UDACITY

CNNs Concepts

Lamia Zain
lamiahasan4@gmail.com
LinkedIn

June,14, 2024

Connect Sessions | Purpose

A Connect Session **IS**:

- Focused on learning, encouragement & graduation for a group of students coached by a Udacity Session Lead
- Setting weekly study goals
- Helping each other with progress (including peer to peer)
- Keeping everyone accountable for their responsibilities
- A way to meet individuals in tech field & learn about the industry
- **Mandatory**

A Connect Session **IS NOT**:

- A social meetup
- A study group
- A substitute for online learning
- **Optional**

UDACITY

Let's check your progress

You are encouraged to spend at lest 10 hours/week to graduate.

Presentation date

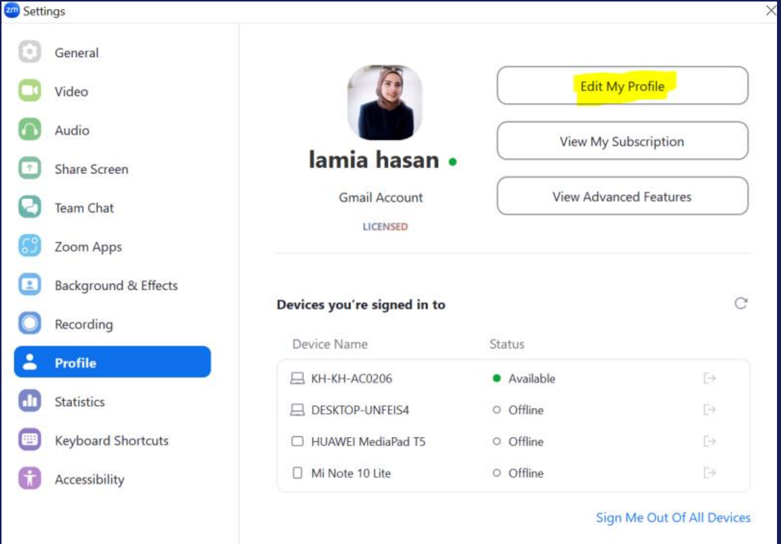
UDACITY

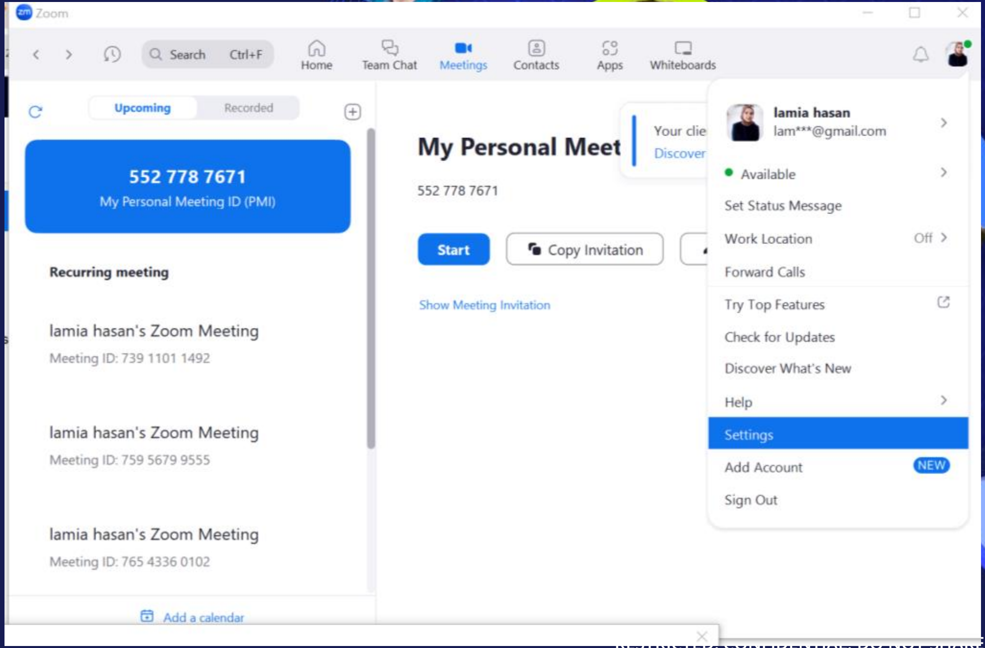
Attendance is taken automatically

Please change your name to be First Name and Last name on Zoom
Like : Lamia Zain

UDACITY

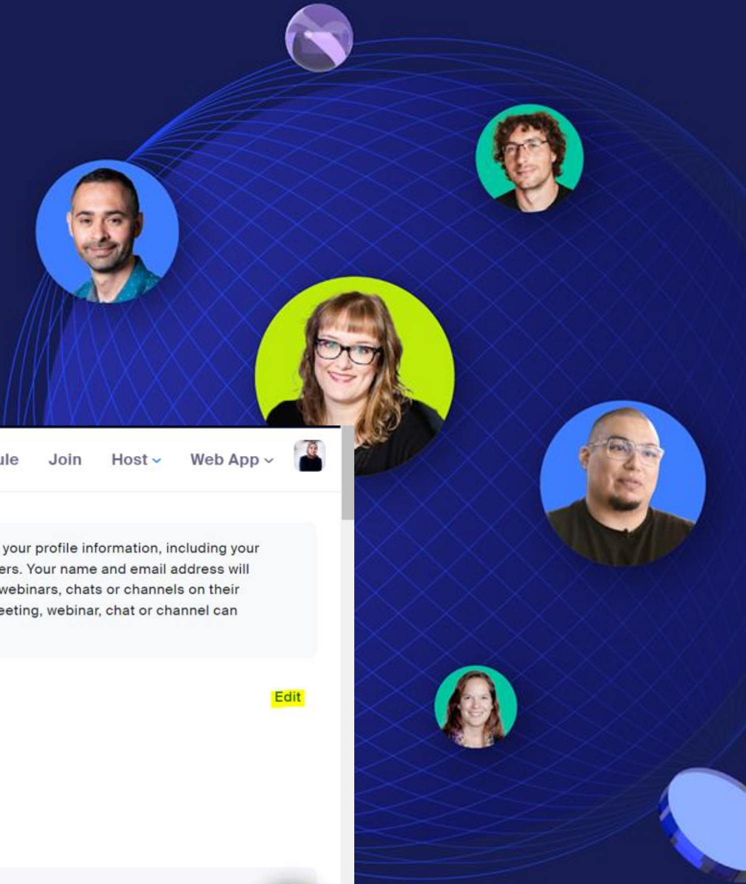
Change your Name on Zoom

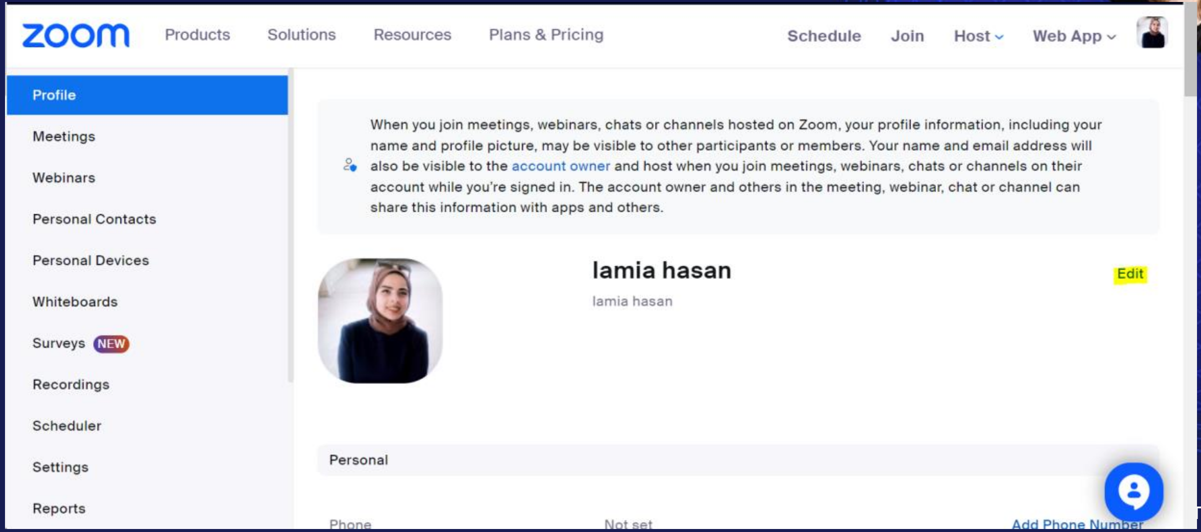




UDACITY

Change your Name on Zoom





Session Lead role:

