Course Name: Machine Learning and Deep learning - Fundamentals and Applications

Professor Name: Prof. M. K. Bhuyan

Department Name: Electronics and Electrical Engineering

Institute Name: Indian Institute of Technology, Guwahati

Week-12

Lecture-45

Greetings everyone. Welcome to the online MOOCs course on machine learning and deep learning fundamentals and application. I am Bivek Goswami the teaching assistant for this course. I am a research scholar under the supervision of professor M.K. Bhuyan sir in IIT Guwahati.

Today we will be having a programming session on generative adversarial network and unet architecture. So let us begin our session today. So this will be the contents. We will be looking into generative adversarial network and unet architecture.

So as we have already seen in our theory classes what generative adversarial network is generative adversarial network consists of two architectures. One is a generative network and other is a discriminative network. The generative network takes input as noise and generate images and the discriminative network what it does is? It compares the results with the real images and it can tell us if the image is a real or a fake. So the steps that we will follow today is first we have to define GAN architecture based on our application. So in today's session our application will be to generate handwritten digits and the discriminator will be trained to distinguish between real and fake data.

So the real data will consider the MNIST dataset. MNIST is a very famous handwritten dataset that we will be considering for this session. So first we will train the generator to fake the data that can fool the discriminator and then continue discriminator and generator training for multiple epochs so that we reach a level where the generator can generate fake data in such a way that the discriminator is unable to identify that. And later we will save the generator model to create new and realistic fake data. So let us start with the coding.

So you have already seen in the previous programming session how to use Google Colab and how to convert the notebook into a GPU or a TPU settings and how can you train convolution neural network. So for training this GAN or U-net we will be also

considering convolution neural network as a backbone. So here you can see we have first imported the MNIST dataset from the tensorflow.keras.dataset and later in the next layers we have imported these layers which is the input layer, the dense layer, the reshape layer and the flatten layer.

We have also imported back normalization. Leaky ReLU we will be using rather than ReLU activation function that will be more suitable for training this GAN network and we will be using a sequential model and the model from Keras model so that we can better optimize our code. For optimizers we will be using Adam optimizer and NumPy and Pyplot will be considering for error manipulation and plotting of the images or the graphs. So first we will be defining our image size. So we have considered an image of 28 x 28 x 1.

So this is basically a grayscale image that is having only one channel. Why we have considered 28 x 28? Because taking too much bigger image will be too much resource and time consuming for training. So we have made our input shape as 28 x 28 x 1. So this is basically our input shape. Now we will be creating a directory so that we can save the images there.

So now let us build the generator module first. So we have defined a function to do that. We will be inputting a noise or latent vector. These are almost the same terminologies that are mostly used separately in separate books or tutorials that you will see. So I have kept both of them.

So we will be creating a 1D array of size 100 that is the noise. So this will be randomly created. As you can see noise shape is written. So this will be randomly created 1 x 100 dimensional noise. Now we will be creating a sequential model.

As you have already seen in the CNN tutorial what is a sequential model. Sequential model basically lets us add layers sequentially means one after another. So we will create a sequential model. This sequential model as you have already seen we have imported from our Keras framework. First we will be adding a dense layer.

So dense layer is basically a fully connected neuron layers that we will be first adding. Then we will be adding an activation function that is the Leaky ReLU function and then we will be adding batch normalization. This momentum basically tells us how fast it is going to train. And Leaky ReLU alpha value will tell us about the activation constraints. So we will be having 256 neurons in the first layer.

In the second layer we will be again having 512 neurons and rest will be same. Your alpha value will be 0.2 and momentum is 0.8. And in the last layer of the generative network will have 1024 neurons with the alpha and momentum value same.

