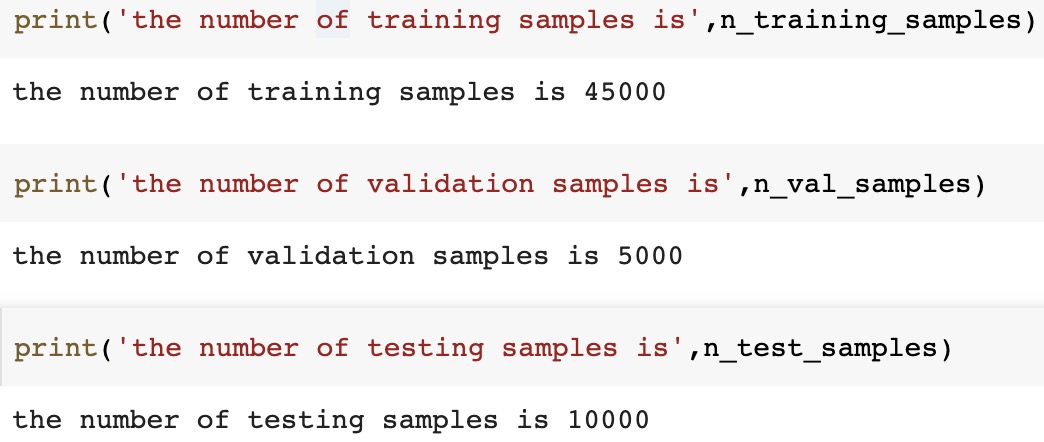
DS 340W: Lab Assignment 3

Part A: ResNet for Image Classification on CIFAR-10

1. Data Preparation

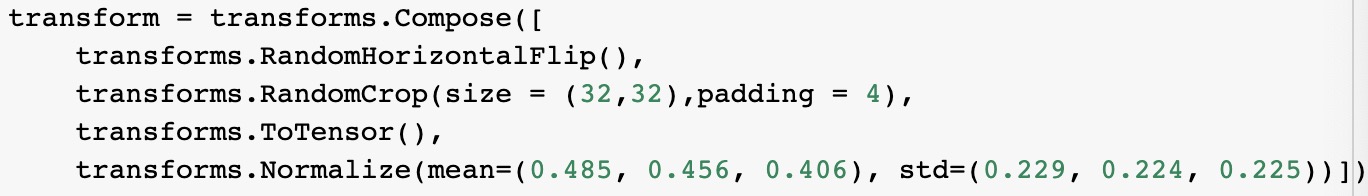
In this lab, I am aiming to train a residual neural network (ResNet) to do an image classification task on the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class (Krizhevsky, 2013). The 10 classes in the dataset are the following: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. There are 50000 training images and 10000 test images. In order to measure the performance of our model, I picked 5000 images from the training set as the validation set. So, the sizes of training, validation, and testing sets are 45000, 5000, 10000, respectively.



I used these following functions from the PyTorch package to conduct data augmentation and normalization: RandomHorizontalFlip, RandomCrop, and Normalize.

For RandomHorizontalFlip function, this function horizontally flips the given image randomly with a given probability, and I used default value (0.5) for this probability. For RandomCrop function, it crops the given at the center, and I set (32,32) as the desired output size of the crop and padding equals 4. These functions use to implement data augmentation, which helps us to have more data from limited data, and then it reduces the risk of overfitting (*PyTorch Tutorials*).

For Normalize function, this function normalizes a tensor image with mean and standard deviation, and I set the mean as [0.485, 0.456, 0.406], and the standard deviation as [0.229, 0.224, 0.225] in this experiment. This function uses to implement the data normalization, which transforms features to be on a similar scale, and then this improves the performance of the model and reduces the risk of overfitting (*PyTorch Tutorials*).



1. Network Setting

In this experiment, we used the ResNet to do an image classification task on the CIFAR-10 dataset.

The diagram that summarizing the network architecture is below:

Input Image

3x3 conv 16

3x3 conv 16

3x3 conv 16

Layer 1

16

3x3 conv 16

3x3 conv 16

3x3 conv 16

3x3 conv 16

3x3 conv 16

3x3 conv 32

Layer 2

32

3x3 conv 32

3x3 conv 32

3x3 conv 32

3x3 conv 32

3x3 conv 32

3x3 conv 64

Layer 3

64

3x3 conv 64

3x3 conv 64

3x3 conv 64

3x3 conv 64

Output

This neural network has an additional conv layer at the beginning, and the size is 3\*3\*16 in this case. This ResNet has three big layers, and each layer has three residual blocks. Every residual block has two 3x3 conv layers. Periodically, it doubles the number of filters and downsamples using spatially using stride 2, and the size of each big layer is 16, 32, and 64, respectively (He et al.,2016). Finally, we got the output.

