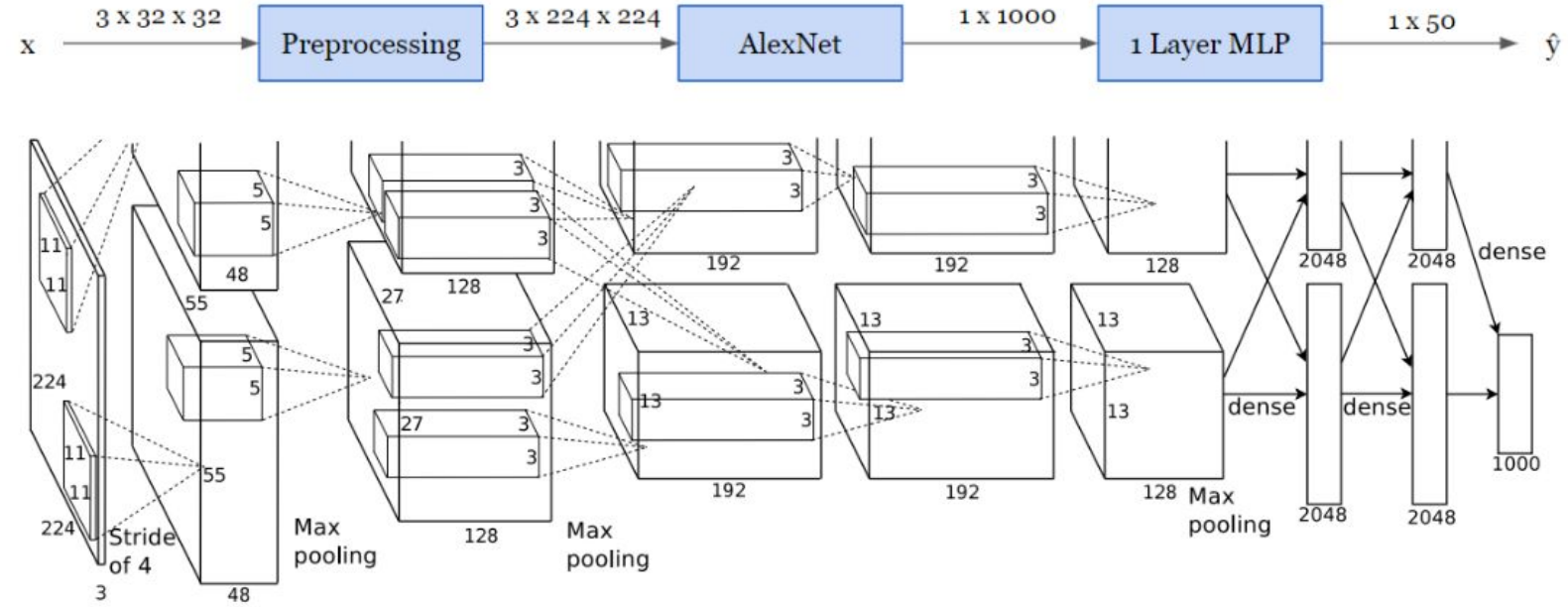
DLCV HW1 Report

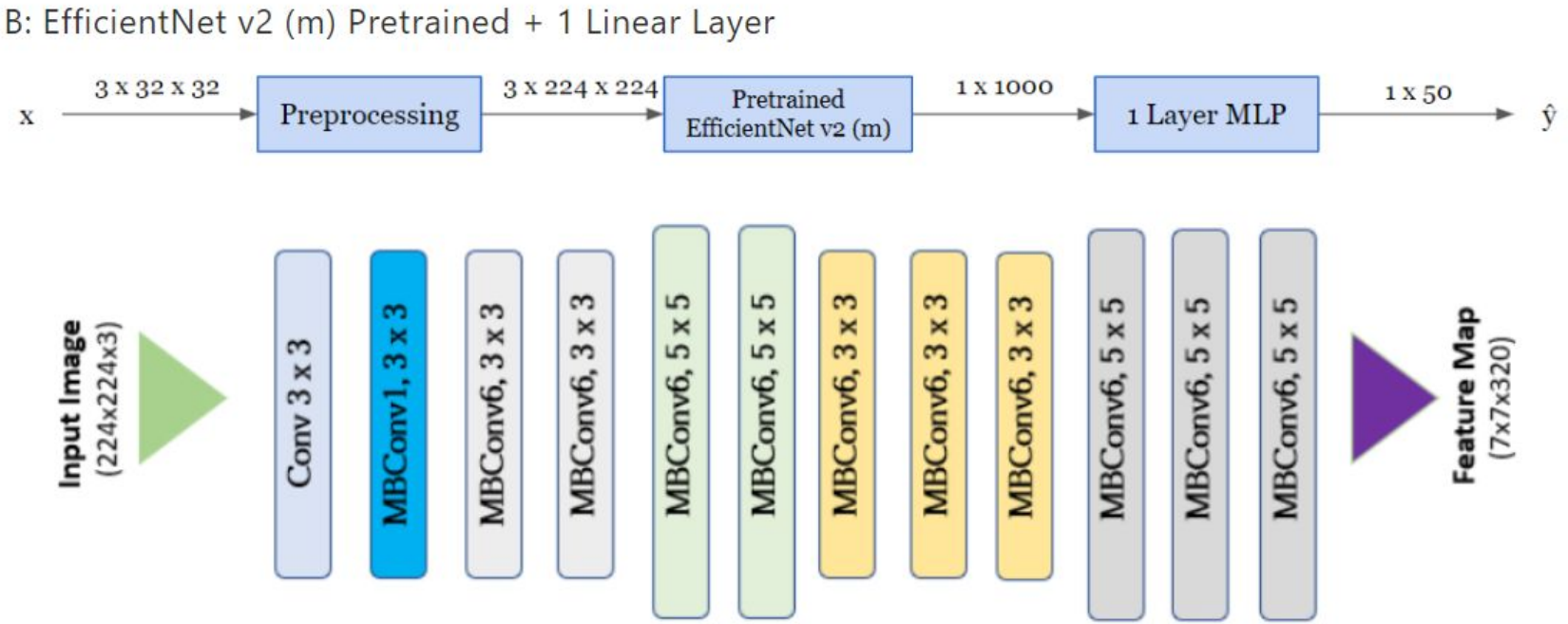
NTU CSIE, R12922051
資工碩一 陳韋傑

Q1-1: Network Architecture

A: AlexNet + 1 Linear Layer



Q1-1: Network Architecture



Q1-2: Accuracy

Model	A	B
Validation Accuracy	0.646	0.8944

Q1-3: Implementation of model A

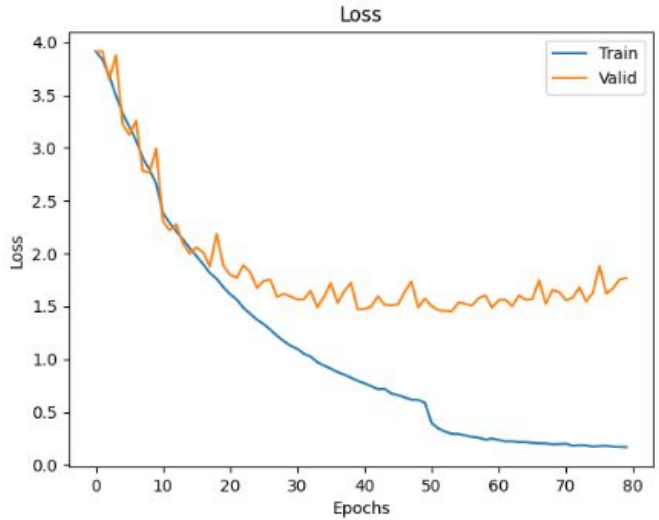
Since model A needs to train from scratch, I choose a relatively small model (AlexNet) for easier trainning.

Parameters

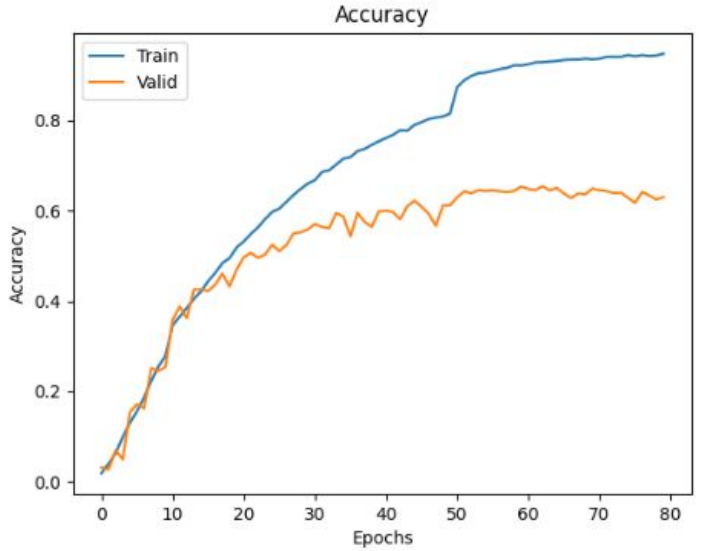
- epochs: 200
- bs: 64 (batch size)
- lr: 0.1 (learning rate)
- wd: 0.001 (weight decay)
- optimizer: SGD
- milestones: [10, 50] (for scheduler)
- gamma: 0.5 (for scheduler)
- scheduler: MultiStepLR
- Early stopping at epoch 84, best valid loss at epoch 54 (patience = 30)
- Data Augmentations (Train):
  - Resize(size=[256, 256])
  - RandomCrop(size=(224, 224))
  - RandomHorizontalFlip(p=0.5)
  - Normalize() # with ImageNet Settings
- Data Augmentations (Test):
  - Resize(size=[224, 224])
  - Normalize() # with ImageNet Settings

Q1-3: Implementation of model A

Loss



Accuracy



Q1-4: Implementation of model B

Instead of AlexNet, I choose to use a stronger model, pretrained EfficientNet v2 (m), as backbone model or feature extractor, then follows by one linear layer.

Model Comparison

AlexNet is an early and simple CNN architecture. On the other hand, EfficientNet, a more recent model, employs a compound scaling method, balancing depth, width, and resolution for efficiency.

AlexNet lacks a systematic scaling approach and is computationally intensive, while EfficientNet achieves high accuracy with fewer parameters and computations.

