# CSCI4980: Homework 5

**Noah Hendrickson**

**Problem #1:**

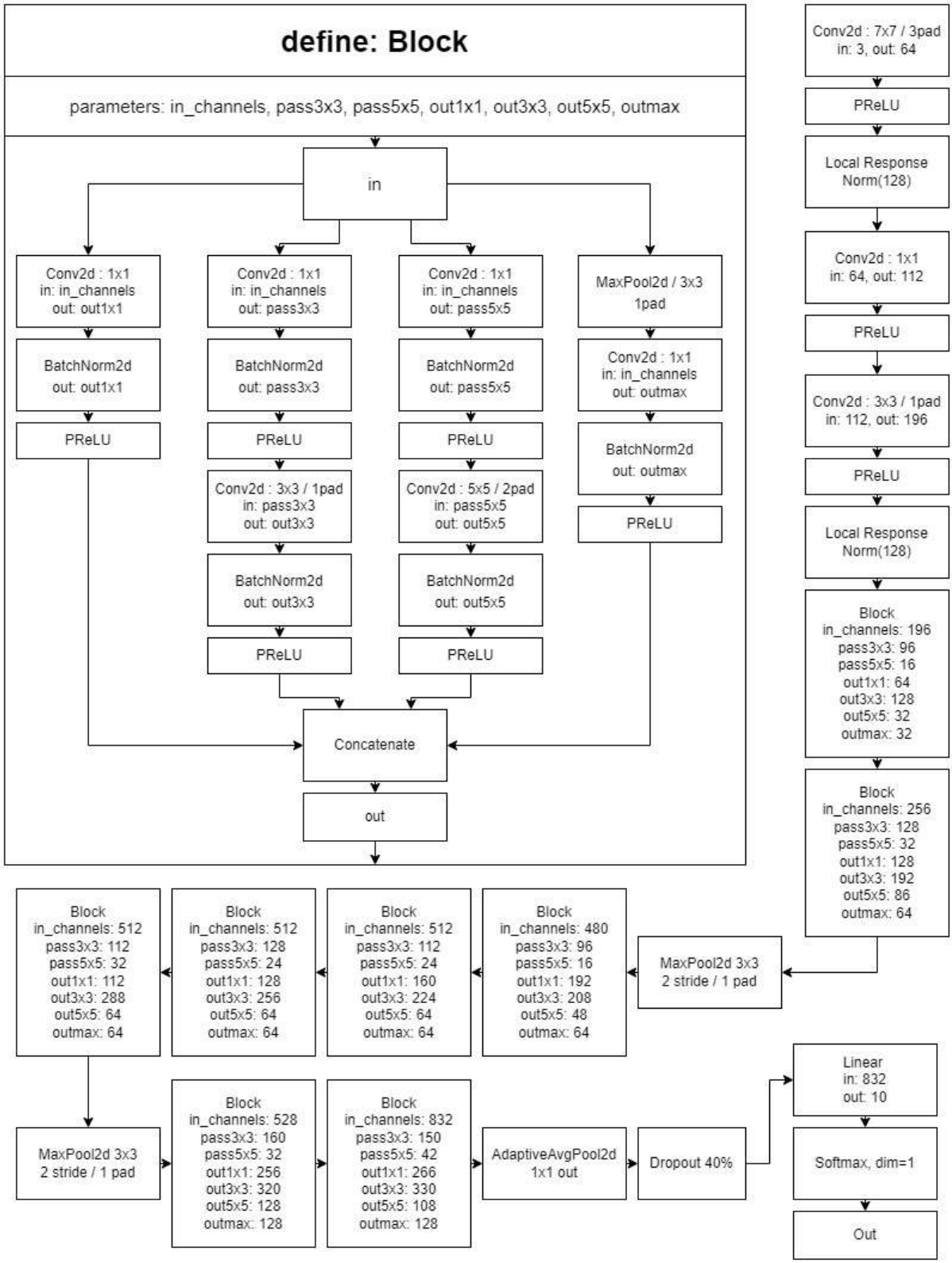
For my model architecture, I decided to follow as closely as I could to the GoogLeNet architecture provided in the paper titled "Going Deeper With Convolutions." I did take some artistic liberties though. The architecture makes use of inception blocks which consist of 4 different operations each separately applied on the input data. The first applies a 1x1 convolution into a batch norm into a ReLU. The second and third do a 1x1 convolution into a batch norm into a ReLU into another convolution but 3x3 and 5x5 respectively into a batch norm and finally into a last ReLU. The fourth applies a 3x3 max pool, but with padding so there is no downsampling, into a 1x1 convolution into a batch norm into a ReLU. Each of these layers is given a separate number of channels to output. The second and third layers are also given "pass on" layers which are passed from the input into the second convolution of the layer. The final step concatenates each layer into a single output with the same image dimensions as the original but with the number of channels equal to the sum of the outputs of each layer.