Communication Chart

Issue	Where to go?
Classroom access/ Withdrawal/ Graduation issues/ Plagiarism/ Project Review Inquiries	Email support@udacity.com
Technical Issues, Attendance, Content Related Issues/ Project inquiries	Session Lead
Session Switch/ Community related issues	Community Moderators

2024

April | May | June | July | August

Program Kickoff
April 10, 2024

Revoking Deadline
May 12, 2024

End of Program
August 28, 2024

First Project Submission
May 1, 2024

Second Project Submission
June 5, 2024

Third Project Submission
July 10, 2024

Fourth Project Submission
August 21, 2024

Program Period
April 10 - August 28, 2024

Four-weeks Agenda, Weekly schedule

Week 10	Jun 12, 2024			Finish the lessons below from **the Convolutional Neural Networks** Introduction to CNNs CNN Concepts **[Work on/submit the #3 project: Landmark Classification & Tagging for Social Media]**	Convolutional Neural Networks Introduction to CNNs CNN Concepts
Week 11	Jun 19, 2024			Finish the lessons below from **the Convolutional Neural Networks** CNNs in Depth **[Work on/submit the #3 project: Landmark Classification & Tagging for Social Media]**	Convolutional Neural Networks CNNs in Depth
Week 12	Jun 26, 2024			Finish the lessons below from **the Convolutional Neural Networks** Transfer Learning **[Work on/submit the #3 project: Landmark Classification & Tagging for Social Media]**	Convolutional Neural Networks Transfer Learning
Week 13	Jul 3, 2024			Finish the lessons below from **the Convolutional Neural Networks** Autoencoders **[Work on/submit the #3 project: Landmark Classification & Tagging for Social Media]**	Convolutional Neural Networks Autoencoders Project Walkthrough: Landmark Classification & Tagging for Social Media
Week 14	Jul 10, 2024	Jul 10, 2024	**Landmark Classification & Tagging for Social Media**	Finish the lessons below from **the Convolutional Neural Networks** Object Detection and Segmentation **[Work on/submit the #3 project: Landmark Classification & Tagging for Social Media]**	Convolutional Neural Networks Object Detection and Segmentation Project Walkthrough: Landmark Classification & Tagging for Social Media

Student Milestone | Revoking

REVOKING

Revoking is the process by which Udacity removes a student from a Nanodegree program.

AWS reserves the right to revoke you from the program if you do not comply with program requirements.

CRITERIA

Students can be revoked if they fail to:

- Submit Project 1
- Complete the required concepts

The date of **Revoking** for the scholarship is **May 12, 2024**

Code of Conduct | Plagiarism

BASIC RULES

- Project submissions must consist of original work
- Submitted projects will be scanned for plagiarism
- Students who are found to have plagiarised will risk their Nanodegree being revoked
- Read the honor code and the rubric carefully for all projects

Recap

Let’s Test the previously trained Network

Calculate testing dataset Loss

Calculate testing dataset Accuracy

Training Techniques

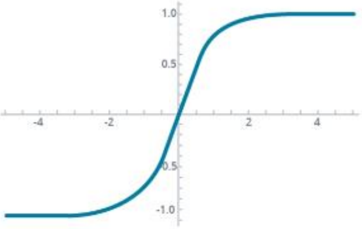
Save the best model with the lowest Validation Loss

```
Threshold = 0.001  #Very small  value
Patience = 0
Patience_Limit = 3

Looping Epochs:
        Looping Training batches:
                Accumulate Calculated Batch Loss
        Looping Validation batches:
                Accumulate Calculated Batch Loss
        Compare The current Validation Loss with the previous one:
                if Comparison <= Threshold
                        Patience +=1
                        If Patience == Patience_Limit:
                                Break;  #Which loop will be broken?
                else:   #To make sure Patience for consecutive epochs
                        Patience = 0
```

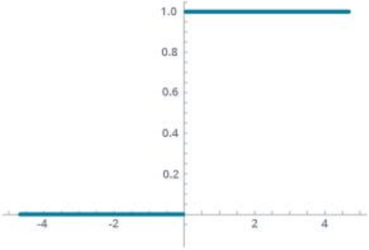
Which Can lead to Vanishing gradients and which can lead to Exploding gradients?

