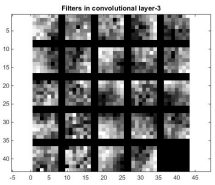
Figure 5.14: AR database- Convolutional Filters learnt for FER - trial 2



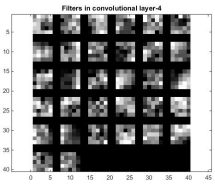
(a) Layer 1 Filters



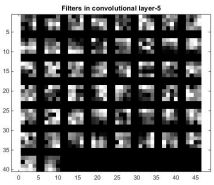
(b) Layer 2 Filters



(c) Layer 3 Filters

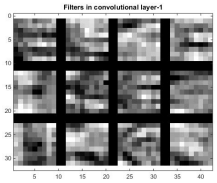


(d) Layer 4 Filters

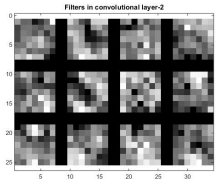


(e) Layer 5 Filters

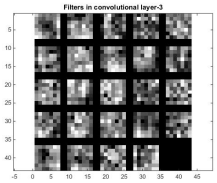
Figure 5.15: AR database- Convolutional Filters learnt for FR - trial 3



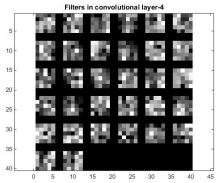
(a) Layer 1 Filters



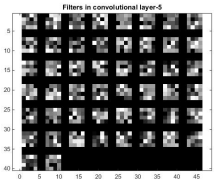
(b) Layer 2 Filters



(c) Layer 3 Filters



(d) Layer 4 Filters

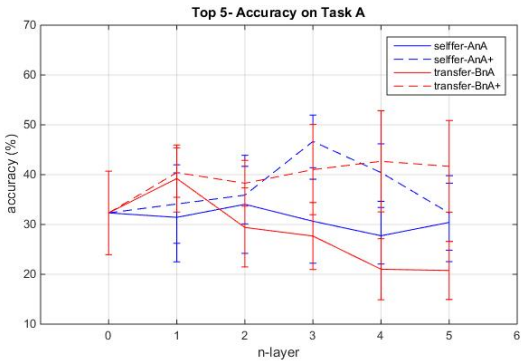


(e) Layer 5 Filters

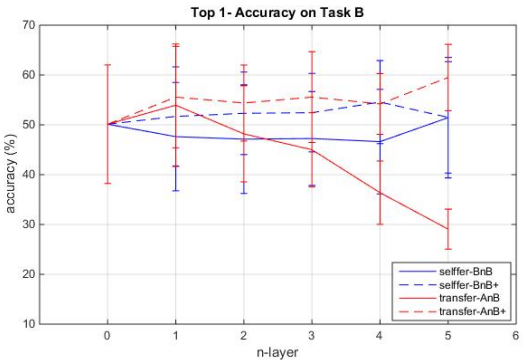
Figure 5.16: AR database- Convolutional Filters learnt for FER - trial 3

5.3.3 Transfer Learning Experiment

The results for this experiment is reported in terms of top-1 accuracy for task B and top-5 accuracy for task A.



(a) Top-5 test accuracy on task A



(b) Top-1 test accuracy on task B

Figure 5.17: Top-1 test accuracy for trial 2

Figure 5.17, demonstrates that a positive answer is achieved to the question of

quantifying the degree to which a layer is general or specific. Similar to results on previous data sets, layer 1 representations were more general as all networks had similar accuracies. Layer 2 to 5 of the selffer networks in both cases had similar performance to the base case where at certain layers (layer 4) drop in performance could indicate neurons have not re-learned the original co-adaptation. Transfer networks exhibited a gradual drop in performance where at layer 4 and 5 the drop in performance is approximately 10-20%. This clearly shows evidence that the representations learnt are separating the various factors of variations. The general trend of selffer networks performing better than transfer network is evident on all three data sets as the only factor hindering the performance gain in selffer networks in the inability of the transferred neurons to co-adapt with other neurons.

Figure 5.17, also demonstrate a positive answer is achieved to the question of determining where this transition occurs (first, middle or last layer). Layers 2, 3 and 4 depict a gradual performance drop indicating that the transition from general to specific is spread across the mid layers of the network.

