Machine Learning Engineer Nanodegree
Capstone Project

Egor Ermilov
2017-06-28

I. Definition

Project Overview

Identifying, extracting and quantifying affective states and subjective information such as human emotions is known as sentiment analysis. It determines the attitude of a subject, his/her emotional reaction to an item, interaction or event.

Sentiment analysis could be applied in marketing to better understand customers' demands and needs, based on their reactions, discussions of a particular item, event or service.

Why do we need sentiment analysis nowadays? Because most of the customers don't have enough time to evaluate the quality of the services/items and to share their opinion explicitly by rating something or leaving a review. Instead they share their opinion implicitly – discuss it on the Internet, talk to friends, express it on the face, etc.

Therefore businesses often don't get valuable feedback and know only a little about how to improve their services. Sentiment analysis allows us to extract this valuable information without even asking the customers to do something extra.

There are several sources of such information: text (forum, social media discussions), voice (talking with colleagues, friends) and facial images (the immediate reaction on something).

The most used and developed area here is text sentiment analysis. Probably because it has the easiest way of acquiring the information – scraping forums, social networks, comments sections on the Internet.

In this project we'll try to apply sentiment analysis to facial images to recognize human emotions. Luckily for us there is a good dataset available on Kaggle:

https://inclass.kaggle.com/c/facial-keypoints-detector/data

Problem Statement

The goal of this project is to build a model to recognize human emotions. The model should be able to recognize emotions by a given human face photo with high accuracy. It makes a classification problem with face photos as inputs and emotion types as outputs.

Our model would be a deep neural network with convolutional layers. Deep Neural Networks are very efficient at finding difficult patterns in the data (including image recognition). Adding a convolutional layer makes a huge impact on the models and their abilities to identify objects on an image.

It should also be able to use not only the test photos from the original dataset, but also any custom photo uploaded. When a custom photo is uploaded the model will define how close is the expression on the face to the following emotions: anger, disgust, fear, happy, sad, surprise, neutral, and will return the most suitable one.

Metrics

If the classes in the dataset are unbalanced (the number of items in any class is much higher than in other classes) we would use the following metrics to evaluate the model:

– precision (the ratio of correctly predicted positive observations to the total predicted positive observations)
– recall (the ratio of correctly predicted positive observations to the all observations in actual class)

If the classes are balanced or we would be able to balance them the following metric would be sufficient:
– accuracy (the ratio of correctly predicted observation to the total observations)

If the classes are unbalanced we could see an accuracy paradox. For example when the dataset contains 90% items of one particular class, the classifier could naively predict that all the items belong to that class. It will give as the accuracy of 90%, which at the first glance is very impressive, but in fact very meaningless.

We can calculate all the metrics with with following formulas:

Accuracy = (TP + TN) / (TP + FP + FN + TN)
Precision = TP / (TP + FP)
Recall = TP / (TP + FN)

Where:
TP (True Positives) – number of positive examples, labeled as such.
TN (True Negatives) – number of negative examples, labeled as such.
FP (False Positives) – number of negative examples, labeled as positive.
FN (False Negatives) – number of positive examples, labeled as negative.

Since we have a classification problem, all those metrics would be perfect to evaluate the classifier.

II. Analysis

Data Exploration

This project will use two types of input:

– the dataset from the Emotion and identity detection from face images Kaggle competition (https://inclass.kaggle.com/c/facial-keypoints-detector/data),
– any custom face photo uploaded to recognize emotion type on it.

The Kaggle dataset consists of 3,761 gray-scaled images of 48x48 pixels in size and a 3,761 label set of seven elements each.

Each element encodes an emotional stretch:
0 = anger,
1 = disgust,
2 = fear,
3 = happy,
4 = sad,
5 = surprise,
6 = neutral.

The value counts for the classes:

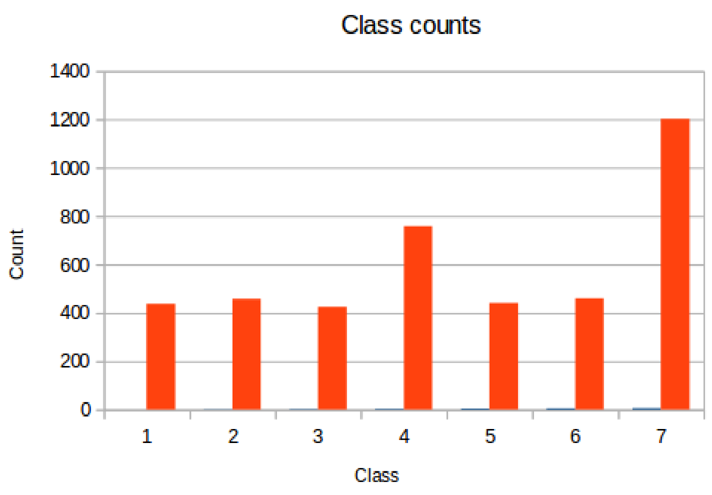
0	437
1	457
2	424
3	758
4	441
5	459
6	1202

