CSE 676: Deep Learning (Fall 2020)
Project I

Name: Vinita Venktesh Chappar
Email ID: vchappar@buffalo.edu
UB Person ID: 50337016

1 Introduction

There are many convolutional neural network architectures developed for various tasks such as classification and detection of images. In this project, 3 of such architectures are studied and implemented - VGGNet16, ResNet18 and Inception V2. Along with implementations of these architectures, various combinations of regularization techniques and optimizers are experimented. The regularization techniques used are - Batch Normalization and Dropout while optimizers include ADAM and SGD. The observations from these 18 experiments can help us analyse which optimizer works best with which regularization technique. Also, we will be able to identify how regularization helps in achieving higher accuracies than models with no regularization. These observations are discussed in detail in the results section of the report.

2 Networks

2.1 VGGNet16

VGGNet16 is a cnn architecture proposed by K. Simonyan and A. Zisserman in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition"[1]. It was one of the most famous models used in image recognition and it even managed to achieve 92.75% accuracy on the ImageNet dataset [2]. The following figure shows the architecture of VGGNet16:

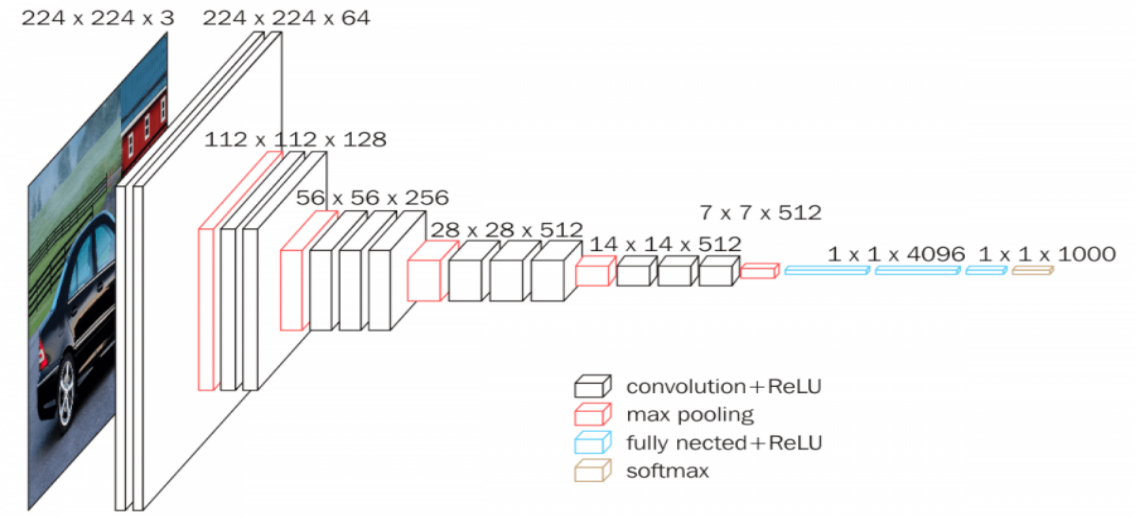


Figure 1: VGGNet16 Architecture

For the ImageNet dataset, they have taken a fixed input size of 224 x 224 RGB image. For CIFAR100 dataset, the input size will be 32 x 32. The images will be passed

through several convolutional layers with different filter sizes: 2 layers of 64 filters, 3 layers of 128, 4 layers of 256 and then 4 layers of 521 filters twice. This shows that the model is very complex and suited for large data-sets. Since we are using CIFAR100 dataset which is comparatively smaller than ImageNet, the model has been modified for this project slightly. In this project, the model is defined as shown:

```
model.add(Conv2D(filters=64, kernel_size=(3, 3),padding='same', activation=tf.keras.layers.LeakyReLU(alpha=0.3), input_shape=input_shape))
model.add(Conv2D(filters=64, kernel_size=(3, 3),padding='same', activation=tf.keras.layers.LeakyReLU(alpha=0.3)))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same', data_format='channels_last'))
```

```
[ ] model.add(Conv2D(filters=128, kernel_size=(3, 3),padding='same', activation=tf.keras.layers.LeakyReLU(alpha=0.3)))
    model.add(Conv2D(filters=128, kernel_size=(3, 3),padding='same', activation=tf.keras.layers.LeakyReLU(alpha=0.3)))
    # model.add(Conv2D(filters=128, kernel_size=(3, 3),padding='same', activation='elu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same', data_format='channels_last'))
```

