Emotion Recognition on *AffectNet* via Convolutional Neural Networks

CS230 - Final Report

Oseas Ayerdi (SUID: oadelso) - MS in MS&E
Christopher Bucknell (SUID: bucknell) - MS in MS&E

Project mentor: Patrick Cho

March 23, 2018

Abstract

Emotion recognition has a wide-range of applications and it is a relevant problem to tackle. Thus, through the use of Convolutional Neural Networks, we seek to recognize different emotions from human faces. Using the *AffectNet* database of manually annotated images (11 categories), we train different network architectures to tackle the task. We obtained a model that exceeded human performance, using accuracy as a metric to judge performance.

1 Introduction

Recognizing emotions from images has a wide variety of applications ranging from health-care to education. Potential applications are centered around human-computer interactions, where service robots having this feature can provide a variety of services (e.g. calling out for help when it detects that a patient is under stress or pain). Other applications include: testing user experience of software and products (Are they enjoying it? Or, are they annoyed by it?); testing the impact of ads; and looking at the effect of emotional states on performance in school or at work (Kolakowska *et al.*).

Dr. Mohammad Mahoor's research team at the University of Denver has one of the largest available datasets to tackle this problem, *AffectNet*. *AffectNet* contains over $440,000$ manually-annotated images into eleven, discrete categories, these being: *Neutral*, *Happy*, *Sad*, *Surprise*, *Fear*, *Disgust*, *Anger*, *Contempt*, *None*, and *Uncertain*. We intend to develop an improved Convolutional Neural Network to recognize all these distinct categories. We begin by giving a brief overview of the data in section 2; in section 3 we walk through the base model and additional steps taken to improve the starter code performance, as well as the final results. We end with concluding remarks, outlining current limitations and potential, next steps in section 4.

2 Data

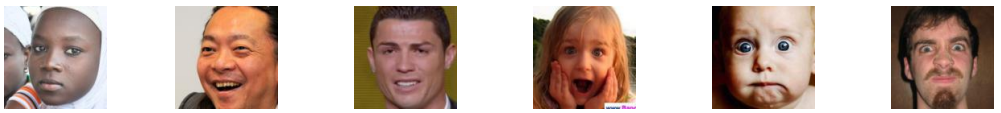
We downloaded the entire *AffectNet* dataset in their original format (i.e. with an average length/width of 425 pixels). As a first step, we resized the images to a 128-by-128-pixel format; we used the Image function from the PIL module to carry out the resizing. Then, we labeled the images with the 11 different categories utilizing the dictionary in *AffectNet*, following the same format as the examples in the starter code (i.e. 'label' followed by an underscore, followed by an image id). During this process, we handled images that were not in the dictionary (did not have a proper label) and documents that were either corrupted images, or an entirely different file-type.

The number of images per category differ greatly, the breakout of the total is as follows: Neutral (18%), Happy (32%), Sad (6%), Surprise (3%), Fear (2%), Disgust(1%), Anger (6%), Contempt (1%), None[1] (8%), Uncertain[2] (3%), and Non-face (19%). Examples of the different type of

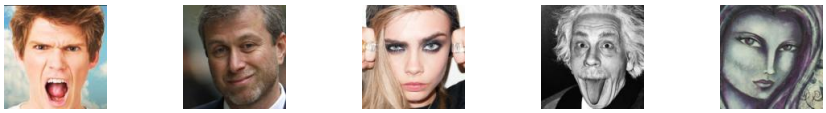
[1] None of the previous emotions.
[2] Labelers were not able to agree.

expressions can be found in Figure 1.

Figure 1: Examples for each category.



