Deep Learning Emotion Classification

By: Nick Razzano

Northern Michigan University

CS480 - Senior Project

30 April 2019

**Table of Contents**

**I. Introduction……………………………………….... 1**

**II. Project Overview…………………………………... 4**

**III. Concepts of Convolutional Neural Network……… 6**

**IV. Technologies Used………………………………….. 9**

**V. Implementation Details……………………………. 11**

**VI. Difficulties and Improvements……………………. 13**

**VII. Conclusion………………………………………….. 14**

**Introduction**

Before I introduce my project, I would like to tell a little about myself. I am Nick Razzano, a Computer Science and Mathematics double major here at Northern Michigan University. Ever since I can remember, I always had a gravitational drawing to technology. Whether it was tearing apart an electronic toy to see how it worked or messing around on a computer to learn how an application was developed, I always wanted to know more about the exponentially growing field of computing. In high school I took my first official programming course in python and had a lot of fun. I kept taking as many programming courses I could, finishing the coding assignments before other classes and asking many questions. Once in college, I continued down the road of Computer Science, enjoying most of my classes but some more than others, as I always enjoyed having more coding projects.

 It wasn’t until I had very fortunate internship opportunity that I learned about machine learning and how effective it was becoming in everyday cases. I had heard of the term, but wasn’t aware of its full capability nor did I have a complete understanding of the whole idea and how it worked. And maybe I still don’t, it is a very complex idea with a lot of interesting mathematics behind it, more than I could research in its entirety over one semester. But I learned what I wanted to learn and some, at least getting a grasp on the basics and tweaking different aspects of my project to figure out how best I could improve my results.

 Enough about me. Because of my interest in machine learning, I decided to tackle a specific problem combining Deep Learning and Computer Vision to recognize and classify facial expressions. Like I mentioned, I had an internship where I had experience working with autonomous vehicles and during that time I learned how to think differently about what could be a useful tool to implement in such a machine. The idea of facial emotion classification didn’t come right away, but once I started to think of the possibilities of what it could mean, I knew it was an interesting problem. For instance, if you were in an autonomous vehicle and the way it was driving, maybe a bit aggressive for your liking, how would you let the vehicle know are feeling uncomfortable? Well, a system that could detect your expressions might be able to realize this without you having to intervene. Another great example, imagine just a regular human operated vehicle. If the vehicle could detect intoxication or drowsiness in the driver, that could be a great safety feature. And yes, it is possible to detect intoxication in the face. It is not as easy as others, but possible according to research I came across during this project it is achievable. In any case, I knew this would be an interesting, challenging, and fun project to work on. And it was. Even if I don’t get everything I wanted done, it is something I could continue working on, or expand upon with more time.

**Project Overview**

If you read my introduction or cover page, you know my project was a facial emotion classifier using deep learning. When deciding how to start this project, I quickly came to the conclusion that Python would be the best language to use for something like this. Python is considered to be the first place in the list of all AI development languages because of its simplicity. Because of its easy to learn syntax, AI algorithms can be more easily implemented compared to a language like Java or C++. Another huge advantage to Python is the vast number of libraries available, even just within machine learning including Pybrain, Tensorflow, Lasagne, Deepy, NeuPy and more. Of these libraries, I decided to use Tensorflow. I’ve heard of this library before and after some research I learned how great of a tool it was with the bonus of having really good documentation, which makes developing much easier.

 After starting to work with Tensorflow for a while, including following a tutorial to build a neural network over the CIFAR10 dataset, I learned about another very useful library. Keras is what I like to consider the ‘Python-version’ of Tensorflow. The reason for that is because, while both are Python libraries, Tensorflow can be cumbersome and confusing to look at and develop with. Keras, on the other hand, makes creating a neural network much easier by sitting on top of Tensorflow and using it as a backend. So really I am using both, but Keras is doing a lot of the dirty work for me while I can spend more time on improving my results by focusing on restructuring my network through rearranging layers and manipulating hyper-parameters for specific layers.

