1. **Part 2 results:**
   1. **Conv pool Conv pool flatten fc1 fc2 fc3**

*#Computes the activation of the first convolution*

*#Size changes from (3, 32, 32) to (16, 32, 32)*

x = F.relu(*self*.conv1(x))

*#Size changes from (16, 32, 32) to (16, 16, 16)*

x = *self*.pool(x)

*#Size changes from (16, 16, 16) to (32, 16, 16)*

x = F.relu(*self*.conv2(x))

*#Size changes from (32, 16, 16) to (32, 8, 8)*

x = *self*.pool(x)

*#Reshape data to input to the input layer of the neural net*

*#Size changes from (32, 8, 8) to (1, 2048)*

*#Recall that the -1 infers this dimension from the othergiven dimension*

x = x.view(-1, 32 \* 8 \*8)

*#Computes the activation of the first fully connected layer*

*#Size changes from (1, 2084) to (1, 64)*

x = F.relu(*self*.fc1(x))

*#Computes the activation of the first fully connected layer*

*#Size changes from (1, 64) to (1, 2)*

x = F.relu(*self*.fc2(x))

x = F.relu(*self*.fc3(x))

*lr*=0.01, *wd*=0.0000)

*momentum*=0.9,

op = chain([

type\_cast(np.float32),

add(-127.5),

mul(1/127.5),

hwc2chw(),

])

epoch 100

train loss: 0.000 +- 0.000

val acc: 0.726

Best Accuracy of 0.7300342969132778 at epoch 74

1. **Conv Conv pool Conv Conv pool flatten fc1 fc2 –fc3> not much better**

epoch 100

train loss: 0.000 +- 0.000

val acc: 0.730

Best Accuracy of 0.7315041646251838 at epoch 87

1. **Conv pool conv pool conv pool flatten fc1 fc2 fc3**

*#Computes the activation of the first convolution*

*#Size changes from (3, 32, 32) to (16, 32, 32)*

x = F.relu(*self*.conv1(x))

*#Size changes from (16, 32, 32) to (16, 16, 16)*

x = *self*.pool(x)

*#Size changes from (16, 16, 16) to (32, 16, 16)*

x = F.relu(*self*.conv2(x))

*#Size changes from (32, 16, 16) to (32, 8, 8)*

x = *self*.pool(x)

*#Size changes from (32, 8, 8) to (64, 8, 8)*

x = F.relu(*self*.conv3(x))

*#Size changes from (64, 8, 8) to (64, 4, 4)*

x = *self*.pool(x)

*#Reshape data to input to the input layer of the neural net*

*#Size changes from (32, 8, 8) to (1, 2048)*

*#Recall that the -1 infers this dimension from the other given dimension*

x = x.view(-1, 64 \* 4 \* 4)

*#Computes the activation of the first fully connected layer*

*#Size changes from (1, 2084) to (1, 64)*

x = F.relu(*self*.fc1(x))

*#Computes the activation of the first fully connected layer*

*#Size changes from (1, 64) to (1, 2)*

x = F.relu(*self*.fc2(x))

x = F.relu(*self*.fc3(x))

*lr*=0.01, *wd*=0.0000

*momentung = 0.9*

epoch 100

train loss: 0.000 +- 0.000

val acc: 0.729

Best Accuracy of 0.7511024007839294 at epoch 28

1. **Discuss how individual techniques of overfitting affect performance(5 combinations)**
2. **With wd=0.00001 actually worse:**

epoch 100

train loss: 0.000 +- 0.000

val acc: 0.743

Best Accuracy of 0.7496325330720235 at epoch 52

1. **With dropout of chance 0.5 after fc1:**

epoch 100

train loss: 0.020 +- 0.019

val acc: 0.748

Best Accuracy of 0.7604115629593337 at epoch 49

1. **With dropout after pool2:**

epoch 100

train loss: 0.235 +- 0.061

val acc: 0.781

Best Accuracy of 0.7922586967172954 at epoch 76

1. **Same as part 1.c but new image augmentation:**

op = chain([

hflip(),

rcrop(32,5,"constant"),

type\_cast(np.float32),

add(-127.5),

mul(1/127.5),

hwc2chw(),

])

epoch 100

train loss: 0.243 +- 0.055

val acc: 0.825

Best Accuracy of 0.8432141107300343 at epoch 97

1. **Implementation of additional augmentation through blurring did not improve.**
2. **Including dropout 0.5 after pool2 and weight decay:**
3. Questions to answer:
   1. How does SGD work? What are your findings on the example data (part1)?
   2. Which network architecture did you choose for part 2 and why? Did you have problems reaching a low training error?
   3. What are the goals of data augmentation, regularization, and early stopping? How exactly did you use these techniques (hyperparameters, combinations) and what were your results (train and val performance)? List all experiments and results, even if they did not work well, and discuss them.
   4. If you utilized transfer learning, explain what you did and your results.