# Problem1 Implement and improve BaseNet on CIFAR-10

## 1. Implement BaseNet：

**BaseNet**

**1.1 Code:**

```
class BaseNet(nn.Module):
    def __init__(self):
        super(BaseNet, self).__init__()

        # TODO: define your model here
        # pass
        # torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True, padding_mode='zeros')
        # torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1,
return_indices=False, ceil_mode=False)
        # torch.nn.Linear(in_features, out_features, bias=True)

        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.conv3 = nn.Conv2d(16,64,3)
        self.pool = nn.MaxPool2d(2, 2)

        self.fc_net = nn.Sequential(
            nn.Linear(400, 200),
            nn.ReLU(inplace=True),
            nn.Linear(200, 10)
        )

    def forward(self, x):

        # TODO: define your model here
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1,400)
        x = self.fc_net(x)

        return x
```

### 1.2 Print command `print(net)`

```
BaseNet(
  (conv_net): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_net): Sequential(
    (0): Linear(in_features=400, out_features=200, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=200, out_features=10, bias=True)
  )
)
```

### 1.3 Final accuracy for train the model

### Final accuracy on the validation set

```
Accuracy of the final network on the val images: 59.9 %
Accuracy of airplane : 62.9 %
Accuracy of automobile : 73.6 %
Accuracy of  bird : 41.4 %
Accuracy of   cat : 40.7 %
Accuracy of  deer : 58.2 %
Accuracy of   dog : 49.8 %
Accuracy of  frog : 69.7 %
Accuracy of horse : 71.1 %
Accuracy of  ship : 65.5 %
Accuracy of truck : 66.4 %
```

```
[1] loss: 1.867
Accuracy of the network on the val images: 44 %
[2] loss: 1.461
Accuracy of the network on the val images: 50 %
[3] loss: 1.327
Accuracy of the network on the val images: 55 %
[4] loss: 1.232
Accuracy of the network on the val images: 56 %
[5] loss: 1.146
Accuracy of the network on the val images: 56 %
[6] loss: 1.082
```

```
Accuracy of the network on the val images: 60 %
[7] loss: 1.024
Accuracy of the network on the val images: 61 %
[8] loss: 0.968
Accuracy of the network on the val images: 60 %
[9] loss: 0.925
Accuracy of the network on the val images: 60 %
[10] loss: 0.875
Accuracy of the network on the val images: 60 %
[11] loss: 0.834
Accuracy of the network on the val images: 61 %
[12] loss: 0.798
Accuracy of the network on the val images: 61 %
[13] loss: 0.764
Accuracy of the network on the val images: 60 %
[14] loss: 0.729
Accuracy of the network on the val images: 59 %
[15] loss: 0.700
Accuracy of the network on the val images: 59 %

Finished Training
```

# 2. Improve BaseNet

### My best model Code:

### 2.1 Final accuracy on validation set

```
Accuracy of the final network on the val images: 86.9 %
Accuracy of airplane : 89.5 %
Accuracy of automobile : 94.3 %
Accuracy of  bird : 77.2 %
Accuracy of   cat : 73.3 %
Accuracy of  deer : 88.5 %
Accuracy of   dog : 76.8 %
Accuracy of  frog : 93.3 %
Accuracy of horse : 92.4 %
Accuracy of  ship : 90.6 %
Accuracy of truck : 92.7 %
```

