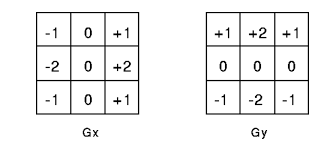
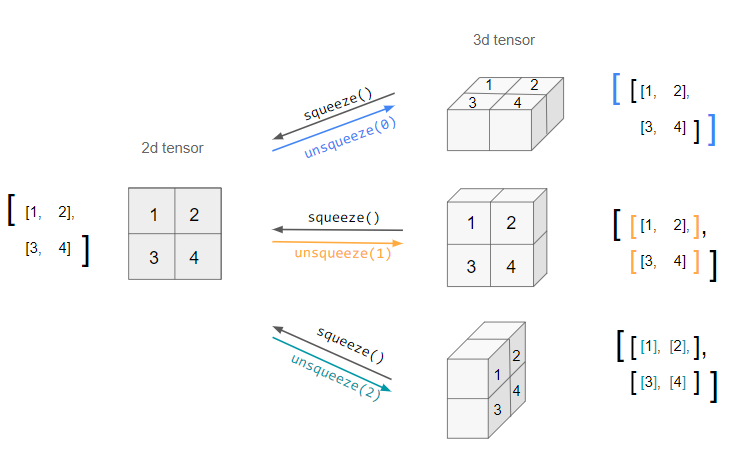
Terminologies

* Nd array – n dimensional array
* Self.linear is a linear equation of form mx+b, where m is weight and b is bias.
* Smooth l1 loss is of from |x-y| till a certain point and then goes to the average of sqrt(x^2-y^2)
* Pyplot.legend function is used to label the graphs
* Edge detection technique



* Think of a batch as a for-loop iterating over one or more samples and making predictions.
* neural networks perform image processing on multi-channeled images. Each channel represents a color, and each pixel consists of three channels. In a color image, there are three channels: red, green, and blue.
* 
* MSELoss- Mean squared Error loss. Mean of the difference between the square of predicted value and the square of the actual value
* Overfitting – Models adapt to the training data and do very well on that data and fluctuations but do very poorly when it comes to new data.
* Epoch- One pass through the entire dataset
* Process
  + Define our model architecture
  + Load our dataset from disk
  + Loop over our epochs and batches
  + Make predictions and compute our loss
  + Properly zero our gradient, perform backpropagation, and update our model parameters.
* Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn, and the amount of computation performed in the network. The pooling layer summarizes the features present in a region of the feature map generated by a convolution layer.
* Res-Net 34 is a 34-layer deep cnn
* Size of feature maps decrease but channels increase as we go deeper as number of kernels increases as we go deeper.
* Steps
  + Auto image processing to output features of all stages of Resnet
  + Load image
  + Preprocess image to tensor
  + Set to eval mode to get features
  + Plot feature map – check how many channels in feature map , batch size dimension is ignored
  + Feature map is normalized
* optimization to adjust the parameters of a model to minimize or maximize a specific objective function
* Cat vs Dog classification
  + Load directory and dataset
  + Convert image into array
  + Process and transform image
  + Load dataset
  + Load to dataloader
  + Augment dataset with trivial augment wide
  + Load augmented dataset
  + Convert datasets to dataloaders
  + Create cnn based image classifier – 3 layers and classifier
  + Forward pass with single image to test model
  + Model in train mode and set train loss and train accuracy values
  + Loop through data
  + Forward pass
  + Calculate loss
  + Optimizer zero grad
  + Loss backward
  + Optimizer step
  + Calculate accuracy
  + Calc average loss and accuracy
  + Test step, model in eval mode
  + Forward pass
  + Calculate loss and accuracy
  + Calc average loss and accuracy
  + Train function
  + Plot loss curves
  + Make prediction based on custom image
  + Load custom image and convert to tensors and divide tensors by 255 to get values between 0 and 1
  + Transform custom image by resizing
  + Model in evaluation mode and make prediction with model(image)
  + Convert prediction from logits to probabilities to prediction labels
  + Then print prediction class
  + Then show image
* Data augmentation technique – rotating, resizing etc to an image to increase amount of data in dataset
* Pooling layer is used to downsize a sample