 Other useful libraries I used alongside Keras and TF were: NumPy which is a powerful package for scientific computing including n-dimensional array objects, Pandas which has very useful csv reading functionality, OpenCV makes processing and manipulating images, and Sklearn which has a utility called train\_test\_split that wraps input validation and a splitting function to easily split data for training and testing separation.

 Utilizing all of these libraries really helped to pull my project together. The main focus of my project was creating a convolutional neural network that trained off of an existing dataset to build a predictive model that would classify human facial emotions. The goal is to feed a live video stream, or part of it, through the network and print the predicted emotion on the detected face. There were some changes made to the project along the way that I thought were for the best. I originally wanted to use my own dataset, by collecting videos of fellow classmates, strip each video by taking each frame, processing it with grayscale and face detection, and then build the neural network from that. After trying this with 8 students, I spent multiple days building networks that only got to roughly 30% accuracy. Not great. Eventually I found a very large, and free dataset, called *FER2013*, that consists of roughly 34034 unique 48x48 images of different facial emotions. I decided to go with this in the hopes I would have better success with multiple times the amount of data to train my model on. That is exactly what happened, with my first model training at roughly 53% accuracy.

**Concepts of a Convolutional Neural Network**

Before I get into Convolutional Neural Network (CNN), I would like to talk about Machine Learning (ML), Neural Networks (NN), Deep Learning (DL), and Deep neural Networks (DNN) briefly. In 1952 Arthur Samuel, who was a developer at IBM, wrote the first game-playing program to achieve sufficient skill to challenge a world champion, in checkers. This is one of the first projects of a machine learning, proving that a program could learn how to do something without being explicitly programmed to do so. Learning in this context is not learning by heart, like humans, but recognizing complex patterns and making intelligent decisions based on data. Samuel’s ML programs worked remarkably well, even greatly improving the way humans played checkers. This was a great example of success for ML. Not much later, in 1957, Frank Rosenblatt from Cornell Aeronautical Laboratory invented the Perceptron (Fig.1). The Perceptron is a very simple linear classifier but by combining a large number of them in a network like structure was shown to create a powerful model (Fig.2).

 **Fig.1** **Single Perceptron**

 

 **Fig.2 Perceptron Network**

 Unfortunately, neural network research was stagnant for many years after Marvin Minsky and some of his colleagues showed that NNs could not solve problems such as the XOR problem, which is the problem of using an NN to predict the output of XOr logic gates given two binary inputs. However, several modifications have since produced a solution to the problem and many more complex problems.

 Machine learning is a subfield of artificial intelligence, and going further we have deep learning (DL) which is a subfield of machine learning, and is called as such because of the way the network is ‘deeper’ in amount of layers, specifically it must have ‘more than one hidden layer’ to be considered deep. This includes an input and output layer, so more than 3 layers is a Deep Neural Network (DNN). In DNNs, each layer of nodes trains on a distinct set of features based on the previous layer’s output. The further you process through the neural network, the more complex the features can be recognized since they aggregate and recombine features from past layers updated weights.

 Now, I would like to talk about a subset of Deep Neural Networks, namely Convolutional Neural Networks (CNN). CNNs were designed to map image data to an output variable. They have proven their effectiveness so much that they are usually the go-to method for any type of prediction problem involving image data as input. It was for this reason that I wanted to use a CNN to train a model against my image data. For a CNN, the input is traditionally a matrix or field that can also be transformed into a 1-dimensional object. The purpose of that is to allow the CNN to be used more generally on other types of data that has some spatial relationship; human faces for example. Most human faces have similar spacing between things like eyes, ears, nostrils, or the shape of a mouth.

 In any Neural Network, there are different types of layers that can be used to achieve different results. Popular layers, for example, include Pooling Layer which is used to reduce the spatial dimensions, but not the depth, on a CNN. By reducing spatial information, you gain computation performance, but you also have less change of over-fitting your model because of less parameters. Another popular layer is called a Dropout Layer. Dropout is a technique that can also help improve over-fitting a network by sacrificing training performance for more generalization, which is usually a good thing. The idea here is fairly simple, it deactivates certain neurons during the training of the model to force your layers to learn with different neurons (left).

