Hw4, Fine-tuning

Report written by Xiaodong Huo
May 1, 2020

Part 1 :

Question 1:

The problem the Dense net is trying to solve is increase the depth of the CNN. Stacking more convolutional layers cause problems. As we back propagate the gradient, we are taking derivatives for each hidden layer and the weights gets small and we can hardly update the weights. ResNets and DenseNets have many layers(100+) compares with LeNet, AlexNet or VGGNet. The input of the next layer of a DenseNet is the concatenation of all the precious layer inputs. For example, for L2, the input is the concatenation of original input, input to L1, input to L2. In between the Dense Blocks, the apply the 1X1 convolution which preserves the spatial resolution but shrinks the depth of the feature map and a max pooling to reduce the feature map size. Advantages: solves vanishing-gradient problem; reuses features; reduce number of parameter; trains much deeper networks; Disadvantages: compares with Resnet, Dense Net uses a lot more memory with tensors concatenated together. Thus, harder to train on cuda since GPU run out of memory.

Question 2:

The basic idea of InceptinoNet is instead of pick one of these filter sizes($1X3$ filter or $3X3$ or $5X5$) or pooling layer, we can just do them all and just concatenate all the outputs and let the network to learn the parameters with those different filter sizes. A 1X1 convolutional layer is also used to decrease the computational cost. The side branches takes the hidden layer as input and make predictions accordingly. Thus, these layers have regularized effect on the network and prevent the network from overfitting. Advantage: The biggest advantage of the inception net is increased depth of the network thus improved performance. Also, we don't need to choose the filter size of the convolutional layer or the max pooling layer. Disadvantage: The biggest weakness of the inception net is the high computational cost and complexity it added to the network. InceptionNet is costly to retrain, pretrained models are used more often. With the dimension reduction, however, the performance can be affected by lowering dimension bottleneck layer.

Question 3:

The AutoEncoder takes input x and reconstruct $\hat{x}$, the loss function is a measure of how good the reconstruction is.

To train the AutoEncoder, we can use

1. Adam optimizer
2. Mean squared error loss function
3. Cross Entropy Loss

The suitable loss functions depends on the inputs. For binary inputs, the suitable loss function is cross-entropy loss loss (sum of Bernoulli cross-entropy's) However, for non-binary real-valued inputs, cross-entropy will not have a perfect Cross-entropy loss; For non-binary inputs, we can use Mean square error loss functions, which is the sum of reconstruction minus the real inputs and squares. We can also use the Adam optimizer.

Question 4:

A Sparse autoencoder are used to learn features for another task, for example, classification. In sparse autoencoders with a sparsity enforcer that directs a single layer network to learn code dictionary. The sparse autoencoder has one hidden layer, which minimizes the error in reproducing the input while reducing number of parameters for reconstruction. A sparse autoencoder can be implemented with de-noising autoencoder

Question 5:

A de-noising autoencoder removes noise from matches. A de-noising autoencoder has similar structure as a regular autoencoder. A extra noise term is introduced the input now becomes (input + noise). For each epoch, there should be a ramdom noise added to the input. The effect of this is to create perturbation to the input and prevent the network from memorizing the network. In another word, prevent overfitting.

Question 6:

Segmentation is the process of object detection and classification where identifying parts of the image and what it belongs to.

Semantic Segmentation detects each pixel that object belongs to. For example, separate all people from the image; Just pixels are produced. Single object is detected.

Instance Segmentation is similar to Semantic Segmentation on the pixel level, where is much deeper.For example, separate each person from the image. Multiple objects are detected.

Question 7:

Classification and localization: Single object detection. First classify the image into known classes. Then localize the image by edge detection, or with windowing techniques.

Semantic Segmentation detects each pixel that object belongs to. For example, separate all people from the image;

Just pixels are produced. Single object is detected.

Instance Segmentation is similar to Semantic Segmentation on the pixel level, where is much deeper.For example, separate each person from the image. Multiple objects are detected.

Question 8:

NMS is a computer vision technique. It selects one entity out of may overlapping entities. The criteria is probability number along with overlap measure.

Question 9:

R-CNN solves the problem of selecting a huge number of regions by doing a selective search; 1. Generate initial segmentation.2. Use greedy algorithm to combine similar regions into larger ones. 3. use generated regions to produce final candidate region proposals.

Fast R-CNN: similar to R-CNN. Instead of feeding the region proposals to the CNN, we take the input image to the CNN to produce a convolutional feature map. the region of proposals are identified from the feature map.

Faster R-CNN: similar to Fast R-CNN Instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals.

Question 10:

YOLO is an open-source framework used for object detection. YOLO stands for 'you only look once,' which implies the framework's fast speed. It can process 45 frames per second. Another advantage of YOLO framework is that it understands generalized object representation.

Question 11:

We could use intersection over union (IoU) concept to measure the performance of object detector. Depending on the value of IoU, we could determine whether the object detection is True Positive (TRP, False Positive (FP), False Negative (FN), or True Negative (TN). After this step, we could use precision and recall as the matrix to compute the performance. Finally, mean average precision (mAP) could be calculated using 11 point interpolation technique by plotting the precision and recall.

