

# ResNet

## Project Overview

For this Lab, we need to train ResNet18 on Tiny ImageNet dataset with SGD as the optimizer for A classification task. I have plotted curves for training loss, training accuracy, validation accuracy and reported the final test accuracy

1. Use Cross-Entropy as the final classification loss function
2. Use Triplet Loss with hard mining as the final classification loss function
3. Use Central Loss as the final classification loss function

## Metrics

Top-5 accuracy is a classification performance metric that measures the percentage of test samples for which the correct label is among the top 5 predicted labels. Top-5 accuracy is a more forgiving metric than standard accuracy, as it allows the model to make some errors in the ranking of the predicted labels. It is commonly used in image classification tasks, where there can be a large number of possible classes, and it may be challenging for the model to identify the correct label with high confidence

## Analysis

### Data Exploration

The Tiny ImageNet is a subset of the ImageNet dataset in the famous [ImageNet Large Scale Visual Recognition Challenge](#) (ILSVRC). The dataset contains 100,000 images of **200 classes** (500 for each class) downsized to 64×64-colored images. Each class has 500 training images, 50 validation images, and 50 test images.



**Fig. 1** Sample Images from the Tiny ImageNet dataset

The dataset has the following folders and files:

- ❖ “**test**”: It’s a folder with testing images
- ❖ “**train**”: It’s a directory that is divided into subdirectories depending on the label, for eg “n02124075”
- ❖ “**val**”: It’s a folder with validation images
- ❖ “**wnids.txt**”: It contains all the ids
- ❖ “**words.txt**”: it contains all the respective mapping of the ids

## Methodology

### Data Preprocessing

The preprocessing consists of the following steps:

- ❖ The images are converted to grayscale to speed up the training process
- ❖ Then they are converted into tensors
- ❖ A dictionary is made to map string IDs in “**wnids.txt**” to an integer mapping
- ❖ A derived class of Dataset called Imagedataset is made to make the data present in the folder compatible with PyTorch.

### Model Architecture

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 32, 32]	3,136
BatchNorm2d-2	[-1, 64, 32, 32]	128
ReLU-3	[-1, 64, 32, 32]	0
MaxPool2d-4	[-1, 64, 16, 16]	0
Conv2d-5	[-1, 64, 16, 16]	36,864
BatchNorm2d-6	[-1, 64, 16, 16]	128
ReLU-7	[-1, 64, 16, 16]	0
Conv2d-8	[-1, 64, 16, 16]	36,864
BatchNorm2d-9	[-1, 64, 16, 16]	128
ReLU-10	[-1, 64, 16, 16]	0
Block-11	[-1, 64, 16, 16]	0
Conv2d-12	[-1, 64, 16, 16]	36,864
BatchNorm2d-13	[-1, 64, 16, 16]	128
ReLU-14	[-1, 64, 16, 16]	0
Conv2d-15	[-1, 64, 16, 16]	36,864
BatchNorm2d-16	[-1, 64, 16, 16]	128
ReLU-17	[-1, 64, 16, 16]	0
Block-18	[-1, 64, 16, 16]	0
Conv2d-19	[-1, 128, 8, 8]	73,728
BatchNorm2d-20	[-1, 128, 8, 8]	256
ReLU-21	[-1, 128, 8, 8]	0
Conv2d-22	[-1, 128, 8, 8]	147,456
BatchNorm2d-23	[-1, 128, 8, 8]	256
Conv2d-24	[-1, 128, 8, 8]	8,192
BatchNorm2d-25	[-1, 128, 8, 8]	256
ReLU-26	[-1, 128, 8, 8]	0
Block-27	[-1, 128, 8, 8]	0
Conv2d-28	[-1, 128, 8, 8]	147,456
BatchNorm2d-29	[-1, 128, 8, 8]	256
ReLU-30	[-1, 128, 8, 8]	0
Conv2d-31	[-1, 128, 8, 8]	147,456