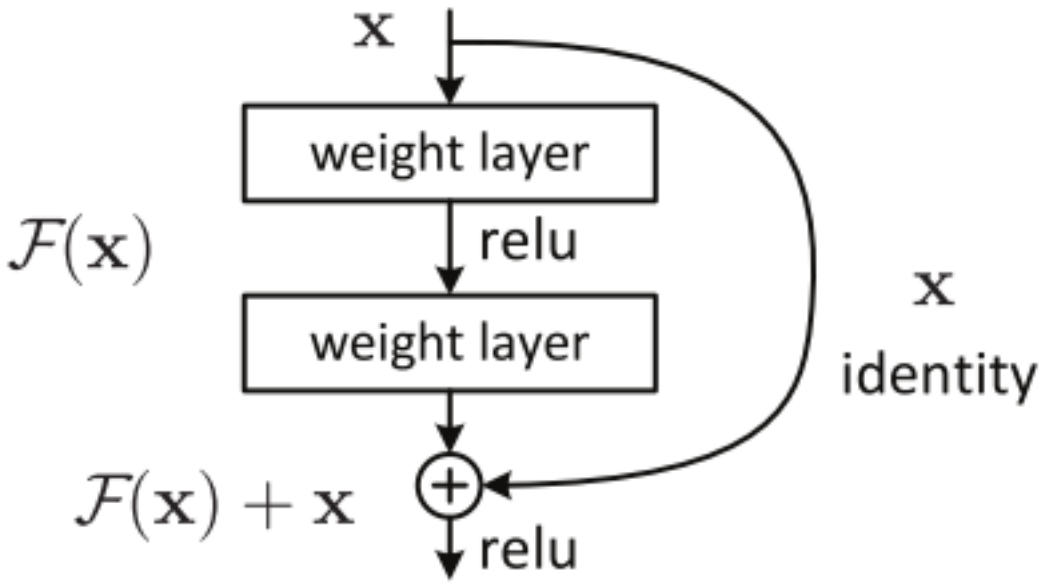
```
[ ] model.add(Conv2D(filters=256, kernel_size=(3, 3),padding='same', activation=tf.keras.layers.LeakyReLU(alpha=0.3)))
    model.add(Conv2D(filters=256, kernel_size=(3, 3),padding='same', activation=tf.keras.layers.LeakyReLU(alpha=0.3)))
    model.add(Conv2D(filters=256, kernel_size=(3, 3),padding='same', activation=tf.keras.layers.LeakyReLU(alpha=0.3)))
    model.add(Conv2D(filters=256, kernel_size=(3, 3),padding='same', activation=tf.keras.layers.LeakyReLU(alpha=0.3)))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same', data_format='channels_last'))
```

Figure 2: VGGNet16 Refined Architecture

The model is made up of 2 cnn layers of filter size 64, then 2 layers of 128 and 4 layers of 256 filters. The discussion regarding why this architecture works for CIFAR100 is mentioned in the results section of the report.
The final layers are- 2 fully connected layers of 4096 channels each and final softmax layer which contains 100 channels for 100-classes of classification of CIFAR-100 dataset.

2.2 ResNet18

The Gradient vanishing problem has been the most troublesome problem in the history of training deep neural network. Residual networks were introduced in hopes of dealing with this problem. A typical residual block is given below:

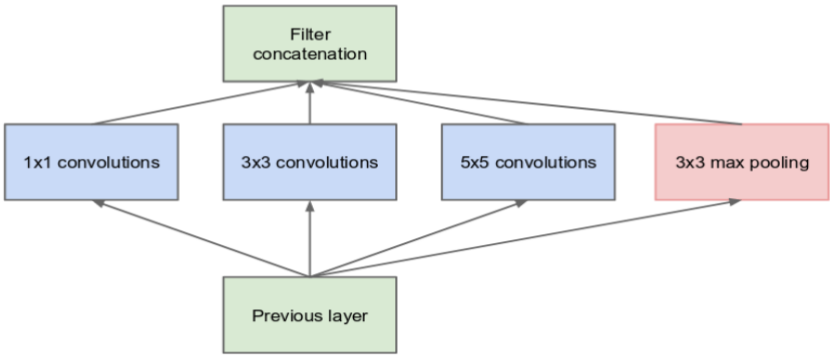


The core idea of ResNet is introducing a so-called âĂIJidentity shortcut connectionâĂİ that skips one or more layers.[4]
The intuition was that, stacking deeper layers should not degrade the performance of the neural network. Which means, the deeper model should perform better wrt to its

