CSE 676
Project 2 Supplementary Material

Sandesh Kumar Srivastava(sandeshk)

This document contains the material which was not included in the report file submitted.

Implementation details:

ML Model

1. I have implemented all the architectures and the variants using Keras and followed the same pattern throughout. I begin by importing the libraries and defining some of the hyper-parameters like `NUMBER_OF_CLASSES, epochs, batch_size, path_best and path_train` where most of the names are self-explanatory apart from `path_best` which stores the path of best weights seen so far and is used for validation and `path_train` is used to store the optimal weights seen during current training phase. ***For verification of the results, we need to set the "path_best" to the actual weight file path.***
2. This is followed by loading the cifar-10 dataset and performing some preprocessing on it by subtracting mean and dividing by standard deviation.
3. I have used the Inception V2 model which is depicted in Fig 1.
4. Model is compiled using the `Adamax` optimizer.
5. We then start the training portion by instantiating `ModelCheckpoint()` which is used to store the best model in the `path_train` location based on the metric specified in the `monitor.`
6. After training is completed, I am plotting the model accuracy and loss for both training and test data.
7. At the end, the model is used to predict the test data values and calculate the precision, recall and accuracy based on best model weights.

Fig 1: Inception V2 Model used

Regularization strategies(Adversarial Training using FGSM):

1. To generate adversarial examples we use the Fast Gradient Sign Method(FGSM) method which creates adversarial examples by adding an imperceptibly small vector, equal to the sign of the elements of the gradient of the cost function, to the training dataset input *x*

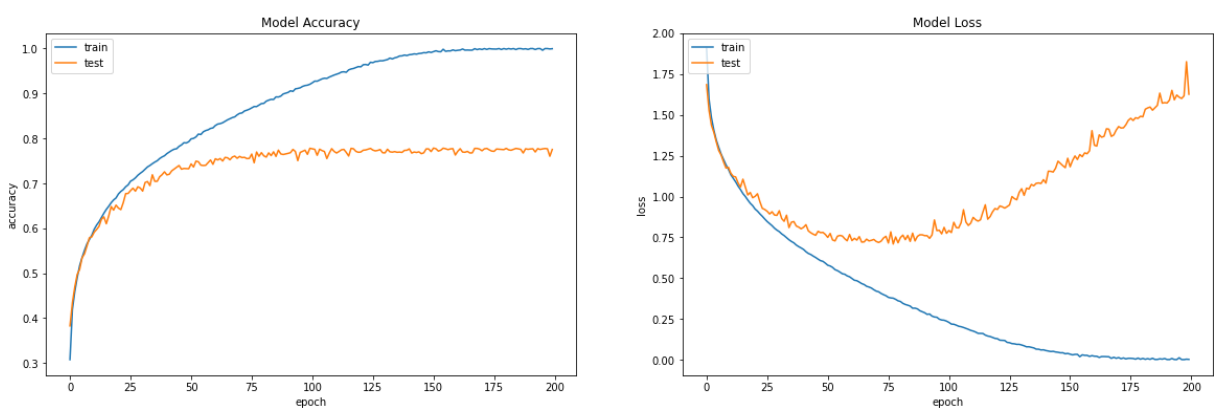
$$x \rightarrow x + \epsilon sign(\nabla_x J(\Theta, x, y))$$

2. For the experiment I have used □ = 0.005 and created 40000 adversarial examples and used it for training the ML model.

Results:

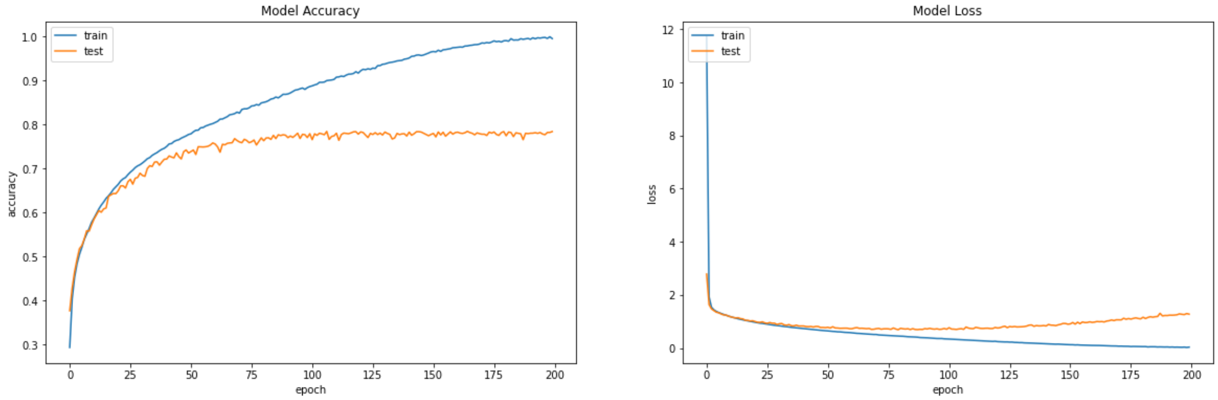
1. I am presenting the graphs obtained during the training phase for all the regularization strategies below:

