# ***Assignment 3***

# CNN, model training, Segmentation

Hui Xue

[hui.xue@nih.gov](mailto:hui.xue@nih.gov)

1. **Multi-choice and short-answer questions**
2. Often, we need to centered the data by subtracting the mean and scaling with the standard deviation. Is it a good idea to compute mean and SD from the entire training set, and then split the dataset to train and validation and apply the data centering?
3. Yes, it is the typical way.
4. No, we should do it differently
5. Neural, there are other ways, but they all are equal.
6. Data augmentation is often used to introduce variation into the training dataset. Which of the following are correct statement about data augmentation?
7. Data augmentation needs to be applied to entire training sets, so we can increase the training set size significantly.
8. Applying data augmentation will increase training accuracy.
9. Applying data augmentation will increase test accuracy.
10. Operations such as image rotation or translation should always be applied consistently to all samples.
11. Which of the following CONV setups will keep the image size?
12. Kernel 5x5, stride 2, padding 2
13. Kernel 5x5, stride 1, padding 2
14. Kernel 11x7, stride 1, padding 5x2
15. Kernel 11x7, stride 1, padding 5x3
16. Input to a CONV layer has the size [B, 32, 240,240] for batch, channel, H and W. The convolution layer has the setup: Kernel 7x7, stride 3, padding 4, number of kernels 48. What will be the output size of this CONV layers?
    * 1. [B, 32, 80, 80]
      2. [B, 48, 81, 81]
      3. [B, 32, 81, 81]
      4. [B, 48, 80, 80]
17. Which of the following layers can be use to upsample input image with learnable parameters?
    * 1. Stride CONV
      2. Transpose CONV
      3. Pooling
      4. Interpolation
18. Fully connected layer can be implemented with equal CONV layer. Suppose the input tensor has the size [B, 4096, 8, 8]. Before using the CONV layer, the input tensor was first flattened and go through a FC layer with 1024 neurons. Now you want to use CONV layer to replace the FC layer. What is the correct setup for this CONV layer?
    * 1. Kernel 1x1, number of filter 1024, stride 1, padding 1
      2. Kernel 8x8, number of filter 1024, stride 1, padding 1
      3. Kernel 8x8, number of filter 1024, stride 1, padding 0
      4. Kernel 1x1, number of filter 4096, stride 1, padding 0
19. Typically we split the data to train, validation/dev sets and have a hold-out test set. Explain in your language why sometimes we need a tra-dev set? What is data mismatch and how to measure its impact in training?
20. Consider the VGG16 net. This network was trained on ImageNet samples with input size[B, 3, 224, 224]. Now you want apply the network on images with different size, e.g. [B, 3, 128, 128]. Typically we don’t have a large dataset comparable to ImageNet. Possible strategies include: a) retrain the VGG16 net; b) keep the CONV layers in the VGG16 net and retrain all FC layers; c) Keep the CONV layers and most FC layers. Only retrain the FC layer immediately after the CONV. Explain what will you do and why?
21. In the ResNet, the residual connection is implemented as . Explain the motivation for this design and why does it allow to train much deeper networks?

In the U-net design, the long range skip-connection concatenates two inputs: x = torch.cat([x2, x1], dim=1). Why does U-net choose to concatenate, rather adding them up as the resnet does?

1. In the lecture, we mentioned the mobile-net and depth-wise convolution. Explain what is depth-wise convolution and how to implement it with group CONV? Compared to conventional CONV, does mobile-net design increase or decrease the model representation capacity? Describe the use cases where the mobile-net design may have advantages.
2. In the lecture, we discussed the 2D CONV in detail. Input to a 2D conv layer is a tensor [B, Cin, H, W]. The output has the size [B, Cout, H’, W’]. It will be useful to know the CONV layer can be extended to process 3D image, e.g. for a video series or for a medical imaging volume, where the image has the size [C, H, W, D]. For the 3D conv, the kernel is a 3D tensor [kx, ky, kd]. The computation of 3D conv is the identical to 2D case where the kernel slides over the entire volume or image. At every location, all pixels for all channels under the kernel was multiplied to the kernel on a point-by-point basis. The results are summed up as the output for this location. Specially, for the 3D CONV, input has the size [B, Cin, H, W, D] and output has the size [B, Cout, H’, W’, D’].

Similarly, the pooling can be extended to 3D. Instead of taking a 2x2 neighborhood as input to a pooling layer, the input for 3D will be 2x2x2 for H, W and D dimensions.

Now you are designing a 3D CONV network to classify a CT image volume. Input volume has the size [B, 1, 256, 256, 128], with one channel and depth being 128 (such, consisting of 128 2D images).

Note the ReLU layers inserted after each CONV are omitted from the table.

