STA 141C Final Report: Human Face Expression Classification Using Deep Learning and Gradient Boosting Methods

Saurabh Maheshwari Hammad Zahid Ben Millam

Introduction

In this study, we build a human emotion image classifier using a combination of deep learning and decision tree algorithms. The aim of this project is to understand if stacking a gradient tree boosting algorithm with Convolutional Neural Network (CNN) can improve the model performance. If yes, we are interested to find the optimal location to extract the activations from. We work on the activations from 2 layers of our image classification model – the last layer of the ResNet module before flattening the activations and the penultimate layer of the entire model. In addition to classifying images with the activations, we also visualize and compare activations using principal components analysis (PCA).

We begin by creating a transfer learning model by using pre-trained weights from the well-known ResNet50 model, the details of which are specified later in the report. Then, we hook activations from the two above mentioned layers and stack a gradient tree boosting model for predicting emotions. Fast.ai and XGBoost libraries are used for the purpose of building CNN and gradient tree boosting models, respectively. Also, we use the GPU support provided by Google Colabs for training the CNN.

As a result of our work, we conclude that XGBoost implementation over the CNN activations is not recommended. Also, PCA shows that adding a two layered used defined fully connected network at the end of ResNet50 is desirable as compared to just one layer. Finally, average pooling is a recommended way for summarizing the activations compared to averaging the activations across channels that leads to undue loss of relevant information that is important for classification.

The rest of the report has the following sections: Data, Methodology, Implementation, Results, Conclusions, and References.

Data

The dataset we will be using for this study is Karolinska Directed Emotional Faces (KDEF) [1]. The dataset consists of 4900 pictures of human facial expressions. The set of pictures contains 70 individuals displaying 7 different emotional expressions: happy, angry, afraid, disgusted, sad, and surprised. Each expression is viewed from 5 different angles and 2 sessions. The size of each image is 562 by 762 pixels and resolution is 72 by 72 dpi. The dataset can be downloaded from the website [1] and the file format for the images is JPEG. Figure 1 shows some of the images from the dataset.

Hence, this is a multi-class image classification problem, with different 7 classes. The emotions have the following acronyms:

AF = afraid, AN = angry, DI = disgust, HA = happy, NE = neutral, SA = sad, SU = surprised

