Assignment 7 – Proof and details regarding the models implemented

1 First model – Fully connected

This is the model I firstly designed for training on CIFAR10 transformations task. It decreases the loss according to Fig. 1 (the figure provides loss scores until stopping criterion is fulfilled) and achieves over 98% accuracy provided that *torch.nn.CosineSimilarity()* with a treshold of 0.95 is used for measuring the similarity score.

1.1 Model's architecture

The following layers have been created for this model:

```
Model(
  (layers): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=3072, out_features=128, bias=True)
    (2): Linear(in_features=128, out_features=784, bias=True)
    (3): Sigmoid()
  )
  (loss): MSELoss()
)
```

1.2 Loss function

MSELoss() (Mean Squared Error loss) was used for training this model, the reason behind this choice residing in the need to measure pixelwise similarity in order to appreciate the distance between the current generated image and the corresponding ground truth. Since this is not a classification task but a generative one, MSELoss is suitable for penalizing differences between the current stage and the target.

1.3 Early stopping criterion

Early stopping is useful for overfitting and waste of resources prevention. In addition, its main beneficial effect is a significant improvement in generalization accuracy and time performance, given

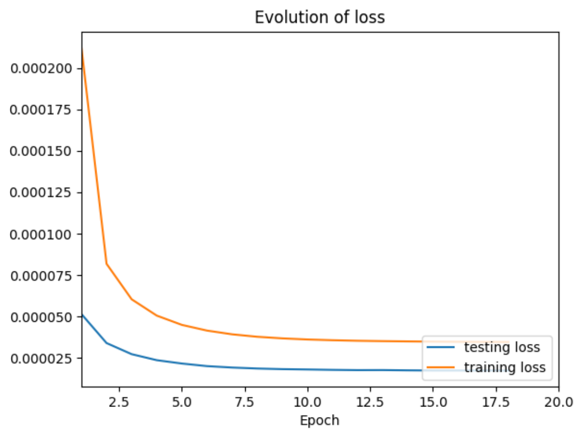


Figure 1: Proof of loss decreasing while training the fully connected model.

the two applications already mentioned. For training the current model, early stopping criterion based on validation loss was used, so as to stop training further when empirically determined minimum difference of 0.000001 is not reached within the last 10 epochs. One alternative variant for this criterion would be to consider the difference between the loss recorded 10 epochs ago and the minimum within the last 10 epochs since then with respect to the specified treshold.

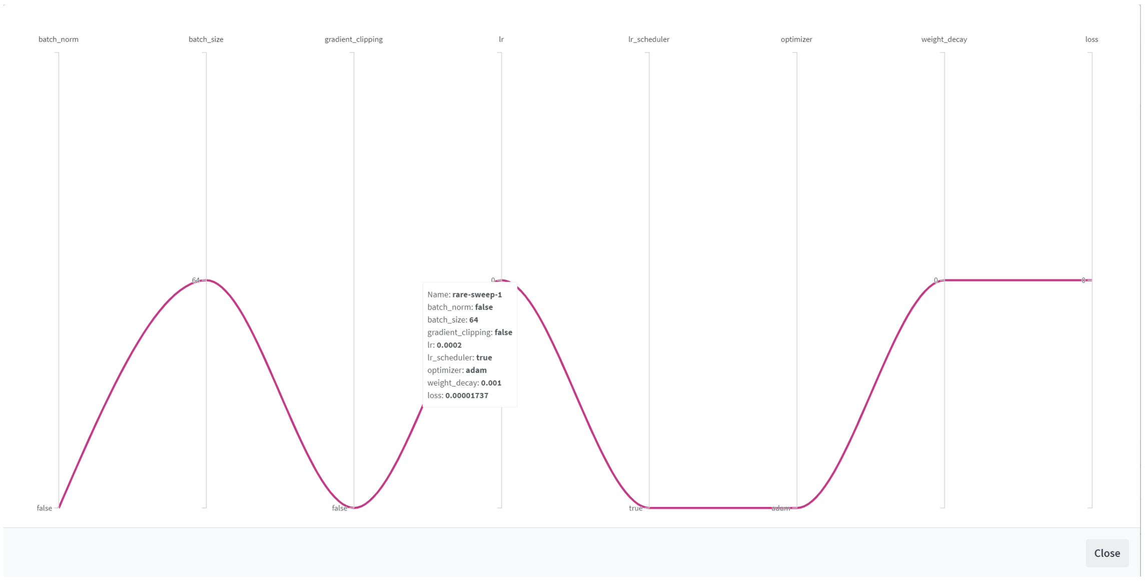
1.4 Inference results

1.4.1 Runtime performance

In terms of runtime performance on GPU, an average of 0.139562 seconds was computed for the inference performance and an average of 0.754336 seconds was observed for the sequencing CPU transformations performance. To ensure higher reliability for the time metrics, the mentioned averages were obtained after 30 runs of the *test_inference_time* function. Transferring the model to CPU resulted in comparable time performance, as the observed average was 0.13146 seconds for the neural network model's inference.