(a) Neutral (b) Happy (c) Sad (d) Surprise (e) Fear (f) Disgust

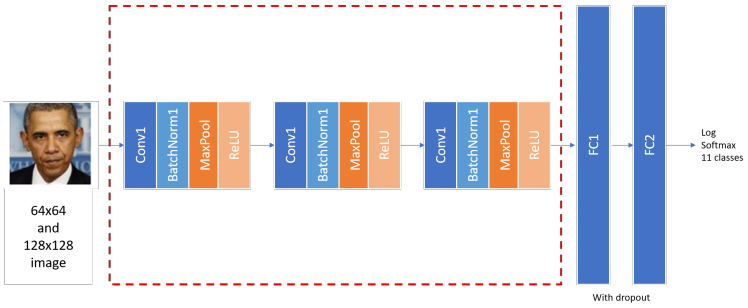


(g) Anger (h) Contempt (i) None (j) Uncertain (k) Non-face

3 Approach

We began by editing the Pytorch Vision project in Github (Nair *et al.*). Specifically, we made the following changes: Firstly, *net.py* was edited by changing the dimensions of the last fully-connected layer. This was originally of size 6 for the six different hand signals. For our project, we need the size to be 11, in order to include the 11 aforementioned, distinct categories. Secondly, we have the script printout the labels and outputs from the model for the validation dataset so we can do error-analysis. Finally, we edited the code so that it would read the correct models for both the train and validate datasets, i.e. we removed the *_sings* extension where applicable. The high-level outline of the network is shown in Figure 2, with the part that we intend to manipulate during our project contained in the red rectangle.

Figure 2: Baseline architecture.



Our approach was to slowly convert the existing architecture into other types that have been proven to be good in image classification, these being: resnets (Zagoruyo *et al.*), densenets (Huang *et al.*), and inception networks (Szegedy *et al.*).

3.1 Baseline Results

We trained the baseline model on a total of 390,969 images; the corresponding validation dataset was composed of 19,542 images. Below are the values for the best achieved accuracy score, and its corresponding values for the loss for both the training and validation sets:

Table 1: Results for baseline model.

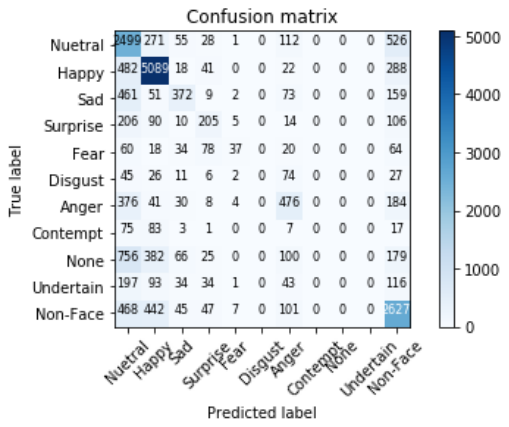
Set	*Accuracy*	*Loss*
Training	0.587	1.276
Validation	0.602	1.221

There is no evidence of over-fitting, so most likely we won't add any further regularization features.

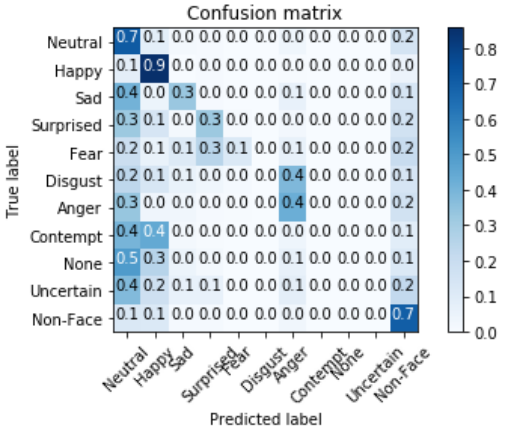
To see what the accuracy is by category, we provide the resulting confusion matrices below with absolute and normalized values:

Figure 3: Confusion matrices.

(a) Absolute Values



(b) Normalized



We see from the results that the model performs extremely well in the *Happy* category, even outperforming the results published in Mahoor's paper (i.e. a reported accuracy for Microsoft's off-the-shelf Cognitive Services Emotion API of 85% for said category). On the other hand accuracy is below 50% for all other categories with the exception of *Neutral* and *Non-Face*. We will try to perform a series of steps to improve the overal performance, these are detailed in the next section.

