ECE 549 Homework 4 Report

Nickel Liang

May 4, 2021

Q1.1. Implement BaseNet

Accuracy:

```
Accuracy of the final network on the val images: 60.5 %
Accuracy of airplane : 59.8 %
Accuracy of automobile : 74.3 %
Accuracy of  bird : 54.8 %
Accuracy of   cat : 46.3 %
Accuracy of  deer : 46.5 %
Accuracy of   dog : 52.9 %
Accuracy of  frog : 64.8 %
Accuracy of horse : 66.2 %
Accuracy of  ship : 69.0 %
Accuracy of truck : 70.3 %
```

Model Architecture:

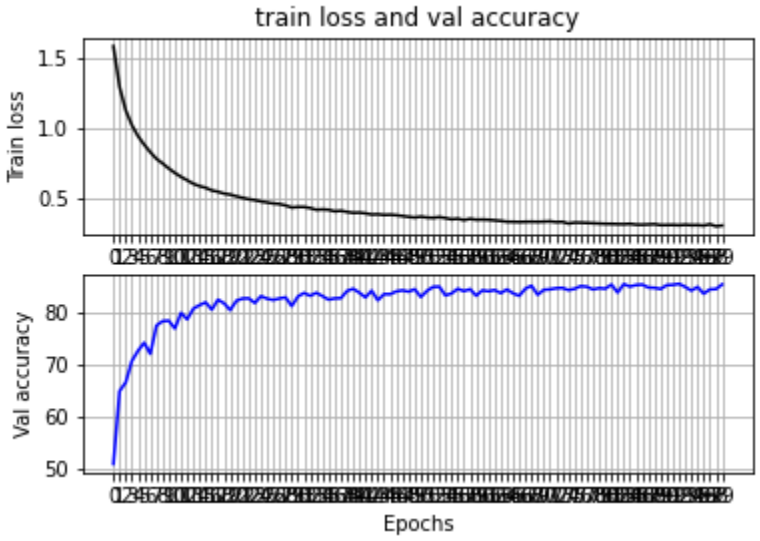
```
BaseNet(
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=400, out_features=200, bias=True)
    (2): ReLU()
    (3): Linear(in_features=200, out_features=10, bias=True)
  )
)
```

Q1.2. Improve BaseNet

Final Accuracy on Validation Set:

```
Accuracy of the final network on the val images: 85.6 %
Accuracy of airplane : 85.1 %
Accuracy of automobile : 94.6 %
Accuracy of  bird : 84.4 %
Accuracy of   cat : 69.8 %
Accuracy of  deer : 85.1 %
Accuracy of   dog : 77.3 %
Accuracy of  frog : 91.8 %
Accuracy of horse : 87.2 %
Accuracy of  ship : 92.2 %
Accuracy of truck : 88.5 %
```

Plots after 100 epoch



Final Architecture Table:

Layer	Layer Type	Kernel Size	Input Dim	Output Dim	Input Chan	Output Chan
1	conv2d	3	32	32	3	8
2	relu		32	32	8	8
3	bn		32	32	8	8
4	conv2d	3	32	32	8	16
5	relu		32	32	16	16
6	bn		32	32	16	16
7	conv2d	3	32	32	16	32
8	relu		32	32	32	32
9	bn		32	32	32	32
10	maxpool2d	2	32	16	32	32
11	conv2d	3	16	16	32	64
12	relu		16	16	64	64
13	bn		16	16	64	64
14	conv2d	3	16	16	64	128
15	relu		16	16	128	128
16	bn		16	16	128	128
17	conv2d	3	16	16	128	256
18	relu		16	16	256	256
19	bn		16	16	256	256
20	maxpool2d	2	16	8	256	256
21	linear		1	1	16384	64
22	relu		1	1	64	64
23	linear		1	1	64	32
24	relu		1	1	32	32
25	linear		1	1	32	16

26	relu		1	1	16	16
27	linear		1	1	16	10

Ablation Table:

Changed Factor	Overall Accuracy
Base implementation	60.5%
Add normalization to both train and test with `Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))`	59.5%
Add `RandomAffine` and `RandomHorizontalFlip` to `train_transform`	63.8%
Deeper network: four `Conv2d` blocks for 15 epoch	66.0%
Add another 15 epoch, 30 epoch total	68.4%
Add `BatchNorm2d` after each `Conv2d` blocks	72.2%
Change kernel size to 3 for all `Conv2d` blocks	76.9%
Add `RandomCrop` to `train_transform` after two transformations mentioned above, change learning rate to 0.001, train 30 epoch	79.8%
Add more fully connected layers, train 45 epoch	77.1%
Deeper network: six `Conv2d` blocks for 100 epoch with 0.01 learning rate, padded all `Conv2d` blocks so `Conv2d` blocks does not change output shape	85.6%

Q1.3. Secret Test Set

Final Accuracy on Test Set:

```
Accuracy: 74.4 %
Accuracy of airplane : 65.0 %
Accuracy of automobile : 70.5 %
Accuracy of  bird : 63.0 %
Accuracy of   cat : 58.5 %
Accuracy of  deer : 74.5 %
Accuracy of   dog : 73.0 %
Accuracy of  frog : 85.5 %
Accuracy of horse : 84.5 %
Accuracy of  ship : 79.5 %
Accuracy of truck : 90.0 %
```

Q2.2. Build On Top Of ImageNet Pre-trained Model

Model Architecture:

The model used in this part is a pre-trained Resnet-18 without the last two layers, and add a conv layer with kernel size 1 and output channel 3, followed by interpolation and normalization.

```
MyModel(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (5): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (6): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
```

```
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (7): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
  )
  (conv2): Conv2d(512, 3, kernel_size=(1, 1), stride=(1, 1))
```

I add interpolation and normalize in the forward section, hence not visible in network printout:

```
def forward(self, x):
        _, _, h, w = x.shape
        x = self.conv1(x)
        x = self.conv2(x)
        x = F.interpolate(x, size=(h, w),
                          mode='bilinear', align_corners=True)
        x = F.normalize(x)
        return x
```

With the following configuration:

```
criterion = nn.L1Loss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-4)
train_dataset = NormalDataset(split='train')
train_dataloader = data.DataLoader(train_dataset, batch_size=8,
                                   shuffle=True, num_workers=2,
                                   drop_last=True)
scheduler = optim.lr_scheduler.CyclicLR(optimizer, step_size_up=30,
base_lr=0.001, max_lr=0.005, cycle_momentum=False)
num_epoch = 120
```

I got the following result on validation set:

```
Validation loss (L1): 0.2596068686972826
Validation metrics: Mean 35.2, Median 27.8, 11.25deg 23.6, 22.5deg 43.1,
30deg 52.6
```

Q2.3. Increase Your Model Output Resolution

Final Model Architecture:

I followed DeepLabV3+ paper and added an Atrous Spatial Pyramid Pooling (ASPP) block and a decoder block, with ResNet-18 as the backbone.

```
MyModel(
  (resnet18_head): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  )
  (resnet18_layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (resnet18_layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (resnet18_layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (resnet18_layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (aspp): ASPP(
    (relu): ReLU(inplace=True)
    (bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (pool): AdaptiveAvgPool2d(output_size=1)
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (conv2): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(6, 6), dilation=(6, 6),
bias=False)
    (conv3): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(12, 12), dilation=(12, 12),
bias=False)
    (conv4): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(18, 18), dilation=(18, 18),
bias=False)
    (conv5): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (conv): Conv2d(1280, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  )
  (decoder): Decoder(
    (conv_ll): Conv2d(64, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn_ll): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv): Sequential(
      (0): Conv2d(304, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): ReLU(inplace=True)
      (6): Dropout(p=0.1, inplace=False)
      (7): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
    )
  )
)
```

With the following configuration:

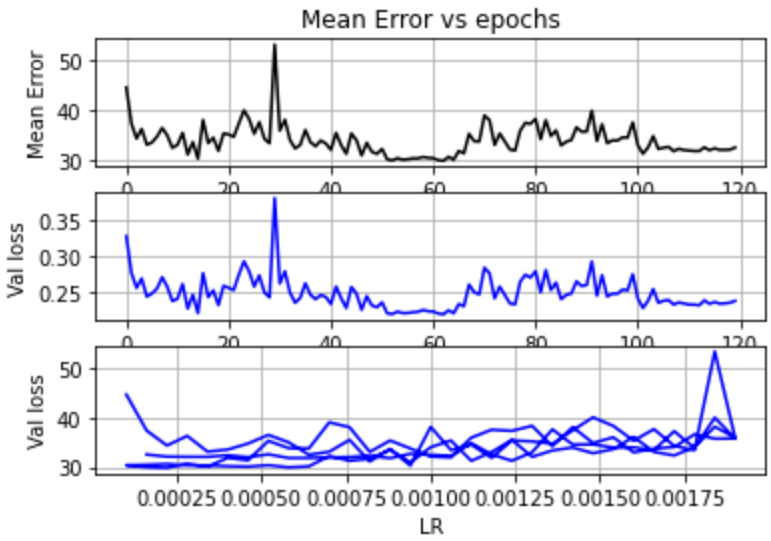
```
criterion = nn.L1Loss()
optimizer = optim.Adam(model.parameters(), lr=0.0001, weight_decay=1e-4)
train_dataset = NormalDataset(split='train')
train_dataloader = data.DataLoader(train_dataset, batch_size=8,
                                   shuffle=True, num_workers=2,
                                   drop_last=True)
scheduler = optim.lr_scheduler.CyclicLR(optimizer, step_size_up=30,
base_lr=0.0001, max_lr=0.0019, cycle_momentum=False)
num_epoch = 60
```

I got the following result:

```
Validation loss (L1): 0.21295185992096224
Validation metrics: Mean 29.0, Median 19.5, 11.25deg 35.2, 22.5deg 54.1,
30deg 62.4
```

Ablation Table:

1. From part 2, I learned that Adam optimizer does a better job than SGD, and I always use L1 loss as it is easier to implement with pyTorch. I did not transform the test set, and I never touched batch size. I learned that Cyclic Learning Rate helps a lot for the performance, and doing a Learning Rate Test (LR Test) is extremely useful. Following the paper, I got that I should have a step size similar to batch size, so I use a step 30. And I learned that the epoch number should be approximately an even number multiple of the step size, so that at the end of the training we are at the low learning rate range. Eventually I used 120 epochs in this part to achieve the best result. After all these, I got base performance in part 2:
   `Mean 35.2, Median 27.8, 11.25deg 23.6, 22.5deg 43.1, 30deg 52.6`
2. I read DeepLabV3+ paper and attempted to construct an ASPP block. This ASPP block takes the output of ResNet-18 as the input, then the output of this ASPP block is directly upsampled 32 times and normalized to get the final output. After 120 epoch, I got:
   `Mean 34.2, Median 24.0, 11.25deg 29.4, 22.5deg 48.2, 30deg 56.3`
3. I then added a decoder, as described in Table 2 of the DeepLabV3+ paper. I'm still using parameters from part 2, and let it run for 60 epoch:
   `Mean 34.3, Median 24.5, 11.25deg 25.1, 22.5deg 47.4, 30deg 56.2`
4. I then performed another LR Test, and decided that a base learning rate of 0.0001 and a max learning rate of 0.0019 would be the most appropriate. I kept everything else as is, and let it run for 120 epoch:
   `Mean 32.5, Median 21.5, 11.25deg 33.5, 22.5deg 51.2, 30deg 58.9`
   In this step, I recorded the L1 loss, mean angular error, and learning rate. I plotted a graph:



The top graph shows the mean error vs. epoch; the middle graph shows the L1 loss on validation set vs. epoch; the bottom graph shows the mean error vs. epoch.
Since my step size for cyclic learning rate is 30, the learning rate will reach 0.0019 at epoch 30 and goes back to 0.0001 at epoch 60. Then the learning rate will go up again, and goes back to 0.0001 again at epoch 120. You can see this trend in the bottom graph.
From the middle and Top graph, we can see that at approximately epoch 60 the model already reaches to a pretty good state, and training for extra 60 epoch only made it worse. This is actually also suggested by DeepLabV3+ paper.

