TDT4265: Computer Vision and Deep Learning

Assignment 3

Håkon Hukkelås
hakon.hukkelas@ntnu.no
Department of Computer Science
Norwegian University of Science and Technology (NTNU)

February 15, 2019

- **Delivery deadline: February 28, 2019, by 2359.**
- **This project count towards 6% of your final grade.**
- You can work on your own or in groups of up to 2 people.
- Upload your code as a single ZIP file.
- Upload your report as a PDF file to blackboard.
- You can choose what framework to use in this assignment. We recommend Pytorch 1.0, but you are also free to use tensorflow. Pytorch is the framework we will give assistance to, but the student assistants might be familiar with keras/tensorflow as well.
- You will have access to use GPU resources on the Google cloud platform and the computers in Tulipan.
- The delivered code is taken into account with the evaluation. Ensure your code is well documented and as readable as possible.

Introduction

In this assignment, we will perform image classification of images in the CIFAR10 dataset. This assignment will focus on pytorch, but other DL frameworks are allowed to use. Pytorch is a deep learning framework which provides two main features: tensor functions similar to numpy that are able to run on GPUs. It also provides automatic differentiation for building and training neural networks, such that backpropagation is done automatically.

Recommended readings

We recommend you to review the lectures slides to get a brief overview of convolutional neural networks before starting this assignment. In addition, these are some recommended readings to get you started on the assignment. There is also given an example of a convolutional neural network implementation in Pytorch in the assignment code.

1. Nielson's book, Chapter 6.
2. Standford cs231n notes on convolutional neural networks
3. Justin Johnson's introduction to Pytorch

Computing the output shape of convolutional layers

This section will give you a quick overview of how to compute the number of parameters and the output shapes of convolutional layers. For a more detailed explanation, look at the recommended resources.

A convolutional layer takes in an image of shape $\mathbf{H_1} \times \mathbf{W_1} \times \mathbf{C_1}$, where each parameter corresponds to the height, width, and channel, respectively. For CIFAR-10, this is $32x32x3$, where the number of channels, C_1, is the number of color channels we have(RGB).

The output of a convolutional layer will be $H_2 \times W_2 \times C_2$. H_2 and W_2 depend on the receptive field size($\mathbf{F}$) of the convolution filter, the stride at which they are applied($\mathbf{S}$), and the amount of zero padding applied to the input($\mathbf{P}$). The exact formula is:

$$W_2 = (W_1 - F_W + 2P_W)/S_W + 1, \tag{1}$$

where F_W is the receptive field of the of the convolutional filter for the width dimension, which is the same as the width of the filter. P_W is the padding of the input in the width dimension, and S_W is the stride of the convolution operation for the width dimension.

For the height dimension, we have a similar equation:

$$H_2 = (H_1 - F_H + 2P_H)/S_H + 1 \tag{2}$$

where F_H is the receptive field of the of the convolutional filter for the height dimension, which is the same as the height of the filter. P_H is the padding of the input in the height dimension, and S_H is the stride the convolution operation for the height dimension. Finally, the output size of the channel dimension, $\mathbf{C_2}$, is the same as the number of filters in our convolutional layer.

Simple example: Given a input image of $32x32x3$, we want to forward this through a convolutional layer with 32 filters. Each filter has a filter size of 4×4, a padding of 2 in both the width and height dimension, and a stride of 2 for both the with and height dimension. This gives us $W_1 = 32, H_1 = 32$, $C_1 = 3$, $F_W = 4, F_H = 4$, $P_W = 2, P_H = 2$ and $S_W = 2, S_H = 2$. By using Equation 1, we get

$W_2 = (32 - 4 + 2 \cdot 2)/2 + 1 = 17$. By applying Equation 2 for H_2 gives us the same number, and the final output shape will be $17 \times 17 \times 32$, where $W_2 = 17, H_2 = 17, C_2 = 32$.

To compute the number of parameters, we look at each filter in our convolutional layer. Each filter will have $F_H \times F_W \times C_1 = 48$ number of weights in it. The total convolutional layer will have a total of $F_H \times F_W \times C_1 \times C_2 = 1536$ weights. The number of biases will be the same as the number of output filters, C_2. In total, we have $1536 + C_2 = 1568$ parameters.