BatchNorm2d-32	[-1, 128, 8, 8]	256
ReLU-33	[-1, 128, 8, 8]	0
Block-34	[-1, 128, 8, 8]	0
Conv2d-35	[-1, 256, 4, 4]	294,912
BatchNorm2d-36	[-1, 256, 4, 4]	512
ReLU-37	[-1, 256, 4, 4]	0
Conv2d-38	[-1, 256, 4, 4]	589,824
BatchNorm2d-39	[-1, 256, 4, 4]	512
Conv2d-40	[-1, 256, 4, 4]	32,768
BatchNorm2d-41	[-1, 256, 4, 4]	512
ReLU-42	[-1, 256, 4, 4]	0
Block-43	[-1, 256, 4, 4]	0
Conv2d-44	[-1, 256, 4, 4]	589,824
BatchNorm2d-45	[-1, 256, 4, 4]	512
ReLU-46	[-1, 256, 4, 4]	0
Conv2d-47	[-1, 256, 4, 4]	589,824
BatchNorm2d-48	[-1, 256, 4, 4]	512
ReLU-49	[-1, 256, 4, 4]	0
Block-50	[-1, 256, 4, 4]	0
Conv2d-51	[-1, 512, 2, 2]	1,179,648
BatchNorm2d-52	[-1, 512, 2, 2]	1,024
ReLU-53	[-1, 512, 2, 2]	0
Conv2d-54	[-1, 512, 2, 2]	2,359,296
BatchNorm2d-55	[-1, 512, 2, 2]	1,024
Conv2d-56	[-1, 512, 2, 2]	131,072
BatchNorm2d-57	[-1, 512, 2, 2]	1,024
ReLU-58	[-1, 512, 2, 2]	0
Block-59	[-1, 512, 2, 2]	0
Conv2d-60	[-1, 512, 2, 2]	2,359,296
BatchNorm2d-61	[-1, 512, 2, 2]	1,024
ReLU-62	[-1, 512, 2, 2]	0
Conv2d-63	[-1, 512, 2, 2]	2,359,296
BatchNorm2d-64	[-1, 512, 2, 2]	1,024
ReLU-65	[-1, 512, 2, 2]	0
Block-66	[-1, 512, 2, 2]	0
AdaptiveAvgPool2d-67	[-1, 512, 1, 1]	0
Linear-68	[-1, 200]	102,600

=====

Total params: 11,272,840

Trainable params: 11,272,840

Non-trainable params: 0

-----

Input size (MB): 0.02

Forward/backward pass size (MB): 5.13

Params size (MB): 43.00

Estimated Total Size (MB): 48.15

-----

## Cross Entropy

The basic idea behind cross-entropy is to measure the difference between the predicted probability distribution and the true probability distribution of the labels. The loss function aims to minimize this difference, which is often referred to as the "cross-entropy loss".

The cross-entropy loss is defined as follows:

$$L = -1/N * \sum(y * \log(y\_hat) + (1-y) * \log(1-y\_hat))$$

where:

- ❖ L is the cross-entropy loss
- ❖ N is the number of samples in the dataset
- ❖ y is the true label (0 or 1)

- ❖  $\hat{y}$  is the predicted probability of the label being 1

The cross-entropy loss penalizes the model more heavily when it makes confident incorrect predictions. For example, if the true label is 0 and the model predicts a probability of 0.9 for label 1, the loss will be larger than if the model predicts a probability of 0.6 for label 1.

## Implementation

- ❖ To implement cross-entropy in a machine learning model, we use a SoftMax activation function in the output layer of the neural network. The SoftMax function converts the output of the network into a probability distribution over the classes.
- ❖ Then, we would calculate the cross-entropy loss for each training example and update the model parameters using backpropagation to minimize the loss. This process is repeated over multiple epochs until the model converges to a good solution.

## Triplet Loss

Triplet loss is a loss function used in deep learning for tasks such as image retrieval and face recognition. The goal of triplet loss is to learn a mapping from an input space to a feature space, where similar inputs are mapped close together in the feature space. The basic idea behind triplet loss is to compare the embeddings of three examples

- ❖ an **anchor** example,
- ❖ a **positive** example
- ❖ a **negative** example.

The anchor and positive examples should be similar, while the negative example should be dissimilar. The loss function aims to minimize the distance between the anchor and positive examples, while maximizing the distance between the anchor and negative examples.