With fine-tuning for the selffer networks BnB+ and AnA+, performance is still comparable to the base cases. This is the result of the absence in loss of co-adaptations between neurons. Similar to results obtained by [43], fine-tuning the transfer networks BnA+ and AnB+ resulted in boost in generalization capability over networks trained directly on target data set itself. The reason given by the author is that, even after fine-tuning of the weights, the effect of having seen the base data set still lingers. This in turn boosts generalization capability for the transfer of any layer n.

To validate the results obtained in these transfer learning experiments, Figure 5.18 and Figure 5.19 show the top-1 feature map activations. Both figures, show that at the first layer the CNN networks focusses on the face structure as whole, highlighting face boundaries and structures within the face. These representations contain shared factors of variation for both FR and FER.

At the second layer, the feature map activations for FR are similar to that of FER. Both layer 2 feature maps focus on whole face-like regions with few of them emphasizing key features within the face such as hair, eyes and mouth. This indicates that the representation learnt at this layer is still common for both FR and FER.

At the third layer, feature maps for both FER and FR focus on a group of facial features within the face such as the eyes, nose and mouth. Unlike the previous datasets, where the facial images did not posses extra information like the presence of spectacles or varying hairstyles. Certain feature maps at this layer, activated for features like hairstyles, facial hair-moustaches and spectacles.

At the fourth layer the feature maps in FR, still focus on key features within the face along with the face boundaries still intact. However, in FER majority of the feature map activations consisted of single facial feature within the face like either eyes, nose or mouth which contribute to expression information.

5.4 Summary of Results

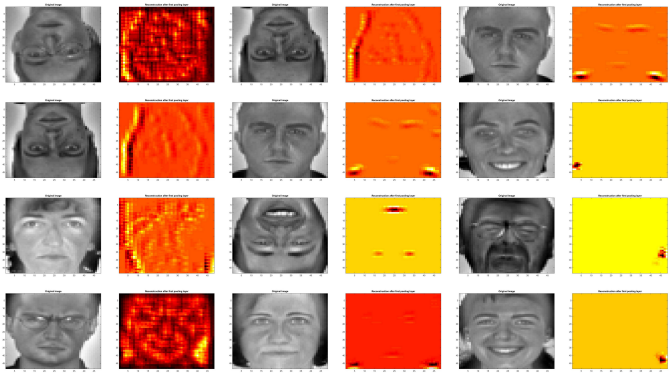
The results drawn for individual datasets are summarized as follows:

- The experiments on JAFFE dataset, test the capability of transfer learning on a very small dataset. Here, the limitations of CNN architectures for small data sets are revealed. Even though some evidence exists on generality and specificity of such small networks, it can not be verified with only performance graphs and requires visual support to verify the disentangling of factors.

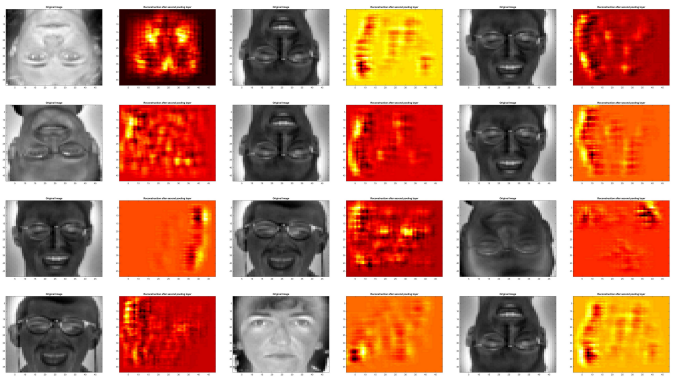
- The experiment on TFEID dataset, gave a better insight into the transfer learning capabilities of a deeper network. Here, performance graphs as well as visual representations of the feature maps on both FR and FER, provided good evidence of disentangling of facial informations components. Layers lower in the hierarchy share common representations across tasks while layers higher in the hierarchy have representations that are specific to a particular data set or task. The mid layers show representations that contains both general and specific characteristics.

