Comparative Analysis between Inception-v3 and Other Learning Systems using Facial Expressions Detection

By

MD. RAYED BIN WAHED
16141024
AKM NIVRITO
12201020

Department of Computer Science & Engineering
BRAC UNIVERSITY

Supervised by: AMITABHA CHAKRABARTY
Co-supervised by: MOIN MOSTAKIM

Thesis report submitted to the BRAC University in accordance with the requirements of the degree of BACHELOR IN COMPUTER SCIENCE AND ENGINEERING in the Dept. of Engineering & Computer Science.

Submitted On:
18TH AUGUST, 2016

ABSTRACT

In the last five years or so, Machine Learning has taken the world by storm. From predictive web browsing, to E-mail classification, to autonomous cars; machine learning is at the heart of every intelligent applications that's in service today. Image Classification and Facial Expression Recognition is another field that has benefited immensely from the emergence of this technology. In particular, an branch of Machine Learning called Deep Learning, has shown tremendous results in this regard even outperforming more conventional methods such as Image Processing. Inspired by neurons in the human brain, Artificial Neural Networks, allow us to map complex functions by stacking layers upon layers of these networks. Our goal in this paper, is to analyze Inception v-3, the best performing high resolution image classifier based on Convolutional Neural Network out there today, with other methods including one of our own to see how it performs on low resolution images detect Facial Expressions.

Dedication and acknowledgements

The first persons that come to our minds are our honorable Supervisor and Co-Supervisor Dr. Amitabha Chakrabarty and Moin Mostakim respectively for their continuous support and dedication toward our work. They really helped us through this journey of gathering knowledge which we immensely enjoyed.

Secondly, this work would not have been possible if not for the tremendous love and support we received from our family and friends, it certainly would have been a tough ride if there supports were not present.

In particular, I, Md. Rayed Bin Wahed, would like to dedicate this work to my friend and admiration, Salma Maliha Mou and a special heartfelt thanks to Kevin Markham of dataschool.io for his video lectures on preparing me for this task.

And I, A.K.M Nivrito, would like to dedicate this to the people who have been supporting me through all the moments of doubts and sleepless nights- Rupa Barua, Shauvik Shadman, Talha Siddique, Sadia Intesar, Shakil Bin Karim and Mahrin Tasfe. And a special thanks to Ian Goodfellow, Yoshua Bengio and Aaron Courville for letting their knowledge be available for us to grab.

And finally, we would like to acknowledge that as a team it had been a joy to learn and grow together. We worked together, we kept each other going and certainly we pulled each other up with covering each others shortcomings. We are certainly stronger and better working together.

AUTHOR'S DECLARATION

We, hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researcher are mentioned by reference. This thesis, neither in whole or in part, has been previously submitted for any degree.

SIGNATURE OF THE AUTHORS:

....
AKM NIVRITO
12201020

....
MD. RAYED BIN WAHED
16141024

SIGNATURE OF THE SUPERVISOR:

....
DR. AMITABHA CHAKRABARTY
ASSISTANT PROFESSOR
DEPT. OF COMPUTER SCIENCE, BRAC UNIVERSITY

SIGNATURE OF THE CO-SUPERVISOR:

....
MOIN MOSTAKIM
LECTURER
DEPT. OF COMPUTER SCIENCE, BRAC UNIVERSITY

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this era of computation, we can observe major improvements of computational resources. But still such resources bring limitations of their own. Overcoming such limitation is the major improvement scopes that various research and studies are focusing on. Retrieving information from images is one of the applications of computer science that requires a lot computational resources to perform well. It is also a domain that has multiple practical application scopes, major being behavioral analysis, natural language processing, and biometrics analysis. In the motivation to overcome the computational resource uses limitation, a convolutional neural network architecture named Inception has been introduced which not only focuses on reducing computational costs without hampering accuracy but also has a clever design so that expanding or modifying the architecture does not take toll on computational resources [27][28]. One of the important field where can image classifiers models can be applied is facial expression recognition. In this paper we try to use Inception to detect facial expression on low resolution greyscale images and compare it with other models on the same criterion.

1.1 Motivation

Machine Learning has generated a great impact on how information are retrieved from visual data. Recent development on that area has shown some increasingly accurate and optimised performance. Specially after the involvement of deep learning, the metrics has shown impressive improvements. The application of such image classifiers is vast. And with the help of recent improvements in processing powers, the domain of application is expanding fast. The recently introduced architecture, Inception shows promise on improved accuracy as well as optimized use of computational resources. In this paper we are putting up a comparative analysis of image classifier models on how well they can perform on recognizing facial expression on low resolution greyscale images, especially how well the Inception architecture stands among the others.

1.2 Objectives

The main objectives of our thesis is to find out if Inception holds on to the promises it makes - better accuracy on the expense of lower computational resources. So we look into generate facial expression recognition on low resolution images and compare and contrast it with other learning models. SO that we can reach a conclusion if the Inception architecture can really be deployed on the application domains where less computational power can be used to achieve high level of image processing.

Report Outline

- **Chapter 1** is the formal introduction to our report.
- **Chapter 2** is the literature review. There we also introduce the learning models involved in the comparison that has been implemented in the base paper. And give an introduction to Inception-v3.
- **Chapter 3** consists of our work. We also discuss about the result and the analysis.
- **Chapter 4** is the conclusion to our report.

Chapter 2

Literature Review

Facial Recognition using deep learning, or more specifically, *Convolutional Neural Network* is not a very recent topic. One of the earlier research on facial recognition using CNN can be cited to the works of Lawrence et al.[20], where they have developed a convolutional model based on the preceding successes of CNN on recognizing handwritten characters[22]. They also discussed about the other face recognition systems like the scheme by Turk et al.[29] popularly known as *Eigenfaces*, and hierarchical neural nets. However, in their approach the used *SOM* (Self Organizing Map) while improving it using Luttrel's method[24] to cut down computational cost. Also *eigenvector expansion via Principle Component Analysis* has been done[20].

Since then improvements in hardware have boosted machine learning systems. The introduction of the Convolutional Neural Network or CNN approach also has been a major improvement on the image classifier models. One of the standard CNN models on application was the approach by LeCun et al.[22] in the LeNet-5 system. Such standard CNN models are structured using multiple layers of neurons where in a layer the neurons form groups based on the feature they work on and these groups feed forward to the next layer, and at the end it connects with a fully connected layer [1]. The layers are in many cases accompanied with max pooling [27]. It had established itself as the most efficient model giving the best result on image classification problems. However, Krizhevsky et al.[19] with their CNN approach where they have used dropout and data augmentation to solve the problem of overfitting, showed a huge amount of accuracy and efficiency in results in ILSVRC 2012 surpassing anything that ever been used before [19]. Krizhevsky et al. developed a *deep convolutional network*[1] using *RelU*[2] as their activation function.

In 2014, Donahue et al.[12] released an open source implementation of *Deep Convolutional*

[1]See 2.3.2.1.
[2]See equation 2.8.

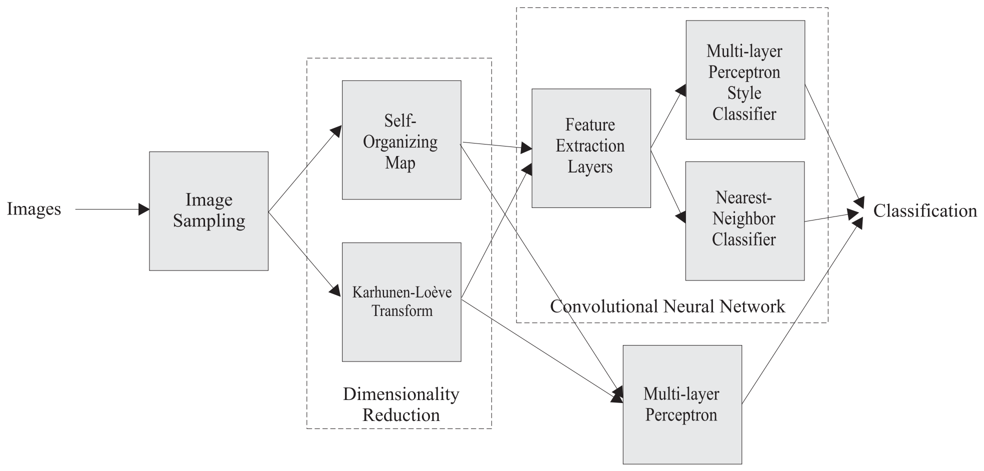


Figure 2.1: Visual representation of the system developed by Lawrence et al.[20] It is noticeable that the final layer of CNN in their approach adopts to different classifiers.