The triplet loss function is defined as follows:

$$L = \max(0, d(a, p) - d(a, n) + \text{margin})$$

where:

- ❖ L is the triplet loss
- ❖ a is the anchor example
- ❖ p is the positive example
- ❖ n is the negative example
- ❖  $d(a, p)$  is the distance between the anchor and positive examples
- ❖  $d(a, n)$  is the distance between the anchor and negative examples
- ❖ margin is a hyperparameter that defines the minimum distance between the anchor and negative examples, for our case the value of margin is chosen to be **0.2**

## Implementation

- To implement triplet loss, we use the ResNet18 model to learn a mapping from the input space to the feature space. The network would take in three examples (anchor, positive, and negative) and output their embeddings in the feature space.
- In this Implementation, hard mining is done where for each anchor, we select the hardest positive (biggest distance  $d(a, p)$ ) and the hardest negative among the batch. According to the [paper](#) cited, the batch hard strategy yields the best performance

## Central Loss

Central loss is a loss function used in deep learning for tasks such as face recognition and clustering. The goal of central loss is to learn a mapping from the input space to a feature space, where similar inputs are mapped close together in the feature space, but with a focus on preserving the intra-class variations.

The basic idea behind central loss is to learn a center for each class in the feature space. The center for each class represents the average feature vector of all examples belonging to that class. The loss function aims to minimize the distance between the feature vector of an example and the center of its corresponding class.

The central loss function is defined as follows:

$$L_c = 1/2 * \sum(y * ||f - C_k||^2)$$

where:

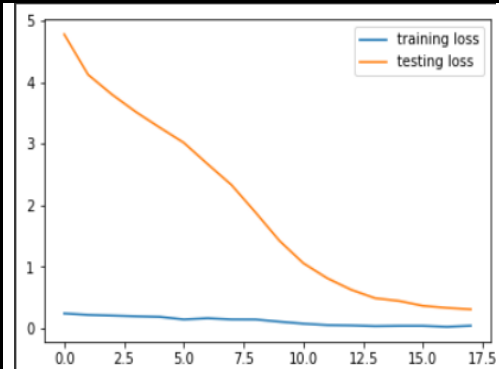
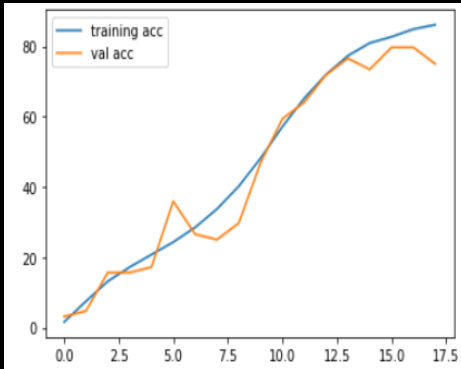
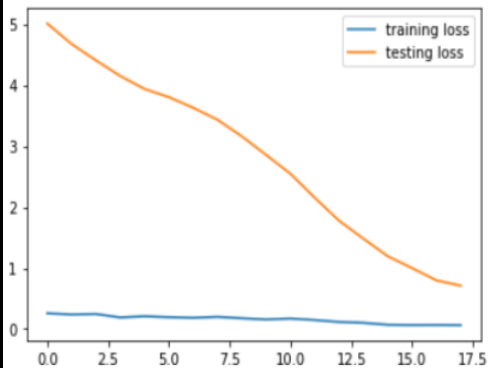
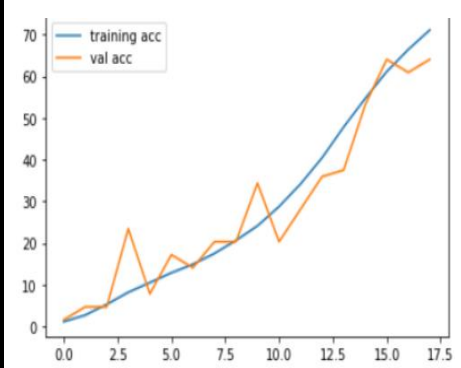
- ❖  $L_c$  is the central loss
- ❖  $y$  is the one-hot label vector for the example
- ❖  $f$  is the feature vector of the example
- ❖  $C_k$  is the center of the corresponding class
- ❖  $||\cdot||^2$  is the L2-norm squared

### *Implementation*



- ❖ To implement central loss, ResNet18 learns a mapping from the input space to the feature space. The network would take in examples and output their embeddings in the feature space.
- ❖ During training, we calculate the central loss for each example and update the model parameters using backpropagation to minimize the loss.
- ❖ We also update the centers for each class in the feature space

