Image classification with Deep Learning techniques

Jose Renato Restom, Aliakbar Abdurahimov
MBZUAI
Jose.Viera@mbzuai.ac.ae, aliakbar.abdurahimov@mbzuai.ac.ae

1. Introduction

Convolutional neural networks or CNNs, are type of neural network used predominantly for data with a grid-like topology. The term convolutional network indicates the use of a special type of linear operation within the architecture called convolution. "CNNs are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers". [3]

For the purpose of this study we are going to analyze the effects of several methods to alter the learning capabilities of a simple LeNet-5 architecture. LeNet-5 was first introduced by Yann LeCun et al. in 1989. The architecture follows the pattern shown in figure 1.

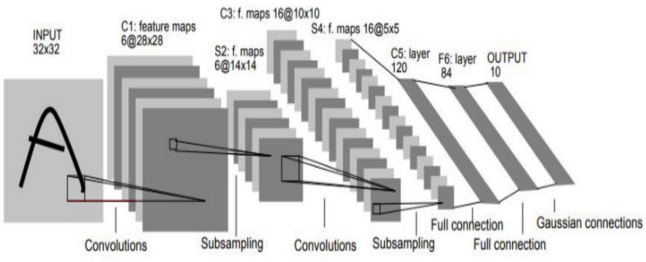


Figure 1. LeNet-5 original architecture. Image Taken from [4]

We will analyze the results of a series of experiments performed by changing the **pooling layers** , adding **dropout layers** and using **batch normalization** throughout the network.

1.1. Methods

Pooling layers

Pooling functions are typically used after the **detector stage** (nonlinear activation function) with the idea of further modifying the layer's output. The pooling function works by replacing the net's output in a neighboring area (window) with a summary statistic. Furthermore, pooling can be useful to shrink the representation size, which translates in to a reduction of the computational and statistical burden on the next layer. The figure 2 shows the inner works of two types of pooling functions over a 4x4 window.

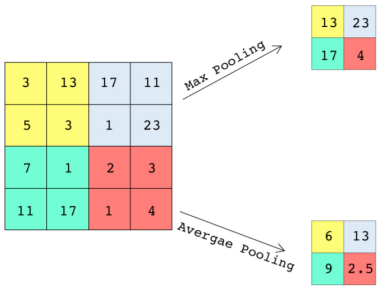


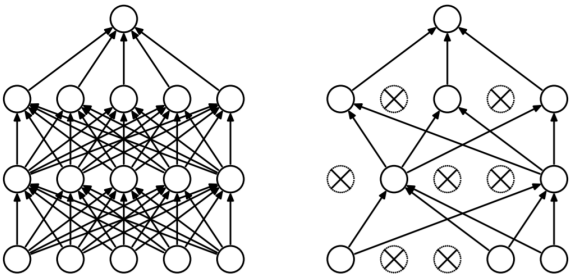
Figure 2. Example of max pooling and average pooling operations. Image Taken from [1]

Dropout layers

Deep neural networks contain several non-linear hidden layers, which grants them the ability to learn complex relationships between their inputs and outputs. However, since the data we use to train the networks is not always "clean" (absent of noise) nor is it unlimited, the possibility of learning relationships that exists as a result of sampling the noise appears. When said situation occurs, we say the model is **overfitting** the data. Dropout is a technique that was developed to address the overfitting challenge by providing "a way of approximately combining exponentially many different neural network architectures efficiently" [6]. The key idea consists of *dropping* neural units out by removing them and their connections from the network temporarily during the training time, as shown in Figure 3 .This process is made at random and independently for each neuron.

Batch normalization

Due to the constant change in the layer's inputs distribution during training, networks are forced to continuously adapt to said changing distribution each time. This phenomenom is called *covariate shift* [5] and it highly impacts the time needed to reach convergence when training a neural network. Batch normalization is a technique created to mitigate the ill effects of the internal covariate shift. It works by whitening (normalizing) the inputs to each layer to take a step forward towards achieving a fixed distribution of in-



(a) Standard Neural Net (b) After applying dropout.