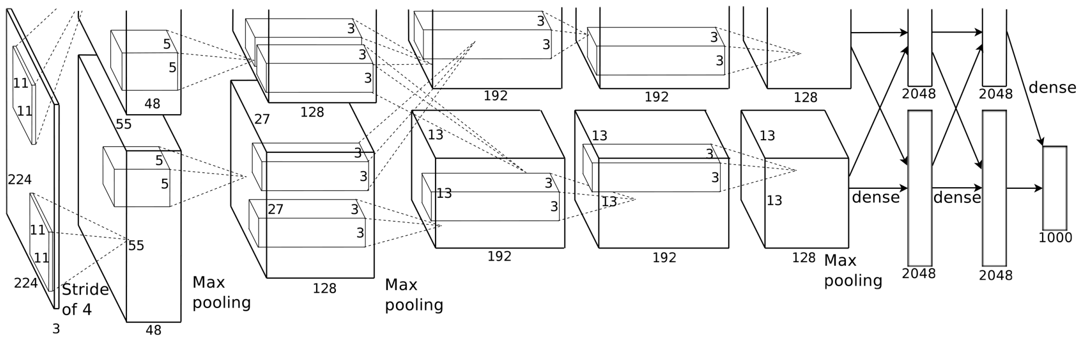


Figure 2.2: Full architecture of CNN by Kirzhevsky et al.[19]

Activation Features named *DeCaf*. Their work was directed to train a CNN using the network Krizhevsky et al.[19] proposed, and then show that such activation features outperforms the conventional visual representations on object recognition tasks, like Caltech-101[13].

Integrating DeCaf, Ouellet[25] had implemented an application of CNN through the use of emotion detection for gaming. However the real time service implementation required use of a huge processing power on a AMD Phenom II X4 955 GPU.

2.1 Inception: Overview

The approach we are concerning in our thesis, *Inception*, was developed based on GoogLeNet architecture seen in ILSVRC 2014[3] [27]. It also took inspiration from the approach based on primate visual cortex dictated by Serre et al. [26] which can handle multiple scales. One of the important criteria of Inception architecture is their adaption of "Network in Network" approach by Lin et al [23] which increased the representational power of the neural networks. This had additionally saved them for computational bottlenecks by dimension reduction to 1×1 convolutions.

The purpose of Inception architecture was to reduce computational resource usage in highly accurate image classification using deep learning[27]. They had focused on finding an optimized position between the traditional way of increasing performance, which is to increase size and depth, and using sparsity in the layers based on the theoretical grounds given by Arora et al. [7]. Both the approach in their own position can cost a huge amount of computational resources. For such a deep learning system like Inception which uses fully learned filters in their 22 layer architecture, this was the main goal to achieve. They focused on the approach of Arora et al.[7] to generate a correlation statistic analysis to generate groups of higher correlation to feed forward to the next layer. And they took the idea of multiscale analysis of visual information in their 1×1 , 3×3 and 5×5 convolution layers. All of these layers then go through dimension reduction to end up in 1×1 convolutions [27].

The Inception architecture used in ILSVRC 2014 had the following structure as denoted by Szegedy et al.[27]:

- An average pooling layer with 5×5 filter size and stride 3.
- A 1×1 layer with 128 filters for dimension reduction and rectified linear activation.
- A fully connected layer with 1024 units and rectified linear activation.
- A dropout layer with 70% ratio of dropped outputs.

We discuss further about Inception Architecture and CNN in our upcoming sections.

2.2 Previous Comparative Analysis

A comparative analysis of prior image processing techniques using machine learning has been done by Chudasama et al.[11] which has denoted a comparison of efficiency of machine learning models on facial expression detection. In our work, it has been taken as the base of comparison, where we look forward to add the analysis of Inception architecture on same objective on the same dataset. In their paper, Chudasama et al.[11] two shallow learning models and one deep

[3]ImageNet Large Scale Visual Recognition Competition.

learning models. One of the shallow learning models was a Support Vector Machine on a single layer learning approach by Knerr et al.[17] The other one was a shallow neural network on Theano[9]. The deep learning model that was implemented was based on the approach by LeCun et al.[21] and Krizhevsky et al.[19] After the analysis they have found out that the convolution neural network approach of Krizhevsky et al. [19] outperformed others by a big measure.

2.3 Learning Models

As previously mentioned, our work takes the work done by Chudasama et al.[11] where they have implemented two ***shallow learning models*** and one ***deep learning model*** for the purpose of comparison. In this section we will discuss about the implemented models and how they work for further clarification. As we added our work result with the comparison of these models, it is crucial to understand how these model works to infer a better conclusion about our comparative analysis.

2.3.1 Shallow Learning Models

2.3.1.1 Support Vector Machine

Support Vector Machine or ***SVM*** is a supervised learning model which generates hyperplanes to generate class identities. SVM is driven by a linear function $\mathbf{w}^\top \mathbf{x} + b$. Here $\mathbf{w}$ is the vector of *weights*, $\mathbf{x}$ is the vector of *inputs* and b is the *bias*. Now as many machine learning algorithms can be written in terms if the dot products between example, the previous function can be rewritten as [8]:

$$\mathbf{w}^\top \mathbf{x} + b = b + \sum_{i=1}^{m} \alpha_i \mathbf{x}^\top \mathbf{x}^{(i)} \tag{2.1}$$

Here, $\mathbf{x}^{(i)}$ is a training example and α is a vector of coefficients. Now if we change the $\mathbf{x}$ with the function $\phi(\mathbf{x})$ and the dot products with a **kernel**, $k(\mathbf{x}, \mathbf{x}^{(i)}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}^{(i)})$, we get our prediction function:

$$f(\mathbf{x}) = b + \sum_{i=1} \alpha_i k(\mathbf{x}, \mathbf{x}^{(i)}) \tag{2.2}$$

This kernel can be changed and one of the most common kernel is the ***Gaussian kernel*** or ***Radial Basis Function***:

$$k(\mathbf{u}, \mathbf{v}) = \mathcal{N}(\mathbf{u} - \mathbf{v}; 0, \sigma^2 \mathbf{I}) \tag{2.3}$$

Where $\mathcal{N}(\mathbf{u}-\mathbf{v};0,\sigma^2\mathbf{I})$ is a ***Gaussian Distribution***[4]. It acts as a template matching, for example, a given training example $\mathbf{x}$ becomes a template for class y, and when a test point x' approaches $\mathbf{x}$ according to Euclidean distance, the Gaussian kernel gives a large positive response.

Our base paper has comparison of SVM implemented based on "one on one" approach by Knerr et al.[17]

2.3.1.2 Artificial Neural Network

Artificial Neural Network or **ANN** are learning models inspired by the biological system of learning and classification. If we simplify the definition, an ANN works on multiple layers connected through *neurons*:

- **Input Layer**: Where the data is given as inputs.
- **Hidden Layer**: The input then passed through the *neurons* based on ***activation functions***.
- **Output Layer**: The outputs are generated here.

[4]See equation A.1 in Appendix A.

