Computer Vision Final Project

Litian Liang
School of Data Science
Fudan University
19307130109@fudan.edu.cn

Haoxian Chen
School of Data Science
Fudan University
18307110276@fudan.edu.cn

Hoiwa Lam
School of Data Science
Fudan University
19307130375@fudan.edu.cn

ABSTRACT

There are 3 important tasks in computer vision: Image classification is to structure image information into category information; Object detection is based on image classification, not only needs to know the category of the object, but also needs to know the location of the object; Semantic segmentation is to perform pixel-level object detection, needs to identify whether the segmented object is part of the target class. In this project, we first test the performance of two semantic segmentation model: SegFormer and DeepLab V3+ in different road scenes. Then we train Faster R-CNN with different initializations and backbones to find the optimal object detection model. We get the optimal model by ImageNet pretrained backbone. Finally, we design a transformer network model with the same amount of network parameters as the mid-term task for image classification and test its performance with different data augmentations. Our model gets an accuracy of 86.45% on CIFAR-100. The github repository of the project is https://github.com/chx7514/CV-Final.

Task 1

1 Introduction

Semantic segmentation is a fundamental task in computer vision. It is related to image classification since it produces per-pixel category prediction instead of image-level prediction. This task aims to test and compare several pre-trained semantic segmentation models. We will use two pre-training models to segment three videos in different situations and output semantically segmented video. Our video demo can be found on here.

2 Models

2.1 SegFormer (2022)

SegFormer[1] has two appealing features:

1) SegFormer comprises a novel hierarchically structured Transformer encoder which generate high-resolution coarse features and low-resolution fine features. It does not need positional encoding, thereby avoiding the interpolation of positional codes which leads to decreased performance when the testing resolution differs from training.

2) SegFormer avoids complex decoders. The proposed MLP decoder aggregates information from different layers, and thus combining both local attention and global attention to render powerful representations.

2.2 DeepLab V3+ (2020)

DeepLabV3+[2] extends DeepLabV3 by adding a simple yet effective decoder module to refine the segmentation results especially along object boundaries.

1) DeepLab V3+ can arbitrarily control the resolution of extracted encoder features by atrous convolution to trade-off precision and available computation resources.

2) DeepLab V3+ adapt the Xception model for the segmentation task and apply depthwise separable convolution to both Atrous Spatial Pyramid Pooling and decoder module, resulting in a faster and stronger encoder-decoder network.

2.3 Models Infos

Method	Encoder	Params(M)	Flops(G)	mIoU
SegFormer	MiT-B4	64.1	1240.6	83.8
DeepLabV3+	MobileNet V2	15.4	135	72.1

We tested the trained models with daytime, night and foggy road conditions videos respectively. The first two videos are 1280*720 and the last video is 1920*1080, the frame rate of all videos is 30FPS.

3 Test Models

We first extract each frame of the video, and process each frame by semantic segmentation, stack the processing results in order finally.

3.1 Daytime Street

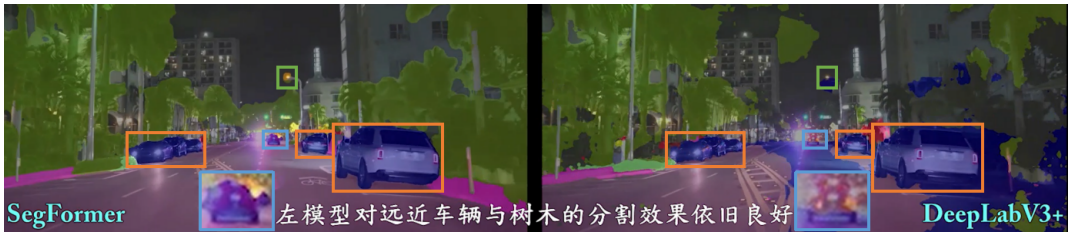
Both models are capable of segmenting larger vehicles, but the segmentation of sidewalks and pedestrians by the SegFormer model are more delicate than the DeepLabV3+ model.

When the front vehicle gradually becomes smaller, DeepLabV3+ model cannot be effectively segmented, while SegFormer can still segment the car behind crowds.

When the vehicle turns at the intersection, the SegFormer model can segment the sidewalk, street signs and pedestrians, while the DeepLabV3+ model has a poor segmentation effect on blurred images.

Even if the crowd is occluded by the car, the SegFormer model can accurately segment them, and the outline of each item is very clear, while the DeepLabV3+ model has a rougher boundary for some cars.

3.2 Night Street



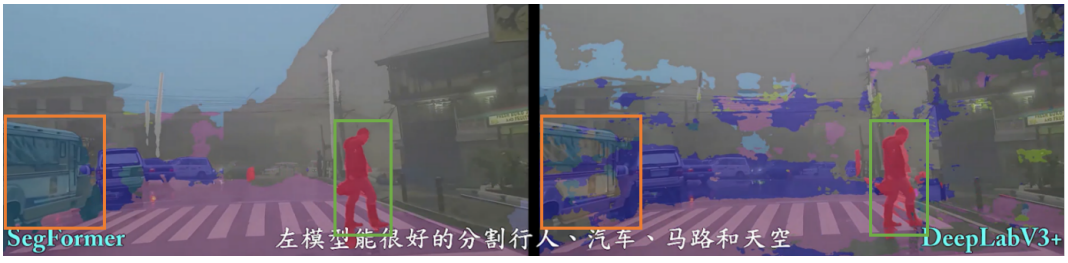
SegFormer has a good segmentation performance for all near and far vehicles, and it can still correctly and accurately segment sidewalk, every tree and even a street light at night, while DeepLabV3+ can only segment part of vehicles and trees roughly.

May be affected by the lights of the police car, the DeepLabV3+ model cannot divide crowds, cars and street sign.

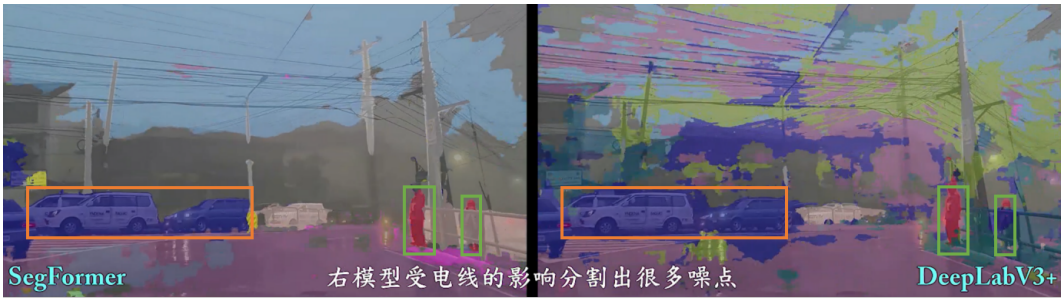
For a motorcycle in the dark, the SegFormer model segments it as a car, while the DeepLabV3+ model does not recognize it.

Both models are affected by the high beams and they are hard to recognize and segment the vehicles in the opposite lane. The SegFormer model even segments it into tree category.

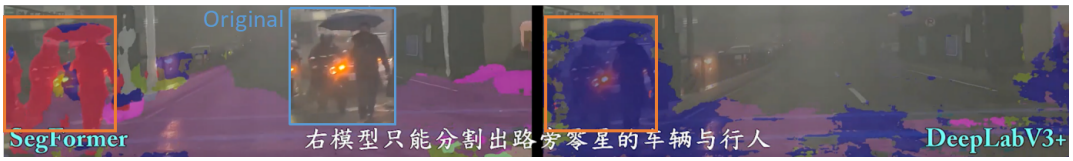
3.3 Foggy Street



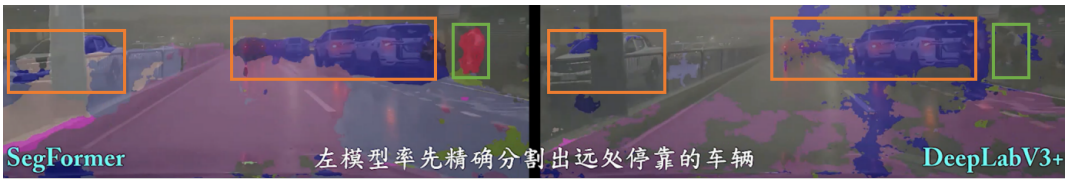
The SegFormer model can clearly segment the pedestrians, cars, utility poles and sky, while the DeepLabV3+ model has very rough segmentation of vehicles and pedestrians, indicating that the foggy day interferes with the DeepLabV3+ model very much.