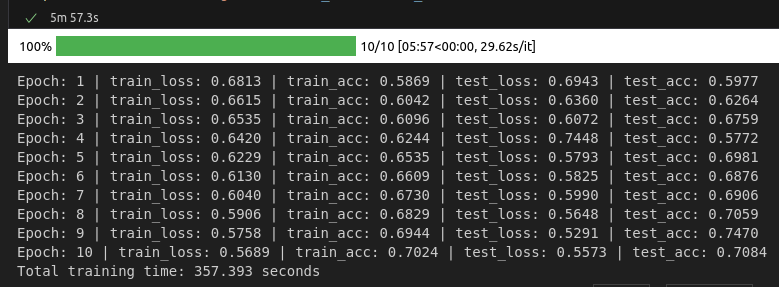
* Fast Gradient Sign Attack - adjusts the input data to maximize the loss based on the same backpropagated gradients. Epsilon increases but the test accuracy decreases
* Dropout is simply a way to reduce dependencies in the Neural Network structure. It encourages each neuron to form its own individual representation of the input data
* Global average pooling is used to to reduce parameters
* ADAM(Adaptive Moment Estimation)- iterative optimization algorithm used to minimize loss function during training
* SGD(Stochastic Gradient Descent)- algorithm to fund the model parameters that correspond to the best fit between predicted and actual outputs
* Activation Layer - function that calculates the output of the node based on its individual inputs and their weights

Function Definitions

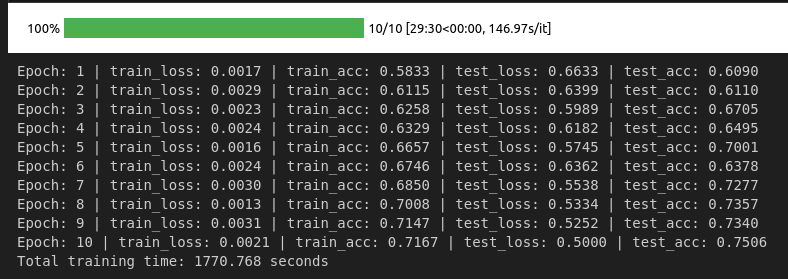
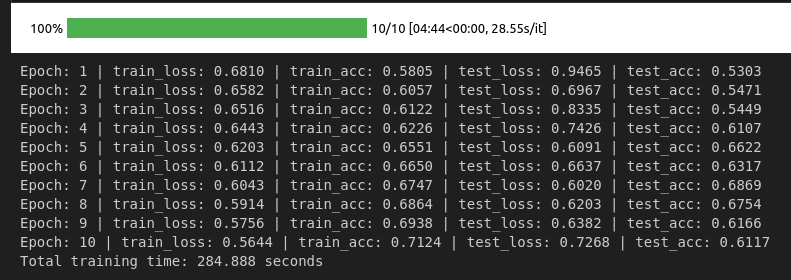
|  |  |
| --- | --- |
| Function name/ Module name | Uses |
| (PIL)- Image.open() | Open image |
| (Numpy)- asarray() | Turn image into array |
| (Matplotlib)- imshow() | Plots image in array form |
| (Matplotlip)- axis(False) | Hides the axis value in a graph |
| (torchvision)- transforms.compose() | List of transforms to compose |
| (torchvision)- transforms.ToTensor() | Converts to tensor |
| (torchvision)- transforms.Resize (size=(width,height)) | Resize the image |
| (Random)-Random.seed(int) | Generates the given argument always to get the same value all the time |
| (Random)-Random.sample(array, int) | A sample of n number of values |
| (torchvision)- datasets.ImageFolder(root, transform) | Root- Target folder of images  Transform – transform function to be applied |
| (Torch.utils.data)- Dataloader(Dataset, batch\_size, num\_workers, shuffle=bool) | Turns datasets to dataloaders |
| next(iter(array)) | Used to get the next item from an iterable object |
| (torchvision)- transforms.TrivialAugmentWide() | Used to augment data. Or to edit the data in different way (crop, zoom, rotate etc) to create more elements to be trained on |
| (Torch.nn)- nn.sequential([]) | Chains outputs to inputs along consequent modules. Ie, thr output of first line will be the input of the second and so on. |
| (torch)- torch.inference\_mode() | One of the several mechanisms that can enable or disable gradients locally |
| (Torch.nn)- nn.CrossEntropyLoss() | Loss is measured as probability between 0 and 1 (0 being perfect model) |
| (torch)-torch.softmax() | Converts vectors of real numbers into a probability distribution |
| (torch)-torch.argmax() | To find the global maximum |