- The AR database allowed the use of a 5 layered deep network. Similar to the TFEID, it showed that the layer 1 is capable of learning filters that resemble low level features like edges or blobs common to all data sets or tasks. It also showed that the transition from generality to specificity was spread across layers 2 to 4, while layer 5 learnt representations that were specific to the target dataset

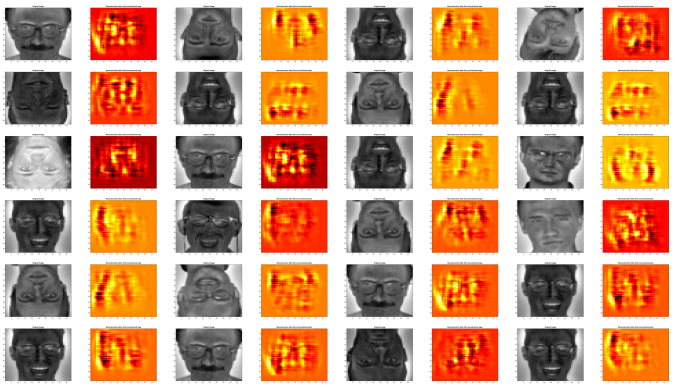
Finally, the common inference that can be drawn from all three experiments is that, in the hierarchical representations learnt by deep CNNs, the transition from generality to specificity occurs when moving from lower layers to higher layers. Layer 1, tends to learn low level features like edges or blobs, that are common across both FER and FR whereas the last layer tends to be highly specific to either FR or FER. This implies that as the layers become more specific, the disentangling of facial information components is observed.