# Results and Discussion

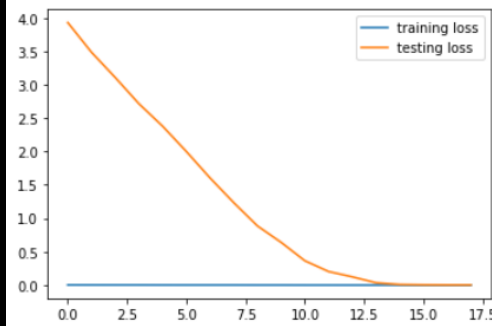
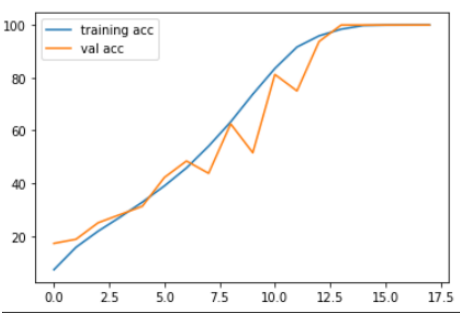
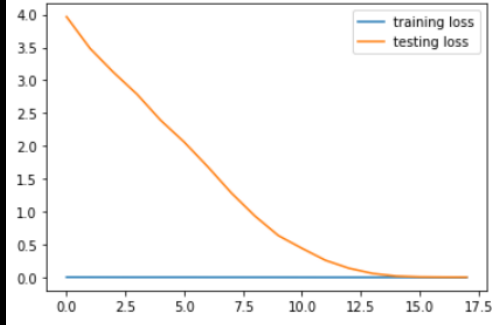
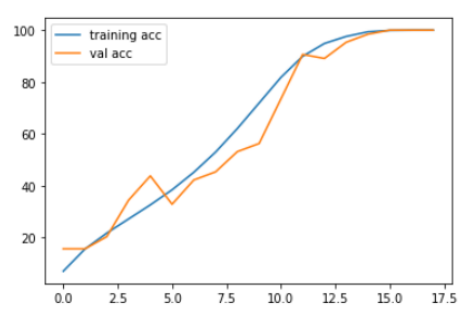
## Cross Entropy

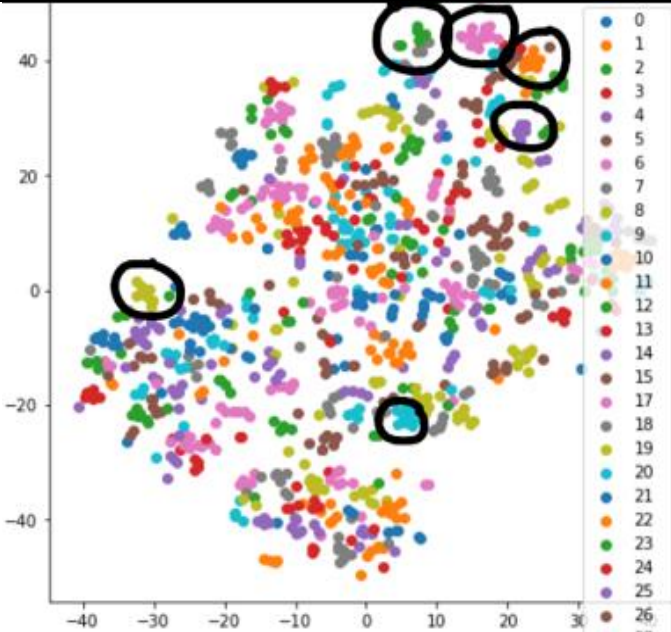
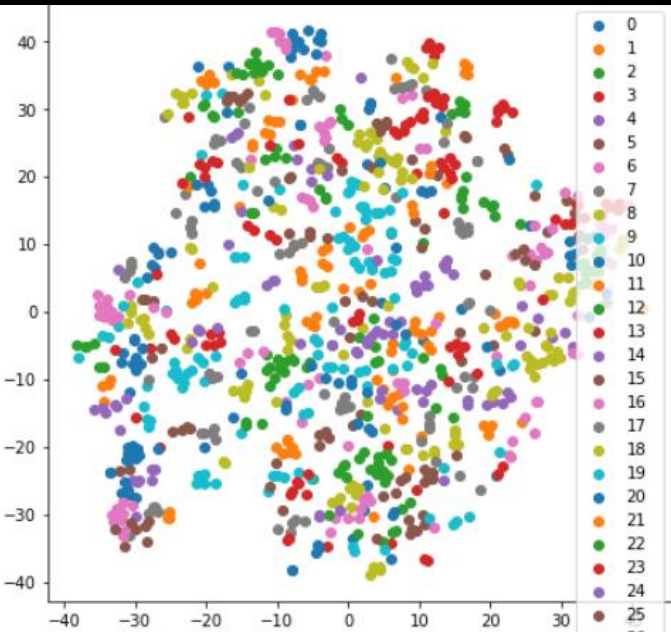
Loss Curves	Accuracy Curves	Top 5 accuracy
		99.71%
		98.44%