Note: If we had used a $F_W, F_H = 5$, we would get 16.5, i.e. not an integer, which means that the filter does not "fit" on the input. This is something to consider when designing neural networks, and most modern frameworks round down these floating points. Pytorch would in this case output $16 \times 16 \times 32$.

Typical convolutional kernels: In today's practice, we generally use very small convolutional filters and the most popular shape is 3×3, but 5×5 and 7×7 are also seen. For stride and padding, we generally use the same size for these. Strides are often 1, but something called "strided convolutions"($S > 1$) are also popular to reduce the output shape. Padding is often used to keep the original input shape. For example, with a convolutional kernel of 5×5 and a stride of 1, we use a padding of 2 such that the input shape is equal to the output shape in width and height.

Pytorch shapes: The pytorch framework use a tensor with the default shape of $N \times C \times H \times W$ for an image, where N is the batch size, and expects this shape as input to a convolutional layer.

GPU resources

In this assignment, we will introduce you to training neural networks on GPU's. This class is limited in GPU resources, but you will have two sources of computing power. Mainly, the Google Cloud Platform, and the computers in Tulipan.

The reason we use GPU's is to train our networks faster, by utilizing data parallelization. For task 2 and 3, it will save you a lot of time, but it should not be needed for task 1.

Tulipan computers

There are 25 computers in Tulipan, all with powerful GPU cards (1080, 1080ti etc). We are 200 students in the class and you have to consider your fellow students when you are using the GPU resources. For this assignment, **we ask you to follow these rules when using the computers:**

1. Each group can only use a single computer at a time.
2. It is not allowed to remote access the computers in school time. This is the time 08:00-20:00 every weekday (Monday-Friday).
3. Before you start utilizing the GPU, check that no one is using it with the command "nvidia-smi".
4. The lab is reserved to another class every Friday 10:00-12:00. Please, let these students have the computers at this time.

Remote access through SSH

It is not allowed to remote access the computer in school time. To remote access the computers, you can use SSH. Use the command: "ssh [ntnu-username]@clab[00-25].idi.ntnu.no". E.g: ssh haakohu@clab00.idi.ntnu.no.

Pytorch on the tulipan computers

Pytorch 1.0 is installed on the tulipan computers and it should work out of the box with the GPU. Just launch python with "python3" in the terminal. If you want to install any additional packages, you can do this by installing it to your username. E.G: pip install −−user python_package_name.

Running tensorflow/keras on the tulipan computers

To run tensorflow or keras on the tulipan computers, you have to set the cuda version by using the command: cuda−ver 9.0

Then, you can run python3 and keras/tensorflow should work.

Google Cloud Platform

In the assignment files, there is provided a short introduction on how to sign up and get connected to the remote server. When you are connected, you can simply copy your files over and start training. To copy your files, we recommend you to use git.

Before adding the GCP coupon, remember to sign up on the GCP platform with the free link given in the gcp tutorial with the assignment files. This will give you a 300$ sign up bonus, with addition to the 50$ class coupon.

For the additional credit in this class, click on this link to request your GCP coupon. This will add 50$ to your GCP account. You will be asked for your name and your student email.

Remember: Do not use all of your credits on this assignment! Assignment 4 will require little to no GPU resources, but your final project will, probably, require a large portion.

1 Convolutional Neural Networks (0.75 points)

In this problem, you will design and train your own neural network with a deep learning framework to classify images from the CIFAR-10 dataset. The CIFAR-10 dataset consists of $60,000$ $32x32$ color images in 10 classes, with 6,000 images per class. There are 50,000 training images and $10,000$ test images.

To load the CIFAR-10 dataset, we have included a simple function to load the entire dataset(for pytorch) in the assignment code. Notice, in the code, we normalize our images with a certain mean and standard deviation by using pytorch transforms. This will improve the convergence of your network significantly. You can further add transforms to simply introduce data augmentation in your network(but this is not required).