* T-SNE = Stochastic neighbour embedding-

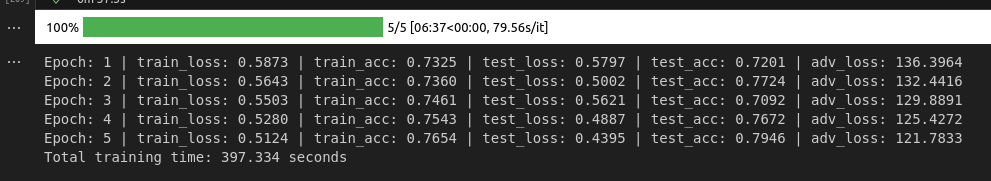
w normal settings



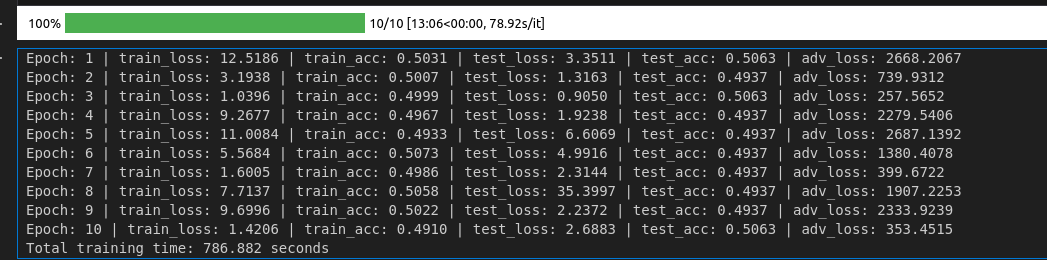
w new test set and old code

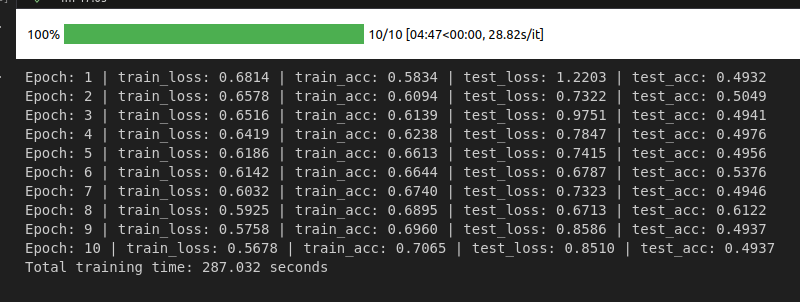
w adverserial style code in train step

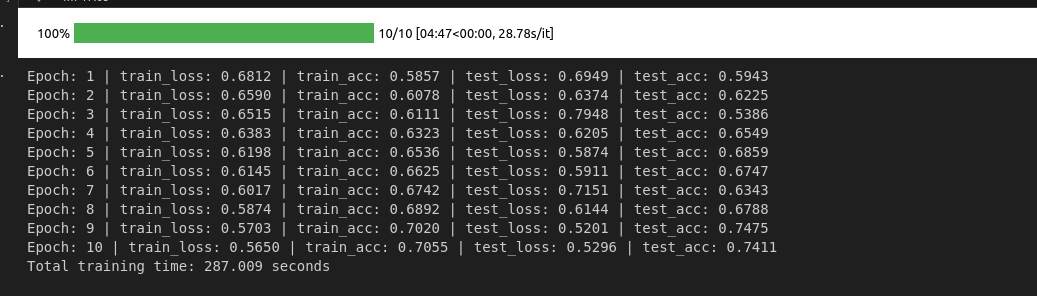
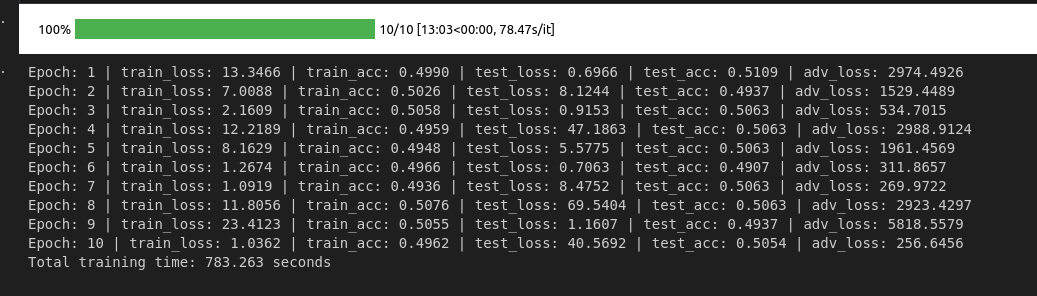
x adverserial style loss adjusted and learning rate changed to 1.0