Visually we can represent a simple ANN as following figure:

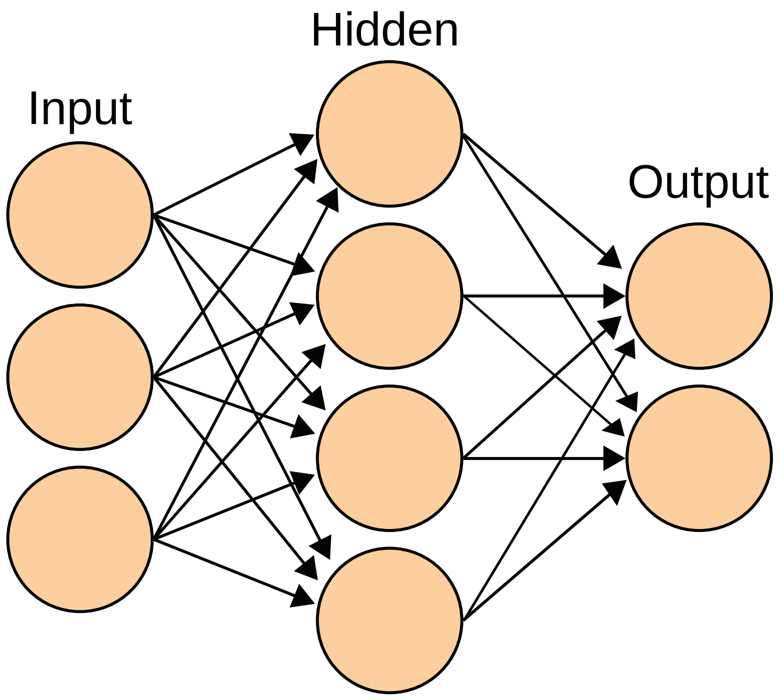


Figure 2.3: A simple visualization showing different layers of ANN connecting through neurons[6]. The arrow directions denote that it is a feedforward network.

While discussing about ANN, it is very important to discuss about two functions: **basis function** and **activation function**. **Basis function**s are the functions that processes the inputs that go into the hidden layers. So for a shallow ANN the way outputs are generated can be described in following function [10]:

$$y(\mathbf{x},\mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \Phi_j(\mathbf{x})\right) \tag{2.4}$$

Here, this $f(\cdot)$ is replaced by the nonlinear **activation function**, and $\Phi_j(\mathbf{x})$ is the **basis function** which gets adjusted with the coefficient w_j supplied through the vector (w).

The **activation function** is the function that activates the *neurons* to push the inputs forward to the next layer. Generally, this **activation function** can be any nonlinear sigmoidal function. A general activation function is the ***logistic sigmoid function***:

$$\sigma(a) = \frac{1}{1+e^{-a}} \tag{2.5}$$

For multi-class problems, ***softmax activation*** is used in place of equation 2.5:

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.6}$$

Other notable activation functions are:

The ***hyperbolic tan function*** :

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2.7}$$

The ***Rectified Linear Units*** or ***RelU*** :

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.8}$$

The ***Exponential Linear Units*** of ***ELU*** :

$$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.9}$$

Our concerned base reference has implemented an ANN with 3723 neurons, based on ***hyperbolic tan function*** for activation.

2.3.2 Deep Learning

Deep Learning Networks are *feedforward neural networks* with multiple layers. It is also known as **Multilayer Perceptrons** or ***MLPs***. General feedforward deep learning networks do not include feedback. There are deep networks with feedback systems called **recurrent neural network** or **RNN**.

2.3.2.1 Convolutional Neural Network

Convolutional Neural Networks or ***CNN*** are a special kind of deep artificial neural networks using ***Convolution*** operation in least one of the layers. CNN is useful to process data with grid like topology. CNN has been outstanding in image classification in challenges like CIFAR and ImageNet[5].

To understand ***convolution*** operation, lets take the following example. Imagine a car is moving and we are tracking it with a sensor. $x(t)$ gives the position of the car at t time. x and t, both are real valued, but our sensor receives a noisy value. SO we decide to smooth it out by averaging the several measurement we get. But the most recent values will have most importance, so we will have to do a weighted average by a weight function $w(a)$ where a is the age. Now if we do that to the value for every t we will get a smoothing function s, which is[8]:

[5]See A.2 Appendix A.

$$s(t) = \int x(a)w(t-a)da \tag{2.10}$$

This smoothing function is called ***convolution*** operation. It is generally denoted with *asterisk* ($*$). So we can write equation 2.10 as:

$$s(t) = (x * w)(t) \tag{2.11}$$

In CNN, x is referred as **input**, w is referred as **kernel**, and the output is referred as **feature map**.

CNN, can generate **Sparse Connectivity** reducing the cost of matrix computation. For example, even if an image has a lot of pixels, the **kernel** can occupy a very small number of pixels. Also, it can have **tied input**, where any weight for an input can be tied with other weight on the network. So less parameters are generated.

The **convolution layer**, where ***convolution*** operation happens, is followed by a **detection** or **activation layer** which similar to the regular ANN acts as the activator to the neuron connections. The next layer is called the **pooling layer**. ***Pooling*** is actually replaces the output with a summary statistic from the near outputs[8]. One of the most popular method is ***max pooling***, primarily suggested by Zhou et al.[30]. In laymen terms, ***max pooling*** given a matrix or a portion of a matrix, picks out the maximum value.

Occasionally, a **dropout layer** follows. ***Dropout*** is done to avoid over-fitting. At training each node is either dropped out based on the probability $1-p$ or kept based on the probability p, where the value p is set during creating the CNN structure. Thus a reduced network is created, optimizing the computational load and preventing the chances of overfitting.

The base paper has comparison based on an implementation of CNN loosely based on the works of Krizhevsky et al.[19].

2.4 Inception-v3

Our main motivation is to focus on ***Inception-v3*** architecture, and do a comparative analysis on accuracy. We have already introduced it in section 2.1. In this section we would look further on the Inception-v3 architecture. We will start with discussing the background and motivation, followed by it's structure. And at the end we would introduce the layer that we are basically retraining with data.

2.4.1 Background and Motivation

As discussed before in chapter 2, Inception arrived based on the GoogleNet in ILSVRC 2014 [27]. We also have talked about how CNN can use less amount of computation in section 2.3.2.1. Now

