Machine Learning Engineer Nanodegree

Facial Emotions Recognition

Karthik Balasubramanian May 16th, 2019

I. Definition

Project Overview

Facial emotions are important factors in human communication that help us understand the intentions of others. In general, people infer the emotional states of other people, such as joy, sadness, and anger, using facial expressions and vocal tone. According to different surveys, verbal components convey one-third of human communication, and nonverbal components convey two-thirds. Among several nonverbal components, by carrying emotional meaning, facial expressions are one of the main information channels in interpersonal communication. Interest in automatic facial emotion recognition (FER) has also been increasing recently with the rapid development of artificial intelligent techniques, including in human-computer interaction (HCI), virtual reality (VR), augment reality (AR), advanced driver assistant systems (ADASs), and entertainment. Although various sensors such as an electromyograph (EMG), electrocardiogram (ECG), electroencephalograph (EEG), and camera can be used for FER inputs, a camera is the most promising type of sensor because it provides the most informative clues for FER and does not need to be worn.

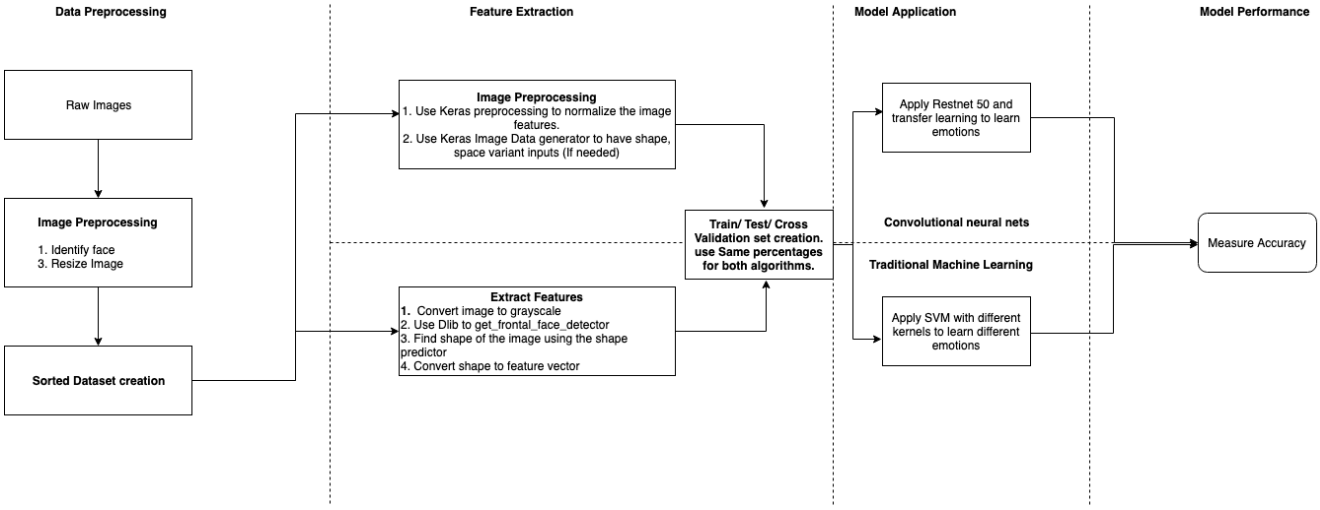
My journey to decide on this project was exciting. My motive was to compare the performances of Deep neural nets in the contemporary research to heuristically learned pattern recognition methods. Facial emotional recoginition/ pattern recognintion had been in research since long. The following academic papers were very helpful in

1. Giving a historic overview of research in Facial Emotional Recognition
2. Deciding on a posed dataset with seven different emotions
3. Developing a baseline algorithm
4. Improving the Facial Emotions Recognition using Deep Convolutional Neuralnets

Problem Statement

The objective of this project is to showcase two different solutions in solving the problem of Facial emotional recognition from a posed dataset. Both the solutions are based on the problem space of supervised learning. But the first solution I propose is more involved and has more human interference than the second solution which uses state of art artificial neuralnets. The goal is to compare the two approaches using a performance metric - i.e how well the supervised learning model detects the expression posed in a still image. The posed dataset has labels associated with it. The labels define the most probable emotion. After running our two different supervised learning model solutions, I will compare the performance of these solutions using the metrics I define in the next section.

Here is the basic structure in attempting to solve the problem.



1. Data preprocessing - Steps done to create a clean dataset
2. Feature Extraction and Normalization methods - The supervised learning approach uses some hand crafted shape prediction methods while the deep neuralnets have data normalization steps. Both these methods are followed by creation of train, valid test set splits
3. Model Application - Model train and test phase.
4. Model Performance - Performance of the models are compared with the metrics defined below.

Metrics

This is a problem of classifying an image to its right emotion. A multi-class classification problem. Our immediate resort to any such problem will be

1. Accuracy - is defined as

   $(TruePositives + TrueNegatives) / (TruePositives + TrueNegatives + FalsePositives + FalseNegatives)$

But when the dataset is not uniformly distributed across classes we use cross entropy loss.

1. Cross entropy loss - Which compares the loss between a vector and a true label in the SVM model.In the deep networks it compares the distance between two vectors. The predicted vector has the probabilities of all the classes. The true vector is a one-hot encoded vector of output labels. The lesser the cross entropy the better the model is. We should also aim at minimizing this loss (i.e closer to 0). In deep learning models, a softmax layer is introduced in the final layer to get the probability vector of classes from the matrix multiplicated computations.

   $CE = -\sum_i^C t_i log(s_i)$

   Where t_i and s_i are the groundtruth (label or one hot encoded vector) and the probability for each class i in C classes.

