Masquerade

(Facial Expression Recognition with Convolutional Neural Networks)

Rudradeep Guha *Vineet Nandkishore* *Rachit Rawat*

Abstract

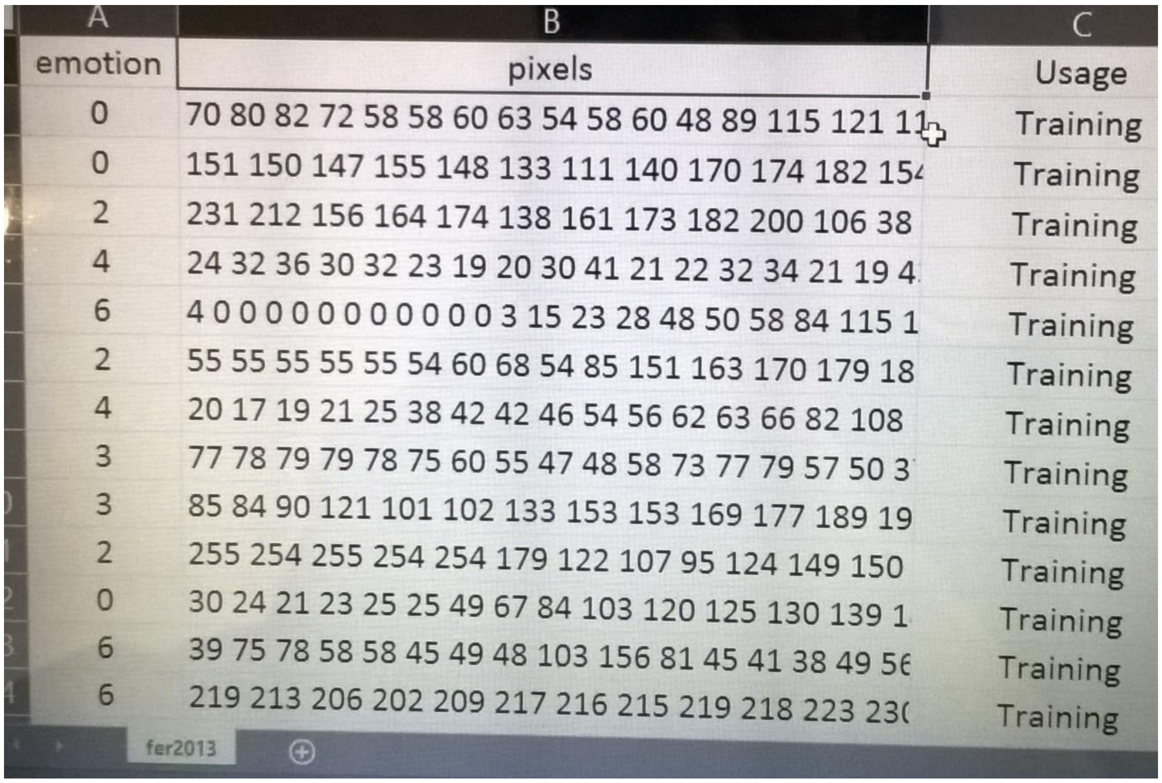
Convolutional Neural Networks are being used to solve increasingly complex problems nowadays. They are preferred over simple neural networks because they take only images as inputs and are specialized to cater to inputs of any image format. For our project, we have used a Convolutional Neural Network to analyze facial expressions of people. The expressions are classified into one of 7 classes by the CNN. The data was obtained from the Kaggle website, which had facial expression recognition as a challenge. We had the option of using a pre-trained network like AlexNet or VGGNet, but we chose to create and train a network from scratch. After experimenting with various configurations, changing the number of layers, optimizers and activation functions used, we managed to get a final training accuracy of 80% on the dataset.

Introduction

Humans are said to have 7 basic emotions: neutral, happy, sad, anger, disgust, fear and surprise. These emotions are usually conveyed through changes in facial expressions. Thus, facial expressions are crucial in recognizing the state of mind of a person. It is so important that humans acquire the ability to differentiate between facial expressions at 14 months old. Deep learning has been used to a great extent to improve communication between machines and humans and getting machines to understand human forms of communication. However, work had been more focused on recognizing speech, or generate speech within context of a conversation. Identifying emotions is one of the most basic forms of communication between humans and thus we are training a machine to learn how to identify expressions correctly.