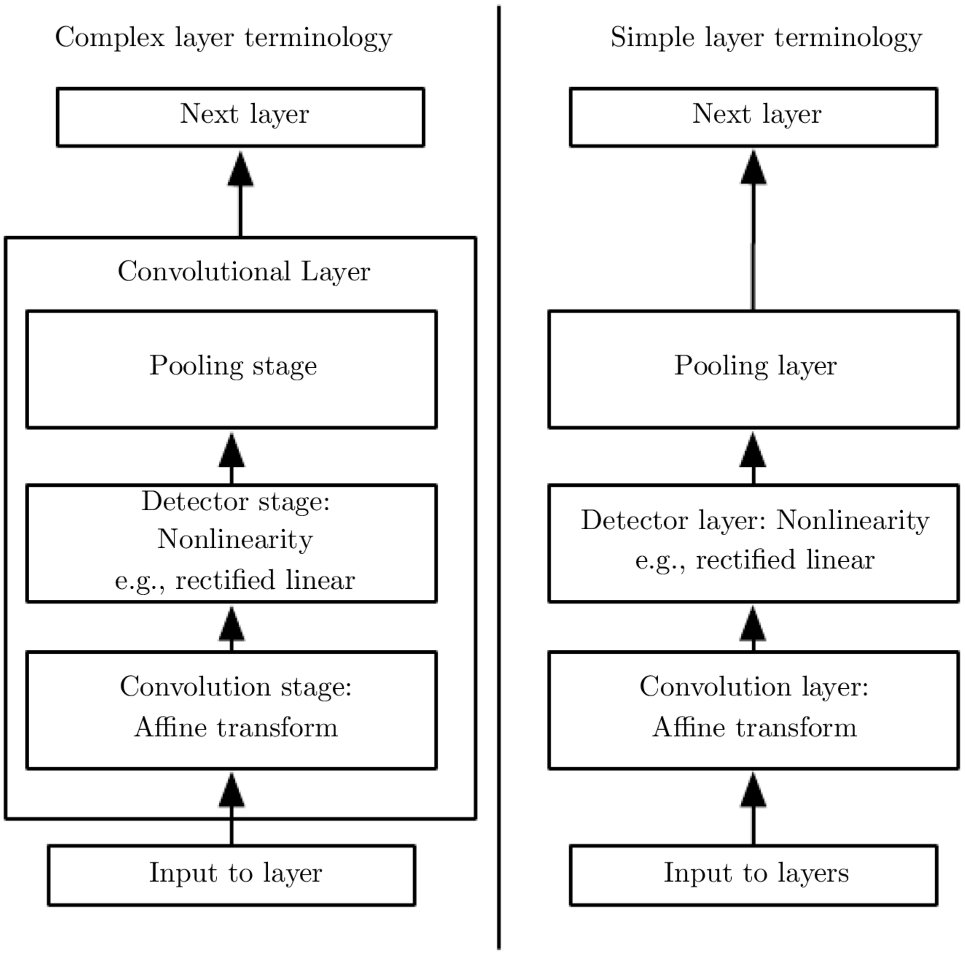
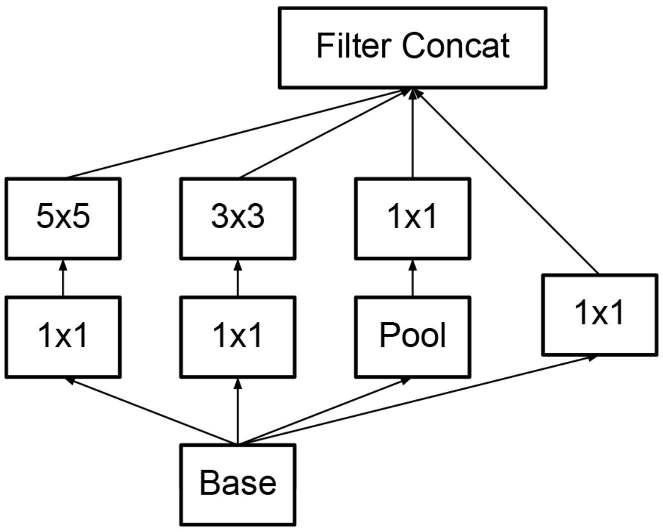

Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

Figure 2.4: Layer structure of simple ***Convolutional Neural Network***. Image taken from [8]

Inception had started as a hypothetical study of approximating a sparse structure in the network. All the motivation was to acquire a good number accuracy with out strainning too much on the computational budget. One of the key thing was generating sparse structure increasing both the depth and width of the network is less costly with computation process. But computation of non-uniform sparse data structure is not much efficient in the current context. So the Inception architecture hypothesized clustering the similar sparse nodes into a dense structure, to overcome the dilemma.

2.4.2 Structure

(a)



(b)

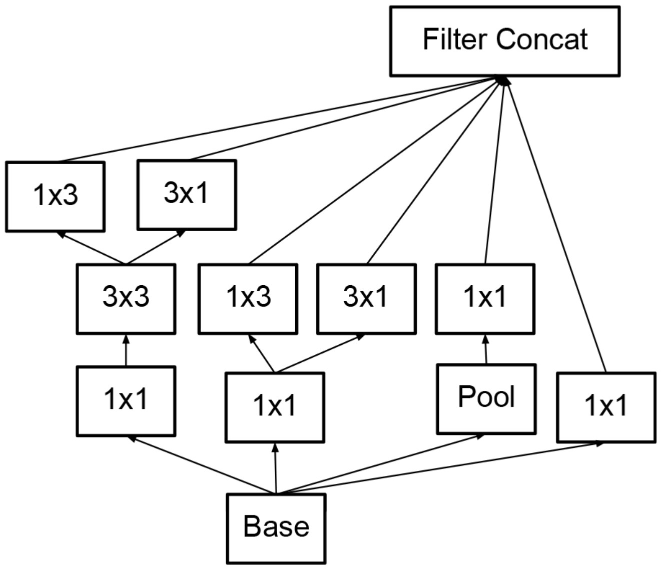


FIGURE 2.5. (a) Original Inception module structure. Figure reproduced from [27]. (b)Inception module with expanded filter banks. This is used on coarsest grids (8×8) to increase high level representation. Figure reproduced from [28].

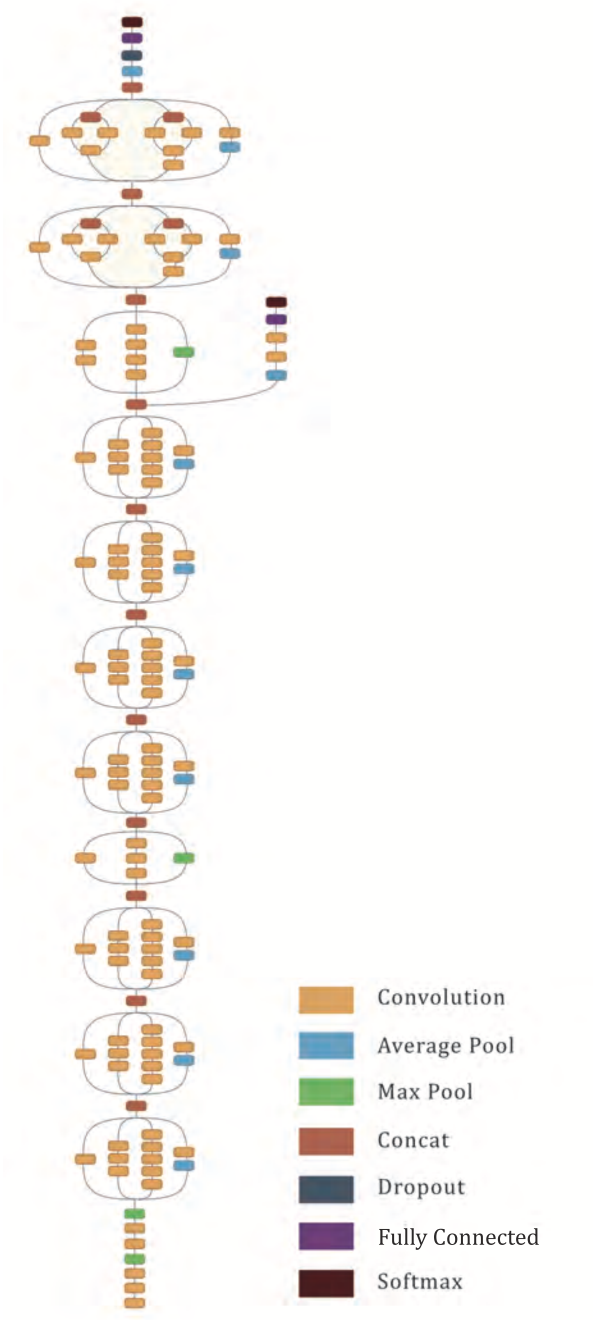


Figure 2.6: Inception-v3 complete architecture. Figure recreated from [4].

2.4.3 Softmax Layer

In our approach of Inception architecture, the last layer of Inception has been retrained using Softmax Regression where we generate probabilities based on our evidences extracted from the images [3] [27]. The evidences are calculated based on a sum of weights detected by the intensity of pixels, with added bias.

$$\text{evidence}_i = \sum_j W_{i,\,j} x_j + b_i \tag{2.12}$$

Here is the equation for the evidence for a class i given an input x. Here W_i is the weights, b_i is the bias, and j is the index for summing over pixels in input [3].

Then our probabilities are generated by passing the evidence through ***softmax function***.

$$y = \text{softmax}(\text{evidence}) \tag{2.13}$$

The ***softmax function*** or ***normalized exponential***, if given a n-dimensional vector generates a same dimensional vectors of values ranging from 0 to 1. The function has been already be noted in equation 2.6.

