Batchout results using VGG19 on CIFAR

Date: 17/2/21

Applied modified batchout on VGG19bn model with the help of results from the paper "Feature space perturbation yields more transferable adversarial examples". The paper uses the l2 distance of features of original image and a target image as loss function and then perturbs the original image with this loss function. As per the paper, there are layers which yield better transferable features. In the following figure x-axis is the depth of the network. We can see a layer in the middle gives better results, than the last layers. The y-axis is % of successful attacks and the four figures are in different situations like normal attack, targeted attack.

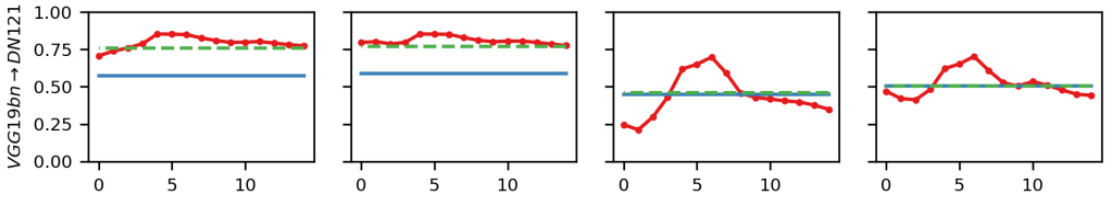


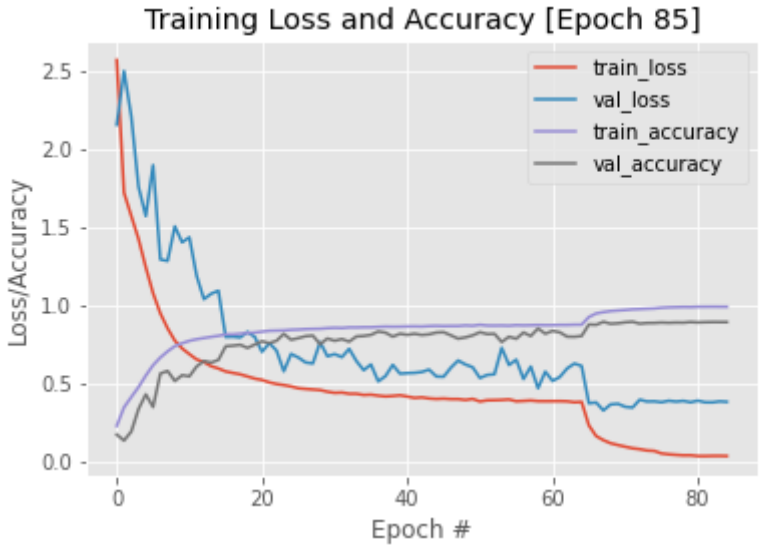
Figure 3: Error, uTR, tSuc, and tTR rates versus depth for multiple transfer scenarios. The top two rows are transfers from a DN121 whitebox model and the bottom two rows are transfers from a VGG19bn whitebox model. Dataset: CIFAR-10.

My batchout experiments included:

Standard training, Batchout on final layer and Batchout on middle layer as per paper

One point to note is in original vgg implementation it was BN -> ReLU. I modified it to ReLU -> BN The value of n used in batchout is 0.2. More on this later on.

1. Standard Training:



Setting: weight_decay: 5e-4. starting lr=1e-2 with drops on epoch 60 and 75.

Accuracy: *89.19%*.

Code and weights can be found at ~/robust/vgg19/

2. Batchout on Final layer

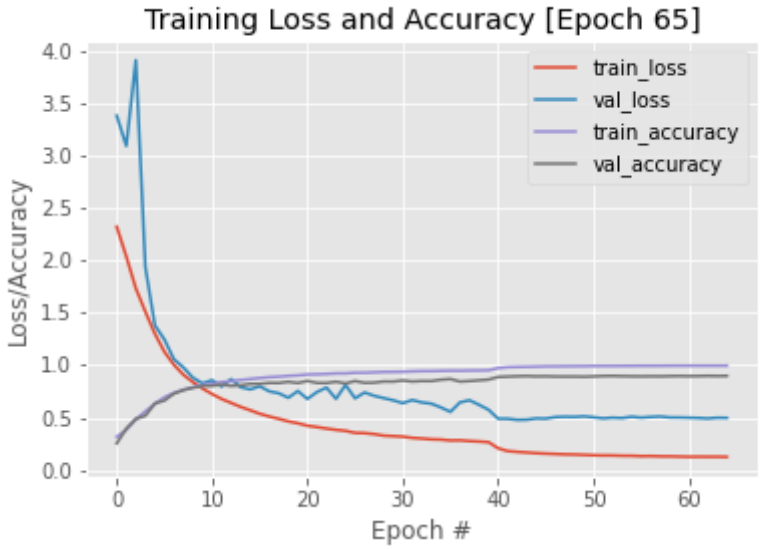
   Setting: weight_decay: 6e-4 (1e-4 was added because I forgot to change code). The training graph will be uploaded later.