Dataset

The data was obtained from the Kaggle website in the form of a csv file. Emotions are encoded as numbers-for example, neutral is 6, happy is 2 and so on. The csv file contains images and the emotion represented by the faces in the image coded into an integer between 0 and 6. The image itself is stored in the form of a string of pixels.



A	B	C
emotion	pixels	Usage
0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 1	Training
0	151 150 147 155 148 133 111 140 170 174 182 15	Training
2	231 212 156 164 174 138 161 173 182 200 106 38	Training
4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 19 4	Training
6	4 0 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84 115 1	Training
2	55 55 55 55 55 54 60 68 54 85 151 163 170 179 18	Training
4	20 17 19 21 25 38 42 42 46 54 56 62 63 66 82 108	Training
3	77 78 79 79 78 75 60 55 47 48 58 73 77 79 57 50 3	Training
3	85 84 90 121 101 102 133 153 153 169 177 189 19	Training
2	255 254 255 254 254 179 122 107 95 124 149 150	Training
0	30 24 21 23 25 25 49 67 84 103 120 125 130 139 1	Training
6	39 75 78 58 58 45 49 48 103 156 81 45 41 38 49 5	Training
6	219 213 206 202 209 217 216 215 219 218 223 23	Training

fer2013

This string of pixels contains all the information in the image and reconstructing an image from its pixel information returns a 48x48 image in grayscale. The images are cropped and centered so that the face appears as close to the center of the image as possible. There are a total of 35887 images, out of which 28709 are for training and 7178 for testing. However, we found that there are comparatively very few images of faces showing disgust. This of course later affected the ability of the CNN to predict a disgusted expression.

Model Architecture and Training

We decided to build a model from scratch instead of fine-tuning a pre-trained network like AlexNet and VGGNet. The training data was extracted from the csv file and stored in a numpy array of shape (28709, 1, 48, 48). The test data is stored in a different numpy array of shape (3589, 1, 48, 48). The model uses the Keras library. A Convolutional Network usually has a standardized structure with an input layer, several convolutional layers, some fully connected layers and a final output layer.

Our Convolutional Neural Network is 13 layers, with 10 convolutional layers and 3 dense/fully connected layers.

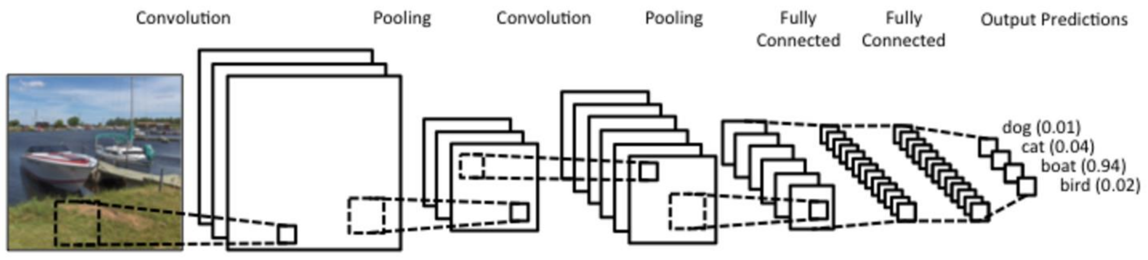
Input (48x48 grayscale images)
Convolution2D - 3 x 3 x 32 | ReLU | BatchNormalization
Convolution2D - 3 x 3 x 32 | ReLU | BatchNormalization
MaxPooling (2, 2)
Convolution2D - 3 x 3 x 64 | ReLU | BatchNormalization
Convolution2D - 3 x 3 x 64 | ReLU | BatchNormalization
MaxPooling (2, 2) | Dropout (0.25)
Convolution2D - 3 x 3 x 128 | ReLU | BatchNormalization
Convolution2D - 3 x 3 x 128 | ReLU | BatchNormalization
MaxPooling (2, 2) | Dropout (0.3)
Convolution2D - 3 x 3 x 256 | ReLU | BatchNormalization
Convolution2D - 3 x 3 x 256 | ReLU | BatchNormalization
MaxPooling (2, 2) | Dropout (0.3)
Convolution2D - 3 x 3 x 512 | ReLU | BatchNormalization
Convolution2D - 3 x 3 x 512 | ReLU | BatchNormalization
MaxPooling (2, 2) | Dropout (0.3)
Fully Connected – 1024 | ReLU | BatchNormalization | Dropout (0.5)
Fully Connected – 1024 | ReLU | BatchNormalization | Dropout (0.5)
Fully Connected – 7 | 'softmax'