In this experiment, I used stochastic gradient descent (SGD) as the optimizer, in which I set momentum equals 0.9. SGD allows minibatch learning, in which the algorithm can split the training dataset into small batches that are used to calculate model error and update model coefficients, and SGD with momentum is a method to accelerate gradients vectors in the right directions, thus leading to faster converging (Brownlee, 2019). For loss function, I used CrossEntropyLoss from the PyTorch package. This criterion combines LogSoftmax and the negative log-likelihood loss (NLLLoss) in one single class. It is useful when training a classification problem with C classes (*PyTorch Tutorials*). The mathematical formula of this loss function is below:

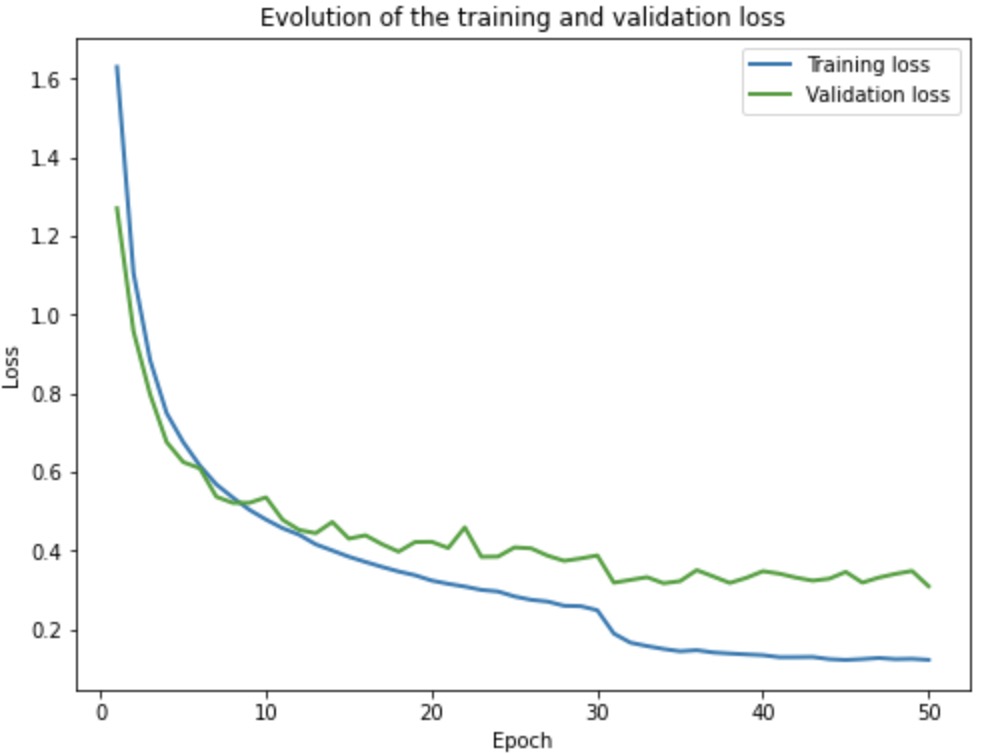
or in the case of the weight argument being specified:

Finally, the losses are averaged across observations for each minibatch.

1. Network Training

For the training process, I used the following parameters: batch size equals 32, the number of epochs equals 50, and an initial learning rate equals 0.1. I used MultiStepLR function to decay the learning rate when the number of epochs reaches one of the milestones, and the training process starts with an initial learning rate of 0.1, and multiply it by a decay factor of 0.1 after 30 and 40 epochs, respectively. Finally, the training process stops after 50 epochs.