The wires in a foggy day make the segmentation performance of the DeepLabV3+ model very poor, and there is a lot of noise, but it can still barely separate pedestrians and vehicles. SegFormer model can confidently segment the sky, pedestrians and vehicles.



The SegFormer model still separates the man in black with an umbrella and a motorcycle in the fog from the background, but the DeepLabV3+ model confuses people and cars.



Compared with the DeepLabV3+ model, SegFormer shows better recognition speed and segmentation ability for vehicles that are more hazy in the distance. For the pedestrian on the right sidewalk, it is difficult to distinguish even for the human eye, but SegFormer can still segment it delicately. But neither model can segment vehicles that are obscured by utility pole.

4 Conclusion

We detailed show the robustness compared with SegFormer and DeepLabV3+ by using three daily driving videos of daytime, night and foggy, analyzed the objects of different sizes, different occlusion or blurring conditions, and the fineness of object segmentation.

We found that:

1) Benefit from positional-encoding-free, hierarchical Transformer encoder and lightweight All MLP decoder approach, SegFormer model can adapt to videos of different clear conditions efficiently, and obtain a relatively delicate segmentation result.

2) For the DeepLabV3+ model ues MobileNet V2 as network backbone, which trades off accuracy, and number of operations measured by multiply-adds, as well as the number of parameters. Shows relatively good segmentation results within daytime and night videos, but degrades when the picture is complex or unclear.

Task 2

1 Introduction

Object detection is the task of detecting instances of objects of a certain class within image, as opposed to image classification, which assigns one label to the entire picture. As the name implies, recognizes the target items inside an image, labels them, and specifies their position. Essentially, object detection combines image classification and object localization.

In this task, we use ResNet-50 as the backbone network of the Faster R-CNN model, and use random initialize, ResNet-50 trained on ImageNet and Mask R-CNN trained on COCO to initialize the backbone network, and then we train them separately to see how different initialization affect network performance.

1.1 Dataset

We use two separate datasets(ImageNet and COCO) for pre-training the models.

1.1.1 ImageNet

The ImageNet is a large-scale image dataset that contains more than 14 million annotated images covering 1,000 labels, it used for classification, localization and detection task evaluation. There are more than 1 million images with tight bounding box annotations for each object instance in the image.

1.1.2 COCO

The COCO dataset is a large-scale object detection, segmentation and captioning dataset. This dataset mainly intercepted from complex daily scenes, and the target in the image is calibrated by precise segmentation. It provides 80 categories, more than 330,000 images, of which 200,000 are labeled, and the number of individuals in the entire dataset exceeds 1.5 million for semantic segmentation.

1.1.3 VOC

VOC dataset widely used as a benchmark for object detection, it contains 20 object categories including vehicles, household, animals and others. Each image's size are various and has pixel-level segmentation annotations, bounding box annotations, and object class annotations. The PASCAL VOC 2007 dataset is split into three subsets: 2,501 images for training, 2,510 images for validation and 4,952 for testing.

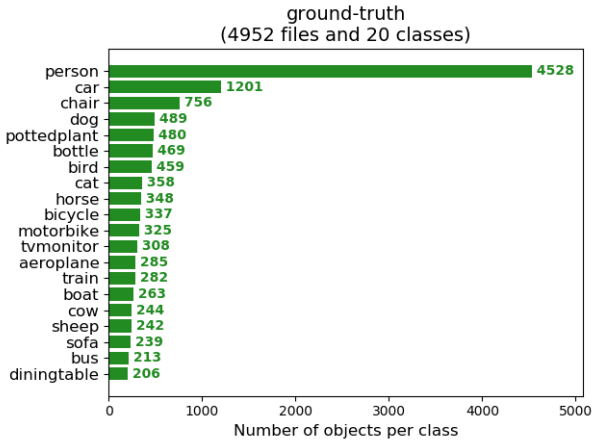


Figure 1: Ground Truth information of VOC test

2 Models

2.1 Faster R-CNN

Since we have showed the Faster R-CNN network in detail in our midterm report, we briefly summarize its process here.

1. Resize the input image proportionally.

2. Use the backbone of CNN to extract the image common feature layers.

3. Use the RPN network to obtain the offset of the ROI relative to the anchor and the probability of containing the object.

4. Since the generated proposals have different sizes, it is necessary to use the ROIPooling layer to convert the proposals in the feature map to the same size.

5. Bounding box regression and classification prediction of ROIPooling output with the convolution and maxpooling of the last layer of resnet.

2.2 Backbone: Feature Pyramid Network (FPN)

Multi-scale detection is becoming more and more important in object detection, especially the detection of small objects. FPN is a well-designed multi-scale detection method and it includes three parts:

1): **Bottom-up link**: The feature extraction process of traditional convolutional neural networks divides several stages according to the size of the feature map.

2): **Top-down link**: Using nearest-neighbor or deconvolution upsampling operations from the top layer to the bottom layer, using nearest-neighbor upsampling is both simple and reduces training parameters.

3): **Horizontal connection:** reduce the number of channels through 1x1 convolution for each layer linked from bottom to top, generate a feature layer with the same size as the upsampled feature map, and then fuse (add) them.

FPN is actually a general architecture that can be used in conjunction with various skeleton networks. The structure of ResNet+FPN used in Mask R-CNN is shown in figure 2.

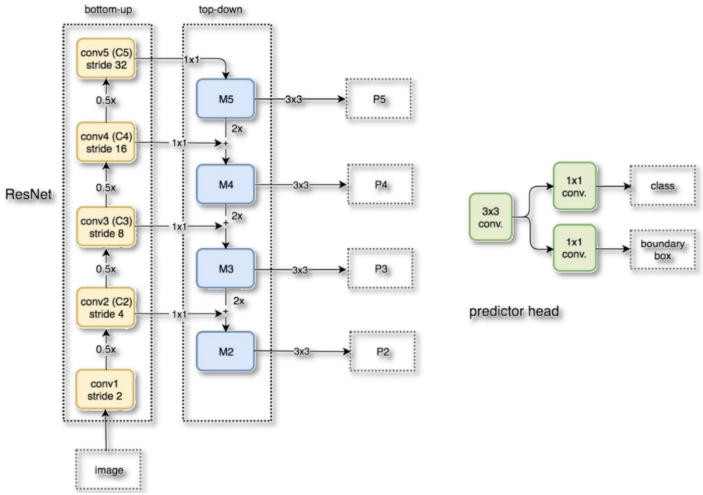


Figure 2: ResNet+FPN

3 Train

On the basis of the baseline, we apply data augmentations including flip, crop, mixup and so on to the training process. To compare the effect of different initialization models, we use ResNet-50 as the backbone in Faster-R-CNN. Using the same network, we do following 4 experiments in task 2.

1. Randomly initialize weights(Random normal weights for convolution layers, RPN and RoIHead). Train 80 thaw epochs on VOC.

2. Randomly initialize weights(Random normal weights for convolution layers, RPN and RoIHead). Train 80 freezing epochs(finetune) on VOC.

3. Initialize ImageNet pretrained ResNet-50 weights(Random normal weights for RPN and RoIHead). Train 80 freezing epochs(finetune) on VOC.