Saturating

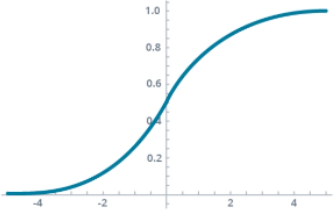
tanh



Step

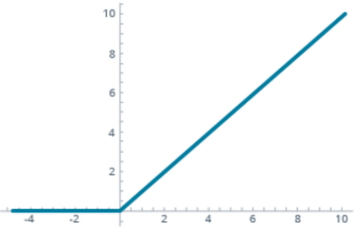


Sigmoid

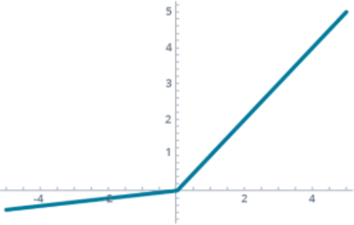


Non-Saturating

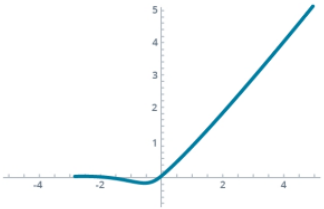
ReLU



Leaky ReLU



GELU



- Min-Max Scaling (Normalization):

This technique scales the data to a fixed range, usually between 0 and 1.

$$\text{X\_normalized} = (X - X.min())/(X.max() - X.min())$$

- **Standardization (Z-score normalization):**

Standardization transforms the data to have a mean of 0 and a standard deviation of 1.

$$\text{X\_standardized} = (X - X.mean())/X.std()$$

- **Log Transformation:**

Log transformation is used to reduce the skewness of the data. It applies a logarithmic function to the data, which can help in making the data more normally distributed.

Transforms

Other Initialization Method,

1- ToTensor(),

- This transforms takes a **np.array** or a **PIL image** of integers in the **range 0-255** and transforms it to a float tensor in the range 0.0 - 1.0.

2-PILToTensor()

- What Is the difference between ToTensor and PILToTensor?

2- Normalize():

- Normalize(mean, std)
- Normalize a tensor image with mean and standard deviation.

3-RandomErasing():

- Randomly selects a rectangle region in a torch.Tensor image and erases its pixels

4-Resize():

- Resize the input image to the given size.
- Img_Transform=transforms.Resize(size=(224,224))
- Img_Transform(Img);

5- resized_crop():

- Crop the given image and resize it to desired size.

- **img** (*PIL Image or Tensor*) – Image to be cropped. (0,0) denotes the top left corner of the image.

Apply as many transforms as you want to your Image using **transforms.Compose**([Transform1,Transform2, ……. ,])

```
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((-0.7387,), (0.6162,)),
                               ])

# Create training set and define training dataloader
dataset = datasets.MNIST('~/Deeplearning with pytorch udacity/DMNIST_Data', download=True, train=True, transform=transform)
```

- It's also a good approach to standardize your dataset. Read about standardization and normalization here

$$Image\ Channel\ Mean(\bar{x}) = \frac{\sum Channel\ pixel\ values}{Number\ of\ all\ pixels(n)} = \frac{\sum x_i}{n}$$

$$Image\ Channel\ Std(S) = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n}} = \sqrt{\frac{\sum (x_i^2 - 2\bar{x}\, x_i + \bar{x}^2)}{n}} = \sqrt{\frac{(\sum x_i^2 - 2\bar{x} \sum x_i + n\bar{x}^2)}{n}} = \sqrt{\frac{\sum x_i^2}{n} - \frac{2\bar{x} \sum x_i}{n} + \bar{x}^2} =$$