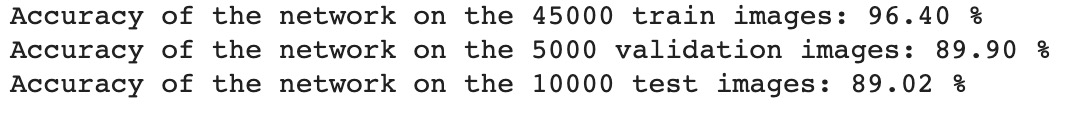
Below is the plot of the training loss and validation loss as functions of the number of epochs:



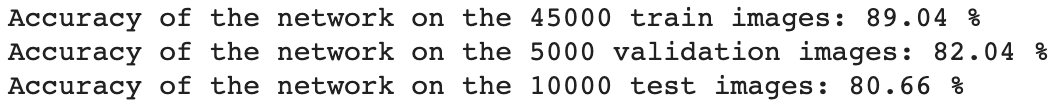
We can see that both curves of training loss and validation loss begin to converge and smooth after the learning rate decays. This result shows MultiStepLR function is very important for this experiment.

1. Network Testing

Testing Performance with Data Augmentation and Normalization:

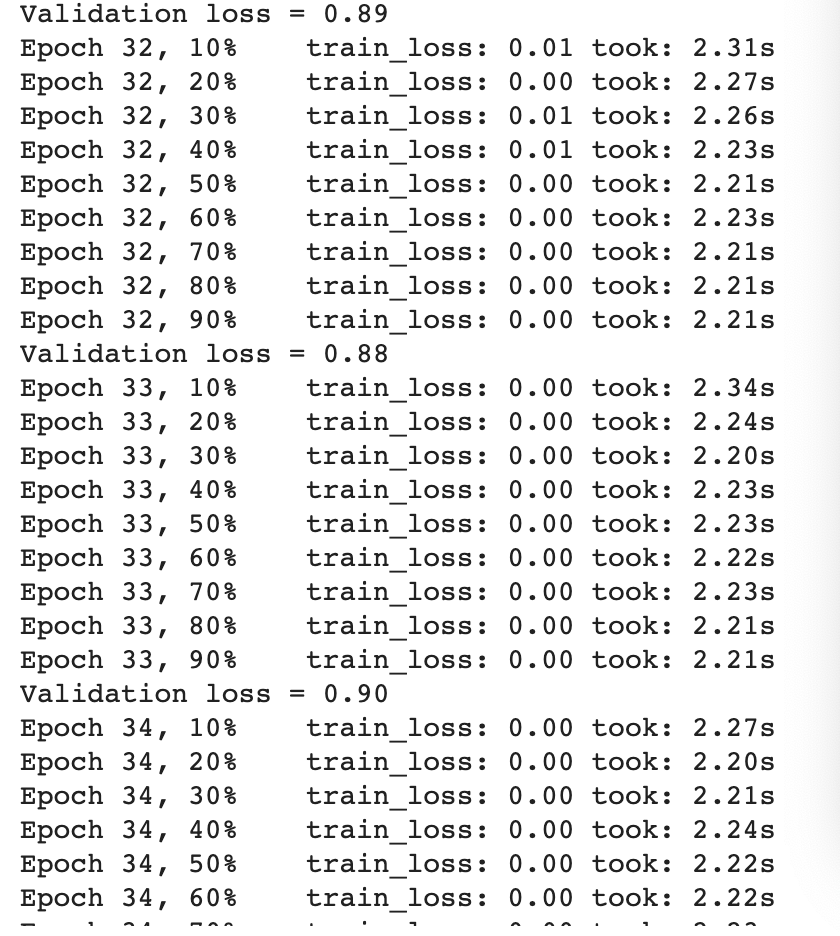


Testing Performance without Data Augmentation and Normalization:



My best testing accuracy was 89.02%, and it was very close to our aimed performance, which is about 90%. We can see that the model with data augmentation and normalization has 89% accuracy on the testing, and the model without data augmentation and normalization has 80% accuracy on the testing data. There is a 9% difference between them, which is very significant. So, we can see that the model with data augmentation and normalization is much better than the model without data augmentation and normalization.

