Final Project 黃詩瑜

我先將 HW10 的 ViT model 套用在 midterm 針對 cifar100 的 code 上試跑，參數使用：

layer	embed dim	patch size	head	epoch
8	64	6	4	100

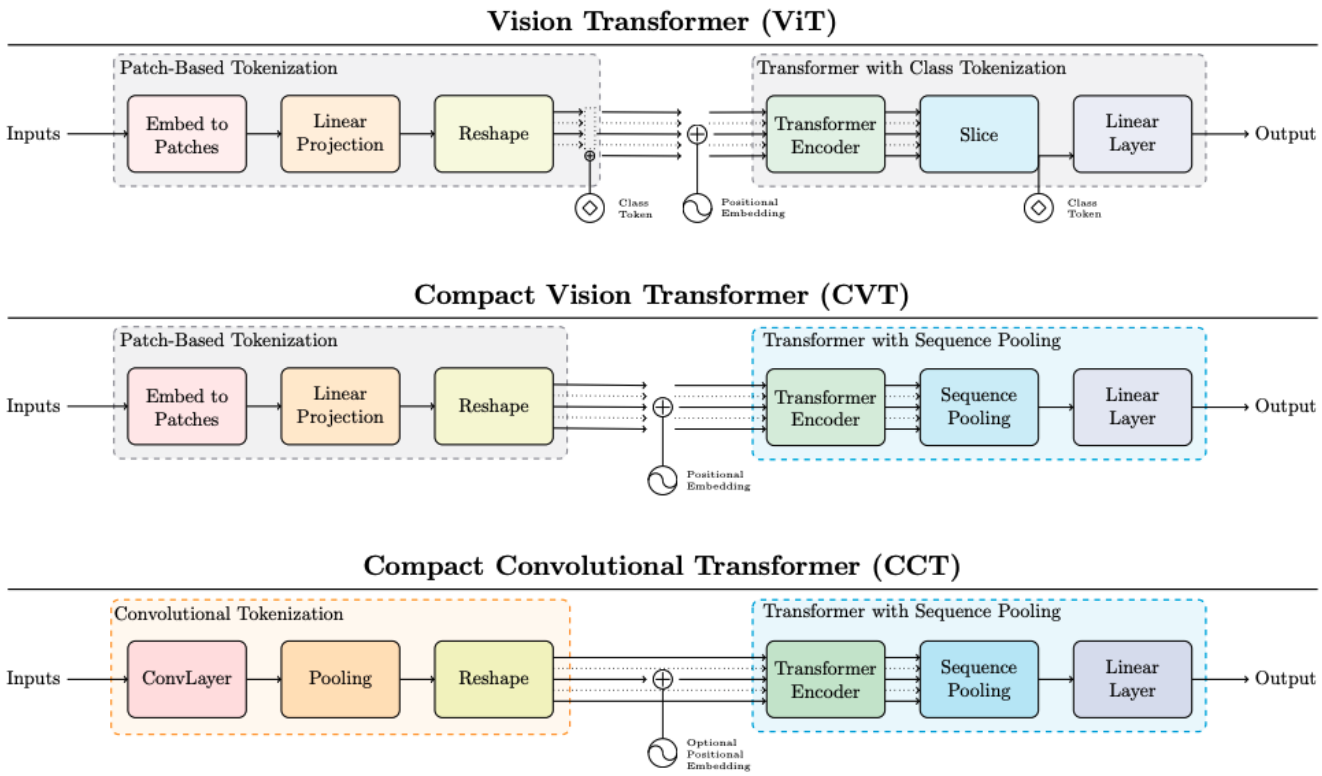
結果 accuracy 只有 50% 左右，parameter size 則是 1.56M，加大整個 model 至 layer 12 層 / embed dimension 768 後，accuracy 依舊在50~60% 左右。後來想到 ViT 主要是因為有使用很大的 dataset 做 pretrain，才能達到高的 accuracy。我便將我的 ViT model 設定成 *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* 中 ViT-Base 的標準架構，再 load pretrain data 後，cifar100 的 accuracy 來到了 81%。