The layers of the main model consist of an initial 3 convolutions each followed by a ReLU and a Local Response Norm. That then goes into 2 sequential inception blocks which feed into a max pool to downsample. There are then 4 sequential blocks into another downsampling max pool. There are 2 final inception blocks before the adaptive avg pool brings the output to a 1 dimensional image. A crucial dropout layer with probability of 40% follows after this. The paper makes sure to note that this dropout is very important at regularizing. A linear layer then connects to the output with a softmax at the end. The exact channels and sizes of each of these layers can be found in the code as it would be a little too verbose for here I think.

The optimizer I used was Adam with a learning rate of 0.001. The paper used SGD but Adam has worked well for me in the past so that's what I decided to stick with. I didn't have much opportunity to tune the learning rate as this model trained incredibly slowly, however, from homework done in the past on MNIST, 0.001 works pretty good for Adam. The loss function was Cross Entropy Loss.

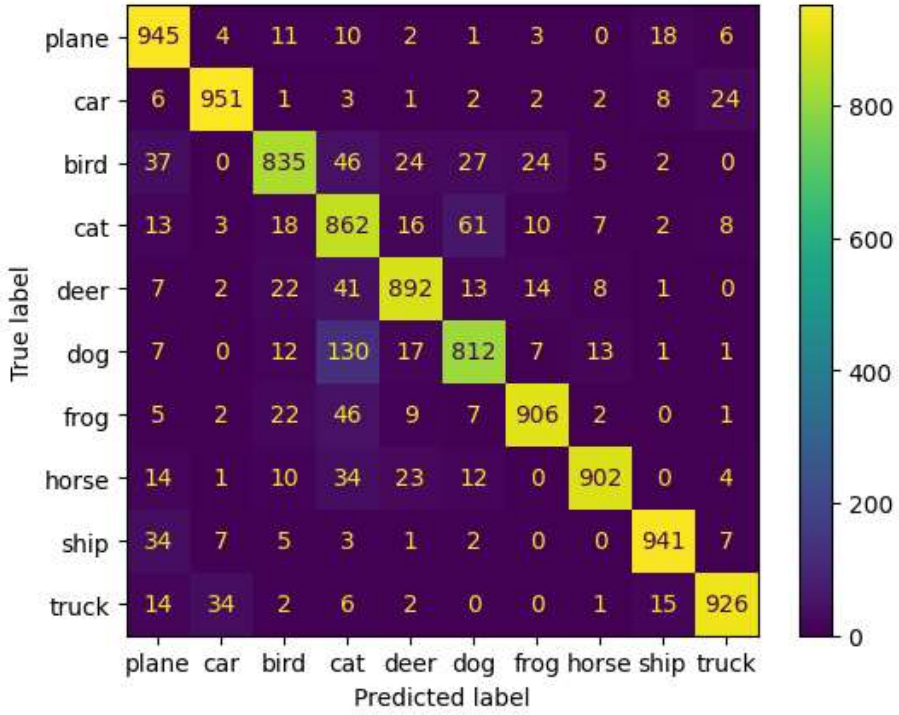
Additionally, to help convergence rates, I initialized every convolution layer's weights as kaiming normal. The reasons behind this can be found in this paper titled "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." Additionally, from that paper I decided to use Parametric ReLU activation functions in order to remove the discontinuity of vanilla ReLU and also add a learnable parameter to it.