4. Initialize COCO pretrained ResNet-50 weights(Random normal weights for RPN and RoIHead). Train 80 freezing epochs(finetune) on VOC.

3.1 Loss Function

The regression loss and classification loss of the proposed frame network are calculated respectively

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

L_{cls} is Logarithmic Loss, L_{reg} is Smooth L1 Loss.

3.2 Learning Rate

We apply the cosine annealing $\eta_t = \frac{1}{2}\eta_{max}\left(1 + \cos\frac{T_t}{T_{max}}\pi\right)$ as the learning rate decay strategy.

3.3 Optimizer

We can use mini-batch gradient descent (SGD) with batch size 4, momentum 0.9 and weight decay 5e-4 while training or Adam optimizer. Our results were got by Adam optimizer.

3.4 Freezing Training(Finetune)

In the freezing stage, the backbone of the model is frozen, and the feature extraction network does not change. The occupied video memory is small, and only fine tune the network. In the thawing stage, the backbone of the model is not frozen, and the feature extraction network will change. The occupied memory is large, and all parameters of the network will change.

TRAIN IMG SIZE	FINETUNE BATCHSIZE | EPOCHS	IOU_THRESHOLD	BATCHSIZE | EPOCHS
[512, 512]	32 | 80	0.5	32 | 80
LEARNING RATE DECAY	MAX LEARNING RATE	MIN LEARNING RATE	BACKBONE
cosine	1e-4	1e-6	ResNet-50

Figure 3: Training Hyperparameters

4 Results

4.1 Loss Curve

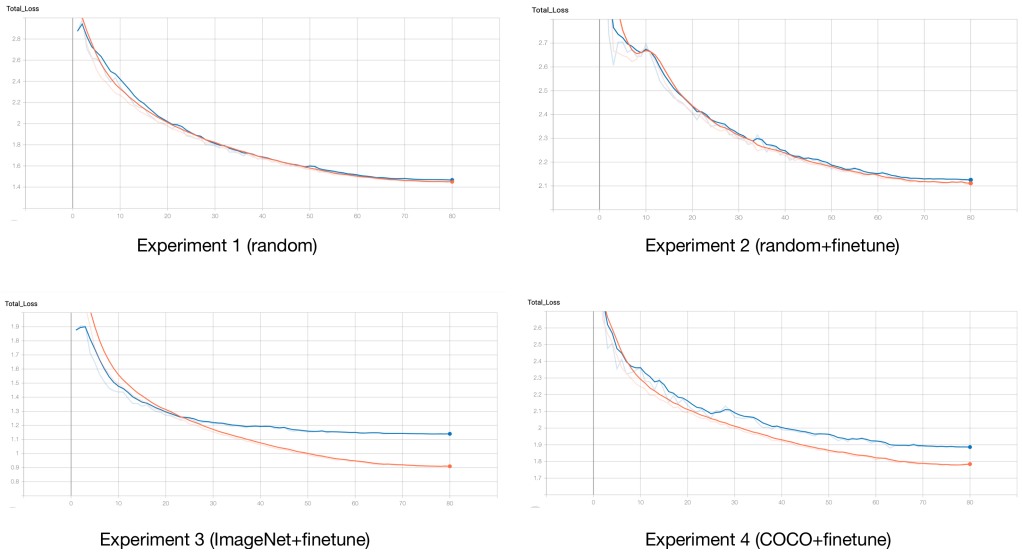


Figure 4: Training(red line) & Validation(blue line)

Figure 4 shows both total training loss and validation loss of all 4 experiments we did. It can be seen that initial losses are similar among three models except ImageNet initialized model. In epoch 80, the loss of ImageNet initialized model is the lowest and the loss of random initialized-finetune model is the highest.

It can be explained by two possible reasons. One is that ImageNet Dataset might be more similar to VOC Dataset or ImageNet contains more information about objects in VOC compared with COCO Dataset. The other one is that the anchors scales in our Faster-R-CNN might not be the same as anchors scales in Mask-R-CNN, or we can say that the Feature Pyramid Network has some difference. As the weights in the pretrained backbone might not be suitable for this task or this model, our finetune training might not get a good effect.

Another phenomenon is that in experiment 1 and 2, the training loss and validation loss are almost the same, but in experiment 3 and 4, validation loss is much higher than training loss. That is, the model is over-fitting on training set with two pretrained experiments. However, random initialized model is different. One possible reason is that it needs more epochs to train for not-pretrained model.

It is interesting that experiment 1 has a better performance than experiment 4. But it's not comparable because we train the full model in experiment 1 but just do finetune in experiment 4. So I have done an extra experiment 2 to confirm that COCO pretrained model is better than random initialized model. In this task, our target is to compare three situations so we have to do such work. If we want to train an optimal model, we should load the pretrained model, do some epochs finetune training and train the full model by some epochs.

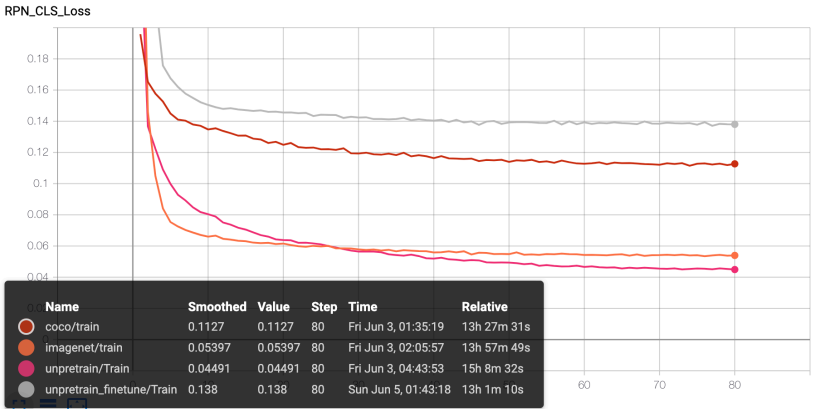


Figure 5: RPN CLS Loss

Figure 5 and Figure 6 shows classification loss and reggression loss in RPN. We can see that the unpretrain model after 30 epochs have even better performance than ImageNet pretrained model which has the best performance in total loss. COCO pretrained model (deep red line) has an awful performance. The results confirm the possible causes we just discussed.

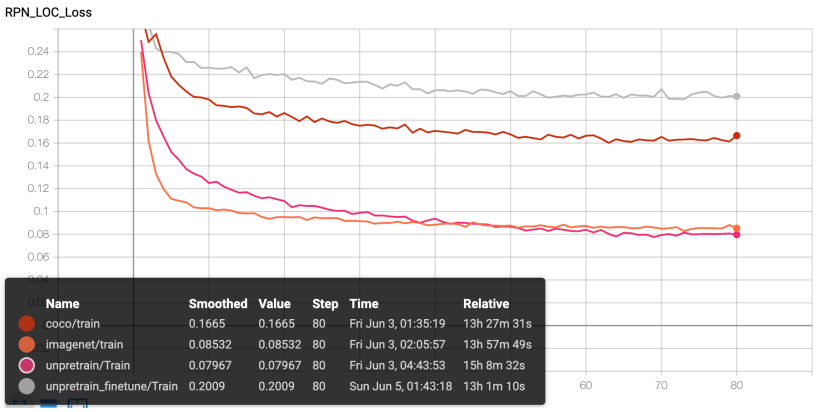


Figure 6: RPN LOC Loss

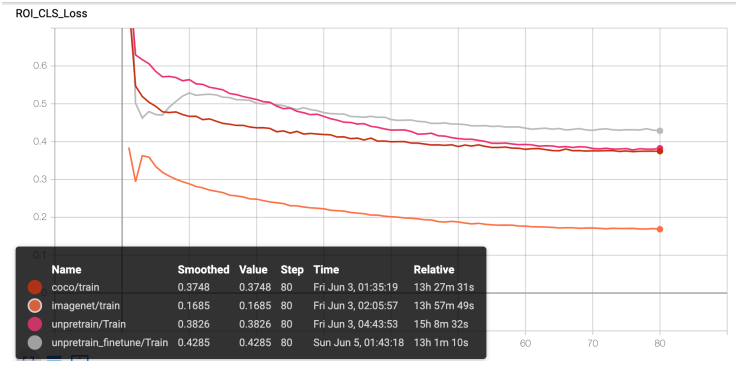


Figure 7: ROI CLS Loss

Figure 7 and Figure 8 shows classification loss and reggression loss in ROIHead. We can see that ImageNet pretrained model has absolutely best performance. Besides, the unpretrain model doesn't converge and it can be still improved.

