CMPT 762 X100, Fall 2024, Computer Vision

Project 2: Deep learning by PyTorch

Name: Wenhe Wang
Student Number: 301586596
Email: wwa118@sfu.ca

1. Improving BaseNet on CIFAR100

Best accuracy on validation set: 77%
Best accuracy on 10% test set showing in Kaggle leaderboard: 78.2%

Final model Architecture Table

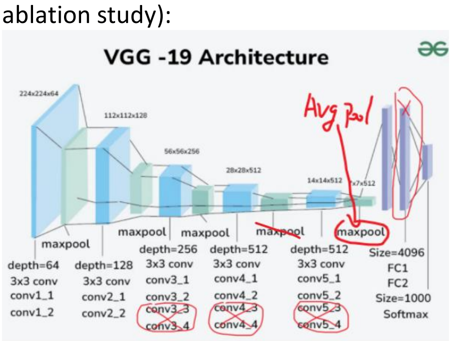
Layer No.	Layer Type	Kernel size	Input|Output Dimension	Input|Output Channels
1	Conv2d	3	128|128	3|64
2	BatchNorm2d	-	128|128	-
3	ReLU	-	128|128	-
4	Conv2d	3	128|128	64|64
5	BatchNorm2d	-	128|128	-
6	ReLU	-	128|128	-
7	Maxpool2d	2	128|64	
8	Conv2d	3	64|64	64|128
9	BatchNorm2d	-	64|64	-
10	ReLU	-	64|64	-
11	Conv2d	3	64|64	128|128
12	BatchNorm2d	-	64|64	-
13	ReLU	-	64|64	-
14	Maxpool2d	2	64|32	-
15	Conv2d	3	32|32	128|256
16	BatchNorm2d	-	32|32	-
17	ReLU	-	32|32	-
18	Conv2d	3	32|32	256|256
19	BatchNorm2d	-	32|32	-
20	ReLU	-	32|32	-
21	Maxpool2d	2	32|16	-
22	Conv2d	3	16|16	256|512
23	BatchNorm2d	-	16|16	-
24	ReLU	-	16|16	-
25	Conv2d	3	16|16	512|512
26	BatchNorm2d	-	16|16	-
27	ReLU	-	16|16	-
28	AdaptiveAvgPool2d	-	16|1	-

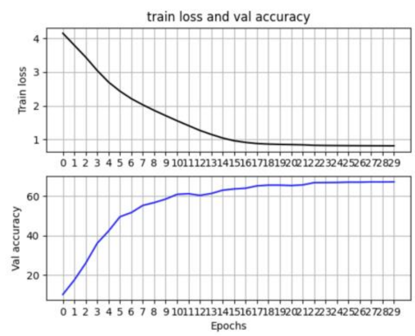
29	Flatten	-	1|512	-
30	Linear	-	512|4096	-
31	ReLU	-	4096|4096	-
32	Linear	-	4096|100	-

My customized model is inspired by VGG-19 and ResNet-18 but modified for improved performance and efficiency. It consists of five convolutional blocks, each with BatchNorm layers to stabilize training. MaxPooling layers are omitted to retain more spatial information, and AdaptiveAvgPool is applied after the fifth block for global feature extraction. The fully connected layers are reduced to streamline the model, and BatchNorm is removed from the linear layers. This design balances complexity and generalization. While testing popular models such as VGG-19, ResNet-18, and GoogleNet, my customized VGG-19 consistently outperformed them without modifying their original architectures.

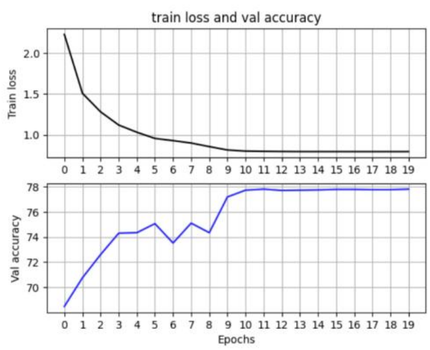
Training and Validation Loss and Accuracy

Training with optimized VGG-19 model + the normalized + resized 128x128 images (as the 14th step in ablation study):





Final round training with 700 epochs data augmentation (as the 15th step in ablation study):



Because the training plateaued after epoch 10 in our last round of training with the original images, so to prevent overfitting, I manually stopped the finetuning process at epoch 20.