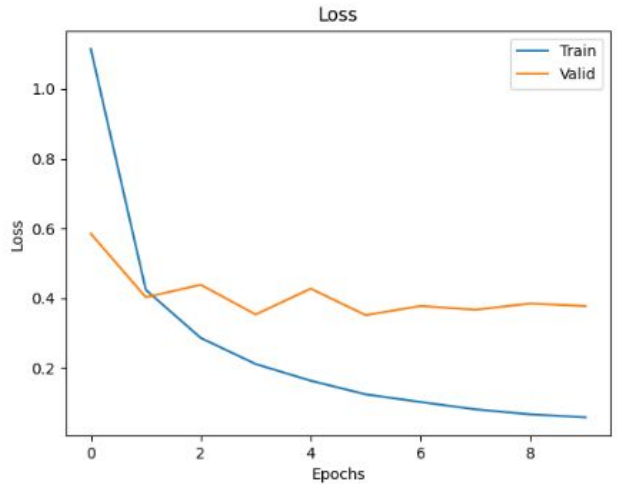
Q1-4: Implementation of model B

Parameters

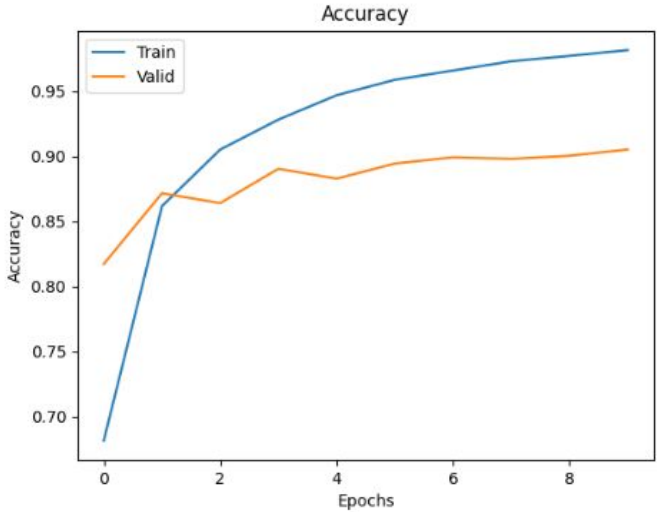
- epochs: 200
- bs: 32 (batch size)
- lr: 0.01 (learning rate)
- wd: 0.0005 (weight decay)
- optimizer: SGD
- milestones: [50] (for scheduler)
- gamma: 0.5 (for scheduler)
- scheduler: MultiStepLR
- Early stopping at epoch 16, best valid loss at epoch 6 (patience = 10)
- Data augmentations are the same as model A

Q1-4: Implementation of model B

Loss



Accuracy



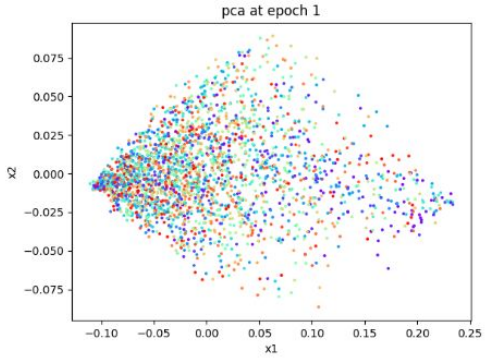
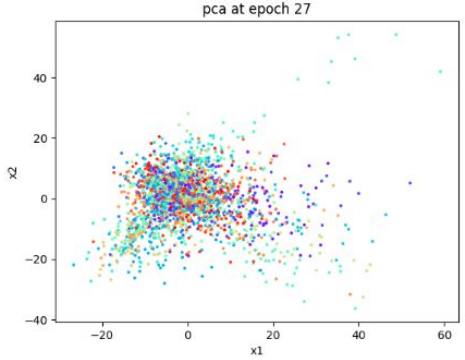
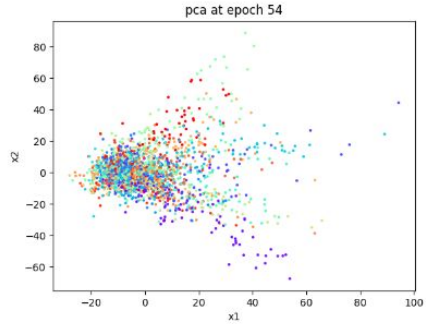
Q1-5: Visualization (Model A, PCA)

The following paragraphs are discussing PCA results at the end of trainning phase (the right figure, epoch = 54), for better understanding, PCA results at the beginning and in the middle are also included but not discussed here.

At the first sight, you might not see a very good clustering result. However, if we look closely, we can see that some points of the same color (which means they’re in the same cluster) are distributed closely.

For example, the purple cluster at the bottom of the figure, or the red / light green cluter around the upper-left area of the figure, and so on. (See appendix for bigger figure)

Q1-5: Visualization (Model A, PCA)

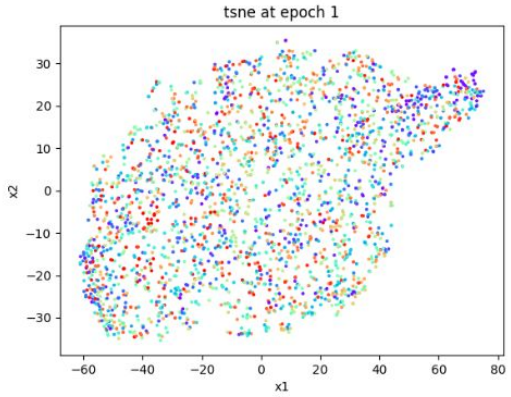
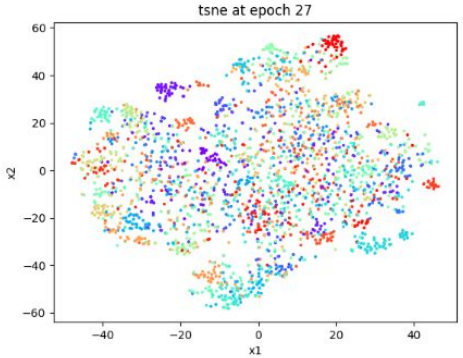
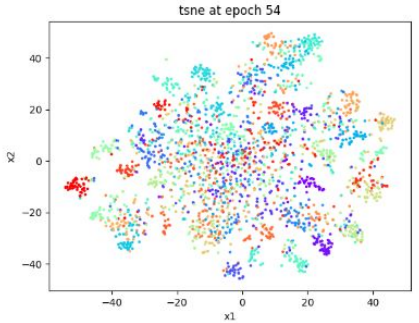
Beginning	Middle	End
		

Q1-6: Visualization (Model A, t-SNE)

We can clearly see that at the beginning of trainning phase, the latent vector cannot represent data and the clustering result is bad.

As the trainning progresses, the latent vector gradually learns a better representation, hence the clustering result becomes better. At the end the cluster result is quite good.