Now we will be in the last layer we will be using a tanh activation function and we will be adding the image shape because at the end of the generator the generator should produce an image that is equal to the size of the input image. So it should produce an image of size 28 x 28. So what it is doing is that it is taking a 100 dimensional array and it is returning as a 28 x 28 dimension image. So it is considering a 100 dimensional noise and it is returning as a image of 28 x 28. Now we will be reshaping it to the image shape that is this one 28 x 28 but initially we have shaped the image to be 28 x 28 x 1 and this is the channel information.

So we will be making the model like this. So what our model will return? It will consider an image and it will consider a noise and it will return as an image. So as I have already shown alpha is a hyper parameter which controls the underlying value to which the function saturates negative network inputs and momentum speed of the training. Now we will be defining our discriminator model. So again we will be calling the sequential model.

Now we will be adding the flatten layer. Now the function of the discriminator is to return if the image generated by the generator is real or fake. As we have already seen in this structure the generator generates fake images and input to the discriminator whereas discriminator also takes a real image and tells us if the image is real or fake. So that is the job of the discriminator. So again we will be adding dense layer of 512 Leaky ReLU dense layer of 256 Leaky ReLU and at the end a dense layer of 1 because it will be outputting the value if it is a real or fake image.

So it is a binary value that it will output. That is why we are using the activation sigmoid because sigmoid works best in case of binary output. So again the model will return the validity. So we will be inputting the image and we will be getting the validity as output. So we will be looking into the training.

So we will be defining a training function. So first we will load the dataset like this and we will first see that the dataset is in Uint format. First we will convert it into a floating 32 and then scale it to -1 to 1. We can also scale it to 0 to 1 but I prefer scaling it to -1 to 1 for the purpose of better training. Now you can see I have added a channel that is I have expanded it.

So for now our image will be like this. In this step I have included another channel to make the image like this. Again I have taken half the batch because I have considered a batch size of 128. This training function will take input as epoch, batch size and save interval. Then after how many interval or iteration it should save the images in the images folder that I have already created as I have shown in the previous slides.

So I have considered half batch. This batch size is 128. So our half batch will be

around 64. Now when epoch in range of epoch so our total number of epochs was will be giving that.

In the input. So for every epoch so this tells us for epochs in range of effects so for each epoch what we are going to do the functionalities we are going to look now. So what we will do in this is when we will loop through a number of epochs to train our discriminator by first selecting a random batch of images from our true dataset, generating a set of images from our generator, feeding both set of images into discriminator and finally setting the loss parameters of both real and fake images as well as the combined loss. What does this mean? This means that first we will be generating some random numbers. This random numbers will be from your 0 to a half batch and it will be X dot train dot shape your X dot train dot shape 0 X dot train dot shape will actually return you an array with the shape of the image. So for example your X train has 300 images of shape 28, 28 and 1.

So your X dot train dot shape will return this as an array. So your shape 0 means this 300, the number of images present or the number of entries present. Okay and we will be selecting those random images. Now what we will do is we will generate some noise points between 0 to 1 of size 64 x 100 Okay. And those images, those noise images will be fed into the generator where the generator will predict or generate a set of fake images.

Now these fake images will be again fed into the discriminator to know the, to get the discriminator's loss fake. And again real images will be fed to the discriminator to get the real image loss. And we will take average of both the losses. Because based on one loss we cannot take a decision so we will be taking average of both the losses. Now since our generator has given us some output and the discriminator has again given us some output, now let us train the generator to better learn it and generate better images so the discriminator cannot identify if it is real or fake.

So the training of the, in the training of the generator phase we will be first again generating a set of noise similar to the last one that we did. And now for fooling the discriminator we will be considering some validation predicted outcomes. So this, what this line does is this will create a vector of 1s. So it is telling the discriminator that whatever fake image that the generator has generated are real images.

And those are valid. So again we will be calculating the generator loss by training the generator on this noise that we have created now and this input predicted values that we have generated. That will be used to fool the discriminator that is the real images. So we will be printing those values and will be if the epoch is divided by the save intervals that is 20 then we will be saving image at each epoch. So basically after every 20 intervals we will be saving images to get to know if our generator is generating good fake images

and if our discriminator is not able to discriminate it or not. Again in this function we will be looking at sample images.