previous layer. There have been many advancements in Resnet architecture such as ResNext etc.
In this project, a typical resnet architecture using 4 stacks of recidual blocks was implemented.

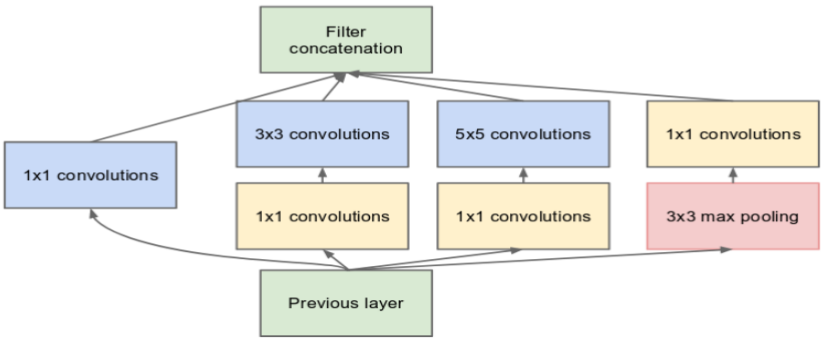
2.3 InceptionNet V2

The inception network is more complex than a typical deep neural network. Inception V2 was introduced [5] to improve the Inception architecture in terms of both speed and accuracy. The intuition was to make the network more wider rather than deeper. The figure shows the Inception V1 block:



(a) Inception module, naïve version

The figure shows that the inception block includes a layers with different filter sizes of 1x1, 3x3 and 5x5. The outputs of these layers are concatenated and sent to the next inception block. [6]
In InceptionV2, to make deep neural networks cheaper, the architects of inception network limited the number of input channels by adding 1x1 convolution layer before the 3x3 and 5x5 layers.



(b) Inception module with dimension reductions

3 Regularizers

The following regularizers are used for this project:

3.1 Early stopping

[9] Sometimes, while training on a large dataset, the model will stop learning and instead focus on the external noise. To avoid this, early stopping is used in all the experiments. This technique tells us just the right time to stop training of the model. In this project, the early stopping is applied to observe saturation of validation loss and patience is set to 20.

3.2 Batch Normalization

[8] Batch Normalization is a technique wherein the input layer is normalized by adjusting and scaling the activations. It normalizes the output of the previous activation layer by subtracting the mean and dividing by the batch standard deviation. This ensures that the model has stable behaviour and avoids overfitting. The training time is also reduced using this.

3.3 Dropout

[8] This method helps in reducing overfitting of the model. The nodes are removed randomly trained network so that it does not overfit to the training data. So, at the test time, we can approximate the average of all the predictions easily since the network is already trained.The dropout rate in this project is specified to be 20%

4 CIFAR100 Dataset for training

CIFAR100 is a dataset used generally for training and testing models for image classification and recognition tasks. There are total 100 classes. There are 500 training images and 100 testing images per class. The images are RGB with size of 32x32.

5 Variations of parameters used

5.1 Lower learning rates

In case of ADAM, the experiments with various learning rates showed that lower learning rate such as "0.0001" yeilds a better solution. This also helps in overfitting problem. Whearas in SGD, a higher learning rate such as "0.01" proved to be better.

5.2 Gradient clipping

[7] Gradient clipping is used to counter the problem of exploding gradients. Exploding gradients occur when the gradients get too large in training period making the model unstable while vanishing gradient means the gradients become too small for model to learn anything. The idea of gradient clipping is ig the gradient gets too large, we clip it to make it small. This helps the gradient descent to have a normal behaviour. Here, gradient clipping is used with clipnorm=5.