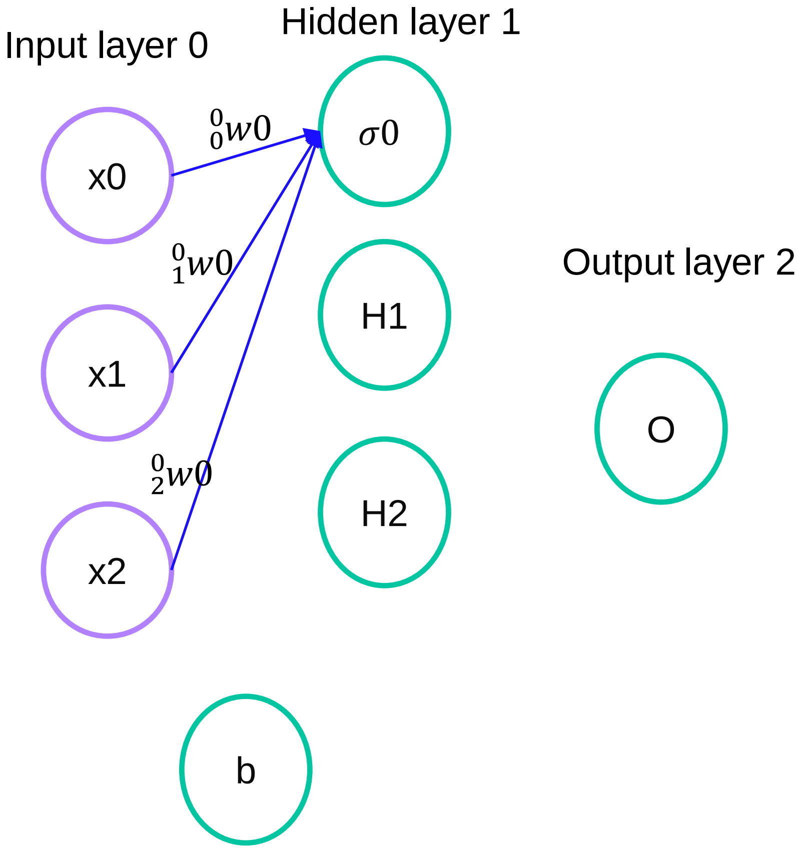
$$\sqrt{\frac{\sum x_i^2}{n} - 2\bar{x}^2 + \bar{x}^2} = \sqrt{\frac{\sum x_i^2}{n} - \bar{x}^2}$$

```
def Normalize(dataloader):
    #finding mean and std for input images

    summ_means, squared_sum_mean, num_batches= 0, 0, 0
    num_batches = len(dataloader)

    for data,label in dataloader:
        # Mean over batch, height and width, but not over the channels
        summ_means += torch.mean(data) #sum of means for all batches
        squared_sum_mean += torch.mean(data**2) #sum of mean of squares for all batches

    mean_gray = summ_means / len(dataloader) #num_batches = len(dataloader)

    # std = sqrt(E[X^2] - (E[X])^2)
    std_gray = (squared_sum_mean / num_batches - mean_gray ** 2) ** 0.5
    print("Mean is ",mean_gray.item()," STD is ",std_gray.item())
    return mean_gray,std_gray


Normalize(trainloader)
```

Let's see a code example

Weights Initialization

Weight Initialization [Methods in Pytorch]



$IL = [2,6,5]$
Initial Weights $= [1\ 0\ 1]$
b=0.3
Activation function → Sigmoid

Dot product

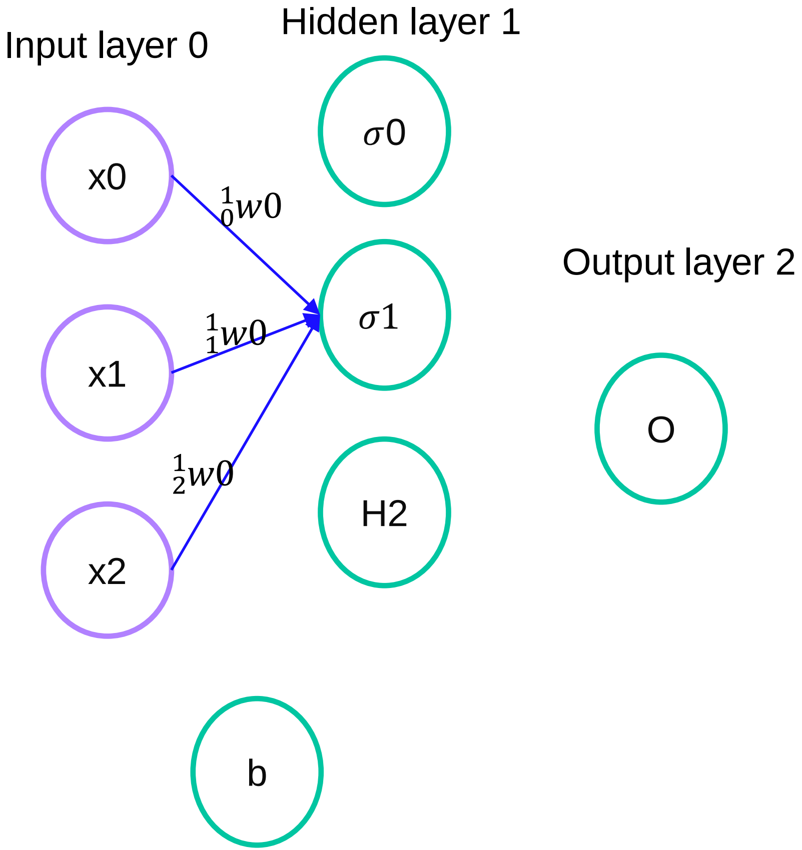
$$H0 = {}^0_0w0 * x0 + {}^0_1w0 * x1 + {}^0_2w0 * x2 = 7$$