For example rows and columns I have defined as 5 and 5. I have generated a random noise and predicted that using a generator. So the generator create some fake images. We have rescale the fake images to from 0 to 1 and we have plotted those things. So if created a figure and axis we have kept a count of it and from the range of row to column I have just in the axis this axis that I have created I have plotted the image and this I am saving after a fixed number of epochs.

So I hope I am clear till this point. Now let us define our optimizer. So as I mentioned we will be using the Adam optimizer and we will be considering the lining rate as 0.0002 and our momentum to be 0.5 here. Now we will be building the discriminator and we will be compiling it using binary cross entropy and matrix will be considering as accuracy.

Why binary cross entropy? Because I have already mentioned the discriminator will discriminate if the image is a real or a fake and that is a binary classification problem. That is why we are using a binary cross entropy loss and the matrix we have used as an accuracy. Again we will be building the generator with the binary cross entropy as well because the generator also generate images that are based on two categories either they are fake or they are real. So we will be inputting a 100 shape noise point to the generator and the generator will generate fake images. Now why this discriminator trainable function? We have set to false here because at the time of training the generator we do not want our discriminator to get trained as well because we have seen and it is a proven fact that when we train two network separately it is giving better outcome and it is actually good.

So while the generator is training we stop the discriminator training and we make sure that the discriminator is not training at the same point of time. This saves us resources as well as it is better when it comes to the outcome. So now since our generator has been trained we will be sending the images generated by our generator into the discriminator and after that we will be combining both the models. So in this step it is very important to understand this step. In this step we have combined the models and also set our loss function and optimizer till now.

So we are training the generator we are setting the discriminator training to false and then again sending the images generated by the generator to the discriminator to get the validation of the images. Now we have to create this combined model where stack generator and discriminator takes. So in this model a noise in input into the system, the system generates images and determines its validity. So this model basically is combining generator and discriminator into the same model and this is what a generative

adversarial network is. Again we have considered binary cross entropy and optimizer, that is Adam, optimizer we are using.

We will be training this for 1000 epochs that means it will run for 1000 iteration with a batch size of 32 and a same interval of 50. And then as I have already mentioned in the steps we will be saving this generator so that we can generate good fake images later after the generator is properly trained. So this is the summary, this is the discriminator summary while this is the generator summary. So we can see in the discriminator the total trainable parameters is this and the total trainable parameters in generator is this. Whereas the total trainable parameters in discriminator is this.

The number of parameters in generator is more compared to discriminator because discriminator is a very simple network which basically does it, it predicts if the image input is valid or not. Whereas the generator generates a set of images from noise data and that is why it is a more complex network. You can see the parameters at every stages and based on these parameters you can actually know how much training time is it going to take or how much memory it is going to take and all those things we can configure from this. How to calculate this training parameters? There are ways to do that, those are different when we are using fully connected networks or convolutional networks. So we will see in the coming slides if we can look into one of those.

So this is the training, you can see here the dataset is getting downloaded as we have imported it directly from the Keras, so it is getting downloaded here and the training has started. You can see this is the discriminator loss and this is the generator loss that is changing at every epoch and these are the number of samples that are considered for each epoch that is based on our batch size. We consider our batch size such that our total number of samples divided by the batch size is the total number of samples that is considered for each epoch. So that is the advantage of having a batch size as you have already seen these things in the previous programming session in the convolutional neural network is just the same. Now since we have trained our GAN and we have saved the model, now it is not possible to save the train the model every time.

So it is important that you save the generative model or the generator model and later you just load the model and you can generate new images. So that is what we have done here, we have imported the model here and what we are doing is we are providing random noise to the model and it is generating an image. So this is the image that the model has generated. You can see this resembles to 9 and it is actually a fake image which somewhat resembles to 9 handwritten 9. Now considering for multiple images, so we have defined a function for generating latent input we just give the latent dimension and the number of samples as input to the function and it generates us a input.