Accuracy: *86.98%*

3. Batchout on a middle layer:

   'VGG19' has the following filters:

   [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', **512**, 512, 512, 512, 'M', 512, 512, 512, 512, 'M'] where 'M' denotes max pooling. In the paper mentioned above, the layer marked **bold** gave the best results and batchout was applied after this layer. (The first two layers were not considered in the paper)

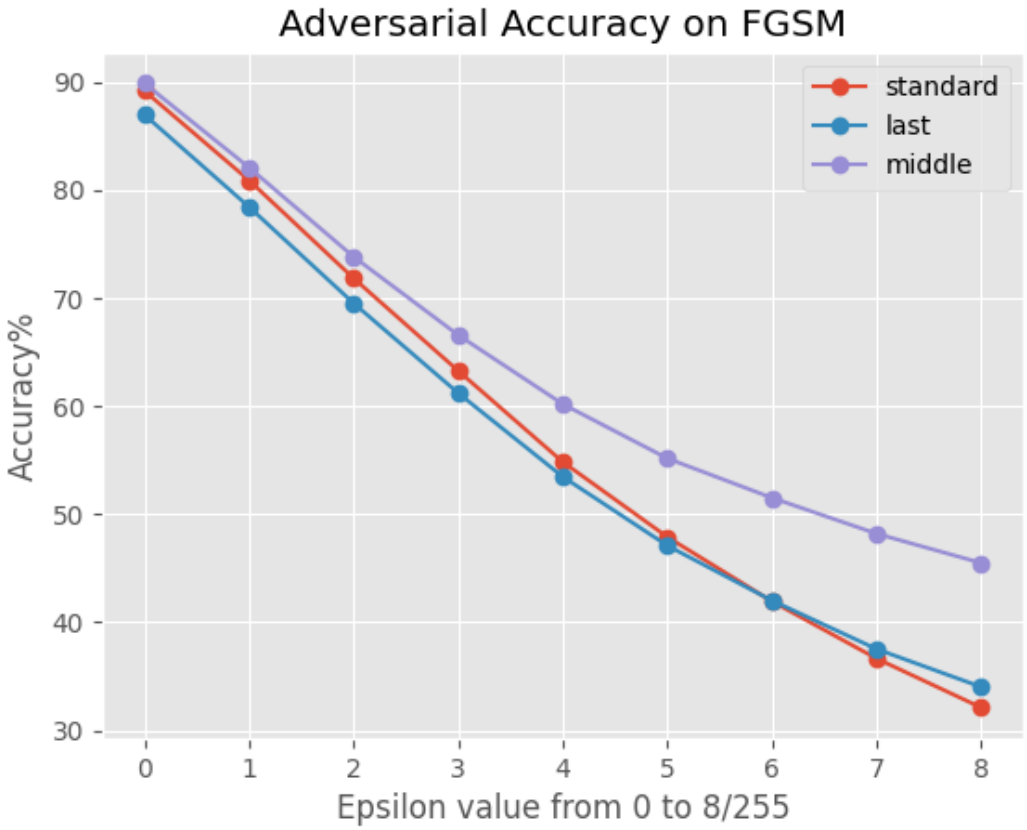


Setting: weight_decay: 6e-4. starting learning rate=1e-2 dropped at 40 and 55 epochs.

Accuracy: *89.89%*

Robust Accuracies:

Applying batchout on a layer which gave good results to the paper mentioned above, gave better robust accuracies as well.



Conclusion: There are certain layers in a model that can give better effect on batchout than the last layers.

Choosing correct values of n in the batchout equation

Batchout update equation: x = original_feature + n * (added_feature - original_feature)

x = (1 - n) * original_feature + n * added_feature

Clearly n cannot be greater than 0.5, otherwise the added_feature is going to dominate giving wrong results.
n = 0 refers to standard training.

Previously, n=0.3 was used on ResNet and it gave also gave good results on batchout. In the batchout paper, n = U[0, 0.15] was frequently used where U is uniform distribution. I don't know why I used such a large value of n=0.3 and thus reduced it to n=0.2 for this experiments on VGG. Further experiments can be done to get a feel of value of n to choose.

Note: Proper comparison of n=0.2 and n=0.3 from this experiment and the previous ones can be made because in the previous experiments there was a 7% better accuracy by batchout trained model than standard model. But in this experiment set, there is 2.3% better accuracy by standard model.

How can we improve BatchOut

1. Proper value of n can be very helpful.
2. Perhaps applying batchout on only one layer is not a good idea. Multiple layers must be used. In case this is done, it must be ensured that for each layer, the same datapoint is added, that was added in the previous layers.

The intuition of batchout is that we are perturbing the features of a class in the direction of another class. An experiment can be performed where batchout is only applied to a pair of classes and evaluation can be done with targeted attacks for the chosen class and other classes. Example: In cifar, there are 10 classes. Let us chose bird and deer. We apply batchout only to datapoints of bird and the perturbed feature will always be deer. Intuitively, we are perturbing the features of bird class in direction of deer class. After training, if we perform targeted attack on deer class, the bird class should give better results than the other classes(frog, automobile, dog etc).

AutoAttack

Found this repo where models robustness can be checked. They are maintaining lists of best robust models from different papers. AutoAttack performs 4 attacks and worst robustness value is the model's robustness. No need of any parameter tuning except for epsilon which is set to 8/255. I attacked the above trained models using autoattack at epsilon 8/255.

The model trained with batchout in middle gave: 26.36% accuracies. Standard model gave 23.06% accuracy. Few points:

1. Highest accuracy obtained is around 65% but majority are between 40 to 60%
2. All the models which obtained high accuracies were of the ResNet family surprisingly. Methods such as pre-training, unlabeled data was used.
3. PGD adversarial training is #29 with accuracy of 44%