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

這次評分標準跟 model 的 parameter size 有關，確定 ViT 在 cifar100 能運作成功後，我想試著使用其他 model 看看成果，便以 Image Classification - CIFAR-100 Benchmark 網站中，選擇幾個 accuracy 排名前 20 的 Transformer-based model，紀錄它們的 Top-1 accuracy 和 model size，計算哪些 model 會讓這次的 final 總分數較高。

發現 CCT 這個 model 的 parameter size 都很小，它源自於 *Escaping the Big Data Paradigm with Compact Transformers*，這篇 paper 提出 了ViT-Lite / CVT / CCT 這 3 個model，算是從 ViT 遞進式依序的演化出來的。



但 ViT-Lite 在 cifar100 的 accuracy 只有 40~69%，這次作業 accuracy 最少要 70% 才有分，因此主要選擇 CVT 和 CCT 跟 ViT 的 model 做比較如下：

pre-trained on ImageNet-21k					
	ViT-S	ViT-B	ViT-B	ViT-L	
cifar100 acc	89.7	91.67	91.97	93.44	
model size(M)	22.2	86	86	307	
score	88.739	25.198	25.547	7.635	
	CvT-7/4	CvT-6/4			
cifar100 acc	73.01	72.25			
model size(M)	3.717	3.19			
score	80.979	70.533			
	CCT 4/3x2	CCT 6/3x1	CCT 6/3x2	CCT-7/3x1	CCT-7/3x2
cifar100 acc	70.46	76.71	74.47	77.05	74.92
model size(M)	0.482	3.168	3.327	3.76	3.853
score	95.436	211.806	134.355	187.500	127.693

> CCT： L/PxC where L is the number of transformer layers, P is the patch/convolution size, and C is the number of convolutional layers.

可看到在 CCT 的分數是比較高的，所以我選擇 CCT 為主要拿來調整的 model。針對 transformer layer，雖然 layer 數越多 accuracy 越高，但 model size 也變多了，反而會造成總分數比較低，因此 transformer layers 我選擇使用 6 層。而 convolutional layer 的部分，這邊有一個小插曲是我原本以為 model size 指的是 'Estimated Total Size'

SIZE	CCT 6/3x1	CCT 6/3x2
Total params	3,191,397	3,333,669
Trainable params	3,191,397	3,333,669
Non-trainable params	0	0
Input size (MB)	0.01	0.01
Forward/backward pass size (MB)	69.50	16.25
Params size (MB)	12.17	12.72
Estimated Total Size (MB)	81.69	28.98

而 Estimated Total Size 除了 parameter size 之外會加上 Forward / backward pass size 等，看以下的源代碼得知 Forward / backward pass size 就是所有 layer 的 output 的大小。

```
for layer in summary:

   total_output += np.prod(summary[layer]["output_shape"])

   # assume 4 bytes/number (float on cuda).
   total_output_size = abs(2. * total_output * 4. /
                           (1024 ** 2.))  # x2 for gradients

   summary_str += "Forward/backward pass size (MB): %0.2f"
                               % total_output_size + "\n"

```

CCT 6/3x1 的 Forward / backward pass size 比 CCT 6/3x2 還要大非常多是因為在前面的 convolution layer，CCT 6/3x1 只有一層，做完經過 flatten 後，之後好幾層 layer 的 output shape 會維持在 [-1, 256, 256]。

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 256, 32, 32]           6,912
              ReLU-2          [-1, 256, 32, 32]               0
         MaxPool2d-3          [-1, 256, 16, 16]               0
           Flatten-4             [-1, 256, 256]               0
         Tokenizer-5             [-1, 256, 256]               0
```

而 CCT 6/3x2 有兩層 conv layer，做完經過 flatten 後，後面的 layer 的 output shape 維持在 [-1, 64, 256]。

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 32, 32]           1,728
              ReLU-2          [-1, 64, 32, 32]               0
         MaxPool2d-3          [-1, 64, 16, 16]               0
            Conv2d-4         [-1, 256, 16, 16]         147,456
              ReLU-5         [-1, 256, 16, 16]               0
         MaxPool2d-6           [-1, 256, 8, 8]               0
           Flatten-7              [-1, 256, 64]               0
         Tokenizer-8              [-1, 64, 256]               0
```

