

# 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 14, 2020

- **Delivery deadline: Friday, February 28, 2020, by 23:59.**
- **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 single PDF file to blackboard.
- You are required to use python3 to finish the programming assignments.
- 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.

With this assignment, we provide you starter code for the programming tasks. You can download this from:

<https://github.com/hukkelas/TDT4265-StarterCode>.

To set up your environment, follow the guide in the Github repo:

[https://github.com/hukkelas/TDT4265-StarterCode/blob/master/python\\_setup\\_instructions.md](https://github.com/hukkelas/TDT4265-StarterCode/blob/master/python_setup_instructions.md)

## 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. [Stanford cs231n notes on convolutional neural networks](#): This includes a thorough walkthrough of CNNs
3. [Justin Johnson's introduction to Pytorch](#)
4. [Pytorch Tutorials](#): Take a look at the "60 minutes blitz" to get a high level overview of Pytorch.
5. [Pytorch Docs](#): Even though documentation is boring, the pytorch documentation is a great resource to get familiar with the framework.

## Delivery

We ask you to follow these guidelines:

- **Report:** Deliver your answers as a **single PDF file**. Include all tasks in the report, and mark it clearly with the task you are answering (Task 1.1, Task1.2, Task 2.1a etc).
- **Plots in report:** For the plots in the report, ensure that they are large and easily readable. You might want to use the "ylim" function in the matplotlib package to "zoom" in on your plots. Label the different graphs such that it is easy for us to see which graphs correspond to the train, validation and test set.
- **Source code:** Upload your code as a zip file. In the assignment starter code, we have included a script (`create_submission_zip.py`) to create your delivery zip. **Please use this**, as this will structure the zipfile as we expect. (Run this from the same folder as all the python files).  
To use the script, simply run: `python3 create_submission_zip.py`
- **Upload to blackboard:** Upload the ZIP file with your source code and the report to blackboard before the delivery deadline.

## Late Delivery & Other Penalties

We have a strict policy on late deliveries. Any delivery submitted after the deadline will be limited in maximum score which you can achieve on the assignment. See "Course Information" on blackboard for more information.

Also, if you fail to follow our delivery guidelines we will **subtract 0.2 points** from your final score. This includes if you do not deliver your report as a PDF file, if you don't deliver your code as .ZIP file, or if you don't use the `create_submission_zip.py` to generate your .zip file.

## 1 GPU resources

In this assignment, we will introduce you to training neural networks on GPU's. Training neural networks on GPUs is extremely efficient, as GPU's are made for matrix multiplication (which is pretty much the only thing NNs perform). For task 3 and 4, utilizing our GPU resources will save you a significant amount of time.

This class is limited in GPU resources, but you will have three sources of computing power.

- Computers in the Tulipan Lab.
- GPU resources on <https://tdt4265.idi.ntnu.no>
- **Google Colab**: Jupyter notebook solution from Google including GPU resources. We will not include a tutorial for this; however, this short tutorial should get you started with Pytorch on Google Colab: <https://jovianlin.io/pytorch-with-gpu-in-google-colab/>
- Google Cloud Platform

### 1.1 Tulipan computers

There are 25 computers in Tulipan, all with powerful GPU cards (1080, 1080ti etc). We are almost 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 13:30-15:00 and every other Monday 14:30-1600 (starting Feb 17th). 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. You need to be on the school network to be able to connect. If you want to connect outside school network, you have to use a VPN. ([Innsida VPN guide](#)).

## 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.

## 1.2 TDT4265 GPU Server

This year we were lucky enough to get funding for a GPU server with 20 T4 GPUs. To get started on the server, you can take a look at the [front page](#) of the sever. Remember, to read everything there!

You need to be connected to eduroam or through [NTNU VPN](#) to get access to this website.

### 1.2.1 Connecting through ssh

You should be able to access the server using the ssh ip: `ssh [ntnu-username]@oppdal.idi.ntnu.no`. Use your ntnu username/password to connect.

### 1.2.2 Sharing the resource

Remember, we are almost 200 students in this course, and this server has only a single GPU. We ask that **each group only uses a single GPU at a time**. Also, when you request a resource, you will get this resource for X hours. If you find out you only needed the resource for 20 minutes, instead of X hours, please release the resource! You can do this by typing in `tdocker kill`.