 The last layer I want to talk about here is the Convolution Layer. So really the primary purpose of a Convolution is to extract features from the input image. It achieves this by using small squares of input data to learn these image features while preserving the spatial relationship between pixels. The idea is to compute a new matrix, called the feature map, by sliding a (smaller than image) matrix over the image matrix. A normal grayscale image would have pixels ranging between 0-255, but for the sake of example here we have an image with pixels 0 or 1.

The feature map is very useful for finding features in an image such as a line or a curve. There are different types of desirable feature maps, one example would be for finding edges in a image. In my case, it makes different facial features stand out over less interesting areas parts of the image.

**Technologies Used**

Before I get into TensorFlow and Keras, I would like to speak towards some of the other technologies and libraries I used in this project to get everything done. To start, GitHub is a useful tool that I’ve used a couple times before, but definitely improved on over the course of this project. Because all of my code, for the most part, was kept on DeepLearner, I didn’t commit all the time, but when making advancements in a certain area I would document it on GitHub in case something ever went wrong in future changes.

Both OpenCV and NumPy came in very handy for image processing purposes. Whether it was striping video’s taken for the purpose of input data, or for resizing and applying a grayscale to images before testing for prediction, both work very well together when working with images.

Because I was using Python for my project, and because there were a lot of certain dependencies throughout the project, I used VirtualEnv to create a isolated environment on my computer to keep everything I needed contained for just this project. This is more of a ‘best-practice’ feature, especially when developing with Python, but it was useful for the ease of upgrading libraries via pip.

I ended up using a new dataset, the *FER2013* dataset, which is a very large csv file. Because of this, I decided to use Pandas to parse the data for manipulation. Pandas is very easy to use and creates a dataframe that can access the different fields like a two-dimensional data structure with labeled axes.

TensorFlow (TF) is an end-to-end open source machine learning platform that has many comprehensive, flexible tools and community resources that lets build and deploy neural networks easily. TF is one of the most popular libraries for Deep Learning currently, as it is owned by Google. The name TensorFlow comes from the idea of a tensor, which is a generalization of vectors and matrices and is easily understood as a multidimensional array. For example, a vector is a first order tensor and a matrix is a second order tensor. TF is very useful for its numerical computation and large-scale machine learning capabilities. It bundles together a vast number of ML and DL models and algorithms. While TF is a Python library, the math operations are actually using high-performance C++ binaries, Python is just routing all of the traffic between all the different pieces which in-turn provides high-level programming abstractions. While TF is able to run on almost any target: a local machine, a cloud cluster, iOS or Android, CPU or GPU, it makes a huge difference to train models on a GPU. Using CPU just isn’t powerful enough to perform that many complex computations in short time. It is doable, but making use of a powerful GPU can cut training time down by 3 times the time on a CPU. This was one of the main reasons I wanted to use DeepLearner and put those Nvidia 1080Ti’s to work.

 Another benefit of TensorFlow is the ability to build and train ML models very easily when utilizing intuitive high-level APIs like Keras. I learned about this API while working with TF and was very interested in using it to understand TF more by seeing it from a high level first. As I mentioned, TF is very powerful, it is also very cumbersome and can feel like an alien language when first using it. I would like to learn more about TF in the future, but with the time restraint of this project I really wanted the freedom of manipulating the layers and hyper-parameters of my CNN easily. Keras made this possible, allowing me to make many changes to my network and retrain the model various times.