所以 CCT 6/3x1 的 Estimated Total Size 比 CCT 6/3x2 還要大非常多。但若是單看 Total params 的話，則是 convolution layer 較少的比較佔優勢，所以我選擇 CCT 6/3x1 為主要 model。

Improve the Accuracy

選定 model 架構後，選擇使用與調整以下的一些方法來去優化 model 與提高 accuracy。

1. Dataset Normalization
2. Data Augmentation
3. Drop Out
4. Drop Path
5. Learning Rate Scheduling
6. Optimizer Selection
7. Criterion

Dataset Normalization

Torchvision package 中提供了常見的數據預處理操作，封裝在transforms 中，transforms 涵蓋了大量對 Tensor 和對 PIL Image 的處理操作，其中包含了進行歸一化的 transforms.normalize() 函數。

CLASS `torchvision.transforms.Normalize(mean, std, inplace=False)` [SOURCE]

Normalize a tensor image with mean and standard deviation. This transform does not support PIL Image. Given mean: `(mean[1],...,mean[n])` and std: `(std[1],..,std[n])` for `n` channels, this transform will normalize each channel of the input `torch.*Tensor` i.e., `output[channel] = (input[channel] - mean[channel]) / std[channel]`

給予 cifar100 計算出的 mean、std， Normalize() 函數可以對每個channel 的圖像進行標準化，均值變為 0，標準差為 1，將數據轉換為標準高斯分佈，優點是可以加快模型的收斂與提高 model 的精準度。

```
mean = {
'cifar100': (0.5071, 0.4867, 0.4408),
}

std = {
'cifar100': (0.2675, 0.2565, 0.2761),
}
```

Data Augmentation

Data Augmentation 是對現有數據的一些修改來增加數據量，可以補足因為數據量太少而對 training dataset 產生 over-fitting 的現象。我使用的修改方法除了之前常見的 RandomHorizontalFlip() 對圖片做水平翻轉之外，還用到了 AutoAugment，就是在各種資料集上搜尋出能夠最佳化其 validation Set 準確率的 Data Augmentation 演算法，AutoAugment 目前只有在 CIFAR-10、SVHN 和 ImageNet 上搜尋出來，所以 CIFAR-100 使用的會是 CIFAR-10 的 AutoAugment。

AutoAugment: Learning Augmentation Policies from Data 這篇 paper 中實測 cifar100 不同 model 使用 AutoAugment 的結果，Test set error rates 可以下降 1~3% 左右。

Model	Baseline	Cutout [25]	AutoAugment
Wide-ResNet-28-10 [56]	18.80	18.41	17.09
Shake-Shake (26 2x96d) [58]	17.05	16.00	14.28
PyramidNet+ShakeDrop [59]	13.99	12.19	**10.67**

下方 Transforms.Compose 中的 CIFAR10Policy() 會隨機選取 CIFAR10 中 25 個 Sub-policies 的其中 1 個來做使用。每一個 Sub-Policies 帶有兩種 Augment 和一個在每個 mini-batch 被使用的 Possibility 以及 Magnitudes。在每次 training step 會選到一種 Sub-Policy，其中的兩個 Augment 各有一個機率去決定是否要被應用在這個 mini-batch 上，如果要使用就會利用當初設定好的 Magnitudes。

```
augmentations = transforms.Compose([
    transforms.RandomCrop(img_size, padding=4),
    transforms.RandomHorizontalFlip(),
    CIFAR10Policy(),
    transforms.ToTensor(),
    *normalize,
])
```

