Emotion Classifier

April 15, 2018

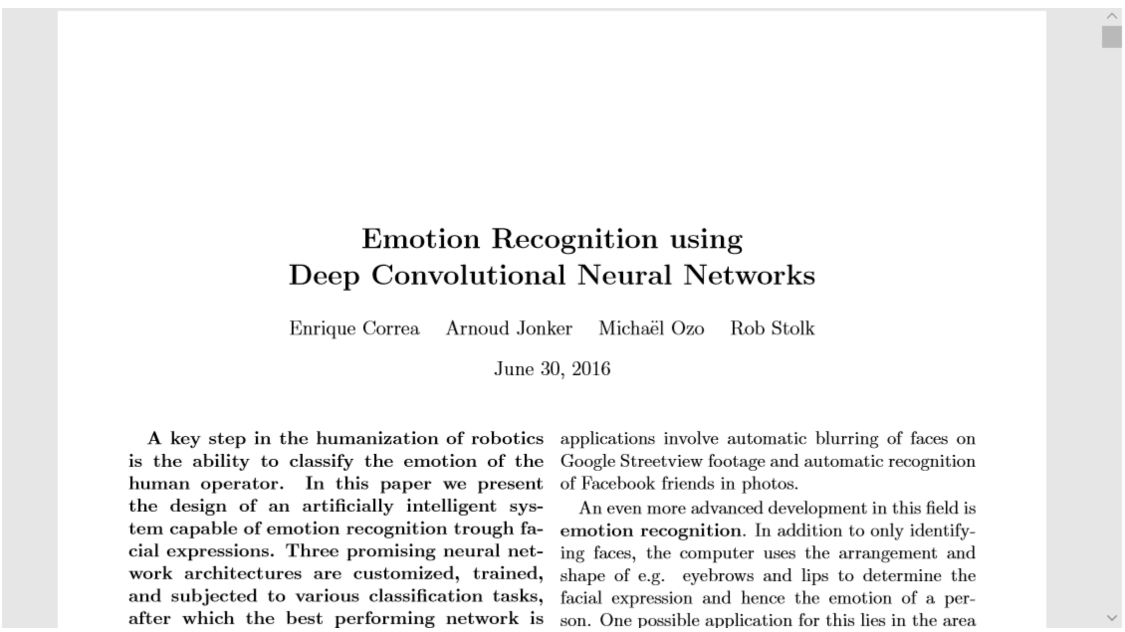
1 Emotion Classification using Deep learning Network

(Anu Priya,Prabhat Johl, Parth Soni, Rohit Pathak)

1.0.1 In this project our objective is to find how can we use Artificial neural network to interpret the facial expression's in human

1.0.2 References:

```
In [4]: from PIL import Image
        path = "MLProject/paper1.PNG"
        display(Image.open(path))
```



Emotion Recognition using
Deep Convolutional Neural Networks

Enrique Correa Arnoud Jonker Michaël Ozo Rob Stolk

June 30, 2016

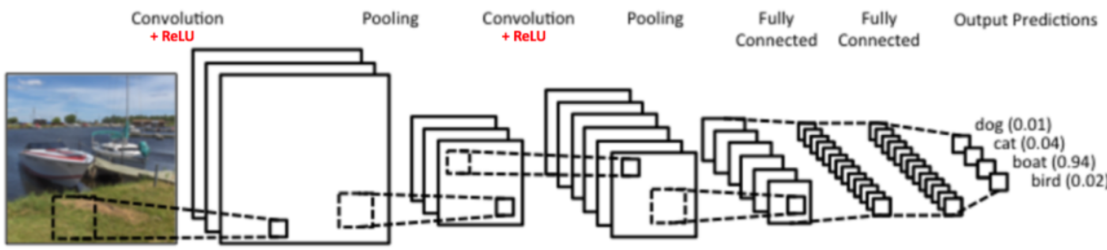
A key step in the humanization of robotics is the ability to classify the emotion of the human operator. In this paper we present the design of an artificially intelligent system capable of emotion recognition trough facial expressions. Three promising neural network architectures are customized, trained, and subjected to various classification tasks, after which the best performing network is

applications involve automatic blurring of faces on Google Streetview footage and automatic recognition of Facebook friends in photos.

