Predicting the Emotion of an Individual in an Image with Convolutional Neural Networks

Jaden Wang

January 9, 2016

Abstract

The task of classifying images is trivial for a human mind. However, that is not the case when the computer is asked to accomplish the same task. Explored here are many methods of applying statistical/machine learning to achieve this goal. The method that will be highlighted in this report is the use of Convolutional Neural Networks, which will attempt to determine a facial expression of a person in a given image.

Chapter 1

Background

1.1 Hypothesis

Utilizing Convolutional Neural Networks (CNNs), a statistical model, an algorithm can learn to differentiate and classify facial expressions of individuals inside images.

1.2 Rationale

The task at hand belongs to the domain of Machine Learning, a combination of both the interesting fields of statistics and computer science. Classifying images specifically belongs to a small subset of problems called computer vision; the computer learns to see just as humans do. Recently, many fascinating breakthroughs in artificial intelligence are all related to use of convolutional neural networks in computer vision. The same algorithms that this report explores are used to power self-driving autonomous cars and defeating world-class players at the game of Go. Using substantial knowledge learned from the *Mathematics of Data Management* (MDM4U1) class, one is able to delve upon and understand the topics required to implement Machine Learning algorithms and techniques.

Chapter 2

Introduction to Neural Networks

2.1 Introduction

In order to understand a convolutional neural network, one must first understand what a simple feed-forward neural network is, the basis of all neural networks.

2.1.1 Architecture

A neural network is a statistical model that comprises of a weighted graph/network modelled after the human brain, hence its name. It can be used for various different tasks such as regression or classification. Due to flexibility of this method, a neural network can be considered a universal function. As shown in Figure 2.1, a network has an input layer and an output layer. The input and the output are vectors (or column matrices).

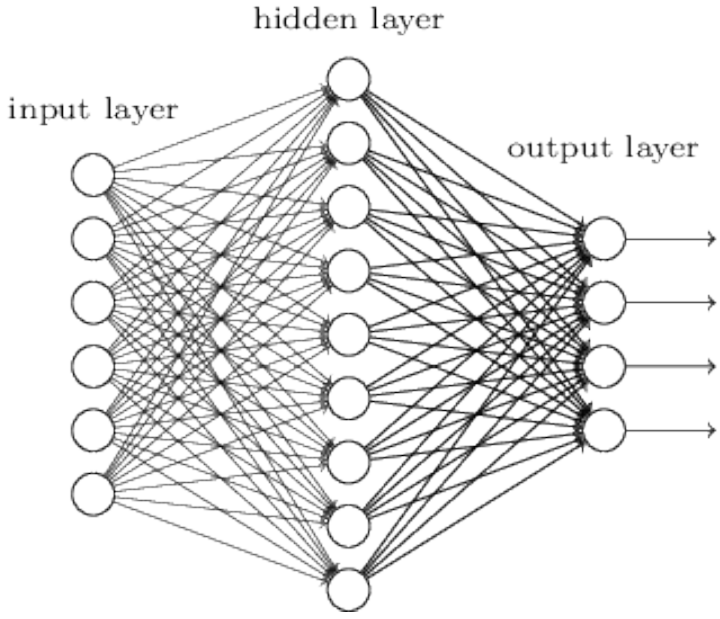

Figure 2.1: A feed-forward neural network.

A network comprises of an arbitrary amount of *layers* of *nodes*. The more layers a neural network has, the *deeper* we say it is. Every node in a layer is fully connected to all the nodes in the adjacent layers. The edges of the network are weighted, meaning that they are each assigned scalar values. Each node contains an activation function.

A few examples of widely used activation functions include:

- Rectified Linear Unit (ReLU)
  $f(x) = max(0, x)$
- Logistic/Sigmoid
  $f(x) = \frac{1}{1+e^{-x}}$

2.1.2 Data Flow

The data in a neural network flows from the input to the output in a single direction. That is why it is called a *feed-forward* neural network. The input values are multiplied by the corresponding edge weight when flowing towards subsequent layers. Since the weights and input are both vectors, then this operation can be represented as $X^T W$ where X is the input vector and W is the weight vector, which would return the values of the next layer. Finally, the activation function is applied upon these new values. This process repeats until all the data has traversed from one side to the other.

2.1.3 Training a Network

The task of training a neural network is an iterative task that falls under the numerical optimization branch of mathematics. The goal of training a neural network is to end up with an optimal set of weights that minimizes the error and maximizes the accuracy.

In order to train a neural network, one must prepare three different datasets: training, cross-validation, and test set. Having these datasets will allow the model to *learn* from its given data and to predict new data. There are many algorithms such as Adam, Momentum, etc to train a neural network. However, almost all of them are based off of back-propagation, which is based off of gradient descent optimization. To get a sense of what gradient descent is capable of, we can try to solve a simple linear regression problem using this algorithm.