The Train Loss of the Model Without Data Augmentation and Normalization:

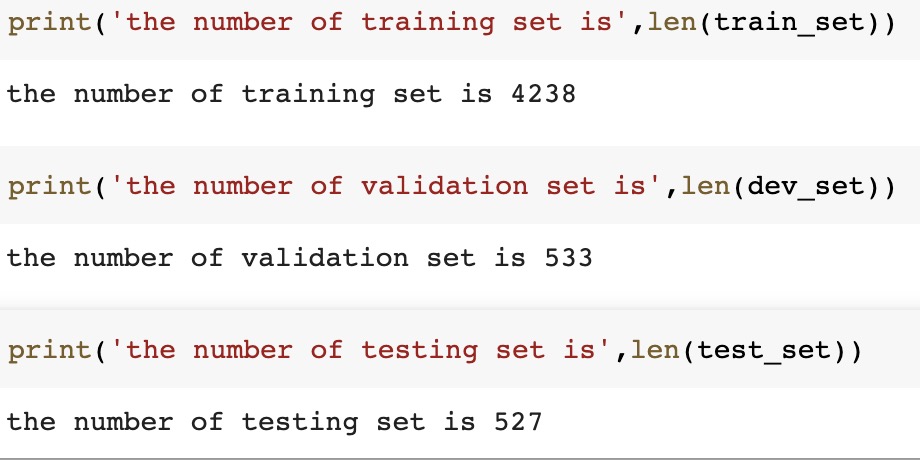


From the above result, we can see that the model without data augmentation and normalization has a very serious overfitting problem. In contrast, as I mentioned above, data augmentation helps us to have more data from limited data, and then it reduces the risk of overfitting, and data normalization transforms features to be on a similar scale, and then this improves the performance of the model and reduces the risk of overfitting. So, overfitting is the reason for this huge gap between these two performances, and this result proves that data augmentation and normalization are very important for the image classification task.

Part B: Experiments of KaoKore Dataset

1. Data Preparation

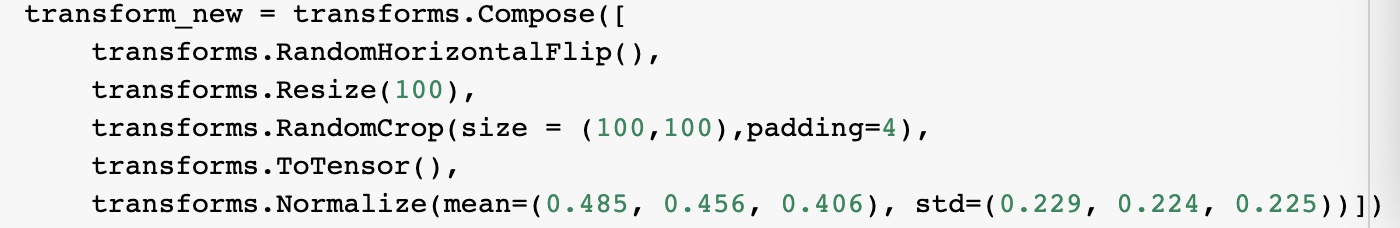
KaoKore is a novel dataset of face images from Japanese illustrations along with multiple labels for each face. This dataset contains 5552 image files, each being an color (RGB) image of size 256x256 as well as two sets of labels gender and social status. In this experiment, we only used the status category, and 4 labels of class status are the following: noble, warrior, incarnation, and commoner (Tian, et al, 2020). The sizes of training, validation, and testing sets are 4238, 533, 527, respectively.



I used these following functions to conduct data augmentation and normalization: RandomHorizontalFlip, Resize, RandomCrop, and Normalize.

For RandomHorizontalFlip function, this function horizontally flips the given image randomly with a given probability, and I used default value (0.5) for this probability. For Resize function, it resizes the input image to the given size, and I reduced the size of input images to 100. This function is very important in this lab because the resolutions of images in the KaoKore dataset are high (256x256), and then I have to reduce their resolutions to save GPU memory and accelerate the training. For RandomCrop function, it crops the given at the center, and I set (100,100) as the desired output size of the crop that same as the size of input images, and padding equals 4. These three functions use to implement data augmentation, which helps us to have more data from limited data, and then it reduces the risk of overfitting (*PyTorch Tutorials*).