1.4.2 Accuracy on inference

Figure 2: Comparison between ground-truth images (left) and neural network output images (right)



Name: **rare-sweep-1**
batch_norm: **false**
batch_size: **64**
gradient_clipping: **false**
lr: **0.0002**
lr_scheduler: **true**
optimizer: **adam**
weight_decay: **0.001**
loss: **0.00001737**

Figure 3: Wandb logs for the first model.

2 Second model – CNN

I also experienced with a model composed mainly of convolutional layers for the CIFAR10 transformation task. I found inspiration for this model in the paper "Learning Based Image Transformation Using Convolutional Neural Networks", which can be accessed at the following address: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8456517`. As far as the model's architecture is concerned, the resizing task is equally shared between the four convolutional layers, while the grayscale and the flipping transformation are provided by the functioning of the network as a whole. A comparison between the performance of the MSE loss function and the perceptual loss function, in terms of accuracy, is provided within Section 2.2.2, pointing out that perceptual loss is much stronger as long as overall structure and color content similarity are concerned (and not pixelwise differences). In terms of runtime performance, this model is clearly slower than the first model presented, no matter perceptual loss is added or only the base convolutional structure is preserved.

2.1 Model's architecture

The base architecture of the model comprises the following features:

```
Model(
  (layers): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(1, 1), padding=(1,
      1))
    (1): ReLU()
    (2): Conv2d(64, 64, kernel_size=(4, 4), stride=(1, 1), padding=(1,
      1))
    (3): ReLU()
    (4): Dropout(p=0.2, inplace=False)
    (5): Conv2d(64, 64, kernel_size=(4, 4), stride=(1, 1), padding=(1,
      1))
    (6): ReLU()
    (7): Dropout(p=0.2, inplace=False)
    (8): Conv2d(64, 1, kernel_size=(4, 4), stride=(1, 1), padding=(1,
      1))
  )
  (loss): MSELoss()
)
```

For the perceptual loss computing stage, the VGG16 architecture is added to the base architecture presented above, while keeping its pretrained weights fixed.

2.2 Loss function and inference results

Perceptual loss stands out as the new concept among loss functions being used up to this point. Perceptual loss fixes the inconsistency with respect to plausible natural perception encountered in the case of MSE loss, while using the pretrained weights of the VGG16 model. The VGG16 architecture is added to the base convolutional structure and is used exclusively for the loss computation step, so as to keep its pretrained weights fixed. Therefore, the outputs of the base structure and the ground truth labels are fed into the VGG architecture and the previous MSE loss is added five other differences, corresponding to the 2nd, 7th, 14th, 21st and 28th layers of the VGG architecture (which are all convolutional layers). Before that, the outputs and labels are upsampled and their channels are replicated and normalized accordingly in order to fit the ImageNet input pattern. The formula for the loss function component over a specific layer is:

$$\mathcal{L}_i(x, T_W(x)) = \frac{1}{C_i \cdot W_i \cdot H_i} \cdot \sum_{c=1}^{C_i} \sum_{w=1}^{W_i} \sum_{h=1}^{H_i} \left(\phi_i(x)_{c,w,h} - \phi_i(T_W(x))_{c,w,h}\right)^2$$

where ϕ represents the VGG16 network and $\phi(x)$ represents the ith hidden activations when feeding the image x to $\phi(x)$. Taking into account that the ith layer is a convolutional layer in our case, $\phi(x)$ is a feature map of shape $[Ci, Wi, Hi]$, where C_i is the number of output channels for the ith convolutional layer, H_i and W_i are the height and width of the given feature map respectively. x comes from the output of the base structure, while $T_W(x)$ comes from the ground truth image.

2.2.1 Runtime performance

Using only the basic four-layer CNN architecture, an average of 0.587102 seconds was computed for the inference performance on GPU. If perceptual loss and, therefore, VGG architecture is added, one training epoch takes approximately 50 minutes to run on a GeForce RTX 4070 GPU and one testing epoch averages 0.594008 seconds (technically, no computational cost should be added with respect to the cost associated with the base structure, for the inference phase). Transferring the model to CPU results in 2.900498 seconds and 2.635234 seconds performance, respectively. The last two results significantly exceed the value of 0.754336 seconds associated to sequencing the same transforms on CPU.