Suppose we have a set of ordered pairs:

Table 2.1: Equation: $y = 2x + 1$

x	y
1	3
2	5
3	7
4	9
5	11
6	13
7	15
8	17
9	19

Since the linear equation follows the form of $y = ax + b$, one may solve for the parameters with the given formula:

$$a = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2} \quad b = \overline{y} - a\overline{x}$$

Computing the values using this formula will yield $a = 2$ and $b = 1$. Gradient descent (GD) is an entirely different approach of solving this problem. GD is an algorithm that minimizes a function iteratively. In order to do that, one must first define an error function. In the case of linear regression, this will be the mean of the sum of all the squared errors:

$$Error = \frac{1}{N}\sum_{i=1}^{N}(y_i - (ax_i + b))^2$$

Next, we must use the partial derivatives for both a and b:

$$\frac{\partial}{\partial a} = \frac{2}{N}\sum_{i=1}^{N} -x_i(y_i - (ax_i + b))$$
$$\frac{\partial}{\partial b} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (ax_i + b))$$

For each iteration, we will update the value of a and b using these partial derivatives. With these derivatives, the algorithm will know whether to increase or decrease and by what value. After each iteration, a new set of partial derivatives will be computed and the error value will decrease until it has converged at a global minimum, yielding the optimal line of best fit. Essentially, your error graph will generally tend to look like this:

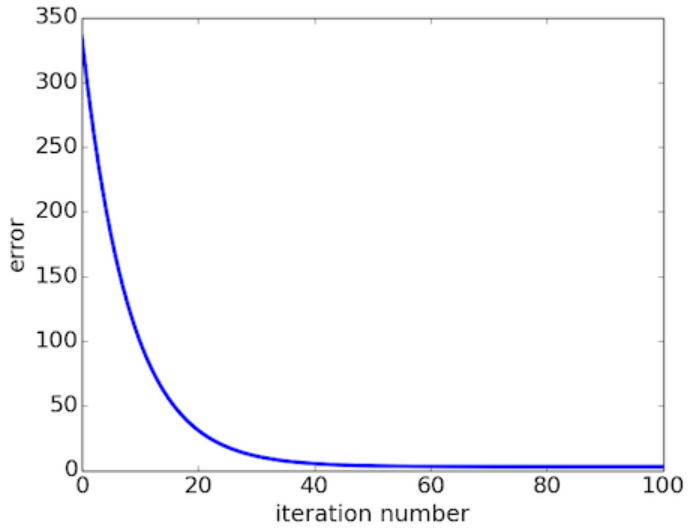

Figure 2.2: Gradient Descent Error

The key component of this algorithm is the computation of the partial derivatives called the *gradients*. Many algorithms, such as the ones list previously to train a neural network, make use of these gradients to optimize their function. In the case of a neural network, the goal is to optimize the weights that will yield the least error.

Back-propagation is the bread and butter of training these neural networks. Many variations of the algorithm have different attributes but they all generally do the same. Here is a brief explanation on how this works. During training of the neural network, we are using the training and the cross-validation labelled datasets. This is to help reduce overfitting, meaning that it would not be biased towards the data set used to train it and will to learn to generalize for all datasets. We start by initializing the network with a random set of weights. In each iteration:

1. The error of both the training and validation is calculated by comparing the network's output to its actual output with its respective data.

2. The gradients are computed to change the weights of the network.

Chapter 3

Procedure

3.1 Data Collection and Preparation

I obtained my datasets from a combination of primary sources and secondary sources. Primary sources involved asking people for a centered eye-level portrait of their face, later to be cropped. Secondary data is from ICML 2013 and was obtained from Kaggle [Goo+13], a website for data science competitions. The training set consisted of 28709 images. Both the test set and validation set were 3589 images each. All the categories were labelled with a unique identifier (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). It is imperative that the data is preprocesesed before being fed into the neural network.

3.1.1 Data Preprocessing

All images were resized to 48 pixels by 48 pixels with Red Green Blue (RGB) channels. Therefore, the input of the neural network is a 48x48x3 tensor. Data preprocessing is required both during the training process and evaluation. The data was then *zero-centered* meaning that the mean of all the images were calculated and subtracted from the image. To calculate the mean, one must add together all the matrices and divide by the amount of matrices in the specified category.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} X_i$$