So we have given 100 dimension and 16 as the number of images that we need. So this is basically plotting of the images where we are considering 4 x 4 dimension and every cell would display an image. So based on this latent points what this is returning we have generated some fake images using our model the saved model and we have plotted here. So you can see this pretty much resembles the handwritten digits and these are actually fake images generated by our generator network. So this is all about generative adversarial network and this is what we have seen the generative adversarial network for generating handwritten digits.

Now we will be looking into another very important architecture that you have already seen in the theory classes the UNET architecture. So UNET architecture is basically used for image segmentation. It was proposed for biomedical image segmentation and that is why I have considered a case of biomedical image segmentation only where we will be segmenting MRI images of brain where we will be segmenting the brain tumors from MRI images. So this is the architecture that we will be looking into our input image will be a 3 channel input image so that is we have RGB image and we will be having this number of filters and these are the concatenation block these are up convolution of 2 x 2 and these are max pooling layers these are convolution layer. So you can see there are multiple convolution layer that are present.

So what we thought better would be to define a convolution function only. I will come to this later first I will tell you so this is the dataset that we have considered is a brain tumor dataset segmentation dataset. So you can upload the zip file of the dataset into your Google Colab file system and then use this snippet of the code to actually extract the zip file to a normal folder. Now we will be implementing this sorry you will be importing this libraries that will be important for our task to implement.

NumPy as you all know is used for matrix manipulation. OS lets us have the idea of the path and everything. Glob is a library which lets us select images and consider number of images or path. CV2 which is a open CV library used for image manipulation.

Matplotlib is for plotting. Random is another library. And tensorflow is a framework that is used to implement or code deep learning models. So these are the path of my images that is the train valid and test path of the images and the mask this I have written like this. Now first thing I have to arrange the dataset that is I have to read every image and store it in an array or a list so that I can read from that list and better train the model. So I have created a data load function where first I have sorted the images according to the path. The path you have already seen for the mask and the images so I have just sorted it and I have set the image size to 128 plus 128.

Again this is done to reduce the time and complexity of the network for training and

resources. So again we have created two lists. One for the image and one for the mask. Now we will iterate through the image list and we will be reading every image using this OpenCV library and we will be resizing those image because those image can be of different shape and different size.

So we will be resizing into the size we have mentioned here of 128. And then we will be appending to this image dataset. Same thing we will be doing for the mask and appending into the mask dataset. Now we will be normalizing the images. It is important to normalize the images so that we can remove the bias towards higher values and that is important for segmentation. That is why it is done and we will return the image dataset and the mask dataset.

So again we have called the data load function providing the path of the image and the mask. For again for test we have done the same and for validation set also we have done the same thing. Now we can see that our training set will have this shape, the validation set and the test set. Now the height will be this one, the width will be again this one and the number of channels is 3.

So when we call the shape of this training set, say train.shape will return me array of this one. So this is the 0th position index, this is the 1 index, this is 2 index and this is the 3rd index. So our first index is our height, the second index is width and the third index is channel. So this is what I have set here and that will be our input shape.

Image height, image width and image channels. Now again the number of levels I have set to 1 because what we are trying to predict here is a mask and the mask is basically a binary object. So it has only a foreground and a background. So mask is basically binary that's why I have set the number of levels to 1 and I have set the batch size to 2 because of the memory constraints that our system that will get trained on will have, that is our Google GPU. So this is one of the image that I have shown.

This is the MRI image that has the brain structure and this is the tumor. So you can see basically this only has a foreground and a background. So this is a binary image so we will have only one class, this is a binary class. So that's why I have set the label to 1. So as I have mentioned already we are having many convolutional blocks. That is a 3 x 3 convolutional block, a batch normalization and having a relu, there is an activation.

