Assignment 3

Computer Vision and Deep Learning
Mamoona Birkhez Shami
mamoona.b.shami@ntnu.no

Delivery Deadline:
Friday, March 4th, 23:59PM.
This assignment counts 4% of your final grade.
You are allowed to work in groups of 2 persons.

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, and it provides automatic differentiation for building and training neural networks, such that backpropagation is done automatically.

Compute 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. Take a look at our tutorials to get started with them:

1. The computers at campus (In the Tulipan/Cybele lab).
2. Our GPU cluster with 20 GPUs.
3. Or you can take a look at Google Colab [1].

Note that for each GPU resources, there are given rules, so **please read the tutorial carefully to ensure that you won't loose access.**

Starter Code. We provide you starter code for the programming tasks. You are required to use the provided files, but you are allowed to create any additional files for each of the subtasks. You can download the starter code from:
https://github.com/TDT4265-tutorial/TDT4265_StarterCode.

Report outline. We've included a jupyter notebook as a skeleton for your report, such that you won't use too much time on creating your report. Remember to export the jupyter notebook to PDF before submitting it to blackboard. You're not required to use this report skeleton, and you can write your report in whatever program you'd like (markdown, latex, word etc), as long as you deliver the report as a PDF file.

Recommended reading.

1. Nielson's book, Chapter 6.
2. Standford 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.

[1] We will not include a tutorial for this; however, this short tutorial should get you started with Pytorch on Google Colab: `https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/c30c1dcf2bc20119bcda7e734ce0eb42/quickstart_tutorial.ipynb`

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.a, Task1.b, Task 2.c etc). There is no need to include your code in the report.
- **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. Remember to include any additional files you have created in `files_to_include`).

  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.
- The delivered code is taken into account with the evaluation. Ensure your code is well documented and as readable as possible.

Any group who does not follow these guidelines or delivers late will be subtracted in points.

Task 1: Theory

For this task we include a short introduction to CNNs **at the bottom of the assignment**. Furthermore, I recommend you to take a look at stanfords cs231n's introduction.

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

(a) A 3×5 image.

-1	0	1
-2	0	2
-1	0	1

(b) A 3×3 Sobel kernel.

Figure 1: 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) [0.2*pt*] Perform spatial convolution by hand on the image in Figure 1a using the kernel in Figure 1b. The convolved image should be 3×5. You are free to choose how you handle boundary conditions, and state how you handle them in the report.

Solution:

-2	1	-11	2	13
-10	4	-8	-2	18
-14	1	5	-6	9

(a) 0 padding

0	4	-16	2	0
0	4	-8	-2	0
0	4	0	-6	0

(b) Reflection padding

Figure 2: **Solution**

(b) [0.15*pt*] CNNs are known to be invariant to small translational shifts in the input image (for example a 1-pixel shift). Of the following layers, which layer reduces the sensitivity to translationial variations in the input?

(i) Convolutional layer

(ii) Activation function

(iii) Max Pooling

Solution: Max Pooling

(c) [0.05*pt*] Given a single convolutional layer with a stride of 1, kernel size of 5×5, and 6 filters. If you want the output shape (Height × Width) of the convolutional layer to be equal to the input image, how much padding should you use on each side?

Solution: 2

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

(d) [0.1*pt*] You are told that the spatial dimensions of the feature maps in the first layer are 504×504, 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 number size, what are the spatial dimensions of these kernels? Give the answer as (Height) × (Width).

Solution: 9x9

(e) [0.1*pt*] If subsampling is done after the first convolutional layer, using neighborhoods of size 2×2, 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) × (Width).

Solution: 252x252

(f) [0.15*pt*] The spatial dimensions of the convolution kernels in the second layer are 3×3. 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) × (Width).

Solution: 250×250

(g) [0.25*pt*] Table 2 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.

Solution:

Conv 1 1	3*32*5*5 + 32 = 2,432
Conv 2	32*64*5*5 + 64 = 51,264
Conv 3	64*128*5*5 + 128 = 204,928
FC 1	4*4*128*64 + 64 = 131,136
FC 2	64 * 10 + 10 = 650
Total	390,410

Table 1: Solution

Table 2: 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** 5×5 **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** 2×2.

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

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 $60,000$ $32x32$ 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. 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 [2], and an implementation of mini-batch gradient descent (using SGD optimizer). Note that we are using train, validation and a test set for this assignment, just that you can check your final performance on the test set when you're done developing your model.