Sigmoid function:

$$y = \frac{1}{1 + e^{-x}}$$

$$\sigma 0 = \frac{1}{1 + e^{-H0}} = 0.990$$

Remember



$IL = [2,6,5]$
Initial Weights $= [1,0,1]$
b=0.3
Activation function → Sigmoid

Dot product

$$H1 = {}^1_0w0 * x0 + {}^1_1w0 * x1 + {}^1_2w0 * x2 = 7$$

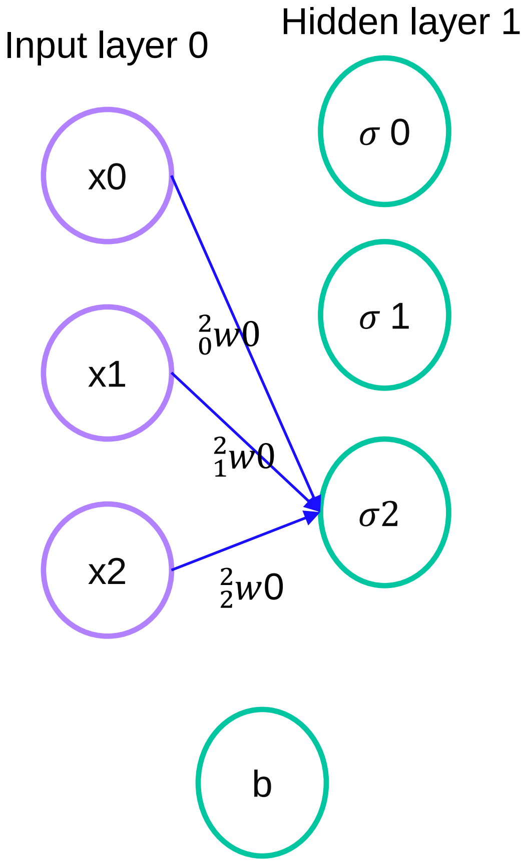
Sigmoid function:

$$y = \frac{1}{1 + e^{-x}}$$

$$\sigma1 = \frac{1}{1 + e^{-H1}} = 0.990$$

UDACITY

Remember



$IL = [2,6,5]$
Initial Weights $= [1,0,1]$
b=0.3
Activation function → Sigmoid

Dot product

$$H2 = {}^{2}_{0}w0 * x0 + {}^{2}_{1}w0 * x1 + {}^{2}_{2}w0 * x2 = 7$$

Sigmoid function:

$$y = \frac{1}{1 + e^{-x}}$$

$$\sigma 2 = \frac{1}{1 + e^{-H2}} = 0.990$$

The default weight initialization method in PyTorch, also known as the default initialization, initializes the weights of a neural network layer using a uniform distribution.

For each weight **W_ij** in the weight matrix **W** of the layer:

- Draw a random value **r** from a uniform distribution **U(-1/sqrt(k), 1/sqrt(k))** where **k** is the number of input units to the layer .
- Assign **W_ij** to the value of **r**.

The range of the uniform distribution is centered around zero, allowing positive and negative weights.

$$\begin{bmatrix} w1,1 & w1,2 & . & . & w1,784 \\ w2,1 & w2,2 & . & . & w2,784 \\ & & . & & \\ & & . & & \\ & & . & & \\ w256,1 & w256,2 & . & . & w256,784 \end{bmatrix} . \begin{bmatrix} I1 \\ I2 \\ . \\ . \\ I784 \end{bmatrix} = \begin{bmatrix} O1 \\ O2 \\ . \\ . \\ O256 \end{bmatrix}$$

[256,784]*[784,1] = [256,1]

$$\begin{bmatrix} O1 \\ O2 \\ . \\ . \\ O256 \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \\ . \\ . \\ b256 \end{bmatrix} = z$$

Other Initialization Method,

2- Uniform Initialization:
- Same as the default Method, but you can specify the range of values for the uniform distribution.
- PyTorch provides the **torch.nn.init.uniform_()** for this.