Part 2 Summery:

AlexNet is more suitable for this particular image classification task for it's similar performance but faster training. I saved my trained model as file: vggmodel and alexmodel; or train the model with code:

model_ft, history = train_model(model, dataloaders_dict, criterion, optimizer_ft, num_epochs=num_epochs)

I am using cuda for this homework. To make things work, please check all the directories and change accordingly.

Network and fine tuning

The networks I use for fine tuning are AlexNet and VGG net. Imported from torchvision.models;

alexnet = models.alexnet(pretrained=False)

vgg16 = models.vgg16(pretrained=False)

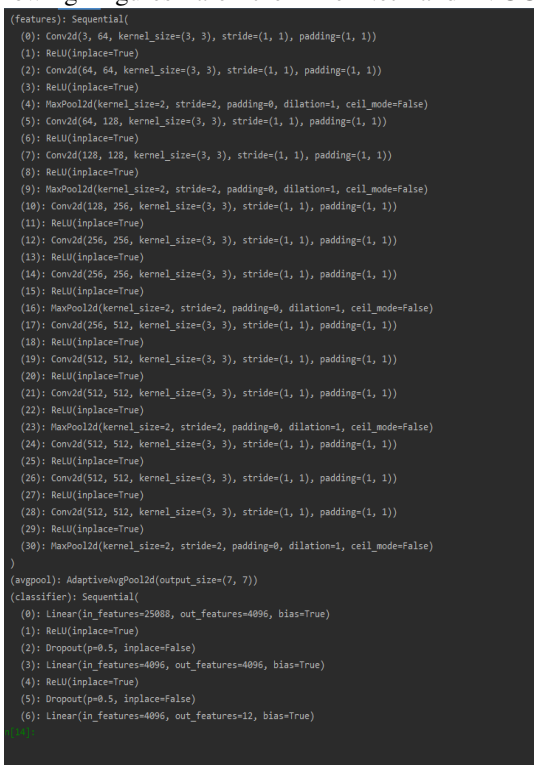
First thing needs to fine tune is the last layer of the model; Last layer of those two models output 1000 classes, I need to tune it to 12 classes for my classification task;code:

model.classifier[6] = nn.Linear(4096, num_classes)

Then the layers parameter needs to randomly initialize; The layers are convolutional layers and fully connected layers

The optimized used is SGD. Another thing for tuning is to normalize the dataset.

Since I am using GPU, the training data and lables are moved to GPU with to.device; The following figures are the AlexNet and VGG model.



vgg_structure

```
In[3]: model
Out[3]:
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=12, bias=True)
  )
)
```

AlexNet_structure

Datasets, Experiments and Results:

As we can see, VGG is a much deeper network and have a lot more parameters to train.

Classification accuracy and ROC curve:

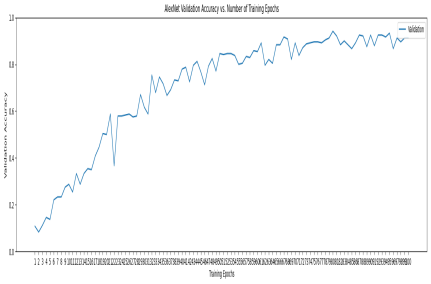
Both networks are trained by 100 epochs, no obvious overfitting is observed.

The best accuracy for Alexnet is 0.94 and for VGG is 0.9333;

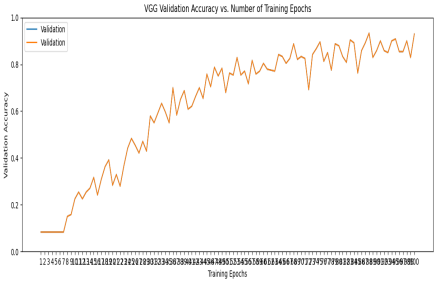
The training time for Alexnet is 13min and for VGG is 46 mins;

The ROC curve shows the Alexnet is a little bit fitter than the VGGNet;

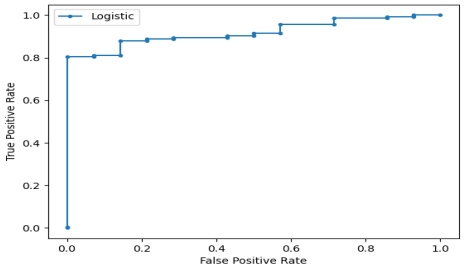
Thus, overall, Alexnet is more suitable for this classfication task for its slightly better performance and much faster training.



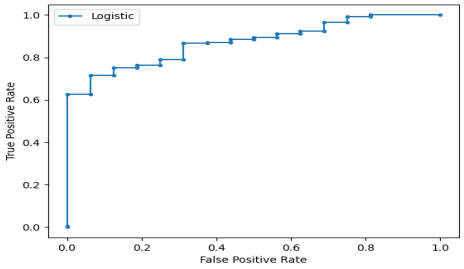
AlexNet validation error



VGGNet validation error



ROC curve for AlexNet



ROC curve for VGGnet

References