II. Analysis

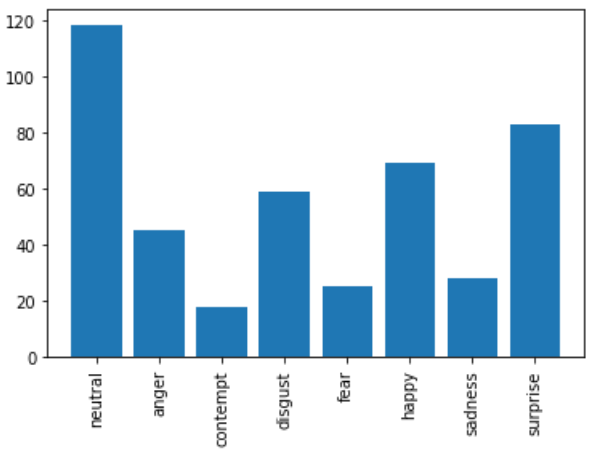
Data Exploration

I use Cohn-Kanade dataset. This dataset has been introduced by Lucey et al. 210 persons, aged 18 to 50, have been recorded depicting emotions.Out of 210 people, only 123 subjects gave posed facial expression. This dataset contains the recordings of their emotions. Both female and male persons are present from different background. 81 % Euro-Americans and 13% are Afro-Americans. The images are of size 640 *490 pixels as well as 640* 480 pixels. They are both grayscale and colored. in total there are 593 emotion-labeled sequences. There are seven different emotions that are depicted. They are:

1. 0=Neutral
2. 1=Anger
3. 2=Contempt
4. 3=Disgust
5. 4=Fear
6. 5=Happy
7. 6=Sadness
8. 7=Surprise

The images within each subfolder may have an image sequence of the subject. The first image in the sequence starts with a neutral face and the final image in the sub folder has the actual emotion. So from each subfolder (image sequence), I have to extract two images, the neutral face and final image with an emotion. ONLY 327 of the 593 sequences have emotion sequences. This is because these are the only ones the fit the prototypic definition. Also all these files are only one single emotion file. I have to preprocess this dataset to make it as an uniform input. I will make sure the images are all of same size and atmost it has one face depicting the emotion for now. After detecting the face in the image, I will convert the image to grayscale image, crop it and save it. I will use OpenCV to automate face finding process. OpenCv comes up with 4 different pre-trained classifiers. I will use all of them to find the face in the image and abort the process when the face is identified. These identified, cropped, resize image becomes input feature. The emotion labels are the output.

Here is the statistical report of the dataset after I do all the preprocessing.



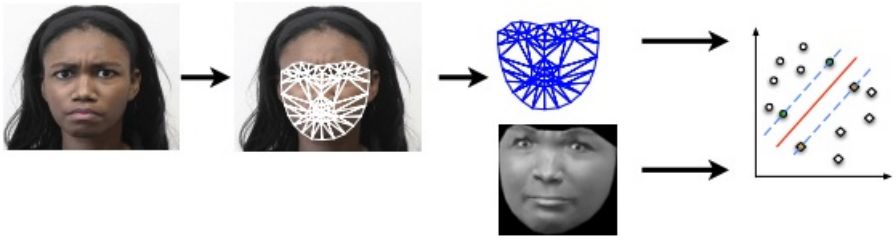
Abnormalities in the Dataset
I am aware of the following abnormalities in the dataset. but I am still going with the dataset.

1. The dataset is not uniformly distributed. So using Accuracy as a metric will lead us into Accuracy Paradox. Hence I have introduced another metric called log loss or categorical cross entropy as another metric.
2. We have a very small dataset per emotion after preprocessing. We had to drop many image sequences as they had no labels.

Exploratory Visualization

Before we jump into finding solutions to the problem by defining the algorithms, we have to understand the feature extraction process. I will be using standard preprocessing methods in deep learning models as I plan to use transfer learning. I transfer learn the emotions from state of art models like VGG16 and Xception. They have their own pre processing methods. All I have to do is present the image in a format (face identified, cropped, resized image) to get preprocessed to a format that these state of art models want to. But there is a significant education needed to understand the feature extraction process to implement the baseline models.

Understanding Feature Extraction Process In Baseline models



The Feature Extraction process has 2 different phases.

1. Finding the face - Libraries like dlib has functions like `get_frontal_face_detector` which is handy to identify the face region
2. Extracting the features in the face - This is where most of the research in the past has gone into. It has been done so far by realizing through manual interference. One of the method is called Facial Action Coding System (FACS) which describes Facial expression using Action Units (AU). An Action Unit is a facial action like "raising the Inner eyebrow". Multiple Activation units when combined expresses the emotion in the underlying face. An example is provided below.