3- Normal Initialization:
- Weights are drawn from a normal (Gaussian) distribution.
- The **torch.nn.init.normal_()** is used for this.
- You can specify the mean and standard deviation of the distribution.

4- Xavier/Glorot Initialization:
- Commonly used for layers with the **tanh** or **sigmoid** activation functions.
- It scales the weights by a factor that depends on the **number of input and output** units of the layer.
- The **torch.nn.init.xavier_uniform_()** and **torch.nn.init.xavier_normal_()** functions are used to perform Xavier/Glorot initialization.

5- He Initialization:
- Commonly used for layers with the **ReLU** activation function
- It scales the weights by a factor that depends on the **number of input units** to the layer.
- The **torch.nn.init.kaiming_uniform_()** and **torch.nn.init.kaiming_normal_()** functions implement He initialization.

Let's see a code example

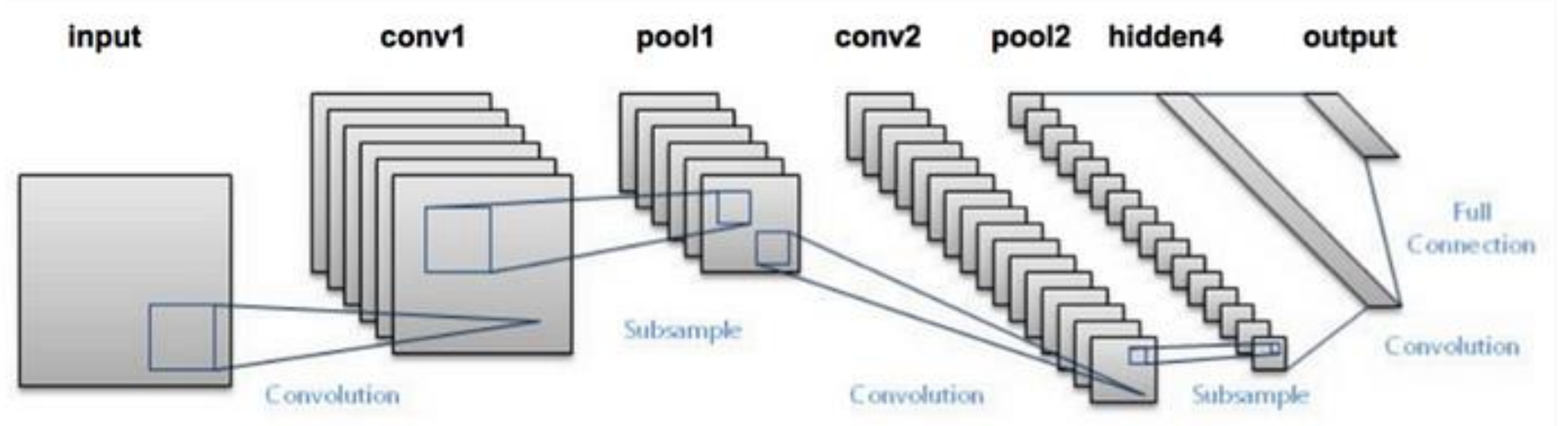
Break (10 minutes)

Satisfaction Survey

History of CNNs Based on Winning the ImageNet Large Scale Visual Recognition Challenge(ILSVRC)

LeNet-5 (1998):

- By Yann LeCun, along with his colleagues.
- LeNet-5 was designed to recognize handwritten digits.



ALEXNET (2012):

- Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton introduced AlexNet
- Deeper architecture (More Filters, Applied RELU Afs, max pooling, dropout, and data augmentation)
- AlexNet was trained for 6 days simultaneously on two Nvidia Geforce GTX 580 GPUs which is the reason for why their network is split into two pipelines.
- Reduced error rate from 26% to 15.3%.

