

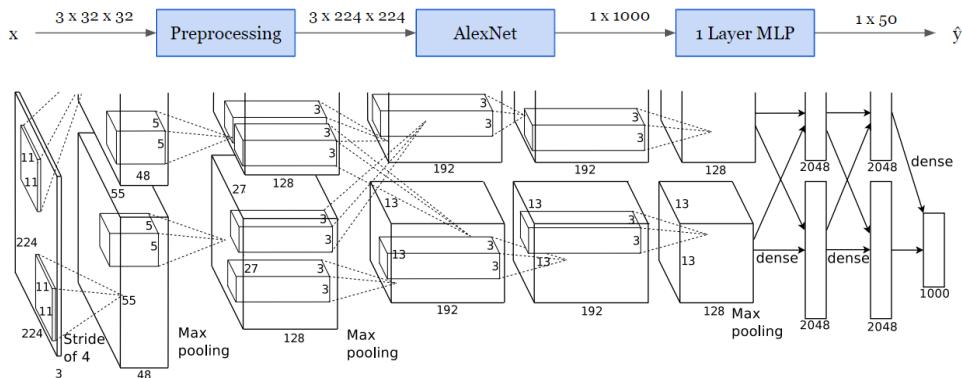
# DLCV HW1 Report

R12922051 資工碩一 陳韋傑

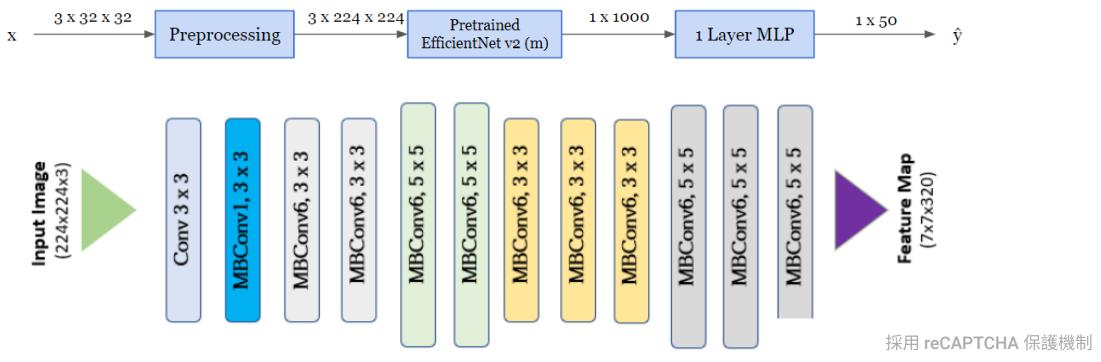
## Problem 1

## 1. Network Architecture

## A: AlexNet + 1 Linear Layer



## B: EfficientNet v2 (m) Pretrained + 1 Linear Layer



## 2. Accuracy

Model	A	B
Validation Accuracy	0.646	0.8944

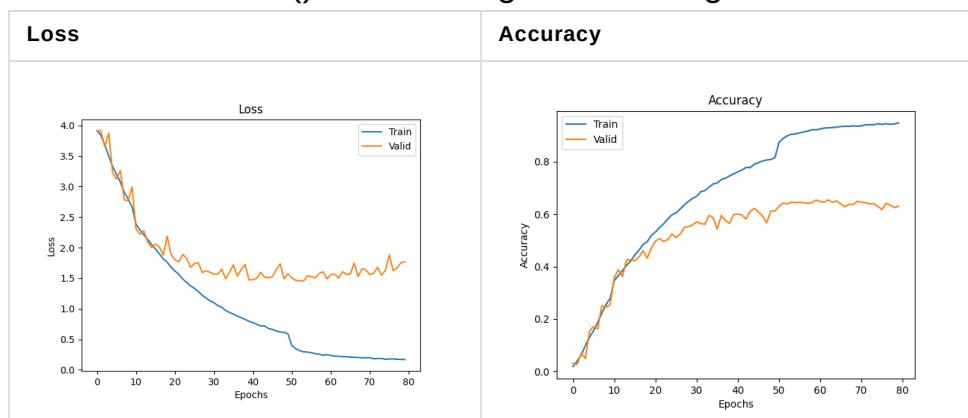
### **3. Implementation of model A**

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

## 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



## 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.

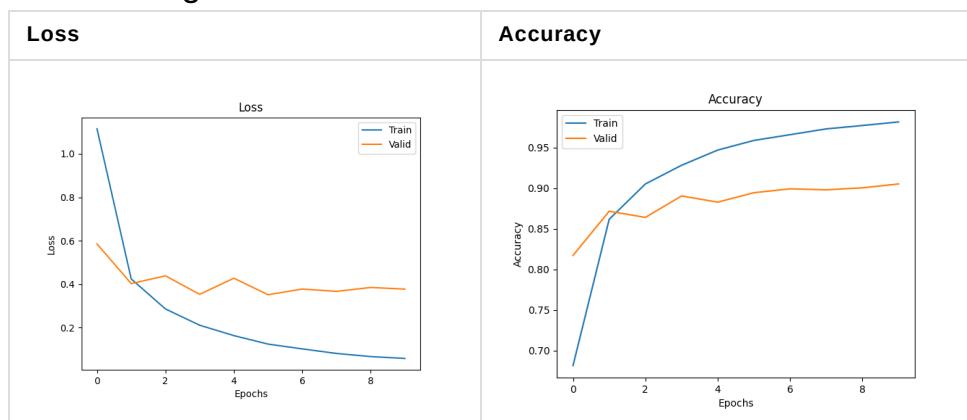
### MODEL COMPARISON

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.

