Assignment 3

BS6207

https://github.com/cyberpunk-newman/Master/tree/main/BS6207/Assignment3

Q1

MaxPool2d: used to accelerate the computing speed and avoid over-fitting.Usually inserted in a continuous convolutional layer. Kernel_size means the size of the window which scans the plane. Stride is the step of moving window, default value is kernel_size. Padding means to add zero on both sides, here used the 'SAME' mode, this mode is to add 0 when the number of remaining rows and columns is insufficient to meet the size of the pooling window, and to keep the pooled area of the window the same. Dilation control the stride of elements and return_indices will return the max indices along with the outputs when it is True. Ceil_mode means to use the ceil instead of the floor to compute the output shape.

Avgpool2d applies the 2D average pooling over an input signal composed of several planes. Cpunt_include_pad means when calculating the average the zero-padding will be included.

$$out(N_i, C_j, h, w) = \frac{1}{kH * kW} \sum_{m=0}^{kH-1} \sum_{n=0}^{kW-1} input(N_i, C_j, stride[0] \times h + m, stride[1] \times w + n)$$

Conv2d: apply the 2D convolution over an input signal composed of several planes. Most parameters have the same meaning of maxpool2d, except group. The para group controls the connections between input and output

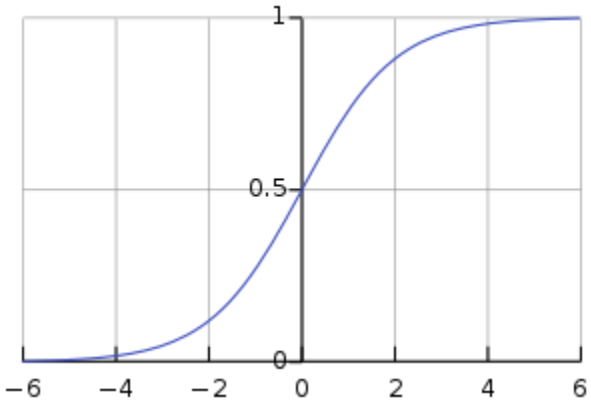
$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

Convtranspose2d: it is used to transpose the convolution layer, more complex than upsampling layer.

Flatten: reshape the one-dimensional tensor, here the end_dim is -1 means it covers all dimensions.

Sigmoid: an activation function used in machine learning, ranging from 0 to 1.

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} = 1 - S(-x).$$

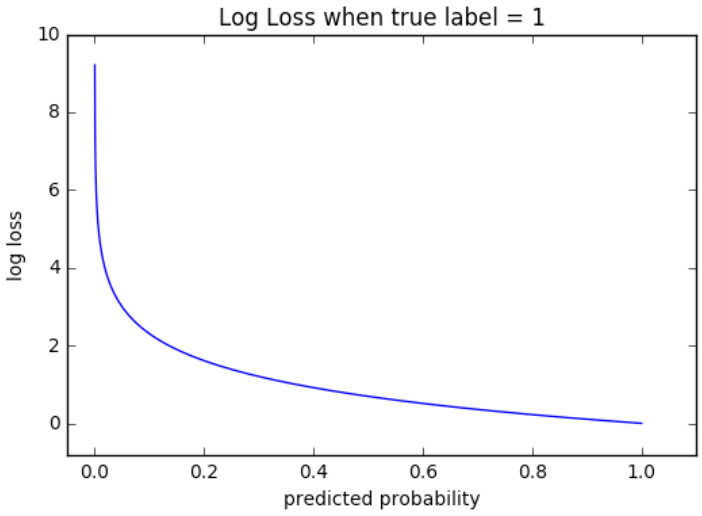


Batchnorm2d: The mean and standard-deviation are calculated per-dimension over the mini-batches and γ and β are learnable parameter vectors of size *C* (where *C* is the input size). The para eps means the value added to the denominator for numerical stability.

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

cross_entropy: measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverge from the actual label.

$$-(y \log(p) + (1-y) \log(1-p))$$



mse_loss: measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value. Y is the vector of observed values of the variable being predicted, with y_hat being the predicted values

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Q2.a

In this question, I use the PyTorch to implement the whole Convolutional Neural Network. For the dataset I directly use toruchvision.dataset function to load the data. According to the request given by the assignment, here is my network design. There are totally 21 layers including the input and output layers.

```
Sequential(
  (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=same)
  (1): ReLU()
  (2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=same)
  (3): ReLU()
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=same)
  (6): ReLU()
  (7): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=same)
  (8): ReLU()
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=same)
  (11): ReLU()
  (12): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=same)
  (13): ReLU()
  (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (15): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=same)
  (16): ReLU()
  (17): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=same)
  (18): ReLU()
  (19): AdaptiveAvgPool2d(output_size=(1, 1))
  (20): Flatten(start_dim=1, end_dim=-1)
  (21): Linear(in_features=128, out_features=10, bias=True)
)
```

During the training process, I did not make accuracy to 80% until I add digital augmentation. This is a method to increase the number of images and the change of images. I rotate each photo to like -30 to 30 degrees before passing it to the model. The batch size is 128 and the epoch is 200. Also, I used SGD instead of Adam because though Adam is fast, but it is not as good as SGD in this case.

```
Accuracy of the network on the 10000 test images: 80 %
```

Also, I compute the average absolute value of each activation, and across all training data print out each layer's image. During I visualize the feature maps through a convolutional layer, though all the outputs are 3D volume, the height and width correspond to the dimensions of the feature map, the depth channel is a distinct encoding feature so we can still print out the image as a 2D image. I just record intermediate activations.

From the image, there are bright areas and dark areas. The bright areas are activated means that the filter has been detected. I printed out all the layers image and through the network, we can say the first 3 layers are 16 images; following 3 layers are 32; following 3 layers are 64; last three layers are 128. This structure fits the network structure as well.

We can see the first few layers retain most of the obvious information in the image, like the edge detectors. As the network goes deeper the feature map looks more abstract and unlike the original image. After the image becomes abstract, it also turns sparser. It is more like to be pixel style.

So in conclusion, as the image goes deeper in the network. It will first be detected the most important features in an image like line or edge. Then it will become more abstract because the network focuses on one part of the image and encode them. Then the filters detect fewer features in the last few layers because the first few layers it contains a lot of pieces of information in the image but it may not appear in every small image so it will go sparser and feature maps will be shown as blank as well.

Q2.b

In this network, we added the batch normalization layer followed by each activation function. Same as the previous model, I added the digital augmentation with rotation, and the accuracy of the model goes up to 82%.

```
Accuracy of the network on the 10000 test images: 82 %
```

In this model, I also calculate each activation’s average absolute values and print out the feature maps. What batch normalization has done is to let distribution back the normalization and increase the gradient and speed up the convergence. So we can see from these images, the dark areas are more than the previous model. As the network goes

deeper, the filters build on each layer and encode more complex patterns. These observations are similar to the previous question. Lower layers encode simple structures and deeper layers more focus on complex patterns.

The difference between model A and model B is that model B is not that sparse as model A, and I think maybe the reason is that the batch normalization causes the density of distribution and the model can be more sensitive to the features throughout the whole network.