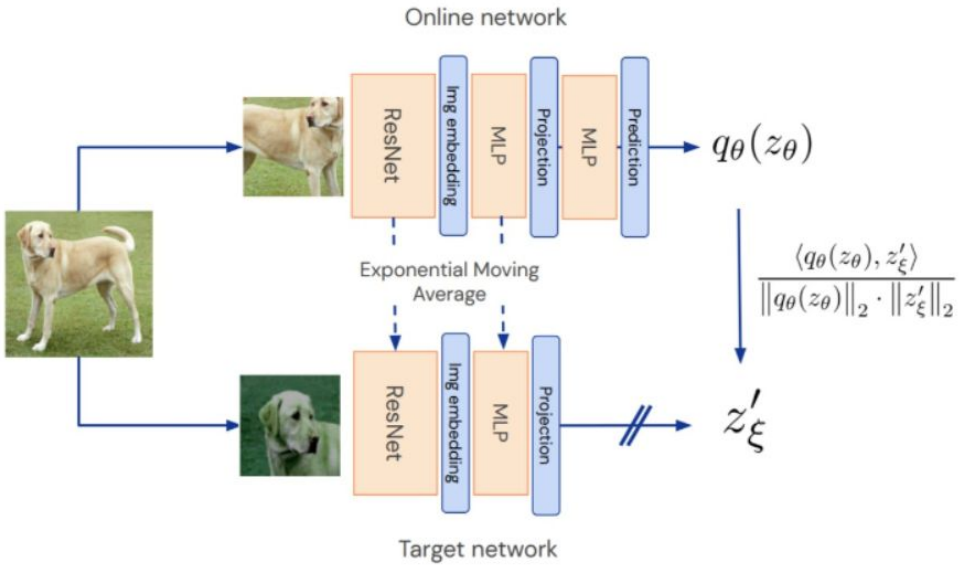
Q1-6: Visualization (Model A, t-SNE)

Beginning	Middle	End
		

Q2-1: SSL Implementation

I used BYOL(Bootstrap Your Own Latent) for self-supervised learning method. BYOL relies on two neural networks, referred to as online and target networks, that interact and learn from each other.

BYOL trains the online network to predict the target network representation of the same image under a different augmented view, then updates the target network with a slow-moving average of the online network.



Q2-1: SSL Implementation

Parameters

- epochs: 800 (using model from epoch 500)
- bs: 128 (batch size)
- lr: 0.0003 (learning rate)
- wd: 0.000001 (weight decay)
- optimizer: Adam
- Data Augmentations (Train):
  - Resize(size=[160, 160])
  - RandomCrop(size=(128, 128))
  - Normalize() # with ImageNet Settings
- Data Augmentations (Test):
  - Resize(size=[160, 160])
  - CenterCrop(size=(128, 128))
  - Normalize() # with ImageNet Settings

Q2-2: Office Home Dataset Results

- The classifier is fixed across all the settings: 4 layer MLP with input & output dimension = 1000, dropout = 0.5 and use Leaky ReLU as activation function, initialized by HE initialization

- The hyperparameters and data augemtation are also fixed through out all the settings for comparison and convenience

Q2-2: Office Home Dataset Results

- The settings are listed in the following figure:

Setting	Pre-training (Mini-ImageNet)	Fine-tuning (Office-Home dataset)
A	-	Train full model (backbone + classifier)
B	w/ label (TAs have provided this backbone)	Train full model (backbone + classifier)
C	w/o label (Your SSL pre-trained backbone)	Train full model (backbone + classifier)
D	w/ label (TAs have provided this backbone)	Fix the backbone. Train classifier only
E	w/o label (Your SSL pre-trained backbone)	Fix the backbone. Train classifier only

- The validation accuracies are:

Setting	A	B	C	D	E
Validation Accuracy	0.4039	0.4581	0.4507	0.2635	0.4384

Q2-2: Office Home Dataset Results

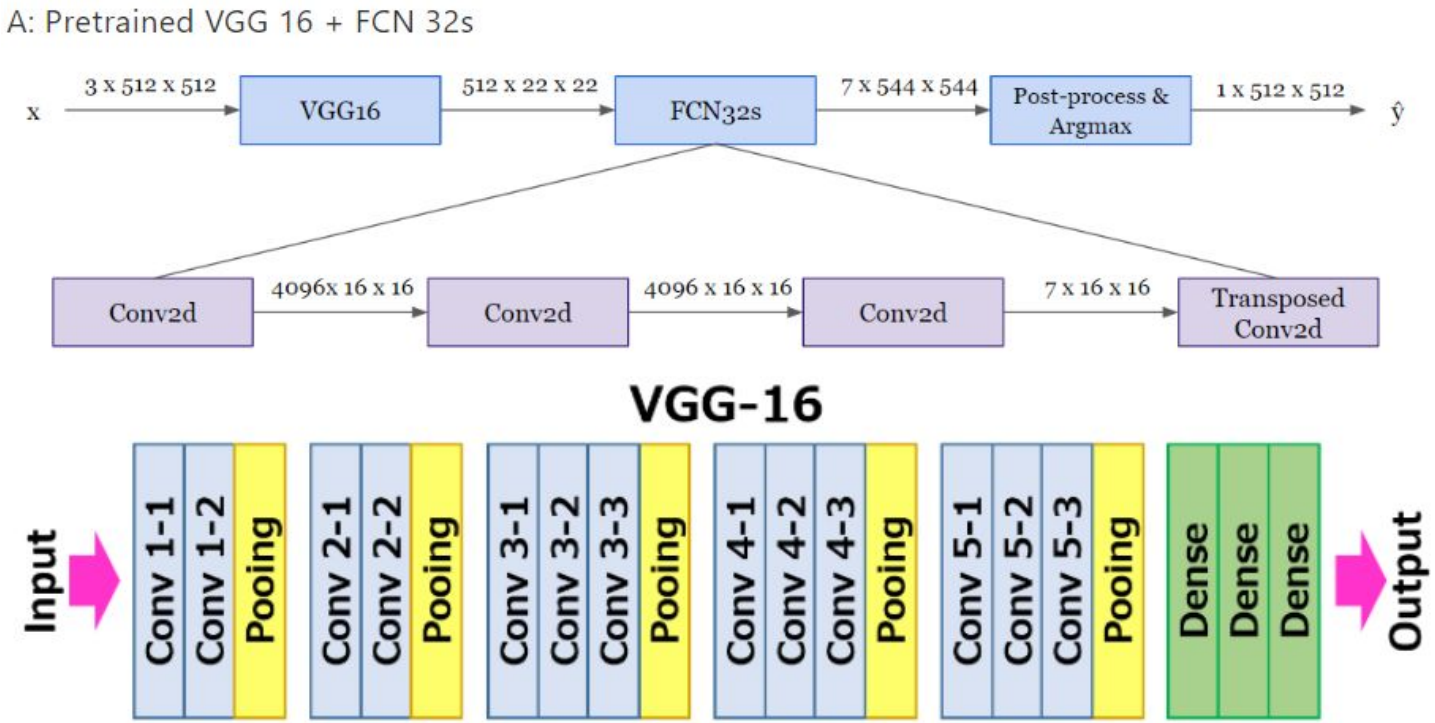
Parameters

- epochs: 300
- bs: 64 (batch size)
- lr: 0.0005 (learning rate)
- wd: 0.001 (weight decay)
- optimizer: Adam
- milestones: [30, 100, 200] (for scheduler)
- gamma: 0.5 (for scheduler)
- scheduler: MultiStepLR
- Data Augmentations (Train):
  - Resize(size=[160, 160])
  - RandomCrop(size=(128, 128))
  - RandomHorizontalFlip(p=0.5)
  - RandomGrayscale(p=0.5)
  - Normalize() # with ImageNet Settings
- Data Augmentations (Test):
  - Resize(size=[160, 160])
  - CenterCrop(size=(128, 128))
  - Normalize() # with ImageNet Settings

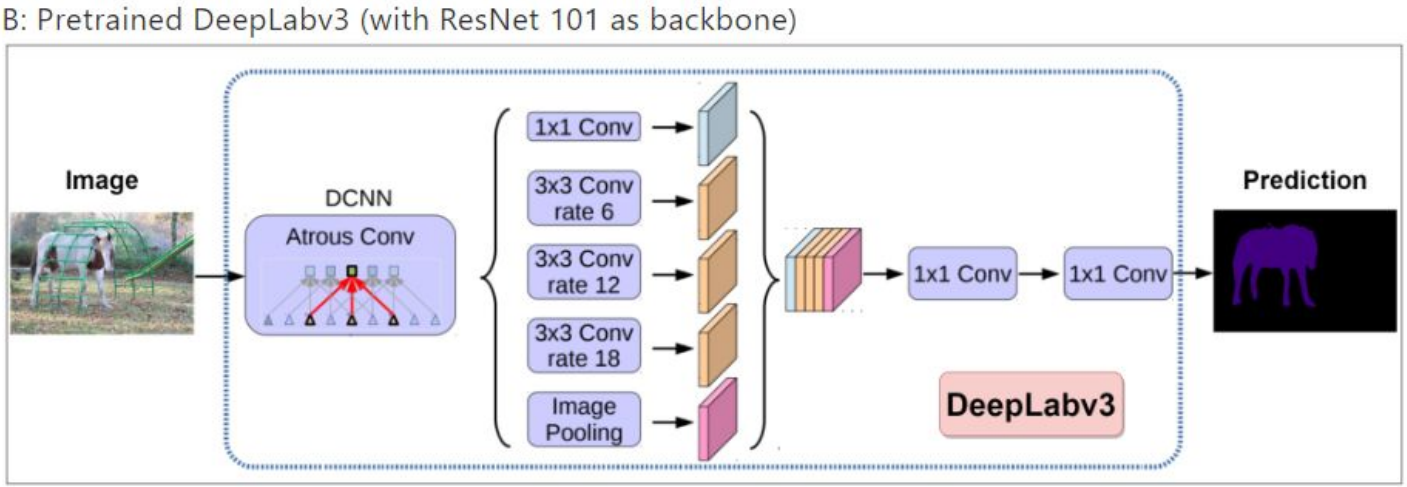
Discussion

- A has 0.4 accuracy: Train whole model from scratch can still perform reasonably well
- B ~= C: Good SSL can almost achieve performance like supervised learning, thus SSL is a decent method when you have no labeled data
- D has only 0.26 accuracy: Not sure why, but I've done several experiments and they were all not good
- E >> D: Maybe because they were on different positions in the error space, thus they will converge to different results
- C > E and B > D: Fix backbone will somehow hurt model preformance

Q3-1: Network Architecture of Model A



Q3-2: Network Architecture of Model B



Model Comparison

VGG16 + FCN32 combines VGG16 with the Fully Convolutional Network, adapting a classification network for semantic segmentation. Here VGG16 acts as feature extractor and FCN32 upsamples the coarse output to match the input image size.

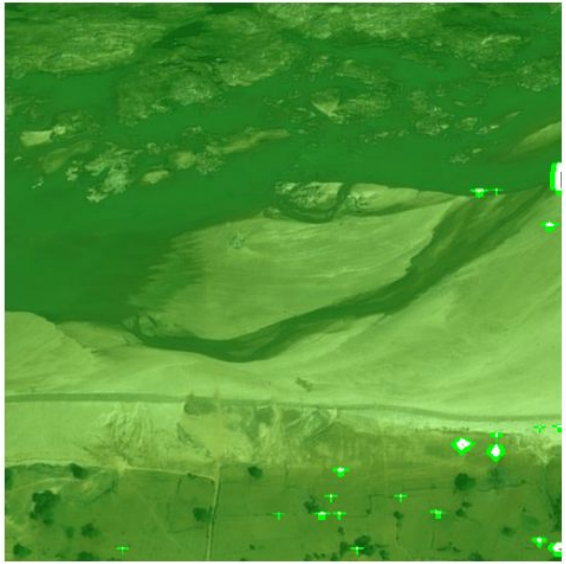
DeepLabv3 is a dedicated semantic segmentation model that employs atrous convolutions, a more complex architecture with multi-scale features, and an advanced decoder. It integrates contextual information effectively, using dilated convolutions to capture a broader context while preserving fine details.