全部用到的 Augment 有：

> shearX：Apply affine shear on the x-axis
> shearY：Apply affine shear on the y-axis
> translateX：Apply affine translation on the x-axis
> translateY：Apply affine translation on the y-axis
> rotate：Apply affine rotation on the y-axis
> color： Color enhancer instance
> posterize：Reduce the number of bits for each color channel
> solarize：Invert all pixel values above a threshold
> contrast：Adjust image contrast
> sharpness：Adjust image sharpness
> brightness：Adjust image brightness
> autocontrast：Maximize (normalize) image contrast
> equalize：Equalize the image histogram
> invert：Invert (negate) the image

Drop Out

使用 torch.nn package 中 的 Dropout ，是個可以避免 model 過於 overfitting 的方法，在訓練時每個 epoch 會以一定的機率丟棄 hidden layer 的神經元，避免神經元之間過度依賴，而測試時不會丟棄神經元。如果在訓練時以機率 p 丟棄神經元，而測試時不會丟棄神經元，會造成測試的結果比訓練大 1/(1−p) 倍，所以在訓練時 outputs 會乘以1/(1−p) 倍。

CLASS `torch.nn.Dropout(p=0.5, inplace=False)` [SOURCE]

During training, randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper Improving neural networks by preventing co-adaptation of feature detectors .

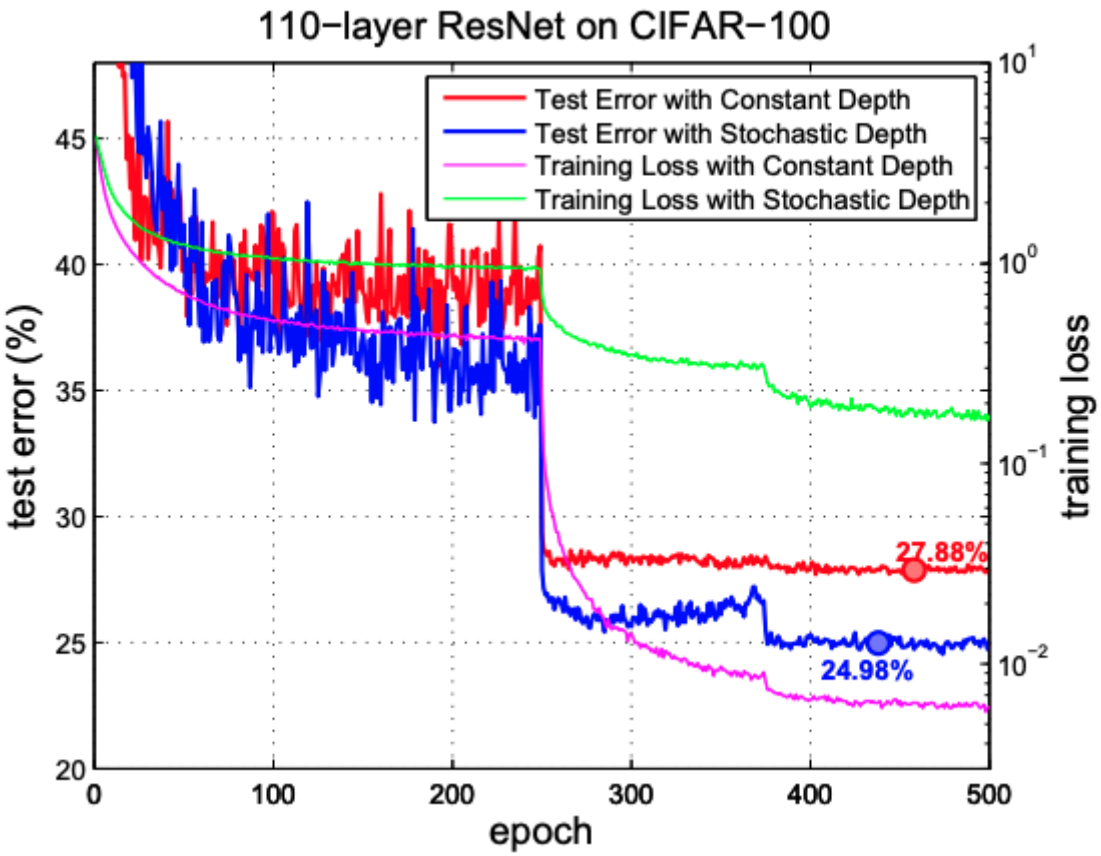
Furthermore, the outputs are scaled by a factor of $\frac{1}{1-p}$ during training. This means that during evaluation the module simply computes an identity function.