AU	*Name*	N	AU	*Name*	N	AU	*Name*	N
1	*Inner Brow Raiser*	173	13	*Cheek Puller*	2	25	*Lips Part*	287
2	*Outer Brow Raiser*	116	14	*Dimpler*	29	26	*Jaw Drop*	48
4	*Brow Lowerer*	191	15	*Lip Corner Depressor*	89	27	*Mouth Stretch*	81
5	*Upper Lip Raiser*	102	16	*Lower Lip Depressor*	24	28	*Lip Suck*	1
6	*Cheek Raiser*	122	17	*Chin Raiser*	196	29	*Jaw Thrust*	1
7	*Lid Tightener*	119	18	*Lip Puckerer*	9	31	*Jaw Clencher*	3
9	*Nose Wrinkler*	74	20	*Lip Stretcher*	77	34	*Cheek Puff*	1
10	*Upper Lip Raiser*	21	21	*Neck Tightener*	3	38	*Nostril Dilator*	29
11	*Nasolabial Deepener*	33	23	*Lip Tightener*	59	39	*Nostril Compressor*	16
12	*Lip Corner Puller*	111	24	*Lip Pressor*	57	43	*Eyes Closed*	9

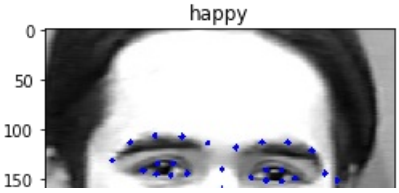
Table 1. *Frequency of the AUs coded by manual FACS coders on the CK+ database for the peak frames.*

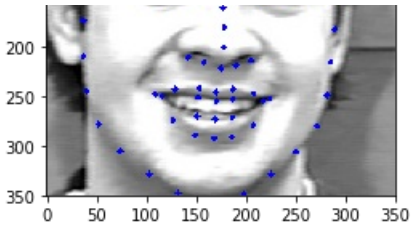
Emotion	Criteria
Angry	AU23 and AU24 must be present in the AU combination
Disgust	Either AU9 or AU10 must be present
Fear	AU combination of AU1+2+4 must be present, unless AU5 is of intensity E then AU4 can be absent
Happy	AU12 must be present
Sadness	Either AU1+4+15 or 11 must be present. An exception is AU6+15
Surprise	Either AU1+2 or 5 must be present and the intensity of AU5 must not be stronger than B
Contempt	AU14 must be present (either unilateral or bilateral)

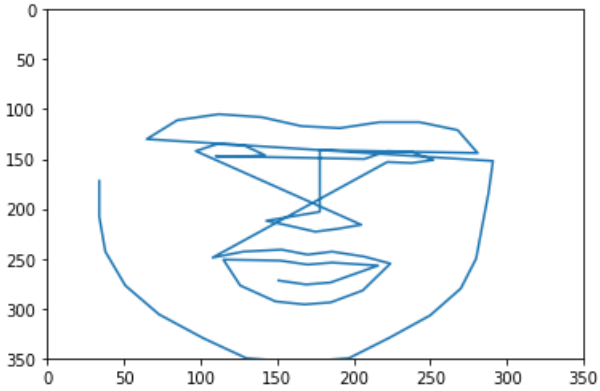
Table 2. *Emotion description in terms of facial action units.*

I use dlib's `shape_predictor` and its learned landmark predictor `shape_predictor_68_face_landmarks.bat` to extract AUs.

The below image helps us understand the facial action units.







The `shape_predictor_68_face_landmarks` above have extracted 67 points in any face in both X and Y axis from the image presented. This X and Y points when combined becomes a Facial Landmark. They describe the position of all the “moving parts” of the depicted face, the things you use to express an emotion. The good thing about extracting facial landmark is that I will be extracting very important information from the image to use it and classify an emotion. But,

There are some problems when we directly capture these facial landmarks.

- They may change as face moves to different parts of the frame. An image could be expressing the same emotion in the top left pixel as in the bottom right pixel of another image, but the resulting coordinate matrix would express different numerical ranges and hence the two images can be classfied to different emotion instead of the same emotion.Therefore we need a location invariant coordinate matrix to help us classify an emotion.

The solution to this problem is derived in the following way.

1. Find the center of the shap predictor vector
2. Calculate the distance between all the shape predictor points to their center
3. Calculate the angle at which these points find themselves relative to the center point.

What we now have is the relationship between all the points with the center point and how they are relatively positioned in the 2D space.Each tuple will have the following values `<x, y, distance_from_center, angle_relative_to_center>`. This additional information to each coordinate makes it location invariant. i.e There is a way to derive these points in the 2D system. These becomes features to our baseline models.

Example feature vector

```
[34.0, 172.0, 143.73715832690573, -163.42042572345252]
```

Algorithms and Techniques - Baseline

I have chosen Support vector machines (SVMs) to map the different facial features to their emotions. SVMs attempt to find the hyperplane that maximizes the margin between positive and negative observations for a specified emotion class. Therefore its also called Maximum margin classifier.

We use libSVM which uses one vs one classifier. i.e It will create $(K * (K-1))/2$ binary classifiers in total - where K here is number of classes 8. A total of 28 binary classfiers are created.

Linear SVM

> Definitions taken from Cohn-Kanade+ paper

A linear SVM classification decision is made for an unlabeled test observation `x*` by,

$$w^T x* >^{true} b$$
$$w^T x* <=^{false} b$$

where w is the vector normal to the separating hyperplane and b is the bias. Both w and b are estimated so that they minimize the risk of a train-set, thus avoiding the possibility of overfitting to the training data. Typically, w is not defined explicitly, but through a

linear sum of support vectors.

Polynomial SVM

The kernel methods in SVM are used when we don't have lineraly seperable data. Kernel methods transform the data to higher dimension to make them seperable. By default, we have our feature set expressed to a 3 degree polynomial.

Parameters passed for baseline models

1. `random_state` - This is like seed value for model to return same response everytime we run it.
2. `probability` - We have asked the model to provide probability scores on different categories.
3. `kernel` - linear/ poly
4. `tolerance` - Tolerence for stopping criteria for the model.

Algorithms and Techniques - Deep Neural Nets

Transfer Learning
I have implemented transfer learned Neural Nets. My goal here is to do very minimal work, reuse the wealthy knowledge of deep networks that have been proven before for image detection. The concepts are the same, but the task to identify is only different.

Here are the steps I am planning to take to make my model more generic and expressive to capture the emotions.