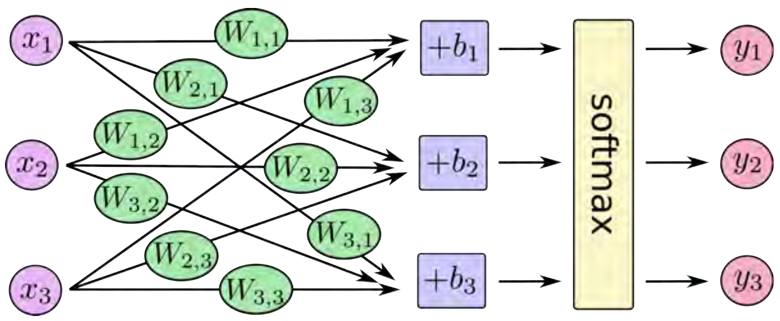


Figure 2.7: Visual Representation of Softmax Regression. Image from [3].

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

Figure 2.8: The visualization from 2.7 can be transformed into above representation showing **Probability Matrix** in Softmax. Image from [3].

Chapter 3

Work and Analysis

3.1 Dataset

3.1.1 Collection

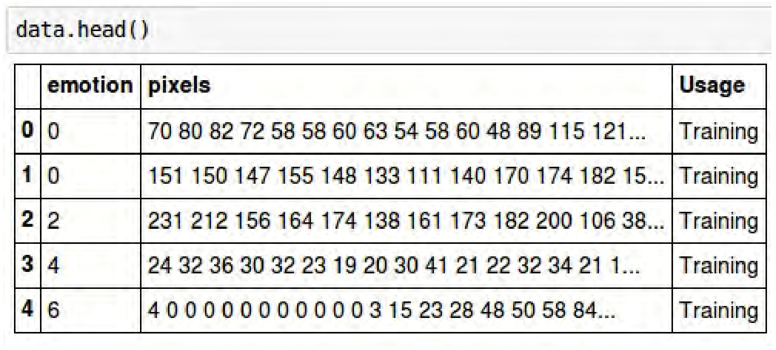
The data has been hosted on Kaggle, and obtained from the source [15]. This dataset was generated for a Facial Expression Recognition Challenge in ICML 2013 by Goodfellow et al.[15] The training data consists of 28,709 examples of 48x48 size images classified into seven groups. And the public test set and private test set included additional 3589 examples in each set. All the data has been collected and generated in labeled images through scripts. The examples are distributed in following ways[11] :

Label	**Training Example**	**Validation Example**	**Test Example**
Angry	3995	467	491
Disgust	436	56	55
Fear	4097	496	528
Happy	7215	895	879
Sad	4830	653	594
Surprise	3171	415	416
Neutral	4965	607	626

Table 3.1: Data Labels Distribution in the Datasets

3.1.2 Extraction

The dataset, however, did not give us the images directly. The images were actually in the form of strings which contained pixel values, along with the labels. The following figure demonstrates the form in which the data was given:



```
data.head()
```

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

Figure 3.1: Original representation of data.

We wrote a script in *Python* to convert the strings of data into 48 × 48 greyscale images. We have included the complete script on Appendix B. The script has generated images in following forms:

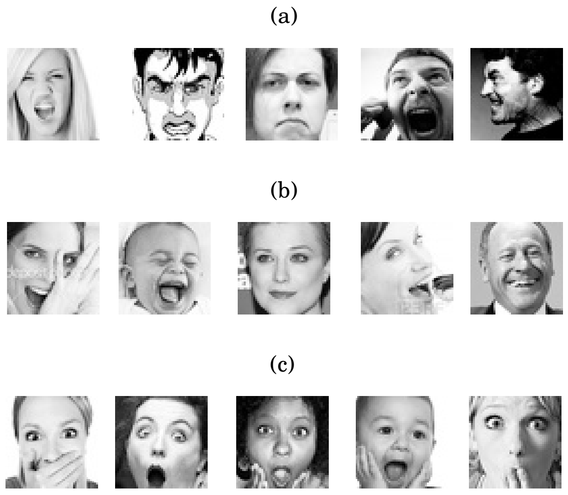


FIGURE 3.2. (a) Sample of images labeled as *Angry*. (b) Sample of images labeled as *Happy*. (c) Sample of images labeled as *Surprised*.

3.1.3 Processing

To run it through ***Inception-v3***, we have not done any pre-processing to the image. In our base paper, some pre-processing has been done before the images has been used. The pre-processing are listed below:

- To feed into ***SVM*** and ***Shallow Neural Network***, they have generated **eigenfaces**[1] from the images.
- Then they have tried to improve the accuracy by providing **Gabor filters**[2] as a input feature.
- For **CNN**, they have subtracted the mean value of each image from every pixel of that image. Then they have subtracted the mean value of all images from every pixel of each image, and divided the value with its variance.

3.2 Processing using Inception-v3

3.2.1 Bottlenecks

The first step was to analyze all the images in the training set and calculate their bottleneck values. **Bottleneck** is an informal term that is often used to refer to the layer just before the final output layer that actually does the classification[2]. Retraining this layer outputs values that eventually help Inception classify the different facial expressions represented in the training data. Simply retraining this layer is sufficient as Inception has already learned from 1,000 different classes as part of ImageNet which is useful to distinguish between new objects.

3.2.2 Training

After bottleneck process in the default setup 4000 training steps have been run, where in each step ten images are chosen randomly to feed through their bottleneck. The resulting predictions then got compared with the actual labels and using back-propagation process the final layer's weight is updated.[2]

The training accuracy gives us the percentage of images used in the current training batch that were labelled correctly. The validation accuracy is the precision on a randomly selected group of images from a different set during training[2]. The training accuracy is less reflective of the performance of the classifier during because it is based on images that have already been learned from and hence, the network is at risk of over-fitting. A better measure is the validation accuracy. If there is significant mismatch between the training accuracy and validation accuracy, then that is indicative that the network is memorizing potentially unhelpful features that don't generalize

[1]See section A.3 of Appendix A.

[2]See section A.4 of Appendix A.

well. **Cross entropy** is a loss function which gives a glimpse into how well the learning process is progressing. The training's objective is to make the loss as small as possible[2]. Inception splits the training data into 3 parts where 80% are used as the training set, 10% are used as validation set, and 10% are used as a testing set during the training. In that way over-fitting is avoided and bottlenecks are fine tuned[2]. Then the validation set has been passed through for tuning and the test set has been sent through for classification.

3.2.2.1 First Training with Inception-v3

Initially we trained the data on the barebones Inception using all the default values for the parameters, which are:

- **Initial Learning Rate** = 0.1
- **Number of Epochs per Decay** = 30.0
- **Learning Rate Decay Factor** = 0.16
- **Step Size** = 4000

3.2.2.2 Result and Analysis after First Training

After our training we have evaluated the test set images. It resulted in an accuracy of 44.83%. Evaluation of test set has given the following result:

	Angry	**Fear**	**Sad**	**Neut.**	**Surp.**	**Disg.**	**Happy**
Angry	**0.383**	0.118	0.036	0.118	0.154	0.075	0.116
Fear	0.196	**0.518**	0.054	0.036	0.089	0.054	0.054
Sad	0.157	0.067	**0.179**	0.079	0.214	0.192	0.113
Neutral	0.116	0.054	0.021	**0.588**	0.075	0.060	0.086
Surprise	0.210	0.069	0.086	0.086	**0.384**	0.031	0.135
Disgust	0.070	0.024	0.043	0.055	0.033	**0.718**	0.055
Happy	0.148	0.049	0.056	0.119	0.161	0.076	**0.390**

Table 3.2: Confusion Matrix of Test Set Evaluation after the First Training

In the confusion matrix, the numbers highlighted in bold is the recall of different classes. Recall is percentage of positive cases that were labeled correctly by the classifier.

$$\text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}} \tag{3.1}$$

We can see that for the class *Sad*, the recall is poor. In the matrix, the cells in red shows that the class *Sad* was misclassified as *Surprise* and *Disgust* in a higher percentage than the recall.

