Facial Expression Recognition using Deep Learning Neural Network

Assa, Yuval
ID: 318612025

Daniel, Sean
ID: 315619643

Submitted as final project report for the DL course, COLMAN, 2021

1 Introduction

Facial expressions play an important role in conveying the emotional states of human beings.
Facial expression recognition is the task of classifying the expressions on face images into various categories such as anger, fear, surprise, sadness, happiness and so on.
We chose this topic because we were interested in images manipulation and understanding in general, and extract relevant face emotional states in specific. This work is probably our first to start working with images, but might not be the last.
We searched for some papers regarding this problem - they will be discussed later in this paper.

1.1 Related Works

1. Deep Facial Expression Recognition: A Survey.
2. Deep-Emotion: Facial Expression Recognition Using Attentional Convolutional Network
3. Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution
4. Improved Regularization of Convolutional Neural Networks with Cutout

2 Solution

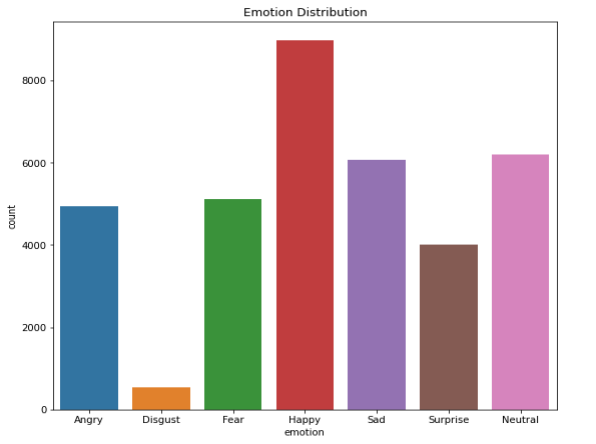
2.1 General approach

The approach is training a neural network with a given tagged, pixelized image dataset while tracking the loss, validate and tune the model's hyper-parameters

using validation set, test and record the accuracy.
We wanted to investigate what will happen to the test's results if we train the net with different image augmentations.
So we decided that the experiment will be: first, training the net with regular image dataset and test it with the test data. Second, train the net with train data with some randomly noises (the method will be explained in the "Design" section).
We decided to use a FER dataset that is classified to 7 emotions: Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral.
Each image is gray scaled and its metadata contains information about the emotion, Pixels of the image ranging from 0-255 and usage indication (training/validation/test).
Each image is in size of 48X48 - total of 2,304 pixels per image.
The dataset contains 35,887 images categorized to 7 different categories (the categories mentioned above).

Below is a histogram that shows the images dataset distribution by emotions:

2.1.1 Images Dataset Emotion Distribution

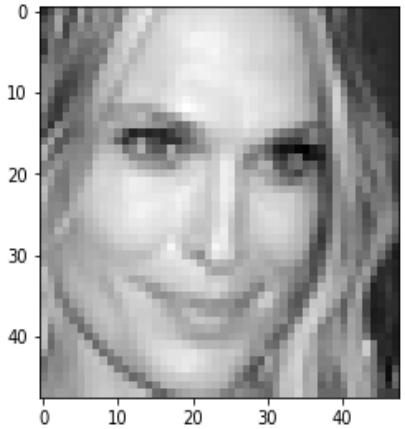


2.2 Design

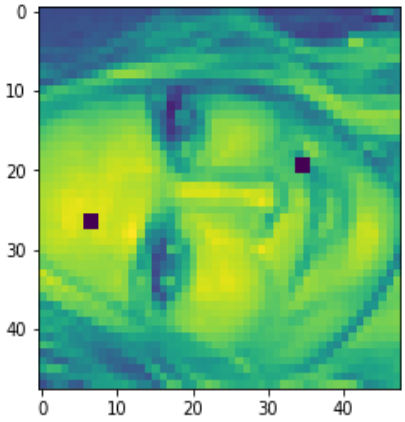
We wrote our code in Jupyter notebook on Google Colab platform.
We decided to use deep CNN model after investigating which model is most suitable for this specific task.
We started by calibrating the model hyperparameters in order to get the maximum accuracy on the test set.
First, we loaded the dataset and converted the object-like "pixels" column from the dataset to array of Integers. After that, we splitted the data into 3 sets: train_set, validation_set, test_set as well as their labels. Then we created helper classes like AlbumentationWrapper as a wrapper to augment some images

and Dataset to store the images (transformed and regular) and their emotions. The augmentation method is as follows: we randomly flip the image, add some noise to the image - by Gauss distribution and cutout.