Figure 3. Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. Image Taken from [6]

puts. [2]

2. Experiments

2.1. Pooling

For this experiment we want to analyse the effects of using two types of pooling functions *Average pooling and Max-pooling*. The table 1 bellow summarizes the results obtained after 30 epochs of training.

Function	Kernel size	Stride	Accuracy	Training time (s)	Inference time (s)
Avg-pooling	2	1	52%	291.3	1.74
Max-pooling	2	1	55%	292.2	1.77
No pooling (plain convolutions)	5	2	52%	288.2	1.67

Table 1. Results with Avg-pooling, Max-pooling and No-pooling. The accuracy reported is the highest one across all epochs.

We can observe from the outcome that pooling functions seem to slightly increase the training and inference time, however the reason for this is because the convolution used for the model without pooling layers is using a stride of 2 to match them. It is important to remember that the pooling function down-samples the inputs by using a statistical metric, hence using larger kernels and strides can help decrease the overall training time and complexity of the network, but it will also affect the performance. Finally, it is worth noticing that our model using Max-pooling layer obtained the best accuracy out of the three. The explanation for this could be that pooling helps to make the representation become approximately invariant to small translations of the input (the output of the pool operation does not change if we translate the input by a small amount) [3], and the max operation is less susceptible to this subtle changes than the average operation.

2.2. Dropout

For this next experiment we want to see the results after applying Dropout layers with different probabilities p.

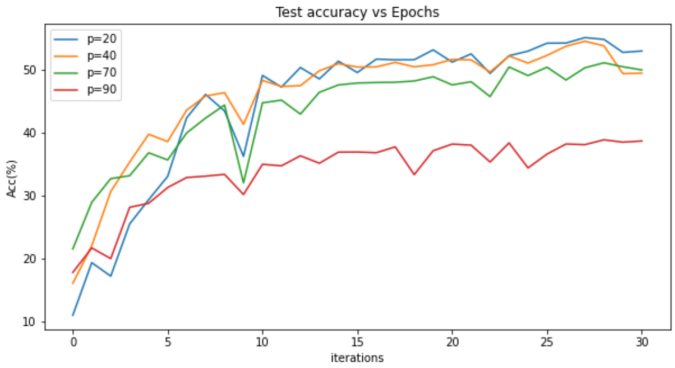


Figure 4. Test accuracy using different values of p for the dropout layers.

We can conclude from the experiment shown in Figure 4, that dropout is useful for regularizing the network, but since our base model was not getting a high accuracy, it is not helping us improve the performance for this particular case. Besides, it is also important to notice how the learning capabilities of the model are significantly hindered when p is too high (since many of the neurons are disconnected in this case).

2.3. Batch normalization

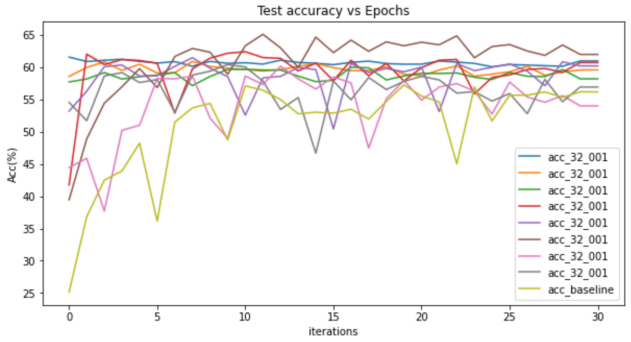
For this third experiment we want to explore the repercussions of using Batch normalization over the baseline model. The table 2 bellow presents a brief synopsis of the metrics resulting from several combinations of different learning rates and batch sizes.