### 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

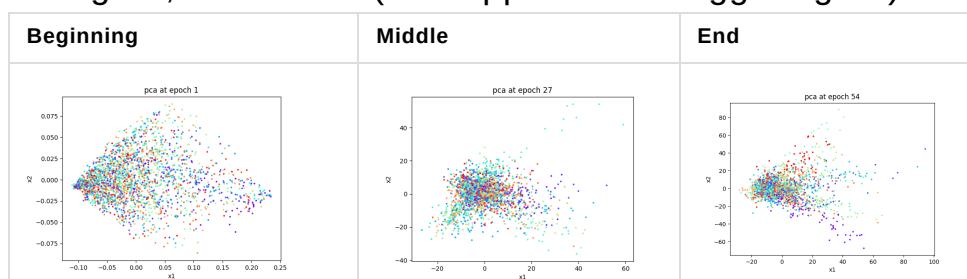


## 5. Visualization (Model A, PCA)

The following paragraphs are discussing PCA results at the end of training 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.

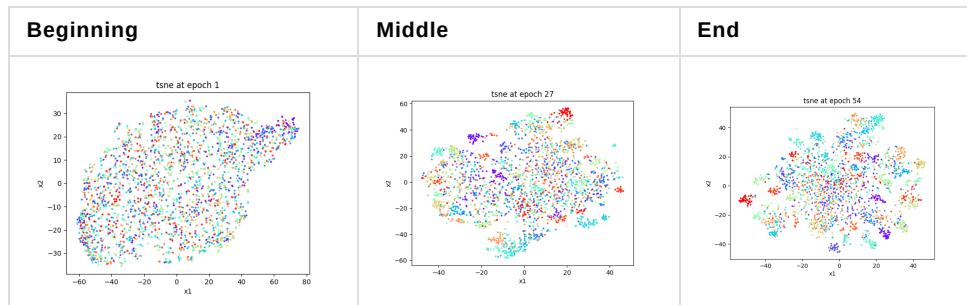
For example, the purple cluster at the bottom of the figure, or the red / light green cluster around the upper-left area of the figure, and so on. (See appendix for bigger figure)



## 6. Visualization (Model A, t-SNE)

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

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



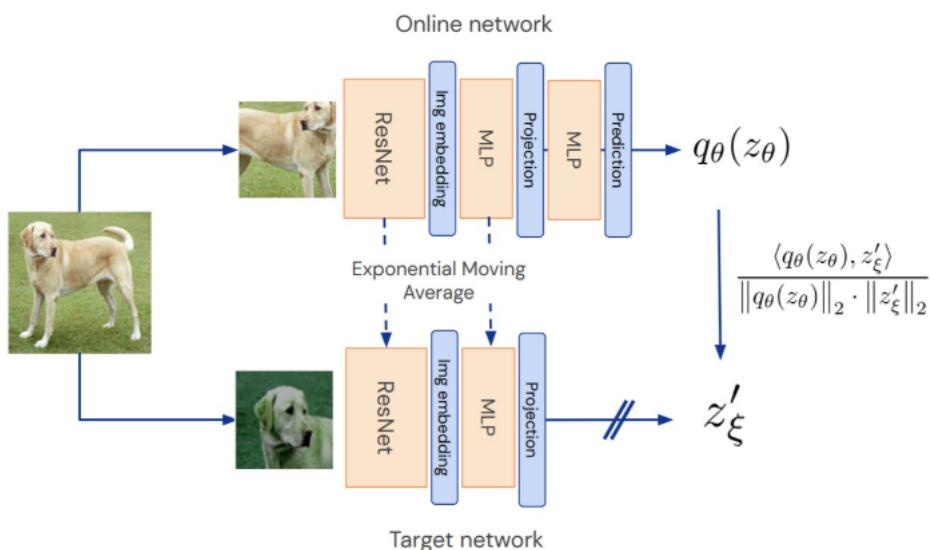
(Full size of the figures from p1-5, p1-6 are appended at the end of the report)

