SHAHAR AZOULAI
Student ID: 157224

Introduction

In this assignment we've been asked to implement a neural network for facial emotion classification given by an image, therefore the natural choice would be to use a CNN. The goal of this work is to see how different architecture and hyper parameters are affecting the behavior of our model and most importantly its error(or accuracy) on the distribution.

FER2013:

- Training set size: 28,709 examples, test set size: 3,589 examples
- Examples are given as 48*48 pictures with 1 channel(greyscale)
- 7 possible labels (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral)
- According to the site paperswithcode.com, the most successful model on the given test set is the "VGGNet" model which was introduced in 2020 and scored 73.28% accuracy.

In this work I have tried at first a pretty simple network and have examined the impact of changing some hyper parameters such as the learning rate(SGD), batch size and more.

1st Architecture – "Same Same, But Different"

The first architecture I have used is the architecture we have seen in class during a practical session on CNNs(figure 1.1) for the CIFAR-10 dataset. The only difference is the number of "in_channels" for the first convolutional layer which is now set 1 instead of 3.

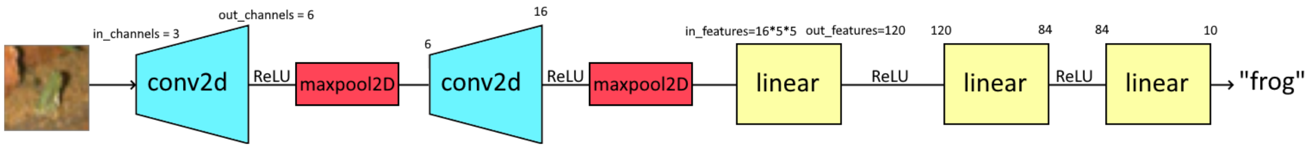

Figure 1.1: 1st architecture diagram

I chose this architecture for few reasons: the first is because I wanted to use something I am familiar with so I can understand the impact of the hyper parameters in a better way. In addition, this is a (relatively) simple architecture which doesn't take a lot of time to train on the machine I am using so I could try more options for the hyper parameters. The third is that even with no changes, the model showed some pretty good results of about 46% which puts this model in the top 40 in the kaggle's challenge leaderboard.

1) Standardization

Using the sklearn.preprocessing package I have standardized the data set. This standardization is like the one learned in class and works in the following way: $x' = \frac{x-\mu}{\sigma}$

Where: $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \quad ; \quad \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$

Running architecture #1 with the same hyper parameters as before has led to an accuracy decrease of 13%. Therefore, in the next tests I would prefer to use the raw data.

SHAHAR AZOULAI
Student ID: 157224

2) Learning rate

Here I have tried 6 different initial learning rates (0.0001, 0.001, 0.005, 0.01, 0.1, 1) for the SGD algorithm. Here I wanted to see the impact of the initial learning rate on the loss minimization(figure 1.2, left) and on the time(seconds)(figure 1.2, right).

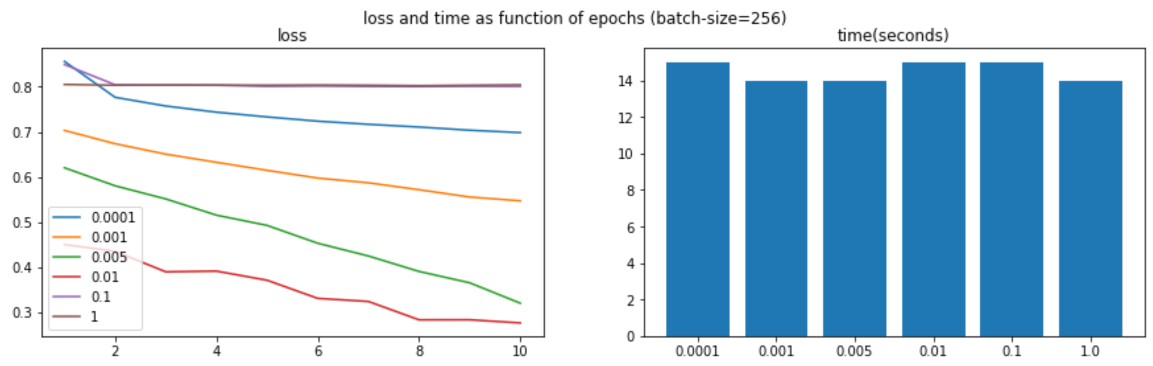

Figure 1.2: loss per learning rate as function of epoch(left) and training time for each learning rate(right) (batch size = 256)

We can clearly see how lr=0.01 minimize the lost better even though lr=0.01 is usually considered pretty big in the research community. The reason that might be behind it is because in the training behind fig.2 I have used a mini-batch size of 256.
In a paper by our beloved Quoc V. Le and other friends at Google Brain(Samuel L. Smith∗ , Pieter-Jan Kindermans∗ , Chris Ying & Quoc V. Le, Google Brain, 2018) we get a clear recommendation "Don't decay the learning rate, increase the batch size" right in its title. The intuition behind this nice slogan is that when we use a bigger batch we basically rely on more examples when performing backpropagation the next time. Thus, we can be more sure when making a step and that is why we can make it bigger.