3.2.2.3 Second Training with Inception-v3

In Inception-v3, for a better result, one needs to make sure that the data are better representation of the application that it will encounter[2]. So, in order to improve our results from previously mentioned in section 3.2.2.2, we went through the dataset and removed all images that was not representative of any facial expression. For example:

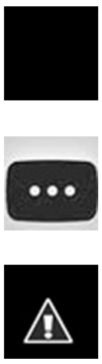


Figure 3.3: Faulty data in the dataset.

Then we retrained Inception with all these faulty images removed. This time also we went with all the default values of the parameters[3].

3.2.2.4 Result and Analysis after Second Training

After evaluating the test set images based on our second training we had an accuracy of 44.99%. And we got following confusion matrix:

	Angry	**Fear**	**Sad**	**Neut.**	**Surp.**	**Disg.**	**Happy**
Angry	**0.259**	0.141	0.176	0.122	0.099	0.056	0.148
Fear	0.089	**0.589**	0.089	0.036	0.054	0.054	0.089
Sad	0.074	0.085	**0.393**	0.071	0.107	0.143	0.127
Neutral	0.054	0.060	0.089	**0.602**	0.049	0.042	0.103
Surprise	0.116	0.092	0.222	0.088	**0.276**	0.020	0.185
Disgust	0.036	0.029	0.125	0.058	0.014	**0.660**	0.077
Happy	0.069	0.056	0.153	0.113	0.107	0.051	**0.45**

Table 3.3: Confusion Matrix of Test Set Evaluation after the Second Training

We can now see that the recalls are better than the inaccurate measurements, a issue we have previously faced on the first training.

[3]Default values have been mentioned in section 3.2.2.1

3.2.2.5 Comparison with the Base Paper

Comparing the result with Chudasama et al.'s results [11] clearly shows us that the performance stands up better than other shallow models. Our result hits an error percentage of 55.01% where other models reach as high as 78.0% to as low as 44.3%.

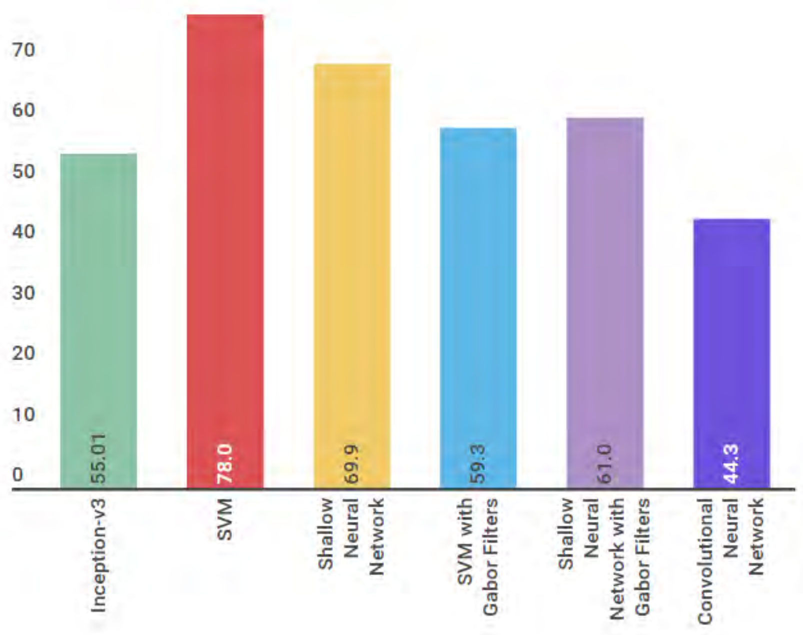


Figure 3.4: Error Percentages between Models

It should be noted that this percentage of error Inception produce is without any preprocessed data or hyper-parameter modification on the model. Where other models have been fed preprocessed version of the data. Taking that into the account, Inception does hold a strong position in error rate comparison.

3.3 Our Simple CNN

3.3.1 Motivation of building a Simple CNN

While comparing the results with the base paper, we could compare it with accuracy. But we could not compare it in terms of training time as the base paper has not mentioned any training time for their tests. However it is important to reach a conclusion on training time as Inception's motivation was in terms of using less computational resources to reduce processing time. Also, for that comparison we can consider only the CNN they have used as others have shown lesser accuracy on the tests. So for such comparison purpose, we have thought of creating a very simple ***CNN*** using **TensorFlow**. As it is a very basic CNN, comparing the result with high level convolutional network like Inception will act as a comparative study between two extreme points of CNN architectures - one is the simplest possible CNN, and other is a 22 layer complicated architecture.

3.3.2 Structure

Our network follows the most basic setup that we have discussed before in section 2.3.2.1. The structure consists of two convolutional layer, two max-pooling layer and a softmax linear regression layer. The activation function for our CNN is the ***RelU*** function mentioned in equation 2.8. The description of the structure is given below:

- **First Convolution Layer**: Our first convolution layer has a patch size of 5×5 and it computes 60 features.
- **First Max Pool Layer**: First max pool layer has a patch size of 2×2 and with stride 2.
- **Second Convolution Layer**: Our second convolution layer computes 120 features for each 5×5 patch.
- **Second Max Pool Layer**: Second max pool layer acts the same as the first max pool layer.
- **Fully Connected Layer**:Fully connected layer takes the input in the reduced size of 12×12 and is connected with 2048 neurons.
- **Dropout Layer**: Our dropout layer performs the dropout function based on the parameter $p = 0.5$[4].
- **Softmax Layer**: Connected with 2048 neurons this final layer performs softmax regression.

All the convolutional layers have a stride of 1.

3.4 Processing data using our CNN

We have processed data with our simple CNN model with a batch size of 20 and with 1434 steps. So in each step it is going to take 20 images from the training set and then learn from it, and this step will be repeated 1434 times.

[4]See the description of dropout layers at section 2.3.2.1 for better understanding

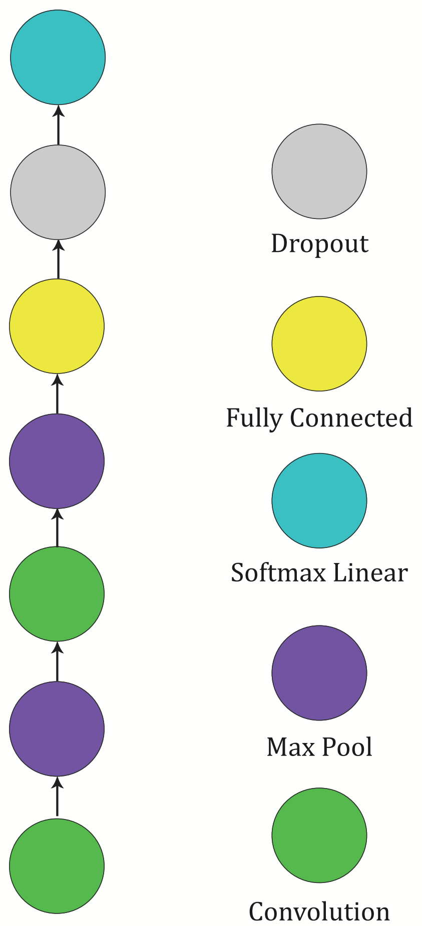
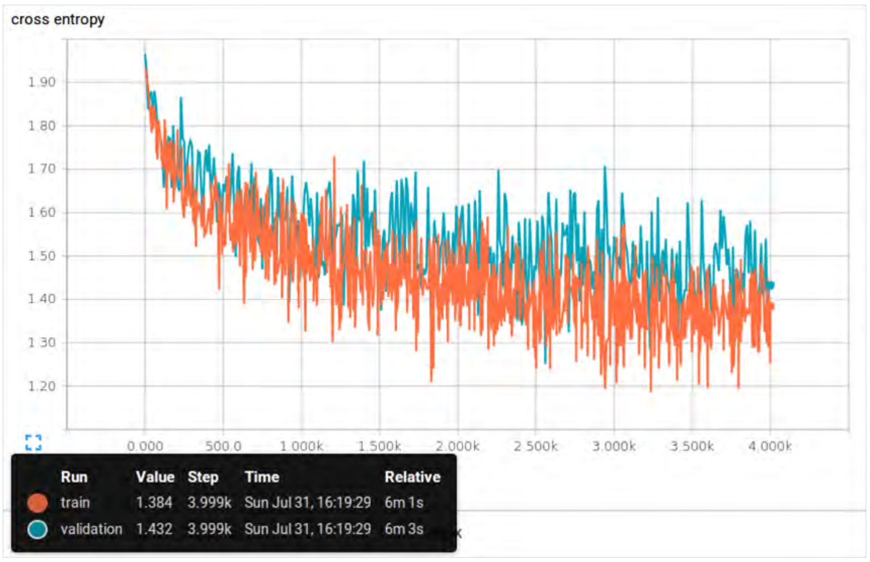


