Welcome to freeCodeCamp News.

This is a free, open source, no-ads place to cross-post your blog articles. Read about it here.

1 OCTOBER 2018 / #MACHINE LEARNING

How I developed a C.N.N. that recognizes emotions and broke into the Kaggle top 10

of weeks old. As it grows, this innate ability improves. By the time it is a few months old, it starts to display social cues and is able to understand basic emotions like a smile.

Thanks to millions of years of evolution, we are able to understand each other without using a single word. Just a look and that is all that takes to understand whether a person is crestfallen or elated. Well, I tried teaching computers to do just that. This article is a detailed account of how the whole experiment turned out. Follow along as we recreate the network.

Cut to the chase Paul, please, give me the code. Don't want fancy reading? No problem. You can find the code for this project here.

A Brief Introduction

> "The best and most beautiful things in the world cannot be seen or even touched. They must be felt with the heart" — **Helen Keller**

Hellen Keller excellently described the essence of human emotions in the aforementioned quote. What was once reserved for animals is no longer limited to them. Machine learning is catching on at a mindnumbing pace. The onset of convolutional neural networks was a breakthrough and changed the way computers "look" at the world.

Facial expressions are nothing more than the arrangement of facial muscles to convey a certain emotional state to the observer. Emotions can be divided into six broad categories — Anger, Disgust, Fear, Happiness, Sadness, Surprise, and Neutral. In this M.L. project, we will train a model to differentiate between these.

Few different types of facial expressions.

We will train a convolutional neural network using the FER2013 dataset and will use various hyper-parameters to fine-tune the model. We will train it on Google Colab, which is a research project created to disseminate ML education. They will allocate you some resources like G.P.U. or T.P.U., and these can be used to train your model faster. The best part is that it is completely free.

Peek at the data

We will start by uploading the FER2013.csv file to our drive so that we can access it from Google Colab. There are 35,888 images in this dataset which are classified into six emotions. The data file contains 3 columns — Class, Image data, and Usage.

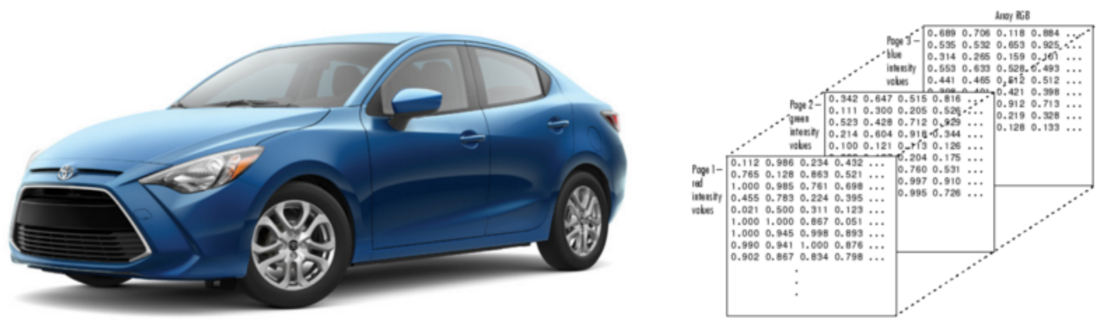
Class: is a digit between 0 to 6 and represents the emotion depicted in the corresponding picture. Each emotion is mapped to an integer as shown below.

```
0 - 'Angry'1 - 'Disgust'2 - 'Fear' 3 - 'Happy' 4 - 'Sad' 5 - 'Su
```

Image data: is a string of 2,304 numbers and these are the pixel intensity values of our image, we will cover this in detail in a while.

Decomposing an image.

As we all know that images are composed of pixels and these pixels are nothing more than numbers. Colored images have three color channels — red, green, and blue — and each channel is represented by a grid (2-dimensional array). Each cell in the grid stores a number between 0 and 255 which denotes the intensity of that cell.


Array RGB
Page 3 – blue intensity values
Page 2 – green intensity values
Page 1 – red intensity values

What you see (L) vs. what a computer sees.

When these three channels are aligned together we get the images that we see.