Dropout: A Simple Way to Prevent Neural Networks from Overfitting 這篇 paper 中對於 cifar100 測試，使用 drop out 對於 error rates 是有優化的。

Method	CIFAR-10	CIFAR-100
Conv Net + max pooling (hand tuned)	15.60	43.48
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	15.13	42.51
Conv Net + max pooling (Snoek et al., 2012)	14.98	-
Conv Net + max pooling + dropout fully connected layers	14.32	41.26
Conv Net + max pooling + dropout in all layers	12.61	**37.20**
Conv Net + maxout (Goodfellow et al., 2013)	**11.68**	38.57

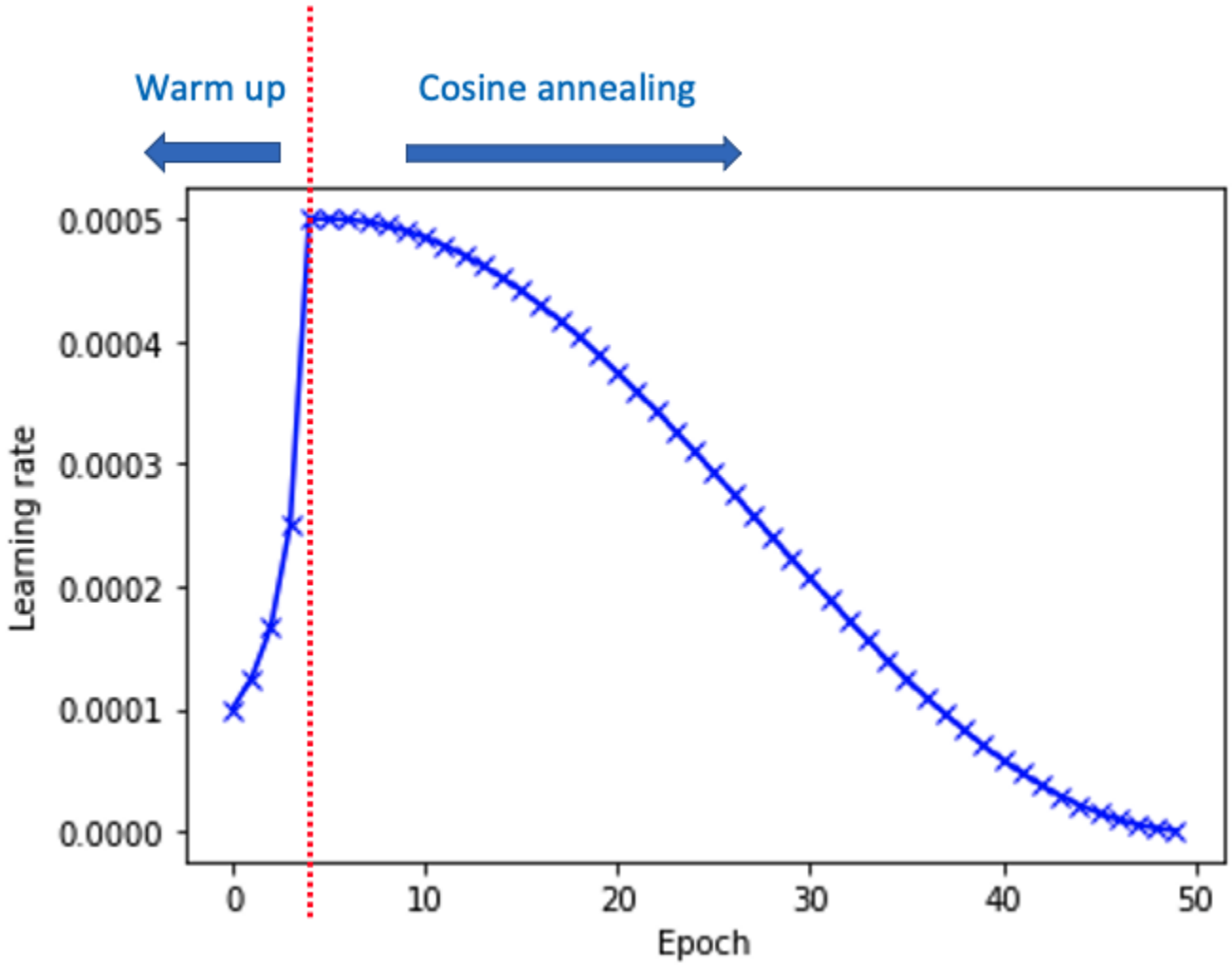
Drop Path

在 CCT 的 transformer encoder 那個 block 使用到了 drop path，類似 drop out ，但它是將網路裡面的分支結構隨機刪除，在訓練過程中，隨機去掉很多層，使用較淺的深度，在測試時再使用完整的網絡，也就是較深的深度，而去掉中間幾層訓練對最終的結果也沒什麼影響，不會影響到收斂性，但可使用較少訓練時間，提高訓練性能。 它源自於 *Deep Networks with Stochastic Depth*，其中針對 cifar100 使用 ResNet 加上 Drop Path，測試結果可看出 Test error rates 可以下降 3 % 左右。



Learning Rate Scheduling

Learning rate 的設定是使用 cosine annealing 加上前 10% epoch 的 warm up。若設定總 epoch 數為 50，我的 learning rate 會呈現如下圖的趨勢。



這個 learning rate 設定在之前 midterm 中就有想要使用，但那時沒實現出來。

Warm up:
在訓練的一開始， weight 的初始值為隨機生成的，對於訓練資料分布的理解度為零，這樣就容易讓學習一開始就往第一筆資料的分佈方向修正，可能導致針對此筆資料的 overfitting，後續要靠多次訓練才能將資料分布拉回來。而 Gradual warm up 這個方法是一開始用小一點的 learning rate，model 不會更新太多，每個 epoch 增加一些 learning rate ，等達到原先設定 warmup 的 epoch 數量後，warm up 結束，learning rate 變回正常值。優點是能達到更佳的收斂點，也可以使學習更加穩定。