No Regularization



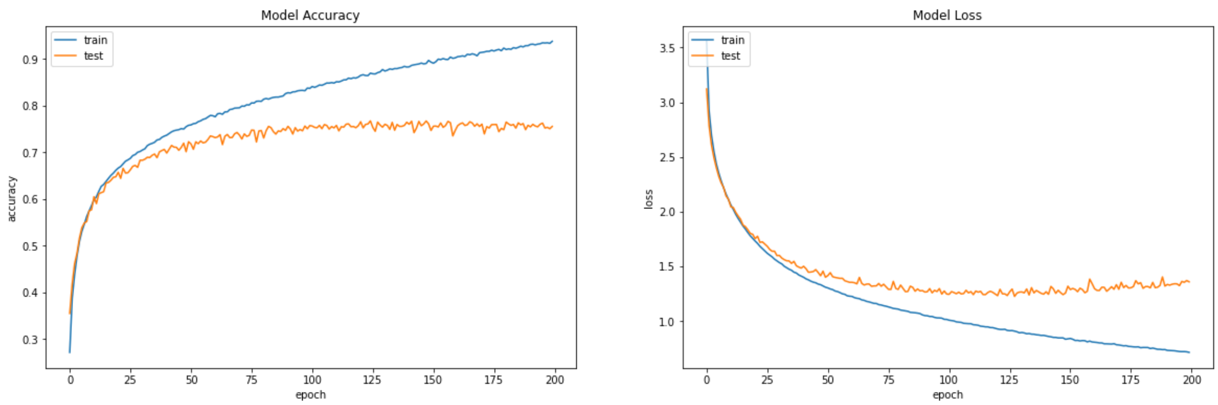
Overfitting can be clearly seen in this case as test data loss keeps on increasing as training data loss decreases and training data accuracy keeps on improving as test data accuracy becomes constant

L1



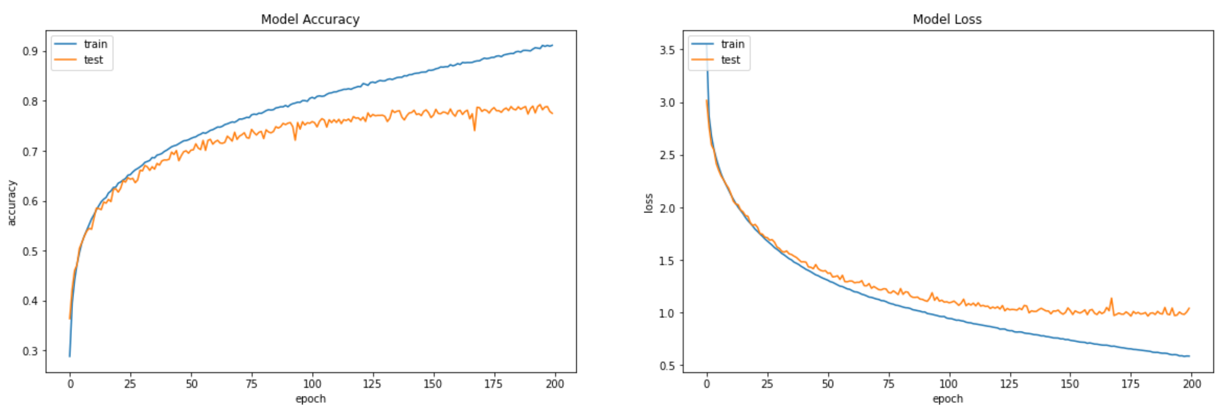
L1 shows improvement as compared to No Regularization case.

