

## Create the model

### **Stem:**

We break each gray-scale image(1, 28, 28) into 7x7 patches (hence, each of size 4x4). That is, we are going to obtain 7x7=49 sub-images out of a single image. we obtain the 7 and 4 in the way that the power of these two dimensions should be result to 28. >>>> ( $4 * 7 = 28$ ) this is for one side of width or height, generally  $49 * 16 = 28 * 28$ . Slicing process done by the reshape method that convert the image of (C, H, W) to (C, Np,  $k^2$ ), C is the number of channels which here we have one channel as the images are gray-scale and Np is the number of patches we have after slicing that equal to  $7 * 7 = 49$ . Also, patches of  $k * k = 4 * 4$  are flattened to a vector of  $k^2$  in the reshape process. Then we use the linear layer to map the patch vector of  $k^2$  to  $d=8$ . Finally the output of stem stage is (batch\_size, Np, d). We have done the stem process following the vision transform ViT process. [2][3][4].

Input images->(batch\_size,C=1,H=28,W=28) ->reshape->

(batch\_size,patch\_num\*patch\_num, patch\_size\*patch\_size)=(batch\_size,7\*7,4\*4)->

linear map->(batch\_size,7\*7,d=8)

### **Backbone:**

We use the N=12 identical blocks for the backbone. Each blocks contains two MLP. First MLP applies on columns of input, maps from size of Np to size of Np.

The input is transposed before each MLP. In conclusion, the output of stem is (batch\_size, Np, d) we transpose it to (batch\_size, d, Np) and feed it to first MLP which defined with the input of size Np. Second MLP applies on rows of input, maps from size of d to size of d, the output of first MLP is (batch\_size, d, Np) we transpose it to (batch\_size, Np, d) and feed it to second MLP which defined with the input of size d.

Each MLP contains two fully connected layers and a non-linear activation function. Hidden\_input is equal to input\_num times to expansion\_factor which have the value of 4. We use the GELU activation function. On top of each MLP a normalization layer of layernorm used. Also we use two dropout layer to avoid overfitting. With dropout rate of 0 in this case. The backbone structure and parameters following the MLP-Mixer architecture[5][6].

stem->(batch\_size, Np=49, d=8)->N blocks [ block [ transpose->(batch\_size,d,Np)->first MLP->(batch\_size,d,Np)->transpose->(batch\_size,Np,d)->second MLP->(batch\_size,Np,d)]]

### **Classifier:**

The output of last block in backbone is (batch\_size,Np,d), after normalization we apply the mean to aggregating the patches before the classification layer. The output is (batch\_size, d) which has the vector d features for each row of sample. Finally feed the features to a linear classification layer with input of d and output of 10 which is number of classes[7].

stem->N blocks[block[MLP1 MLP2]] -> (batch\_size,Np,d) -> normalization -> mean -> (batch\_size, d) -> linear classification(d,10) -> (batch\_size, 10)