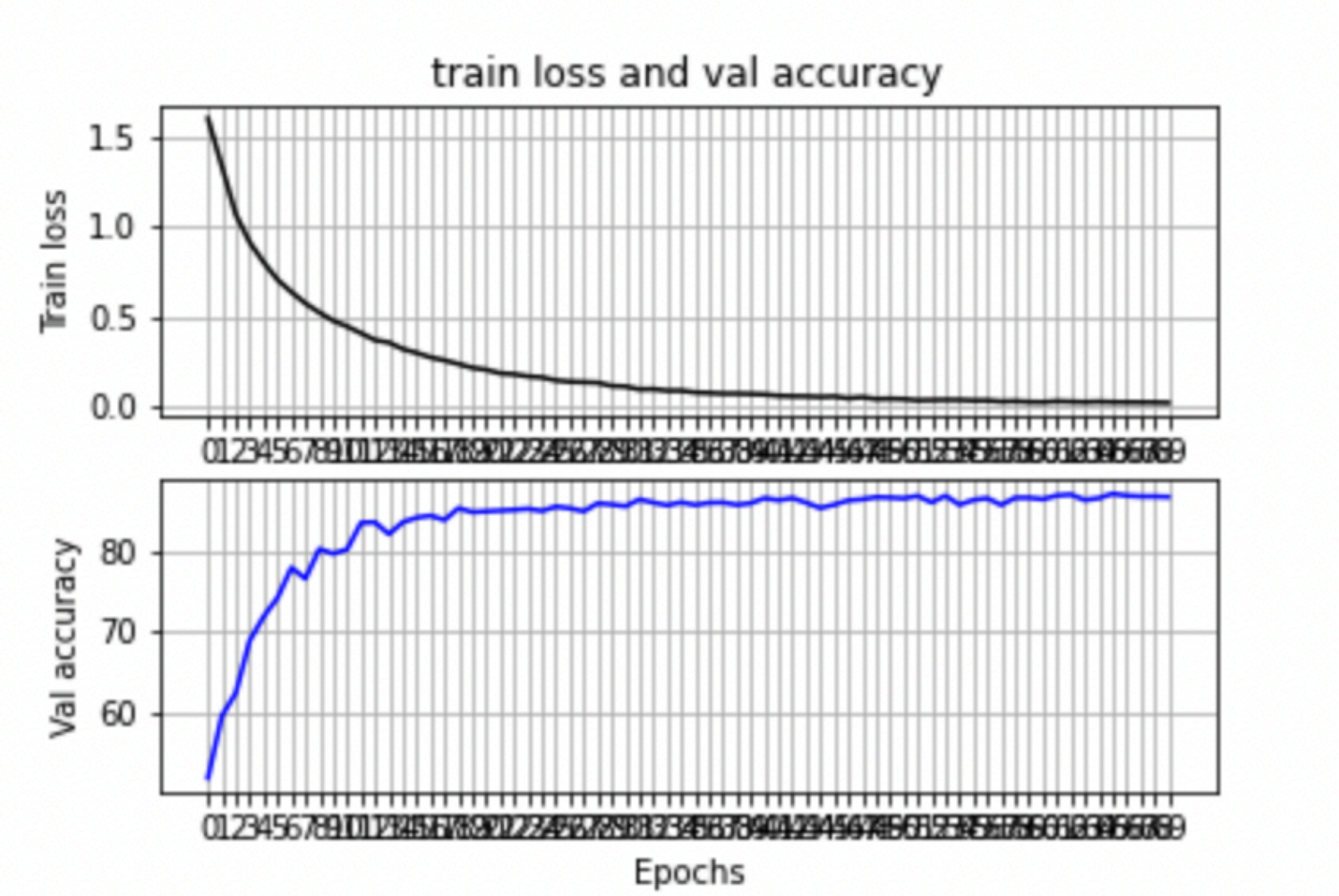
## 2.2 Table defining your final architecture