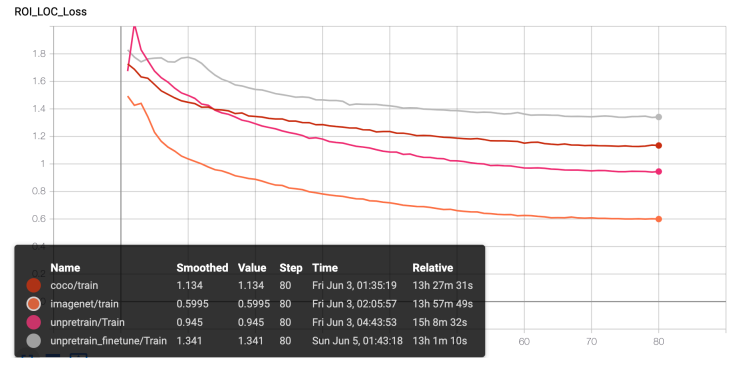


Figure 8: ROI LOC Loss

4.2 mAP Curve

Figure 9 shows mAP@0.5 curve of 4 experiments. VOC pretrained model shows its best performance as 0.687. However, in our midterm experiments, we just trained the model with pretrained model for 30 epochs(20 epochs for finetune and 10 epochs for full model) and got best mAP as 0.794. So we just compare the results of 4 experiments here. This result corresponds to the Total Loss result and we get the same conclusion.

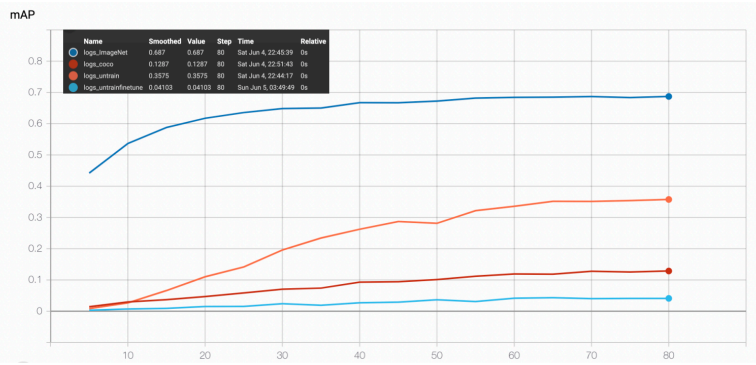
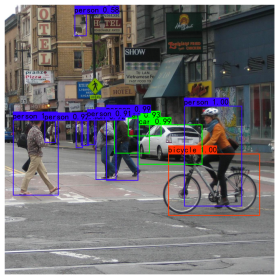
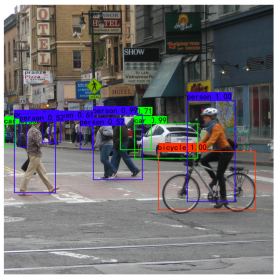


Figure 9: mAP Curve

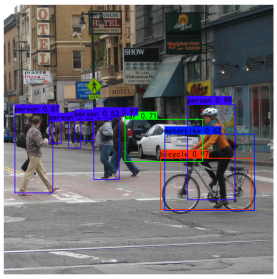
4.3 Object Detecting Examples



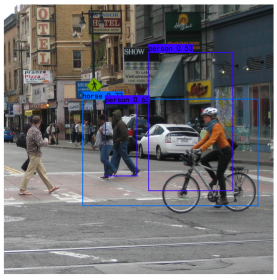
midterm model



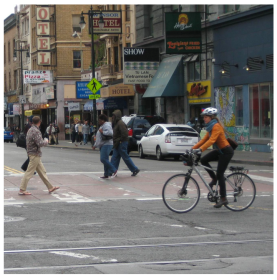
ImageNet



random



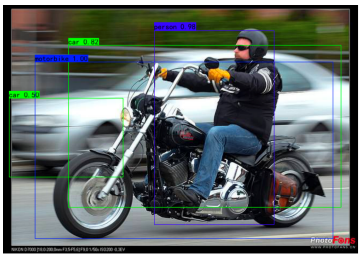
COCO



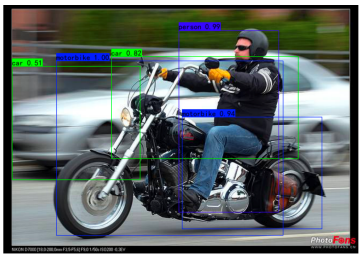
random+finetune

Figure 10: Object Detecting Results

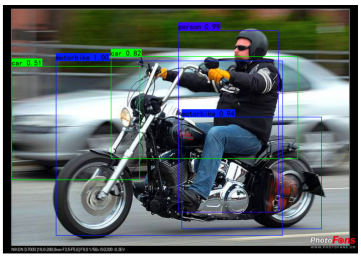
Figure 10 clearly shows different performances of 5 models. The model we trained in midterm task is the most sensitive and identified almost all people on the street. But it detect a wrong person at the up-left corner. The model trained from ImageNet backbone also shows a good performance. It has detected the half-part car on the left edge of the picture and many people. However, two men who have a little overlap were recognized as one person. So it is not so accurate. The remaining three models did not work well.



midterm model



ImageNet



random

COCO

random+finetune

Figure 11: Object Detecting Results

Figure 11 is a difficult task for object detecting task. In the original figure, there is a fuzzy car in the

background. The main part of the car is blocked by the motorcycle. Thus, first three networks divided the car into two parts. COCO pretrained model just recognized the left part of the car. The performance of the last model is as bad as ever. The reason why it tends to detect some persons who actually don't exist might be in Figure 1. In VOC, person is the class that has most objects.

Figure 12: Object Detecting Results

Figure 12 also shows that ImageNet pretrained model comes closest to our midterm model.

5 Conclusions

Mask R-CNN adds a branch of prediction segmentation Mask on the basis of Faster R-CNN, and replaces ROI Pooling layer with ROIAlign layer. In this task, we have done 4 experiments to test the performance of different backbone in Faster-R-CNN. The results show that if we just do finetune training on this model, the backbone ResNet-50 initialized by ImageNet has the best performance.

Task 3

1 Introduction

Image classification intents to judge whether an image contains a certain object belongs to several classes. In general, it falls under the domain of supervised machine learning, meaning that solutions involve splitting a labeled dataset into training and test sets, and evaluating the test performance of a model fit to the training data. Image classification is preliminary to more complicated image recognition tasks, including object detection, automatic captioning, robotic decision-making, etc.

1.1 Convolutional Neural Networks (CNNs)

Since CNN has inductive biases property, it is helpful for the learning and extraction of image features. Specifically, 1. Using the weight sharing mechanism, the features extracted by the convolution layer have translation equivariance, so that it does not pay attention to the global position of the feature, but only cares about the existence of some decisive features. 2. The feature map of convolution has local sensitivity, that is, CNN is good at extracting local effective information, but it is difficult to extract long-distance features between the overall data.

For example, when we use CNN to train a face recognition model, CNN can effectively extract features such as eyes, nose, mouth, etc., but it cannot connect them to form such long distance features as "eyes are above nose" and "mouth is below eyes".