An even more advanced development in this field is **emotion recognition**. In addition to only identifying faces, the computer uses the arrangement and shape of e.g. eyebrows and lips to determine the facial expression and hence the emotion of a person. One possible application for this lies in the area

1.0.3 How does the Network look like

```
In [2]: from PIL import Image
        path = "MLProject/ConvNet.PNG"
        display(Image.open(path))
```



1.0.4 Dataset:

We use the Cohn Kanade Image Database, a set of 30,000 pictures of people displaying 6 emotional expressions (angry, fear, happy, sad, surprised and neutral)

The networks are programmed with use of the TFLearn library on top of TensorFlow, running on Python. This environment lowers the complexity of the code, since only the neuron layers have to be created, instead of every neuron

```
In [ ]: #we begin by importing our libraries
        import tflearn
        from tflearn.layers.core import input_data, dropout, fully_connected, flatten
        from tflearn.layers.conv import conv_2d, max_pool_2d, avg_pool_2d
        from tflearn.layers.merge_ops import merge
        from tflearn.layers.normalization import local_response_normalization
        from tflearn.layers.estimator import regression
        from constants import *
        from os.path import isfile, join
```

```
In [ ]: # Next we construct our layers for Convolution Neural Network

        self.network = conv_2d(self.network, 64, 5, activation = 'relu')
        #self.network = local_response_normalization(self.network)
        self.network = max_pool_2d(self.network, 3, strides = 2)
        self.network = conv_2d(self.network, 64, 5, activation = 'relu')
        self.network = max_pool_2d(self.network, 3, strides = 2)
        self.network = conv_2d(self.network, 128, 4, activation = 'relu')

        self.network = fully_connected(self.network, 3072, activation = 'relu')
```

```
In [ ]: # followed by the layers we begin training of the images
        def start_training(self):
            self.load_saved_dataset()
            self.build_network()
            if self.dataset is None:
              self.load_saved_dataset()
            # Training
            print('[+] Training network')
```

```
            self.model.fit(
              self.dataset.images, self.dataset.labels,
              validation_set = (self.dataset.images_test, self.dataset._labels_test),
              n_epoch = 100,
              batch_size = 50,
              shuffle = True,
              show_metric = True,
              snapshot_step = 200,
              snapshot_epoch = True,
              run_id = 'emotion_recognition'
            )
```

1.0.5 Cost Function

We used simple softmax cost entropy function as our loss function

```
In [4]: from PIL import Image
        path = "MLProject/CrossEntropy.PNG"
        display(Image.open(path))
```

$$J(T,O) = -\frac{1}{N}\sum_{n=1}^{N}\left[t_n \ln(o_n) + (1-t_n)\ln(1-o_n)\right]$$

Where N is the size of the batch, (T1...Tn) are the input values and (O1...On) are the output values

1.0.6 Results

```
In [9]: from PIL import Image
        path = "MLProject/happy.PNG"
        display(Image.open(path))

        from PIL import Image
        path = "MLProject/surprise.PNG"
        display(Image.open(path))


        from PIL import Image
        path = "MLProject/neutral.PNG"
        display(Image.open(path))


        from IPython.display import HTML, display
        import tabulate
        table = [["Training Accuracy",'79%'],
```

```
        ["Testing Accuracy",'74%']]
display(HTML(tabulate.tabulate(table, tablefmt='html')))
```