## Problem 2

### 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.



## 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

## 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 augmentation are also fixed throughout all the settings for comparison and convenience
- 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

## 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

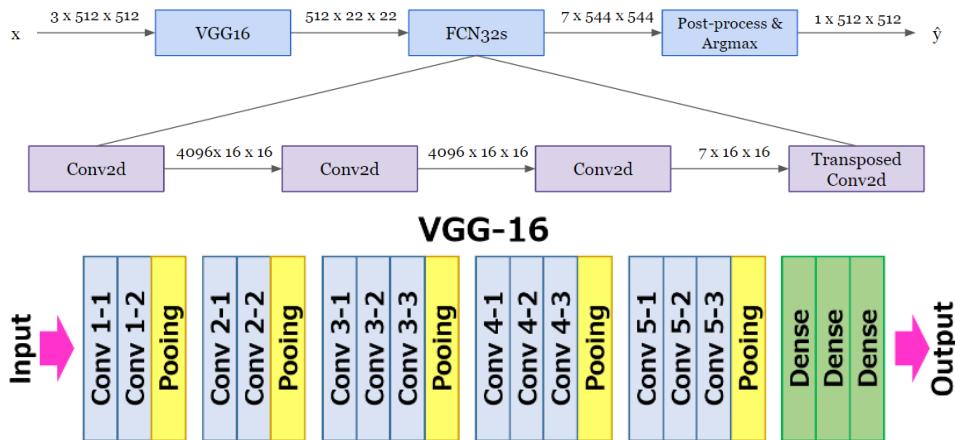
## DISCUSSION

- 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

# Problem 3

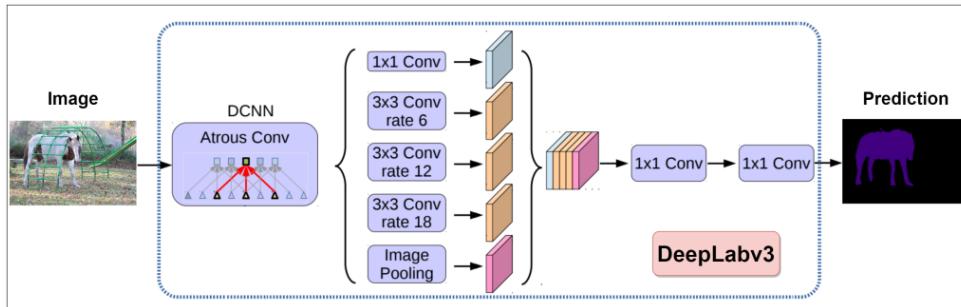
## 1. Network Architecture of Model A

A: Pretrained VGG 16 + FCN 32s



## 2. Network Architecture of Model B

B: Pretrained DeepLabv3 (with ResNet 101 as backbone)



## 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.

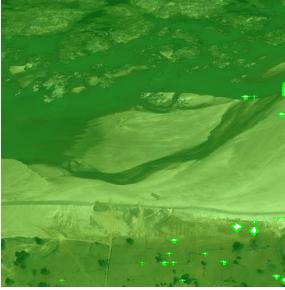
DeepLabv3 is a dedicated semantic segmentation model that employs atrous convolutions, a more complex architecture with multi-scale features, and an advanced decoder. It integrates contextual information effectively, using dilated convolutions to capture a broader context while preserving fine details.