Ablation Study

Dataset Size:

```
Train set size: 45000
Val set size: 5000
Test set size: 10000
```

Baseline Hyper-parameters:

```
# <<TODO#5>> Based on the val set performance, decide how many
# epochs are apt for your model.
# ---------
EPOCHS = 15
# ---------

IS_GPU = True
TEST_BS = 256
TOTAL_CLASSES = 100
TRAIN_BS = 32
PATH_TO_CIFAR100_SFU_CV = "/data/"
```

```
########################################################################
# 3. Define a Loss function and optimizer
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Here we use Cross-Entropy loss and SGD with momentum.
# The CrossEntropyLoss criterion already includes softmax within its
# implementation. That's why we don't use a softmax in our model
# definition.

import torch.optim as optim
criterion = nn.CrossEntropyLoss()

# Tune the learning rate.
# See whether the momentum is useful or not
optimizer = optim.SGD(net.parameters(), lr=0.005, momentum=0.9)

plt.ioff()
fig = plt.figure()
train_loss_over_epochs = []
val_accuracy_over_epochs = []
```

Baseline model (default BaseNet) :

```
[1] loss: 3.345
Accuracy of the network on the val images: 19 %
[2] loss: 3.262
Accuracy of the network on the val images: 21 %
[3] loss: 3.203
Accuracy of the network on the val images: 21 %
[4] loss: 3.155
Accuracy of the network on the val images: 22 %
[5] loss: 3.096
Accuracy of the network on the val images: 21 %
[6] loss: 3.066
Accuracy of the network on the val images: 23 %
[7] loss: 3.032
Accuracy of the network on the val images: 24 %
[8] loss: 2.992
Accuracy of the network on the val images: 24 %
[9] loss: 2.966
Accuracy of the network on the val images: 24 %
[10] loss: 2.931
Accuracy of the network on the val images: 23 %
[11] loss: 2.904
Accuracy of the network on the val images: 25 %
[12] loss: 2.889
Accuracy of the network on the val images: 25 %
[13] loss: 2.866
Accuracy of the network on the val images: 24 %
[14] loss: 2.838
Accuracy of the network on the val images: 24 %
[15] loss: 2.825
Accuracy of the network on the val images: 24 %
Finished Training
```

1. Baseline + data normalization ((0, 0, 0), (1, 1, 1))
Accuracy on val: 24% -> 26%

2. Baseline + data normalization ((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
Accuracy on val: 24% -> 26%

3. Baseline + data normalization + data augmentation:

```python
train_transform = transforms.Compose(
    [
        transforms.RandomCrop(size=32, padding=4),                 # Random cropping with padding
        transforms.RandomHorizontalFlip(),                         # Random horizontal flip
        transforms.RandomVerticalFlip(),                           # Random vertical flip
        transforms.RandomRotation(180),                            # Random rotation up to 180 degrees
        transforms.ColorJitter(brightness=0.2, contrast=0.2),      # Random gamma correction for brightness
        transforms.Grayscale(num_output_channels=3),               # Convert to grayscale
        transforms.ToTensor(),                                     # Convert image to tensor
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))     # Normalize to [-1, 1] range
    ]
)
test_transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

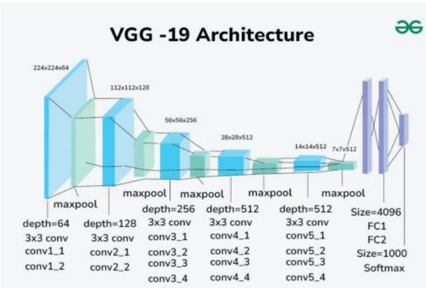
Accuracy on val: 24% -> 8%

The initial baseline may be too simple to capture features with multiple augmentations applied simultaneously. A better approach could be to apply each augmentation separately across the entire training set, resulting in 7 variations of the 45,000 images. However, this would significantly increase training time. For now, we'll hold off on this and focus on designing a new model architecture. Even with 45,000 images and multiple augmentations using default hyper-parameters, the baseline takes around 13 minutes to train with 15 epochs.

4. Baseline + data normalization + data augmentation (without grayscale):
Accuracy on val: 24% -> 19%

The grayscale method often used in segmentation to capture shape/edge information, isn't suitable for our classification task as it removes color information, which is crucial when combined with other augmentation methods.

5. VGG-19 + data normalization:
Accuracy on val: 26% -> 49%



6. VGG-19 (delete the middle FC layer) + data normalization:
Accuracy on val: 49% -> 51%