- Feature Extraction - Preprocess the images. Convert all the images to 4 dimensional tensors.
- Get Bottlenect features using the State-of-Art network weights.
- Create a new model to train the bottleneck features to capture the emotions
- predict the emotions from Bottleneck test features.
- Analyze the confusion matrix
- Compare performances with Baseline model results.

State-of-Art networks used

1. VGG16
2. Xception

Model implemented to train bottleneck features

VGG16 bottleneck based model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 4104
=================================================================
Total params: 4,104
Trainable params: 4,104
Non-trainable params: 0
_________________________________________________________________
```

Xception bottleneck based model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_2 ( (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 16392
=================================================================
Total params: 16,392
Trainable params: 16,392
Non-trainable params: 0
_________________________________________________________________
```

Parameters passed for deep neural nets

1. Train set - Shuffle and randomly choose 80 % of training data
2. Validation set - Shuffle and randomly choose 20 % of training data
3. `batch_size` = 20 (Use 20 examples to train, backpropagate and learn every epoch)
4. `epochs` - I have used 20 epochs. That is 20 instances of training.
5. `callbacks` - ModelCheckpoint to save the best epoch weight
6. `gradient optimizer` - Adam
7. `loss` - categorical crossentropy

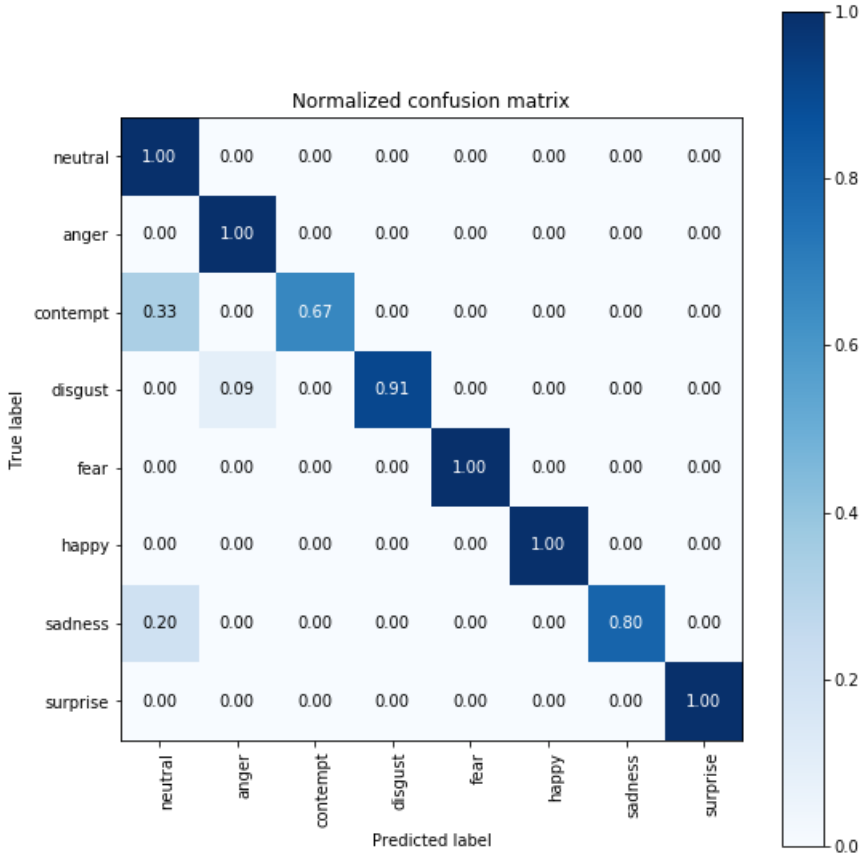
Benchmark

The Baseline models perform pretty well. They are applied after doing handcrafted feature extractions specified above in the Exploratory Visualization section.

Baseline Algorithm	Cross Entropy Loss - Train	Cross Entropy Loss - Test	Accuracy - Train	Accuracy - Test
Linear SVM	0.31	0.57	1.0	0.84
Polynomial SVM	0.31	0.61	1.0	0.81

I chose Linear SVM as the baseline model and I want to see if the transfer learned neural nets are performing better in terms of getting lower Cross Entropy loss and higher accuracy in the test set.

Linear SVM Test data Confusion Matrix



III. Methodology

Data Preprocessing

The data preprocessing steps are two-fold. The first is extracting and converting image files to dataset. Second is extracting features and converting them to train/test/valid sets to suit for Baseline and Deep Neural nets.

Dataset creation steps

1. Download the following zip files from Cohn-kanande+ website
   - extended-cohn-kanade-images.zip
   - Emotion_labels.zip
2. Download haarcascade files from the github
3. Download shape predictor file from here

4. rename the images folder as `source_images`
5. rename the lables folder as `source_emotion`
6. Create two folders `pre_dataset` and `dataset` . `pre_dataset` arranges images by emotions and `dataset` checks if images has a face and resizes the faces of all images to 350 * 350 grayscale images.
7. copy the peak frames of each image sequence to the specified emotions folder in pre_dataset. Please read about image sequences in this paper.
8. Save all the first frame for all image sequences per person. Choose one first frame image per person at random. They will go into neutral folder of `pre_dataset`
9. After doing a face identification check with haarcascades filters, move the images to their respected emotions folder from `pre_dataset` to `dataset` parent folder.

Feature Extraction and Train/Test Split - Baseline

After educating yourself from the Exploratory Visualization section, use the following helper functions.