To extract and track long correlated features in these raw data, the model needs to expand the receptive field using deeper convolutions. But this will greatly increase the complexity of the model, and may cause the problem of gradient vanish, resulting in the network is hard to train and converge. To solve this problem, Residual connections and Dilated Convolutions can increase the gradient propagation depth to a certain extent and expand the receptive field of the model.

1.2 The Transformer

Transformer architecture was first applied to the problem of machine translation. With the success in NLP, the emergence of vision transformer by Dosovitskiy et al.[3] demonstrated superior performances on many vision task, where the key idea is to split the image into patches so that it can be linearly embedded with positional embedding. Transformer-based architectures have achieved remarkable success most recently.

The Transformer leverages attention in groundbreaking ways to eliminate the recurrent hidden state. In particular, it introduces self-attention as a special case of an attention mechanism: where the query, key, and value representations are all calculated from the same sequence. It enables the model to capture both short and long range visual dependencies. In addition attention is not calculated directly on the the input, but on linear embeddings calculated with weight matrices for the queries, keys, and values.

1.3 CIFAR-100

We use CIFAR-100 as our dataset. CIFAR-100 is a large data set for identifying common objects. There are 100 classes in the CIFAR-100 which are grouped into 20 superclasses. Each class contain 500 training images and 100 testing images. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).

2 Vision Transformer (ViT-Base)

2.1 Model Overview

ViT is a convolution free transformer based architecture which uses the encoder block of the original transformer proposed by Vaswani et al. and contains 12 such encoder blocks. Each of of these encoders contains a Multi-Head Self-Attention block(MHSA) and a Multi-Layer Perceptron(MLP). The main architecture of ViT is shown in Figure 13.

ViT start by dividing the input image into n non-overlapping and fixed-size patches. These patches are flattened and linearly projected through fully connected layers to obtain the patch embeddings. To include

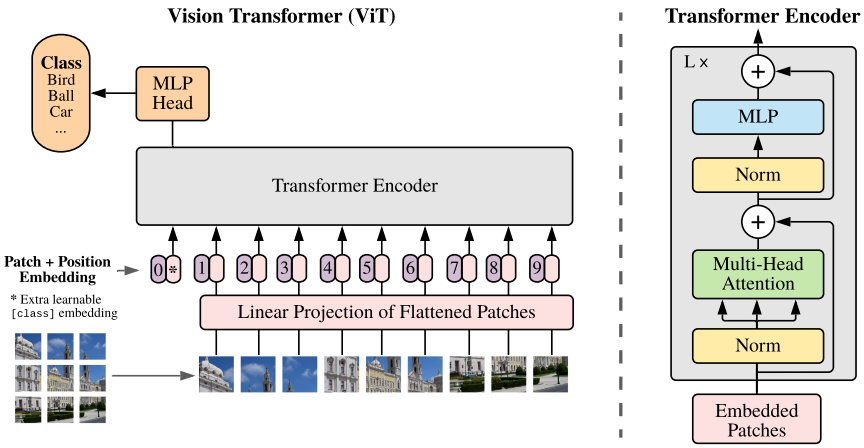


Figure 13: ViT along with the transformer encoder block

structural information, patch embeddings are combined with learnable positional embeddings and a class CLS token. These combined embeddings are supplied as input to the MHSA block of the transformer encoder.

The MHSA block is responsible for computing self attention i.e., to encode the interaction among these n patches in terms of global contextual information (Khan et al.). It uses 3 learnable weight matrices $W^Q \in \mathbb{R}^{d\times d_q}$, $W^K \in \mathbb{R}^{d\times d_k}$ and $W^V \in \mathbb{R}^{d\times d_v}$ which are multiplied with the combined embedding $X \in \mathbb{R}^{n\times d_k}$ to output the matrices: Query $Q = XW^Q$, Key $K = XW^K$ and Value $V = XW^V$. The output $Z \in \mathbb{R}^{d\times d_v}$ is computed as follows:

$$Z = \text{softmax}\left(\frac{QK^T}{\sqrt{d_q}}\right)V$$

Here d_q is the dimension of the query matrix Q and $d_q = d_k$. In the ViT-Base model, the MHSA block comprises of 12 heads, each one having their own learnable weight matrices (W^Q, W^V and W^K) to produce the output Z.

For a single MHSA block, these heads are computed all together in parallel. Outputs from all these heads are concatenated together and passed to the necessary Norm, Dropout and Residual modules for better convergence speed and performance.. Further, the output from one transformer encoder is passed to the next. Eventually, output of the last transformer encoder is supplied as input to the last MLP Head which is a feed forward netowrk. The MLP head provides the final outputs as class predictions.

2.2 Problems in ViT

ViT demonstrated that pure transformer can also achieve superior performances. Compared with CNNs, the performance is better when the amount of data is larger, but ViT also has the following problems:

• The inductive bias feature of CNN runs through the entire model, but only the MLP layer in ViT has this feature, and the self-attention layer is global, so the training results of ViT on small and medium-sized datasets are not as good as CNN, requires pre-training on large datasets.

• ViT can't be directly applied to images with different sizes, because the patch size is fixed, when the picture size changes, the sequence length changes and the position encoding cannot be used directly. The practice of interpolation generally causes performance loss, which needs to be solved by the Finetune model.

• The neighborhood structure is only used when splitting the image into patches and fine-tuning with different resolution images to adjust the positional encoding. The location encoding doesn't carry any location information about the patches and spatial relationship between the patches must be learned from scratch.

3 Lite Vision Transformer (LVT)

LVT mainly modifies ViT's attention module and utilizes both **Convolutional Self-Attention** and **Recursive Atrous Self-Attention** these two novel self-attention to pursue both performance and compactness. LVT follows a standard four-stage structure but has similar parameter size such as MobileNetV2.

3.1 Convolutional Self-Attention (CSA)

Convolution layer is good at processing low-level features, prior arts have been proposed to combine convolution and self-attention with the global receptive field. Instead, CSA combines 3×3 kernels and local self-attention as a powerful layer in the first stage of the model.

3.1.1 Convolution

Convolution is computed by sliding windows. In each window, we can use $x, y \in \mathbb{R}^d$ be input and output feature vectors to express convolution as:

$$y_i = \sum_{j \in N(i)} W_{i \to j} x_j$$

where $N(i)$ represents the spatial locations in a local neighborhood that centered at location i. $W_{i \to j} \in \mathbb{R}^{d \times d}$ is the projection matrix relative spatial relationship from i to j.

3.1.2 Self-Attention (SA)

Self-Attention needs three projection matrices W_q, W_k, W_v to compute query, key and value. Consider sliding window based self-attention. In each window, we can express self-attention as

$$y_i = \sum_{j \in N(i)} \alpha_{i \to j} W_v x_j, \quad \alpha_{i \to j} = \frac{e^{(W_q x_i)^T W_k x_j}}{\sum_{z \in N(i)} e^{(W_q x_i)^T W_k x_z}} \in (0, 1)$$

where $\alpha_{i \to j}$ controls the contribution of the value in each spatial location in the summation. α is normalized by softmax operation such that $\sum_j \alpha_{i \to j} = 1$.