L2



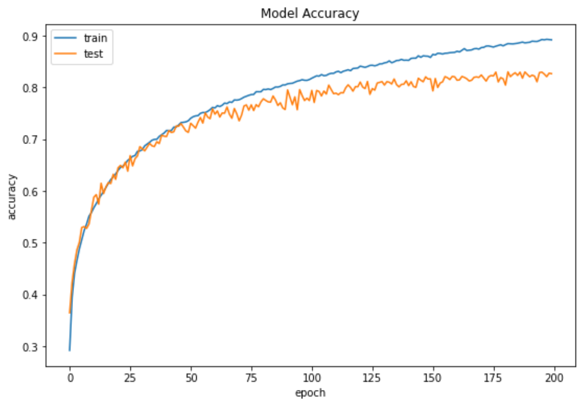
L2 shows improvement as compared to No Regularization case.

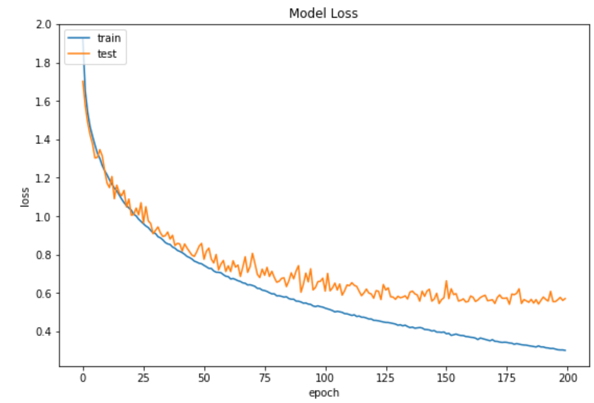
L1L2



L1L2 shows improvement as compared to No Regularization case.

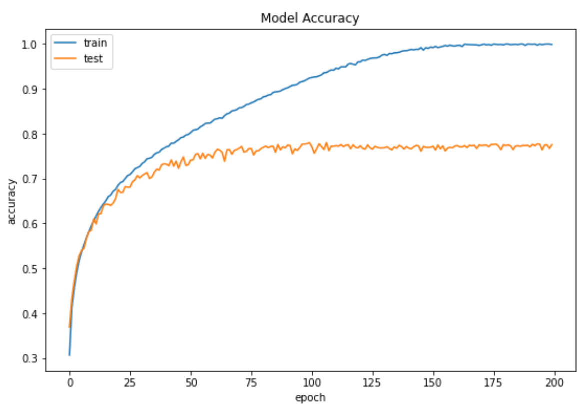
Dataset Augmentation

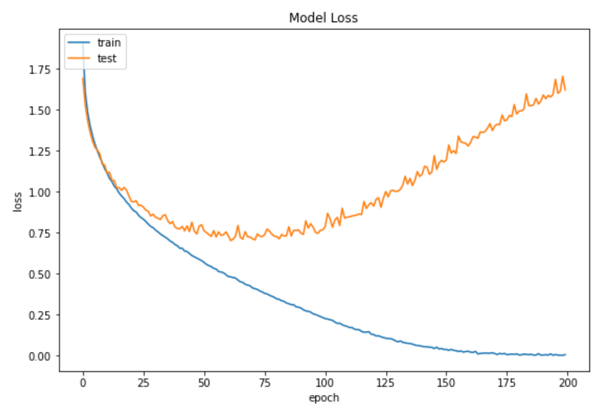




Dataset Augmentation shows improvement as compared to No Regularization case.