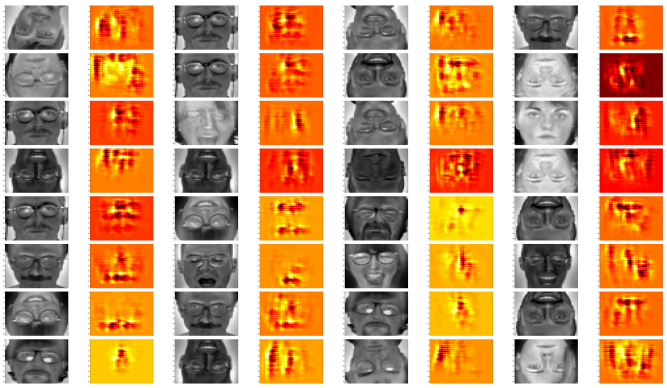
(a) Layer 1 Feature Maps



(b) Layer 2 Feature Maps

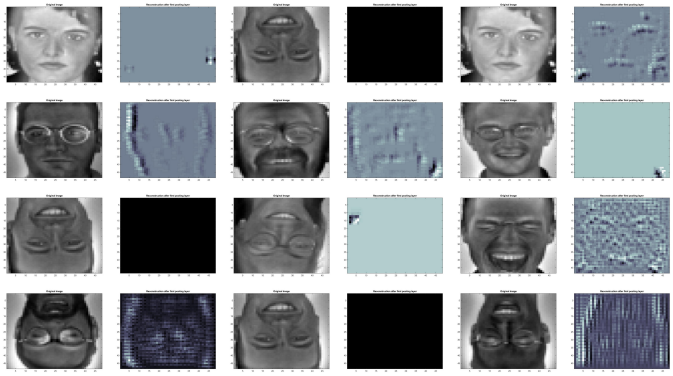


(c) Layer 3 Feature Maps

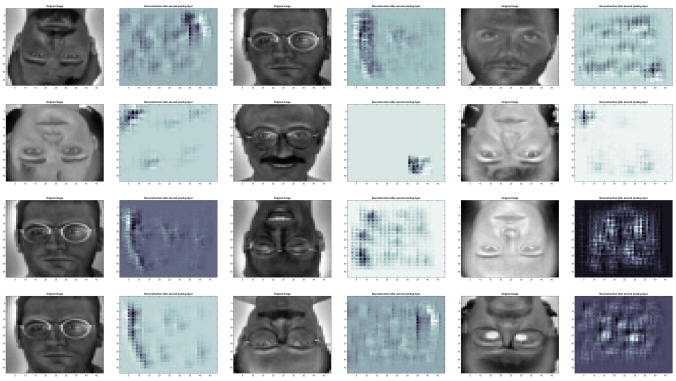


(d) Layer 4 Feature Maps

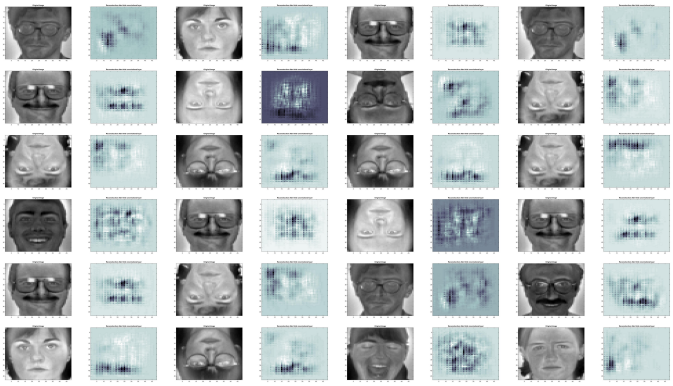
Figure 5.18: AR database- Top-1 Feature Map Activations for Task B /FER - Architecture 2



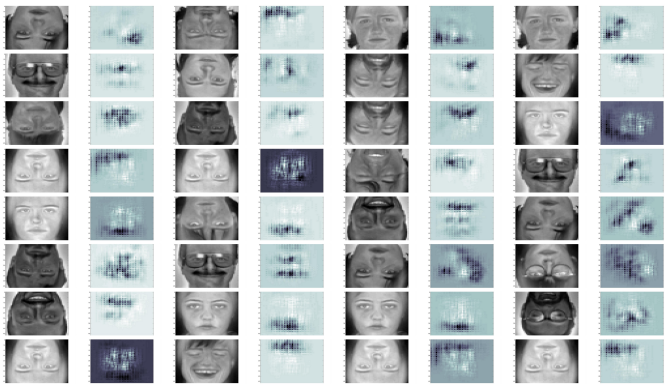
(a) Layer 1 Feature Maps



(b) Layer 2 Feature Maps



(c) Layer 3 Feature Maps



(d) Layer 4 Feature Maps

Figure 5.19: AR database- Top-1 Feature Map Activations for Task B /FR - Architecture 2

Chapter 6

Discussion

This chapter provides a discussion of the project. A personal reflection is given, discussing the strengths and weaknesses of the project and the lessons learned through the course of the project. Several suggestions for directions of future work are given along with a brief justification.

6.1 Personal Reflection

The topic of representation learning and the associated models to learn representations from data proved quite interesting to me. Through this project I could familiarise with some of the work carried out in deep learning community and the on-line tutorials by researchers like Yoshua Bengio, Geoffrey Hinton, Hugo Larochelle and Andrew Ng that was pleasing and instrumental in understanding concepts for my project.

One of the main difficulties that I experienced was the training of the models presented in this project. Since there is no fixed procedure for obtaining the optimum architecture, my initial approach was based on trial and error for a set of architecture configurations. I began by varying a single parameter while varying other. In one scenario, the number of layers were fixed while the filter sizes were varied. Testing all possible combinations was time consuming and hence I confined the model selection to architectures used in the literature.

While selecting hyper parameters values for the optimizer, I used the values found in the literature. Although this proved to work well for my current architecture selection, it would be inappropriate to repeat this procedure for a different architecture and

data set. I would definitely invest more time to obtain the right parameter settings by model selection through the strategy mentioned in [4].

The training time for the architectures is another aspect of the project that I would like to improve upon. Through my experimental work I observed that transfer networks took a greater number of epochs while in comparison selffer networks took lesser number of epochs to converge (refer section B.2). The method I used to deal with this variable timing, was to train all networks to the longest training time, and store the results of every epoch in a separate file. Finally, I used the method of early stopping to determine the epoch at which the networks begin to over fit. The only drawback of this approach is that it utilize a lot of system storage. Additionally, with the current CPU system (see section 4.2), training and testing all the networks in the transfer learning experiment took approximately 4-5 days. This training time could be prohibitive when training a more deeper network on a larger data set (refer section B.1)

6.2 Future Work

Deep learning algorithms have a large number of hyper-parameters and one approach to further enhance the work carried out in this project is to fine-tune these parameter settings through model selection using the approach mentioned in [4]. This can include random sampling from a prior distribution for a given hyper-parameter configuration.

Another possible direction to improve the work carried out in this project is to enhance the architecture designs for FR. Architectures that were tested for TFEID and AR database reported a lower top-1 accuracy when compared to the top-5 accuracy. Many improved architecture designs are available in literature for FR such as multiple convNets used together on different patches of the input [36] or the use of local connectivity at deeper layers [14]. However, how such architectures would benefit FER will have to be further explored.