| Layer No. | Layer Type No. | Layer Type | Kernel Size | Input Dim | Output Dim | Input Channels | Output Channels |
|---|---|---|---|---|---|---|---|
| 1 | Conv2d 1 | Conv2d | 3 | 32 | 32 | 3 | 64 |
| 2 | BatchNorm2d | BatchNorm2d | - | 32 | 32 | 64 | 64 |
| 3 | Relu | Relu | - | 32 | 32 | 64 | 64 |
| 4 | Conv2d 2 | Conv2d | 3 | 32 | 32 | 64 | 64 |
| 5 | BatchNorm2d | BatchNorm2d | - | 32 | 32 | 64 | 64 |
| 6 | Relu | Relu | - | 32 | 32 | 64 | 64 |
| 7 | Conv2d 3 | Conv2d | 3 | 32 | 32 | 64 | 64 |
| 8 | BatchNorm2d | BatchNorm2d | - | 32 | 32 | 64 | 64 |
| 9 | Relu | Relu | - | 32 | 32 | 64 | 64 |
| 10 | Conv2d 4 | Conv2d | 3 | 32 | 32 | 64 | 64 |
| 11 | BatchNorm2d | BatchNorm2d | - | 32 | 32 | 64 | 64 |
| 12 | Relu | Relu | - | 32 | 32 | 64 | 64 |
| 13 | Conv2d 5 | Conv2d | 3 | 32 | 32 | 64 | 64 |
| 14 | BatchNorm2d | BatchNorm2d | - | 32 | 32 | 64 | 64 |
| 15 | Relu | Relu | - | 32 | 32 | 64 | 64 |
| 16 | Conv2d 6 | Conv2d | 3 | 32 | 32 | 64 | 128 |
| 17 | BatchNorm2d | BatchNorm2d | - | 32 | 32 | 128 | 128 |
| 18 | Relu | Relu | - | 32 | 32 | 128 | 128 |
| 19 | Conv2d 7 | Conv2d | 3 | 32 | 32 | 128 | 128 |
| 20 | BatchNorm2d | BatchNorm2d | - | 32 | 32 | 128 | 128 |
| 21 | Relu | Relu | - | 32 | 32 | 128 | 128 |
| 22 | Maxpool 1 | Maxpool | 2 | 32 | 16 | 128 | 128 |
| 23 | Conv2d 8 | Conv2d | 3 | 16 | 16 | 128 | 128 |
| 24 | BatchNorm2d | BatchNorm2d | - | 16 | 16 | 128 | 128 |
| 25 | Relu | Relu | - | 16 | 16 | 128 | 128 |
| 26 | Conv2d 9 | Conv2d | 3 | 16 | 16 | 128 | 128 |
| 27 | BatchNorm2d | BatchNorm2d | - | 16 | 16 | 128 | 128 |
| 28 | Relu | Relu | - | 16 | 16 | 128 | 128 |
| 29 | Conv2d 10 | Conv2d | 3 | 16 | 16 | 128 | 128 |
| 30 | BatchNorm2d | BatchNorm2d | - | 16 | 16 | 128 | 128 |
| 31 | Relu | Relu | - | 16 | 16 | 128 | 128 |
| 32 | Conv2d 11 | Conv2d | 3 | 16 | 16 | 128 | 128 |
| 33 | BatchNorm2d | BatchNorm2d | - | 16 | 16 | 128 | 128 |
| 34 | Relu | Relu | - | 16 | 16 | 128 | 128 |
| 35 | Conv2d 12 | Conv2d | 3 | 16 | 16 | 128 | 256 |
| 36 | BatchNorm2d | BatchNorm2d | - | 16 | 16 | 256 | 256 |
| 37 | Relu | Relu | - | 16 | 16 | 256 | 256 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 38 | Conv2d 13 | Conv2d | 3 | 16 | 16 | 256 | 256 |
| 39 | BatchNorm2d | BatchNorm2d | - | 16 | 16 | 256 | 256 |
| 40 | Relu | Relu | - | 16 | 16 | 256 | 256 |
| 41 | Maxpool 2 | Maxpool | 2 | 16 | 8 | 256 | 256 |
| 42 | Conv2d 14 | Conv2d | 3 | 8 | 8 | 256 | 256 |
| 43 | BatchNorm2d | BatchNorm2d | - | 8 | 8 | 256 | 256 |
| 44 | Relu | Relu | - | 8 | 8 | 256 | 256 |
| 45 | Conv2d 15 | Conv2d | 3 | 8 | 8 | 256 | 256 |
| 46 | BatchNorm2d | BatchNorm2d | - | 8 | 8 | 256 | 256 |
| 47 | Relu | Relu | - | 8 | 8 | 256 | 256 |
| 48 | Conv2d 16 | Conv2d | 3 | 8 | 8 | 256 | 256 |
| 49 | BatchNorm2d | BatchNorm2d | - | 8 | 8 | 256 | 256 |
| 50 | Relu | Relu | - | 8 | 8 | 256 | 256 |
| 51 | Conv2d 17 | Conv2d | 3 | 8 | 8 | 256 | 256 |
| 52 | BatchNorm2d | BatchNorm2d | - | 8 | 8 | 256 | 256 |
| 53 | Relu | Relu | - | 8 | 8 | 256 | 256 |
| 54 | Conv2d 18 | Conv2d | 3 | 8 | 8 | 256 | 512 |
| 55 | BatchNorm2d | BatchNorm2d | - | 8 | 8 | 512 | 512 |
| 56 | Relu | Relu | - | 8 | 8 | 512 | 512 |
| 57 | Conv2d 19 | Conv2d | 3 | 8 | 8 | 512 | 512 |
| 58 | BatchNorm2d | BatchNorm2d | - | 8 | 8 | 512 | 512 |
| 59 | Relu | Relu | - | 8 | 8 | 512 | 512 |
| 60 | Maxpool 3 | Maxpool | 2 | 8 | 4 | 512 | 512 |
| 61 | Conv2d 20 | Conv2d | 3 | 4 | 4 | 512 | 512 |
| 62 | BatchNorm2d | BatchNorm2d | - | 4 | 4 | 512 | 512 |
| 63 | Relu | Relu | - | 4 | 4 | 512 | 512 |
| 64 | Conv2d 21 | Conv2d | 3 | 4 | 4 | 512 | 512 |
| 65 | BatchNorm2d | BatchNorm2d | - | 4 | 4 | 512 | 512 |
| 66 | Relu | Relu | - | 4 | 4 | 512 | 512 |
| 67 | Conv2d 22 | Conv2d | 3 | 4 | 4 | 512 | 512 |
| 68 | BatchNorm2d | BatchNorm2d | - | 4 | 4 | 512 | 512 |
| 69 | Relu | Relu | - | 4 | 4 | 512 | 512 |
| 70 | Conv2d 23 | Conv2d | 3 | 4 | 4 | 512 | 512 |
| 71 | BatchNorm2d | BatchNorm2d | - | 4 | 4 | 512 | 512 |
| 72 | Relu | Relu | - | 4 | 4 | 512 | 512 |
| 73 | Conv2d 24 | Conv2d | 3 | 4 | 4 | 512 | 1024 |
| 74 | BatchNorm2d | BatchNorm2d | - | 4 | 4 | 1024 | 1024 |
| 75 | Relu | Relu | - | 4 | 4 | 1024 | 1024 |
| 76 | Conv2d 25 | Conv2d | 3 | 4 | 4 | 1024 | 1024 |
| 77 | BatchNorm2d | BatchNorm2d | - | 4 | 4 | 1024 | 1024 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 78 | Relu | Relu | - | 4 | 4 | 1024 | 1024 |
| 79 | Maxpool 4 | Maxpool | 2 | 4 | 2 | 1024 | 1024 |
| 80 | Linear | Linear | - | 1 | 1 | 1024 | 512 |
| 81 | Relu | Relu | - | 1 | 1 | 512 | 512 |
| 82 | Linear | Linear | - | 1 | 1 | 512 | 10 |
| | | | | | | | |