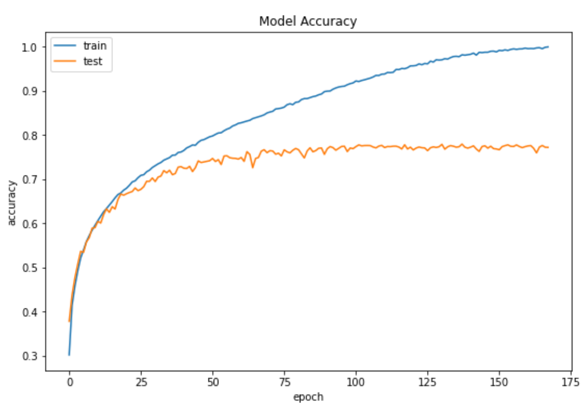
Noise robustness

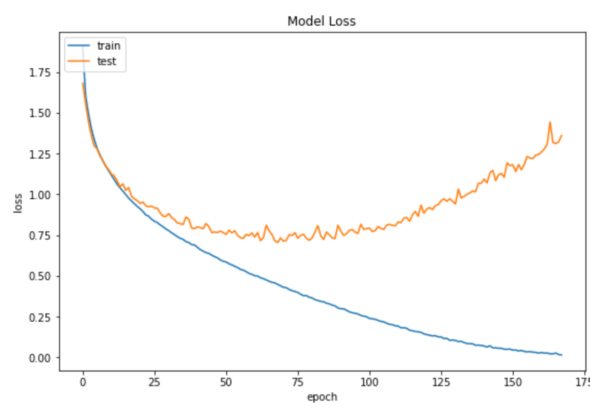




Noise robustness shows slight improvement as compared to No Regularization case.

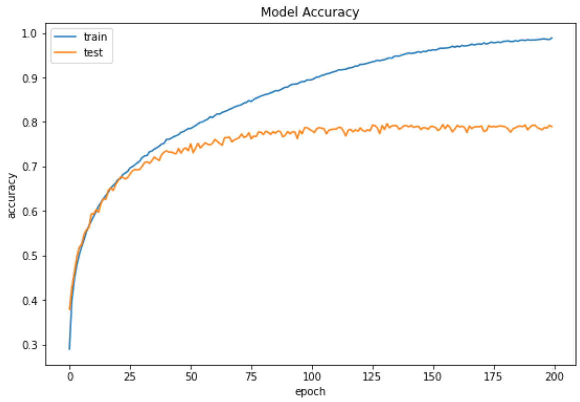
Early Stopping

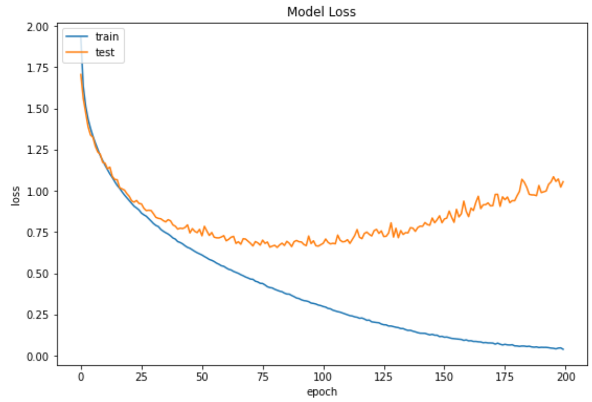




Early Stopping achieves comparable performance as No Regularization case but in fewer epochs.

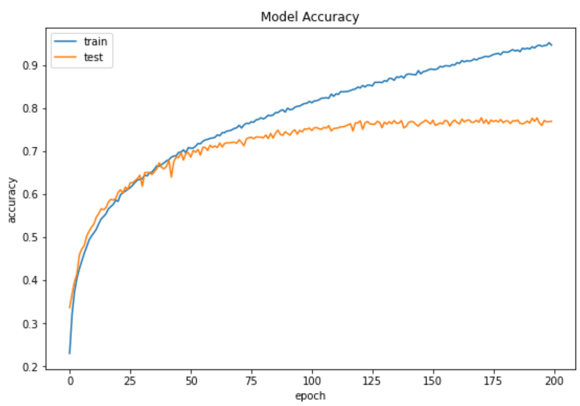
Dropout

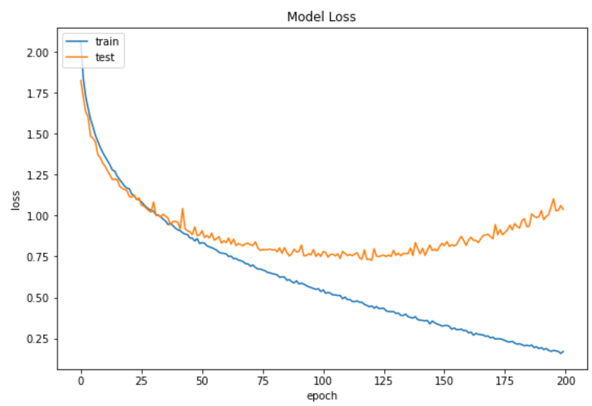




Dropout shows improvement as compared to No Regularization case.