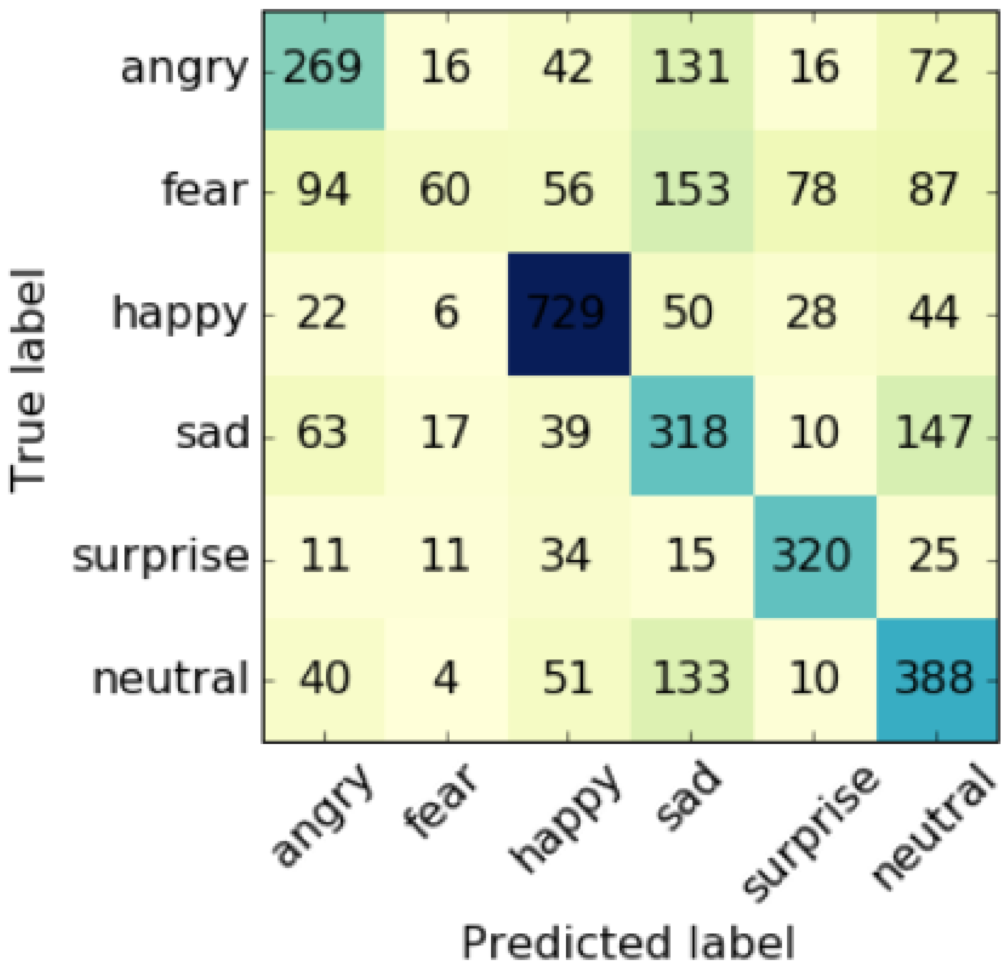
<IPython.core.display.HTML object>

```
In [ ]: ## Predict function

        import tensorflow as tf
        import numpy as np
        import os,glob,cv2
        import sys,argparse


        # First, pass the path of the image
        dir_path = os.path.dirname(os.path.realpath(__file__))
        image_path=sys.argv[1]
        filename = dir_path +'/' +image_path
        image_size=128
        num_channels=3
        images = []
        # Reading the image using OpenCV
        image = cv2.imread(filename)
        # Resizing the image to our desired size and preprocessing will be done exactly as don
        image = cv2.resize(image, (image_size, image_size),0,0, cv2.INTER_LINEAR)
        images.append(image)
        images = np.array(images, dtype=np.uint8)
        images = images.astype('float32')
        images = np.multiply(images, 1.0/255.0)
        #The input to the network is of shape [None image_size image_size num_channels]. Hence
```

```
x_batch = images.reshape(1, image_size,image_size,num_channels)

## Let us restore the saved model
sess = tf.Session()
# Step-1: Recreate the network graph. At this step only graph is created.
saver = tf.train.import_meta_graph('emotions-model.meta')
# Step-2: Now let's load the weights saved using the restore method.
saver.restore(sess, tf.train.latest_checkpoint('./'))

# Accessing the default graph which we have restored
graph = tf.get_default_graph()

# Now, let's get hold of the op that we can be processed to get the output.
# In the original network y_pred is the tensor that is the prediction of the network
y_pred = graph.get_tensor_by_name("y_pred:0")

## Let's feed the images to the input placeholders
x= graph.get_tensor_by_name("x:0")
y_true = graph.get_tensor_by_name("y_true:0")
y_test_images = np.zeros((1, 2))


### Creating the feed_dict that is required to be fed to calculate y_pred
feed_dict_testing = {x: x_batch, y_true: y_test_images}
result=sess.run(y_pred, feed_dict=feed_dict_testing)
# result is of this format [probabiliy_of_rose probability_of_sunflower]
print(result)
```

