Report Lab 1 Giuliani Gianluca

A. Introduction

In this report I will train some architectures on the MNIST and CIFRAR10 datasets. The objective of this lab is to verify the capabilities of a FC and CNN network, modify its parameters and structure and use predefined networks such as ResNet and compare their performances and perform fine tuning to check their performances.

B. Datasets

MNIST database of handwritten digits 28*28. It has a training set of 60,000 examples, and a test set of 10,000 examples.

CIFRAR10 contains 60000 32*32 color images divided into 10 classes, of which 6000 images per class. 50,000 training and 10,000 testing.

C. Assignment 1

At the end of this section I will put one table with all the results about Test accuracy and execution time.

For the following experiments I will use always the these parameters in not explicitly said: Epochs=10, lr for SGD = 0.01, lr for Adam = 0.0001. If not explicitly said i will use SGD optimizer.

In order to begin this study, I start working on MNIST, and I create a simple FC with 3 Linear layers. The results are quite good, I reach the 96% in accuracy, the good results surely are based on the fact that the dataset, MNIST, is quite simple.

I try to improve the architecture: The first thing I can do is adding some normalization layers after each linear layer. Doing so the accuracy goes to 97% and the execution time increase a little.

Now I try to increase the number of neurons so I can create larger layers. The performances are quite similar to the smaller FC with normalization. So I can apply normalization layers this new net created and the performances grows up to 98%. I can even distribute the neurons in more layers and apply for each linear layer the normalization and so I do it but the performances are quite similar.

Next I use another model: CNN. I use a CNN with 2 convolutional layers with kernel size = 3 and and 3 linear layers and a maxPool to divide these layers. The results of accuracy: 98%, so with a ”base” version of this new architecture i get the score of a way more enhanced FC. Let’s enhance this model changing e.g. kernel size. The baseline version has a 3*3 kernel so I decide to change it to 5*5. With Kernel size = 5 in fact I record 99% of accuracy. I repeat the experiment changing the kernel size of the layers: The first con2d layer will use kernel 3*3 and the second will use a 9*9, in this case I get the same accuracy score but in less time.

Then I change the layer between the convolutional and linear part of the net: The base CNN uses a maxPool so I will test avgPool and stride to reduce dimensionality of the feature maps. With stride value = 2 I get 98% of accuracy, with avgPool, 96%, and dilation value = 2, 99%.

The improvement done until now are about the model and its structure or the parameters of the layers, but I can change even the parameters of the optimizer like the learning rate. So I make the training of the base CNN with lr = 0.1 and 0.001 What I notice is that with higher lr is that the 99% is reached at the first epoch, so I would need way less epoch to do all the training, instead with lr=0.0001 the convergence is way slower, in fact at epoch=10 i reach only 89% of accuracy test. I can make more variations considering what optimizer I am using, in SGD I can add momentum, in this case I get similar results. I can even change the optimizer, like Adam, I get 99% as accuracy starting from first epoch and lr = 0.0001.

If i want to change the lr dinamically not using Adam I can use a scheduler, in this case Multistep, that changes the lr when my epoch reach a milestone, in this case 2, 5, 7. Lastly I try to change the batch sizes, the accuracy does not change, only the execution time becomes smaller as the batch size grows.

Version	Accuracy	Execution time
Basic FC	96%	115.84 s
Norm	97%	122.81 s
Larger FC	97%	123.26 s
Larger Norm	98%	121.60 s
Deeper Norm	98%	128.14 s
Basic CNN	98%	128.88 s
Stride CNN	98%	128.40 s
CNN kernel=5	99%	134.73 s
CNN different kernels	99%	128.32 s
AvgPool CNN	96%	131.29 s
Dilation CNN	99%	132.14 s
Base CNN lr = 0.0001	89%	133.44 s
Base CNN lr = 0.1	99%	136.80 s
Base CNN momentum	99%	136.80 s
Base CNN Adam	99%	128.17 s
Scheduler	99%	131.87 s
Batch 32	99%	137.79 s
Batch 128	99%	126.62 s

D. Assignment 2

Now I move to CIFRAR10 dataset in order to verify the results I got, I adapt the CNN I used as baseline, in order to accept 32*32 images. As before, at the end of this section i will put a table with all the results for accuracy and training time. If not said explicitly, the parameters I will use are: Number of epochs = 20, weight decay = 1e-4, lr = 0.01, optimizer = SGD with momentum = 0.9, batch size=64. In order to improve the results I compute mean and std of CIFRAR10 dataset simply computing mean and std of each element of the trainset and use the results in order to do a normalization on dataset. The results I get now are much worse: The training is much slower, I only reach 64% in accuracy and the loss curves are diverging, in fact the test loss curve is growing and the training is decreasing, it suggests that the architecture is overfitting.