1. In the table below, fill in the output tensor size in the format of [B, Cout, H’, W’, D’] and compute the number of weights and biases:

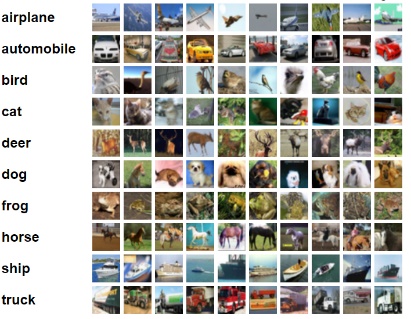
|  |  |  |  |
| --- | --- | --- | --- |
| Layers | Output dimensions | # kernel size or weights | # biases |
| Input | 1x256x256x128 | - | - |
| 3D CONV, kernel 5x5x3, number of kernels 32, stride 1, padding 2x2x1 |  |  |  |
| 3D pooling, 2x2x2 |  |  |  |
| 3D CONV, kernel 5x5x3, number of kernels 64, stride 2, padding 2x2x1 |  |  |  |
| 3D pooling, 2x2x2 |  |  |  |
| 3D CONV, kernel 3x3x3, number of kernels 128, stride 2, padding 1x1x1 |  |  |  |
| 3D pooling, 2x2x2 |  |  |  |
| FC layer, 4096 neurons |  |  |  |
| FC layer, 1024 neurons |  |  |  |
| FC layer, 5 neurons |  |  |  |

1. Early fusion. For data type as a video clip, there is another method to combine the temporal information into neural network model, called early fusion. This method treats the time domain as the different channels. For example, the input video can have the size [T, H, W] for T time frames. In this case, the 2D convolution was perform on input tensor [B, T, H, W] and generates output tensor [B, T, H’, W’].

Fill in the following form for the output tensor size after 2D CONV and 2D pooling and calculate the number of weights and biases. Suppose the video has the size 128x256x256 for TxHxW.

|  |  |  |  |
| --- | --- | --- | --- |
| Layers | Output dimensions | # kernel size or weights | # biases |
| Input | 128x256x256 | - | - |
| 2D CONV, kernel 5x5, number of kernels 32, stride 1, padding 2x2 |  |  |  |
| 2D pooling, 2x2 |  |  |  |
| 2D CONV, kernel 5x5, number of kernels 64, stride 2, padding 2x2 |  |  |  |
| 2D pooling, 2x2 |  |  |  |
| 2D CONV, kernel 3x3, number of kernels 128, stride 2, padding 1x1 |  |  |  |
| 2D pooling, 2x2 |  |  |  |
| FC layer, 4096 neurons |  |  |  |
| FC layer, 1024 neurons |  |  |  |
| FC layer, 5 neurons |  |  |  |

1. We now described two architectures in question a) and b). Both networks take inputs with the size 256x256x128 and produce a 5-class classification logits. Will you prefer 3D CONV or early fusion? Discuss their pros and cons.
2. We learned batch normalization can improve the training robustness and reduce sensitivity to hyperparameters. For the 2D image, the spatial BN is often used (BxCxHxW, normalize along BxHxW). For the 3D volume, the spatial BN can be extended to become the spatial-temporal BN (BxCxDxHxW, normalize along BxDxHxW). If you add a BN layer after every CONV in networks for a) and b), how many new parameters are introduced?
3. CNN CIFAR-10. In this task, we will work on the CIFAR-10. This time, we will build and train a number of CNNs:

Recall: the CIFAR-10 dataset consists of 60000 32x32x3 color images in 10 classes, with 6000 images per class. Typically, there are 50000 training images and 10000 test images.

Note: the training loop is now left to you to implement. You should reuse the

1. The first model you will build is a small CNN with the following architecture:

Im 🡪 Block(5x5, 32)🡪BlockDownSample(5x5, 64)🡪 Block(3x3, 64)🡪BlockDownSample(3x3, 128)🡪 Block(3x3, 128)🡪BlockDownSample(3x3, 256) 🡪 Block(3x3, 256)🡪flatten🡪FC-256🡪 ReLU🡪FC-10

Block(kernel\_size, N) : 🡪 CONV🡪 BN🡪 ReLU🡪CONV🡪BN🡪ReLU 🡪, this block will keep the image size; the first CONV will output N channels; the second CONV will keep the number of channels unchanged. Two CONVs have the same kernel\_size.

BlockDownSample(kernel\_size, N): 🡪 CONV, stride 2 🡪 BN🡪 ReLU🡪CONV🡪BN🡪ReLU 🡪, the first CONV will downsample the image size by x2 and outputs N channels and the second CONV will keep the image size and number of output channels unchanged. Two CONVs have the same kernel\_size.

The CE loss will be used after FC-10.

Read and understand the code in a3\_small\_cnn.py, model.py and train.py. Complete the functions to build the network. Using the He initialization for CONV and Xavier initialization for FC weights. Add the optimizers for either SGD or Adam. Add the learning rate scheduler.