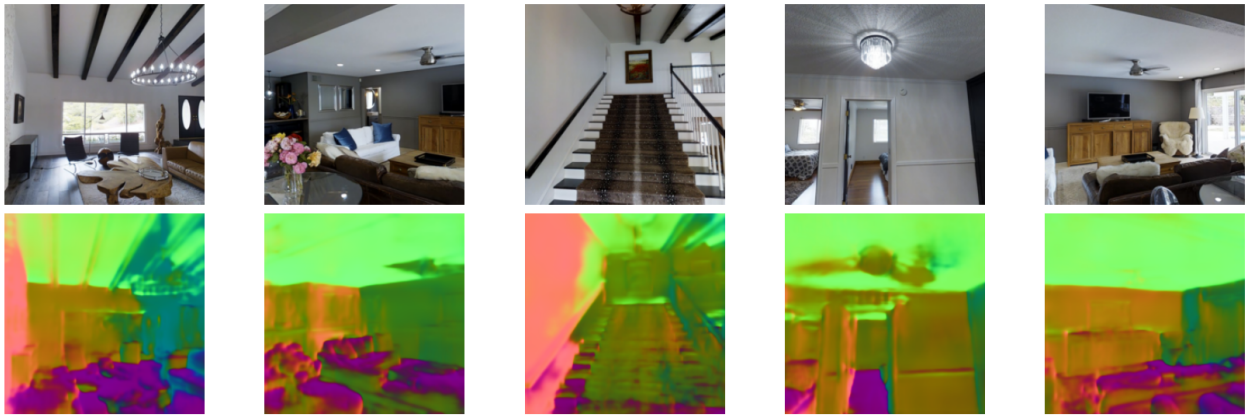
5. Hence, I decided to reduce epoch to 65. In my last run the Mean Angular Error already went below 30, and this time it should be able to do that again. I also decided to save any model that has a Mean Angular Error less than 30.5.

Eventually I figured out that this model reached it's best state at epoch 57. I used model saved at epoch 57 to get the best result:

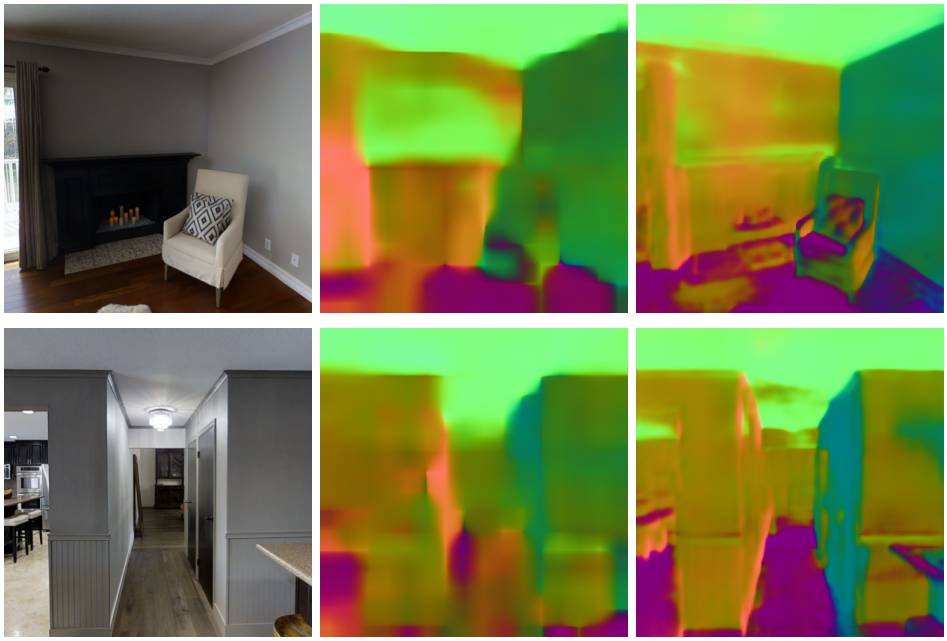
```
Mean 29.0, Median 19.5, 11.25deg 35.2, 22.5deg 54.1, 30deg 62.4
```

Q2.4. Visualize Your Prediction

Best Model Prediction for Five Indoor Images:



Compare part 2 and part 3:



From these two images above, we can see that the decoder in part 3 did a really good job at upsample the image.

The output from part 2 (middle) has very few details and you can barely tell there's a sofa in the first image. However the output from part 3 (right) can predict surface normal for the sofa.

For the second image, we can see that for objects like walls, part 3 provides a more consistent result for a specific surface. For example, the floor were marked very accurately by part 3 output, while part 2 did a very poor job.

Q2.5. Secret Test Set

Best performance on test set:

```
Test metrics:
mean error 27.6
median error 17.3
accuracy at 11.25deg 36.6
accuracy at 22.5deg 58.0
accuracy at 30deg 66.5
```