**FACIAL EMOTION RECOGNITION**

*Soham Kulthe, Yash Kulkarni, Abhishek Kuber*

*October 2020*

Today, the most common way of communication among people is virtual platforms, whether using internet or phones The current generation uses online applications and platforms to communicate and exchange conversations. However, communicating emotions is difficult. As a result, small and simple pictures, otherwise known as emoji characters, are used to supplement emotions when using written language. Emoji characters are becoming more and more popularized therefore the variation of these characters has increased.

Emojis or avatars are ways to indicate nonverbal cues. These cues have become an essential part of online chatting, product review, brand emotion, and many more. It also lead to increasing data science research dedicated to emoji-driven storytelling.

With advancements in computer vision and deep learning, it is now possible to detect human emotions from images. In this deep learning project, we will classify human facial expressions to filter and map corresponding emojis or avatars.

**About the dataset**

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The task is to categorize each face based on the emotion shown in the facial expression in to one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).

train.csv contains two columns, "emotion" and "pixels". The "emotion" column contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image. The "pixels" column contains a string surrounded in quotes for each image. The contents of this string a space-separated pixel values in row major order. test.csv contains only the "pixels" column and your task is to predict the emotion column.

The training set consists of 28,709 examples. The public test set used for the leaderboard consists of 3,589 examples. The final test set, which was used to determine the winner of the competition, consists of another 3,589 examples.

This dataset was prepared by Pierre-Luc Carrier and Aaron Courville, as part of an ongoing research project. They have graciously provided the workshop organizers with a preliminary version of their dataset to use for this contest.

**IDE used:**

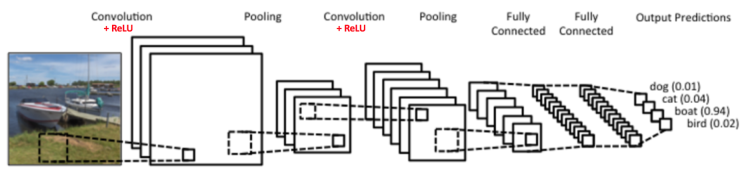
Spyder

**Convolutional Neural Networks**

Convolutional Neural Networks (ConvNets or CNNs) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. ConvNets have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self driving cars.

LeNet was one of the very first convolutional neural networks which helped propel the field of Deep Learning. This pioneering work by Yann LeCun was named LeNet5 after many previous successful iterations since the year 1988 [3]. At that time the LeNet architecture was used mainly for character recognition tasks such as reading zip codes, digits, etc.

Below, we will develop an intuition of how the LeNet architecture learns to recognize images. There have been several new architectures proposed in the recent years which are improvements over the LeNet, but they all use the main concepts from the LeNet and are relatively easier to understand if you have a clear understanding of the former.



There are four main operations in the ConvNet shown in Figure 3 above:

* Convolution
* Non Linearity (ReLU)
* Pooling or Sub Sampling
* Classification (Fully Connected Layer)

**What are channels in images**

Channel is a conventional term used to refer to a certain component of an image. An image from a standard digital camera will have three channels – red, green and blue – you can imagine those as three 2d-matrices stacked over each other (one for each color), each having pixel values in the range 0 to 255.

A grayscale image, on the other hand, has just one channel. For thisproject, we have used grayscale images, so we will have a single 2d matrix representing an image. The value of each pixel in the matrix will range from 0 to 255 – zero indicating black and 255 indicating white.

**The Convolution Step**

ConvNets derive their name from the “convolution” operator. The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. We will not go into the mathematical details of Convolution here, but will try to understand how it works over images. In CNN terminology, the matrix that slides over the image array is called a ‘filter‘ or ‘kernel’ or ‘feature detector’ and the matrix formed by sliding the filter over the image and computing the dot product is called the ‘Convolved Feature’ or ‘Activation Map’ or the ‘Feature Map‘. It is important to note that filters acts as feature detectors from the original input image.

**Introducing Non Linearity (ReLU)**

An additional operation called ReLU has been used after every Convolution operation. ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by:

**output = max (zero, input)**

ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation – element wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).

**The Pooling Step**

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.