The following image is an example of a regular image from the trainig set:



The following image is an example of the same image but augmented:



At last, we normalized the image's pixels to be from range 0-255 to 0-1.
We used the following Deep CNN as the model:
Conv2d (in: 3, out: 32, kernel size: (3, 3), padding: 1)
ReLU
BatchNorm2d (32)
As input block and the same is repeated once, after that MaxPool2d(2, 2) and the same input block 3 times, another MaxPool2d(2, 2) and the same block

2 times until the input channel is size of: 1024 and the output channel is size of: 1024. After that another MaxPool2d(2, 2) and start reduce number of channels until 256, then AvgPool2d(kernel size: 4) and the last layer is: Conv2d(in channels: 256, out channels: 7, kernel size: (1, 1), padding: 0) with log_softmax on top (to classify the output).

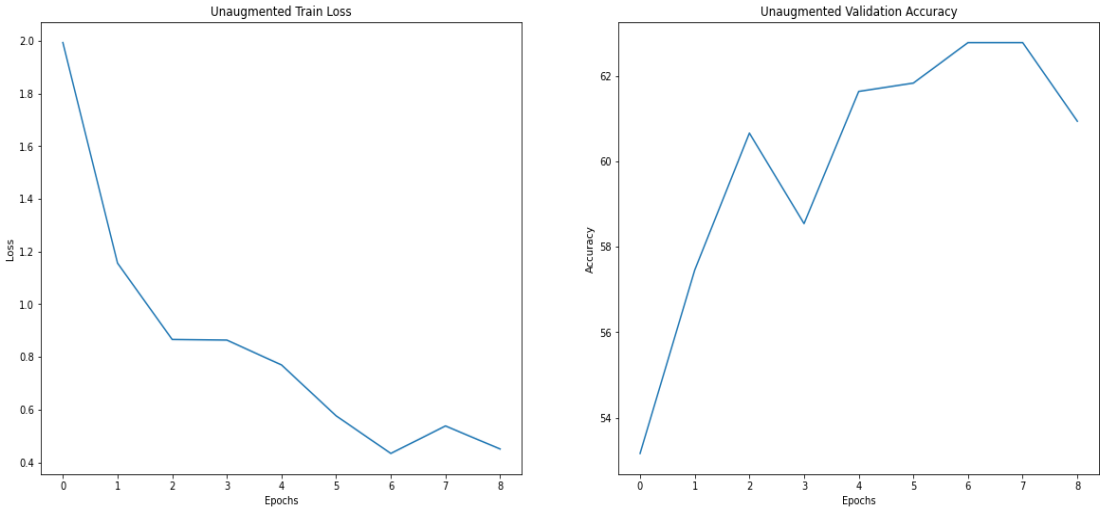
The training process took 15min and 37s for 9 epochs. While training the net, we validated it with the validation dataset and tuned the net's hyperparameters according to the results - in order to preform better.
Meanwhile, we ran the test dataset through the net and record the accuracy change. We did the same process for the experiment mentioned in the next section.

3 Experimental results

The image dataset (total of 35,887 images) was divided in advance into train, validation and test sets with the percentage of 80% (28,709), 10% (3,589), 10% (3,589) accordingly.
To tune our net, we had to do some experiments with the net's hyperparameters. We found out that our net got the highest accuracy by training the net with learning rate of 0.003.
We used batches of 64, CrossEntropyLoss as criterion and SGD as optimizer with momentum of 0.9 and weight decay of 0.0009.
As mentioned above, we trained our network using two different ways. The first one is using augmented training data (images with randomly noise), and the second one is using regular, unaugmented images. Training the model using the regular images from the dataset, caused the model to be over-fitted very quickly (after approximately 7 epochs). We wanted the model to not be over-fitted, so we decided to add some noise to the regular training images.