## 2.3 Training loss plot and test accuracy plot for final model



### 2.4 An ablation table, listing all factors that you tried to make improvement to your final model as well as the corresponding validation accuracy.

| Improvement| Detail | performance|

| Improvement | Detail | performance |
|---|---|---|
| Data normalization | : Normalizing input data makes training easier and more robust. Normalize the data to made it zero mean and fixed standard deviation. So I use the same transforms.Normalize() for both train and test dataset. | 78.89% |
| Data augmentation | Augment the training data using random crops, horizontal flips, etc. I use transforms.RandomHorizontalFlip() for my train dataset. train_transform = transforms.Compose([<br>transforms.RandomHorizontalFlip(),<br>transforms.ToTensor(),<br>transforms.Normalize(mean = (0.5, ), std = (0.5, )),<br>])<br><br>test_transform = transforms.Compose(<br>[transforms.ToTensor() ,<br>transforms.Normalize(mean = (0.5, ), std = (0.5, )),<br>]) | 78.89% |
| Deeper network. | I added a lot of convolutional layers for training to build a deeper network. And end up having an model with around 80-90 layers. | 70.23%, 80.93% |
| Normalization layers. | For each of them, I also added the normalization layers after conv layers (`nn.BatchNorm2d`). | 78.87% |
| Change optimizer | I tried optimizer like SGD and Adam. | 83.89% |
| Change learning rate | I tried the learning rate of 0.005 and 0.01 | 83.71% |
| Increase epochs | I found the accuracty is still increasing, so I increase the epochs to 70, and the accuracy is almost stable at that time | 70.23% |

# 3. Secret test set

Autograder Output

```
Q1 evaluation results:
Accuracy: 77.5 %
Accuracy of airplane : 77.0 %
Accuracy of automobile : 72.5 %
Accuracy of  bird : 71.5 %
Accuracy of   cat : 65.0 %
Accuracy of  deer : 80.5 %
Accuracy of   dog : 67.5 %
Accuracy of  frog : 83.5 %
Accuracy of horse : 85.5 %
Accuracy of  ship : 81.5 %
Accuracy of truck : 91.0 %


100%|██████████| 200/200 [00:15<00:00, 12.69it/s]
Q2 evaluation results:
Test metrics:
mean error 24.8
median error 16.2
accuracy at 11.25deg 38.3
accuracy at 22.5deg 61.5
accuracy at 30deg 71.0


------------------------------------------------------------
Results found for Q1, good work!
------------------------------------------------------------
------------------------------------------------------------
Results found for Q2, good work!
------------------------------------------------------------
```

# Problem2 Surface normal

## 2.Build on top of ImageNet pre-trained Model

### print(net)

```
MyModel(
  (layers): ModuleList(
    (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(
```

```
    (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)
```