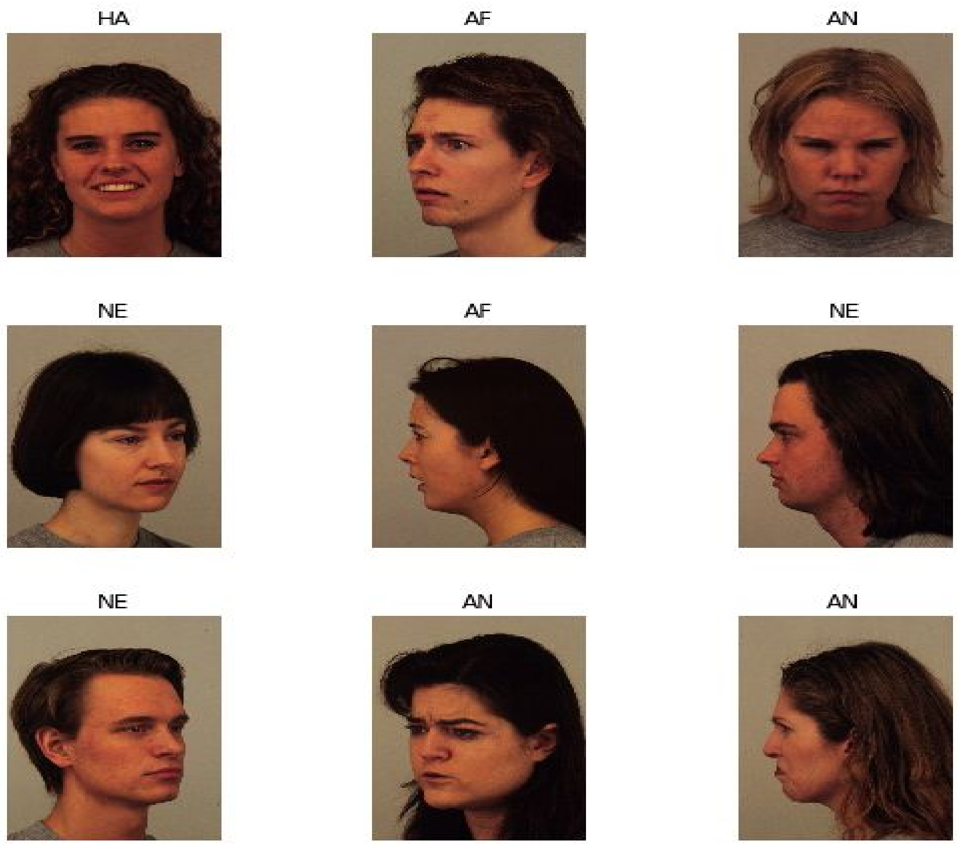


Figure 1. Emotion Dataset Images

Methodology

Training the CNN Network

Convolutional Neural Networks (CNN) are the state of the art algorithms for image classification problems. We begin by creating a CNN model by using a popular technique called transfer learning. Transfer learning is a method that enables initializing weights from a model that has been pre-trained on a large corpus of data. For this work, we use ResNet 50 architecture that has been pre-trained on the imagenet dataset (>14 Million images, over 1000 categories). Transfer learning has the following advantages:

1) Provides a starting point for weight initialization, where CNN layer know how to distinguish between different aspects of an image like corners, curves, colors, etc. Hence, the gradient moves in a less random path
2) As the model is not trained from scratch, the training time is reduced significantly

After the CNN model is trained, the activations were extracted from the following 2 layers for further analysis:

- Layer 1: Last layer (2048 X 24 X 18) of the CNN model before we flatten the activations
- Layer 2: Last layer (512 X 1 X 1) of the CNN model before we output activations for computing probabilities

where, in (aXbXc), a - number of channels, bXc - dimension of a channel

For layer 1, we consider 2 sets of activations, channel and window activations:

- Channel activations: for each of the 2048 channels in layer 1, average pooling is performed to get a vector of 2048 values for an image
- Window activations: for each of the coordinate in the channel, average is taken across all the channels, resulting in a matrix of size 24 by 18 for each image .

We then reshape the layer 1 window activations to achieve a vector of size 432 for an image. Activations from layer 1, channel and window, both represent an image, but in different ways. Each coordinate in Layer 1 window activations could be thought of as representing cumulative features of nearby pixels of the image, whereas each value in Layer 1 channel activations represents a feature of the image (texture, color, etc). Since, layer 1 channel activations are 2048 in length, as compared to 432 length window activations, they seem to contain more information for an image. Figure 2 shown a pictorial representation of the CNN + XGBOOST architecture.