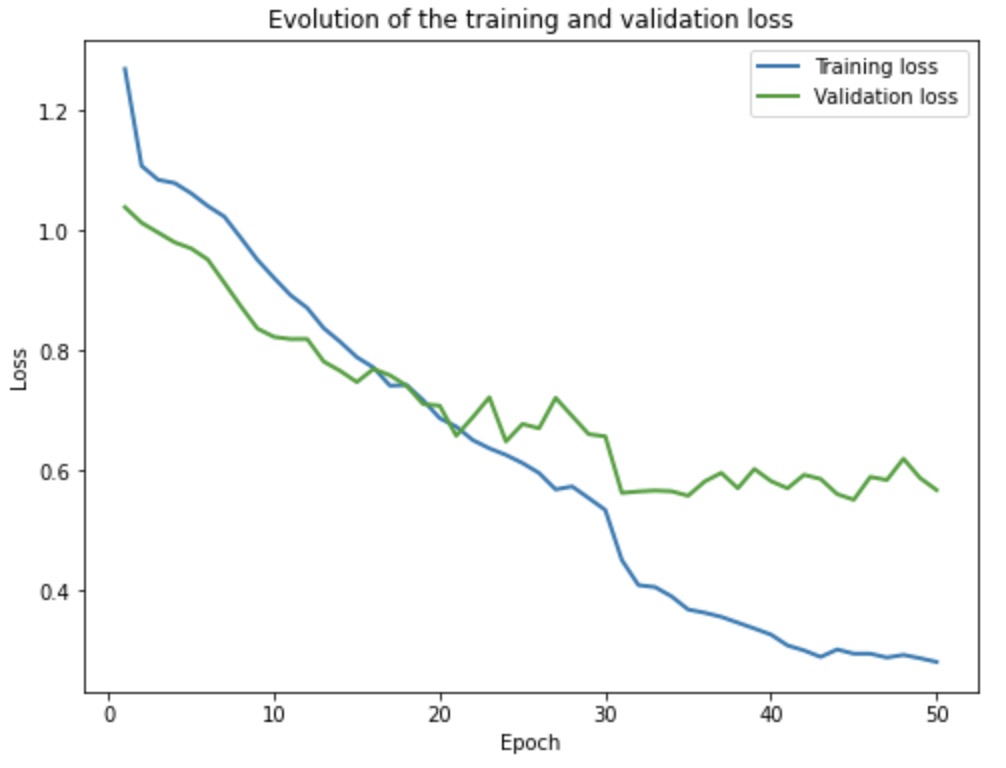
For Normalize function, this function normalizes a tensor image with mean and standard deviation, and I set the mean as [0.485, 0.456, 0.406], and the standard deviation as [0.229, 0.224, 0.225] in this experiment. This function uses to implement the data normalization, which transforms features to be on a similar scale, and then this improves the performance of the model and reduces the risk of overfitting (*PyTorch Tutorials*).



1. Network Training

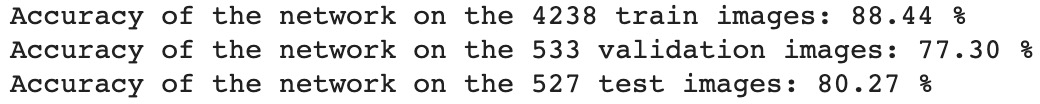
For the training process, I used the following parameters: batch size equals 32, the number of epochs equals 50, and an initial learning rate equals 0.1. I used MultiStepLR function to decay the learning rate when the number of epochs reaches one of the milestones, and the training process starts with an initial learning rate of 0.1, and multiply it by a decay factor of 0.1 after 30 and 40 epochs, respectively. Finally, the training process stops after 50 epochs.

Below is the plot of the training loss and validation loss as functions of the number of epochs:



We can see that the training loss curve is very similar to the training curve in part A. This curve is a very typical convergence curve, and it becomes gradually smooth after 30 and 40 epochs. The curve of validation loss is a little bit different from the curve of validation loss in part A, in which the increase of epochs does not have a that great impact on the validation loss. However, this curve is also getting smooth after two milestones.

1. Network Testing



My best testing accuracy was 80.27%, and it achieved our aimed performance, which is about 80%.

Reference

Brownlee, J. (2019, August 19). *A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size*. Retrieved February 27, 2020, from <https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/>

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi: 10.1109/cvpr.2016.90

Krizhevsky, A. (2013.d.). *The CIFAR-10 dataset*. Retrieved April 10, 2020, from https://www.cs.toronto.edu/~kriz/cifar.html

*PyTorch Tutorials*¶. (n.d.). Retrieved from <https://pytorch.org/tutorials/>

Tian, Y., Suzuki, C., Clanuwat, T., Bober-Irizar, M., Lamb, A., & Kitamoto, A. (2020). KaoKore: *A Pre-modern Japanese Art Facial Expression Dataset*. Retrieved from https://arxiv.org/abs/2002.08595

Reference on Code

I used the Pytorch, pandas, matplotlib, numpy packages to complete the code, so some of the code may very similar.