So what I have done is I have created a function for those convolution block. So it's taking the input size, the filter size and these are the number of filters. So this is how convolution block is added as you have already seen in the previous programming session. This I am adding the batch normalization and then an activation layer. So this filter size will be 3 x 3 as you will see here when I am inputting the filters I am considering 3 x 3 and the number of filters will change for every layer that you have seen

already that this are having 32 filters, these are having 64 filters, 128, 256, 512 and so on.

So first we will be considering the downsampling. Downsampling you are knowing that is this layer. This is known as the downsampling that is extracting the features and this is known as the upsampling and this is the downsampling. Now first we will be defining the downsampling layer. So the first having convolution layer having 32 filters, 3 x 3 is the filter size and inputs. So your input will be basically a input layer with input shape and your data type will be float32.

So this function that I have created for defining unet will take in as the input shape. So here our input shape will be 128 x 128 x 3. This we have defined earlier only. The number of classes we know will be 1 because it is a binary segmentation.

So input layer we have built. This is the first CNN layer. Now we will be having the first max pooling layer. So see we are having two CNN layers and then we are having a max pooling layer. So we have defined two CNN layers and then we are defining the max pooling layer with pool size 2 x 2. Again we are considering 64 filters of same 3 x 3 and we will be having again two CNN layers here.

So here two CNN layers we are having, sorry, here we are having two CNN layers. Again here also we are having two CNN layers. This you can see here we are having two CNN layers. We will be having two feature maps at every step.

Again we will be having a max pooling. Again two CNN layers but this time 128 filters. Again max pooling, 256 filters, two CNN layers, max pooling and finally 512 filters. Now you know that in an unit architecture it is important to upscale as well as it is important to concatenate as well to get the spatial features and also to get the features which are extracted by the downsampling or the encoder section. So those are important and to revise the spatial feature we do the up-convolution. So in the upsampling layers we will first do an upsampling or the up-convolution and then we will be concatenating the last layer with this layer.

So we will be first upsampling this or convolving this and then concatenating from this one. So we are upsampling it and concatenating it. So 512 we are having, we are upsampling it and we are concatenating this one. So it is basically 256, 256 and then we will be having a convolution to get the 256 filters here.

So that is what we have done. Similarly for the next layers we have done the same thing. But in the final layer I have used a 1 x 1 convolution to number of classes. So what 1 x 1 convolution does is 1 x 1 convolution reduces the number of filters to the required number of filters keeping the spatial dimension same. So from this layer I will

be getting an image of 128 x 128 x 1 that is similar to the input image that we have initially input to the network. Again we will be applying batch normalization and sigmoid layer.

Sigmoid layer why? Because we are considering binary segmentation. So this final layer an input considering this will be creating the unet model and will return the model for this function that we have created for unet architecture. Now we will be calling the unit architecture with the shape 128 x 128 x 1 and we are printing the summary of the network as we have seen for generator and discriminator summaries. Again we will be compiling the optimizer with Adam binary cross entropy and matrix will be considering is that accuracy. So this is basically the summary of the network you can see this is our input this is the next layer that we are having this is again the next layer we are having and these are the number of parameters for every layer. So for this convolution layer how did we get this parameter is that we are having the number of filter size as 3 x 3 okay and what is the number of filters here also 3 so it will be 3 x 3 x 3 + 1 into the number of filters that is 32 that is 28 x 32 if you do this you will get to 896.

So this is how we get parameters for convolution layers. So we can see again till this point we are having 32 filters number of filters get on increasing this is the first convolution block we have 32 filters again the second convolution block we had 32 filters only okay from the third convolution block the number of filters became 64 and it goes on like this 64 then 128 then 256 512 and it again decreases in the up sampling block and finally we get the input dimension only and this is the total trainable parameters that we are having okay. So we will start the training we have imported time so note the start time and the end time so we will train this for 15 epoch just to give the demo we have made shuffle false so that the data is not shuffled it is the same when the this every epoch we consider the number of images it is not shuffled it is taken sequentially. So in the model.fit I have provided with the mask and image verbose one it sets the model to print these values while it is training this values while it is training okay batch size we have already set the batch size to 2 and this is the validation set the validation set is actually used to fine tune the network and we have trained it for 15 epochs again we have taken the stop time to just get the execution time and later we have saved the unet model. So this is the training in each epoch this number of samples are considered and you can see the loss is converging the validation loss is also converging and we are getting surge in our accuracy okay.