Train the model with “python3 src/a3\_small\_cnn.py” with default parameters and submit the plotting for training and validation loss and accuracy and report the testing accuracy you get. You are required use wandb for the plot. Your test accuracy should be higher than 0.6.

1. Use the mobile net CONV to implement the BlockMobileNet and BlockDownSampleMobileNet modules in the a3\_small\_cnn.py and replace the Block in the previous network. Note the mobile net CONV is to assume a convolution is separable and is implemented with depth-wise convolution and 1x1 convolution.

Train your model to use mobile net block ( “python3 src/a3\_small\_cnn.py --reg 0.005 --use\_mobile\_net\_conv”). Does the training getting faster? Submit the plotting for training and validation loss and accuracy and report the testing accuracy you get. Your test accuracy should be higher than 0.6.

1. You will now implement a ResNet model with more depth in a3\_small\_resnet.py and resnet\_model.py. The architecture is as the following:

Im🡪 CONV(5x5,32)🡪ResBlock(3x3, 32) 🡪ResBlock(3x3, 32)🡪 ResBlockDownSample(3x3, 64)🡪ResBlock(3x3, 64) 🡪ResBlock(3x3, 64)🡪 ResBlockDownSample (3x3, 128) 🡪ResBlock(3x3, 128)🡪ResBlock(3x3, 128)🡪 ResBlockDownSample (3x3, 256)🡪 BatchNorm2d 🡪 ReLU 🡪 CONV(1x1, 512) 🡪 BatchNorm2d 🡪 ReLU 🡪 CONV(?x?, 512)🡪flatten🡪FC-256🡪 BatchNorm1d 🡪 ReLU🡪FC-10

CONV(?x?, 512) will output a tensor with size [B, 512, 1, 1], what is the correct kernel size?

ResBlock(kernel\_size, N) : 🡪 BN🡪ReLU🡪CONV🡪 BN🡪 ReLU🡪CONV🡪+🡪

|-----------------------------------------------------------|

The input to the block is added to the output. Note the starting CONV layer and structure of ResBlock to allow identity mapping.

ResBlockDownSample (kernel\_size, N): 🡪 BN🡪ReLU🡪 CONV, stride 2🡪 BN🡪ReLU 🡪CONV🡪+🡪

|--------------------------------|

Train your model and submit the plotting for training and validation loss and accuracy and report the testing accuracy you get. Your test accuracy should be close to 0.8.

1. **Build a U-net for segmentation**. Now you will build a U-net model for the pixel-wise segmentation task. We will use a modified version of the Kaggle carvana-image-masking-challenge dataset (https://www.kaggle.com/c/carvana-image-masking-challenge):