For this task, please:

(a) [0.35*pts*] Implement the CNN described in Table 2 and complete the following functions:

- The function `compute_loss_and_accuracy` in `trainer.py`. This should iterate over the entire dataset and compute the average loss and the total accuracy.
- The function `__init__` for `ExampleModel` in `task2.py`. This function should initialize all your weights and layers for your convolutional model.
- The function `forward` for `ExampleModel` in `task2.py`. This should implement a forward pass through all convolutional layers and fully-connected layers.

Train the network until convergence or early stopping kicks in. **(report)** In your report, include a plot of the train and validation loss over the training period.

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

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

[2]Pytorch implements a specific weight initialization (kaiming unfirom for `torch.nn.Conv2d` layers) by default. You can choose to override this for task3

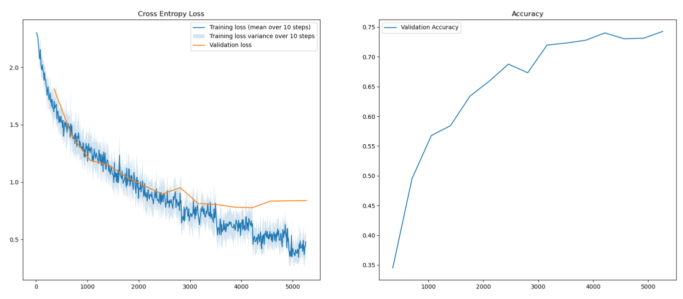

Figure 3: Training graph for task 2a. Train acc = 0.834, val acc = 0.74, test acc = 0.721. train loss = 0.485, val loss = 0.768, test loss = 0.819

Task 3: Deep Convolutional Network for Image Classification

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 [3].

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) [$0.6pts$] Report your two models. Describe the network architecture (similar to Table 2) 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.

[3] An "epoch" is one pass through the whole training set.

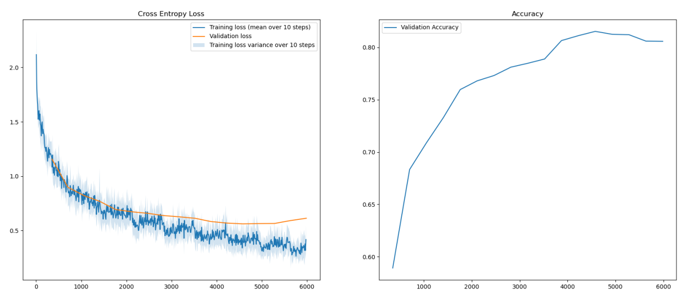

Figure 4: Training graph for task 3a. Train acc = 0.876, val acc = 0.8101, test acc = 0.823. Train loss = 0.362, val loss = 0.573, test loss = 0.55

(b) [0.1*pts*] Include a table including the final train loss, training accuracy, validation accuracy and test accuracy for the two models. This should all be in one table.

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

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

(c) [0.15*pts*] Discuss briefly what methods you found useful, and answer the following questions.

- Which method did you see the improvement with? Why do you think the network improved with this method?
- Which method did not work? Why did it not work?

(d) [0.15*pts*] 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!

(e) [0.35*pts*] Improve on your best model and reach an accuracy of 80% on the test set within 10 epochs. Include a plot of validation accuracy over training, and report your final test accuracy.

(f) [0.1*pts*] For your best model, do you see any signs of overfitting or underfitting?

