Computer Vision practical work

Joakim Olsen

May 2019

Introduction

For the practical work, I have first worked a bit with advanced topologies with the cifar data, to see if it was possible to improve on the results optained from the project of RNA, and to gain a better understanding of how these topologies work. Then I have tried to use these topologies to solve the gender recognition problem, to obtain 1) test accuracy above 95 % and 2) test accuracy above 90 % with fewer than 100k parameters. Finally, I have played around a bit with style transfer, to get a better understanding of how it works, as well as making some cool images.

Residual networks with CIFAR data

In this part we will try to fit different types of residual networks and dense networks to the CIFAR-10 data. The motivation for making residual networks is because the shortcuts make training deeper networks possible. We will therefore first try to make the standard CNN deeper to see what happens, and then add the shortcuts to see what happens.

The starting point for this analysis will be a convolutional network inspired by a ResNet-implementation for CIFAR-10 [1], since the structure proposed in the class slides was made for the ImageNet dataset, and can thus not be expected to be ideal for the CIFAR-10 data. The starting point will be a standard convolutional network, with an initial 3x3 convolution with 16 filters, followed by BN and relu activation (except for the densenet, where preactivation is used). Then a set of blocks follows, where one block consists of two 3x3 convolutional filters with BN and relu, and stride=2 will be applied when the numbers of filters increases. Finally, average pool of size 8 will be added, before a dense layer with 10 nodes (since there are 10 classes) with softmax activation is applied.

Batch size is 128, and the Adam optimizer is used with step lenght 1e-3 for epoch 1-24, 1e-4 for 25-49 and 1e-5 for the rest, 75 epochs in total. There is a simple data augmentation with horizontal flip and height/width shift range equal to 0.1. This configuration is run with different number of blocks as described in the table below, and then different structures are added to create the advanced topologies. The implementation can be read in the python-code, and the result is shown in the Table 1 below:

Layer blocks	Standard CNN	Residual Network	Bottleneck
3x16, 3x32, 3x64	12.29 %, 272k p	11.14 %, 273k p	12.36 %, 222k p
6x16, 6x32, 6x64	14.36 %, 566k p	10.28 %, 566k p	10.34 %, 435k p
9x16, 9x32, 9x64	20.48 %, 859k p	9.77 %, 858k p	10.36 %, 649k p
12x16, 12x32, 12x64	28.90 %, 1147k p	10.16 %, 1150k p	10.43 %, 863k p
Layer blocks	Wide, k=2	Wide, k=4	Wide w/ dropout, k=2
3x16, 3x32, 3x64	8.46 %, 1082k p	7.06 %, 4303k p	7.45 %, 1082k p
6x16, 6x32, 6x64	7.91 %, 2253k p	7.05 %, 8956k p	7.38 %, 2253k p
9x16, 9x32, 9x64	7.86 %, 3412k p	6.99 %, 13609k p	7.30 %, 3412k p
12x16, 12x32, 12x64	7.69 %, 4589k p	9.26 %, 18262k p	7.46 %, 4589k p
Layer blocks	Dense	Dense-Bottleneck	Dense B-Compression (0.5)
3 blocks, 4 layers	12.50 %, 217k p	11.05 %, 231k p	11.27 %, 183k p
3 blocks, 10 layers	7.91 %, 1255k p	7.41 %, 946k p	8.10 %, 683k p
3 blocks, 16 layers	6.90 %, 3154k p	**6.13 %**, 2106k p	6.76 %, 1454k p
3 blocks, 8 layers, k=2	7.51 %, 3060k p	6.35 %, 2486k p	6.67 %, 1846k p

Table 1: Test error with the CIFAR-10 data using different topologies.

Conclusion

First of all, we see that increasing the depth in a standard convolutional network doesn't really work. This is expected, there is the vanishing gradient problem, that makes training deep networks without shortcuts difficult.

We also see that when adding shortcuts for the residual network, the performance is improved a lot. Using a bottleneck architecture can decrease the number of parameters some, with a performance that is almost as good.

We see that using wide residual networks gives very good performance, but also uses very many parameters. Dropout can improve the performance of wide residual networks. We also see for the wide residual networks that there is not that much to gain in training very deep networks, which is fortunate since the number of parameters grows very fast. Thus, wide and shallow networks seems to work well on the CIFAR data, but it is still possible to improve the performance by making the networks deeper, if many parameters is not a problem.