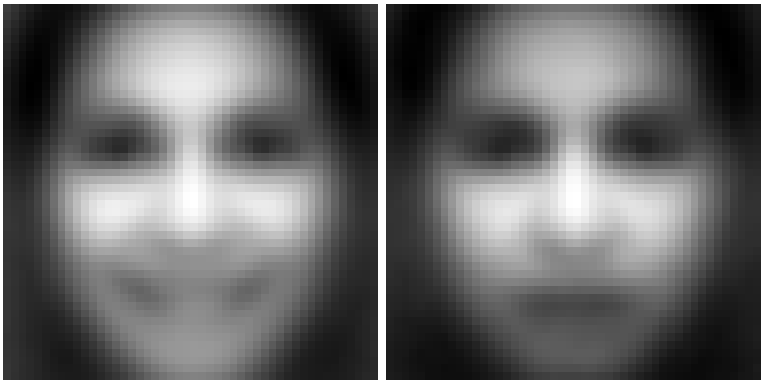

Figure 3.1: The element-wise mean images of happy (left) and sad (right) faces.

The standard deviation of the images were calculated and divided from the image. Normalizing the images uses the same formula for z-scores in normal distributions. One can calculate the standard deviation of all the images and normalize them with these formulae:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|x-\mu\|^2} \; X_{normalized} = \frac{X-\mu}{\sigma}$$

3.1.2 Data Augmentation

Data augmentation is a technique to forge extra artificial training data. New training data is randomly created with the following attributes:

- Flipping left or right.
- Rotations at a small angle.
- Blurring in random positions.

3.2 Training the Convolutional Neural Network

A convolutional neural network follows the same basic principle of a feed-forward neural network, but has many architectural variations that all help the process of image classification.

3.2.1 Architecture

The process of designing a neural network for a specific task is empirical, meaning that trial and error is involved. After various sessions of testing, here is the architecture used for the task of classifying emotions, inspired from architectures of those who have attempted the same task [Enr+16], from left to right:

Normalization	Conv2d	Pool	Conv2d	Pool	Conv2d	Dropout	Full	Full(Softmax)

Convolutional Layers (Conv2d) are not fully connected but are similar to fully-connected layers. Instead these act as layers that will search through the entire image instead of a specific location. Max pooling layers (Pool) are used to reduce the dimensions of the image to save computational resources. Dropout layer is used to randomly throw away information to prevent overfitting. The layers listed above all have the Rectified Linear Unit (ReLU) as their activation function. Since this problem is a classification problem and not a regression problem, the final fully-connected layer has a softmax activation function, which returns a probability distribution of all the classes as the output of the entire network instead of a set of real numbers.

An input tensor has the shape of where the RGB channels are nested inside the 48 x 48 pixels:

$$\begin{bmatrix} [r & g & b] & \dots & \dots \\ \vdots & & \ddots & & \\ \vdots & & & & X_{48,48} \end{bmatrix}$$

Training the entire network took close to one hour each round. Each time a small parameter was tweaked, such as number of nodes and layers, the model had to be retrained.

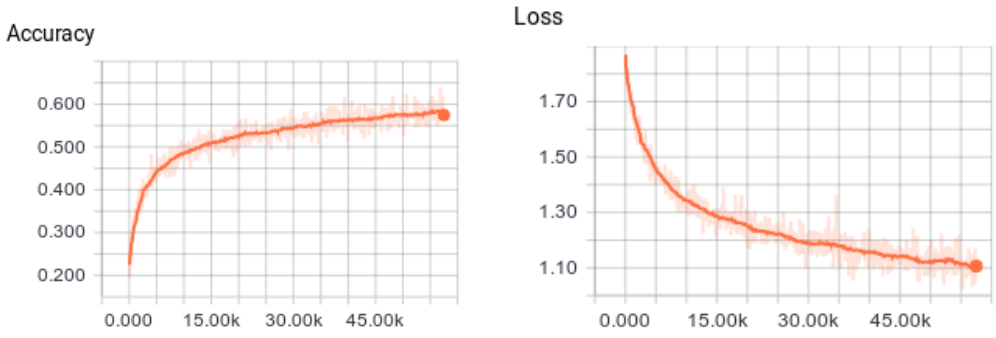

Figure 3.2: After each iteration, the error decreased and the accuracy increased, similar to what is shown in Figure 2.2

3.2.2 Software

A Python software library called *Tensorflow*, written by Google, was used to help with the neural network process. The library is optimized for mathematical operations such as parallel computing in training a network and can take advantage of certain hardware.

3.2.3 Hardware

The software must take advantage of a certain hardware. Since training a neural network is essentially matrix multiplication, this process can be parallelized. The best hardware component for parallel computing is a Graphics Processing Unit (GPU), used for displaying a matrix of graphics on computer monitor. A Nvidia GTX 1050 Ti GPU was purchased to optimize the speed of the training process.