5.3 Activation functions

The most used activation function 'relu' is used in most of the experiments. But, through multiple experiments it is seen that relu does not fit well with ADAM optimizer. The model becomes very unstable and the loss functions varies a lot. Hence,

to tackle this unstable behaviour, 'elu' and sometimes 'leakyrelu' activations are used in experiements considering ADAM optimizers.

6 Results

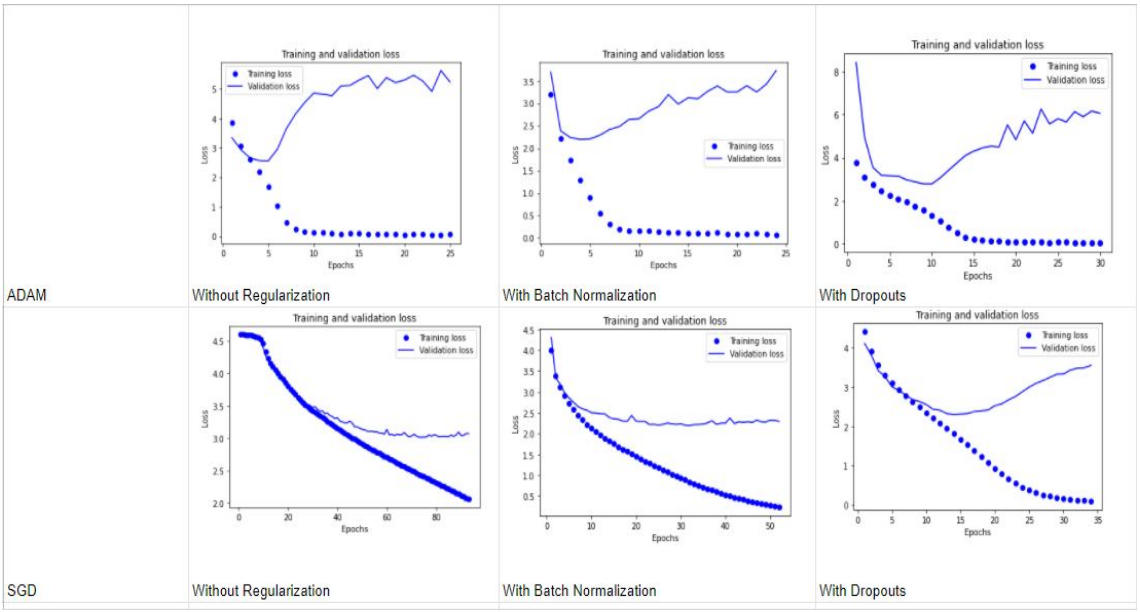
The results of all the experiments is summarized in the following table:

	Architecture	VGG 16			ResNet 18			Inception v2		
Optimizer	Score Setting	Presision	Recall	Accuracy	Presision	Recall	Accuracy	Presision	Recall	Accuracy
SGD	With BatchNorm	0.44	0.43	0.43	**0.54**	**0.53**	**0.53**	0.37	0.37	0.36
	With DropOut	0.46	0.45	0.45	**0.5**	**0.48**	**0.48**	0.34	0.29	0.29
	No Regularization	0.32	0.3	0.3	**0.48**	**0.45**	**0.45**	0.28	0.27	0.27
ADAM	With BatchNorm	**0.54**	**0.5**	**0.5**	**0.52**	**0.51**	**0.52**	0.38	0.38	0.38
	With DropOut	**0.5**	**0.49**	**0.49**	0.49	0.48	0.48	0.33	0.31	0.3
	No Regularization	0.39	0.38	0.38	**0.47**	**0.46**	**0.46**	0.4	0.4	0.4

The detailed discussion of these results is as follows:

6.1 VGGNet16

The VGGNet16 was trained on the CIFAR100 dataset using a combination of 2 optimizers and 3 reularization types. VGGNet when trained with the whole architecture was very expensive for a smaller dataset like CIFAR100. Therefore, the layers were reduced as mentioned in sections before. This yeiled faster running times and better results. Following are the graphs of VGGNet on different combinations of the experiment:



In the above graphs, we can see that the validation loss for ADAM optimizer was very high wrt to the SGD optimizer.
The following scenarios were considered:

1. Having all 18 VGG layers.
   Precision = 0.28
   Recall = 0.27
   F1 Score = 0.27
   In this scenario, the reults were so bad because our model was overfitting to the training data, we know this because the training accuracy was 96% while the test accuracy was 28%.