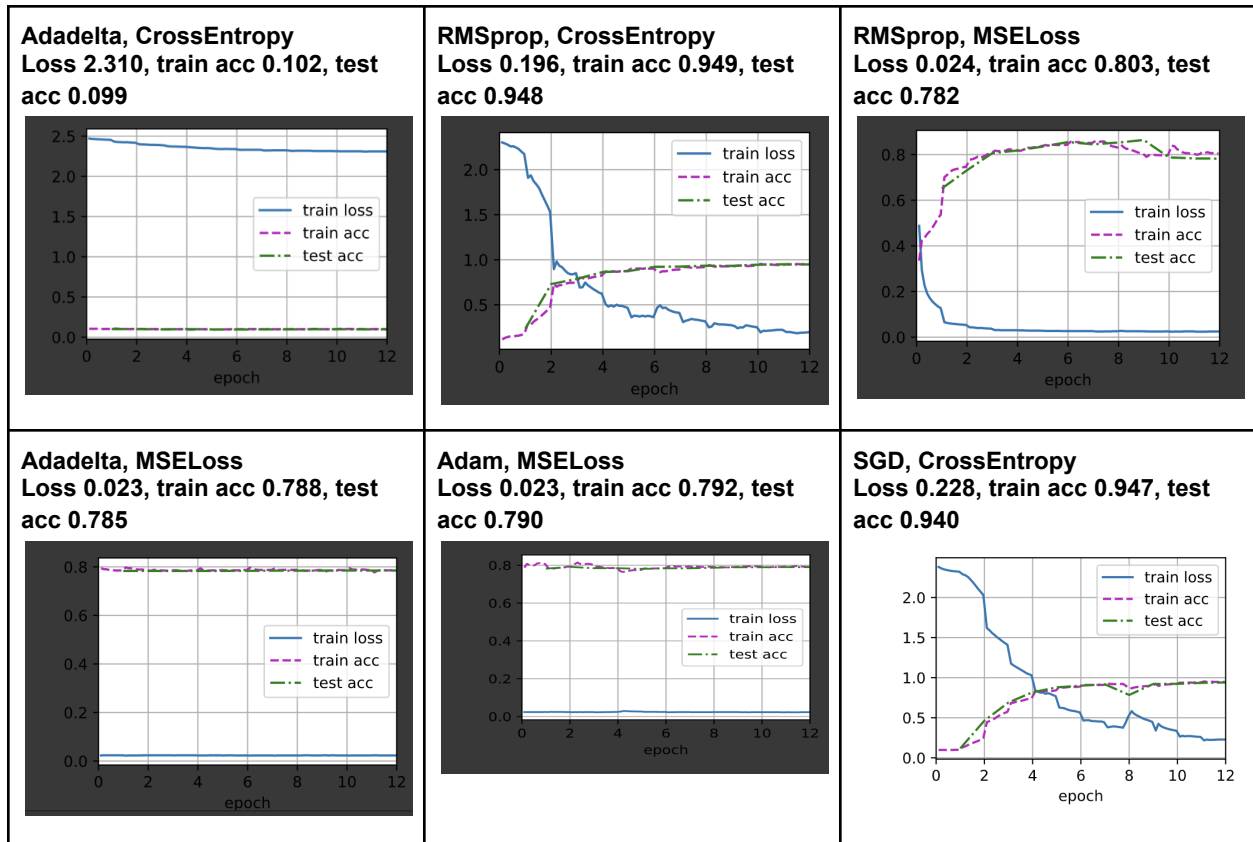
## Training Experiments

### **Loss function and optimiser**

We used different loss function and optimizer combinations such as cross entropy and mean squared error as loss functions and Adam, Adadelta, RMSprop, SGD(wd=0 and wd=0.0005) as

optimisation algorithms, methods chosen for experiment is inspired by[1]. We provide some plots of the accuracy, loss vs epoch for some of the different experiments in the following. The maximum accuracy and minimum Loss achieved when we use Adam optimiser with cross entropy loss function.

Learning\_rate:0.005, epochs:12, batch:128 with default structure parameters mentioned before

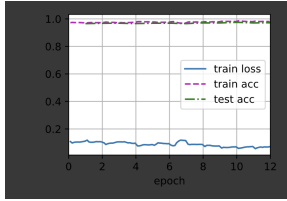
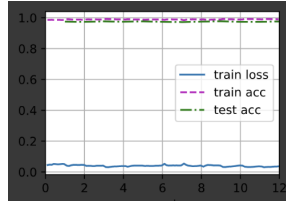


## Weight Initialization

We used different weight initialization methods such as xavier\_uniform\_, xavier\_normal\_, Kaiming\_uniform\_, Kaiming\_normal\_, trunc\_normal\_, orthogonal\_, sparse\_. The best result achieved by xavier\_uniform\_.

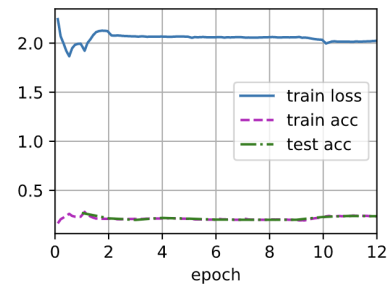
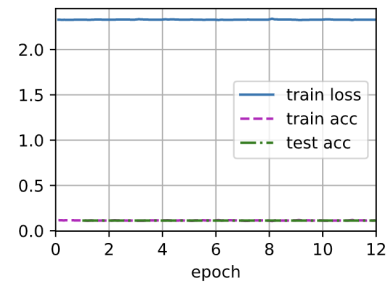
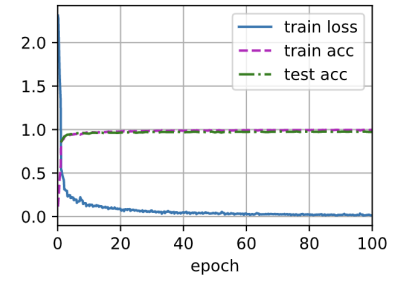
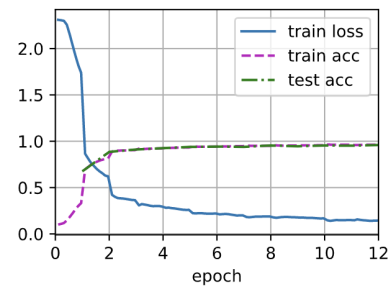
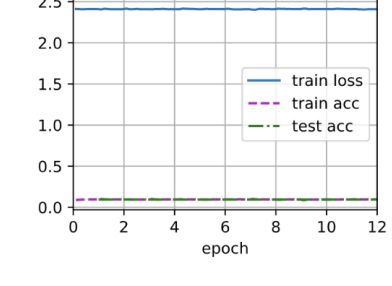
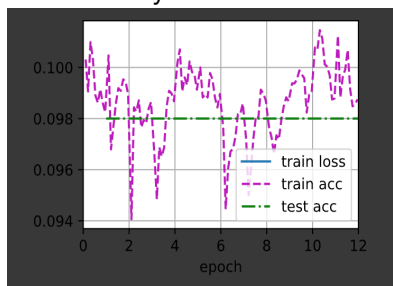
Learning\_rate:0.005, epochs:12, batch:128, Adam, CrossEntropy