3.1.3 CSA

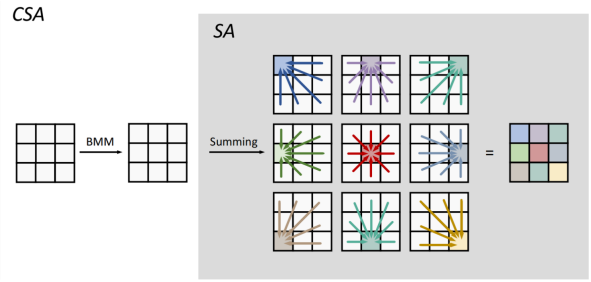


Figure 14: CSA in a 3 × 3 local window with batched matrix multiplication and summation

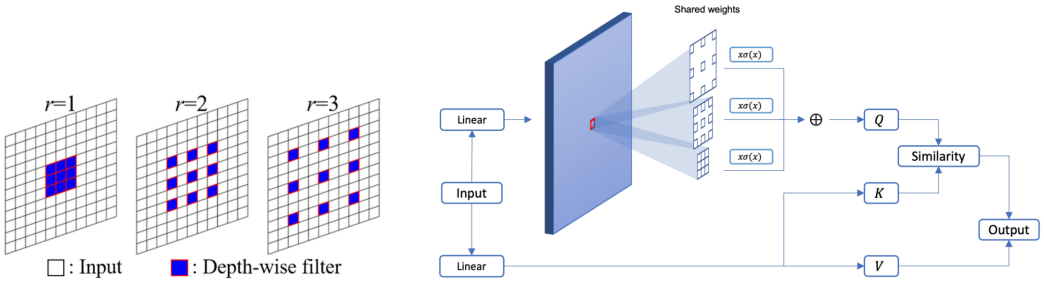
Generalize self-attention and convolution into a unified convolutional self-attention as shown in Figure 14, we can express self-attention as

$$y_i = \sum_{j \in N(i)} \alpha_{i \to j} W_{i \to j} x_j$$

We can know that: when $\alpha_{i \to j} = 1$ where all the weights are the same, CSA is the convolution for the output center. When $W_{i \to j} = W_v$ where all the projection matrices are the same, CSA is self-attention. By this generalization, CSA has both input-dependent kernel and learnable filter. It is designed for stronger representation capability in first stage of vision transformers.

3.2 Recursive Atrous Self-Attention (RASA)

Light-weight models are relative efficient and suitable for on-device applications limited by the small number of parameters. For high-level features, we focus on enhancing their representation capabilities with marginal increase in the number of parameters.



(a) Atrous Convolution (b) Illustration of ASA

3.2.1 Atrous Self-Attention (ASA)

Atrous convolution is proposed to capture the multi-scale context with the same amount of parameters as standard convolution. Weights sharing atrous convolution is also demonstrated in boosting model performances.

Unlike standard convolution, the feature response of self-attention is a weighted sum of the projected input vectors from all spatial locations. These weights are determined by the similarities between the queries and keys, and represent the strength of the relationship among any pair of feature vectors. Thus we add multi-scale information when generating these weights shown in Figure.

$$Q = \sum_{r\in\{1,3,5\}} \text{SiLU}\left(\text{Conv}\left(\hat{Q}, W_q^{k=3}, r, g=d\right)\right) \text{ where } \quad \hat{Q} = \text{Conv}\left(X, W_q^{k=1}, r=1, g=1\right)$$

$X, Q \in \mathbb{R}^{c\times H\times W}$ are the feature maps, k, r and g represent the kernel size, dilation rate, and group number of the convolution. ASA calculates the multi-scale query by three depth-wise convolutions after the 1×1 linear projection. These convolutions share kernel weights but have different dilation rates: 1, 3, 5. The parameter cost is further reduced by setting the group number equal to the feature channel number.

The parallel features of different scales are added with the weights calculated by sigmoid function for the purpose of self-calibration. This can be implemented by the $\text{SiLU}(x) = x \odot \text{sigmoid}(x)$. By this design, the similarity calculation of the query and key between any pair of spatial locations in self-attention uses the multi-scale information.

3.2.2 RASA

Use ASA as the non-linear activation function, RASA is a recursive method for self-attention and the design follows the pipeline of standard RNNs.

$$h_{t-1} = X_{t-1}, \quad X_t = \text{ASA}\left(\text{F}\left(X_{t-1}, h_{t-2}\right)\right)$$

The initial hidden state $h_{-1} = 0$. $\text{F}(X, h) = W_F X + U_F h$ is the linear function combining the input and hidden state. W_F, U_F are projection weights. We set the recursion depth as two in order to limit the computation cost.

3.3 Model Architecture

The architecture of LVT is shown in Tab 1. It adopt the standard four-stage structure. All stages comprise the transformer blocks. Each block contains the self-attention layer followed by an MLP layer. CSA is embedded in the stage-1 while RASA in the other stages. They are enhanced self-attention layers proposed to process local and global features in LVT.

4 Combining CNN and Transformer

In this task, we are required to compare CNN and Transformer with same parameters and FLOPS. Unfortunately, it's hard to find two network with same parameters and FLOPS, due to their special structures. In order to solve this, we introduce a network that can adjust this almost freely by combining current networks, which is inspired by CrossVit[4].

Architecture	Stage1	Stage2	Stage3	Stage4
SA Type	CSA	RASA	RASA	RASA
SA Kernel	3×3	Global	Global	Global
Layer Number	2	2	2	2
Downsample Stride	4	8	16	32
Feature Dimension	64	64	160	256
Heads Number	2	2	5	8
MLP Ratio	4	8	4	4

Table 1: Architecture of LVT

4.1 CrossViT

CrossVit is a method to learn multi-scale features for image classification. We know that in ViT, the granularity of the patch size affect the performance and the complexity. Smaller the patch size, ViT can perform better together with higher FLOPs and memory consumption. CrossViT introduce a dual-branch ViT where each branch operates at a different scale (or patch size in the patch embedding) and propose a simple yet effective module to fuse information between the branches.

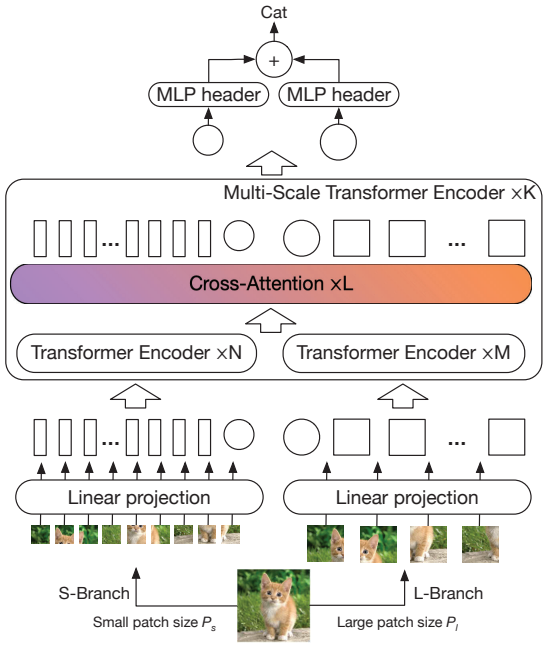


Figure 15: CrossViT

We focus on the two branches and header of CrossViT in our project. The two branches is of two different scales, one is large and the other is small. Then the two branches is sent into the transformer. At the header of the network, the two output of the branches go through an Mlp respectively and combine by taking their mean. This architecture shows us a way to fusion two transformers of different size.

We make a small modification on CrossViT. We replace the header of it, which is simply taking the mean. Instead, we first cat the output of the two branches, and then send it to a single Mlp. It's easy to see that the original header is a special case of our propose. We will discuss this later in the experiment.

4.2 CNN-Cross-LVT

However, in our project, due to the restricted resources, it's hard to train a transformer from scratch (we will show this later in the experiment). This means we can not combine two transformer, for example, LVT, of different patch size together. We think that the combination of multi-scaled CNN and Transformer is possible,

since multi-scale CNN and Transformer have been proved effective.

Combining different scaled CNN and Transformer has two advantages in our task. Since CNN and Transformer have their own advantages and disadvantages, the combination may get a better performance. The feature extracted by CNN may modify the Transformer. The other advantage is that the training of the novel network is easier than training a Transformer, since the training of CNN is fast and easy and we can use the pretrained Transformer.

But we should remark that this is a simple and shallow fusion of CNN and Transformer, and deeper and better fusion of the two kind of networks should be consider for better performance. The architecture of our model is shown in Figure16.