**Time accounting:** We are tracking the amount of time you use on the cluster. If the wait time will be large at the end of deadlines, we might start prioritizing students that has used less resources. Also, we might cut off students completely if your usage is significantly higher than the rest of the class. However, we will give you a clear warning if we start to enforce this.

## 1.3 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.

The additional coupon application is still in progress. We will post a referral link for those interested at a later time.

# Convolutional Neural Networks

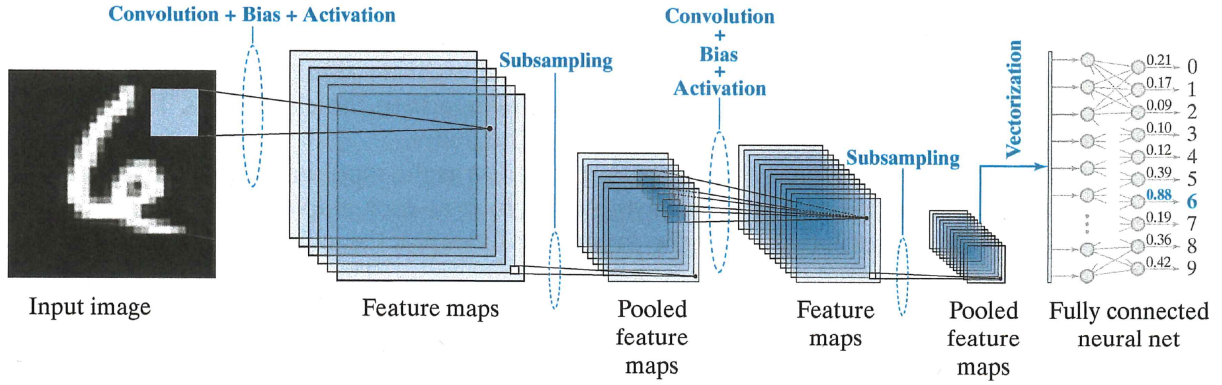


Figure 1: A CNN containing all the basic elements of a LeNet architecture. The network contains two convolutional layers, two pooling layers, and a single fully-connected layer. The last pooled feature maps are vectorized and serve as the input to a fully-connected neural network. The last fully-connected layer uses softmax as the activation function. Figure source: Chapter 12, Digital Image processing (Gonzalez)

The basic operations of CNNs are very similar to Fully Connected Neural Networks (FCNNs): (1) a sum of products is formed, (2) a bias value is added, (3) the resulting number is passed through an activation function, and (4) the activation value becomes a single input to a following layer. However, there are some crucial differences between these two networks.

A CNN learns 2-D features directly from raw image data, while a FCNN takes in a single vector. To illustrate this, take a close look at Figure 1. In a FCNN, we feed the output of every neuron in a layer directly into the input of every neuron in the next layer. By contrast, in a convolutional layer, a single value of the feature map is determined by a convolution<sup>1</sup> over a spatial neighborhood of the input (hence the name convolutional neural net). Therefore, CNNs are not fully connected and they are able to re-use parameters all over the image.

## Computing the Output Shape of Convolutional Layers

This section will give you a quick overview of how to find 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  $32 \times 32 \times 3$ , where the number of channels,  $C_1$ , is the number of color channels we have.

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:

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

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.

<sup>1</sup>Actually, in convolutional networks we use correlation instead of convolution; however, they are called to convolutional nets (most likely due to its connection to traditional signal processing).

For the height dimension, we have a similar equation:

$$H_2 = (H_1 - F_H + 2P_H)/S_H + 1 \quad (2)$$

where  $F_H$  is the receptive field 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 of the convolution operation for the height dimension. Finally, the output size of the channel dimension,  $C_2$ , is the same as the number of filters in our convolutional layer.

**Simple example:** Given a input image of  $32 \times 32 \times 3$ , we want to forward this through a convolutional layer with 32 filters. Each filter has a filter size of  $4 \times 4$ , a padding of 2 in both the width and height dimension, and a stride of 2 for both the width 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. From the previous example we know that 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.

## Task 1: Theory [1.5 points]

3	0	1	7	3
6	4	0	4	5
4	6	3	2	4

(a) A  $3 \times 5$  image.

0	1	0
1	-4	1
0	1	0

(b) A  $3 \times 3$  convolutional kernel.