Below is chats that shows the results by training the net using regular unaugmented images:

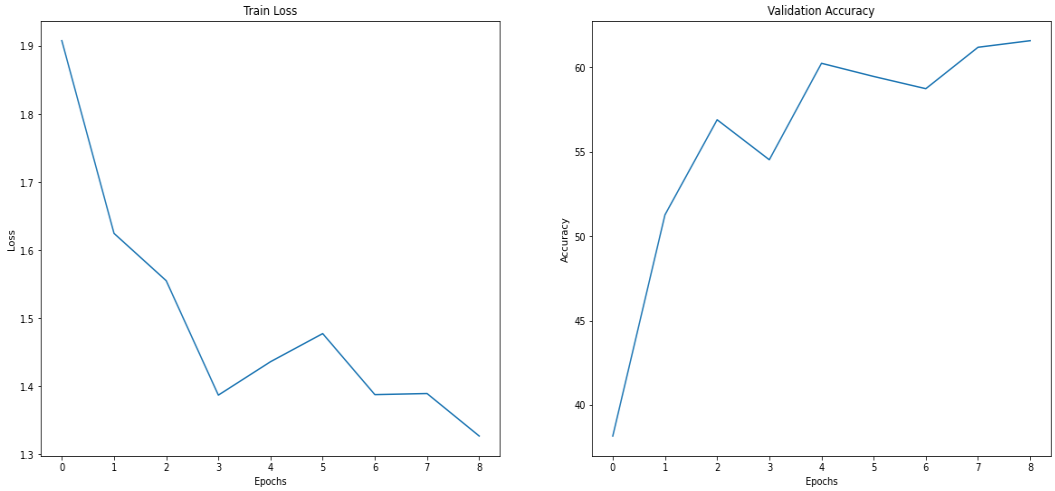
3.0.1 Unaugmented (regular) images Training loss & Validation accuracy in each epoch - 9 epochs



We can see from the left chart that the train loss is reducing vary fast as the number of epochs increase.
It leads to over-fitting - on the right chart we can see that the last 2 epochs, the validation accuracy is decreasing.

Below is chats that shows the results by training the net using both regular and augmented images:

3.0.2 Augmented+regular images Training loss & Validation accuracy in each epoch - 9 epochs



We can see from the left chart that the train loss is reducing not so fast as the number of epochs increase. We can also see that the loss is still high after all the epochs.
But we can see from the right chart that the model is not over-fitted as the accuracy still stays high and almost the same accuracy as the one recieved by the unaugmented image experiment.

4 Discussion

First, we wanted to implement our own Facial Expression Recognition classifier. But then we were interested in how we can improve the classifier so we decided to run some experiments.
After conducting the experiments mentioned in this paper, we can conclude that training the net with data that is almost the same as the test data, the net is tending to be over-fitted quickly.
It is better to train the net with some other "out of domain" examples in order to get a better net.

5 Code

https://github.com/yuvala15/DL_COURSE_PROJECT

References

[1] Deep Facial Expression Recognition: A Survey,
https://arxiv.org/pdf/1804.08348.pdf

[2] Deep-Emotion: Facial Expression Recognition Using Attentional Convolutional Network,
https://arxiv.org/pdf/1902.01019.pdf

[3] Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution,
https://arxiv.org/pdf/1608.01041.pdf

[4] Improved Regularization of Convolutional Neural Networks with Cutout,
https://arxiv.org/pdf/1708.04552.pdf

[5] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,
https://arxiv.org/pdf/1502.03167.pdf

[6] PyTorch and Albumentations for image classification,
https://albumentations.ai/docs/examples/pytorch_classification/

[7] Kaggle fer2013 dataset,
https://www.kaggle.com/deadskull7/fer2013

[8] Google Colab Platform,
https://colab.research.google.com/