The input layer only accepts images that have some fixed dimensions, which is why we stored the training data in a numpy array which can be reshaped into the desired shape. Since we are taking pictures using the webcam in a laptop, the images that we input to the model do not possess the required characteristics. To modify the images, we use the openface library as well as OpenCV. An image is taken as an argument for a python script which implements the openface library to detect the face in the image and then crops the image around the face so that the face is in the center and there is no surrounding noise. The face detected in the image is also aligned such that even if the face is, say angled somewhat sideways, it is straightened by the openface module. The OpenCV module converts the image to grayscale and resizes it to 48x48. Thus, the image dimensions are changed from (3, 48, 48) to (1, 48, 48), which is the input shape required by the first layer of our network.

The Convolution2D layers take the image stored in the numpy array and use filters to scan the entire image. Each filter covers an area of (3, 3) at one time, and scans the image and is responsible for the creation of one feature map. We set the number of filters in the input layer to be 32 and doubling after every two layers.

Since the convolutional layers keep adding up, it increases the computation time. The MaxPooling layer is used to reduce the dimensionality of the input. It does this by covering an area of (2, 2) and going through the whole image and choosing only the maximum pixel values in that field. We used a MaxPooling layer after every two convolution layers.

The final output layer returns an array of values, with the index of each value representing the class according to the emotion encoding. Since there are 7 classes, the final dense fully connected layer outputs an array of dimension 7. We are using 'softmax' as the activation function because we want the probability of an image belonging to each class. Finally, the network was trained for 300 epochs with the optimizer 'adam'. The chance of overfitting was reduced through the use of BatchNormalization and Dropout after every layer and every two layers respectively.

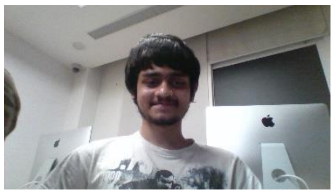


(A rough idea of the structure of our network just with different inputs and outputs)

Alternatives Used: We changed the optimizer to SGD with a learning rate of 0.01, but that gave very sub-optimal results. So, we switched back to 'adam'.

Results and Conclusion

Upon completion of training, the training accuracy was 78% while validation accuracy was just 52%. Calculation of test metrics of the model showed the test accuracy to be 48%. Data augmentation actually resulted in an almost stagnant accuracy of 35.5% after 50 or so epochs, while also taking significantly longer to train. So, it was dropped. Initially, only an 11-layer network was implemented, but it gave only 63% training accuracy (although we ran it for fewer epochs). This 13-layer model gave the best results among the model variations we tried.


Prediction: Happy

Prediction: Neutral

Prediction: Neutral

Second Prediction: Sad

Prediction: Happy

Prediction: Fear

Second Prediction: Surprise

Prediction: Angry

The results of our model could have been better, but not by much (the highest test accuracy ever recorded is close to 65%.) We probably could have achieved greater accuracy by fine-tuning a pre-trained network but we wanted to try making a model from ground up. Although time and computation power restrictions may also have caused slightly lesser accuracy, this score is reasonable because predicting emotions is difficult. There is a lot of difference in the way most people express the same emotions, and accurately predicting the emotions of people based on just their expressions is not a very common ability even in humans. Most of the times there is no single emotion expressed by a person. This lead to an interesting discovery that often the class with the second highest probability was the correct emotion. We then utilized this finding in the prediction part of the code. If the network predicts the emotion incorrectly the first time, it tries again, this time returning the class with the second highest probability.

Model	Training Accuracy	Test Accuracy
11-layer CNN	63%	37%
13-layer CNN (with data augmentation)	68.88%	35.5%
13-layer CNN	78%	48%

Thus, a score in this range is very reasonable for a machine trying to predict emotions of humans based on just their facial expression.