The initial network you will implement is similar to one of the first succesfull CNN architechtures, the LeNet. It was originally designed for classifying handwritten digits(MNIST), but it works well as a starting point for the CIFAR-10 dataset. The network is described in Table 1. **Use a filter size of 5x5 with a padding of 2** for each convolution. A convolutional filter is the weight matrix that is used in the convolution operation; it is often referred to as a convolutional kernel as well. The **max pool operation should use a kernel size of 2x2 with a stride of 2.**

In your report, please:

(a) *(0.4 points)* Implement the CNN described in Table 1. Plot the training, validation, and test loss

Layer Type	Number of Hidden Units / Number of filters	Activation Function
Conv2D	32	ReLU
MaxPoolDd	–	–
Conv2D	64	ReLU
MaxPool2D	–	–
Conv2D	128	ReLU
MaxPool2D	–	–
Flatten	–	–
Dense	64	ReLU
Dense	10	Softmax

Table 1: A simple multi-layer convolutional neural network. Number of hidden units specifies the number of hidden units in a fully-connected layer. The number of filters specifies the number of filters/kernels in a convolutional layer. The activation function specifies the activation function that should be applied after the fully-connected/convolutional layer.

in a single plot.

(b) *(0.1 points)* Report your final training, validation and test accuracy.

(c) *(0.25 points)* How many parameters is in the network in Table 1? The number of parameters is the number of weights + the number of biases.

2 Deep Convolutional Network for Image Classification (2.5 points)

This task is an open-ended problem, where it's your job to design your own convolutional neural network, and experiment with architectures, hyperparameters, loss functions, and optimizers to **train two different models** that reach **at least 75% accuracy** on the CIFAR-10 **test set** within 10 epochs.

Recommended things to try out:

- **Filter size:** The starting architecture has 5x5 filters; would small filter sizes work better?
- **Number of filters:** The starting architecture has 32, 64 and 128 filters. Do more of less work better?
- **Pooling vs strided convolutions:** Pooling is used to reduce the input shape in the width and height dimension. Strided convolution can also be used for this ($S > 1$).
- **Batch normalization:** Try adding spatial batch normalization after convolution layers and vanilla batch normalization after affine layers. Do your networks train faster?
- **Network architecture:** The network above has two layers of trainable parameters. Can you do better with a deep network? Good architectures to try include:
  - (conv-relu-pool)xN $\rightarrow$ (affine)xM $\rightarrow$ softmax
  - (conv-relu-conv-relu-pool)xN $\rightarrow$ (affine)xM $\rightarrow$ softmax
  - (batchnorm-relu-conv)xN $\rightarrow$ (affine)xM $\rightarrow$ softmax
- **Regularization:** Add L_2 weight regularization, or perhaps use Dropout.
- **Weight initialization:** Try out Xavier Initialization to initialize your convolutional weights.

- **Optimizers:** Try out Adam optimizer

In your report, please:

(a) *(1.0 points)* Report your two models. Describe the network architecture (similar to Table 1) and include training details such as optimizer, regularization, learning rate, batch size, weight initialization and other details that are required such that a person reading it can closely replicate your results.

(b) *(0.5 points)* Include a table including the final train loss, training accuracy, validation loss, validation accuracy and test accuracy for the two models. This should all be in one table.

(c) *(0.25 points)* A plot of the training, validation and test accuracy vs the number of epochs(or vs the number of training steps) for your *best model.*

(d) *(0.25 points)* A plot of the training, validation and test loss vs the number of epochs(or vs the number of training steps) for your *best model.*

(e) *(0.5 points)* Discuss briefly what method you saw the most improvement with, what methods that did not work, and why you think these methods worked out or not. This should be in a paragraph or two.

3 Transfer Learning with ResNet (2.75 points)

Transfer learning aims at solving new tasks using knowledge from solving related tasks. In this assignment, you will learn how to apply variants of transfer learning for the task of image classification. Deep architectures of convolutional neural networks do not train well on small datasets. It is a common practice to use an existing model, trained on a very large scale dataset, as the feature extractor or initial layers of a ConvNet. Then, these layers can be fine-tuned to learn the underlying distribution of the dataset for our new task.

We will classify images in the CIFAR-10 dataset, using a pre-trained CNN model. The CNN model we will use is the Resnet18([He et al., 2016]) model, pre-trained on the ImageNet dataset, a very large scale visual recognition dataset.

The ResNet CNN model, originally trained to classify images into $1,000$ classes, will be used to classify images in the CIFAR10 dataset. The model revolutionized training of deeper networks, by introducing something called a residual block into the network. The residual block allows the gradient to flow through our network on a "highway", diminishing the problem of *vanishing gradients*. Further, it reduced the effect of the *degradation problem*, which is that if we make a network too deep, the accuracy saturates. With these improvements, they were able to train a network with 152 layers successfully, improving the previous state-of-the-art on ImageNet classification and object detection.