Finally, to offset the extensive training time for the networks, this project would benefit from the use of in-built GPU capability of the MatConvNet toolbox. Apart from the reduced training time, the above two suggestions for improvements would indeed benefit from increased processing capacities.

Chapter 7

Conclusions

The main objective of this dissertation was to test the transfer learning hypothesis for two facial image processing tasks; face recognition and facial expression recognition. In doing so, show that there is an implicit transfer learning process in deep neural networks to disentangle different facial information components.

It covered all the necessary background material required for the project with focus on the broader research area of Representation Learning while trickling down to subdomain research areas such as deep CNNs and transfer learning. The deliverables of this project were fulfilled by documenting the entire research methodology, software implementation, experimental results and analysis along with a software program developed using MatConvNet toolbox, to test the transfer learning aspects.

The experimental work was carried out three benchmark data sets: JAFFE, TFEID and AR database and results obtained from all three show an improvement in classification accuracy while transferring lower layers in a deep CNN. Representations learnt at these layers contain information suited to both tasks FR and FER. However, a drop in classification accuracy at higher layers suggest that the representations have become more specific to each task. Visualization was another verification method used which visually represent the activations at each layer in terms of the input. Together with both of the aforementioned methods, a strong justification is provided in favour of the transfer learning hypothesis in the experimental section of this dissertation.

Finally, three suggestions for future work were provided based on the experimental work and lessons learnt through the course of the project.

Bibliography

[1] R. Benavente A.M. Martinez. The ar face database, cvc technical report 24. http://www2.ece.ohio-state.edu/~aleix/ARdatabase.html, 1998.

[2] Peter N. Belhumeur, João P Hespanha, and David Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997.

[3] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. *Unsupervised and Transfer Learning Challenges in Machine Learning, Volume 7*, page 19, 2012.

[4] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[5] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[6] Xudong Cao, David Wipf, Fang Wen, Genquan Duan, and Jian Sun. A practical transfer learning algorithm for face verification. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 3208–3215. IEEE, 2013.

[7] Dong Chen, Xudong Cao, Liwei Wang, Fang Wen, and Jian Sun. Bayesian face revisited: A joint formulation. In *Computer Vision–ECCV 2012*, pages 566–579. Springer, 2012.

[8] Ke Chen and Ahmad Salman. Extracting speaker-specific information with a regularized siamese deep network. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference*

on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain., pages 298–306, 2011.

[9] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 539–546. IEEE, 2005.

[10] JosephLin Chu and Adam Krzyżak. Analysis of feature maps selection in supervised learning using convolutional neural networks. In Marina Sokolova and Peter van Beek, editors, *Advances in Artificial Intelligence*, volume 8436 of *Lecture Notes in Computer Science*, pages 59–70. Springer International Publishing, 2014.

[11] Cs231n.github.io. Cs231n convolutional neural networks for visual recognition. http://cs231n.github.io/convolutional-networks/, 2015.

[12] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2013.

[13] Guillaume Desjardins, Aaron Courville, and Yoshua Bengio. Disentangling factors of variation via generative entangling. *arXiv preprint arXiv:1210.5474*, 2012.

[14] Haoqiang Fan, Zhimin Cao, Yuning Jiang, Qi Yin, and Chinchilla Doudou. Learning deep face representation. *CoRR*, abs/1403.2802, 2014.

[15] Beat Fasel. Robust face analysis using convolutional neural networks. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 40–43. IEEE, 2002.

[16] Ian J. Goodfellow, Dumitru Erhan, Pierre Luc Carrier, Aaron Courville, Mehdi Mirza, Ben Hamner, Will Cukierski, Yichuan Tang, David Thaler, Dong-Hyun Lee, Yingbo Zhou, Chetan Ramaiah, Fangxiang Feng, Ruifan Li, Xiaojie Wang, Dimitris Athanasakis, John Shawe-Taylor, Maxim Milakov, John Park, Radu Ionescu, Marius Popescu, Cristian Grozea, James Bergstra, Jingjing Xie, Lukasz Romaszko, Bing Xu, Zhang Chuang, and Yoshua Bengio. Challenges in representation learning: A report on three machine learning contests. *Neural Networks*, 64(0):59 – 63, 2015. Special Issue on "Deep Learning of Representations".