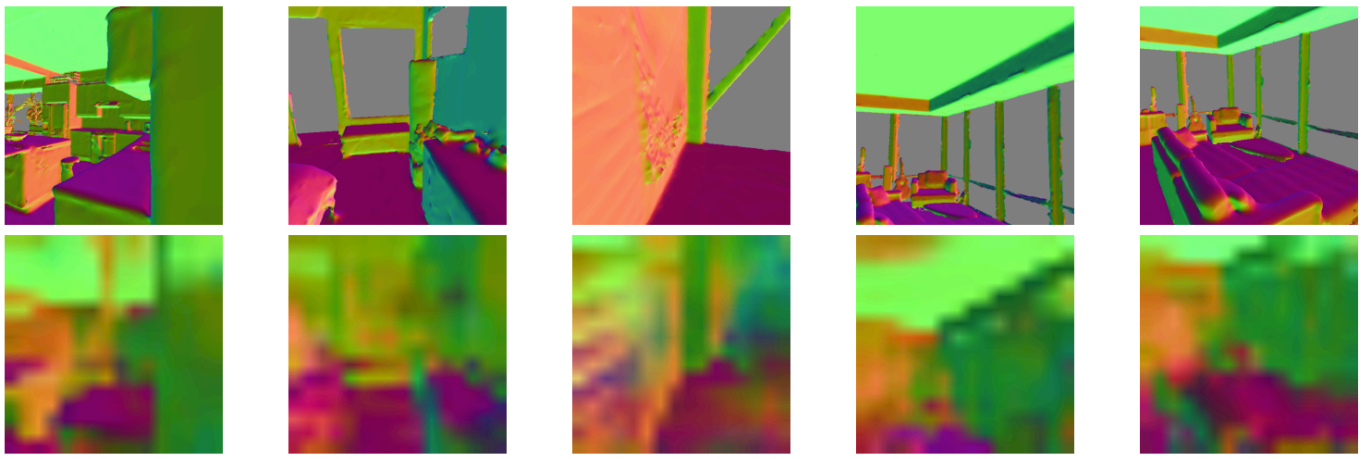
```
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (6): Sequential(
      (0): BasicBlock(
        (conv1): 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,
```

```
        (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)
      )
    )
    (8): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU(inplace=True)
    (11): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1))
    (12): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): ReLU(inplace=True)
    (14): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1))
    (15): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): ReLU(inplace=True)
    (17): Conv2d(64, 3, kernel_size=(1, 1), stride=(1, 1))
    (18): Upsample(size=(512, 512), mode=bilinear)
  )
)
```

## Final performance on validation set (all 5 metrics)

| mean angular error (lower the better) | median angular error (lower the better) | accuracies at 11.25 degree (higher the better) | accuracies at 22.5 degree (higher the better) | accuracies at 30 degree (higher the better) |
|---|---|---|---|---|
| 34.1 | 26.7 | 23.6 | 44.4 | 53.9 |

```
Validation loss (L1): 0.2499153511837507
Validation metrics: Mean 34.1, Median 26.7, 11.25deg 23.6, 22.5deg 44.4, 30deg 53.9
```



# 3. Increase your model output resolution

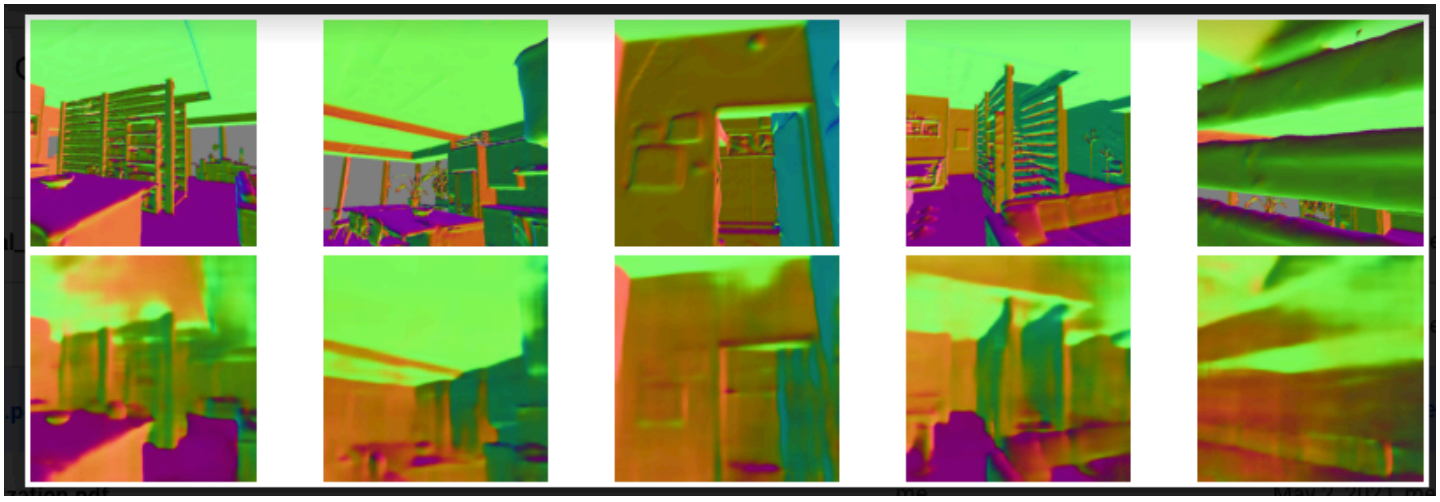
## Your best model. Include final performance on validation set (5 metrics).