Q3-3: mIoUs

Model	A	B
Validation mIoU	0.6981	0.7397

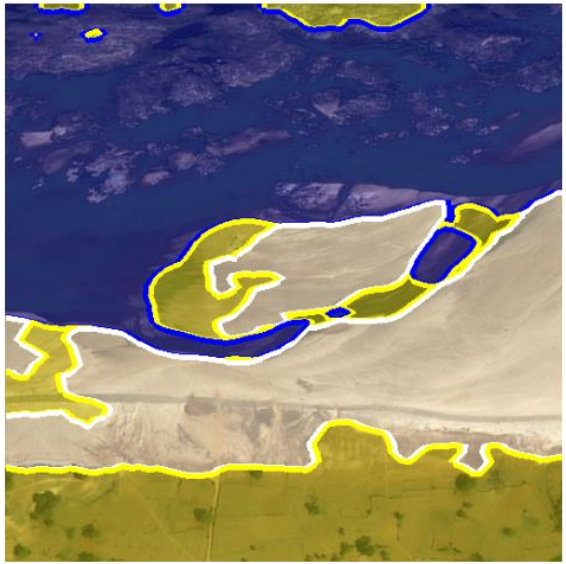
Q3-4: Predicted Segmentation

At the beginning (Epoch = 1, miou = 0.01)

0013_sat.jpg	0062_sat.jpg	0104_sat.jpg
		

Q3-4: Predicted Segmentation

In the middle (Epoch = 9, miou = 0.68)

0013_sat.jpg	0062_sat.jpg	0104_sat.jpg
		

Q3-4: Predicted Segmentation

At the End (Epoch = 18, miou = 0.74)

0013_sat.jpg	0062_sat.jpg	0104_sat.jpg
		

Q3-4: Predicted Segmentation

Ground Truth

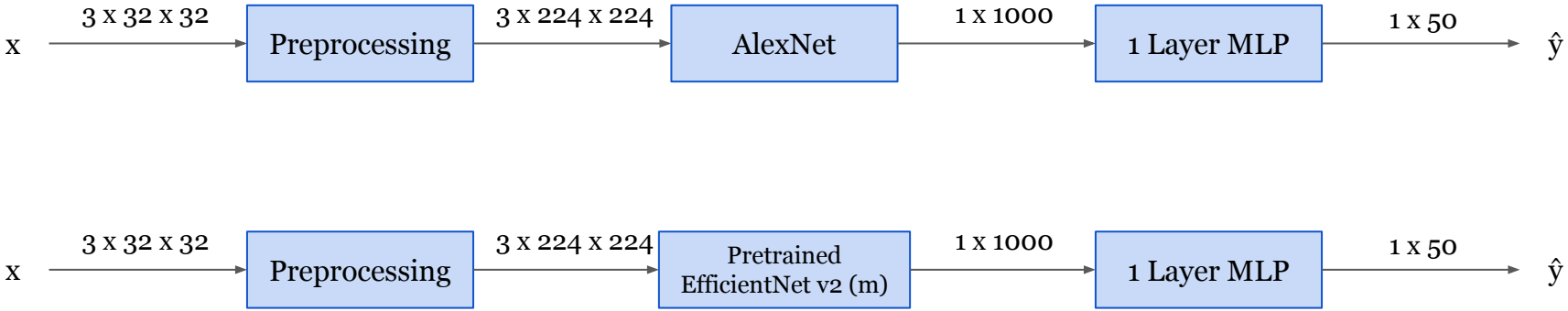
0013_sat.jpg	0062_sat.jpg	0104_sat.jpg
		

References

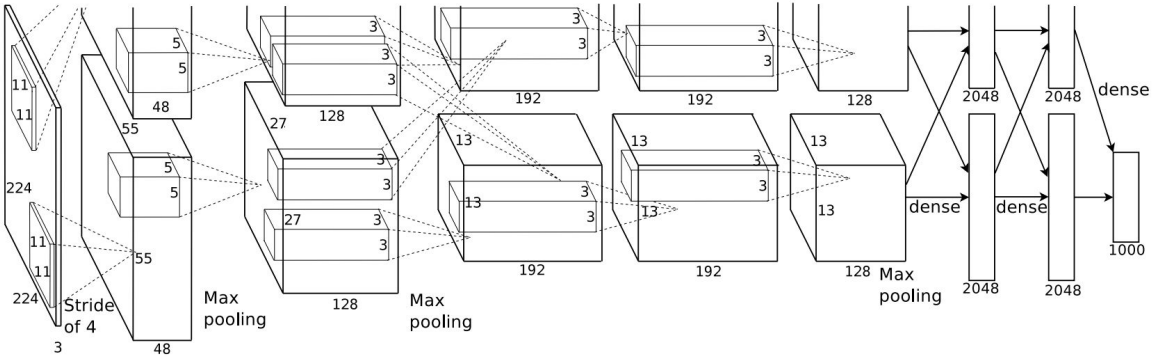
- Problem 2
  - byol-pytorch
- Problem 3
  - fcn32-pytorch
  - pytorch-fcn
  - pytorch_vision_deeplabv3_resnet101
- I also use ChatGPT to help with my report with some modifications made by myself