 Lastly, I would like to talk about Python. It is a language I have used sparingly, but really enjoyed having the opportunity to do a full project in. As someone who has used C++ for almost every class and internship, switching to Python felt like cheating. It did not take long to learn most of the ‘ins and outs’ of the language. Its ease of use and readability greatly helped me throughout my project

**Implementation Details**

One of the most difficult aspects of my project was setting up DeepLearner and the environment for TensorFlow. I haven’t used Linux for very long, but I went into this with a little experience with developing on in the operating system. Something I have never done is work with configuring Nvidia drivers. Setting up TensorFlow can be a really tedious and frustrating process, especially when trying to get TF to communicate with the GPUs. I used TF 1.12.0 because it is the most recent official stable build for Ubuntu 18.04. To get TF 1.12.0 to work, CudNN (>= 7.3.0); for CudNN you need Cuda (>= 9.0); for Cuda you need Nvidia drivers (>= 384.x); and for the Nvidia drivers you need GCC (>= 6). For some of these, like GCC, a simple ‘sudo apt install’ is enough. For Nvidia drivers, Cuda, and CudNN the process is not as simple and I ran into many issues along the way getting these to work. Eventually I had to reimage DeepLearner when I got stuck in a loop of conflicting dependencies. The solution to this was using TF’s official documentation to understand what I needed to install in what order. After doing that, I got everything installed and working together, and was relieving; I could finally start working on my project.

For the implementation of this project, I was advised start with a TensorFlow tutorial to understand the basic idea of building a neural network. Starting with TF and an official tutorial really helped me begin to understand the structure of a neural network and importance of the layers, before jumping in blind to build one myself. After completing the tutorial, I used what I learned from that to implement my own CNN using Keras. Because Keras is very user friendly, high-level, and easy to use, the implementation of the CNN is almost underwhelming to look at if you don’t understand what is going on. Just initializing a model, and adding a few different layers to the model is only a couple lines of code. Below is an example of one of the first implementations of my CNN. It is literally one line to initialize the model, and to add layers. This is before compiling and adding the training and testing sets, and then fitting the model.



But with a simple implementation like this, restructuring the network, changing parameters and activation functions was where I spent a majority of my time, apart from waiting for the model to finish training which took anywhere from 5-16 hour.

 The final part of my project is getting a live video feed from a webcam and passing the frames through the trained CNN to display the predicted emotion. I wanted to add this feature so that I could show off the project in a way that anyone could come up and test out the result. For this, I am again NumPy and OpenCV for the image processing and computations on the image matrices. Keras has a function that allows you to save a model after it is built and then load it later on, which is how I can achieve this part of the project without having to build the network each time.

**Difficulties and Improvements**

 Because of the immense amount of issues I had with getting the system configured to work with TensorFlow and utilize the GPUs, I think that was my biggest difficulty. It was an issue that involved many codependent libraries that needed to be built from source in specific order and also working with communicating with the hardware on the system. After much time spent on official support sites, various forums, and time with professors, I finally decided to reimage DeepLearner, as I mentioned above, to get the Nvidia drivers installed properly to communicate with the GPUs. After that, I mapped out exactly what was needed for each library or application and installed them in order according to TensorFlow’s official instructions; which is what I should have done to begin with. This was a major oversight on my part.

 Apart from the setup, the other challenging aspect of my project was tweaking with my CNN to try and boost my accuracy results. I spent a lot of time researching what others have done to gain success with image classification using CNNs, and found some very interesting and helpful ideas. Not everything worked, but each time I make a change to the network I was gaining information about the effects of the different layers. I also learned how to be patient when waiting for my model to train, and to not be disappointed when the results did not improve the accuracy.

**Conclusion**

I knew going into this project that there was going to be a lot of work involved. I wasn’t sure how much, but I was ready to work hard around any issues that I faced. I certainly had my fair share of challenges, but that should be the case with almost any development project being worked on. There were times that I spent hours on an issue that I thought was impossible to overcome, but asking the right questions or asking for a helping hand from a classmate or professor really brought a new perspective to light and helped solve the problem.

What I wanted to gain from this project was experience with Machine Learning, Computer Vision, Python and all-around improvements to my developing skills. I think I have learned quite a lot about all of these, or at least enough to continue working on similar problems with a strong enough background to understand what is happening and what need to be asked when things go wrong. I had a lot of fun working on this, even though at times it felt impossible or like I was wasting time. There were things that I now clearly would have changed given the information I have now gained, but I think that is an important part of the process.