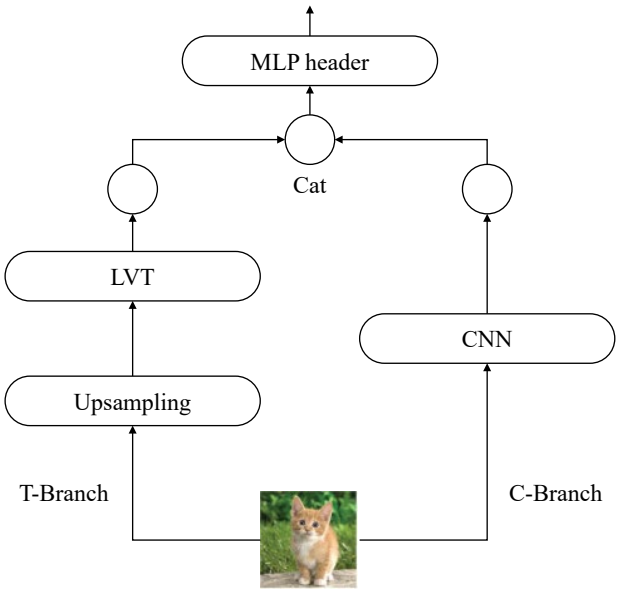


Figure 16: Our Proposed Model: CNN-Cross-LVT

5 Data Augmentation

Data augmentation is a implicit regularization technique that increases the amount and diversity of data. It improves the generalization performance and helps learning invariance at almost no cost, because the same example can be transformed in different ways over epochs. However, operating on one image at a time limited chances of exploring beyond the image manifold.

Mixup operates on two or more examples at a time, interpolating between them in the input space or feature space, while also interpolating between target labels for image classification. This flattens class representations, reduces overly confident incorrect predictions, and smooths decision boundaries far away from training data.

5.1 Baseline

Baseline actually is a lightweight strategy, by **Transforming** the original data, the richness of the data is enhanced, thereby making the model more generalizable.

Random Crop	-	Pad 4 pixels around the image and crop it randomly with size 32×32
Random Flip	-	Flip horizontally and vertically randomly with a probability $p = 50\%$
Normalization	-	Normalize each channel with mean=(0.491, 0.482, 0.447), std=(0.247, 0.243, 0.261)

5.2 MixUp

Zhang et al.[5] assigned a weighted linear interpolation of random image pairs from the training data and enhance the robustness to adversarial samples. Given two images and their ground truth labels$(x_i, y_i), (x_j, y_j)$, a synthetic training example :

$$(\hat{x}, \hat{y}) = \lambda(x_i, y_i) + (1 - \lambda)(x_j, y_j)$$

Where $\lambda \sim \text{Beta}(\alpha, \alpha) \in [0,\ 1]$, and $\alpha \in [0,\ +\infty)$ controls the intensity of interpolation. but these type mixup images are overlays and tend to be unnatural.

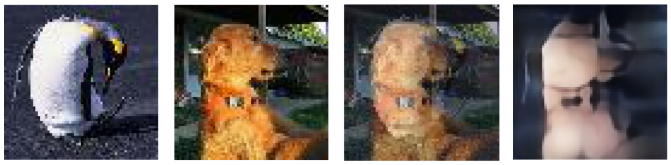


Figure 17: Mixup & Manifold Mixup

5.3 Manifold Mixup (Best Performance In PJ2)

Instead linear interpolate the input data, Verma et al.[6] combine the data in the hidden layers. Since the convolution layers act as a feature extractor, the data through the hidden layer can be seen as the feature of the original data, and therefore the linear interpolation of them can combine their features better. Manifold Mixup can learn robust features and the representations learned by it are more discriminative and compact.

5.3.1 CutMix

CutMix replace the masked regions in CutOut with a patch from another image, and mix the ground truth label base on the proportion of augmented images. Since CutOut of the image force the model to learn to make predictions based on the robust of features, the added patches further enhance localization ability by requiring the model to identify the object from partial view, so this technique can work better than simple MixUp and CutOut usually.

5.4 MixToken

MixToken is a special data augmentation for Transformers. It can seen as a variant of CutMix. Vision transformers rely on patch-based tokenization to map each input image to a sequence of tokens. If we apply CutMix directly on the raw image, some of the resulting patches may contain content from two images, leading to mixed regions within a small patch. It's hard to assign each output token a clean and correct label.

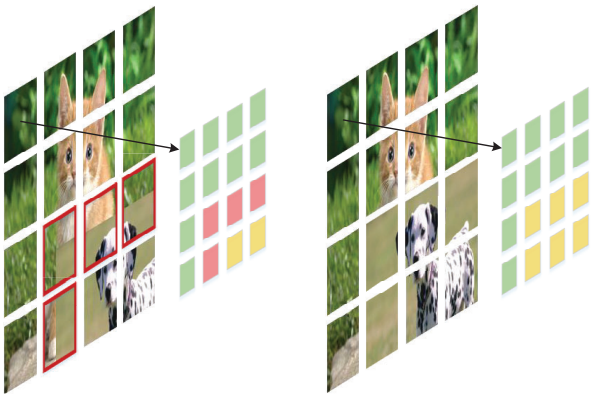


Figure 18: CutMix (Left) and MixToken (Right)

MixToken solve this problem by doing patch-based CutMix, where the CutMix is not pixels but patches now. The equations is similar as CutMix. Figure shows a comparison of CutMix and MixToken. Using MixToken is helpful for token labeling and thus improve the performance.

6 Experiment

6.1 Experiment Setting

6.1.1 Network

To compare the difference between CNN and Transformer, we train the following networks, which is devised based on the above ideas. To be specific, we train the original LVT, LVT with wideresnet added and compare with ResNet18. With the idea introduced above, we can adjust the parameters and FLOPS freely and making the parameter and FLOPs of CNN and Transformer are almost the same. But due to the special structure, we can not get the same parameter and FLOPs at the same time. Therefore, we compare them respectively. We also test CrossViT in our experiment.

Model	Params (M)	FLOPs (Mac)
LVT	5.52	839.6 M
LVT + WideResNet10-9	11.28	1.64 G
LVT + WideResNet10-10	12.69	1.83 G
ResNet18	11.23	1.82 G
CrossViT	7.01	1.29 G

Table 2: Parameters and FLOPS of the Testing Models

In our experiment, we found that the training of Transformer spend much more time than CNN. Therefore, we do not train our model from scratch. Instead, we use the model pretrained on ImageNet and simply replace the head to adapt to the 100 classes of CIFAR-100 and finetune it.

6.1.2 Data Augmentation

We apply three data augmentations we introduced above, i.e. **MixUp**, **Manifold Mixup**, **CutMix** and **MixToken** on the training of the original LVT and compare it with the baseline.

6.1.3 Loss Function

We use **Cross Entropy** Loss as the loss function. Denote x as the predicted value of a sample and y as the one-hot label, then the Cross Entropy Loss can be represented as below.

$$l(x, y) = -\log(p_i), \text{ where } y_i = 1, \ p_i \quad = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

6.1.4 Learning Rate

We apply the **Cosine Annealing** as the learning rate decay strategy.

$$\eta_t = \frac{1}{2}\eta_{max}\left(1 + \cos\frac{T_t}{T_{max}}\pi\right)$$

η_t denotes the learning rate at the T_t epoch. We set η_{max} as $1e-3$ and T_{max} as 30, since we train from a pretrained model and **30 epochs** is enough for the it. Cosine annealing strategy has the advantage that the learning rate decays slowly in the beginning epochs, and decays faster in the later epochs, which is consistent with the training process.

6.1.5 Optimizer

We use **Mini-Batch Gradient Descent** (SGD) with **batch size 16**, **momentum 0.9** and **weight decay 5e-4** while training.