Figure 2: An image  $I$  and a convolutional kernel  $K$ . For the image, each square represents an image pixel, where the value inside is the pixel intensity in the  $[0, 7]$  range (3-bit).

- (a) [0.3pt] Perform spatial convolution by hand on the image in [Figure 2a](#) using the kernel in [Figure 2b](#). The convolved image should be  $3 \times 5$ . You are free to choose how you handle boundary conditions, and state how you handle them in the report.
- (b) [0.2pt] CNNs are known to be invariant to small translational shifts in the input image. Of the following layers, which layer reduces the sensitivity to translational variations in the input?
  - (i) Convolutional layer
  - (ii) Activation function
  - (iii) Max Pooling
- (c) [0.1pt] Given a single convolutional layer with a stride of 1, kernel size of  $5 \times 5$ , and 6 filters. If you want the output shape (Height  $\times$  Width) of the convolutional layer to be equal to the input image, how much padding should you use on each side?

Consider a CNN whose inputs are RGB color images of size  $512 \times 512$ . The network has two convolutional layers. Using this information, answer the following:

- (d) [0.2pt] You are told that the spatial dimensions of the feature maps in the first layer are  $504 \times 504$ , and that there are 12 feature maps in the first layer. Assuming that no padding is used, the stride is 1, and the kernel used are square, and of an odd size, what are the spatial dimensions of these kernels? Give the answer as (Height)  $\times$  (Width).
- (e) [0.1pt] If subsampling is done after the first convolutional layer, using neighborhoods of size  $2 \times 2$ , with a stride of 2, what are the spatial dimensions of the pooled feature maps in the first pooling layer? Give the answer as (Height)  $\times$  (Width).
- (f) [0.2pt] The spatial dimensions of the convolution kernels in the second layer are  $3 \times 3$ . Assuming no padding and a stride of 1, what are the sizes of the feature maps in the second layer? Give the answer as (Height)  $\times$  (Width).
- (g) [0.4pt] [Table 1](#) shows a simple CNN. How many parameters are there in the network? In this network, the number of parameters is the number of weights + the number of biases. Assume the network takes in an RGB image as input and the image is square with a width of 32.

Table 1: A simple CNN. 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. Each convolutional layer has a **filter size of  $5 \times 5$  with a padding of 2 and a stride of 1**. The flatten layer takes an image with shape (Height)  $\times$  (Width)  $\times$  (Number of Feature Maps), and flattens it to a single vector with size (Height)  $\cdot$  (Width)  $\cdot$  (Number of Feature Maps) (*Similar to the `.view()` function in pytorch*). Each MaxPool2D layer has a **stride of 2 and a kernel size of  $2 \times 2$** .

Layer	Layer Type	Number of Hidden Units / Number of Filters	Activation Function
1	Conv2D	32	ReLU
1	MaxPool2D	–	–
2	Conv2D	64	ReLU
2	MaxPool2D	–	–
3	Conv2D	128	ReLU
3	MaxPool2D	–	–
	Flatten	–	–
4	Fully-Connected	64	ReLU
5	Fully-Connected	10	Softmax

## Task 2: Convolutional Neural Networks (0.5 points)

In this task, 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  $32 \times 32$  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. You can further add transforms to simply introduce data augmentation in your network (but this is not required).

For this task, we have included a standard boilerplate for you to start training and validating your CNN. This includes concepts covered in the previous assignments, such as input normalization, data shuffling, weight initialization<sup>2</sup>, and an implementation of mini-batch gradient descent (using SGD optimizer).

However, we have removed parts of the code and we expect you to complete the following functions:

- The function `compute_loss_and_accuracy`. This should iterate over a whole dataset and compute the average loss and the total accuracy.
- The function `__init__` for `ExampleModel`. This function should initialize all your weights and layers for your convolutional model.
- The function `forward` for `ExampleModel`. This should implement a forward pass through all convolutional layers and fully-connected layers.

**In your report, please:**

- (a) [0.4pts] Implement the CNN described in Table 1 and complete the functions described above. Train the network until convergence or early stopping kicks in.

In your report, include a plot of the train and validation loss over the whole training period.

*Hint:* If everything is implemented correctly, you should expect an accuracy between 68-72%.