Figure 3.5: Simplified visual representation of our CNN implementation

3.4.1 Comparison with Simple CNN

For a better estimation of performance, following cross entropy graph can be referred which was generated after our first training:



	Run	Value	Step	Time	Relative
●	train	1.384	3.999k	Sun Jul 31, 16:19:29	6m 1s
●	validation	1.432	3.999k	Sun Jul 31, 16:19:29	6m 3s

Figure 3.6: Cross Entropy Graph

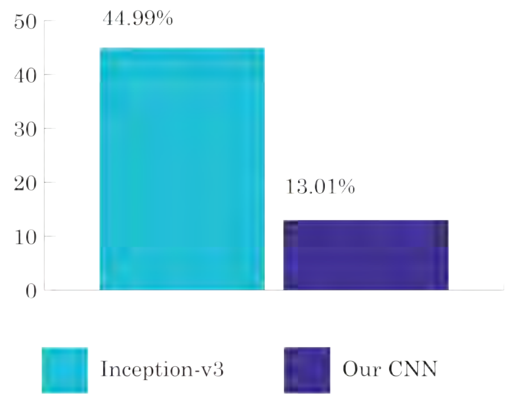
Taking a look at the graph, we can see that it took only 6 minutes to train an entire training set consisting more than 28 thousand images on a Intel Core i5 CPU.

Comparing the results with our CNN shows some vast differences in accuracy. Comparing to Inception's 44.99% accuracy, simple CNN generates a meagre 13.01% of accuracy.

Such differences were expected due to high level nature of Inception architecture. As we have discussed Inception-v3 is a 22 layer convolutional network, while our CNN has only two convolutional layer. But main takeaway from this comparison is not on accuracy, but on *training time*, the amount of time it takes to train more than 28 thousand images. The clear comparison shows us how fast Inception is even with the amount of structural complexity it has. On a 4-core Intel Core I5 CPU, it processes better than the most simplest CNN model possible. Inception has efficiently used a combination of sparse connectivity attribute of convolutional network and combining multiple sparse connectivity together to represent in a dense structure, which ultimately lead to faster learning.

Now for the base paper, even though there is no mention of training time, we can still infer that their CNN model would take longer than the Inception-v3, as their model is structurally more complex than our basic CNN. So if we had to run them on our CPU, training time would be longer.

(a)



(b)

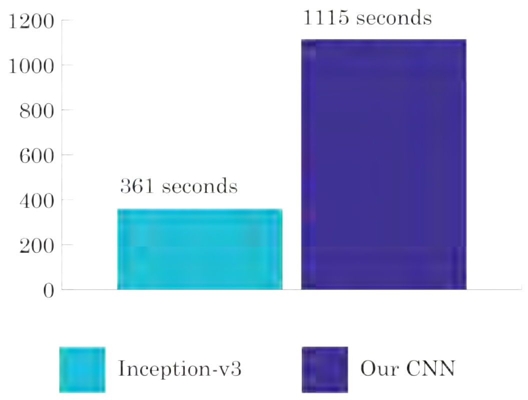


FIGURE 3.7. (a) Accuracy comparison between Inception-v3 and our CNN. (b) Training time comparison between Inception-v3 and our CNN.

3.5 Limitations

3.5.1 Ambiguity in the Dataset

FER-2013 is a very large dataset consisting a huge number of low resolution greyscale images. Being such large scale dataset it is not free from faulty data, and we have seen some example in figure 3.3. But another important issue that arises is ambiguity among data. Some pictures in different classes look very similar, and they are bound to create confusion in the classification. Example of such data is:

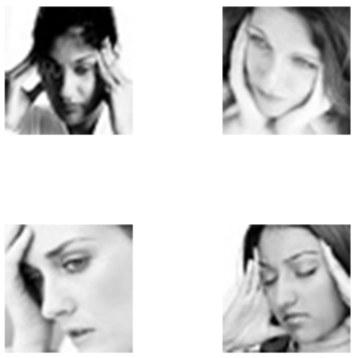


Figure 3.8: This example has 2 images from *Fear* and 2 images from *Sad*. Though one might label all of them as *Sad* due to their ambiguity.

Such limitation imposed by the dataset has resulted into a lesser accuracy. The ambiguity in FER-2013 dataset is so frequent that human accuracy on labeling the images is very low as well. Ian Goodfellow has found out that the human accuracy on FER-2013 dataset was $65 \pm 5\%$ [15].

3.5.2 Less Computational Resource

Our limitation was that we did not have higher processing power to run the comparison of accuracy on a faster processing units. That would have enabled us to use hyperparameters, through which we could have tinkered with the learning rates, and processing batches etc. We believe further experiments with hyperparameters would have resulted into better accuracy. However these parameters will put a lot of computational loads if done in CPU instead of GPU, which can lead upto days.

3.5.3 Requirements of Advanced Knowledge

Inception architecture is highly modular and intricate in structure. Even though the structure can be represented in a understandable way, the subtleties and nuances are very complex in their own places. They pose such a complicated role that, working around a layer can put the model in such a status that we will not be able to trace the results. Understanding such corners and edges of architecture requires advanced amount of knowledge along with further time and resources, including going through the entire source code. So we could not be critical about Inception to the granular points of it.

Chapter 4

Conclusion

4.1 Future Scope

The performance of the Inception can be far better by modifying the hyperparameters, to an extent that it outperform others. Considering it's efficient use of computation power, and recent improvement in processing units for embedded hardware, it can have a vast application on devices that can utilise image processing with less computational cost. One of such application can be facial expression recognition using cellphone cameras. We plan to work such application based on Inception in future.

4.2 Conclusion

Throughout our work, Inception has shown us that (a) It gives a good accuracy when it comes to recognizing facial expression from low resolution images, and (b) it trains faster on a low processing power better than a simple CNN. So we can conclude that the promises made by Inception are held well when tested. However, this is not the final definitive conclusion, and there are many further ways we can move from this point. Considering the limitations we had, the Inception has not been critically analysed from ground up. Overcoming the limitations would result into better understanding the pros and cons of such architecture. Nonetheless, so far the results are promising enough to open doors to many aspects of practical applications. Inception can be the best architecture to be implemented into the devices with low processing units, until a better model succeeds the results Inception has shown.

APPENDIX A

A.1 Gaussian Distribution

Gausssian Distribution is defined by the following equation [8]:

$$\mathcal{N}(x;\mu,\sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}}\, exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right) \tag{A.1}$$

A.2 CNN in CIFAR-10 and CIFAR-100

CIFAR-10 and CIFAR-100 are large dataset collected by Kirzhevsky et al.[18]. In both CIFAR-10 and CIFAR-100 classification challenges, CNN has been dominating according to the result mentioned in [5]. Benjamin Graham with the Fractional Max Pooling[16] approach reached 96.53% accuracy in CIFAR-10, and with the Spatially Sparse Convolutional Network approach he reached 75.7% accuracy in CIFAR-100.