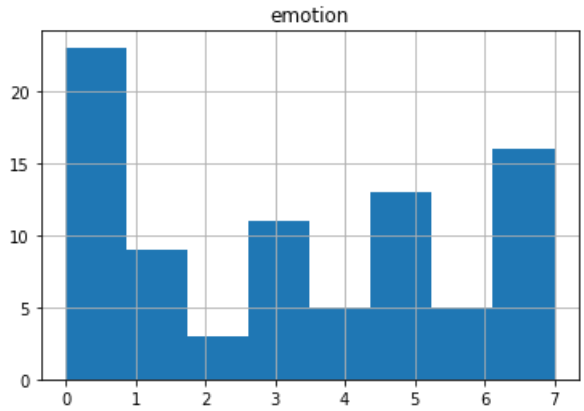
1. `get_files` to randomly split the data in each emotion folder to training and test files. I have used 80/20 split.
2. `make_sets` runs across each emotion folder and gets the training and test files from `get_files` function. It then converts each image to feature using `get_landmarks` function and tags the file to the emotion label. Thus we have our training_data, training_labels, test_data, test_labels

Feature Extraction and Train/Test/Valid Split- Deep Neural Nets

Now that we have train and test sets obtained from the above process, we want to make our predictions more robust. So I introduced validation set as well in the Deep Neural Nets training. Here are the steps needed for creating train/test/validation steps for Deep Neural Nets. The feature extraction process is in-built with the State-of-Art Neural Net that we are planning to use.

1. Shuffle and split the train set into 80/20 train, validation set.
2. get the image files and data for Train/Test/Valid Set
3. convert the files into pixel features and create bottleneck VGG16 Tensors
4. convert the emotion label into one-hot encoded features

The non-uniformity of image distribution among different emotions could not be resolved as the data itself is very constrained. The papers I read had defined ways in extracting the emotions via peak frames. If I have to introduce any data from outside the dataset, I would have to go against the methodology of Cohn-Kanade data collection. The final test set had the following distribution of emotions.



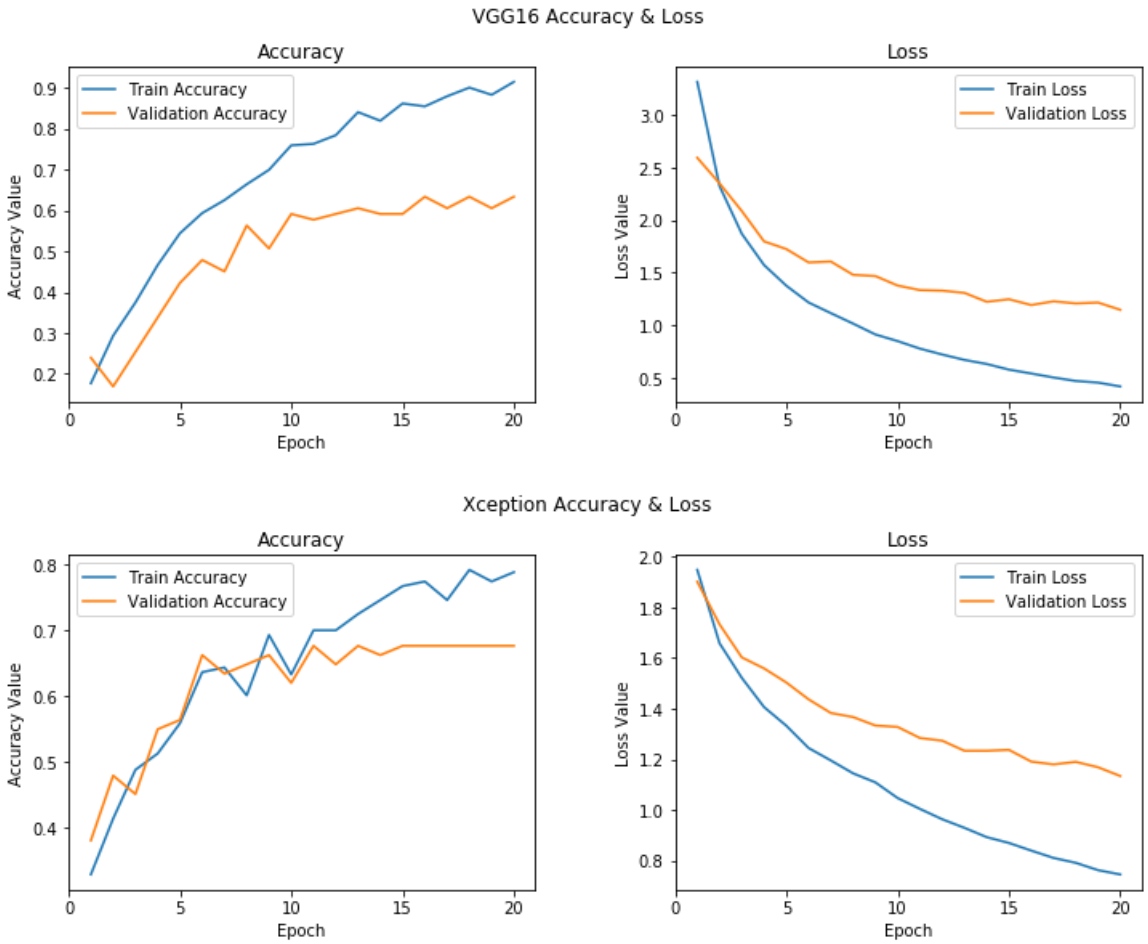

```
emotion
0    23
1     9
2     3
3    11
4     5
5    13
6     5
7    16
dtype: int64
```

Implementation

After obtaining train test and valid sets, I used the APIs defined in `SKLearn` and `Keras` to implement the algorithms with its parameters. I have clearly defined the algorithms that were implemented in the project in the `Algorithms and Techniques` section.