Cosine annealing:
Cosine learning rate 是 learning rate decay 的一種，現在似乎蠻普遍使用。當訓練經過的 epoch 越多時，通常會越接近 loss 的低點，這時 step 應該要小一點，也就是 learning rate 要越來越小。而 cosine learning rate 就是利用 cosine 函數來降低 learning rate，隨著 epoch 的增加，cosine 值首先緩慢下降，然後加速下降，再次緩慢下降。這種下降模式能和 learning rate 配合。

Optimizer Selection

Optimizer 是使用 AdamW，雖然不確定結果是否真的比使用 Adam 還要好。Adam 算是一個蠻主流的 adaptive learning rate，有快速收斂與調整參數容易的優點，但有些論文提出說 Adam 在 testing 時的誤差會比 training 差上許多，且有一些收斂的問題。後來在 *Fixing Weight Decay Regularization in Adam* 這篇 paper 說 Adam optimizer 中實現的 weight decay 的方法似乎是錯誤的，並提出了新的 AdamW 這個 optimizer 來解決，調整了計算 regularization term 的位置，讓 Adam 的 weight decay 與 SGD 這類 optimizer 的行為一致。

```
optimizer = torch.optim.AdamW(model.parameters(), lr=lr,
                              weight_decay=weight_decay)
```

Criterion

我之前作業針對分類的 dataset 所使用的 criterion 都是 cross entropy，而這次 model 使用的是 label smoothing cross entropy。一般的 cross entropy loss 在訓練過程中 model 會往正確標籤和錯誤標籤差值最大化的方向學習，在訓練數據量較小的情況下，會導致過於擬合。Label smoothing 可以解決這個問題，提升圖像分類的準確率，通過軟化傳統的 one-hot 類型標籤，它讓所有的標籤都能參與到 loss 函數的計算過程中而不只有正確的標籤，有效避免過於 overfitting 的現象。