Classes 6 and 4 appear much more often than other classes, therefore the classes are unbalanced. There are no missing values or outliers.

Among images there are no anomalies or they are hard to detect (pixel outliers for example).

Exploratory Visualization

The distributing of the original classes is shown here:



We can see that classes 4 and 6 make the distribution not uniform. We will discuss the way of correcting it in a section below.

Here are some face photo examples from the dataset:

Algorithms and Techniques

For this project a convolution neural network will be built and trained on the Kaggle dataset. The dataset will be firstly preprocessed, normalized and split into training, testing and validation datasets. The application will be written in Python and use Tensorflow for building the model.

The convolutional layer is the core building block of our model. The layer's parameters consist of a set of learnable filters, which have a small receptive field, but extend through the full depth of the input volume. During the learning process, each filter is convolved across the width and height of the input volume, producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

To train the model we will use the back propagation algorithm, which repeats a two phase cycle, propagation and weight update. When an input vector is presented to the network, it is propagated forward through the network, layer by layer, until it reaches the output layer. The output of the network is then compared to the desired output, using a loss function, and an error value is calculated for each of the neurons in the output layer. The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which roughly represents its contribution to the original output.

Backpropagation uses these error values to calculate the gradient of the loss function. In the second phase, this gradient is fed to the optimization method, which in turn uses it to update the weights, in an attempt to minimize the loss function.

The approximate architecture of the network would be the following:

– Input layer

– Convolutional layer 1
– Pooling layer 1
– Convolutional layer 2
– Pooling layer 2
– Fully connected layer 1
– Fully connected layer 2
– Output layer

The model will get and 48x48 array of pixels as input and will return a vector with logits for 7 classes.

Benchmark

The benchmark model for this project would be this model:
http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/
which uses a very similar dataset and has an accuracy of 82.5%

However we should also take into account that the winner of the original Kaggle competition had an accuracy of 50%. He/she used exactly the same dataset.

III. Methodology

Data Preprocessing

Since the distribution of the classes does not reflect the real appearance of the emotions among people, we could randomly remove some images in classes 3 and 6 to balance the dataset.

The value counts after this procedure are here:

0	437
1	457
2	424
3	450
4	441
5	459
6	450

Every image in the dataset is a String with integer for pixel values separated by space. We transformed and reshaped them into 48x48 numpy arrays.

The pixel values ranged from 0 to 255, therefore we normalized them to range from 0 to 1 with the following formula:

Normalized_value = (Original_value – Min_in_dataset) / (Max_in_dataset – Min_in_dataset)

In addition the the original dataset preprocessing, we added some preprocessing steps for the custom photo upload. It includes resizing the uploaded photo to 48x48 and grayscaling them.

Implementation

To solve the problem a deep neural network with convolutional layers was built. The network has four major parts: input layer, convolutional layers, fully-connected layers and output layer.

The input layer has the shape of input images and receives the images for the network. The convolutional part consist of 3 convolutional layers with a Relu activation function. The fully-connected part consist of three fully-connected layers with Relu activation function. The output layer has 7 logit neurons (one neuron for every class).

In order to avoid overfitting, regularization is applies to convolutional and fully-connected layers. Convolutional layers use maxpooling and fully-connected layers use dropout.

The back propagation algorithm is applied to train the network. The algorithm uses the gradient descend to reduce the cross entropy of the output logits.

The development process required making changes and adjustments to the model. It included parameter changes, adding similar network layers and repetitive chunks of code. Therefore implementing functional programming (for example writing functions for networks layers) was a good challenge here.

Refinement

The initial solution used the following network architecture (Architecture 1):

Layer	**Operation**
– Input layer (48x48)	
	* Convolution (5x5, s=1, n=32)
– Convolutional layer (48x48x32)	
	* Max pooling (2x2, s=2)
– Pooling layer (24x24x32)	
	* Flattening (6x6x128 = 4608)
– Flat layer (18432x1)	
– Fully connected layer (512x1)	
– Output layer (7x1)	

This initial architecture gave the accuracy of 0.543, which is already good, given the fact that the winner of the original Kaggle competition had the accuracy of 0.5.