## Triplet Loss

Loss curves	Accuracy curves	Top-5 accuracy
		8.57%

Central Loss

Loss curves	Accuracy curves	Top-5 accuracy	lambda
 <p>Plot of training loss (blue) and testing loss (orange) versus epoch (0.0 to 17.5). Training loss is constant at 0.0. Testing loss decreases from 4.0 to 0.0.</p>	 <p>Plot of training accuracy (blue) and validation accuracy (orange) versus epoch (0.0 to 17.5). Training accuracy increases from 10% to 100%. Validation accuracy increases from 20% to 100% with some fluctuations.</p>	99.97%	0.1
 <p>Plot of training loss (blue) and testing loss (orange) versus epoch (0.0 to 17.5). Training loss is constant at 0.0. Testing loss decreases from 4.0 to 0.0.</p>	 <p>Plot of training accuracy (blue) and validation accuracy (orange) versus epoch (0.0 to 17.5). Training accuracy increases from 10% to 100%. Validation accuracy increases from 20% to 100% with some fluctuations.</p>	99.98%	1

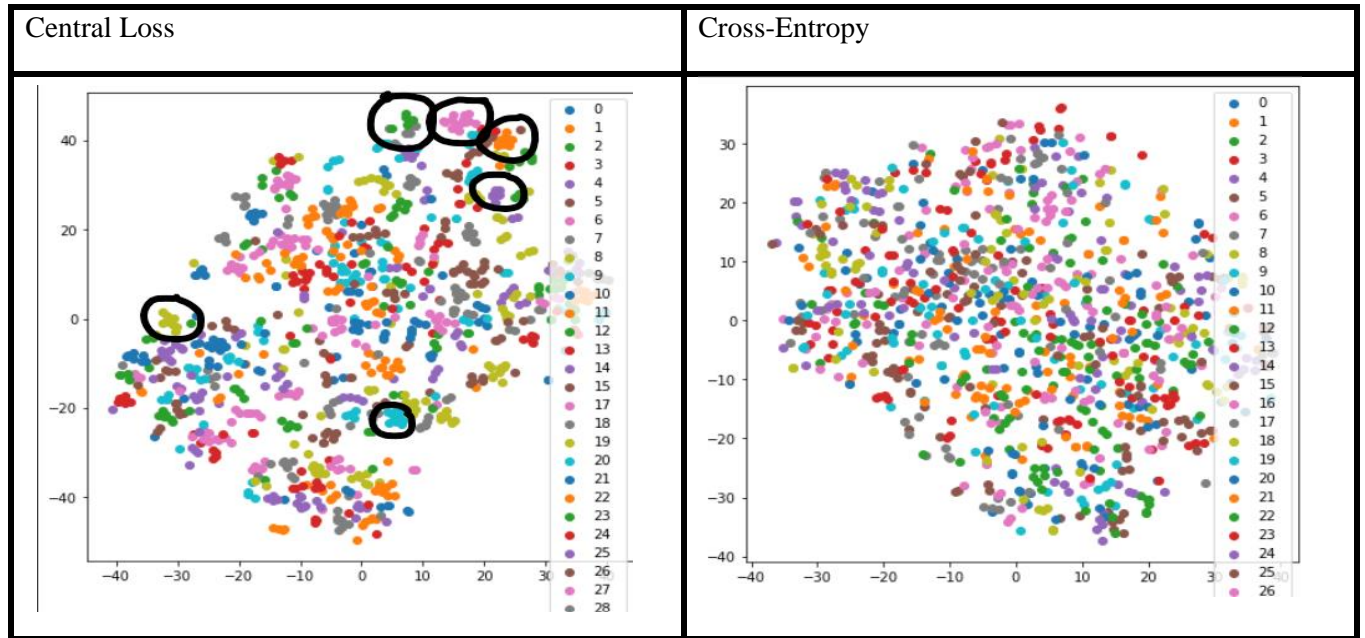
t-SNE plot	Hyperparameter $\lambda$
 <p>A t-SNE plot showing 26 classes (0-25) as colored dots. The classes are highly overlapping and not well-separated. Four specific clusters are circled with black lines, indicating poor separation between different classes.</p>	0.1
 <p>A t-SNE plot showing the same 26 classes. Compared to the first plot, the classes are much more distinct and separated from each other, indicating that a higher value of the hyperparameter <math>\lambda</math> leads to better class separation.</p>	1

