ECS6P9U/P: NEURAL NETWORKS & DEEP LEARNING

2021/22 – Semester 2

Dr. Yorgos Tzimiropoulos

Spring 2022 - Coursework (Report)

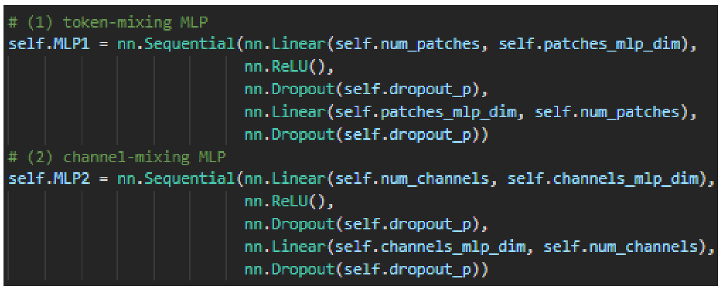
Jordan Akisanya – 200884501

1 Introduction

Google Researchers in their ‘MLP-Mixer: An all-MLP Architecture for Vision’ research paper, proposed the *MLP-Mixer* architecture which is based exclusively on multi-layer perceptrons (MLPs). The model’s high performance and simplicity is what makes it a viable alternative to existing popular CNN architectures. My implementation of the model performs well on the Fashion MNIST dataset, achieving an 89.17% top-1 validation accuracy.

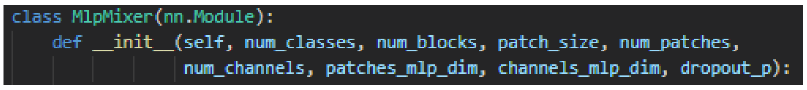
2 Model Architecture

My model code closely follows the FLAX code published in the research paper in the sense that I created the same two classes *MixerBlock* and *MlpMixer*. Where my model differs is by (1) instantiating the token-mixing and channel-mixing MLPs in the MixerBlock itself rather than instantiating them in their own class,



```
# (1) token-mixing MLP
self.MLP1 = nn.Sequential(nn.Linear(self.num_patches, self.patches_mlp_dim),
                          nn.ReLU(),
                          nn.Dropout(self.dropout_p),
                          nn.Linear(self.patches_mlp_dim, self.num_patches),
                          nn.Dropout(self.dropout_p))
# (2) channel-mixing MLP
self.MLP2 = nn.Sequential(nn.Linear(self.num_channels, self.channels_mlp_dim),
                          nn.ReLU(),
                          nn.Dropout(self.dropout_p),
                          nn.Linear(self.channels_mlp_dim, self.num_channels),
                          nn.Dropout(self.dropout_p))
```

and (2) adding additional instance attributes to the *MlpMixer* class.



```
class MlpMixer(nn.Module):
    def __init__(self, num_classes, num_blocks, patch_size, num_patches,
                 num_channels, patches_mlp_dim, channels_mlp_dim, dropout_p):
```

3 Experiments

3.1 Model Configuration

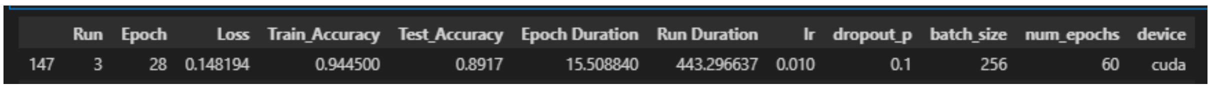
I trained the MlpMixer on the configuration {**num_classes: 10, num_blocks: 8, patch_size: (4,4), num_patches: 49, num_channels: 16, patches_mlp_dim: 1024, channels_mlp_dim: 512**}. The choice of smaller patch size was random but in line with the non-overlapping patch constraint. My rationale for choosing an 8-block architecture was due to the addition of skip-connections in the MixerBlock’s forward method. The skip-connections prevent the gradients from vanishing as the model backpropagates through a greater number of layers.

3.2 Parameter Tuning

When training the model, I varied the learning rate {0.01, 0.001}, dropout {0.0, 0.1} and batch size {256, 512}. I opted for a fixed number of epochs {60} that I believed would be sufficient in allowing for the model's test accuracy to reach its top.