As for training, I ended up training for 100 epochs with a batch size of 48. I trained for 100 epochs because that's when the training loss started to really converge and also I was losing a bit of patience after training it for so long :) and the batch size was a mix between trying to save GPU memory on my machine and getting as close to the 64 that I normally use on smaller models for CIFAR-10. I didn't have a ton of time to optimize the batch size, however I did try it with a couple smaller batches and it didn't quite kick off as well as it does with 48.


define: Block
parameters: in_channels, pass3x3, pass5x5, out1x1, out3x3, out5x5, outmax
in
Conv2d : 1x1
in: in_channels
out: out1x1
BatchNorm2d
out: out1x1
PReLU
Conv2d : 1x1
in: in_channels
out: pass3x3
BatchNorm2d
out: pass3x3
PReLU
Conv2d : 3x3 / 1pad
in: pass3x3
out: out3x3
BatchNorm2d
out: out3x3
PReLU
Conv2d : 1x1
in: in_channels
out: pass5x5
BatchNorm2d
out: pass5x5
PReLU
Conv2d : 5x5 / 2pad
in: pass5x5
out: out5x5
BatchNorm2d
out: out5x5
PReLU
MaxPool2d / 3x3
1pad
Conv2d : 1x1
in: in_channels
out: outmax
BatchNorm2d
out: outmax
PReLU
Concatenate
out
Conv2d : 7x7 / 3pad
in: 3, out: 64
PReLU
Local Response
Norm(128)
Conv2d : 1x1
in: 64, out: 112
PReLU
Conv2d : 3x3 / 1pad
in: 112, out: 196
PReLU
Local Response
Norm(128)
Block
in_channels: 196
pass3x3: 96
pass5x5: 16
out1x1: 64
out3x3: 128
out5x5: 32
outmax: 32
Block
in_channels: 256
pass3x3: 128
pass5x5: 32
out1x1: 128
out3x3: 192
out5x5: 86
outmax: 64
MaxPool2d 3x3
2 stride / 1 pad
Block
in_channels: 480
pass3x3: 96
pass5x5: 16
out1x1: 192
out3x3: 208
out5x5: 48
outmax: 64
Block
in_channels: 512
pass3x3: 112
pass5x5: 24
out1x1: 160
out3x3: 224
out5x5: 64
outmax: 64
Block
in_channels: 512
pass3x3: 128
pass5x5: 24
out1x1: 128
out3x3: 256
out5x5: 64
outmax: 64
Block
in_channels: 512
pass3x3: 112
pass5x5: 32
out1x1: 112
out3x3: 288
out5x5: 64
outmax: 64
MaxPool2d 3x3
2 stride / 1 pad
Block
in_channels: 528
pass3x3: 160
pass5x5: 32
out1x1: 256
out3x3: 320
out5x5: 128
outmax: 128
Block
in_channels: 832
pass3x3: 150
pass5x5: 42
out1x1: 266
out3x3: 330
out5x5: 108
outmax: 128
AdaptiveAvgPool2d
1x1 out
Dropout 40%
Linear
in: 832
out: 10
Softmax, dim=1
Out

**Problem #2:**

The resulting accuracy of the model after 100 epochs was initially 90.33%, however, I made the mistake of only saving the best of the epochs if the training ended early instead of just generally, so the **ACTUAL** final accuracy on the testing set after 100 epochs was **89.72%**. I probably could have fixed that and ran it for a couple more epochs to get it back up, however, I was a little bit lazy and had other things to do. The confusion matrix is shown below.




**Problem #3: See Problem 2** 🙂