Task 4: Transfer Learning with ResNet

Transfer learning aims at solving new tasks using knowledge from solving related tasks. Deep architectures of convolutional neural networks do not train well on small datasets, but transfer learning can often improve results with 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 training 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.

```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, 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

In your report, please:

(a) [0.5*pt*] 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 (224×224). Note that if training takes too long, you can train it on a resolution of 112×112 instead.
- 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: imagenet-normalization-implementation.

(report) Include a plot of training, and validation loss vs the number of training steps. Also, report your final test accuracy.

(b) [0.35*pts*] **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 6 shows the visualized filters for a different set of indices.

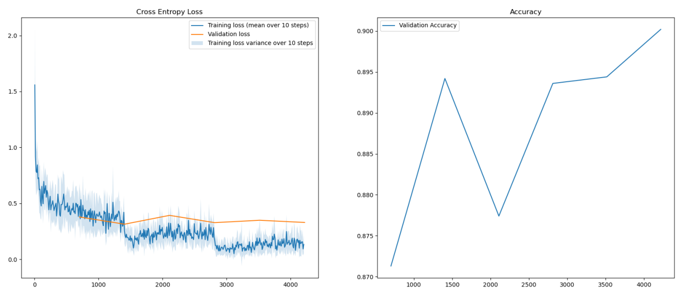

Figure 5: Training graph for task 4a. Train acc=0.93, val acc = 0.88, test acc = 0.887. Train loss = 0.212, val loss=0.362, test loss = 0.343

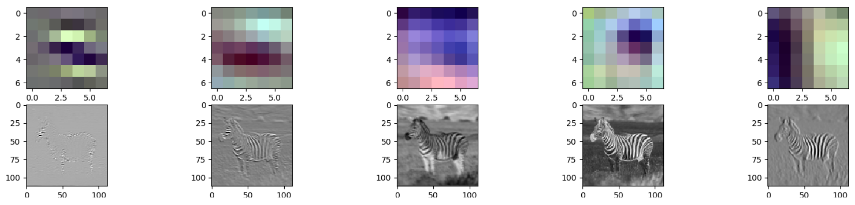

Figure 6: The filter weights (top row) and the activation to the corresponding filter (bottom row) of the first convolutional layer in ResNet18. This is visualized for the following indices: [0, 1, 2, 3, 14]

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. **(report)** Include your visualization in your report.

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

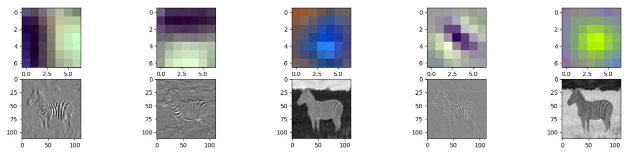

Figure 7: Solution for task 4b.

Solution:

- Filter index 14 detects vertical edges
- Filter index 26 detects horizontal edges
- Filter index 32 detects the color blue, like the sky
- Filter index 49 detects diagonal edges
- Filter index 52 detects the color green, like the grass

(c) [0.25*pts*] 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. **(report)** Include your visualization in your report, and 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 [4].

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

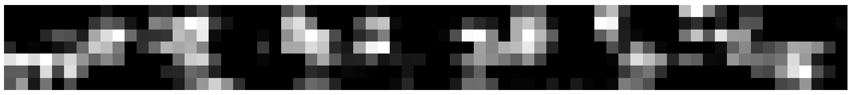

Figure 8: Solution for task 4c.

[4]Actually, this will return the activation from the batch normalization layer after the last convolutional layer, but that's fine.

A Brief Introduction to Convolutional Neural Networks

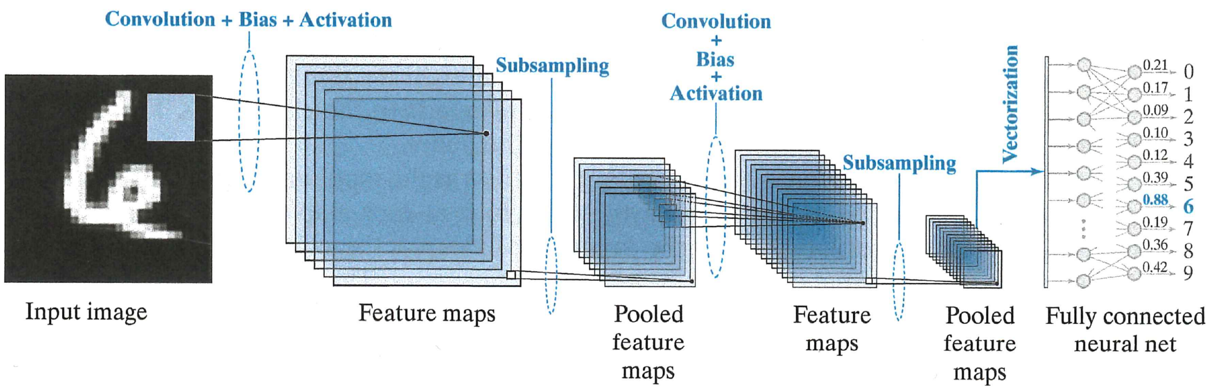

Figure 9: A CNN containing all the basic elements of a LeNet architecture. The network contains two convolutional layers, two pooling layers, and a single fully-connected layer. The last pooled feature maps are vectorized and serve as the input to a fully-connected neural network. The class to which the input image belongs is determined by the output neuron with the highest value. 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 9. 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 output is determined by a convolution 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 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. 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 (**F**) of the convolution filter, the stride at which they are applied (**S**), and the amount of zero padding applied to the input (**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. Including all filters in the convolutional layer, the 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 you get a SSL error like this: `[SSL: CERTIFICATE_VERIFY_FAILED]`

Add the following lines to `dataloaders.py` right after imports:

```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

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.