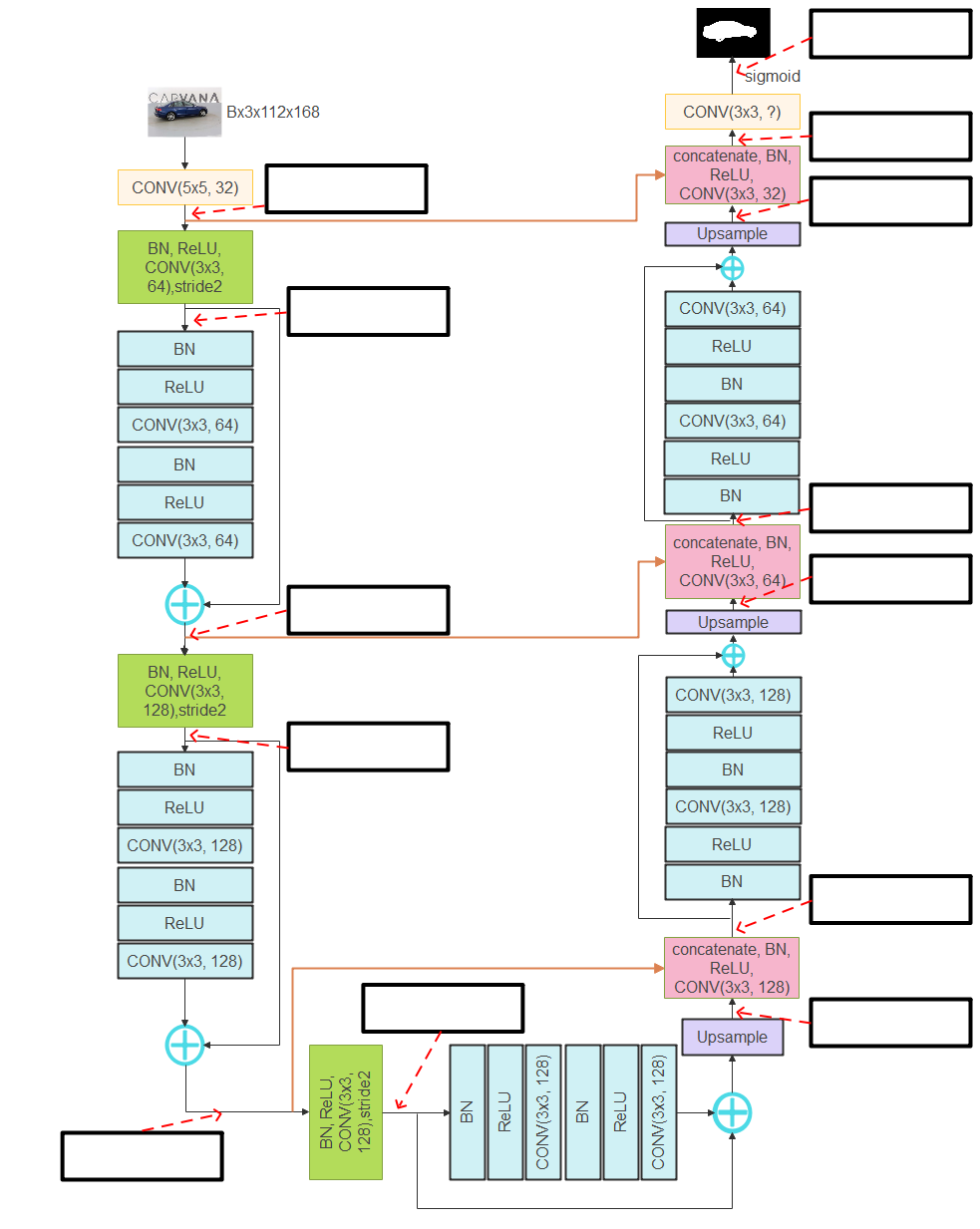


The task is to design and train a model to segment the car in the picture. Every image has the size 112x168x3. The mask has the size 112x168x1, with pixels inside the car being 1 and background being 0.

You can find the dataset in the data directory. In the training folder, there are N=4588 samples. In the test folder, there are N=500 samples.

1. Write a dataset for the carvana challenge. For every sample, the dataset class needs to return a tuple of (image, mask). Skeleton code is given in dataset.py.
2. We will use the U-net architecture for this task. The basic network architecture is laid out for you. Given the input [B, 3, 112, 168], fill in the tensor sizes in the following network architecture chart at the locations of red dashed arrows.

Note the image size is reduced through the downsampling (green box), with gradually increased conv filters. When image size is restored through upsampling, the number of conv filters are gradually reduced.



A few notes about this model:

* CONV: 2D convolution layer; BD: Spatial BatchNorm2D; Upsample: use interpolation; Concatenation: concatenate the tensor along the channel dimension
* All CONV, except input CONV, has 3x3 kernel
* DownSample CONV has 3x3 kernel, stride 2, padding 1 (CONV, stride 2)
* Use the interpolation for upsampling (check torch.nn.functional.interpolate).
* The skip-connection is shown in orange and will pass the tensors from the downsample side to the upsample side. The residual connection will sum the tensors.

Attention is needed for the output layer (CONV 3x3, ?). What is number of output channels of this CONV layer? What is the size of output logits?

How will you design the loss function, given the output logits? (Hint: check the torch.nn.BCELoss).

1. Implement the model, loss function and training loop in the a3\_unet.py and unet\_model.py. Train the network on the train set and test it on the test set.

To evaluate the segmentation performance, the Dice ratio was used: , the ratio of 2x overlapped area over the total areas of A and B. What is best Dice ratio we can expect for a perfect segmentation?

Apply the trained model on the entire test set and compute and submit the mean Dice. Also, submit a random set of test samples with images, ground-truth masks and model generated masks.

1. This model is not a toy example but something which can work in practice. Where could this architecture be improved?

Further, if you need to do hyperparameter searching, which hyperparameters will be in your search space and what are their searching ranges (e.g. a list of possible values, or min/min pairs)?

1. **Pytorch Lightening.**

Now you had built and trained a number of deep learning models using Pytorch. While the Pytorch is very flexible and easy-to-use, it requires users to implement the training loop, as you did in this assignment. Pytorch Lightening is one toolbox built on top of the Pytorch and wraps training loop and many other features (logging, model saving, profiling, auto finding the batch size and learning rate and more) to make the training easier. Get yourself familiar with Pytorch Lightening by going through tutorials: <https://www.pytorchlightning.ai/tutorials/>. It consists of a set of short videos and will not take long to go through.

Rewrite the small resnet model in question 3c using Pytorch Lightenting. You will find skeleton code in a3\_small\_resnet\_pytorch\_lightening.py. Complete the coding and train the model with default parameters. Submit the plot for training and validation loss over epochs.