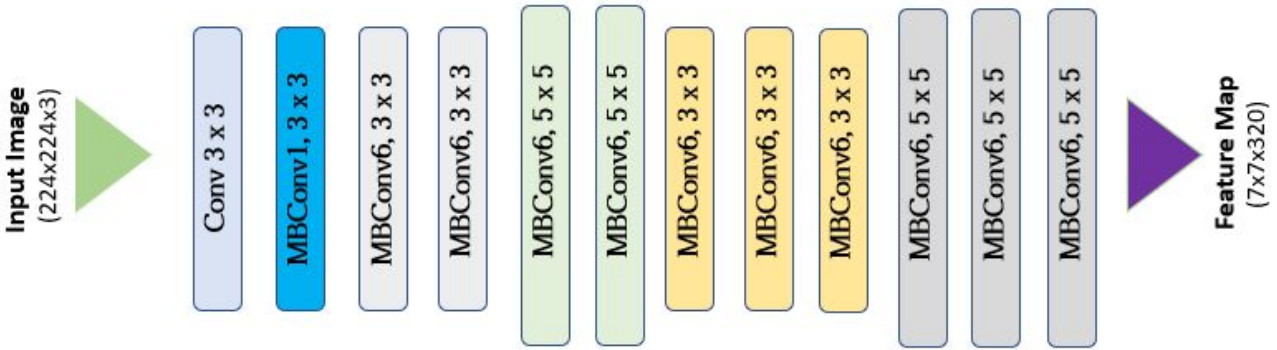
Appendix: Model Graphs

HW1-1


x
3 x 32 x 32
Preprocessing
3 x 224 x 224
AlexNet
1 x 1000
1 Layer MLP
1 x 50
ŷ
x
3 x 32 x 32
Preprocessing
3 x 224 x 224
Pretrained
EfficientNet v2 (m)
1 x 1000
1 Layer MLP
1 x 50
ŷ

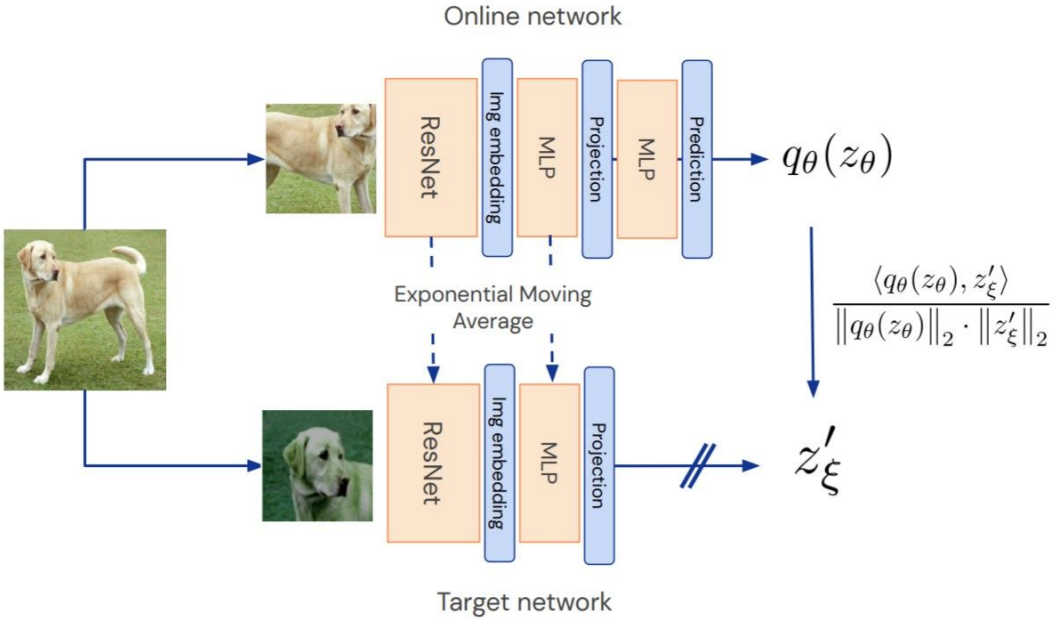
HW1-1


224
224
3
11
11
Stride
of 4
55
48
5
5
48
Max
pooling
27
27
128
3
3
3
3
3
3
128
Max
pooling
13
13
13
13
192
3
3
3
3
192
192
3
3
3
3
128
128
Max
pooling
dense
dense
dense
2048
2048
2048
2048
1000

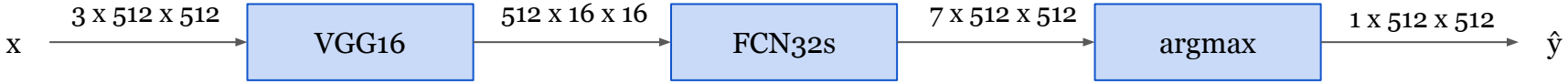

Input Image
(224x224x3)
Conv 3 x 3
MBConv1, 3 x 3
MBConv6, 3 x 3
MBConv6, 3 x 3
MBConv6, 5 x 5
MBConv6, 5 x 5
MBConv6, 3 x 3
MBConv6, 3 x 3
MBConv6, 3 x 3
MBConv6, 5 x 5
MBConv6, 5 x 5
MBConv6, 5 x 5
Feature Map
(7x7x320)

HW1-2


x
3 x h x w
Preprocessing
3 x 128 x 128
ResNet 50
1 x 1000
4 Layer MLP
(Out = 1000)
1 x 50
ŷ


Online network
ResNet
Img embedding
MLP
Projection
MLP
Prediction
$q_\theta(z_\theta)$
Exponential Moving Average
$\frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\|q_\theta(z_\theta)\|_2 \cdot \|z'_\xi\|_2}$
ResNet
Img embedding
MLP
Projection
z'_ξ
Target network