Here, we are not going to use 152 layers deep neural network due to computation limitations, but we will use a 18-layer CNN. We will utilize the pre-trained feature extractor(18 layers of convolutions) and replace the $1,000$-layer softmax with a fully-connected layer with 10 outputs. How to define your model in pytorch is shown in Code Listing 1. Notice the initial upsampling layer, which resizes the image to (256, 256) (this is the common size of images in ImageNet). To fine-tune our CNN on the CIFAR10-dataset, we freeze the initial convolutional layers and only update the weights in the last layers of our model. Specifically, we freeze all convolutional layers, except the last 5. This way we can fine-tune our last 5 convolutional layers and our final fully-connected layer.

If you want to use either Keras or tensorflow, there is currently no official pre-trained model for Resnet18. We recommend you to use VGG19 or ResNet50, which will show similar results on CIFAR10. If you use either, please state this in your report. Remember to replace the fully-connected layers in the model and only update the weights of the final convolutional layers.

```python
import torchvision
from torch import nn

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = torchvision.models.resnet18(pretrained=True)
        self.model.fc = nn.Linear(512*4, 10) # No need to apply softmax,
                                             # as this is done in nn.CrossEntropyLoss

        for param in self.model.parameters(): # Freeze all parameters
            param.requires_grad = False
        for param in self.model.fc.parameters(): # Unfreeze the last fully-connected
            param.requires_grad = True           # layer
        for param in self.model.layer4.parameters(): # Unfreeze the last 5 convolutional
            param.requires_grad = True               # layers

    def forward(self, x):
        x = nn.functional.interpolate(x, scale_factor=8)
        x = self.model(x)
        return x
```

Code Listing 1: Transfer learning with ResNet18 in pytorch.

In your report, please:

(a) *(0.5 points)* Implement transfer learning with Resnet18. Briefly report your hyperparameters, including optimizer, batch size, learning rate and potential data augmentation used. If you use keras or tensorflow, you can choose another pre-trained model with a minimum of 16 convolutional layers.

Hint: We recommend you to use the Adam Optimizer, with a learning rate of $5 \cdot 10^{-4}$ and a batch size of 32. It should converge within 5 epochs.

(b) *(0.3 points)* Plot training, validation and test accuracy vs the number of epochs.

(c) *(0.3 points)* Plot training, validation and test loss vs the number of epochs.

(d) *(0.4 points)* Plot training and validation loss for the transfer-learned model and your best model in the same plot. Discuss shortly what you observe.

(e) *(0.5 points)* Visualize filters(minimum 10) from the first convolutional layer of the trained model by passing an image through the filter. Plot the original image and its filter activation together. Shortly discuss what you observe from the filter activation (2-3 sentences).

The way you do this, is that you take the image you want to predict and instead of performing the full forward pass through the whole network, you only pass the image through the first convolutional layer. Then, you take this activation and plot it as an image.

Hint: The first conv layer is named "conv1" in the resnet18 model and you can access it through: torchvision .models.resnet18(pretrained=True).conv1.

Remember to resize your image to 256×256. This can be done with torch.nn.functional.interpolation. Then, the output shape of the convolution layer will be $1 \times 64 \times 128 \times 128$, since it has 64 filters and we use a batch size of 1.

To get the best visualization, you need to normalize your data similarly as we do in the data loader. You can do this by using the torchvision .transforms.functional .normalize(image, mean, std) function, with the same mean and standard deviation.

(f) *(0.25 points)* Visualize the filters(minimum 10) from the last convolutional layer of the trained model by passing an image through the filter. Plot the original image and its filter activation together. Shortly discuss what you observe from the filter activation (2-3 sentences).

This can be done the exact same way as previously, except that you need to forward the image through the whole network up until the last convolutional layer and show the activation.

Hint: You can visualize the whole network architecture by printing the model, print(model). You can access the different blocks of the network by using the function model.children(). The activation from the last convolutional layer is found by forward passing the image through all the modules returned by model.children(), except the last two. This will return the activation from the last convolutional layer after it is forward passed through a batch normalization layer, but this is fine.

The activation of the filter should have shape $1 \times 512 \times 8 \times 8$.

(g) *(0.5 points)* Visualize the weights of the filters in the first convolutional layer and discuss briefly what they represent.

Hint: If you are unsure of what the filter represents, you can visualize the activation of an image and see what the result of the filter is.

References

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.