<sup>2</sup>Pytorch implements a specific weight initialization (kaiming unifirom for `torch.nn.Conv2d` layers) by default. You can choose to override this for task3



- (b) [0.1pts] Report your final training, validation and test accuracy. Calculate this over the whole train/val/test datasets.

## Task 3: Deep Convolutional Network for Image Classification (2 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. **Train two different models** that reach **at least 75% accuracy consistently** on the CIFAR-10 test set within 10 epochs <sup>3</sup>.

### Recommended things to try out:

- **Data Augmentation:** Data augmentation is a simple trick to extend your training set. To use this, get familiar with the [torchvision transforms](#) abstraction.
- **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 or 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 1-dimensional batch normalization after fully-connected 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.
- **Optimizers:** Try out a different optimizer than SGD.
- **Activation Functions:** Try replacing all the ReLU activation function.

### In your report, please:

- (a) [1.0pts] 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) [0.25pts] 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.

Include a plot of the validation and test accuracy vs the number of training steps for your *best model*.

Include a plot of training, validation, and test loss vs the number of training steps for your *best model*.

---

<sup>3</sup>An "epoch" is one pass through the whole training set.

- (c) [0.25pts] 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.

For the method/technique that you saw the largest amount of improvement on, include a plot of the train and validation loss before/after applying this technique (Similar to what you did in task 3 for assignment 2). Remember to include this in the same graph!

- (d) [0.5pts] Improve on your best model and reach an accuracy of 80% within 10 epochs. Include a plot of validation and test accuracy over training.

## Task 4: Transfer Learning with ResNet (2 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 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*. Furthermore, 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 [Listing 1](#) 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.

---

```
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, 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 = self.model(x)
        return x
```

---

Listing 1: Transfer learning with Resnet18

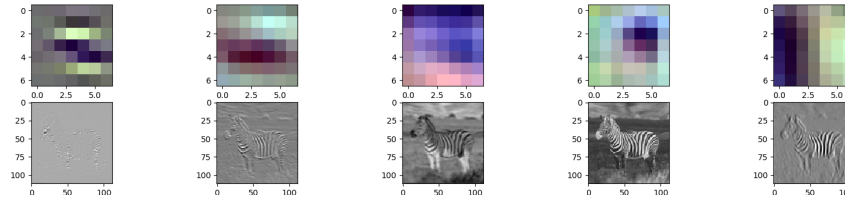


Figure 3: The filter weights (top row) and the activation to the corresponding filter (bottom row) of the first convolutional layer in ResNet18. This is visualized for the following indices: [0, 1, 2, 3, 14]

### In your report, please:

- (a) [1pt] Implement transfer learning with Resnet18. Briefly report your hyperparameters, including optimizer, batch size, learning rate and potential data augmentation used.

*Hints:*

- 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. You should expect an accuracy of 88% or more.
- You should include a `transforms.Resize(...)` in the dataloader pre-processing to resize the images to the correct size ( $224 \times 224$ ).
- You should change the mean and standard deviation that you use to normalize your images. For pytorch/ImageNet, the standard is to use the values shown here: <https://s.ntnu.no/pytorch-ImageNet-normalization>.

Include a plot of the validation/test accuracy vs the number of training steps. Also, include a plot of training, validation, and test loss vs the number of training steps.

- (b) [0.7pts] **We have included starter code for the remaining tasks in `task4b.py`.** Do the following tasks on the image `zebra.jpg`.

Visualize the filters from the first convolutional layer of the trained model by passing an image through the filter. Plot the values of the filter and the activation of that filter together. Visualize the filters with indices [14, 26, 32, 49, 52]. Shortly discuss what you observe from the filter activation (2-3 sentences). Figure 3 shows the visualized filters for a different set of indices.

More specifically, take the zebra image and instead of performing the full forward pass, only pass the image through the first convolutional layer (giving you the activation). Then, plot this activation 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`.

- (c) [0.3pts] Visualize the **ten first filters** from the last convolutional layer of the trained model by passing an image through the filter. You only need to plot the activation of the filter, and not the filter itself. 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 visualize 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<sup>4</sup>.

The activation of the filter should have shape  $1 \times 512 \times 7 \times 7$ .

<sup>4</sup>Actually, this will return the activation from the batch normalization layer after the last convolutional layer, but that's fine.

## 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.