### Final performance on validation set (all 5 metrics)

| mean angular error (lower the better) | median angular error (lower the better) | accuracies at 11.25 degree (higher the better) | accuracies at 22.5 degree (higher the better) | accuracies at 30 degree (higher the better) |
|---|---|---|---|---|
| 26.9 | 18.6 | 36.2 | 56.0 | 65.5 |

### Validation loss (L1): 0.1972911968254126

### Validation metrics: Mean 26.9, Median 18.6, 11.25deg 36.2, 22.5deg 56.0, 30deg 65.5



## An ablation table, listing all factors that you tried to make improvement to your final model as well as the validation performance.

I have tried several different models.

| Models and methods | details | validation performance |
|---|---|---|
| Improve on Resnet18 | I added several Relu() and Batch() layers, | Validation metrics: Mean 32.7, Median 26.8, 11.25deg 23.2, 22.5deg 43.6, 30deg 54.4 |
| Unet | I use the U-net | Validation metrics: Mean 34.8, Median 25.4, 11.25deg 26.1, 22.5deg 46.2, 30deg 55.3 |
| DeepLabv3+ + Adam optimizer | I use this optimizer torch.optim.Adam(model.parameters(),lr=0.005) | Validation metrics: Mean 30.2, Median 22.8, 11.25deg 27.7, 22.5deg 49.6, 30deg 60.0 |
| DeepLabv3+ + SGD optimizer | I use this optimizer optim.SGD(model.parameters(), lr=0.005, momentum=0.9, weight_decay=0.001) | Validation metrics: Mean 27.4, Median 19.4, 11.25deg 32.8, 22.5deg 55.0, 30deg 65.0 |
| Increase epochs | For the Unet, I have tried to use epcohs as large as 100. For the Deeplabv3+, I use around 30. | Validation metrics: Mean 34.8, Median 25.4, 11.25deg 26.1, 22.5deg 46.2, 30deg 55.3 |
| Change learning rate | I tried 0.005 and 0.01 | Validation metrics: Mean 30.2, Median 22.8, 11.25deg 27.7, 22.5deg 49.6, 30deg 60.0 |
| Change weight decay | I tried weight decay of 0, 0.001 to 0.0. Finally, I use 0.001 | Validation metrics: Mean 34.8, Median 25.4, 11.25deg 26.1, 22.5deg 46.2, 30deg 55.3 |
| Tried different activitiy layers | Tried different activitiy layers: Relu(), sigmoid | Validation metrics: Mean 34.8, Median 25.4, 11.25deg 26.1, 22.5deg 46.2, 30deg 55.3 |
| Tried normalization layers | Tried normalization layers: such as batchNorm2d().after conv() layers | Validation metrics: Mean 34.8, Median 25.4, 11.25deg 26.1, 22.5deg 46.2, 30deg 55.3 |
| Add more layers and build a deeper network | I start with the simpe one of res18 and I got a very complex network at the end. | Validation metrics: Mean 30.2, Median 22.8, 11.25deg 27.7, 22.5deg 49.6, 30deg 60.0 |
| Tried different ways of upsample. | Like, upsample once of scale 32 or upsample scale 2 for several times. | Validation metrics: Mean 30.2, Median 22.8, 11.25deg 27.7, 22.5deg 49.6, 30deg 60.0 |

Also, for the main model, I also Have the model store the current best performance, in case we will overfit,

## More Description for my final model.

I inplement the  a DeepLabv3+ decoder.