Finally, densenet seems to work very well. The number of parameters grows much slower with densenet, but the network need to be deeper to have a good performance. It takes more time to train when it's deeper, even though there are much fewer parameters. Adding bottleneck to densenet improves the performance as well as reducing parameters, and this is the network that achieves the best error in my results, 6.13 %. I have also tried a configuration with a "wide denseNet", which is not so deep, but with a growth rate multiplied by 2. We see that this performs very well too, but it has quite a lot of parameters, similar to wideResNet. It trains faster however, since it is not so deep, so that is a nice feature. Adding compression can further reduce the number of parameters without losing that much performance. I added very high compression, which makes the difference in performance notably worse.

These results are still with the same data augmentation and 75 epochs, so I expect we would have been able to improve the performance further by playing around with these configurations. All in all, we have been able to implement networks that by far outperform the networks implemented in RNA last semester. It seems the advanced topologies definitely are good to use.

Gender recognition

For this task, I will try to use some of the different topologies explored in the previous task, and see how they work on a very different problem. First, I will implement a standard convolutionary network, similar to the one used in the end of RNA, as described below. Then I will use a wideResNet, and finally a denseNet. The main goal is to get test accuracy > 95 %, as well as test accuracy > 90 % with fewer than 100k parameters. Based on the performance from the last task, I expect denseNet to be the best candidate to achieve the goal with few parameters, since a low growth rate gives few parameters, and

then a deeper network can be trained with few parameters. Batch size, data augmentation, epochs and learning rate is as in the last task. The process is described below in:

1. Standard convolutional network, init 16@3x3 with stride=2, 2x32, 2x64, 2x128, 2x256 and 2x512 conv layers. Finally dense + softmax.

2. WideNet, K=2, dropout=0.1, init 16@3x3, stride=2, 3x16, 3x32, 3x64 blocks and average pool size 8. Finally dense+softmax.

3. DenseNet-BC, growth rate=16, compression=0.8, init 16@3x3 stide=2, 3 blocks of 10 layers each. Then average pool size 8, and dense+softmax.

4. Reducing growth rate to 8.

5. Reducing growth rate to 4.

6. 12 layers per block instead of 10.

7. 8 layers per block instead of 12, and growth rate=6 instead of 4.

8. Increasing height/width shift range to 0.2.

9. Adding featurewice center/normalization.

Run	Error	Parameters
1	2.27 %	4727k
2	1.59 %	1084k
3	1.81 %	832k
4	2.34 %	229k
5	3.25 %	69k
6	3.1 %	91k
7	2.53 %	97k
8	2.19 %	97k
9	**1.51** %	97k

Table 2: Summary of performance history for networks on gender data.

Conclusion

We see that the standard convolutional network used in the last subject with the CIFAR data already achieves the goal of > 95 % accuracy. This network was optimized to work with the CIFAR-10 data, and achieved good results. The network has very many parameters and does not seem like a good start to achieve similar results with few parameters. In the last task, we managed to improve the performance on the CIFAR-10 data using advanced topologies, so we further explore how they work on the gender data.

As expected, the wideResNet perform very well. This is a shallow network, and thus it doesn't have too many parameters, but it still performs very well, and has the best performance with test error 1.59 % before optimizing data augmentation. It still has quite a lot of parameters, and since it is already so shallow, this might not be the best starting point for reducing number of parameters.

The performance of denseNet is quite similar to the wide residual network, even though it has fewer parameters. The initial run also has quite high growth rate, so there is much more potential to reduce the number of parameters. Indeed we see that by lowering the growth rate, the number of parameters drops drastically, and the performance only drops a litte. Some adjusting in the depth of the network and growth rate gives a final configuration with less than 100k parameters that performs very well.

After playing around with data augmentation, we get a final error equal to 1.49 %, which is actually our best result, and very good considering our goal of > 90 %. It would probably be possible to get better results with more parameters and data augmentation as well. This result could also possibly be improved more by using more epochs, but that takes time and does not really give much more insight. All in all, the goal is achieved, and we see again that the advanced topologies, and especially denseNet, performs very well compared to the number of parameters it has.