```python
self.fc_net = nn.Sequential(
    # nn.Dropout(0.2),
    nn.Linear(512 * 1 * 1, 4096),  # Adjusted for 1x1 input
    # nn.BatchNorm1d(4096),
    nn.ReLU(inplace=True),
    # nn.Dropout(0.2),
    # nn.Linear(4096, 4096),
    # nn.BatchNorm1d(4096),
    # nn.ReLU(inplace=True),
    # nn.Dropout(0.2),
    nn.Linear(4096, TOTAL_CLASSES)
)
```

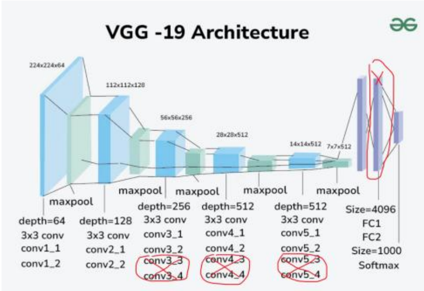
In the experiment with the classification head design, I tested applying BatchNorm1d after each FC layer and after all FC layers, along with adding dropout. However, none of these approaches improved accuracy; instead, accuracy significantly dropped to around 25-30%.

7. VGG-19 (delete the middle FC layer and Max Pooling layers) + data normalization:
Accuracy on val: 51% -> 40%

In the experiment with the VGG body design, I tested deleting Max Pooling layer after each convolution block and after all blocks. However, accuracy dropped to around 35-40%.

8. VGG-19 (delete the middle FC layer) + data normalization + He's initialization:
Accuracy on val: 51% -> 38%

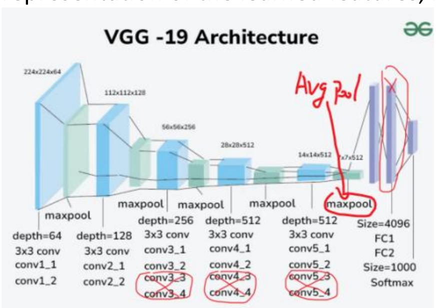
9. VGG-19 (delete the middle FC layer and 6 Conv layers and 6 Batch Norm layers) + data normalization:
Accuracy on val: 51% -> 56%



10. Step 9 + ReduceLROnPlateau scheduler + 30 epochs:
Accuracy on val: 56% -> 58%
The learning rate will decrease when the validation accuracy plateaus.

```
################################################################################
# 3. Define a Loss function and optimizer
# ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Here we use Cross-Entropy loss and SGD with momentum.
# The CrossEntropyLoss criterion already includes softmax within its
# implementation. That's why we don't use a softmax in our model
# definition.

import torch.optim as optim
from torch.optim.lr_scheduler import ReduceLROnPlateau
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)

# Tune the learning rate.
# See whether the momentum is useful or not
optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)

scheduler = ReduceLROnPlateau(optimizer, mode='max', factor=0.1, patience=3)

plt.ioff()
fig = plt.figure()
train_loss_over_epochs = []
val_accuracy_over_epochs = []
```

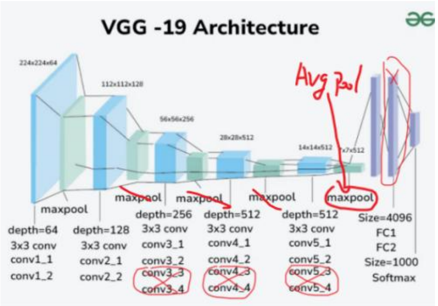
11. Step 10 - the maxpool after block 5 + AdaptiveAvgPool2d:
Accuracy on val: 58% -> 58%
Although the validation accuracy didn't improve, using average pooling after the fifth Conv block led to faster and more stable convergence to 58% accuracy during training compared to max pooling. Referring to ResNet-18 design, average pooling at the end provides a more generalized and smooth representation of the learned features, helps reduce overfitting.



12. Step 11 + image resize to 64:
Accuracy on val: 58% -> 62%
Training time increased significantly: training with 32x32 images took around 25-30 seconds per epoch, while 64x64 images took about 1.5 minutes per epoch.

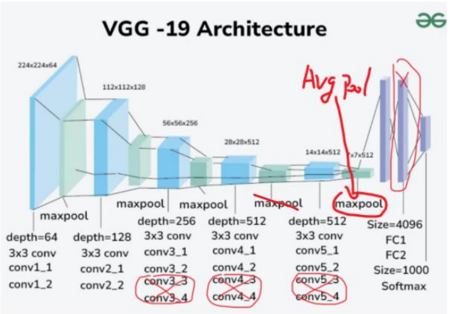
13. Step 11 + image resize to 128:
Accuracy on val: 58% -> 67%
Training with 128x128 images took about 4.5 minutes per epoch. To save time and resources, we will just keep our image size to 128x128 in our following steps without keep exploring resize to higher resolution.

14. Step 11 - the maxpool after block 2, 3, 4:
Accuracy on val: 58% -> 61%



With only the first one maxpool layer as shown in the image, training on one epoch and 64 batch size and one A100 GPU takes 3 min and 36.3 GB GPU RAM. So we have to keep the maxpool layer 1, 2 and 3 to reduce the computation cost.

15. Step 14 + maxpool after block 2, 3 + image resize to 128 + data augmentation(6 methods individually):
Accuracy on val: 67% -> 77%