3.3 Main Results

Fig. (1), (2) and (3) display the loss, test accuracy and train accuracy curves respectively. They show minimal variance between the different sets of parameters. The run which achieved the best test accuracy **(89.17%)** used the following parameter set: {lr=0.01, dropout_p=0.1, batch_size=256, num_epochs=60}.



```
     Run  Epoch      Loss  Train_Accuracy  Test_Accuracy  Epoch Duration  Run Duration     lr  dropout_p  batch_size  num_epochs  device
147    3     28  0.148194        0.944500         0.8917       15.508840    443.296637  0.010        0.1         256          60    cuda
```

Fig. (1) – Loss

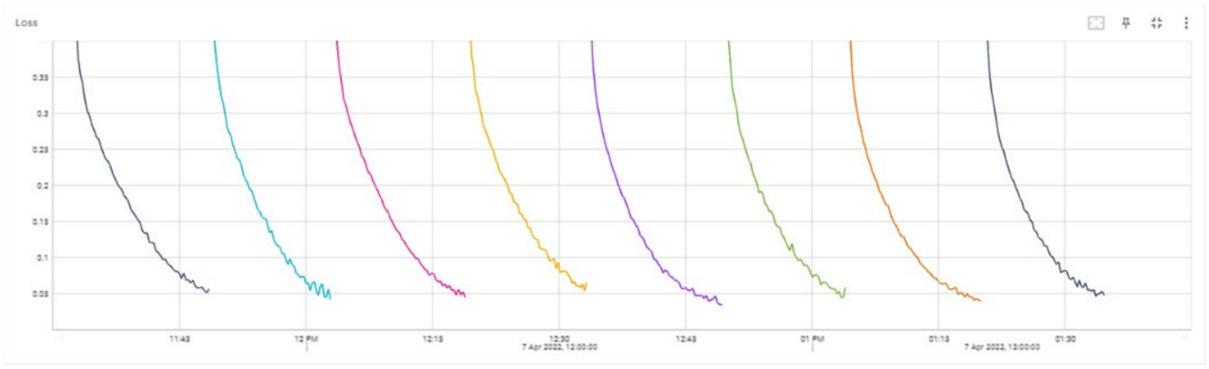


Fig. (2) – Test Accuracy

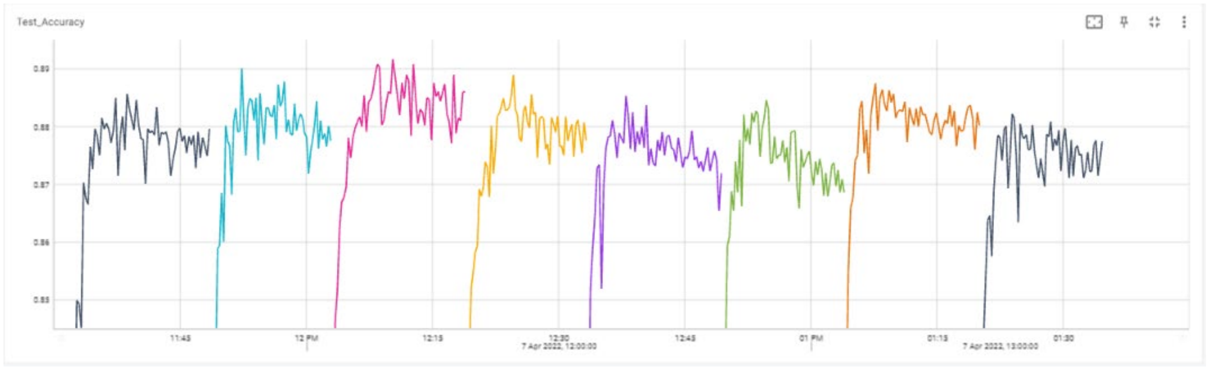
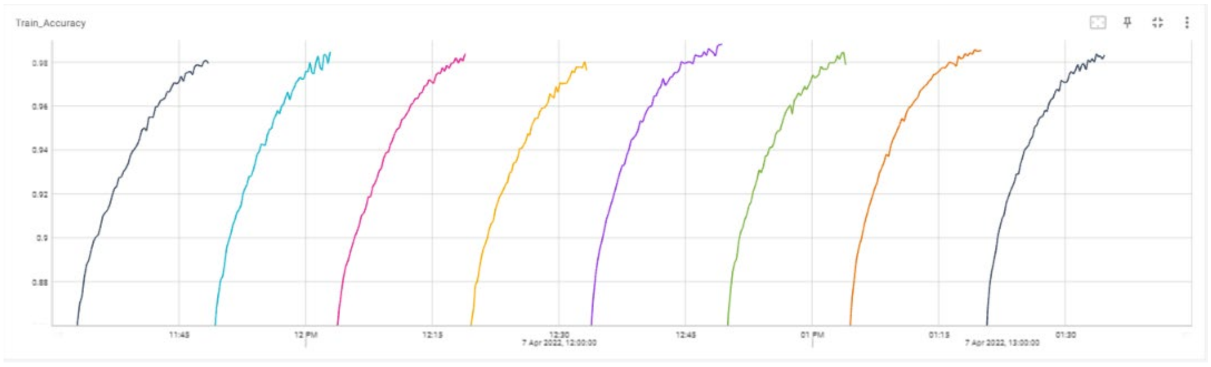


Fig. (3) – Train Accuracy


Train_Accuracy
11:45
12 PM
12:15
12:30
7 Apr 2022, 12:00:00
12:45
01 PM
01:15
7 Apr 2022, 13:00:00
01:30