[17] Ralph Gross, Iain Matthews, Jeffrey Cohn, Takeo Kanade, and Simon Baker. Multi-pie. *Image and Vision Computing*, 28(5):807–813, 2010.

[18] G. Hu, Y. Yang, D. Yi, J. Kittler, W. Christmas, S. Z. Li, and T. Hospedales. When Face Recognition Meets with Deep Learning: an Evaluation of Convolutional Neural Networks for Face Recognition. *ArXiv e-prints*, April 2015.

[19] Gary B Huang, Honglak Lee, and Erik Learned-Miller. Learning hierarchical representations for face verification with convolutional deep belief networks. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2518–2525. IEEE, 2012.

[20] Info.usherbrooke.ca. Hugo larochelle. http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html, 2015.

[21] Samira Ebrahimi Kahou, Christopher Pal, Xavier Bouthillier, Pierre Froumenty, Çaglar Gülçehre, Roland Memisevic, Pascal Vincent, Aaron Courville, Yoshua Bengio, Raul Chandias Ferrari, et al. Combining modality specific deep neural networks for emotion recognition in video. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 543–550. ACM, 2013.

[22] Takeo Kanade, Jeffrey F Cohn, and Yingli Tian. Comprehensive database for facial expression analysis. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 46–53. IEEE, 2000.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[24] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[25] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 253–256. IEEE, 2010.

[26] S. Z Li and Anil K Jain. *Handbook of face recognition*. Springer, 2005.

[27] Yu-Shiuan Yen Li-Fen Chen. Taiwanese facial expression image database (tfeid). http://bml.ym.edu.tw/tfeid/, 2007.

[28] Jiro Gyoba Miyuki Kamachi Michael Lyons. The japanese female facial expression (jaffe) database. http://www.kasrl.org/jaffe.html., 1998.

[29] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[30] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. Self-taught learning: Transfer learning from unlabeled data. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 759–766, New York, NY, USA, 2007. ACM.

[31] Salah Rifai, Yoshua Bengio, Aaron Courville, Pascal Vincent, and Mehdi Mirza. Disentangling factors of variation for facial expression recognition. In *Computer Vision–ECCV 2012*, pages 808–822. Springer, 2012.

[32] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[33] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. In *Advances in Neural Information Processing Systems*, pages 1988–1996, 2014.

[34] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3476–3483. IEEE, 2013.

[35] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Hybrid deep learning for face verification. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1489–1496. IEEE, 2013.

[36] Yi Sun, Xiaogang Wang, and Xiaoou 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.

[37] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. June 2014.

[38] Yichuan Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

[39] Ufldl.stanford.edu. Unsupervised feature learning and deep learning tutorial. http://ufldl.stanford.edu/tutorial/supervised/ExerciseConvolutionAndPooling/, 2015.

[40] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for matlab. *CoRR*, abs/1412.4564, 2014.

[41] Andrea Vedaldi and Karel Lenc. Matconvnet convolutional neural networks for matlab. http://www.vlfeat.org/matconvnet/matconvnet-manual.pdf, 2015.

[42] Paul Viola and MichaelJ. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[43] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[44] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

[45] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2018–2025. IEEE, 2011.

[46] Zhenyao Zhu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning identity-preserving face space. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 113–120. IEEE, 2013.

Appendix A

MATLAB Code

A.1 Visualization of feature maps

A.1.1 Estimation of Switches

```
function [switches,conv_layer_size_x] =
MaxPoolLoc( conv_layer,pool_layer,stride_pool)
%MAXPOOLLOC finds the switch variable for max pooling operation
%   The switch variable is a pool_layer^2 x 2 x depth matrix of (x,y) coordinates
%    where pool_layer^2 is the number of pooled values in a single pooled feature map
% ---------------------------------------------------------------------------
% Estimating the Pooled Switch Variables
% ----------------------------------------------------------------------------
pool_layer_size_x=size(pool_layer,1);
pool_layer_size_y=size(pool_layer,2);
pool_layer_size_depth=size(pool_layer,3);
conv_layer_size_x=size(conv_layer,2);
switches=[];
for depth=1:pool_layer_size_depth
    count=1;
    xind=1;
    for x=1:pool_layer_size_x
        yind=1;
        for y=1:pool_layer_size_y
% Extract the 2 x 2 grid from conv layer
a=conv_layer(xind:xind+stride_pool-1,yind:yind+stride_pool-1,depth);
            [r c]=find(a==pool_layer(x,y,depth));
```

```
          % compare with pool layer value and find the position in conv layer
            if (size(r,1)==1)
            switches(count,1,depth)=xind+r-1;% assign x coordinate
            switches(count,2,depth)=yind+c-1;% assing y coordinate
            else
            switches(count,1,depth)=xind+r(1)-1; % assign x to 1
            switches(count,2,depth)=yind+c(1)-1; % assign y to 1
            end

            yind=yind+stride_pool;
            count=count+1;
        end
        xind=xind+stride_pool;
    end
end
end
```