Refinement

Observations from implementations of Transfer Learning



1. In both VGG16 and Xception Transfer learning experiments we saw that the Training Accuracy and Training loss imporves drastically. But the Validation accuracy and losses suffer reaching a plateau in a few iterations.
2. The deep models has not got enough data to express the right emotions.

So my next logical step was to augument the images and feed to the model.

Image Augumentation

The Neural network agents that learn the emotions behind scenes only learn the pixel. These pixels are basically RGB inputs.Our underlying problem of less data to learn can be solved by introducing different versions of the same images transforming their scale, position and rotations of the image.This gives more concrete representations among each emotion and might increase the accuracy of the transfer learned network. Our goal is to see if this avoid overfitting.

If seen above we have non-uniform data for different emotion classes and it is limited. To take care of this problem, we do data augmentation by rotating, shifting and flipping and more such combinations to get more data with the already existing one. Majorly rotation, width and height shift, and horizontal flip will be required. This cannot solve the problem of non-uniformity completely as the data difference is large in this case.

I have employed the following transformations:

1. featurewise_center = `False` - Do not set input mean to 0 over the dataset
2. samplewise_center = `False` - Do not set each sample mean to 0
3. featurewise_std_normalization = `False` - Do not divide inputs by std of the dataset, feature-wise.
4. samplewise_std_normalization = `False` - Do not divide each input by its std.
5. zca_whitening = `False` - Do not do ZCA whitening. Learn more about whitening transformations here.
6. rotation_range = `50` - Rotate images upto 50 degrees
7. width_shift_range = `0.2` - Shift width by 20 %
8. height_shift_range = `0.2` - Shift height by 20 %
9. horizontal_flip = `True` - Flip the image horizontally randomly
10. vertical_flip = `False` - Do not Flip the images vertically
11. fill_mode = `nearest` - fill in new pixels for images after we apply any of the preceding operations (especially rotation or translation). In this case, we just fill in the new pixels with their nearest surrounding pixel values.

I will create two `datagen.flow` objects. One for train and one for validation. The train flow will have a batch of 32 inputs trained and learned using backpropagation. The valid set is used to get validation loss and accuracy after training the model from each batch.

Image augumented VGG16 model components
There are total of 6 components to our image augumented VGG16 model.

1. VGG16 model itself. Our Input to the VGG16 model will be a image of (resized length, resized width, # of channels). We will not take the last layer of the VGG16 model. We will make sure none of the layers of our VGG16 model is trained. We will also make sure that we pass down the output of the VGG16 model as input to the lower layers of our image augumented custom VGG16 model.
2. Global Average Pooling layer 2D which will take the 512 max values from the output of the VGG16 model.
3. A dense layer which will result 8 different class output.
4. Compile the model with loss function `categorical_crossentrophy` and gradient optimizer as `Adam`.Also we are interested in `Accuracy` metric.
5. We will develop 3 callback functions. `ModelCheckpoint` will save only the model with best validation loss. `ReduceLROnPlateau` will automatically reduce the learning rate of the gradient optimization function by a percentage if it sees a plateau like validation loss for last 5 epochs. `EarlyStopping` will stop the training if it sees no imporvement in validation loss for the last 3 continuous epochs. Which infact makes `ReduceLROnPlateau` dummy though.
6. Steps per epoch and validation steps detemine how many batches in a epoch needs to be trained and validated respectively. Its generally a proportion of the total # of samples / batch size

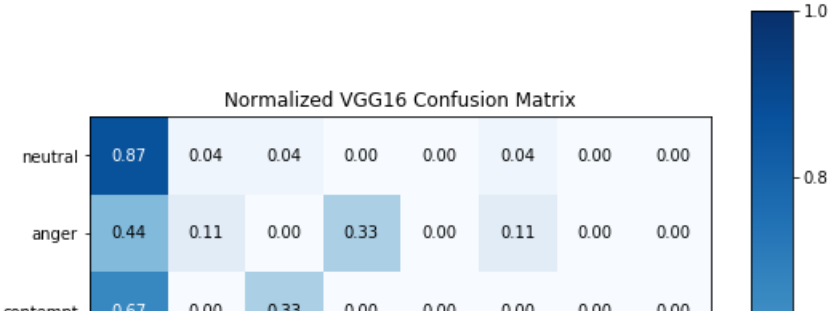
But!

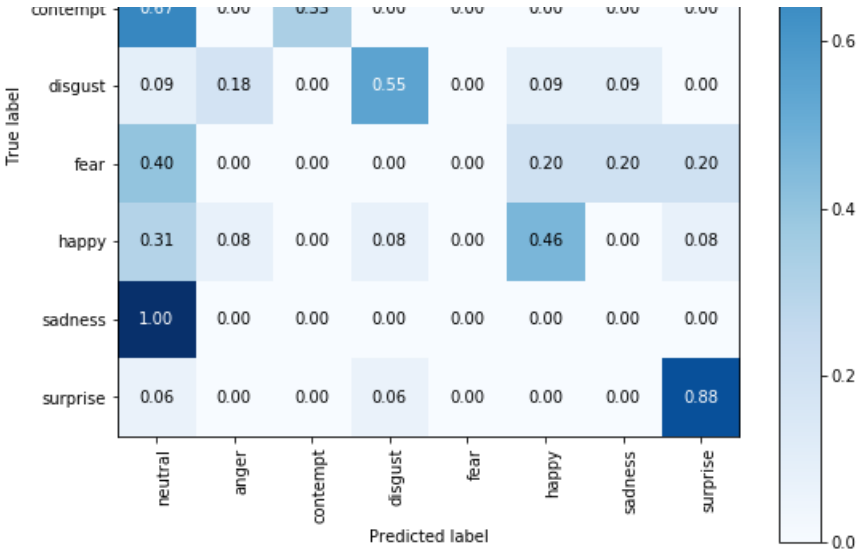
The process could not be improved. It only got worse. I realized my mistakes later on after completing the project. I had no time to do them so I will document them as part of improvements.