HW1-3


x
3 x 512 x 512
VGG16
512 x 16 x 16
FCN32s
7 x 512 x 512
argmax
1 x 512 x 512
ŷ

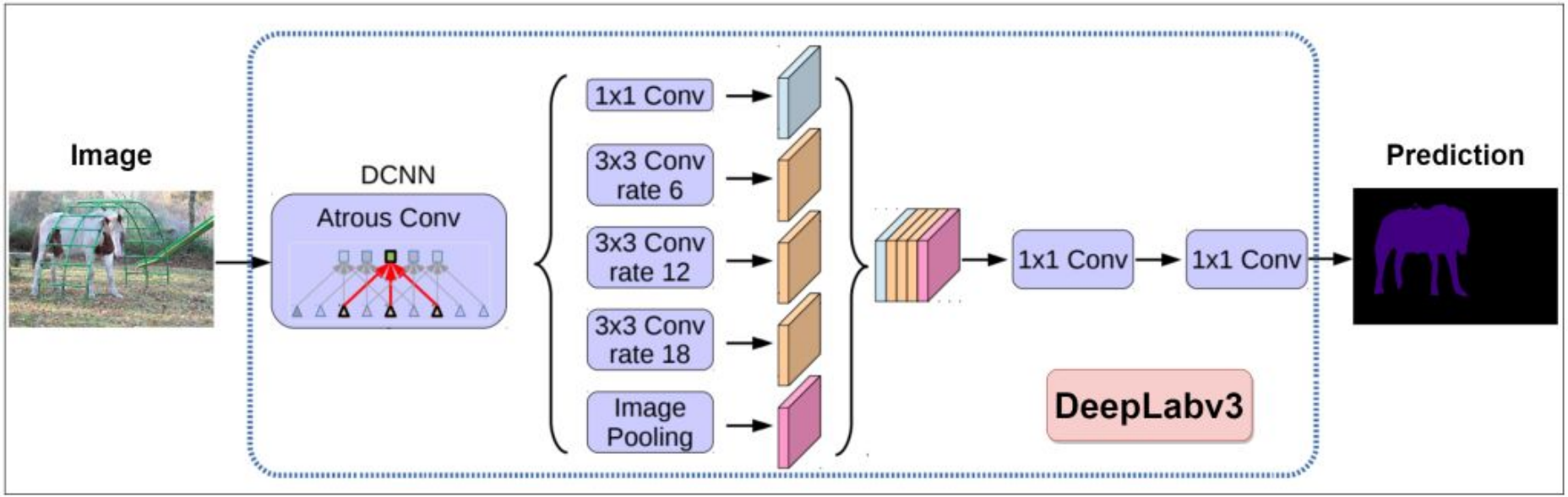
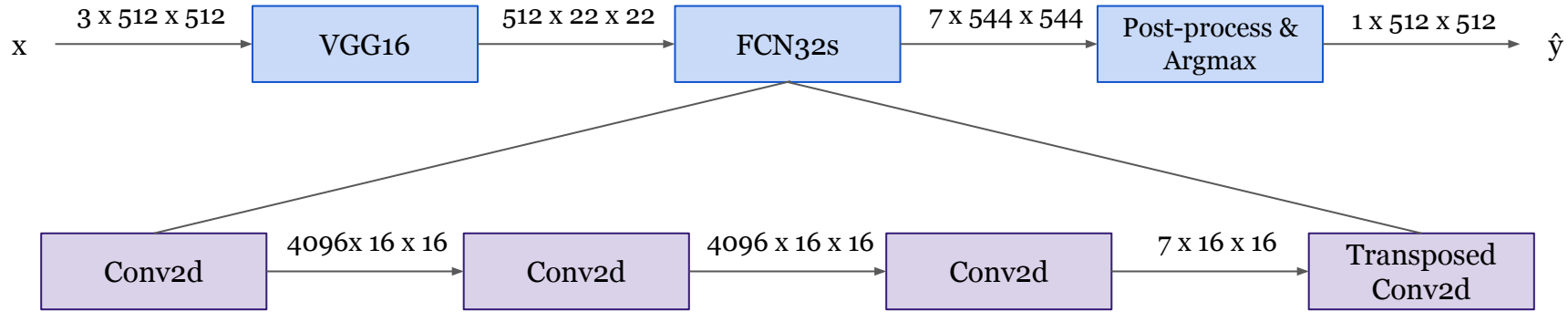
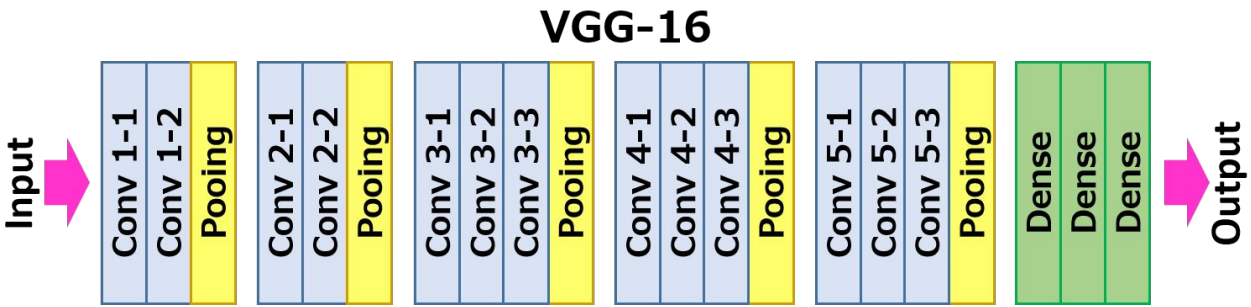


Image
DCNN
Atrous Conv
1x1 Conv
3x3 Conv rate 6
3x3 Conv rate 12
3x3 Conv rate 18
Image Pooling
1x1 Conv
1x1 Conv
Prediction
DeepLabv3

HW1-3


x
3 x 512 x 512
VGG16
512 x 22 x 22
FCN32s
7 x 544 x 544
Post-process & Argmax
1 x 512 x 512
ŷ
Conv2d
4096x 16 x 16
Conv2d
4096 x 16 x 16
Conv2d
7 x 16 x 16
Transposed Conv2d


VGG-16
Input
Conv 1-1
Conv 1-2
Pooing
Conv 2-1
Conv 2-2
Pooing
Conv 3-1
Conv 3-2
Conv 3-3
Pooing
Conv 4-1
Conv 4-2
Conv 4-3
Pooing
Conv 5-1
Conv 5-2
Conv 5-3
Pooing
Dense
Dense
Dense
Output