3.2 Other Models

We began by carrying the following steps and having as our underlying metric be the overall accuracy. First, we increased the resolution of the images from 64 by 64 to 128 by 128, and ran the resulting dataset on the base model. The scripts *data_loader.py* and *net.py* were edited accordingly, specifically the resizing row commented out in the former, and the input dimensions of the first fully-connected layer was changed so that the number of predicted labels will remain unchanged.

As a second step, we did a small alteration to the architecture by passing the output of the first ReLU activation to the output of the third batch normalization, before the final maxpooling; additionally, we increased the number of channels in the convolution layers. The resulting model, *ResNet*, along with the dimensions of the convolution layers is presented at the end of this section.

A third step was to do further data augmentation. We did this by adding a second transformation to the data, in addition to the random flip already in place in the starter code. The transformation is the *ColorJitter* function, with the following parameters: brightness $= 0.4$, contrast $= 0.4$, saturation $= 0.4$, and hue $= 0.1$.

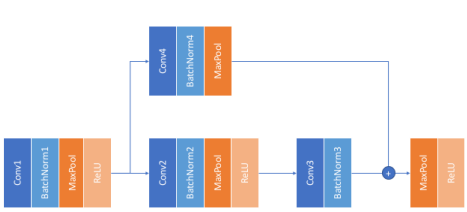
A fourth attempt to improve the overall accuracy is to build a simple densenet by changing the *ResNet* model shown above in the following ways. First removing the maxpool component of the first layer; secondly, adding a fourth Conv-NormBatch-MaxPool-ReLU block; thirdly, joining the output of the first ReLU component to said additional layer, before its ReLU component; finally, adding the output of the second ReLU component to this final (fourth) layer, also before its ReLU component. The resulting architecture, labeled *DenseNet*, as well as the dimensions of the different convolution layers is outlined at the end of this section.

A fifth attempt was to increase the dense net section of the architecture by adding a fifth Conv-NormBatch-MaxPool-ReLU component (labeled *Conv8*), as well as adding an inception layer be-

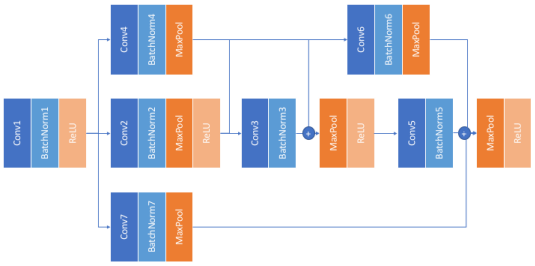
fore the two fully connected layers. More specifically, we also added *Conv9* to connect the output of *Conv1* to *Conv8*, *Conv10* to connect the output of *Conv2* to *Conv8*, and *Conv11* to connect output of *Conv3* to *Conv8*. The numbering of the *'Convs'* depends on when they were added in the process, and otherwise stated, the filtering, padding, and stride are 3, 1, and 1, respectively.

The output of *Conv8* is then put into a typical inception layer, concatenating the outputs of: one maxpool-conv1x1, one Conv1x1, one conv1x1 followed by *Conv12* (5x5 Convolution, with padding of 2), and one conv1x1 followed by *Conv13*. The resulting concatenating tensor is then passed on to *Conv15*.