1.0.7 Confusion Matrix

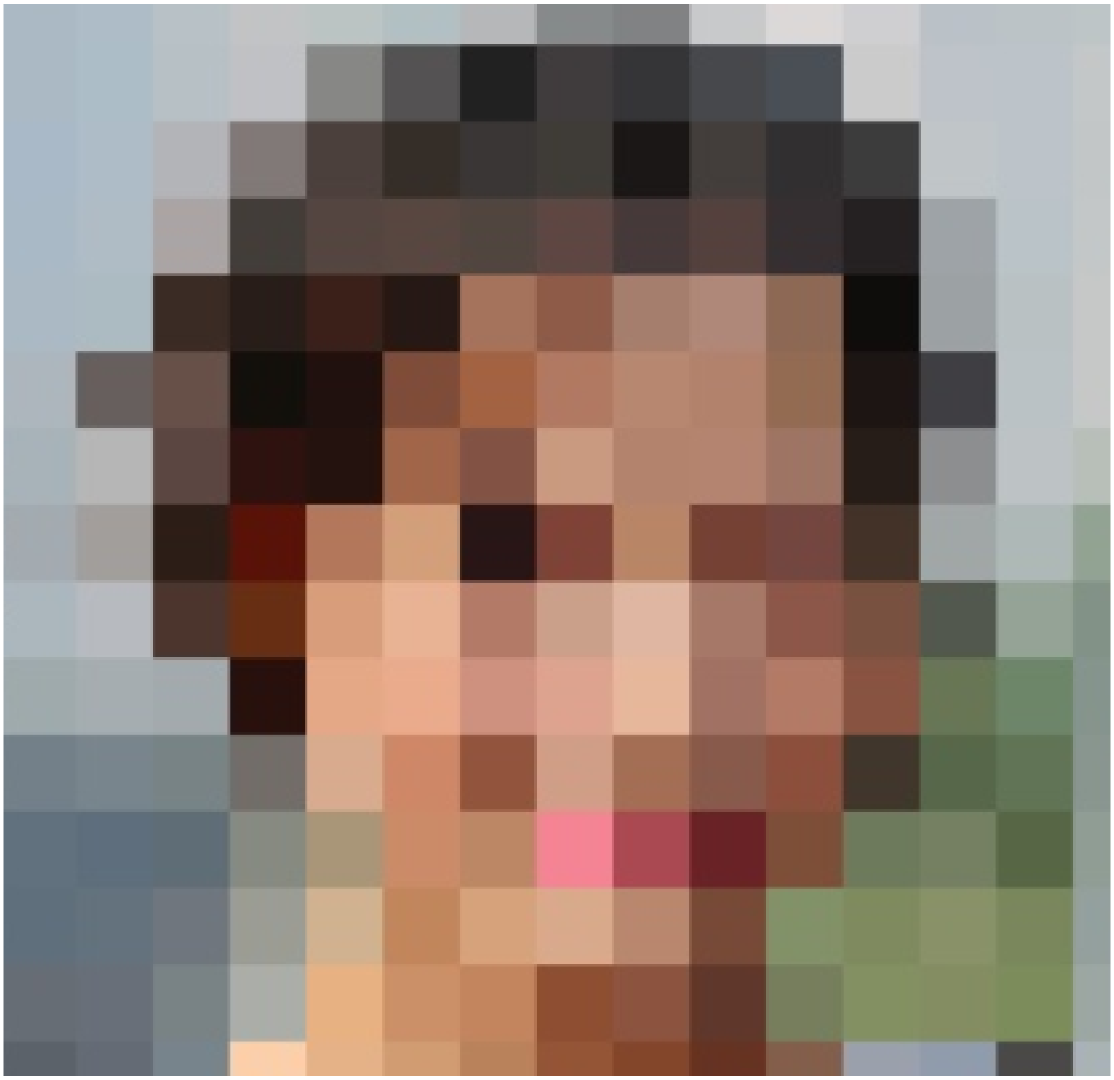
```
In [6]: from PIL import Image
        path = "MLProject/confusionMatrix.PNG"
        display(Image.open(path))
```



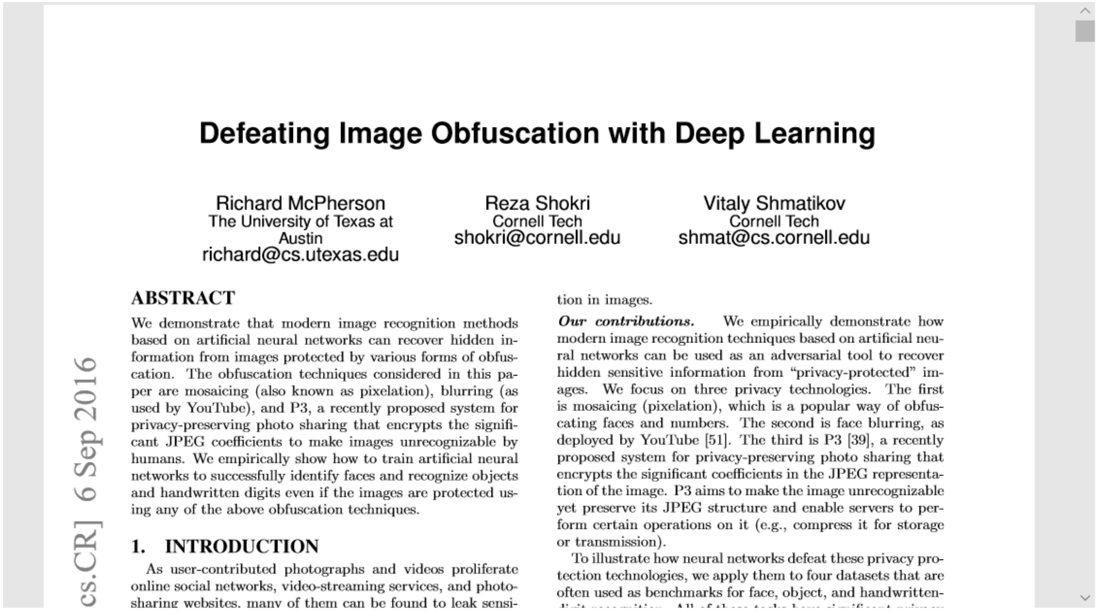
This is the confusion matrix on test set . For "happy" the precision is highest (around 77%) and for "sad" its 40%