2.2.2 Accuracy on inference

Due to the amount of time it needs to pass through each epoch and its demonstrative purpose, this model was trained for 5 epochs only.

2.2.3 Accuracy on inference

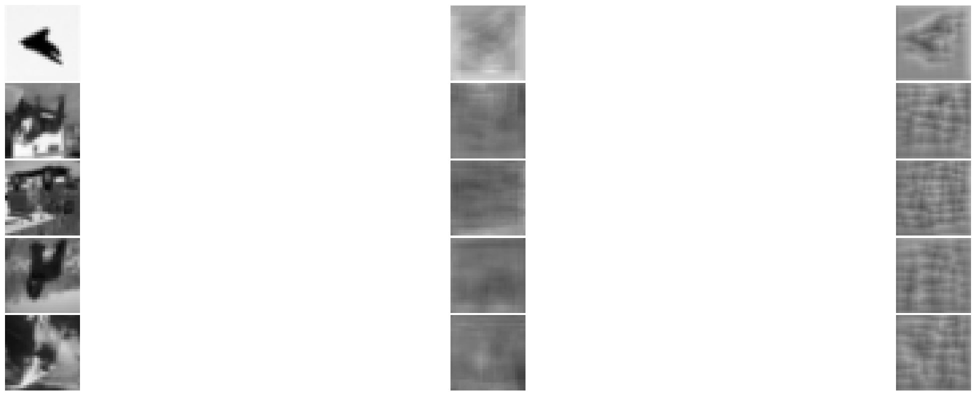
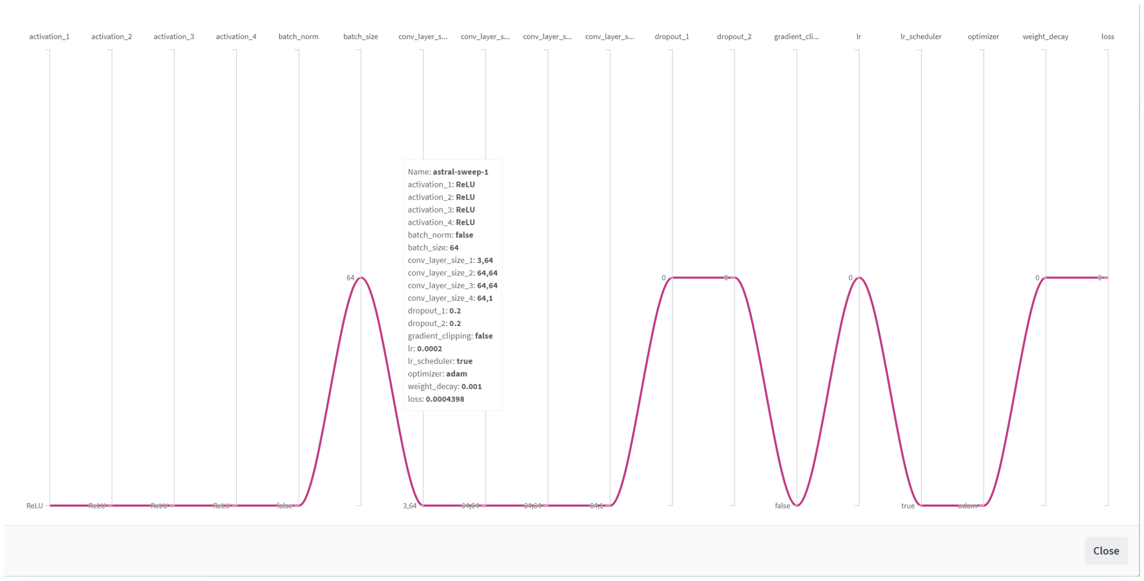


Figure 4: Comparison between ground-truth images (left)
and neural network output images with MSE loss (center) and perceptual loss (right)



```
Name: astral-sweep-1
activation_1: ReLU
activation_2: ReLU
activation_3: ReLU
activation_4: ReLU
batch_norm: false
batch_size: 64
conv_layer_size_1: 3,64
conv_layer_size_2: 64,64
conv_layer_size_3: 64,64
conv_layer_size_4: 64,1
dropout_1: 0.2
dropout_2: 0.2
gradient_clipping: false
lr: 0.0002
lr_scheduler: true
optimizer: adam
weight_decay: 0.001
loss: 0.0004398
```

Figure 5: Wandb logs for the second model.