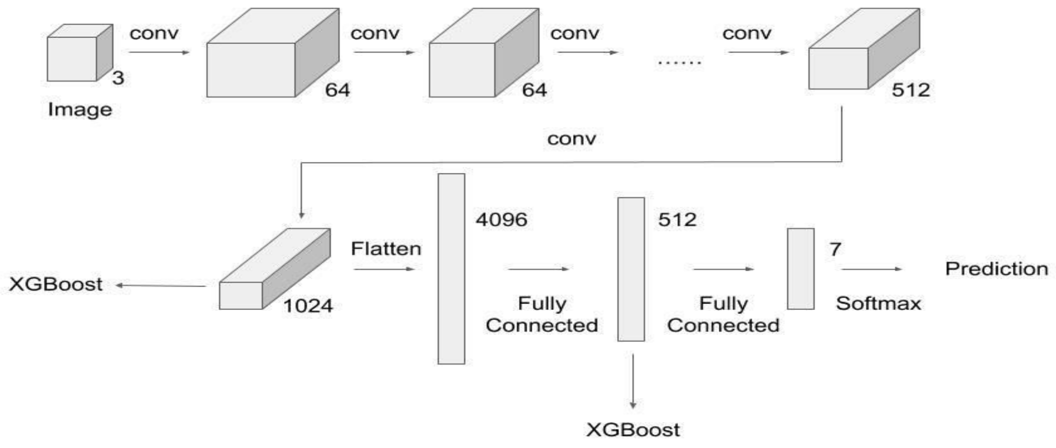


Figure 2. Architecture (CNN+XGBOOST)

XGBoost Classification on CNN Layer Activations

XGBoost is a popular open-source gradient boosting library allowing classification via an ensemble of 'weak' decision tree learners. We used XGBoost to fit classification models on Layer 1 activations (channel and window) and Layer 2 activations.

Implementation

CNN Implementation

Fast.ai is a library that is built on top of pytorch that enables you to create highly sophisticated neural networks with a few lines of code. It provides features such as data augmentation, tuning and using differential learning rates, and others that makes the process of machine learning user friendly. Thus, we decided to use Fast.ai for building the CNN model.

The CNN layers at the beginning are well versed with classifying primitive features like corners and lines, whereas the last layers are trained to recognize complex data specific features like dog versus cat [2]. Once the weights are imported, we replace last few layers from the model with a fully connected 2 layer neural network (with dropout, batch normalization and ReLu activations). Then, we freeze the ResNet layers and just train the newly added layers to make the model learn features specific to our dataset. Once, the last layers are trained, we unfreeze all the layers and fine tune them with differential learning rates. Initial layers are trained with lower learning rate as compared to last layers because the final layer weights need to be trained to learn highly complex image features unlike the first few layers.

Important considerations while creating the CNN ResNet50 model:

1) For the purpose of classification, we will be splitting the dataset into training, validation and testing sets. While splitting, we made sure that a subject appears only in one of these sets to keep the process of learning unbiased. 8 subjects (4 male, 4 female) have been randomly allocated to validation set, and similarly 8 other subjects were added to test set. Validation set is used for tuning the parameters, while final accuracy is judged on the test set
2) The learning takes place in 3 steps. We first use 128 by 128 pixel cropped images, then, we update weights using 256 by 256 pixel images and finally full sized images. The idea behind the progressive learning is that starting with smaller images will help the weights to converge to a particular space and then training with bigger images will be faster
3) Data Augmentation: This is a technique in which we modify the training set images by applying various methods like zooming, cropping, rotating, warping, and changing the contrast, such that different versions of images are created with the same label. The main advantage of this technique is that additional images are created for training without spending resources for getting more images. Also, this helps in preventing the model from overfitting so it will generalize better. It has also been known to increase the classification accuracy [3].
4) Tuning learning rate: One of the most important aspects of training a deep learning model is to tune the learning rates. Fast.ai enables you to find the optimum learning rate by plotting losses for various learning rates (Figure 3a). It is recommended to choose a learning rate corresponding to the maximum slope on the decreasing part of the curve.