6.2 Results

We do several experiments on CIFAR-100. First, we try to train LVT from scratch and finetune the pretrained model respectively. Then we train our model CNN-Cross-LVT, and compare it with the original LVT and CNN with same parameters and FLOPS. In addition, we test the performance of different data augmentations on LVT. Finally, we visualize some results of LVT.

6.2.1 LVT and CrossViT

We train LVT from scratch first. We train it for 100 epochs, and the other settings are the same as we introduce before. LVT can only reach an accuracy of 64.93%. This shows that simply training Transformer on CIFAR-100 can not get a good performance. More data or technique should be applied for better performance.

To get a better performance, we train LVT using the model pretrained on ImageNet. We replace the linear layer and finetune it for 30 epochs. We upsample the image first by using bilinear interpolation and send it to LVT. As we can see in Figure19, this can get a better performance on CIFAR-100. It can reach an accuracy of 85.82%. This indicates that using the pure CIFAR-100 can not train a good classification transformer easily. Transformer performs better in big dataset.

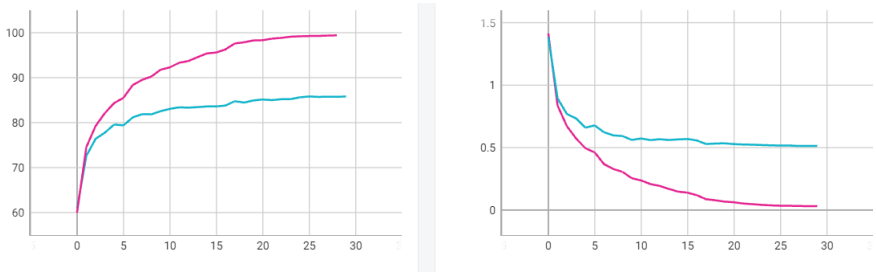


Figure 19: Training Curve of LVT Finetuned
Red: Train; Blue: Valid

Therefore, to get a better performance, for the following training of transformer, we adopt the above procedure. To be fair, for CNN, we also use this.

We also train CrossViT and verify our modification is effective. From the result, we can see that the our modification does work for an increase of 0.66%. This makes sense because the original one is a special case of the modified one.

Model	LVT from scratch	LVT finetuned	Original CrossViT	Modified CrossViT
Accuracy (%)	64.93	84.82	84.93	85.59

Table 3: Accuracy of LVTs and CrossViTs

6.2.2 Comparison between CNN and Transformer

The comparison of CNN and Transformer is shown in table4. For CNN and Transformer with same parameters and FLOPs, Transformer performs better than CNN. Even the original LVT, with less parameters and FLOPS, performs better than ResNet18. This shows the power of Transformer in image classification.

The results also verifies that the way to combine CNN and Transformer together works. When this adds the parameters and FLOPs, it also improve the performance.

Model	Accuracy (%)
LVT	84.82
LVT + WideResNet10-9	85.22
LVT + WideResNet10-10	85.26
ResNet18	82.52

Table 4: Comparison between CNN and Transformer

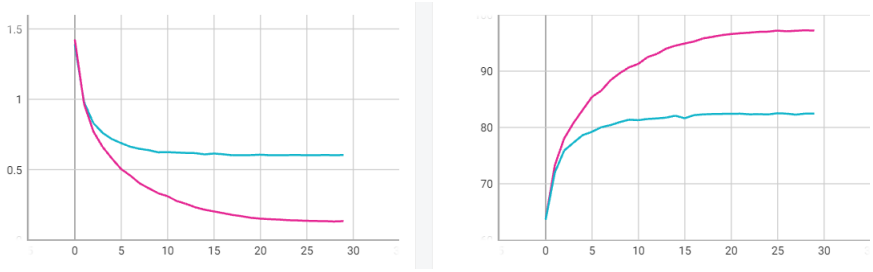


Figure 20: Training Curve of ResNet18
Red: Train; Blue: Valid

6.2.3 Data Augmentations on Transformer

We test different data augmentations we introduced on the training of original LVT. The result is show in table5. All the data augmentations improve the performance. However, the data augmentations have different performance on Transformer and CNN. In the midterm project, we find Manifold Mixup performs best on CNN, while here, on Transformer, CutMix and MixToken performs better. And the difference between MixUp and Manifold Mixup is small. MixToken does improves CutMix by a little.

We think that the CutMix-like augmentation performs better than MixUp-like augmentation. The reason may be Transformers transform the image to tokens firstly. The CutMix-like augmentations changes the image on token layer, and this may cause better performance.

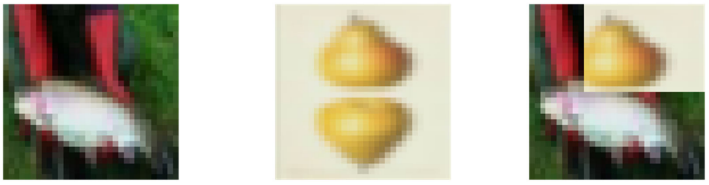
Data Augmentation	Accuracy (%)
Baseline	84.82
MixUp	85.57
Manifold Mixup	85.57
CutMix	86.28
MixToken	86.45

Table 5: Different data augmentation on LVT

6.2.4 Visualization

We visualize the attention map of LVT. The attention map is the feature matrix generated by Transformer, and it shows which region the Transformer focuses on. Figure shows some attention map of the LVT.

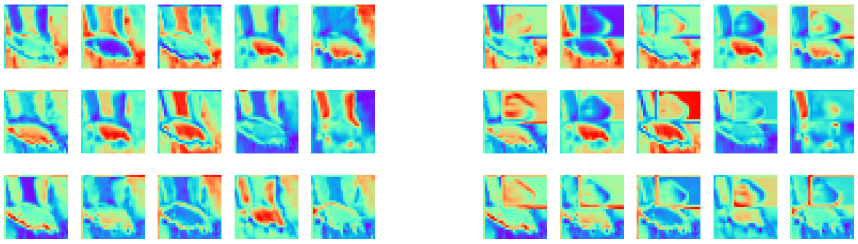
We choose two images to visualize, one of which is labeled 'trout' and the other is 'pear'. Figure21 shows the two images and the CutMix image of them. We visualize some attention map of the 'trout' and the CutMix image in Figure.



trout pear CutMix

Figure 21: Testing Images

From the attention map of 'trout', we can find that Transformer mainly focus on the trout, while some focus on the hand of man or the background, but still near the trout. When we look at the attention map of the CutMix image, we find the attention map changes a lot. More attention are on the pear, although the pear region is smaller than the trout. From human perception, pear and trout play the same role in this image.



(a) Attention Map of 'trout'

(b) Attention Map of CutMix

Figure 22: Attention Map

This suggests that the target label of CutMix is not quite suitable, since the importance should not be totally depended on the area of the image. It's hard to solve this in CNN, but in Transformer, it's possible to make use of the attention map to get a better target label.

7 Conclusion

In this task, we introduce some prevalent vision Transformer and propose our model. We test their performance on CIFAR-100 and compare them with CNN. We also test the performance of different data augmantions on Transformer. Finally, we visualize the Transformer by showing the attention map and analysis it.

References

[1] Enze Xie, Wenhai Wang, Zhiding Yu, et al. SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers 2.1

[2] Liang-Chieh Chen, Yukun Zhu, et al. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation 2.2

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. An image is worth 16x16 words: Transformers for image recognition at scale 1.2

[4] Chen C F R, Fan Q, Panda R. Crossvit: Cross-attention multi-scale vision transformer for image classification 4

[5] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, & David Lopez-Paz. Mixup: Beyond Empirical Risk Minimization 5.2

[6] Verma V, Lamb A, Beckham C, et al. Manifold mixup: Better representations by interpolating hidden states 5.3

[7] Jiang Z H, Hou Q, Yuan L, et al. All tokens matter: Token labeling for training better vision transformers