A.3 Eigenfaces

Eigenfaces are the *eigenvectors* derived from the *covariant matrix* of the *probability distribution* in the vector spaces of facial images [29].

A.4 Gabor Filters

Gabor Filter is a linear filter used for edge detection. It has been named after *Daniel Gabor*, and modeled ofter the simple cells of mammalian visual cortex[14].

APPENDIX B

The following portion displays the code used for converting the dataset given as strings of pixel data to 48×48 grayscale images:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from sklearn import svm, metrics

#Read csv file
data = pd.read_csv('fer2013.csv')

#Number of samples
n_samples = len(data)
n_samples_train = 28709
n_samples_test = 3589
n_samples_validation = 3589

#Pixel width and height
w = 48
h = 48

#Separating labels and features respectively
y = data['emotion']
X = np.zeros((n_samples, w, h))
for i in range(n_samples):
    X[i] = np.fromstring(data['pixels'][i], dtype=int, sep=' ').reshape(w, h)

```

```python
#Training set
X_train = X[:n_samples_train].reshape(n_samples_train, -1)
y_train = y[:n_samples_train]

#Classifier
clf = svm.SVC(gamma=0.001, kernel='rbf', class_weight='balanced')

print('Training Classifier...')
clf.fit(X_train, y_train)
print('Done!!!')


#Testing set
X_test = X[n_samples_train : (n_samples_train + n_samples_test)].reshape(n_samples_test,
    -1)
y_test = y[n_samples_train : (n_samples_train + n_samples_test)]

#Prediction
expected = y_test
predicted = clf.predict(X_test)

#Results
print("Classification report for classifier %s:\n%s\n" % (clf, metrics.
    classification_report(expected, predicted)))
```

BIBLIOGRAPHY

[1] *Conv nets: A modular perspective*.
http://colah.github.io/posts/2014-07-conv-nets-modular/.

[2] *How to retrain inception's final layer for new categories*.

[3] *Mnist for ml beginners*.
https://www.tensorflow.org/versions/r0.7/tutorials/mnist/beginners/index.html.

[4] *tensorflow/models*.
https://github.com/tensorflow/models/tree/master/inception.

[5] *What is the class of this image ?*
http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html43494641522d3130.

[6] *Artificial neural network - wikipedia, the free encyclopedia*, 2016.
https://en.wikipedia.org/wiki/Artificial_neural_network.

[7] S. ARORA, A. BHASKARA, R. GE, AND T. MA, *Provable bounds for learning some deep representations.*, in ICML, 2014, pp. 584–592.

[8] I. G. Y. BENGIO AND A. COURVILLE, *Deep learning*.
Book in preparation for MIT Press, http://www.deeplearningbook.org, 2016.

[9] J. BERGSTRA, F. BASTIEN, O. BREULEUX, P. LAMBLIN, R. PASCANU, O. DELALLEAU, G. DESJARDINS, D. WARDE-FARLEY, I. GOODFELLOW, A. BERGERON, ET AL., *Theano: Deep learning on gpus with python*, in NIPS 2011, BigLearning Workshop, Granada, Spain, Citeseer, 2011.

[10] C. BISHOP, *Pattern Recognition and Machine Learning*, Information Science and Statistics, Springer, 2006.
https://books.google.com.bd/books?id=qWPwnQEACAAJ.

[11] B. CHUDASAMA, C. DUVEDI, AND J. P. THOMAS, *Learning facial expressions from an image*.

[12] J. DONAHUE, Y. JIA, O. VINYALS, J. HOFFMAN, N. ZHANG, E. TZENG, AND T. DARRELL, *Decaf: A deep convolutional activation feature for generic visual recognition.*, in ICML, 2014, pp. 647–655.

[13] L. FEI-FEI, R. FERGUS, AND P. PERONA, *One-shot learning of object categories*, IEEE transactions on pattern analysis and machine intelligence, 28 (2006), pp. 594–611.

[14] H. G. FEICHTINGER AND T. STROHMER, *Gabor analysis and algorithms: Theory and applications*, Springer Science & Business Media, 2012.

[15] I. GOODFELLOW, D. ERHAN, P.-L. CARRIER, A. COURVILLE, M. MIRZA, B. HAMNER, W. CUKIERSKI, Y. TANG, D. THALER, D.-H. LEE, Y. ZHOU, C. RAMAIAH, F. FENG, R. LI, X. WANG, D. ATHANASAKIS, J. SHAWE-TAYLOR, M. MILAKOV, J. PARK, R. IONESCU, M. POPESCU, C. GROZEA, J. BERGSTRA, J. XIE, L. ROMASZKO, B. XU, Z. CHUANG, AND Y. BENGIO, *Challenges in representation learning: A report on three machine learning contests*, 2013.

[16] B. GRAHAM, *Fractional max-pooling*, CoRR, abs/1412.6071 (2014).

[17] S. KNERR, L. PERSONNAZ, AND G. DREYFUS, *Single-layer learning revisited: a stepwise procedure for building and training a neural network*, in Neurocomputing, Springer, 1990, pp. 41–50.

[18] A. KRIZHEVSKY AND G. HINTON, *Learning multiple layers of features from tiny images*, (2009).

[19] A. KRIZHEVSKY, I. SUTSKEVER, AND G. E. HINTON, *Imagenet classification with deep convolutional neural networks*, in Advances in neural information processing systems, 2012, pp. 1097–1105.

[20] S. LAWRENCE, C. L. GILES, A. C. TSOI, AND A. D. BACK, *Face recognition: A convolutional neural-network approach*, IEEE transactions on neural networks, 8 (1997), pp. 98–113.

[21] Y. LECUN AND Y. BENGIO, *Convolutional networks for images, speech, and time series*, The handbook of brain theory and neural networks, 3361 (1995), p. 1995.

[22] Y. LECUN, B. BOSER, J. S. DENKER, D. HENDERSON, R. E. HOWARD, W. HUBBARD, AND L. D. JACKEL, *Backpropagation applied to handwritten zip code recognition*, Neural computation, 1 (1989), pp. 541–551.

[23] M. LIN, Q. CHEN, AND S. YAN, *Network in network*, arXiv preprint arXiv:1312.4400, (2013).

[24] S. P. LUTTRELL, *Hierarchical self-organizing networks*, in Proc. 1st IEE Conf. Artificial Neural Networks, 1989, pp. 2–6.

[25] S. OUELLET, *Real-time emotion recognition for gaming using deep convolutional network features*, arXiv preprint arXiv:1408.3750, (2014).

[26] T. SERRE, L. WOLF, S. BILESCHI, M. RIESENHUBER, AND T. POGGIO, *Robust object recognition with cortex-like mechanisms*, IEEE transactions on pattern analysis and machine intelligence, 29 (2007), pp. 411–426.

[27] C. SZEGEDY, W. LIU, Y. JIA, P. SERMANET, S. REED, D. ANGUELOV, D. ERHAN, V. VANHOUCKE, AND A. RABINOVICH, *Going deeper with convolutions*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.

[28] C. SZEGEDY, V. VANHOUCKE, S. IOFFE, J. SHLENS, AND Z. WOJNA, *Rethinking the inception architecture for computer vision*, arXiv preprint arXiv:1512.00567, (2015).

[29] M. TURK AND A. PENTLAND, *Eigenfaces for recognition*, Journal of cognitive neuroscience, 3 (1991), pp. 71–86.

[30] Y. ZHOU AND R. CHELLAPPA, *Computation of optical flow using a neural network*, in Neural Networks, 1988., IEEE International Conference on, IEEE, 1988, pp. 71–78.