The other problem was that the model was too complex for the dataset. Hence we tried reducing the layers and obtained following results.

2. Eliminating 4 layers of 512 channels.
   Precision = 0.38
   Recall = 0.37
   F1 Score = 0.37
   This was as far as VGG could achieve without regularization.
   Next trials to improve results using Regularization techniques were executed which are evident in the results table 1.

6.1.1 InceptionV2

The Inception network was particularly hard to train. This model took the most training time among the other models. The reason being, 4 blocks of inceptions were used in the start. In the end, following graphs were obtained:

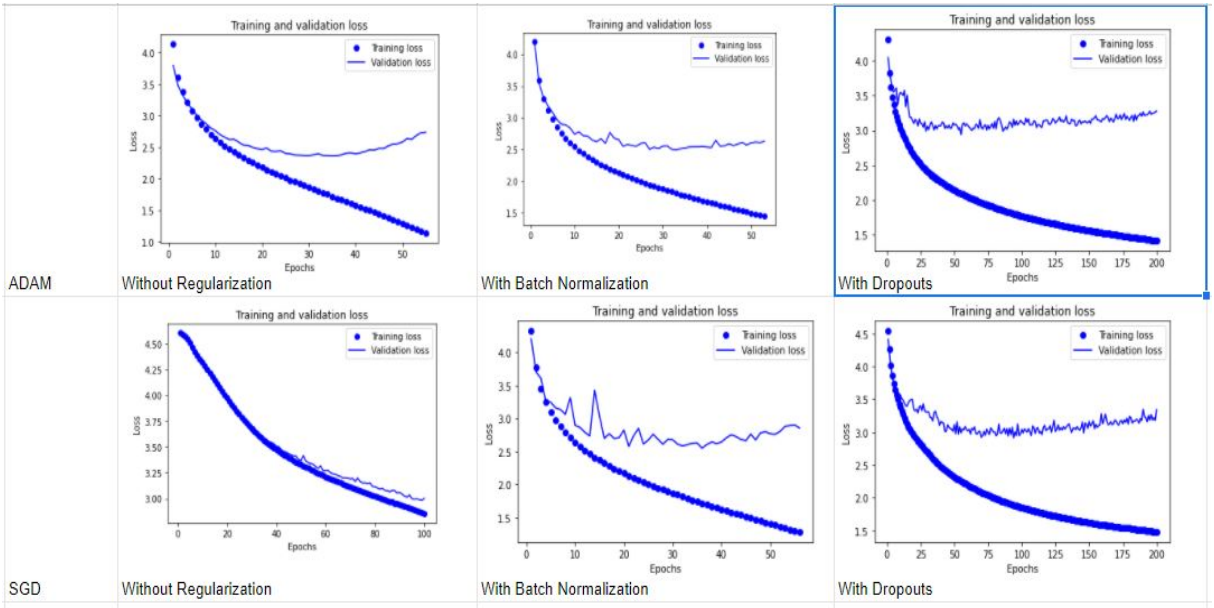


Figure 3: Loss graphs for InceptionV2

The following scenarios were considered:

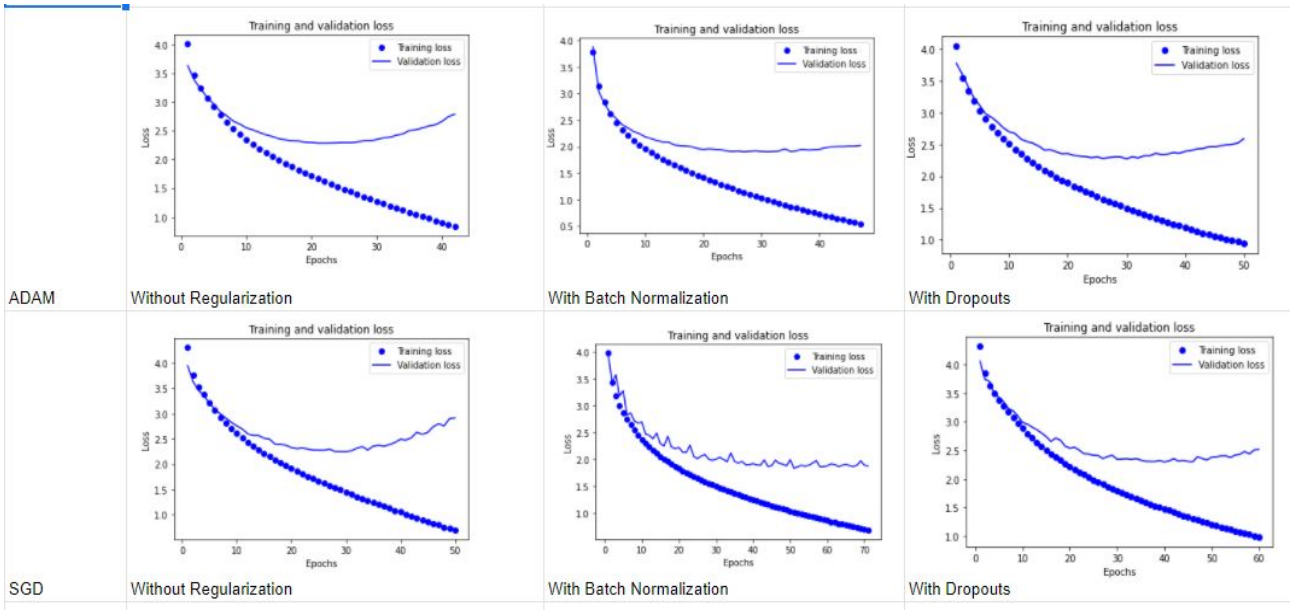
1. Model with one Inception Block:
   Precision = 0.26
   Recall = 0.23
   F1 Score = 0.23
   This model clearly lacked the depth and hence gave smaller accuracies for both training as well as test data.

2. Model with 4 inception blocks:
   Precision = 0.40
   Recall = 0.40
   F1 Score = 0.40
   This model worked really well as far as accuracy is concerned. But it took a lot of time to train it. Regardless, since time wasn't that much of an issue, this model was selected for further experiments.

One particular abnormality observed in Inception was that, in case of ADAM optimizer, the regularized models yielded worse results than naive inception model. Hence, ADAM might not be the best option suited for the Inception model.

6.1.2 ResNet18

ResNet18 was again trained on the CIFAR100 dataset using a combination of 2 optimizers and 3 reularization types. The following graphs were obtained:



Resnet18 was the most efficient to train and results obtained were better than VGGNet or Inception. As it is evident from the graphs, the validation loss in almost all the cases had a smoother downward curve than other two architectures. It also yeilded the best results with SGD optimizer and Batch normalization as shown in the results table 1.

7 Conclusion

As seen from the above results, ResNet18 might be the better solution for CIFAR100 Dataset. Although these experiments are not sufficient to solidify this conclusion, the curve nature of the loss fucntion itself gives us a better idea of ResNet's improvement over other model since the curves in resnet are smother than others.
These results also let us know that ADAM does not work well with 'relu' activations while it gives decent results with elu and leaky relu. SGD works well with relu as well as elu and leaky relu.
Another observation can be made that **Dropouts** if not used correctly, can force models to perform badly. Using dropouts at a fully connected layer of before the output layer yeilds better results.
While in case of batch normalization, it has to be applied before an activation function. This not only yields better accuracies, it also helps in training the model faster. It might be interesting to further demonstrate these experiments with other optimizers and regularizers in the future.

8 References

1. Very Deep Convolutional Networks for Large-Scale Image Recognition : https://arxiv.org/abs/1409.1556

2. Large Scale Visual Recognition Challenge 2014 (ILSVRC2014): http://www.image-net.org/challenges/LSVRC/2014/results

3. VGG16 âĂŞ Convolutional Network for Classification and Detection : https://neurohive.io/en/popular-networks/vgg16/

4. An Overview of ResNet and its Variants: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

5. Rethinking the Inception Architecture for Computer Vision : https://arxiv.org/pdf/1512.00567v3.pdf

6. A Simple Guide to the Versions of the Inception Network: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

7. What is Gradient Clipping?: https://towardsdatascience.com/what-is-gradient-clipping-b8e815cdfb48

8. Breakout session 5 on Sep 16, 2020.

9. A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/