A.1.2 Unpooled Maps

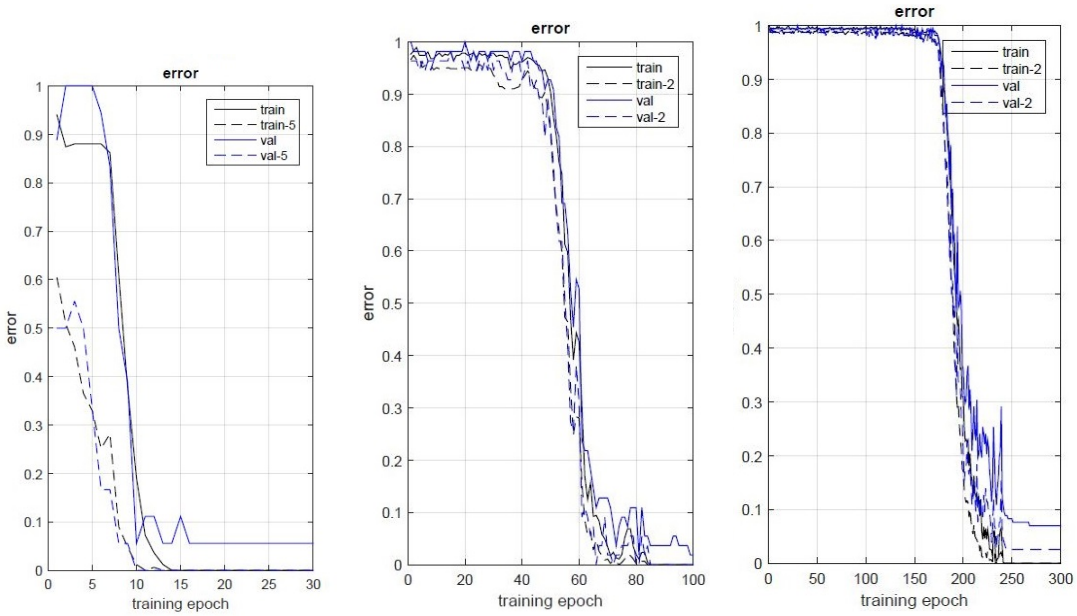
```
function [ Unpooled_map ] = create_UnpooledMaps(switches,pool_layer,conv_layer_size_x)
%CREATE_UNPOOLEDMAPS creates a unpooled map using the switch
variables
% 1. Creates a Unpooled map of original pool feature map dimensions
% 2. Extract the pooled values into a matrix
%    of dimension equal to switch variable matrix
%    (pool_layer^2 x 1 x depth)
% 3. Plugs in the pooled values in the required location using switch variables
% ---------------------------------------------------------------------------
% Formation of Unpooled Maps
% ---------------------------------------------------------------------------
pool_layer_size_depth=size(pool_layer,3);
pool_layer_size_y=size(pool_layer,1);

Unpooled_map=zeros(conv_layer_size_x,conv_layer_size_x,pool_layer_size_depth);
% Transpose the pool layer maps
for depth=1:pool_layer_size_depth
pool_layer(:,:,depth)=pool_layer(:,:,depth)';
end
% Reshape Pool Maps to dim (col^2 X 1 X depth)
pool_layer=reshape(pool_layer,pool_layer_size_y^2,1,pool_layer_size_depth);
```

```
% Create Unpooled Maps
for depth=1:pool_layer_size_depth
    for m=1:pool_layer_size_y^2
        m;       Unpooled_map(switches(m,1,depth),switches(m,2,depth),depth)
=pool_layer(m,1,depth);
    end
end
end
```