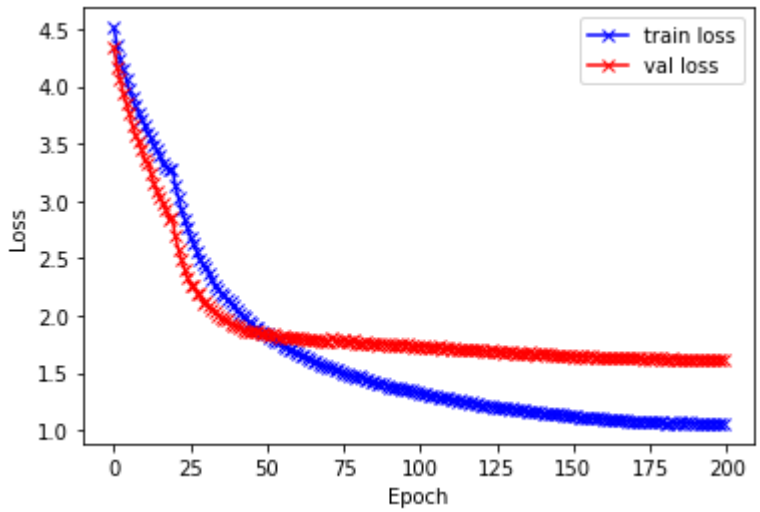
```
criterion = LabelSmoothingCrossEntropy()
```

Model Size

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 256, 32, 32]           6,912
              ReLU-2          [-1, 256, 32, 32]               0
         MaxPool2d-3          [-1, 256, 16, 16]               0
           Flatten-4             [-1, 256, 256]               0
         Tokenizer-5             [-1, 256, 256]               0
           Dropout-6             [-1, 256, 256]               0
         LayerNorm-7             [-1, 256, 256]             512
            Linear-8             [-1, 256, 768]         196,608
           Dropout-9          [-1, 4, 256, 256]               0
           Linear-10             [-1, 256, 256]          65,792
          Dropout-11             [-1, 256, 256]               0
        Attention-12             [-1, 256, 256]               0
         Identity-13             [-1, 256, 256]               0
        LayerNorm-14             [-1, 256, 256]             512
           Linear-15             [-1, 256, 512]         131,584
          Dropout-16             [-1, 256, 512]               0
           Linear-17             [-1, 256, 256]         131,328
          Dropout-18             [-1, 256, 256]               0
         Identity-19             [-1, 256, 256]               0
TransformerEncoderLayer-20             [-1, 256, 256]               0
        LayerNorm-21             [-1, 256, 256]             512
           Linear-22             [-1, 256, 768]         196,608
          Dropout-23          [-1, 4, 256, 256]               0
           Linear-24             [-1, 256, 256]          65,792
          Dropout-25             [-1, 256, 256]               0
        Attention-26             [-1, 256, 256]               0
         DropPath-27             [-1, 256, 256]               0
        LayerNorm-28             [-1, 256, 256]             512
           Linear-29             [-1, 256, 512]         131,584
          Dropout-30             [-1, 256, 512]               0
           Linear-31             [-1, 256, 256]         131,328
          Dropout-32             [-1, 256, 256]               0
         DropPath-33             [-1, 256, 256]               0
TransformerEncoderLayer-34             [-1, 256, 256]               0
        LayerNorm-35             [-1, 256, 256]             512
           Linear-36             [-1, 256, 768]         196,608
          Dropout-37          [-1, 4, 256, 256]               0
           Linear-38             [-1, 256, 256]          65,792
          Dropout-39             [-1, 256, 256]               0
        Attention-40             [-1, 256, 256]               0
         DropPath-41             [-1, 256, 256]               0
        LayerNorm-42             [-1, 256, 256]             512
           Linear-43             [-1, 256, 512]         131,584
          Dropout-44             [-1, 256, 512]               0
           Linear-45             [-1, 256, 256]         131,328
          Dropout-46             [-1, 256, 256]               0
         DropPath-47             [-1, 256, 256]               0
TransformerEncoderLayer-48             [-1, 256, 256]               0
        LayerNorm-49             [-1, 256, 256]             512
           Linear-50             [-1, 256, 768]         196,608
          Dropout-51          [-1, 4, 256, 256]               0
```