* PCA tries to reduce dimensionality by maximizing variance in the data while t-SNE tries to do the same by keeping similar data points together (and dissimilar data points apart) in both higher and lower dimensions.
* Assuming X is your tensor with shape (batch\_size, channels, height, width)
* mean = torch.mean(X, dim=(0, 2, 3))

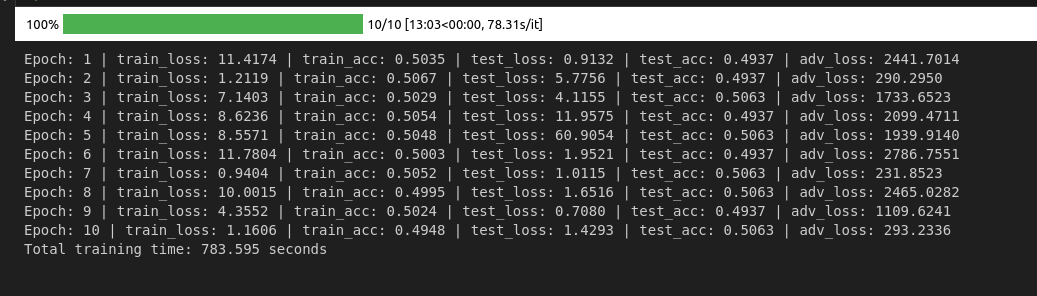
new augmented test datasets with advstyle(lr =1.0)

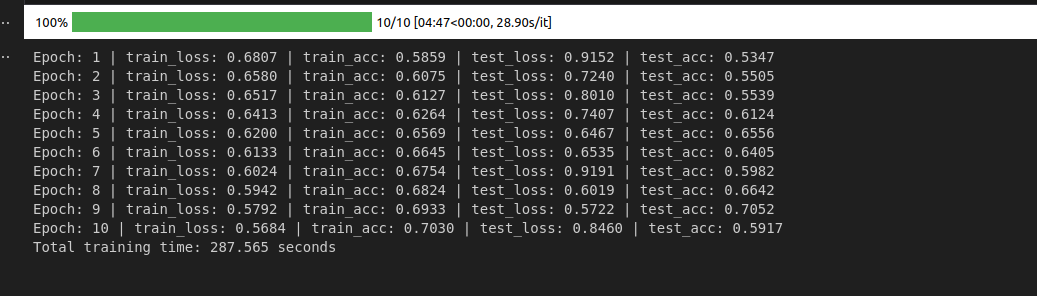
new augmented test datasets without advstyle

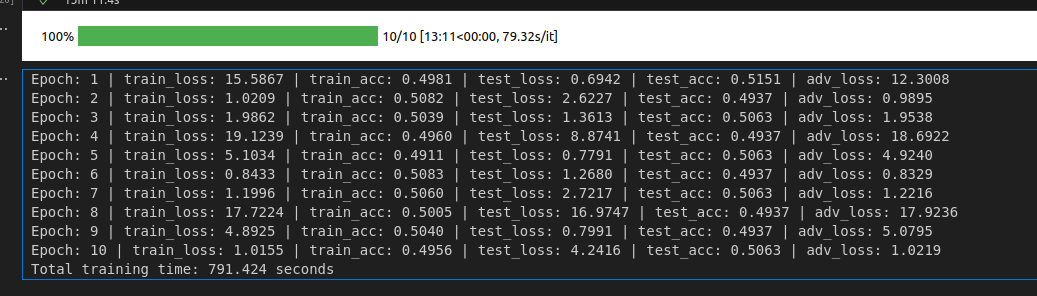
orignal test dataset with advstyle(lr=1.0)