```
# --------------------

train_transform = transforms.Compose(
    [
        transforms.Resize(160),
        transforms.RandomCrop(128),
        # transforms.RandomCrop(size=32, padding=4),             # Random cropping with padding
        transforms.RandomHorizontalFlip(),                      # Random horizontal flip
        transforms.RandomVerticalFlip(),                        # Random vertical flip
        transforms.RandomRotation(20),                         # Random rotation up to 10 degrees
        transforms.ColorJitter(brightness=0.2, contrast=0.2),   # Random gamma correction for brightness
        transforms.Grayscale(num_output_channels=3),            # Convert to grayscale
        transforms.ToTensor(),                                  # Convert image to tensor
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalize to [-1, 1] range
    ]
)
test_transform = transforms.Compose(
    [
    transforms.Resize(128),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
# --------------------
```

When training VGG-19, ResNet-18, VGG-19-ResNet-18 Mix, VGG-19-ResNet-18-Concat Mix, GoogleNet, or ResNetV2 with mixed data augmentations (random crop, horizontal/vertical flip, rotation, gamma correction, grayscale), the models showed little improvement, with accuracy stuck around 68-71%. However, when training with each augmentation separately for 100 epochs (a total of 700 epochs), the models showed significant improvement.

2. Transfer Learning

Model and Hyper-parameters

Because the size of ResNet-18, ResNet-34 and ResNet-50 are pretty similar, so I just used ResNet-50 as my backbone model. And as per the hyper-parameters and data augmentation methods I used in my first part, I just used the same settings here.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

```
#TODO: Vary Hyperparams

NUM_EPOCHS = 50
LEARNING_RATE = 0.01
BATCH_SIZE = 8
RESNET_LAST_ONLY = True #Fine tunes only the last layer. Set to False to fine tune entire network

root_path = '/data/' #If your data is in a different folder, set the path accodordingly

data_transforms = {
    'train': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        #TODO: Transforms.RandomResizedCrop() instead of CenterCrop(), RandomRoate() and Horizontal Flip()
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(10),
        transforms.ToTensor(),
        #TODO: Transforms.Normalize()
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ]),
    'test': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        #TODO: Transforms.Normalize()
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ]),
}
```

Using the ResNet-50 as a fixed feature extractor (RESNET_LAST_ONLY = True)

Training accuracy: 63.73%

```
TRAINING Epoch 35/50 Loss 0.2394 Accuracy 0.5870
TRAINING Epoch 36/50 Loss 0.2197 Accuracy 0.6267
TRAINING Epoch 37/50 Loss 0.2365 Accuracy 0.5950
TRAINING Epoch 38/50 Loss 0.2335 Accuracy 0.6037
TRAINING Epoch 39/50 Loss 0.2355 Accuracy 0.6137
TRAINING Epoch 40/50 Loss 0.2421 Accuracy 0.5880
TRAINING Epoch 41/50 Loss 0.2343 Accuracy 0.6090
TRAINING Epoch 42/50 Loss 0.2301 Accuracy 0.5997
TRAINING Epoch 43/50 Loss 0.2170 Accuracy 0.6170
TRAINING Epoch 44/50 Loss 0.2189 Accuracy 0.6243
TRAINING Epoch 45/50 Loss 0.2381 Accuracy 0.6087
TRAINING Epoch 46/50 Loss 0.2279 Accuracy 0.6177
TRAINING Epoch 47/50 Loss 0.2192 Accuracy 0.6217
TRAINING Epoch 48/50 Loss 0.2105 Accuracy 0.6330
TRAINING Epoch 49/50 Loss 0.2209 Accuracy 0.6267
TRAINING Epoch 50/50 Loss 0.2157 Accuracy 0.6373
Finished Training
```

Test accuracy: 38.34%

```
test(model, criterion)

Test Loss: 0.5254 Test Accuracy 0.3834
```

Fine-tuning the whole network (RESNET_LAST_ONLY = False)

After many experiments, this hyper-parameter settings worked the best.