<b>xavier_normal_</b> loss 0.112, train acc 0.971, test acc 0.967	<b>Kaiming_normal_</b> train acc 0.987, test acc 0.972	<b>orthogonal_</b> train acc 0.992, test acc 0.977	<b>sparse_</b> train acc 0.992, test acc 0.975
--	---	---	---

<b>Kaiming_uniform_</b> loss 0.071, train acc 0.981, test acc 0.972		<b>Trunc_normal_</b> loss 0.037, train acc 0.990, test acc 0.976	
--	---	---	---

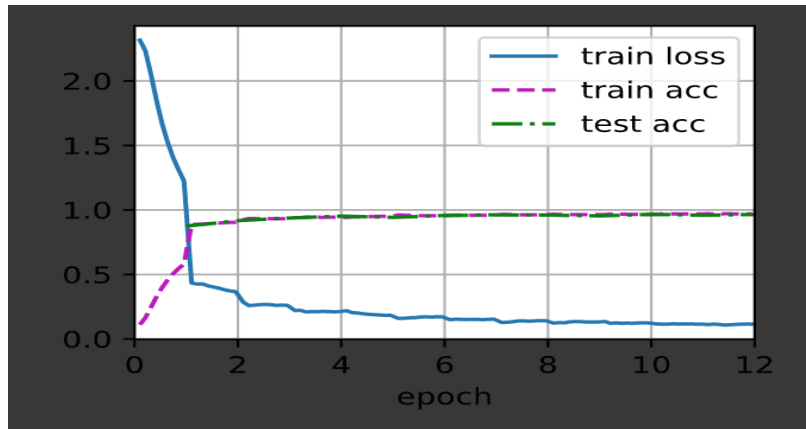
## Hyperparameters :

Experience with Adam, CrossEntropy and xavier\_uniform\_weights

<b>learning rate:0.01, epochs:12, batch:128</b> loss 2.023, train acc 0.234, test acc 0.237	<b>learning rate:0.005, epochs:12, batch:256</b> loss 2.330, train acc 0.113, test acc 0.112	<b>learning rate:0.005, epochs:100, batch:128</b> loss 0.015, train acc 0.996, test acc 0.972
		
<b>learning rate:0.005, epochs:12, batch:128, use Relu activation function in MLP</b> loss 0.145, train acc 0.962, test acc 0.959	<b>learning rate:0.005, epochs:12, batch:128, use N=5 blocks in network structure</b> loss 2.409, train acc 0.096, test acc 0.096	<b>Learning_rate:0.5, epochs:100, batch:128, Adam, CrossEntropy, without normalization layers</b> Test accuracy 0.09
		
		<b>Learning_rate:0.5, epochs:20, batch:128, Adam, CrossEntropy, with normalization layers</b> loss 2.307, train acc 0.103, test acc 0.114

## Results:

Learning\_rate:0.005, epochs:12, batch:128, Adam, CrossEntropy  
loss 0.115, train acc 0.969, test acc 0.965



## References

- [1] Ghosh, Arkadip, Aishwarjyamoy Mukherjee, and Chinmoy Ghosh. "Simplistic Deep Learning for Japanese Handwritten Digit Recognition." International Conference on Innovation in Modern Science and Technology. Springer, Cham, 2019.
- [2] Yuan, Li, et al. "Tokens-to-token vit: Training vision transformers from scratch on imagenet." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021.
- [3] [towardsdatascience.Slicing images into overlapping patches at runtime](#)
- [4] [mlearning-ai.Vision Transformers from Scratch \(PyTorch\): A step-by-step guide](#)
- [5] Tolstikhin, Ilya O., et al. "Mlp-mixer: An all-mlp architecture for vision." Advances in Neural Information Processing Systems 34 (2021): 24261-24272.
- [6] Hou, Qibin, et al. "Vision permutator: A permutable mlp-like architecture for visual recognition." IEEE Transactions on Pattern Analysis and Machine Intelligence (2022).
- [7] Touvron, Hugo, et al. "Resmlp: Feedforward networks for image classification with data-efficient training." arXiv preprint arXiv:2105.03404 (2021).