My optimizer is SGD with learning rate of 0.005

```
optimizer= optim.SGD(model.parameters(), lr=0.005, momentum=0.9, weight_decay=0.001)
```

And my loss function is

```
loss = (F.l1_loss(prediction, target, reduction = "none") * mask).nansum()/512/512
```

My model now looks like this:

```
Using cache found in /root/.cache/torch/hub/pytorch_vision_v0.9.0
MyModel(
  (deeplab): DeepLabV3(
    (backbone): IntermediateLayerGetter(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
      (layer1): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=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)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=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)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

          (relu): ReLU(inplace=True)
```

```
        )
        (2): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=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)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
      )
      (layer2): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
          (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=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): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=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)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
```

```
        )
        (2): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=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)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (3): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=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)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
      )
      (layer3): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=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)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

          )
```

```
        )
        (1): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (3): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (4): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)


          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
```

```
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (5): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (6): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (7): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
```

```
        )
        (8): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (9): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (10): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (11): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)


          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
```

```
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (12): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (13): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (14): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
```

```
        )
        (15): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (16): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (17): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (18): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)


          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
```

```
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (19): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (20): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
        (21): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
```

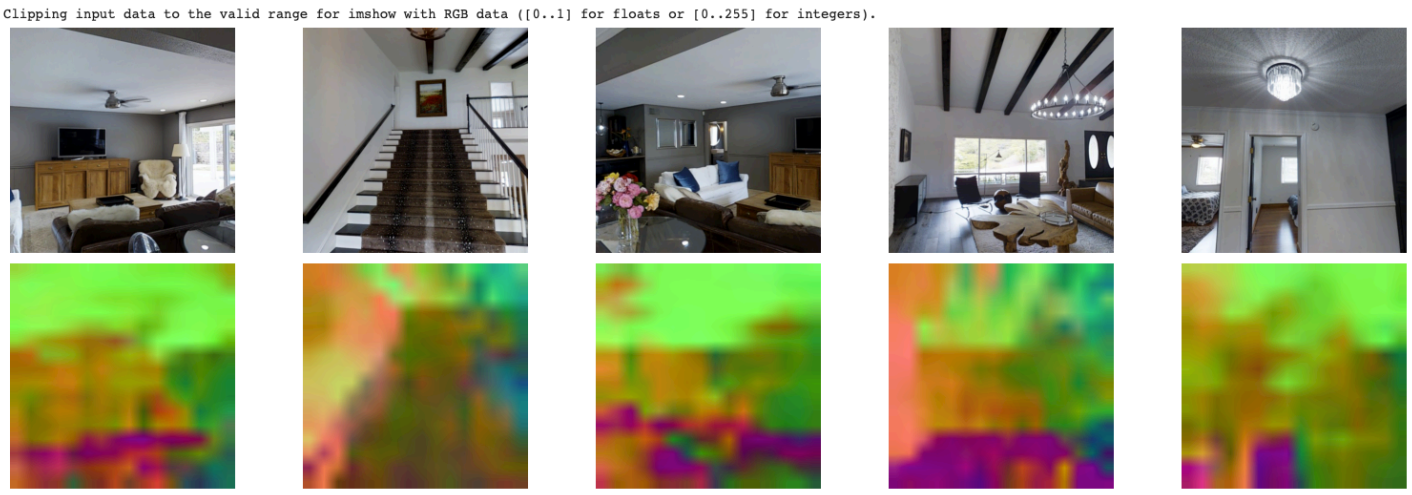
```
        )
        (22): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
      )
      (layer4): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2),
dilation=(2, 2), bias=False)
          (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(4, 4),
dilation=(4, 4), bias=False)
          (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
```

```
        )
        (2): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(4, 4),
dilation=(4, 4), bias=False)
          (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
        )
      )
    )
    (classifier): DeepLabHead(
      (0): ASPP(
        (convs): ModuleList(
          (0): Sequential(
            (0): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): ReLU()
          )
          (1): ASPPConv(
            (0): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(12, 12),
dilation=(12, 12), bias=False)
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): ReLU()
          )
          (2): ASPPConv(
            (0): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(24, 24),
dilation=(24, 24), bias=False)
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): ReLU()
          )
          (3): ASPPConv(
            (0): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(36, 36),
dilation=(36, 36), bias=False)
            (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): ReLU()

          )
```