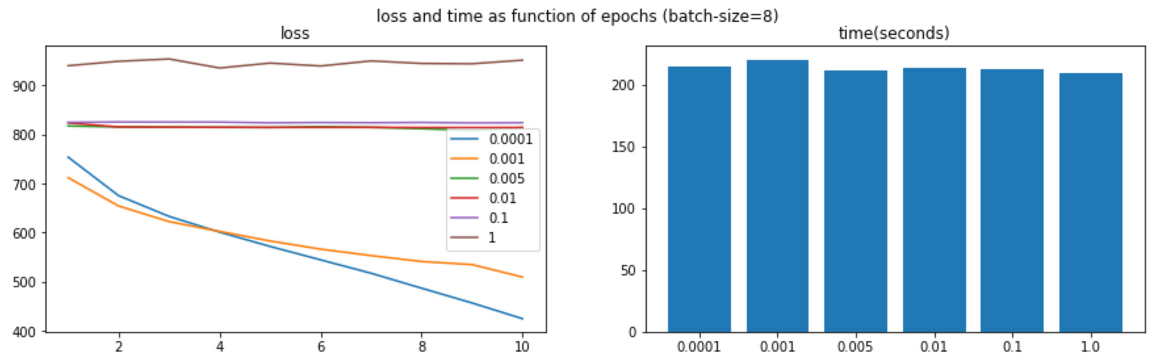

Figure 1.3: loss per learning rate as function of epoch(left) and training time for each learning rate(right) (batch size = 8)

In figure 1.3 we can see the same test running with batch size = 8 and here the smallest learning rate I have used(lr=0.0001) have shown better results. This stands together with the hypothesis mentioned above about the relation between the initial learning rate and the batch size. Another interesting thing we see is that for the bigger learning rates, SGD couldn't minimize the loss at all and probably haven't even achieved convergence at any point. In addition, we can also see about 10x increase in time taken for the training.

SHAHAR AZOULAI
Student ID: 157224

3) Batch size

Batch-size	Accuracy(%)	Time(seconds)	Epochs(#)
1	48	1,676	10
32	48	56	
256	47	15	
1024	24	13	
2048	24	13	
8192	14	16	
32	<u>48</u>	<u>112</u>	19

Here I wanted to see the impact of the batch size on the accuracy of the model as long on training time. I have ran 6 different batch sizes for 10 epochs each:

Using the classic SGD(batch-size=1) took a massive time of 1,676 seconds but also scored a 48% of accuracy on test set. When using a batch-size of 32 we get a 96.6% drop in time while preserving the accuracy. That is why we most often use batches.

2nd Architecture – “Yes, It is really from the Dataset”

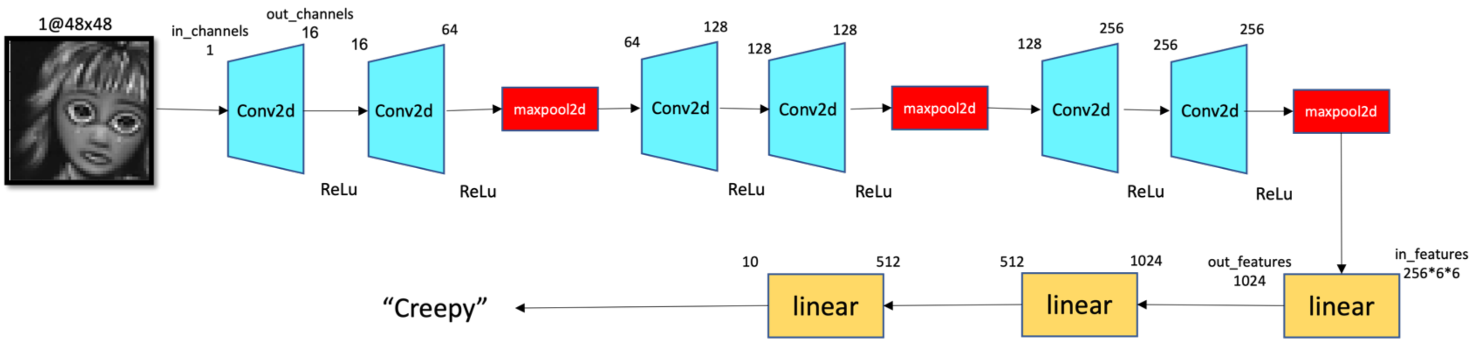

Figure 2.1: 2nd architecture diagram

Now I have used a bigger network archtecture which consists of one more pooling layer, 2 concolution layers before each pooling and 3 dense layers.In figure 2.2 we can see that after about 18 epochs the change in loss is not significant so we can train this for not so many epochs. And indeed, using the same hyper parameters(batch size, learning rate, etc.) we got an accuracy of 59% (808 seconds) just by changing the architecture.