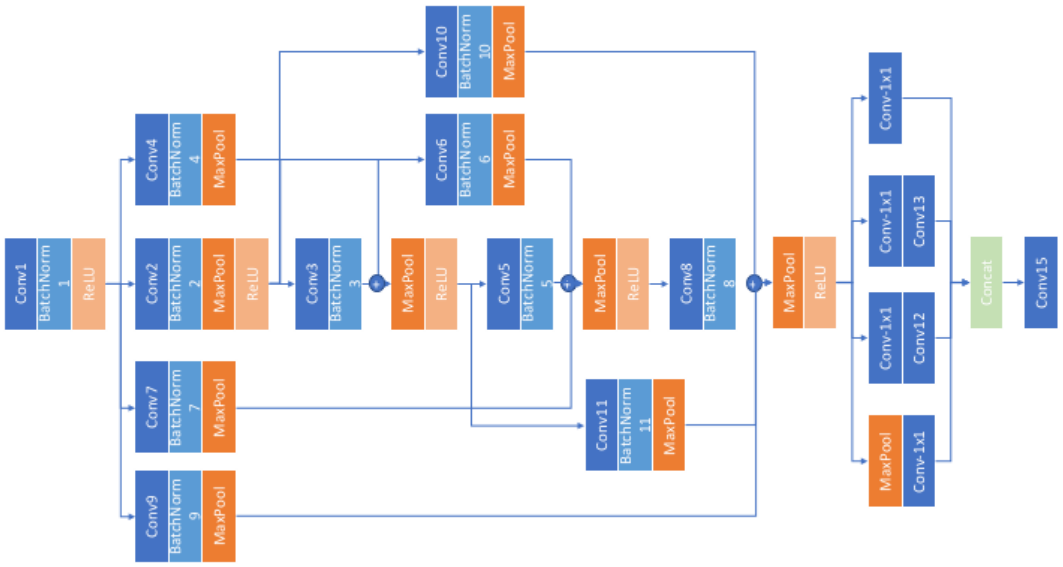
Figure 4: Different network architectures.



(a) ResNet



(b) DenseNet



(c) Hybrid

The dimensions of the different convolutions' outputs (n_h, n_w, n_c) are specified in Table 2 for the instance with the 128-by-128 images (with the exception of the base model).

3.3 Results

The results of these changes can be seen in the Table 3. For all the models, the hyperparameters stayed fixed; these were: learning rate = 0.001, batch size = 32, drop out rate = 0.8 and number of channels = 32. As a reference, we put the performance of both team members in trying to label a total of 100 images into the 11 categories, considering two scenarios: with and without training.

Below we show the confusion matrix of the latest model (i.e Hybrid), showing that these changes in architecture do lead to modest improvements over the baseline, in particular with the label *Fear*.

Table 2: Dimensions of convolutions.

Component	$Dimensions_{Hybrid}$	$Dimensions_{DenseNet}$	$Dimensions_{ResNet}$	$Dimensions_{BaseModel}$
Conv1	[128, 128, 32]	[128, 128, 32]	[128, 128, 32]	[64, 64, 32]
Conv2	[128, 128, 96]	[128, 128, 96]	[64, 64, 96]	[32, 32, 64]
Conv3	[64, 64, 192]	[64, 64, 192]	[32, 32, 192]	[16, 16, 128]
Conv4	[128, 128, 192]	[128, 128, 288]	[64, 64, 192]	n/a
Conv5	[32, 32, 288]	[128, 128, 288]	n/a	n/a
Conv6	[64, 64, 288]	[64, 64, 288]	n/a	n/a
Conv7	[128, 128, 288]	[128, 128, 288]	n/a	n/a
Conv8	[16, 16, 192]	n/a	n/a	n/a
Conv9	[128, 128, 192]	n/a	n/a	n/a
Conv10	[64, 64, 192]	n/a	n/a	n/a
Conv11	[32, 32, 192]	n/a	n/a	n/a
Conv12	[16, 16, 32]	n/a	n/a	n/a
Conv13	[16, 16, 32]	n/a	n/a	n/a
Conv15	[16, 16, 128]	n/a	n/a	n/a

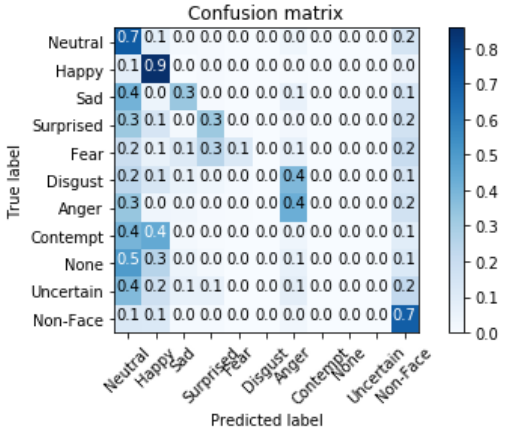
Table 3: Results for all models, including human performance (with and without training)