## Discussion

The center loss includes a hyperparameter  $\lambda$ , which controls the strength of the regularization. Increasing  $\lambda$  increases the separation of the class centers. As we can observe in the t-SNE plots above. The higher the value of  $\lambda$ , the more clustered and separated from each other we observe. This is because center loss is a regularization strategy that encourages the model to learn widely-separated class representations. The center loss augments the standard supervised loss by adding a penalty term proportional to the distance of a class's examples from its center.



## Central Loss t-SNE plot vs Cross-Entropy t-SNE plot



### Discussion

We can observe that through central loss, the neural network has learned good features since we can see clusters of points that correspond to each class. The points of the same color are grouped together, (represented by black circles in the image) while the points of different colors are separated by some distance. However, in the case of cross-entropy, we see a random scattering of points that is not related to the different classes. This is further validated by the fact central loss gives better accuracy

## Conclusion

From the above, results and discussions we can conclude that Triplet loss performs the best on a tiny Image dataset

---

References used: -

<https://stats.stackexchange.com/questions/496270/what-is-a-center-loss>

<https://omoindrot.github.io/triplet-loss>

[https://ml-cheatsheet.readthedocs.io/en/latest/loss\\_functions.html](https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html)

# Layer-wise pretraining

## Data Exploration

The STL-10 dataset is an image recognition dataset for developing unsupervised feature learning, deep learning, self-taught learning algorithms. It is inspired by the [CIFAR-10 dataset](#) but with some modifications. In particular, each class has fewer labeled training examples than in CIFAR-10, but a very large set of unlabeled examples is provided to learn image models prior to supervised training. The primary challenge is to make use of the unlabeled data (which comes from a similar but different distribution from the labeled data) to build a useful prior.

## Data processing

The preprocessing consists of the following steps:

- ❖ Images are changed to grayscale to speed up the training process
- ❖ They are resized to 96\*96
- ❖ Random Horizontal flip is performed
- ❖ They are then converted to Tensor
- ❖ They are then Normalized


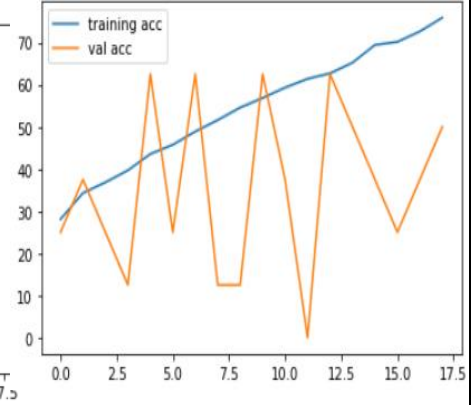
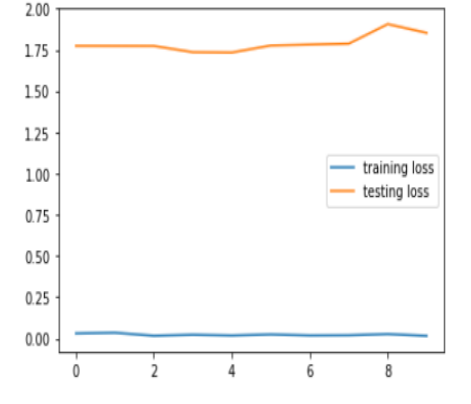
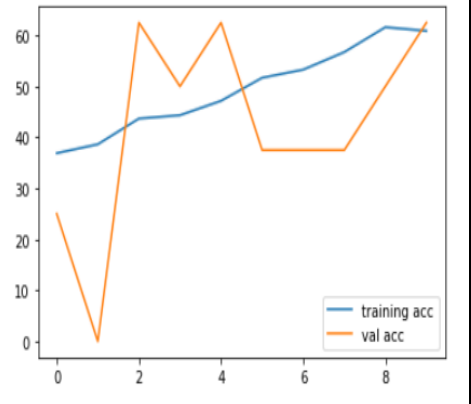
## Implementation of Greedy Layer wise Pretraining