```
           Linear-52             [-1, 256, 256]          65,792
          Dropout-53             [-1, 256, 256]               0
        Attention-54             [-1, 256, 256]               0
         DropPath-55             [-1, 256, 256]               0
        LayerNorm-56             [-1, 256, 256]             512
           Linear-57             [-1, 256, 512]         131,584
          Dropout-58             [-1, 256, 512]               0
           Linear-59             [-1, 256, 256]         131,328
          Dropout-60             [-1, 256, 256]               0
         DropPath-61             [-1, 256, 256]               0
TransformerEncoderLayer-62             [-1, 256, 256]               0
        LayerNorm-63             [-1, 256, 256]             512
           Linear-64             [-1, 256, 768]         196,608
          Dropout-65          [-1, 4, 256, 256]               0
           Linear-66             [-1, 256, 256]          65,792
          Dropout-67             [-1, 256, 256]               0
        Attention-68             [-1, 256, 256]               0
         DropPath-69             [-1, 256, 256]               0
        LayerNorm-70             [-1, 256, 256]             512
           Linear-71             [-1, 256, 512]         131,584
          Dropout-72             [-1, 256, 512]               0
           Linear-73             [-1, 256, 256]         131,328
          Dropout-74             [-1, 256, 256]               0
         DropPath-75             [-1, 256, 256]               0
TransformerEncoderLayer-76             [-1, 256, 256]               0
        LayerNorm-77             [-1, 256, 256]             512
           Linear-78             [-1, 256, 768]         196,608
          Dropout-79          [-1, 4, 256, 256]               0
           Linear-80             [-1, 256, 256]          65,792
          Dropout-81             [-1, 256, 256]               0
        Attention-82             [-1, 256, 256]               0
         DropPath-83             [-1, 256, 256]               0
        LayerNorm-84             [-1, 256, 256]             512
           Linear-85             [-1, 256, 512]         131,584
          Dropout-86             [-1, 256, 512]               0
           Linear-87             [-1, 256, 256]         131,328
          Dropout-88             [-1, 256, 256]               0
         DropPath-89             [-1, 256, 256]               0
TransformerEncoderLayer-90             [-1, 256, 256]               0
        LayerNorm-91             [-1, 256, 256]             512
           Linear-92               [-1, 256, 1]             257
           Linear-93                  [-1, 100]          25,700
TransformerClassifier-94                  [-1, 100]               0
================================================================
Total params: 3,191,397
Trainable params: 3,191,397
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 69.50
Params size (MB): 12.17
Estimated Total Size (MB): 81.69
----------------------------------------------------------------
```

從下方的 loss 可看出當 training 的 epoch 數量越多時， train loss 和 validation loss 會越來越小，代表漸漸的接近 loss 的低點。但在 epoch 數 50 之前，train loss 比 validation loss 還要大，我認為可能是 data augmentation 導致這樣的現象，因為 data augmentation 的目的就是把訓練集變得豐富，製造數據的多樣性和學習的困難來讓network 更 robust，但是在 validation 的時候是不對數據進行太多的 data augmentation，所以 loss 反而較小。也有可能是 drop out 和 drop path 的影響，因為在 training 時 drop out 和 drop path 會隨機屏蔽掉一些神經元與分支，等 validation 的時候在全部一起用上，loss 就會更小了。大概過了 70 個 epoch 後，train loss 小於 validation loss，且差距越來越大，這是因為產生 overfitting 的現象。



Final Result

CCT model 不需要 pretrain data，這是我對 cifar100 訓練 200 個 epoch 後的結果。

cifar100 acc(%)	Model Size(M)
75.145	3.191397