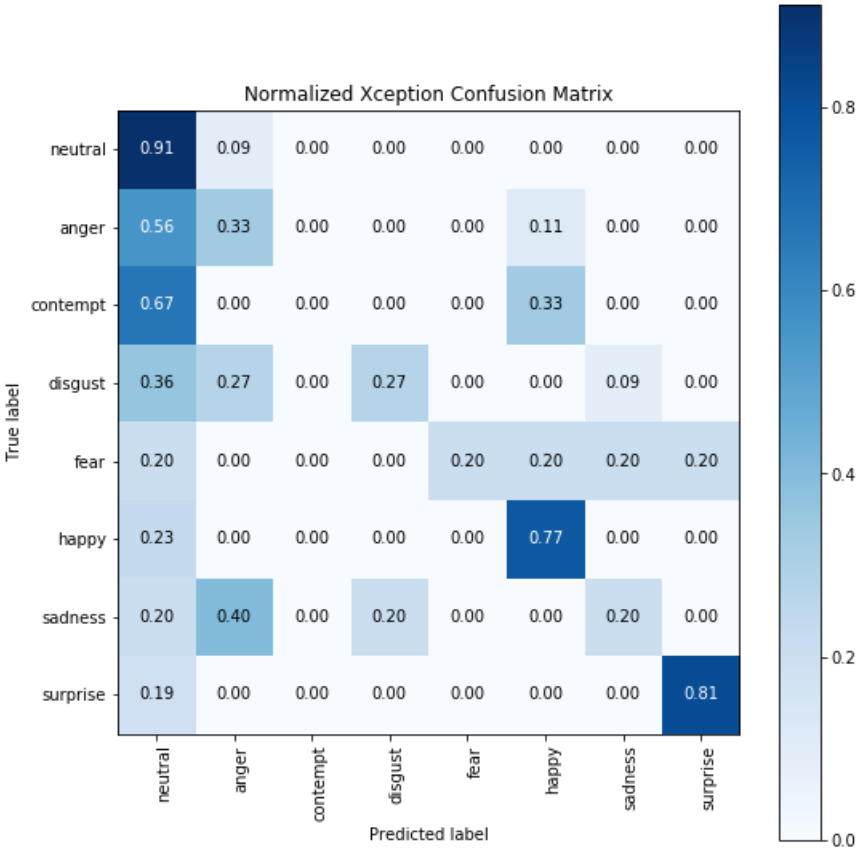
IV. Results

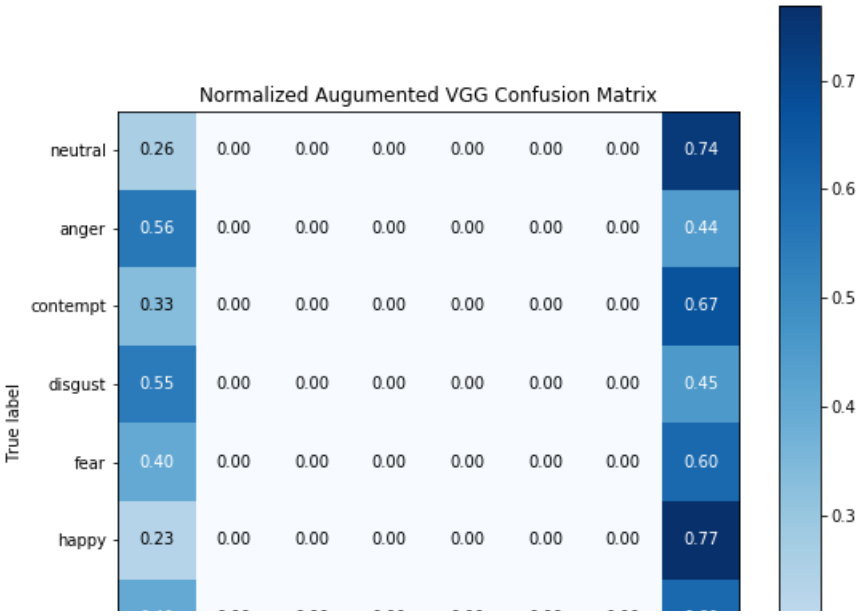
Model Evaluation and Validation

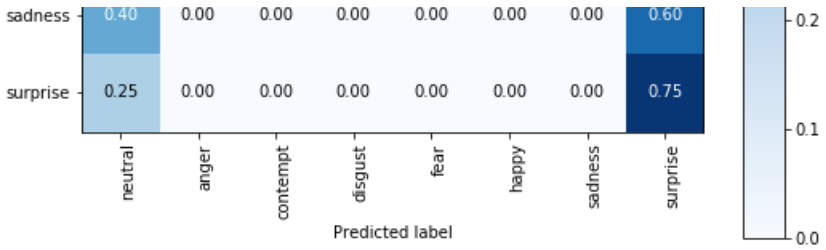
Algorithm	Cross Entropy Loss - Train	Cross Entropy Loss - Valid	Cross Entropy Loss - Test	Accuracy - Train	Accuracy - Valid	Accuracy - Test
Transfer Learned Xception	0.74	1.13	1.14	0.79	0.68	0.61
Transfer Learned VGG16	0.42	1.15	1.27	0.91	0.63	0.56
Data Augumented VGG16	1.71	1.83	1.78	0.25	0.24	0.21




contempt
disgust 0.09 0.18 0.00 0.55 0.00 0.09 0.09 0.00
fear 0.40 0.00 0.00 0.00 0.00 0.20 0.20 0.20
happy 0.31 0.08 0.00 0.08 0.00 0.46 0.00 0.08
sadness 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
surprise 0.06 0.00 0.00 0.06 0.00 0.00 0.00 0.88
True label
neutral anger contempt disgust fear happy sadness surprise
Predicted label
0.6
0.4
0.2
0.0