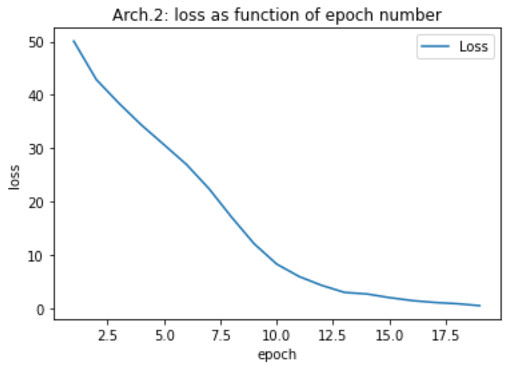

Figure 2.2: Loss as function of epochs in 2nd architecture (ReLU)

SHAHAR AZOULAI
Student ID: 157224

4) Activation Function

In this architecture I also wanted to check the impact of the activation function and especially was interested to see the difference between the ReLU and the Leaky ReLU which are very similar on paper. In figure 2.3 we can see the loss and I will tell you that with the same other hyper parameters the model scored 60% accuracy for 815 seconds.

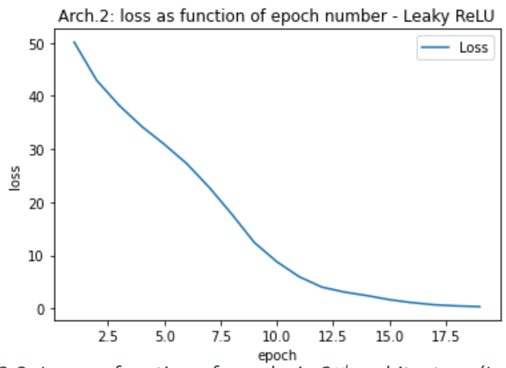

Figure 2.3: Loss as function of epochs in 2nd architecture (Leaky ReLU)

3rd Architecture – "Bigger is Better?"

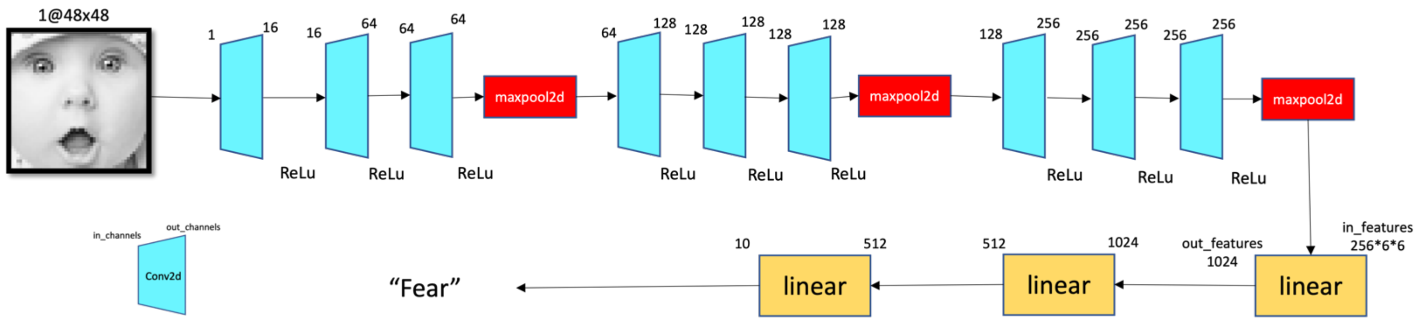

Figure 3.1: 3rd architecture diagram

In this architecture I have added before each pooling another convolutional layer which doesn''t change the number of channels that goes into it(in_channels=out_channels). I wanted to see if adding more convolutional layers can improve the model and if so, what would the cost be.
This model achieved 56% accuracy (1,277 seconds) so we can say that the addition of layers and complexity doesn't always improve our model. In figure 3.2 we can see the reduction in loss.

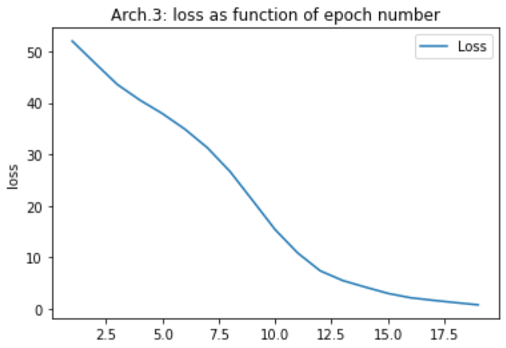

Figure 3.2: Loss as function of epochs in 33rd architecture

Summary

When addressing this assignment at first, I didn't quite understand its purpose. However, when starting to use the tools we have learned about, curiosity began to grow, and I began asking some questions such as:

- What is the impact of different values for different parameters? On accuracy and on running time.
- Are there any relations between different hyper parameters?
- Can a small compromise in accuracy have a significant improve in time?

In this assignment I have used some practical tools to answer the questions I have had and after having different observations I have looked for explanations for the different behaviors I have seen. However, more questions and thoughts started to appear while working on this assignment so I have many more things to do.