3.3 Analysis and Conclusions

The accuracy seemed to peak at 51 percent of the test set. Of the 3589 test set images, 1856 of them were classified correctly. Although these may seem to be poor results, one must consider a random classifier. Since there were 7 classes to choose from, a random classifier would have a 1 in seven probability of predicting the correct category; around 14 percent accuracy. In a binomial distribution with 7000 images to be tested, then the expected value of correctly classified images would be 1000, from the formula $E(x) = np(x)$. The same distribution *using* the classifier yields an expected value of 3570 $(7000 * 0.51)$, which is an apprehensible result.

3.3.1 Demonstration

To test images from other sources, here is a picture of a celebrity, Nicholas Cage.

Figure 3.3: Nicholas Cage Picture

After running the image through the classifier, it had predicted the expression to be happy. A human would have predicted the same. Here is a graph of the probablity values that it had returned:

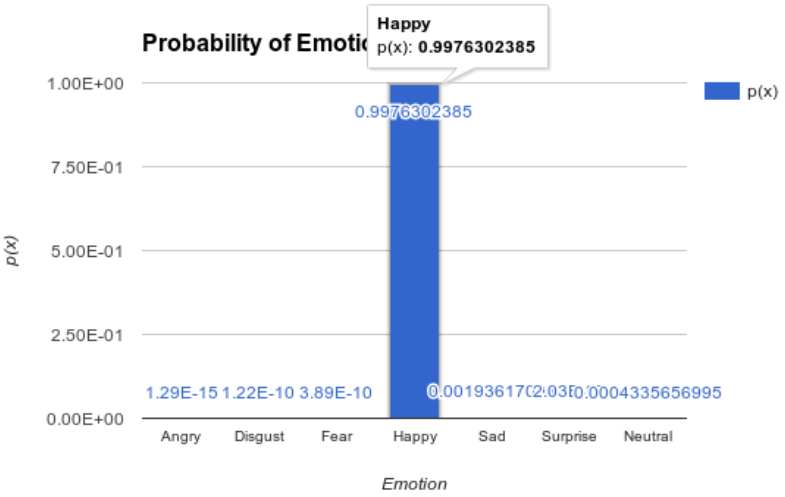

Figure 3.4: Graph of probability values.

The other values were negligible, with the probability of *happy* being 99 percent. It was very confident of its prediction.

3.3.2 Experiment

To make classification experimenting easier, a video camera program was made to take screenshots every second and make a prediction on the fly. After testing, one could assume that the classifier was very sensitive towards lighting and facial position. A small change in lighting could drastically change the end result. Having the face fully centered and at eye-level would produce the most accurate results. Failure to meet these conditions caused the classifier to have a tendency to misclassify everything as *fear*. Another interesting investigation was that the classifier was receptive to mouth movements. Baring teeth or creating an "O" shape would change the prediction. One could conclude that the weights identifying the mouth have been successfully trained.

To help further analyse where the classifier is going wrong, we can construct a confusion matrix based on the test set that deals with false/true positives/negatives, but for multiple classes. Here is a matrix with the actual label on the side verses the predicted label on top.

	Angry	*Disgust*	*Fear*	*Happy*	*Sad*	*Surprised*	*Neutral*
Angry	**96**	37	25	104	134	23	72
Disgust	7	**16**	2	12	15	3	0
Fear	22	25	**94**	120	130	89	48
Happy	3	6	6	**805**	33	12	14
Sad	18	9	45	144	**257**	17	104
Surprised	2	3	32	59	23	**276**	21
Neutral	9	4	26	117	123	35	**312**

The *happy* category seemed to have a remarkable accuracy rate. Coincidentially, the *happy* class also had the largest training data. This could have also caused many other predictions to misclassify as *happy*. Other satisfactory classifications include the *surprised* and *neutral* classes with respect to their respective sample sizes. The performance of the *angry* and the *disgust* classes were poor. Extra training data could perhaps solve the problem of sample size bias.

3.3.3 Problems Encountered

Most of the problems encountered were technical computer problems. A lot of trial and error was involved in creating a network that would produce satisfactory results. During data collection, many participants were unwilling to offer a facial portrait of themselves.

3.3.4 Conclusion

Convolutional neural networks seem to have a satisfactory rate of correctly classifying facial expressions in images. Improvements can definitely be made in the architecture of the network, the preprocessing of the network, and perhaps even the computational power used to train the network.

3.3.5 Applications in Real Life

The use of facial expression classification is particularly useful for companies to gain feedback on their service of their clients. For example, an amusement park could station cameras loaded with facial expression capacity, at their exits to instantly receive feedback of user satisfaction.

Bibliography

[Goo+13] Ian Goodfellow et al. *Challenges in Representation Learning: A report on three machine learning contests*. 2013. URL: `http://arxiv.org/abs/1307.0414`.

[Enr+16] Correa Enrique et al. *Emotion Recognition using Deep Convolutional Neural Networks*. 2016.