Normalized Xception Confusion Matrix
neutral 0.91 0.09 0.00 0.00 0.00 0.00 0.00 0.00
anger 0.56 0.33 0.00 0.00 0.00 0.11 0.00 0.00
contempt 0.67 0.00 0.00 0.00 0.00 0.33 0.00 0.00
disgust 0.36 0.27 0.00 0.27 0.00 0.00 0.09 0.00
fear 0.20 0.00 0.00 0.00 0.20 0.20 0.20 0.20
happy 0.23 0.00 0.00 0.00 0.00 0.77 0.00 0.00
sadness 0.20 0.40 0.00 0.20 0.00 0.00 0.20 0.00
surprise 0.19 0.00 0.00 0.00 0.00 0.00 0.00 0.81
True label
neutral anger contempt disgust fear happy sadness surprise
Predicted label
0.8
0.6
0.4
0.2
0.0


Normalized Augumented VGG Confusion Matrix
neutral 0.26 0.00 0.00 0.00 0.00 0.00 0.00 0.74
anger 0.56 0.00 0.00 0.00 0.00 0.00 0.00 0.44
contempt 0.33 0.00 0.00 0.00 0.00 0.00 0.00 0.67
disgust 0.55 0.00 0.00 0.00 0.00 0.00 0.00 0.45
fear 0.40 0.00 0.00 0.00 0.00 0.00 0.00 0.60
happy 0.23 0.00 0.00 0.00 0.00 0.00 0.00 0.77
True label
0.7
0.6
0.5
0.4
0.3



After several iterations of the parameters in the models, I present the output above. From the given output we can clearly see that none of the tried models is resonable. They have barely learnt emotion other than the skewed Neutral emotion in the hold out test dataset. Accuracy has been a very bad metric to put into evaluation. We can see from the confusion matrix that the models have barely learnt anything. Still due to its non-uniformity, Accuracy has been upto 68%. Data Augumentation has performed horribly. Augumentations to data has greatly affected the loss performance. The models above cannot be trusted for a production usecase. I could not decide on the final model based on the performance results I got.

Justification

I declare that my Deep Neural Nets models did not do better than the Baseline SVM models. There are several reasons behind it. I will capture some here. Before that I want to document the following.

1. There has been no Final result model identified to compare with the benchmark result.
2. I may have not thoroughly analyzed to get to a final solution mostly because I was not aware of ways to navigate to the final solution and partly because I had very little time.
3. No significant solution found to the problem than the baseline model. Which performed decently.

Since I could not decide on a solution to the problem via Deep Learning, I will use this section to document what I could have done in finding a solution using Deep Neural nets. Mostly my take is on the dataset.

1. I used data augumentation to just augument the existing data. But what I later realized while writing the report is that, I did not enhance/add new images to the dataset. Doing that might have got multiple instances of the same emotion and model might have performed well.
2. Choice of dataset and constraining myself to taking the peak frames for emotions made me drop many images. I could have labeled last few peak frames to the emotion label rather than naming the penultimate frame the emotion.
3. I could have pivoted to another dataset. The methodology holds valid for any dataset. Cohn-Kanade dataset particulary was `acted out` rather than authentically expressed. Acted facial expressions are usually more exaggerated than what we see in the real world. If we were to classify new images with these models, they would most likely underperform. Systems trained on datasets created in a controlled lab setting generally fail to generalize across datasets.

So if I have another chance, I would take a emotions dataset in wild to capture the emotions.

V. Conclusion

Reflection

To summarize the whole pipeline of the project, I started with identifying a posed dataset which had captured 8 different emotions of the subjects. I defined the metrics to compare, explained the feature extraction methods to be applied in the data preprocessing phase. Then carried out Data preparation where I preprocessed the data, resized and extracted relevant features to form train, test and valid sets. I then applied different algorithms and showcased their performances. Later I stumbled upon problems pertaining to less data and reasoned out the probabale ways I could have resolved them. I did not find one good model to solve the problem of Facial Emotional Recognition but I am confident that this pipleline when enhanced can become a decent tool for the problem. The most interesting aspects of this project to me was in getting a historic overview of how heuristic pattern recognition methods have taken place so far. Most difficult aspects were in learning the steps that I have to take in improving the model with less data.

Improvement

There is definitely more work to do to make this project more robust.

1. https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf - This paper talks about One-Shot Neural networks. The basic gist is to do Image/Pattern recognition when we have less data. A perfect fit for our problem.
2. Exploring out our full pipeline with another dataset which has more datapoints.
3. Realtime facial recogition using camera.
4. Achieving high accuracy as well as low cross entropy loss across multiple datasets with several variations of features like pose, illumination etc. This makes our pipeline more generalizable.

References

http://www.paulvangent.com/2016/08/05/emotion-recognition-using-facial-landmarks/ https://github.com/mayurmadnani/fer https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a