After the initial step different parameters has been changed/added:

– Adding maxpooling layers to the convolutional layers and dropout to the fully-connected layers avoids overfitting. The testing accuracy reaches 0.597 (Architecture 2).

– Using sigmoid activation function in the convolutional layers had a bad impact on the model. Testing accuracy = 0.388. Using in in the fully-connected layers didn’t make a big difference (Architecture 3).

– Adding two more convolutional layers increased the accuracy to 0.639, however increased the training time significantly (Architecture 4).

The intermediate solution is here (Architecture 5):

Layer	**Operation**
– Input layer (48x48)	
	* Convolution (7x7, s=2, n=16)
– Convolutional layer 1 (24x24x16)	
	* Max pooling (2x2, s=2)
– Pooling layer 1 (12x12x16)	
	* Convolution (5x5, s=2, n=32)
– Convolutional layer 2 (6x6x32)	
	* Max pooling (2x2, s=2)
Pooling layer 2 (3x3x32)	
	* Flattening (3x3x32 = 288)
– Flat layer (288x1)	
– Fully connected layer 1 (256x1)	
	* Dropout (p=0.7)
– Fully connected layer 2 (128x1)	
	* Dropout (p=0.7)
– Output layer (7x1)	

The accuracy of this model was 0.712, which is high, but not enough compared to the benchmark model.

Finally we have chosen this architecture for the final model (Architecture 6):

Layer	**Operation**
– Input layer (48x48)	
	* Convolution (5x5, s=1, n=16)
– Convolutional layer 1 (48x48x16)	
	* Max pooling (2x2, s=2)
– Pooling layer 1 (24x24x16)	
	* Convolution (5x5, s=1, n=32)

– Convolutional layer 2 (24x24x32)	
	* Max pooling (2x2, s=2)
Pooling layer 2 (12x12x32)	
	* Convolution (5x5, s=1, n=64)
– Convolutional layer 3 (12x12x64)	
	* Max pooling (2x2, s=2)
– Pooling layer 3 (6x6x64)	
	* Flattening (6x6x64 = 2304)
– Flat layer (2304x1)	
– Fully connected layer 1 (512x1)	
	* Dropout (p=0.9)
– Fully connected layer 2 (256x1)	
	* Dropout (p=0.9)
– Fully connected layer 3 (128x1)	
	* Dropout (p=0.9)
– Output layer (7x1)	

IV. Results

Model Evaluation and Validation

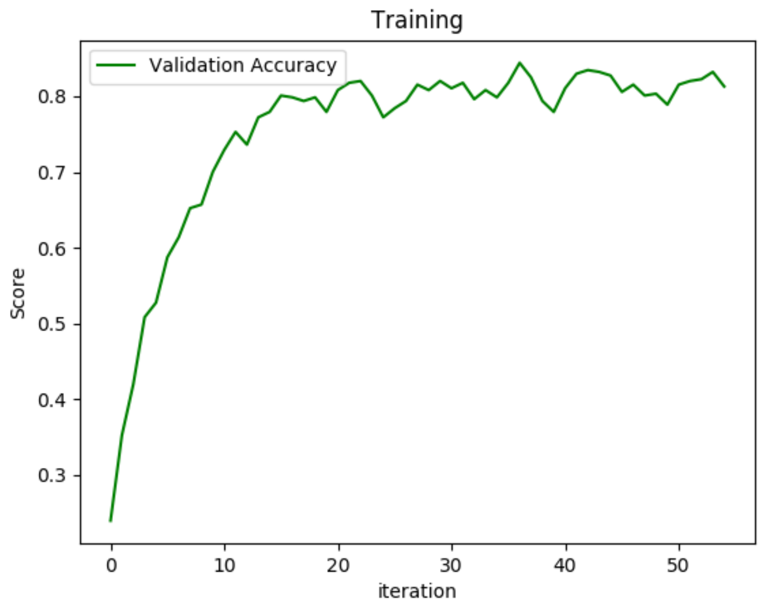
Since we balanced the classes in the dataset, we can use the accuracy as the evaluation and validation metric.