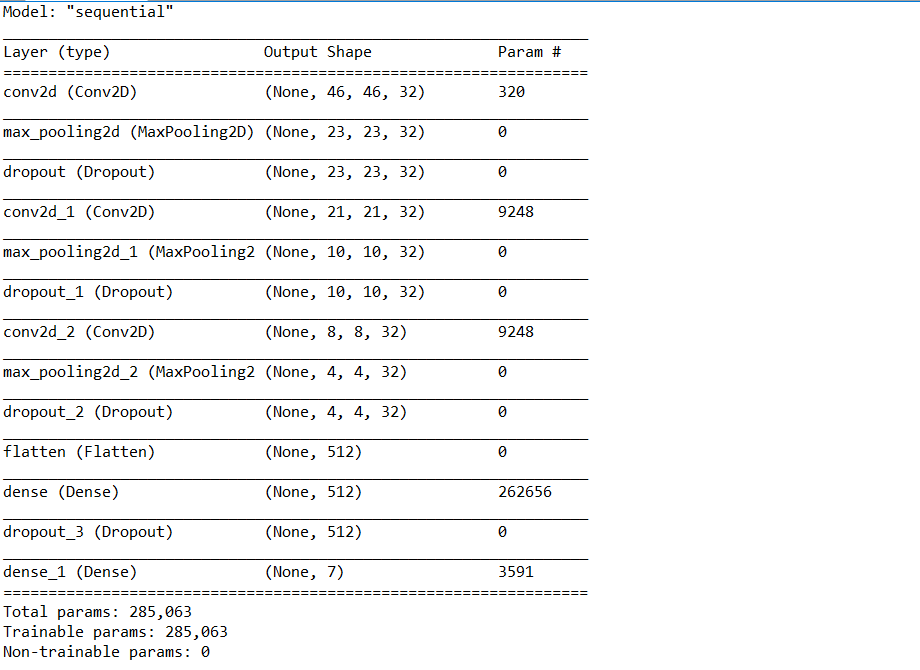
In case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.

**Fully Connected Layer**

The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer. The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer. The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset. For example, the image classification task we set out to perform has seven possible outputs, namely ‘angry’, ‘disgust’, ‘fear’, ‘happy’, ‘neutral’, ‘sad’, ‘surprise’.

**Our Model**

Our model consists of 3 convolution layers, each one having its own MaxPool and a Dropout layer with it. We then added a layer consisting of 512 neurons followed by an output layer. The model is summary is as follows:



* We added a kernel constraint that doesn’t allow the weights to exceed 3. This constraints the weight matrix which is a kind of a regularization.
* The Dropout layer randomly sets input units to 0 with a frequency of rate (0.25 in this case) at each step during training time, which helps prevent overfitting.

While compiling the model, we set the loss to ‘Categorical Crossentropy’. This computes the cross entropy loss between the labels and predictions.

The optimizer we used was the Adam optimizer. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

The amount that the weights are updated during training is referred to as the step size or the “learning rate.” The learning rate can be decayed over a fixed number of training epochs, then kept constant at a small value for the remaining training epochs to facilitate more time fine-tuning.

***cnn.compile(loss='categorical\_crossentropy',optimizer=Adam(lr=0.0001, decay=1e-6),metrics=['accuracy'])***

**Capturing our image through the webcam**

We needed an image on which we needed our model to make predictions. So for this purpose we capture an image using the computer’s webcam. We use OpenCV and face\_recognition for this. The library automatically detects the user’s face and upon pressing the key ‘t’, we capture the image. This is the image we feed into the model for predictions.

But we have to make some changes before we can do this. We rescale the image (feature scaling). Then we have to change the image from rgb to grayscale (as the model was fed grayscale images while training).

**Accuracy**

Naturally the metric used was accuracy. The hyper parameters are mentioned in the ‘Our Model’ section. After training over 25 epochs, the accuracy was around 49.7%.

**References**

<https://keras.io/api/>

<https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/#:~:text=Deep%20learning%20neural%20networks%20are,the%20model%20weights%20are%20updated>.

<https://www.tensorflow.org/api_docs/python/tf/keras/>

<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>