So these are the loss curves and the accuracy curve that we have plot based on our model so you can you have seen when while training I have saved the training in this unet history and later on from this unet history I am just plotting this graphs by considering the loss and validation loss and considering the total number of epochs and dividing it into 15 so 1 to 15 and how our loss is decreasing with the number of epochs

we are looking into this and how our accuracy is also increasing with the number of epochs. So since we have you are considering very less amount of accuracy sorry less amount of epoch so we are having this oscillation in the graph if we consider it for more epochs and consider a smaller learning rate then you will have a very smooth curve that is actually learning fast and it will converge to a point where your model will be able to segment any new data that you input into this. So this is all about the graphs that we have plot we are just plotting this B gives us the color and this is the label that we are providing for every this graph so this blue one is the training accuracy while the red one is the validation accuracy and similarly for loss okay. Now predicting using unet so for predicting we have considered the mean IOU and the F1 score that we have imported like this IOU is intersection over union it has the formula of so it basically tells you if these two are the images it tells you this intersection over the total union and how much overlap is there between the mask and the image.

So it gives us the ratio of the overlap by the area of the union okay. Another metric that we have considered is the F1 score which basically is the mean of the precision and recall the harmonic mean. What precision told us to tells us is that the ratio of the true positive to the total number of data points that are predicted positive by the model whereas recall tells us the ratio of the samples predicted positive out of the total sample of in the classes So the both tells us about the prediction of positive so F1 score gives us a relative ratio between both of this and that is why it is important when considering biomedical images or data because in biomedical images we cannot rely only on accuracy as it gives the overall accuracy is more important to know the false positive and then false negative in case of biomedical data okay. So here again we have considered randomly image from the test set and the ground truth we have extracted we have expanded the dimension and we have made the model to predict. So the model is predicting on the test image and we are checking if for all the prediction that are greater than 0.5 as set as 1 and less than 0.5 as set as 0 to get this mask okay and then we are just plotting this mask using this subplots we have created 3 subplots and one is the testing image this is the testing level and this is the prediction level. So you can see we have got the mean IOU of 91.43% which is considerable this we have considered like this number of classes and the prediction and the IOU we have calculated. Again we have done it for multiple images so we have taken a set of images in our test set so we have iterated in our test set for every image we have calculated the predicted mask and we have calculated the mean IOU using this one and we have appended the IOU into a array.

So finally we have calculated the F1 score and we have calculated the mean IOU and the F1 score. So average F1 of all the frames we have got to be 98.02 which is quite good and new IOU on the test set is 71.27% which is also considerable. So if we increase the number of epochs and we adjust the learning rate the accuracy or the

performance could be better improved for this network. So in today's class what we have seen is we have seen two very important deep learning networks that is the generative adversarial network and another is the unet architecture.

Generative adversarial network is used to generate fake images that could be used for different purposes such as since there are less images present for some application and it requires more images that could not be tackled by generating some fake images that could be even generative adversarial networks are very important for data augmentation task where conventional data augmentation could be replaced with generative adversarial networks for better implementation or better generation of fake images. Later we have seen unit architecture which is another very important deep learning architecture for image segmentation that is mostly used in biomedical image processing or biomedical segmentation task and is a very important architecture because it gives us an encoder decoder structure where we input the image and we get the mask output of the same dimension.

So those are the two networks that we have seen in today's session. So let us stop here today. Thank you. Thank you.