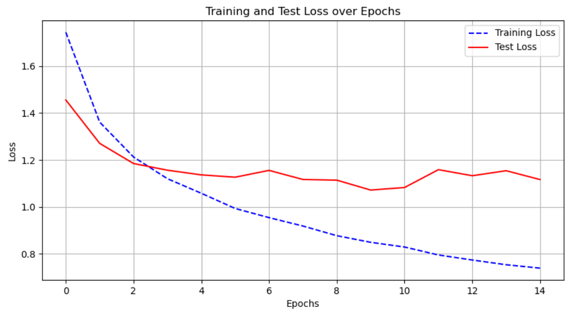

Figure 1. Loss plot of CNN Baseline

Now I apply some Schedulers: MultiStepLR and CosineAnnealingLR on my baseline. The accuracy grows and the overfit effect seems to be less strong. I train even Resnet18 on CIFRAR10, the accuracy test score are higher than the last cases but the model overfits.

To overcome I apply Augmentation, specifically Random Crop and Random Horizontal Flip.

The Baseline now does not overfit anymore, even if the accuracy does not increase much.

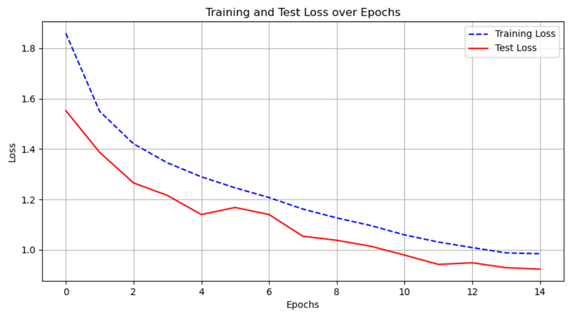

Figure 2. Loss plot of CNN Baseline after applying Augmentation

I try to use a FC architecture with 10 linear layers and normalization bach layers. It's really slow and less performing than the baseline. The worst part is that the class accuracies are not balanced, half o classes has an accuracy of 30% or 40%, even if trained with CosineAnnealing scheduler. Next thing is training a custom CNN with 2 blocks, each block uses 2 con2d layers, one batch nornmalization, one maxPool amd a dropout. The execution time are really long but the accuracy of class is really good: Only one class has accuracy test of 68%, all other are near 80% , I use CosineAnnealing scheduler.

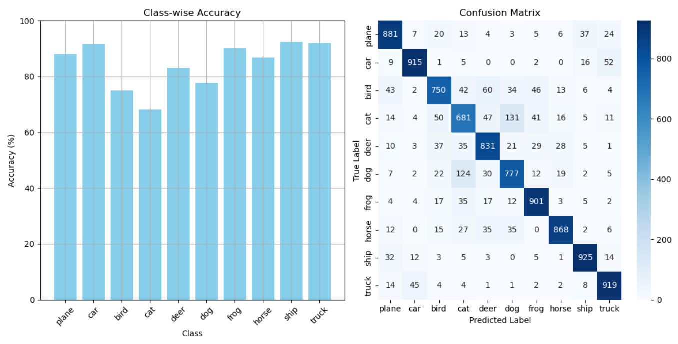

Figure 3. Class Accuracy of Custom CNN

Now using the pretrained Resnet18 with CosineAnnieling I finally have some class accuracy score that are balanced, only one class has 64%, all the other are around 80% .

After that I even train VGG16 and Alexnet in pretrained version. In order to work with VGG16 I simply change the last linear layer so that the output is 10, the number of classes. For Alexnet I changed the first MaxPooling.

The results of the pretrained nets are better then the baseline. What I see is that Resnet18 accuracies per classes are more balanced then VGG16 and Alexnet so from now on I will work with the baseline and Resnet18. Here the Loss curve of Resnet18 trained on augmented data.

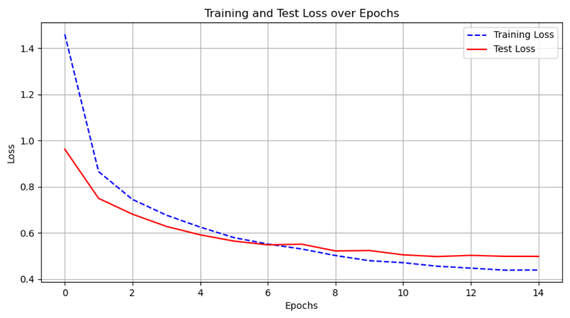

Figure 4. Loss Plot of Pretrained Resnet18 with Augmentation

Now I test the training of baseline using more transformation for augmentation: I added Color Jitter and RandomRotation. What I notice is that the execution time really increased 1382.75 s and accuracy is 67% . In general training an architecture using transformation too much complicated does not bring a good improvement, slowing only the training process I will only use RandomHorizontalFlip and RandomCrop.