Importing Necessary Libraries

```
%matplotlib inlineimport matplotlib.pyplot as plt
```

```
import numpy as npfrom keras.utils import to_categoricalfrom sl
```

```
from keras.models import Sequential #Initialise our neural netwo
```

Define Data Loading Mechanism

Now, we will define the load_data() function which will efficiently parse the data file and extract necessary data and then convert it into a usable image format.

All the images in our dataset are 48x48 in dimension. Since these images are gray-scale, there is only one channel. We will extract the image data and rearrange it into a 48x48 array. Then convert it into unsigned integers and divide it by 255 to normalize the data. 255 is the maximum possible value of a single cell. By dividing every element by 255, we ensure that all our values range between 0 and 1.

We will check the *Usage* column and store the data in separate lists, one for training the network and the other for testing it.

```
def load_data(dataset_path):



data = []  test_data = []  test_labels = []  labels =[]



 with open(dataset_path, 'r') as file:     for line_no, line i
```

testing and training data by one to accommodate the channel. Then, we will one hot encode all the labels using the to_categorical() function and return all the lists as *numpy* arrays.

We will load the data by calling the load_data() function.

```
dataset_path = "/content/gdrive/My Drive/Colab Notebooks/Emotion
```

```
train_data, train_labels, test_data, test_labels = load_data(dat
```

```
print("Number of images in Training set:", len(train_data))print
```

Our data is loaded and now let us get to the best part, defining the network.

Defining the model.

We will use Keras to create a Sequential Convolutional Network. Which means that our neural network will be a linear stack of layers. This network will have the following components:

1. Convolutional Layers: These layers are the building blocks of our network and these compute dot product between their

2. Activation functions: are those functions which are applied to the outputs of all layers in the network. In this project, we will resort to the use of two functions— *Relu* and *Softmax*.
3. Pooling Layers: These layers will downsample the operation along the dimensions. This helps reduce the spatial data and minimize the processing power that is required.
4. Dense layers: These layers are present at the end of a C.N.N. They take in all the feature data generated by the convolution layers and do the decision making.
5. Dropout Layers: randomly turns off a few neurons in the network to prevent overfitting.
6. Batch Normalization: normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. This speeds up the training process.

```
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(48,
```

We will compile the network using Adam optimizer and will use a variable learning rate. Since we are dealing with a classification problem that involves multiple categories, we will use *categorical_crossentropy* as our loss function.

```
adam = optimizers.Adam(lr = learning_rate)



model.compile(optimizer = adam, loss = 'categorical_crossentropy
```

Callback functions

Callback functions are those functions which are called after every epoch during the training process. We will be using the following callback functions:

1. ReduceLROnPlateau: Training a neural network can plateau at times and we stop seeing any progress during this stage. Therefore, this function monitors the validation loss for signs of a plateau and then alter the learning rate by the specified factor if a plateau is detected.

```
lr_reducer = ReduceLROnPlateau(monitor='val_loss', factor=0.9, pa
```

2. EarlyStopping: At times, the progress stalls while training a neural network and we stop seeing any improvement in the validation accuracy (in this case). Majority of the time, this means that the network won't converge any further and there is no point in continuing the training process. This function waits for a specified number of epochs and terminates the training if no change in the parameter is found.

```
early_stopper = EarlyStopping(monitor='val_acc', min_delta=0, pa
```

3. ModelCheckpoint: Training neural networks generally takes a lot of time and anything can happen during this period that may result in loss of all the variables and weights. Creating checkpoints is a

```
checkpointer = ModelCheckpoint('/content/gdrive/My Drive/Colab N
```

Time to train

All our hard work is about to be put to the test. But before we fit the model, let us define some hyper-parameters.

```
epochs = 100batch_size = 64learning_rate = 0.001
```

Our data will pass through the model 100 times and in batches of 64 images. We will use 20% of our training data to validate the model after every epoch.