## 3. mIoUs

Model	A	B
Validation mIoU	0.6981	0.7397

## 4. Predicted Segmentation

AT THE BEGINNING (EPOCH = 1, MIoU = 0.01)

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

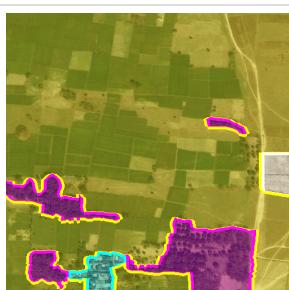
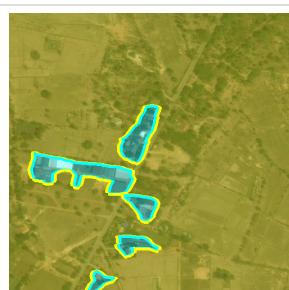
IN THE MIDDLE (EPOCH = 9, MIoU = 0.68)

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

AT THE END (EPOCH = 18, MIoU = 0.74)

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

GROUND TRUTH

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

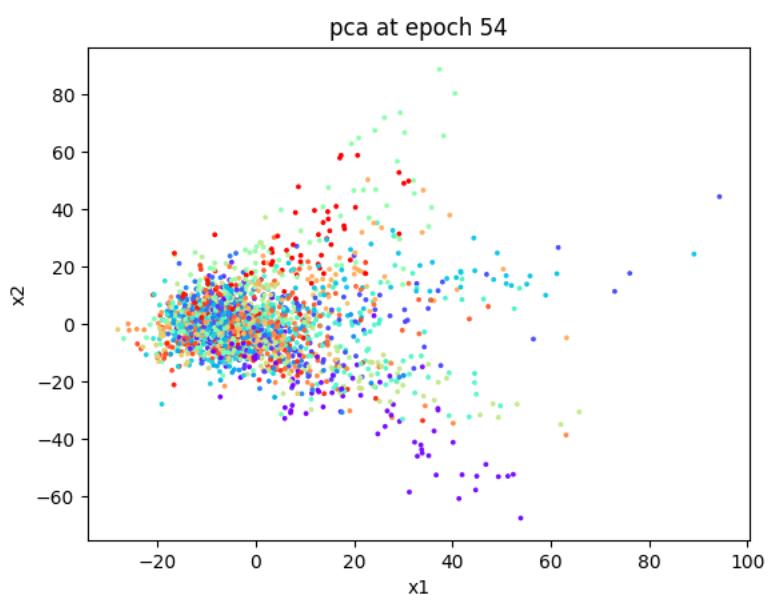
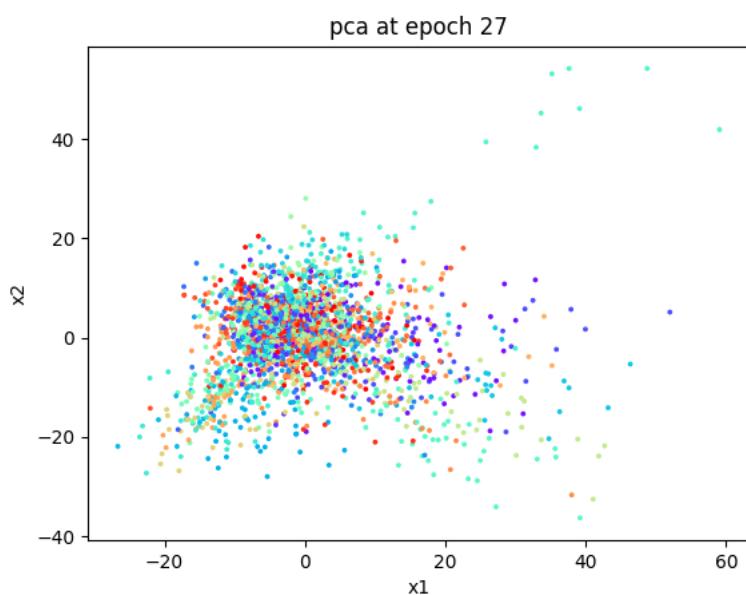
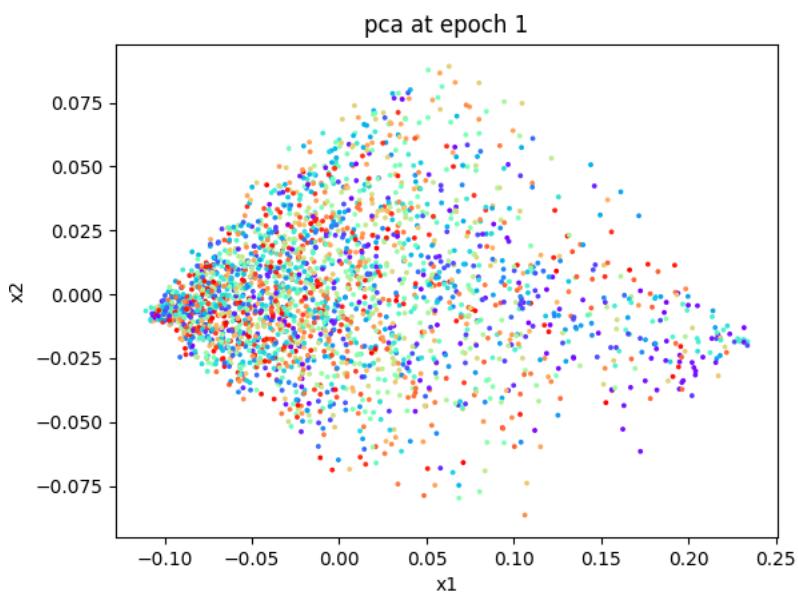
## References