Batch size	Learning rate	Best Accuracy	Training time (s)	Inference time (s)
16	0.01	62%	526.9	1.77
	0.03	61%	521.5	1.72
	0.05	60%	535.4	1.75
32	0.01	65%	378.6	1.75
	0.03	62%	378.1	1.74
	0.05	62%	377.0	1.76
64	0.01	62%	305.9	1.74
	0.03	60%	306.05	1.77
	0.05	58%	305.1	1.75

Table 2. Results for different combination of parameters for batch normalization.

Looking at the table 2 we can notice how just by using batch normalization we are able to easily best the baseline architecture by a considerable margin. Batch size and learning rate affect the overall performance, and we tend to obtain higher accuracy with smaller batches and smaller learning rates, although the training time is afflicted. Furthermore, even with higher learning rates we are able to obtain a better accuracy than the baseline, this is because unlike with traditional deep networks, where too high learning rate can result in exploding and vanishing gradients or getting stuck in local minima, in our model this issue is mitigated with the normalization of the activations throughout the network. [6]

On the other hand, looking at the Figure 5 we can conclude that Batch normalization helps the network to converge much faster, so even though it is not an optimization algorithm in itself, it's useful for optimizing the neural network training (the yellow line corresponding to the baseline model converges much slower compare to all the others using Batch normalization layers).



2.4. Batch normalization and Dropout

For experiment we try using both Batch normalization and Dropout layers to analyse their combined effect on the performance and time to converge.

Batch size	Learning rate	Best accuracy BN and Dropout	Best Accuracy BN	Time traning BN and Dropout	Time training BN
16	0.03	58%	61%	553.6	521.5
32	0.03	57%	62%	396.5	378.1
64	0.03	56%	60%	310.3	306.05

Table 3. Comparison between using BN and BN and Dropout.

In table 3 we can see that the architecture using only batch normalization bested it's counterpart with dropout for every experiment. The explanation for this phenomenon can be found in the original paper when batch normalization was introduced [6]. It is said by the authors that batch normalization possesses a regularizing effect by itself, so by applying dropout with it with are significantly decreasing the learning capabilities. Is is recommended to use batch normalization without dropout to further optimize the process.

2.5. Best architecture

Finally, we combine several of the methods shown in this study to try to improve the accuracy as much as possible. For this purpose we started by using *TORCHVISION.TRANSFORMS* on the data to as a mean to perform data augmentation. Then, making use of several convolutional layers, fully connected layers and batch normalization, we created a Lenet-5-like architecture with around 100.000 parameters. We decided to avoid using dropout for the reasons stated in the section 2.4.

The results of this network are shown bellow:

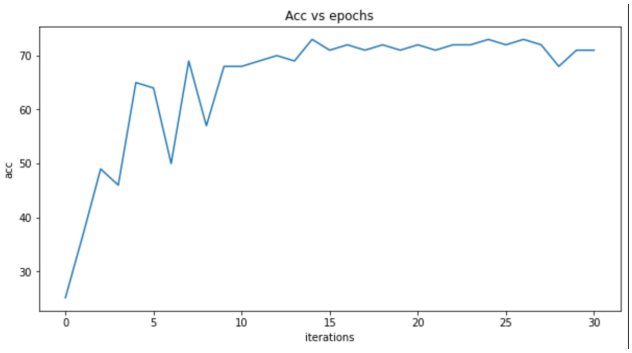


Figure 6. Test accuracy for the final architecture per epoch.

3. Conclusions

Pooling

This layers seem to be crucial for approximately making the outputs invariant to small translations in the input. Their kernel size and stride will account a lot for saving computational cost, but could hurt the performance.

Dropout

Useful regularizing technique that does not require much computation. Too high a probability p as parameter will hurt the model performance and too low will not regularize the network, finding the right value in between is the challenge.

Batch normalization

Besides accelerating the convergence of the model, it also has a regularizing effect. Typically not recommended to use with dropout.

References

[1] Nura Aljaafari. *Ichthyoplankton Classification Tool using Generative Adversarial Networks and Transfer Learning*. PhD thesis, 02 2018. 1

[2] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. 2

[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. 1, 2

[4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[5] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000. 1

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 1, 2, 3