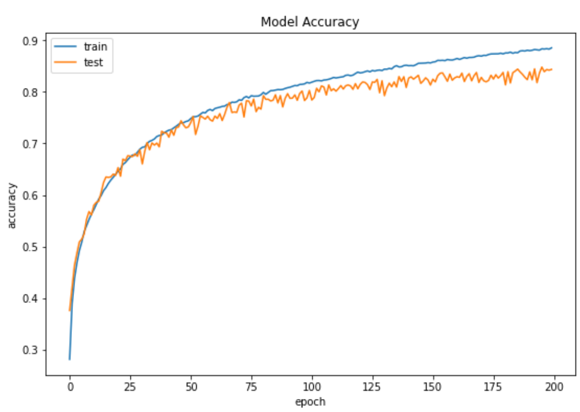
Adversarial Training(FGSM)

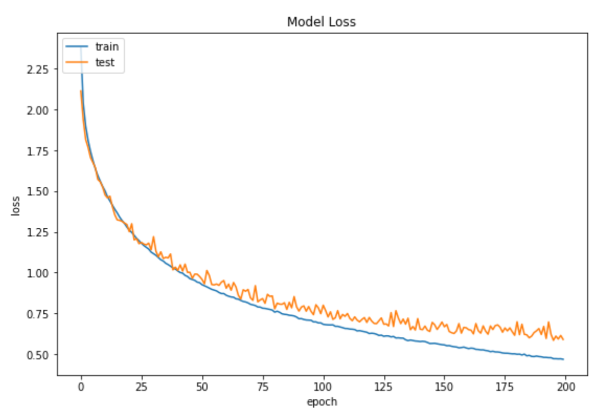




Adversarial Training(FGSM) achieves comparable performance as No Regularization case but the model is trained on adversarial examples and so provides more security.

Combination

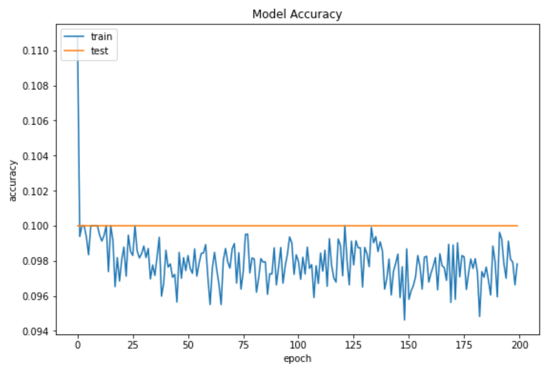


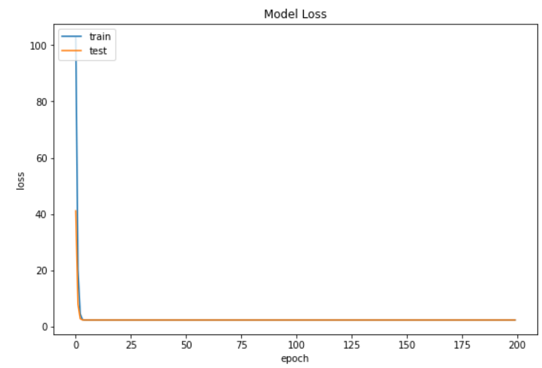


Combination uses L2, Dropout and Dataset Augmentation and provides the best results.