Not Augmented:

Model	Accuracy	Execution time
CNN baseline	64%	213.37 s
Baseline MultiStep	66%	204.66 s
BaselineCosineAnnealing	67%	219.40 s
ResNet18	81%	300.29 s
ResNet18 FineTuned	68%	256.33 s

Augmented:

Model	Accuracy	Execution time
BaselineCosineAnnealing	68%	280.37 s
FC CosineAnnealing	51%	1333.23 s
Custom CNN	85%	1650.51 s
ResNet18 Pretrained	83%	338.31 s
VGG16 Pretrained	82%	719.33 s
Alexnet Pretrained	80%	391.08 s

What I can conclude is that when my architecture overfits, a good solution consists in data augmentation. The transformation I select must be chosen accordingly with the problem, if my transformation are too much difficult, like Color Jitter, the training process could become really slow and bring no real improvements to performances. In general selecting a scheduler, in order to change dinamically the learning rate, enhance the performances, CosineAnnealing gives this chance. Using a model from Pytorch, is a good solution, in this case the accuracies are much higher the baseline architecture even if they are slower to train.

E. Assignment 3

Again I will put at the end of this section a table with all the results. Here I am going to use a Resnet18 pretrained in order to fine tune it and see the results. The augmentation I tested are: RandomHorizontalFlip(p=0.5), transforms.RandomCrop(32, padding=4). If not explicitly said, the parameters I am gonna use are: number of epochs = 15, batch = 64, lr = 0.01 .

At the beginning I start training Resent 18 with SGD optimizer on my dataset: Accuracy is 83% and

I change the fully connected layer of the net and repeat. The accuracy is 42% and the pattern of loss seems to grow in last epoch. The class accuracy is unbalanced and low. What I want to do now consists in changing the last fc layer into a block with 2 different linear layers divided by a LeakyRelu and a Dropout, I change the optimizer: I use

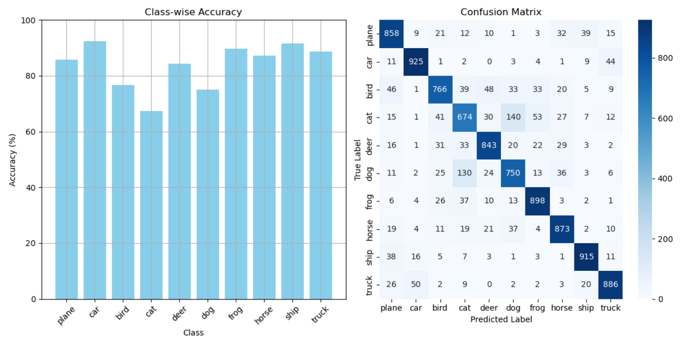

Figure 5. Class Accuracy of Resnet18

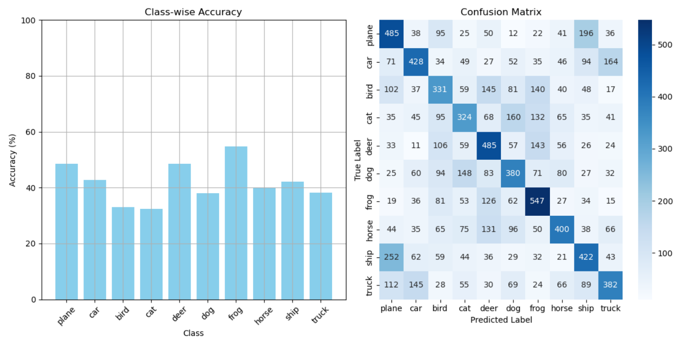

Figure 6. Class Accuracy of Resnet18 Fine Tuning FC layer

Adam and lr=0.0001, using CosineAnnealing as scheduler. Accuracy is 85% and class accuracy is quite balanced, only one class near 66%, all other are near 80% . Now the class

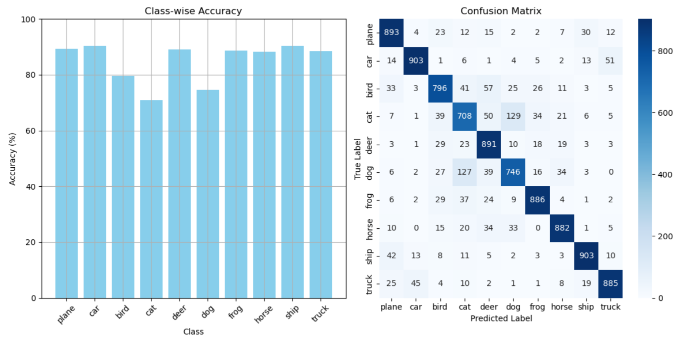

Figure 7. Class Accuracy of Resnet18 Varying FC layer

accuracy is really similar to trained Resnet18.