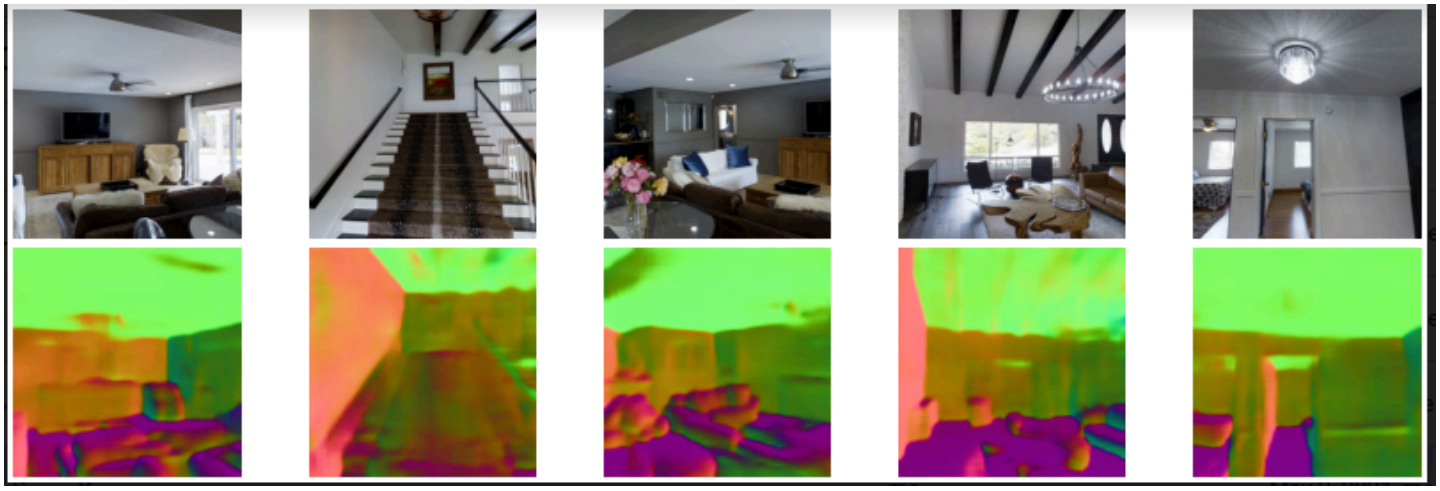
```
          (4): ASPPPooling(
            (0): AdaptiveAvgPool2d(output_size=1)
            (1): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (3): ReLU()
          )
        )
        (project): Sequential(
          (0): Conv2d(1280, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (2): ReLU()
          (3): Dropout(p=0.5, inplace=False)
        )
      )
      (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (3): ReLU()
      (4): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
    )
    (aux_classifier): FCNHead(
      (0): Conv2d(1024, 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()
      (3): Dropout(p=0.1, inplace=False)
      (4): Conv2d(256, 21, kernel_size=(1, 1), stride=(1, 1))
    )
  )
)
```

# 4. Visualize your prediction

### Visual comparisons of the output from part 2




### Visual comparisons of the output from part 3



### Visual comparisons of the output from part 2 and 3 on 2 images from the validation dataset, discuss your observations.

- We can see that the output from part3 is more smooth that the part2
- Also, we can see that the differences between different part of the pictures is most clear, have a better contour for part 3.
- We can see more details in the pictures generata by part 3.
- The output of part3 is very close to the origin pictures.

# 5. Secret test set

Autograder Results

Results Code Leaderboard

STUDENT
Xiao Tan

Autograder Output

```
Q1 evaluation results:
Accuracy: 77.5 %
Accuracy of airplane : 77.0 %
Accuracy of automobile : 72.5 %
Accuracy of  bird : 71.5 %
Accuracy of   cat : 65.0 %
Accuracy of  deer : 80.5 %
Accuracy of   dog : 67.5 %
Accuracy of  frog : 83.5 %
Accuracy of horse : 85.5 %
Accuracy of  ship : 81.5 %
Accuracy of truck : 91.0 %


100%|██████████| 200/200 [00:15<00:00, 12.69it/s]
Q2 evaluation results:
Test metrics:
mean error 24.8
median error 16.2
accuracy at 11.25deg 38.3
accuracy at 22.5deg 61.5
accuracy at 30deg 71.0


------------------------------------------------------------
Results found for Q1, good work!
------------------------------------------------------------
------------------------------------------------------------
Results found for Q2, good work!
------------------------------------------------------------
```

AUTOGRADER SCORE
**10.0 / 10.0**

| mean angular error (lower the better) | median angular error (lower the better) | accuracies at 11.25 degree (higher the better) | accuracies at 22.5 degree (higher the better) | accuracies at 30 degree (higher the better) |
|---|---|---|---|---|
| 24.8 | 16.2 | 38.3 | 61.5 | 71.0 |

## Rank as top 1 so far.

| RANK | SUBMISSION NAME | Q1: OVERALL ACCURACY | Q2: MEAN ANGULAR ERROR | Q2: MEDIAN ANGULAR ERROR | Q2: ACCURACY AT 11.25 DEG | Q2: ACCURACY AT 22.5 DEG | Q2: ACCURACY AT 30 DEG |
|---|---|---|---|---|---|---|---|
| **93** | xxx | **77.55** | **24.8** | **16.18** | **38.28** | **61.54** | **71.03** |
| 92 | Romo | 0 | 27.29 | 17.99 | 36.05 | 56.96 | 66.16 |
| 91 | Naiyo | 79.6 | 27.55 | 16.96 | 36.25 | 59.24 | 67.57 |