- Problem 2
  - **byol-pytorch** (<https://github.com/lucidrains/byol-pytorch>)

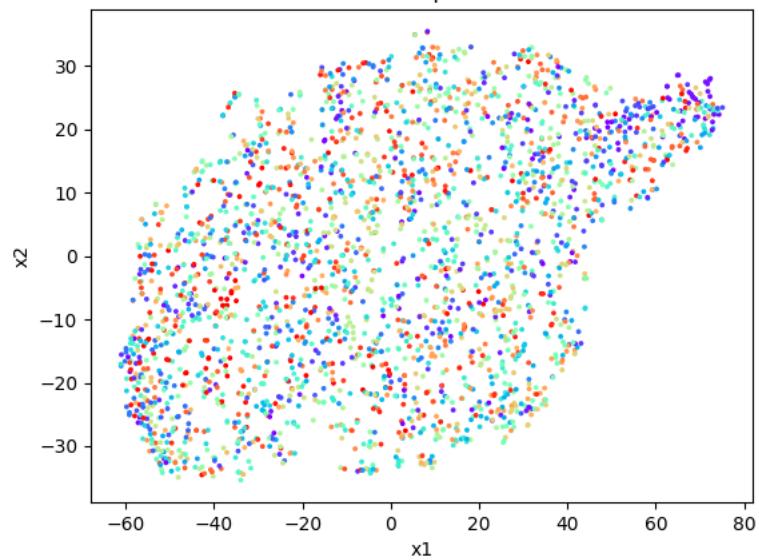
- Problem 3
  - **fcn32-pytorch** (
  - **pytorch-fcn** (
  - **pytorch vision deeplabv3 resnet101** (
- I also use ChatGPT to help with my report with some modifications made by myself

## Appendix: Figures from P1-5, P1-6

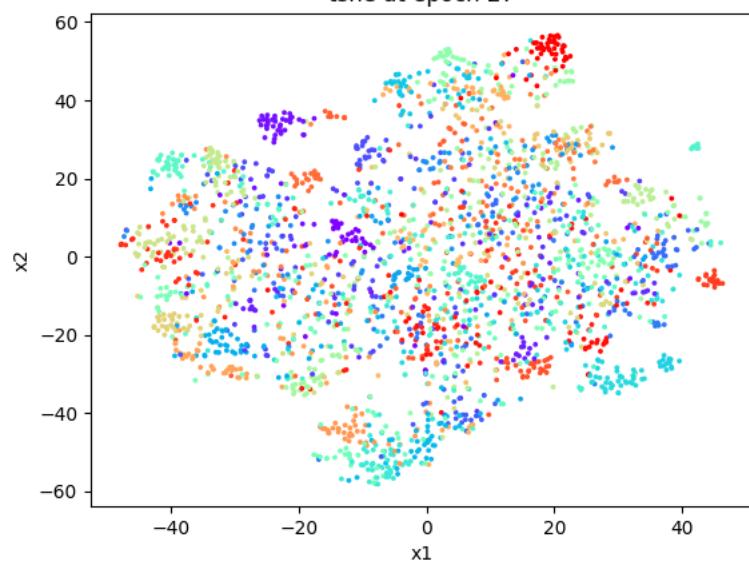
---



tsne at epoch 1



tsne at epoch 27



tsne at epoch 54