Different architectures during the refinement process gave us different values of accuracy. A short summary table for all architectures that has been built:

Architecture	Accuracy
1	0.543
2	0.597
3	0.388
4	0.639
5	0.712
6	0.821

Architecture 6 has the highest accuracy, therefore we will use it as our final solution.

The validation accuracy for every epoch is shown here:



The final testing accuracy is 0.821, which is very good, given that the naive accuracy for 7 classes would be around 0.14.

The final console output is saved in the console_output.txt file.

The model has also been tested with several reshuffled training/validation/testing dataset. It always showed very similar results with the accuracy ranging from 0.79 to 0.83.

Justification

Our model performed very well, compared to the benchmark model with the accuracy of 0.825.

Also the winner of the original Kaggle competition with exactly the same dataset had the accuracy of 0.5.

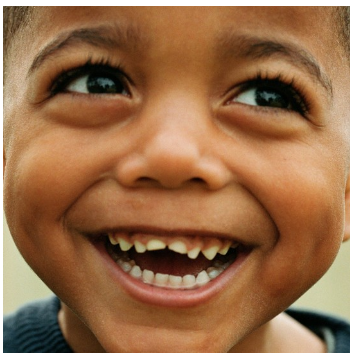
In the field of emotion recognition it is hard to be 100% accurate. People themselves don’t always understand their emotions, let alone computers. Therefore we would call the solution successful.

V. Conclusion

Free-Form Visualization

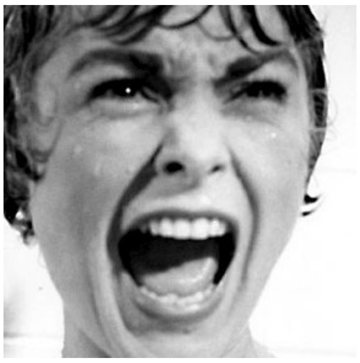
It’s good to have a testing dataset, it allows us to evaluate our model. But we also implemented the opportunity to upload any custom facial image to classify.

Let’s take a look at our first image which clearly expresses happiness:



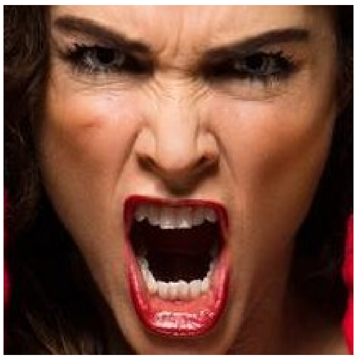
The model classified this as “happy”, which is true.

The next image:



Was classified as “fear”, which is also true.

The next one:



Was classified as “anger”, which is obviously true.

However it was interesting to test the model on the famous Mona Lisa smile:

The model output was “surprise”. No wonder, every one sees different kinds of emotions on this face.

Reflection

I find the whole area of understanding human behavior very exciting. If we can teach a model to understand human feelings, we would be able to solve different problems. For example understanding customer satisfaction, predicting conflicts, etc.

The most difficult problem in this particular type of tasks (building a convolutional neural network classifier) was defining good hyper-parameters for the model (convolution size, strides, number of layers, regularization, etc. But it all comes with practice.

Also a very important aspect of the emotion recognition area itself is finding a good dataset. And here comes a problem – it’s hard to find a good dataset.

All the photos in the Kaggle dataset look a bit artificial, like professional actors played 7 types of emotions when they have been asked to do it. If we want real emotion – some of those 7 emotions are really hard to capture. Who, for example, would take a photo when he/she experiences fear of anger?

I am pleased with the 82% accuracy. I didn’t expect it to be 99%, because as we know from the Mona Liza example above, there could not be a certain type of emotions on the face. Sometimes it’s hard to say whether it’s anger or fear, sometimes both maybe.

Improvement

I see two major ways of improvement – improving the model and improving the dataset.

We could improve the model by increasing the depth of the neural network – adding more convolutional layers, pooling layers and fully-connected layers. Increasing complexity has its downsides – training time could be very long.

We already briefly discussed the way of improving the dataset above. I want to add that we could improve the labeling method. Instead of using one class label for every image, images could belong to several classes. For example: {“fear”: 0.9, “anger”: 0.7, “happy”: 0.0, “sad”: 0.1, …}

References

https://inclass.kaggle.com/c/facial-keypoints-detector

http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2731770/

https://en.wikipedia.org/wiki/Convolutional_neural_network

http://cs231n.github.io/convolutional-networks/

http://deeplearning.net/tutorial/lenet.html

https://en.wikipedia.org/wiki/Backpropagation