```python
import torch.optim as optim
from torch.optim.lr_scheduler import ReduceLROnPlateau

#TODO: Vary Hyperparams

NUM_EPOCHS = 12
LEARNING_RATE = 0.01
BATCH_SIZE = 128
RESNET_LAST_ONLY = False #Fine tunes only the last layer. Set to False to fine tune entire network

plt.ioff()
fig = plt.figure()
train_loss_over_epochs = []
val_accuracy_over_epochs = []

root_path = '/data/' #If your data is in a different folder, set the path accodordingly

data_transforms = {
    'train': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        #TODO: Transforms.RandomResizedCrop() instead of CenterCrop(), RandomRoate() and Horizontal Flip()
        # transforms.RandomResizedCrop(256),
        transforms.RandomHorizontalFlip(),
        # transforms.RandomVerticalFlip(),
        transforms.RandomRotation(20),
        # transforms.ColorJitter(brightness=0.2, contrast=0.2),
        transforms.ToTensor(),
        #TODO: Transforms.Normalize()
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ]),
    'test': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        #TODO: Transforms.Normalize()
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ]),
}
```

Training accuracy: 99.37%

```
TRAINING Epoch 1/25 Loss 0.0417 Accuracy 0.0260
TRAINING Epoch 2/25 Loss 0.0326 Accuracy 0.2517
TRAINING Epoch 3/25 Loss 0.0226 Accuracy 0.4780
TRAINING Epoch 4/25 Loss 0.0153 Accuracy 0.6423
TRAINING Epoch 5/25 Loss 0.0104 Accuracy 0.7820
TRAINING Epoch 6/25 Loss 0.0072 Accuracy 0.8617
TRAINING Epoch 7/25 Loss 0.0049 Accuracy 0.9177
TRAINING Epoch 8/25 Loss 0.0035 Accuracy 0.9543
TRAINING Epoch 9/25 Loss 0.0024 Accuracy 0.9713
TRAINING Epoch 10/25 Loss 0.0017 Accuracy 0.9853
TRAINING Epoch 11/25 Loss 0.0014 Accuracy 0.9890
TRAINING Epoch 12/25 Loss 0.0010 Accuracy 0.9937
```

Test accuracy: 64.13%

```
test(model, criterion)

Test Loss: 0.0110 Test Accuracy 0.6413
```

Visualizing the model predictions

class: 199.Winter_Wren predicted: 199.Winter_Wren

class: 096.Hooded_Oriole predicted: 096.Hooded_Oriole

class: 169.Magnolia_Warbler predicted: 176.Prairie_Warbler

class: 150.Sage_Thrasher predicted: 150.Sage_Thrasher

class: 106.Horned_Puffin predicted: 106.Horned_Puffin

Reference

[1] VGG-Net Architecture Explained. (2024, June 07). Retrieved October 12, 2024, from https://www.geeksforgeeks.org/vgg-net-architecture-explained/

[2] Big Transfer (BiT): General Visual Representation Learning. (2021, June 18). Retrieved October 12, 2024, from https://github.com/google-research/big_transfer

[3] Structure of the Resnet-18 Model. (2022, December). Retrieved October 12, 2024, from https://www.researchgate.net/figure/Structure-of-the-Resnet-18-Model_fig1_366608244
[4] Pytorch-cifar100. (2022, March 27). Retrieved October 12, 2024, from https://github.com/weiaicunzai/pytorch-cifar100
[5] GoogleNet. (2020, May 5). Retrieved October 12, 2024, from https://github.com/pytorch/vision/blob/6db1569c89094cf23f3bc41f79275c45e9fcb3f3/torchvision/models/googlenet.py#L62
[6] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky (2009). Retrieved October 12, 2024, from https://www.cs.toronto.edu/%7Ekriz/cifar.html
[7] ResNet (34, 50, 101)…what actually it is ?, chandini (2020). Retrieved October 13, 2024, from https://medium.com/@aschandinip/resnet-34-50-101-what-actually-it-is-c63da24ba695