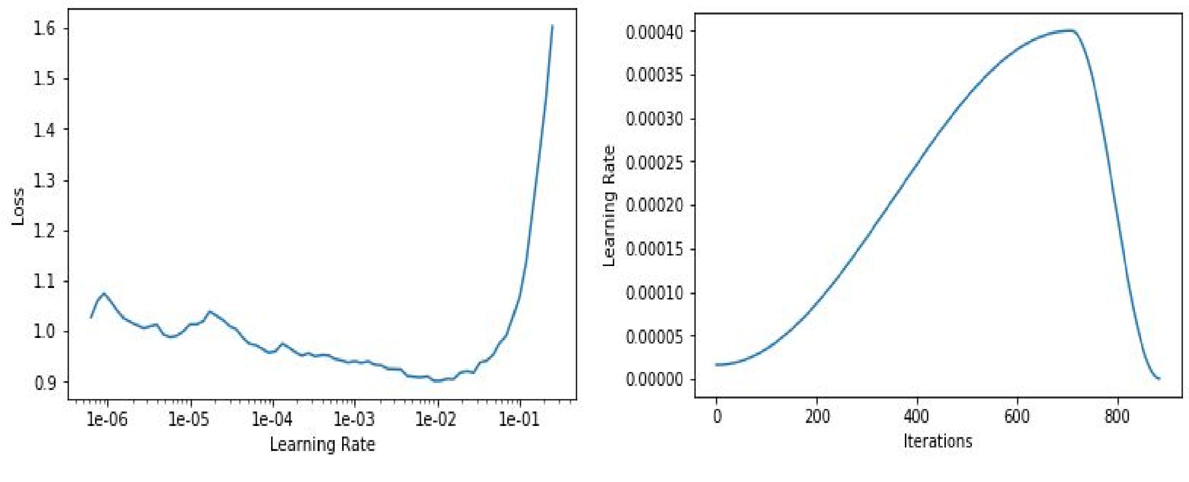


Figure 3(a). Tuning learning rate, (b) One cycle learning.

5) One cycle learning: This is a technique, where we train the weights with a lower learning rate initially that prevents them from jumping around randomly. Then, we increase the learning rate to make the learning faster. Once, the weights get into the optimum space, we lower the

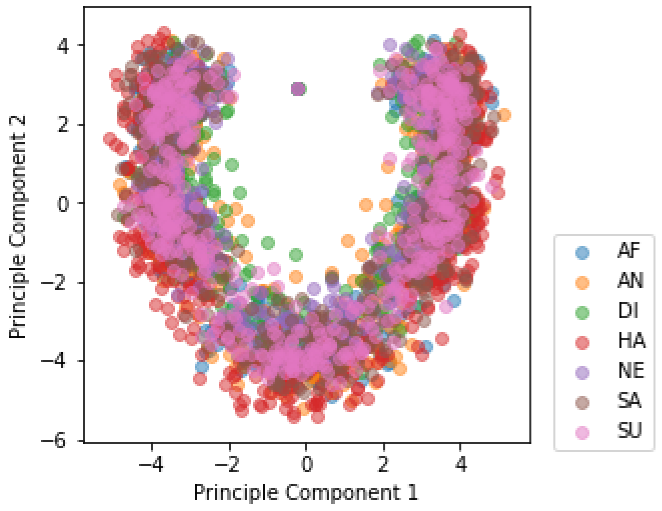
learning rate to make the weights converge. Figure 3b depicts this phenomena. This process has been known to produce improved results [4].

XGBoost Implementation

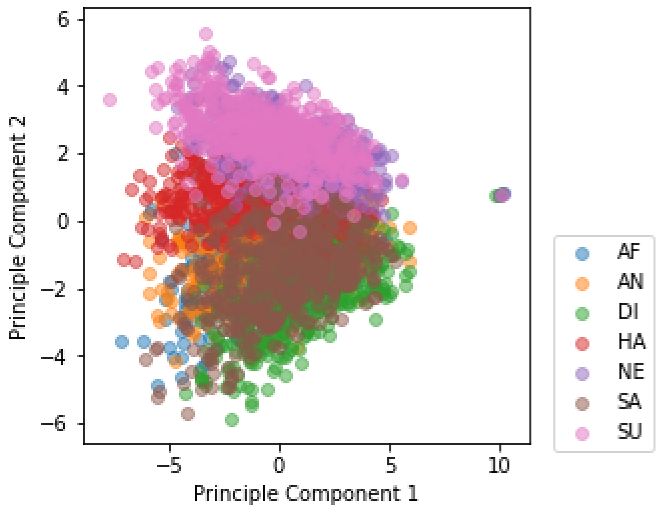
We used the Python package implementation of XGBoost running in Jupyter notebooks, and we used grid search to tune XGBoost's hyperparameters in the following order: maximum tree depth together with minimum sum of weights for all observations required in a child, gamma (the minimum reduction in loss required for a split), the row and column subsampling, L1 and L2 regularization, and finally we decreased the learning rate.