2. Effect of hyperparameter “*kernel_regularizer”* on L2 regularization is presented below:

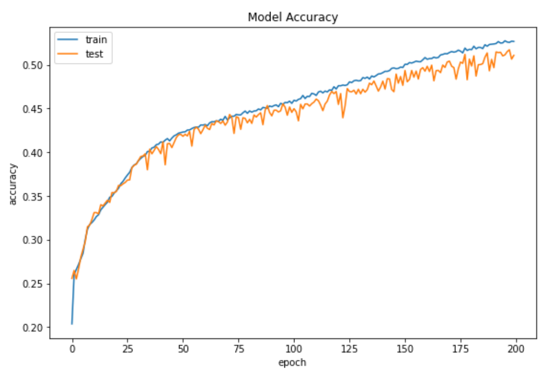
kernel_regularizer = 1e-1

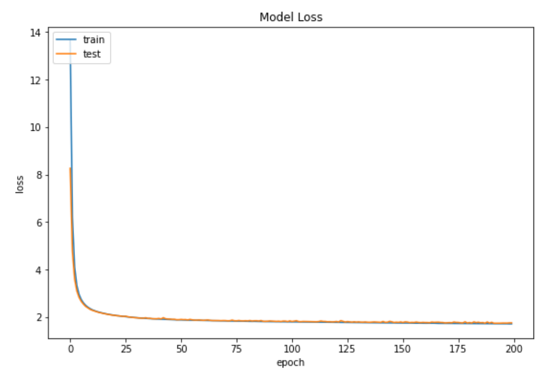




The parameter norm penalty Ω(Θ) is so large that the model is not able to learn efficiently.

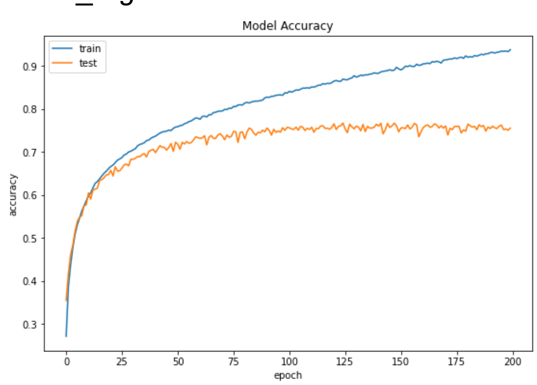
kernel_regularizer = 1e-2

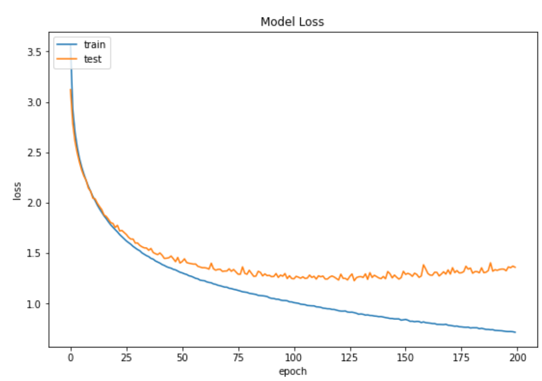




It performs better than previous case but still the norm penalty Ω(Θ) has too much effect on the objective function.

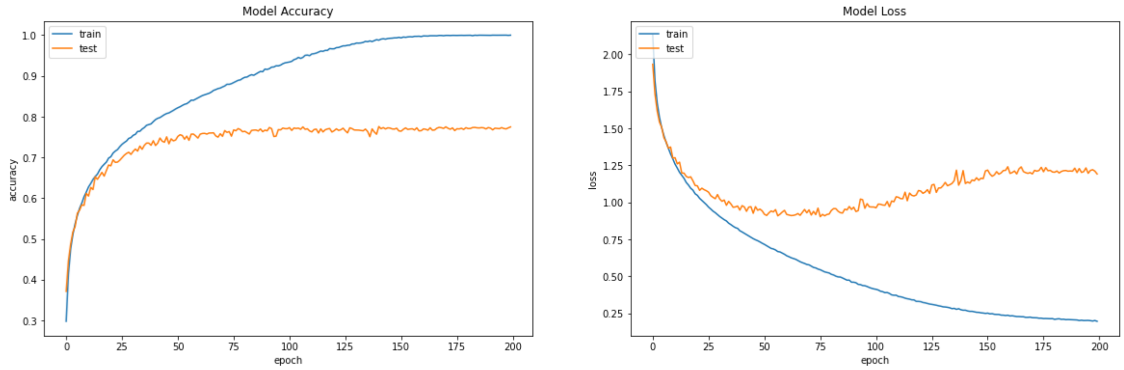
kernel_regularizer = 1e-3





This gives the best performance for the hyperparameter value.

kernel_regularizer = 1e-4



In this case, the value of hyperparameter is so small that it starts reaching towards the "*No Regularization"* case.

References:

[1]. https://www.deeplearningbook.org/contents/regularization.html
[2]. https://keras.io/api/layers/
[3]. https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/
[4]. https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/
[5]. https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/
[6]. https://github.com/EvolvedSquid/tutorials/blob/master/adversarial-attacks-defenses/adversarial-tutorial.ipynb