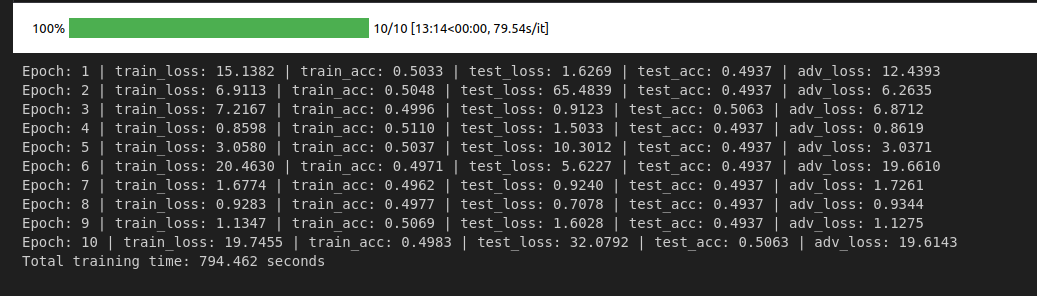
orignal test dataset without advstyle

new dataset with advstyle(lr=1.0)

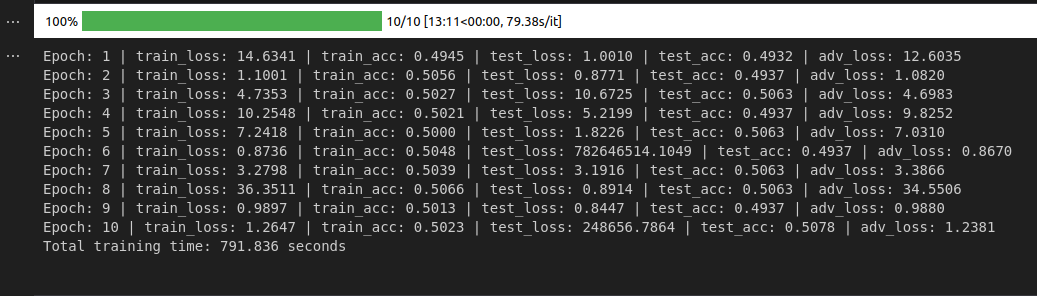
new test dataset without advstyle

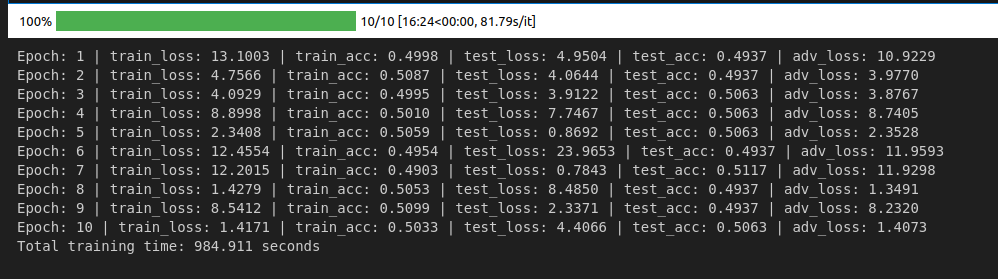
orignal test dataset with advstyle(lr = 0.5)

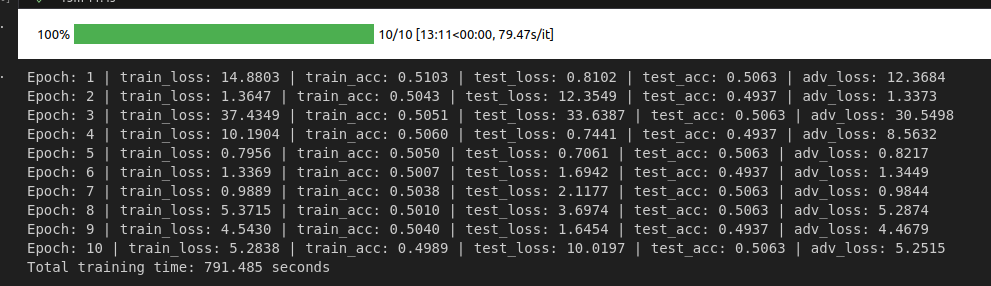
orignal test dataset with advstyle (lr=3.0)



new test set with adv style lr =0.5

new test set wit adv style lr = 3.0

augmented test dataset with advstye lr = 0.5

augmented test dataset with advstyle lr = 3.0