Visualizing the Activations with Principal Component Analysis

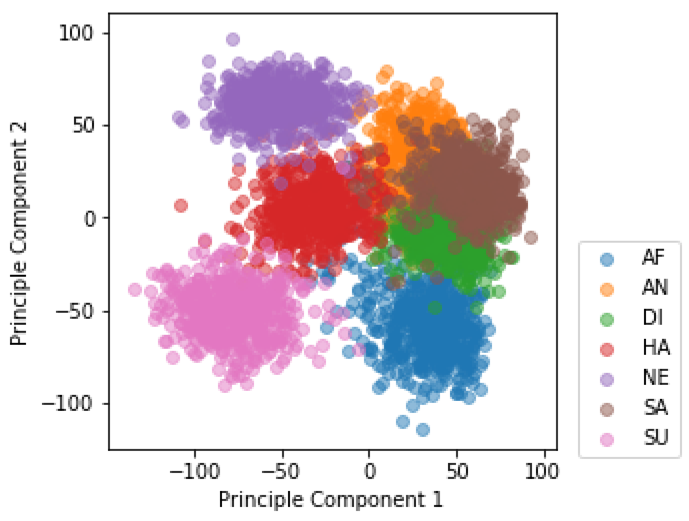
After extracting the activations, we apply principal component analysis (PCA) to study the cluster structure. Figure 4 represent the PCA plots on 3 sets of activations. The principal components for activations in layer 2 have a well separated structure as compared to layer 1 activations. This means that by layer 2, the CNN model has successfully learnt features to distinguish among different categories.



(a)



(b)



(c)

Figure 4. PCA on (a) Layer 1 window activations, (b) Layer 1 channel activations, (c) Layer 2 activations

Results

Here we present the test set accuracy results for the CNN and XGBoost classification:

Algorithm	**Test set accuracy (%)**
CNN	93.32
CNN + XGBOOST (Layer 1 Window Activations)	35.33
CNN + XGBOOST (Layer 1 Channel Activations)	89.80
CNN + XGBOOST (Layer 2)	92.79

Conclusions

We summarize with the following conclusions:

1. CNN and CNN + XGBOOST (Layer 2) model show comparable performance on the test set. This implies that the benefit of replacing the last layer with the gradient tree boosting model is not recommended for this dataset
2. Neural Network features such as Batch Norm and Dropout help control the overfitting given the complexity of these models. Capabilities of Fast.ai help implementing these features with ease, which led to a better accuracy with minimal manual effort for fine tuning the parameters unlike XGBoost

3. PCA shows that the 2 layer fully connected network at the end is just sufficient for successfully predicting emotions. Adding fully connected network with only one layer would be insufficient as can be seen from the pca plot and adding an additional layer to the existing layers could lead to increased training time with not as much improvement in the accuracy

4. CNN + XGBOOST model on Layer 1 channel activations are far better than the window activations. This implies that taking the mean across channels leads to loss of important information. Thus, average pooling is a recommended way to summarize activations over taking the mean across the channels.

5. XGBoost performed better on Layer 2 vs Layer 1 (channel and window), which is expected, since the Layer 2 activations benefit from the preceding layers after Layer 1. The preceding layers and Layer 2 were trained to further learn the image data, so these last activations contain more refined information relevant to classification (lower entropy and reduced noise) vs the earlier Layer 1

References

1) http://kdef.se/home/aboutKDEF.html
2) https://arxiv.org/pdf/1703.01127.pdf
3) https://drive.google.com/viewerng/viewer?url=https://arxiv.org/pdf/1712.04621.pdf
4) https://towardsdatascience.com/finding-good-learning-rate-and-the-one-cycle-policy-7159fe1db5d6