Appendix B

Training time of CNNs

B.1 Training time versus architecture size



(a) 3 layered network- JAFFE database (b) 4 layered network- TFEID (c) 5 layered network- AR database

Figure B.1: Increase in convergence time (epochs) for deeper networks

B.2 Training time for transfer and selffer networks

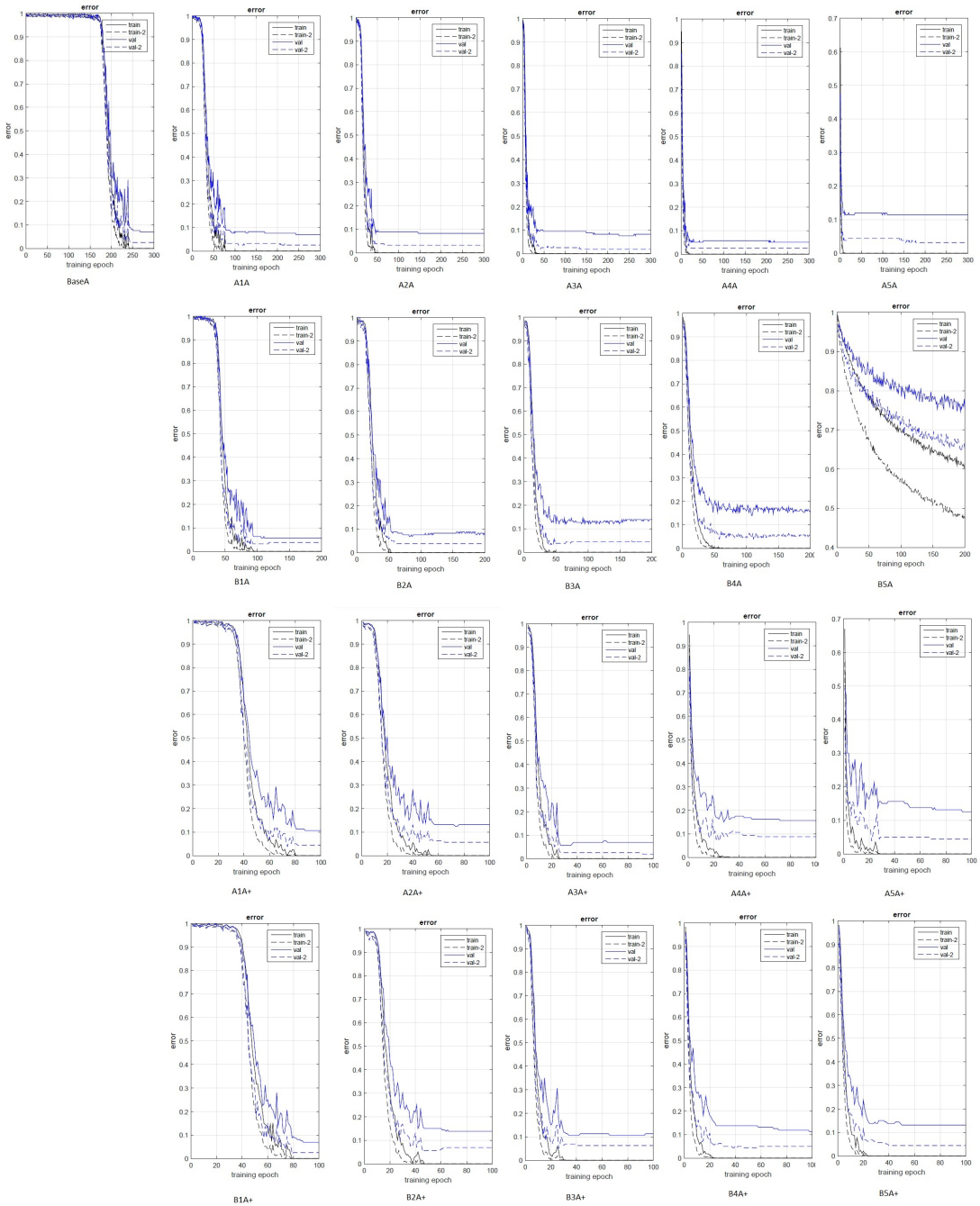


Figure B.2: Training time for base, transfer and selffer networks for task A - AR database