Model	*Accuracy*	*Loss*	*#Parameters*
Human Error w/o Training (Oseas)	0.570	n/a	n/a
Base Line 64x64 Images	0.602	1.221	1, 143, 307
Base Line 128x128 Images	0.607	1.200	4, 289, 803
ResNet Line 64x64 Images	0.607	1.188	2, 613, 227
ResNet Line 128x128 Images	0.611	1.183	9, 691, 115
Base Line 128x128 Images w/ DA	0.602	1.210	4, 289, 803
Human Error w/ Training (Christopher)	0.630	n/a	n/a
DenseNet 128x128 Images w/ DA	0.632	1.131	22, 320, 971
Hybrid	0.634	1.131	11, 961, 579

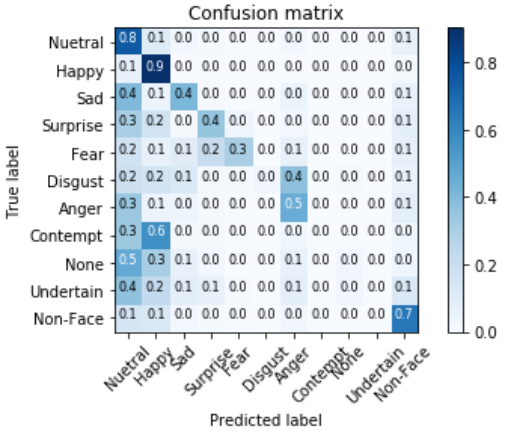
The normalized confusion matrix is presented in Figure 5 for the base and dense model. Through error analysis, we see that *sad* images are often categorized as *neutral*. Also, *contempt* is often labeled as *happy*. We believe that the these two latter categories have shared features in pictures, as it can be seen in Figure 1 (i.e. the mouth gesture of *contempt* may be interpreted as *happy*). However, further analysis must be conducted to prove this hypothesis, specifically through the use of saliency maps.

Figure 5: Normalized confusion matrix for base and dense model.

(a) Base Model



(b) Hybrid Model



4 Concluding Remarks

Facial recognition is a feature with multiple applications, ranging from health-care to education. In this paper we have shown the steps taken to train and improve the general accuracy of the base model using the *AffectNet* database. Testing the human accuracy on a set of 100 images does point out that the starting model did perform quite comparably, and through the presented modifications of the original network-architecture, we improved the performance to the level of a trained human. Future work would center around hyper-parameter tunning (e.g. introduce decay on the learning rate), modifying the loss function to account for the data imbalance of the dataset, and use of 1x1 convolution to reduce the total number of parameters of the final model.

5 References

A. Kolakowska, A. Landowska, M. Szwoch, W. Szwoch, and M.R. Wróbel, "Emotion Recognition and Its Applications," in Advances in Intelligent Systems and Computing, 2014.

A. Mollahosseini; B. Hasani; M. H. Mahoor, "AffectNet: A Database for Facial Expression, Valence, and Arousal Computing in the Wild," in IEEE Transactions on Affective Computing, 2017.

C. Szegedy; V. Vanhoucke; S. Ioffe; J. Shlens; Z. Wojna, "Rethinking the Inception Architecture for Computer Visions," Google, 2016.

G. Huang; Z. Liu; L. van der Maaten; Q. Weinberger, "Densely Connected Convolutional Networks," Facebook Research, 2017.

S. Nair; Genthial, Guillaume; Moindrot, Olivier, "Classifying Images of Hand Signs", GitHub repository: https://github.com/cs230-stanford/cs230-code-examples/tree/master/pytorch

S. Zagoruyo; N. Komodakis, "Wide Residual Networks," Paris Tech, 2017.

6 Contributions

Below is a table outlining the main work that was done in preparation for this report:

Name	*Contribution*
Oseas Ayerdi	Edit of starter code, training models, report write-up.
Christopher Bucknell	Download and preparation of data, code debugging, and report write-up.

The sections of the starter code that were edited, in addition to the resizing code, can be found in Github, via the link: https://github.com/oadelso/emotion_recognition