The next modification consists in training layer 4 and fc with different lr. I repeat the test in different conditions: Adam optimizer with fc lr = 0,1 and l4 lr = 0.001; fc lr = 0.01 and l4 lr = 0.001 and fc lr = 0.001 and l4 lr = 0.0001. The best results I record are with L4 lr = 0.0001 and fc lr = 0.01. As accuracy I get 66% .

Now I repeat fine tuning layer 2 and fc because I want to discover the effect of fine tuning layer at different depths of the net. In the same configuration the accuracy is lower, 60% .

After more experiments I understand that I need to change something in order to improve the performances so I apply a resize to my transformation that now are: transforms.Resize(224), transforms.RandomCrop(224, padding=4), transforms.RandomHorizontalFlip(), transforms.ToTensor(), transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)). Clearly I use in test only resize, normalization and transformation to tensor. This will increase the computational cost, so i will move this tests from my pc to colab.

I start training a pretrained Resent18 changing only the last linear layer in order to give 10 classes in output: Results are clearly different. The max accuracy recorded is 95% and class accuracy is really balanced. Even if the execution time increased a lot. As optimizer I use SGD.

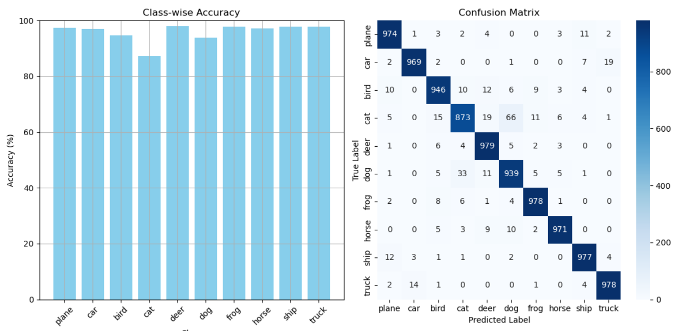

Figure 8. Class Accuracy of Resnet18 Fine tuning only FC layer with Resize to Data Augmentation

Fine tuning the last layer let me score 81% in accuracy, the class accuracy are unbalanced and the execution time is lower then training the pretrained net.

Then I try to change the last layer in a sequential of 2 linear layers separated by a Leaky Relu and a Dropout. The accuracy is 95% and accuracy for each class is not under 90% . As optimizer I use Adam with lr=0.0001.

Keeping this new fc layer I fine tune the fc and layer 4 using Adam with different lr: For fc 0.01 and for Layer4 0.0001 . The accuracy is 95% but class accuracy is not well balanced as training the full resnet18. I repeat even with SGD but the results are way worse with the same lr: 88% . So i decide to change the learning rate of the case fc + layer 4: I increase and decrease by a factor 10 the last learning rates but the results are worse.

I repeat the same experiment fine tuning: fc + layer 2 and fc + layer 1. But the accuracy are worse than the case fc + layer4: 91% in the two test case.

Last test is doing an early stop on the best results I have recorded: Pretrained resnet18 and, fine tuned fc+layer4 and fine tuned fc: I record the same results at slightly different epochs: For resnet18 pretrained the execution stops at epoch 20, in the other case at epoch 13. The result of accuracy are the same.

No resize:

Fine Tuned Layers	Execution	Accuracy
No Fine Tuning	273.76 s	83%
FC	230.09 s	42%
FC modified	275.36 s	85%
FC + L4	285.30 s	66%
FC + L2	267.63 s	60%
FC + L1	248.40 s	62%

With Resize:

Fine Tuned Layers	Execution	Accuracy
No Fine Tuning	2266.31 s	95%
FC	1553.25 s	82%
FC modified	2257.74 s	95%
FC + L4	2255.91 s	93%
FC + L4 on SGD	2243.12 s	88%
FC + L4 Lower LR	2279.74 s	91%
FC + L4 Bigger LR	2136.31 s	90%
FC + L2	2229.09 s	91%
FC + L1	1855.24 s	91%

F. Conclusions

I tested on Cifrar10 some custom networks, some models given from Pytorch and some fine tuned architectures. What I can conclude from my experiments are that a custom network adapted for my task and dataset is generally faster in terms of training time even if the results of the inference can be not accurate as the results of a pretrained or fine tuned model. Clearly a pretrained model is trained on other datasets so it means that if the new data are not similar enough to the original data given to the model, it will not perform properly. In this experiment I noticed that training a pretrained model, even if requires more time, brings better or at least equal results as fine tuning it even if training a fine tuned model is a faster process. About fine tuning I can say that changing few layer brings better results especially if they are the deeper layers of the model.