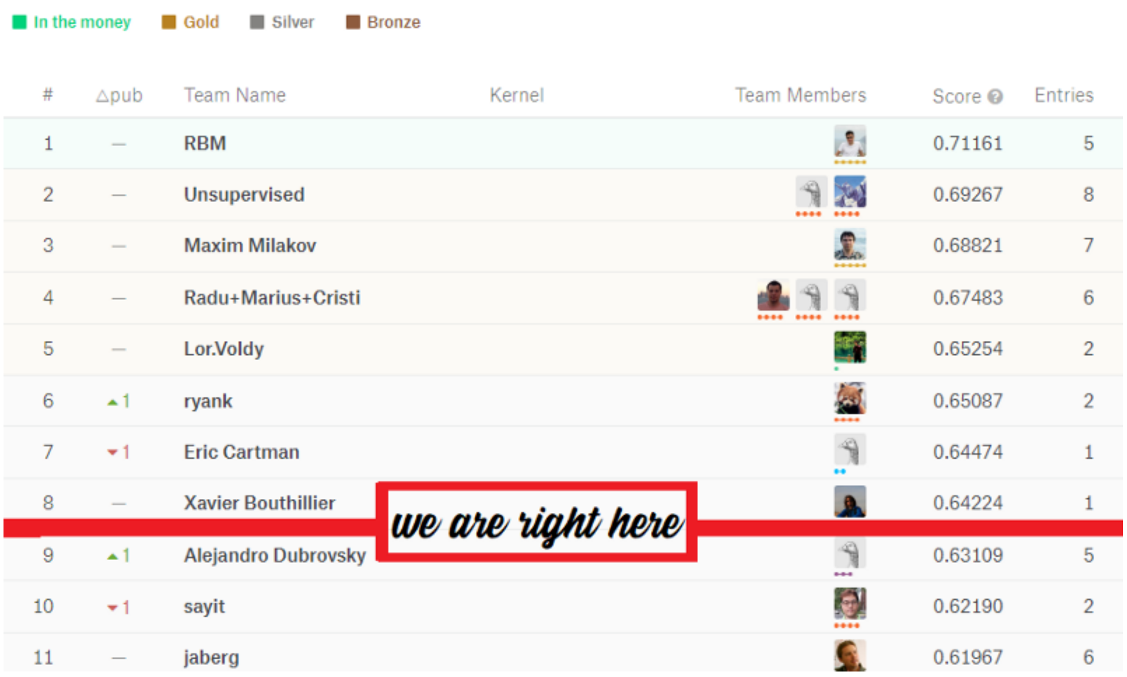
```
model.fit(          train_data,          train_labels,
```

Now that the network is being trained, I suggest that you go and finish that book you started or go for a run. It took me about an hour on Google Colab.

Test the model

Remember the private set we stored separately? That was for this very moment. This is the moment of truth and this is where we will reap the fruit of our labor.

Well, the results came back and we scored 63.167%. On first glance, it isn't much but we broke into the ninth position of the Facial Emotion Recognition Kaggle competition.



🟩 In the money 🟨 Gold ⬜ Silver 🟫 Bronze

#	△pub	Team Name	Kernel	Team Members	Score	Entries
1	—	RBM			0.71161	5
2	—	Unsupervised			0.69267	8
3	—	Maxim Milakov			0.68821	7
4	—	Radu+Marius+Cristi			0.67483	6
5	—	Lor.Voldy			0.65254	2
6	▲1	ryank			0.65087	2
7	▼1	Eric Cartman			0.64474	1
8	—	Xavier Bouthillier			0.64224	1
9	▲1	Alejandro Dubrovsky			0.63109	5
10	▼1	sayit			0.62190	2
11	—	jaberg			0.61967	6

It's not a big deal though

Now, pat yourself on the back and start brainstorming about the ways in which you can improve this model. We can use better hyper-parameters or create a different network architecture altogether to achieve higher accuracies.

Quickly save the model using *model_from_json* from *keras.models*.

```
 from keras.models import model_from_json



odel_json)# serialize weights to HDF5model.save_weights("/content/g
```

Wrapping it all up

We started off by defining a loading mechanism and loading the images. Then we created a training set and a testing set. Then we defined a fine model and defined a few callback functions. We went over the basic components of a convolutional neural network and then we trained our network.

I extended this project by creating a python application which is able to detect faces and recognize their emotions in real time. That will be covered in a later post.

We just accomplished something that was part of science fiction a few decades ago. Yet there is a lot left to learn. The internet provides us with a plethora of information to constantly create and learn. May the learning never cease.

Start Discussion 0 replies