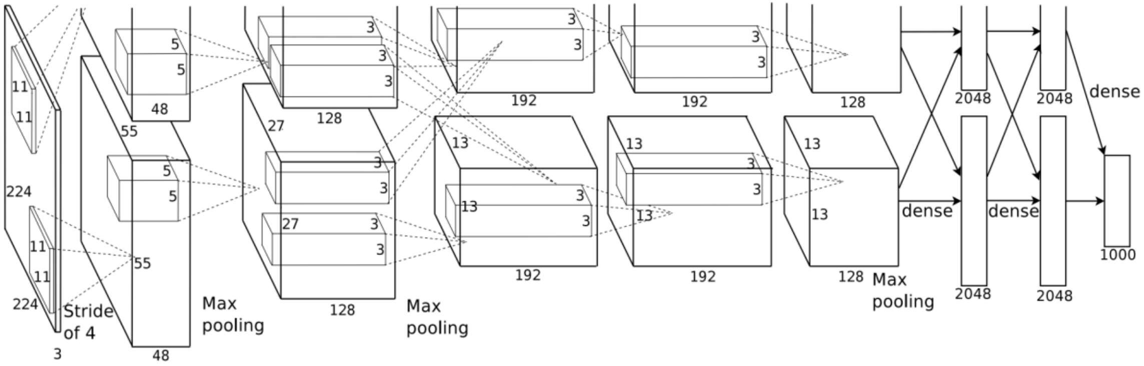
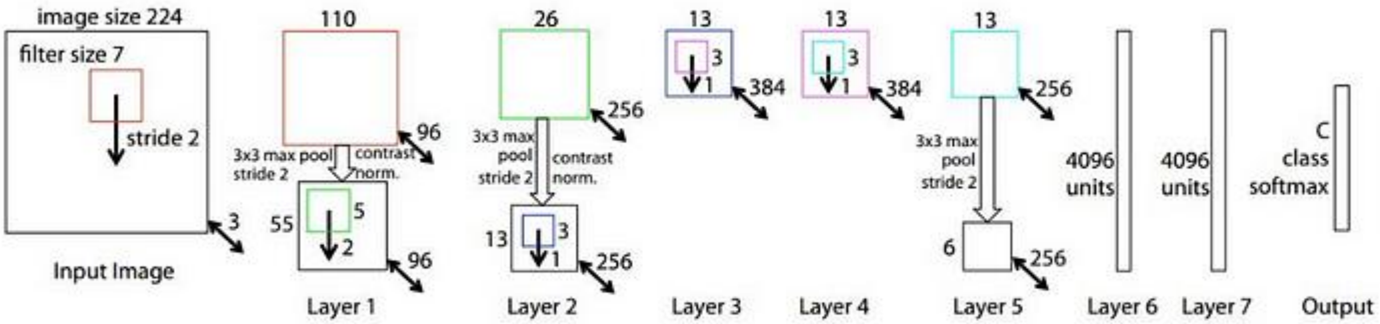


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

ZFNet(2013):

- It was mostly an achievement by **tweaking the hyper-parameters** of **AlexNet** while maintaining the same structure with additional Deep Learning elements.
- It achieved a top-5 error rate of 14.8% which is now already half of the prior mentioned non-neural error rate.



GoogLeNet/Inception(2014):

- It used a novel element which is called an **inception module** (batch normalization, image distortions and RMSprop).
- It achieved a top-5 error rate of 6.67%.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

VGGNet (2014):
- 16 Convolutional layers
- Many Filters.
- Trained on 4 GPUs for 2–3 weeks.
- VGGNet consists of 138 million parameters, which can be a bit challenging to handle.

ResNet (2015):
- Residual Neural Network (ResNet) by Kaiming He t al Trained on 4 GPUs for 2–3 weeks.
- introduced anovel architecture with “skip connections” and features heavy batch normalization
- Thanks to this technique they were able to train a NN with 152 layers while still having lower complexity than VGGNet
- error rate of 3.57%

The difference between CNNS and multi perceptron networks when classifying Images

	CNNs	MLPs
Architecture	• Convolutional layers that apply filters to the input image. Capturing local spatial patterns. • followed by pooling layers that downsample the feature maps	• MLPs consist of fully connected layers where each neuron is connected to every neuron in the previous and subsequent layers • MLPs do not take into account the spatial structure of the input data. • Too Unnecessary Many Parameters
Parameter Sharing	• Use **parameter sharing, i.e,** the same filter is applied to different regions of the input image. • Smaller number of learnt parameters compared to MLPs • Hence better for huge input data like images	• Each neuron is trained independently • Network learns unique weights for every connection. • Large number of parameters. • Computationally expensive for image data.
Translation Invariance	Due to their use of convolutional and pooling layers, they can detect local patterns regardless of their position in the image.	Sensitive to the position of pixels in the input image, making them less robust to translation or slight changes in input position.

Any Question?

Thank you

UDACITY