Style transfer

So lets add some color to this otherwise very black and white report. Figure 1 shows my hometown.

Figure 1: My hometown.

It looks kind of nice and calm. Lets try and make it more dramatic, by mixing it with Scream of Edvard Munch, shown in figure 2:

Figure 2: A dramatic painting, Scream by Edvard Munch.

The result after performing the style transfer is shown in figure 3.

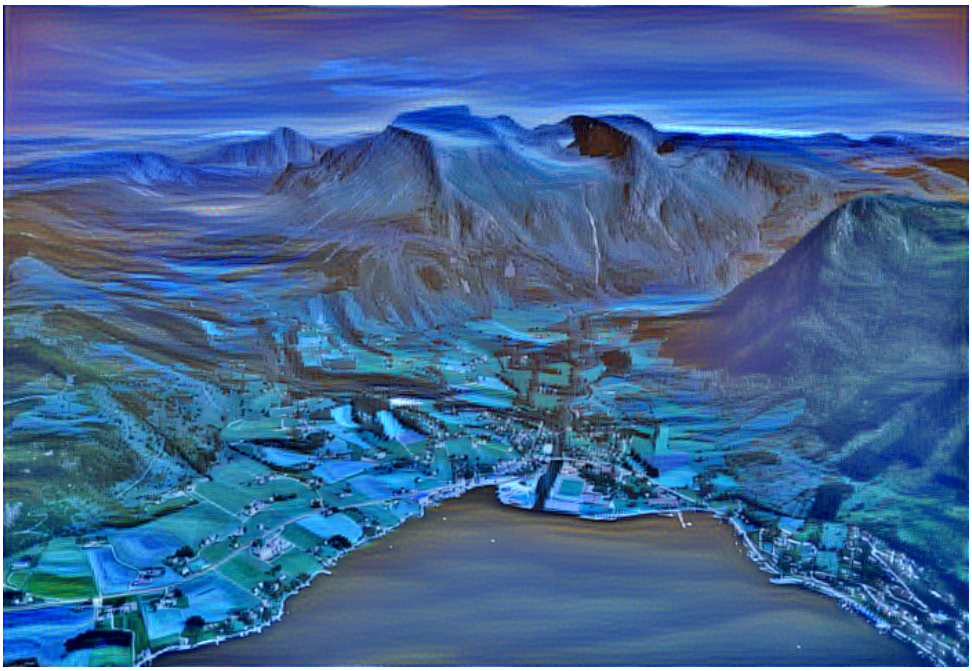

Figure 3: My hometown after performing style transfer from Scream by Edvard Munch.

So the result looks quite cool, it kind of looks like my hometown painted in the same style as the painting, which is what we expect. The implementation in python is attached, but not really changed much from the code posten on the github for the CV lab. The main changes are the image size and the weights, where the content weight is set to 0.1 and the style weight is set to 20.0, while the variation weight is 1.0 as before. So how does it work?

The main idea is based on running the images through a pretrained CNN, and measure the distance between the output and the input images in given layers. A VGG16 pretrained network is here used. For the content layer we should choose block 2, conv 2 according to Johnson et al. [2], as shown in the code. For the style, several layers are chosen, and changing these can make some variations. The content loss is defined as the distance between the input content image and the output image, while the style loss is is the distance between the style image and the output image. For the style, the gram matrix is used, and the total loss is summed over the layers used for the style. There is also a variation loss, working as a spatial smoother. Then the weights are used to define a total loss, and the BFGS optimization algorithm is used to minimize the loss. This is done iteratively for as long as requested, so the output converges to an image which has a short distance to the content image in the content layer, and short distances between the style image and the output in the style layers. The result is as we can see, an image that has much of the content of the content image, but the style of the style image, and this combination can be used to make some cool images.

References

[1] Ruiz, P.R. *ResNets for CIFAR-10*
https://towardsdatascience.com/resnets-for-cifar-10-e63e900524e0

[2] Surma, G. *Style Transfer - Styling Images with Convolutional Neural Networks*
https://towardsdatascience.com/style-transfer-styling-images-with-convolutional-neural-networks-7d215b58f461