Greedy layer-wise pretraining using an autoencoder is a technique for pretraining a deep neural network in a layer-wise manner before fine-tuning the whole network for a specific task, such as classification. The following steps were used to implement this technique: Firstly, we

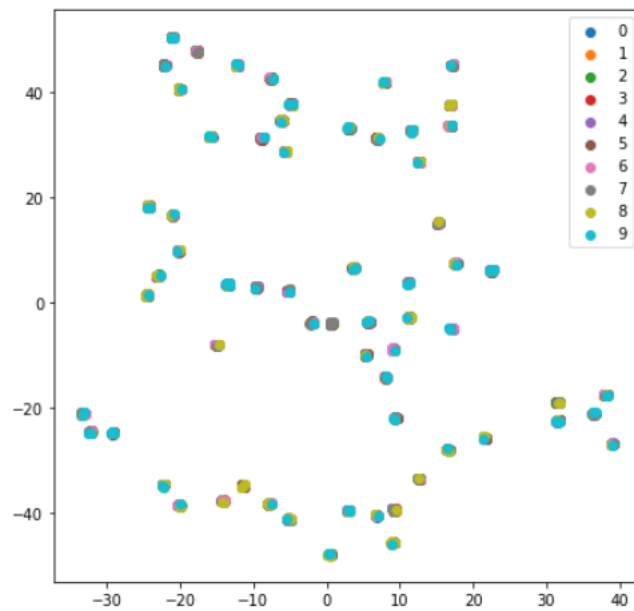
1. Build the autoencoder: The first step is to build an autoencoder that consists of an encoder and a decoder. The encoder maps the input data to a different -dimensional representation, while the decoder maps the different-dimensional representation back to the input data.
2. We will train the first autoencoder using the input data. The encoder is trained to map the input data to a different dimensional representation, while the decoder is trained to reconstruct the input data from the different-dimensional representation.
3. We will then train the second autoencoder using the embeddings from the first autoencoder
4. This step is repeated 6 times since we had to pre-train 6 different layers, where the output of an autoencoder becomes the input to the next autoencoder
5. Build a new classifier: A new classifier is built according to the dimensions listed in X. The process of fine-tuning the classifier is done on the classification task with the weights of the classifier initialized by weights of the autoencoder.

By using greedy layer-wise pretraining using an autoencoder, the initial weights of the network are learned in an unsupervised manner, which can help the network to converge faster and achieve better performance on the classification task

## Results and discussion

Loss Curves	Accuracy curves	Accuracies	epoch
		Accuracy of 0: 56.62% Accuracy of 1: 31.88% Accuracy of 2: 55.00% Accuracy of 3: 21.88% Accuracy of 4: 31.75% Accuracy of 5: 25.88% Accuracy of 6: 45.12% Accuracy of 7: 31.12% Accuracy of 8: 45.00% Accuracy of 9: 40.50%	18
		Accuracy of 0: 50.62% Accuracy of 1: 30.38% Accuracy of 2: 59.25% Accuracy of 3: 25.00% Accuracy of 4: 42.75% Accuracy of 5: 24.38% Accuracy of 6: 35.62% Accuracy of 7: 28.50% Accuracy of 8: 50.50% Accuracy of 9: 47.62%	10

## t-SNE plot for X[3] in the classifier



### *Discussion*

We can observe in the t-SNE plot that certain classes have overlapping clusters, meaning that the model has learned features in a less distinctive manner. This could explain the low accuracy of classes.