This is simple multi class classification of images that expresses emotion. To make it more interesting we changed some input for our Convolutional Neural Network. Instead of an actual image we gave our network an redacted image

```
In [7]: from PIL import Image
        path = "MLProject/redacted_image.PNG"
        display(Image.open(path))
```



In [3]:
```
from PIL import Image
path = "MLProject/paper2.PNG"
display(Image.open(path))
```



cs.CR] 6 Sep 2016

Defeating Image Obfuscation with Deep Learning

Richard McPherson
The University of Texas at Austin
richard@cs.utexas.edu

Reza Shokri
Cornell Tech
shokri@cornell.edu

Vitaly Shmatikov
Cornell Tech
shmat@cs.cornell.edu

ABSTRACT

We demonstrate that modern image recognition methods based on artificial neural networks can recover hidden information from images protected by various forms of obfuscation. The obfuscation techniques considered in this paper are mosaicing (also known as pixelation), blurring (as used by YouTube), and P3, a recently proposed system for privacy-preserving photo sharing that encrypts the significant JPEG coefficients to make images unrecognizable by humans. We empirically show how to train artificial neural networks to successfully identify faces and recognize objects and handwritten digits even if the images are protected using any of the above obfuscation techniques.

1. INTRODUCTION

As user-contributed photographs and videos proliferate online social networks, video-streaming services, and photo-sharing websites, many of them can be found to leak sensi-

tion in images.

Our contributions. We empirically demonstrate how modern image recognition techniques based on artificial neural networks can be used as an adversarial tool to recover hidden sensitive information from "privacy-protected" images. We focus on three privacy technologies. The first is mosaicing (pixelation), which is a popular way of obfuscating faces and numbers. The second is face blurring, as deployed by YouTube [51]. The third is P3 [39], a recently proposed system for privacy-preserving photo sharing that encrypts the significant coefficients in the JPEG representation of the image. P3 aims to make the image unrecognizable yet preserve its JPEG structure and enable servers to perform certain operations on it (e.g., compress it for storage or transmission).

To illustrate how neural networks defeat these privacy protection technologies, we apply them to four datasets that are often used as benchmarks for face, object, and handwritten-

The accuracy of our neural network on the original dataset is 65%. .Once again, a more sophisti- cated network architecture could likely achieve much better results, but our experiments show that even a simple network can defeat the image obfuscation techniques.

```
In [9]: from PIL import Image
        path = "MLProject/redact.PNG"
        display(Image.open(path))
```

1.0.8 Cost Function

```
In [10]: from PIL import Image
         path = "MLProject/mean_error.PNG"
         display(Image.open(path))
```

$$\text{ME} = \frac{\sum_{i=1}^{n} y_i - x_i}{n}.$$

For the above picture the error is 2.4%. i.e the network was abled to recover approximately 96% of the original image.

We then take this predicted dataset and ran our original Emotional classification.

```
In [12]: from IPython.display import HTML, display
         import tabulate
         table = [["Training Accuracy",'65%'],
                  ["Testing Accuracy",'52%']]
         display(HTML(tabulate.tabulate(table, tablefmt='html')))
```

```
<IPython.core.display.HTML object>
```

1.0.9 Where can we use this kind of model

Recognizing digits can help infer the contents of written text or license plate numbers. It turns out extracting original image from redacted image can break privacy protection technologies. You can also do the same for speech synthesis but that would require a more complex model.

```
In [ ]: #https://en.wikipedia.org/wiki/Convolutional_neural_network
        #http://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-im
```