Convolutional Neural Networks

Deep Learning

- Quick Review of Previous Topics and NNs
- New Theory Topics
- Discuss Famous MNIST Data Set
- Solve MNIST with a "normal" NN
- Learn about CNN
- Solve MNIST with CNN
- CNN Exercise and Solutions Afterwards

Let’s get started!

Quick Review

Let’s quickly review what we’ve covered so far!

Deep Learning

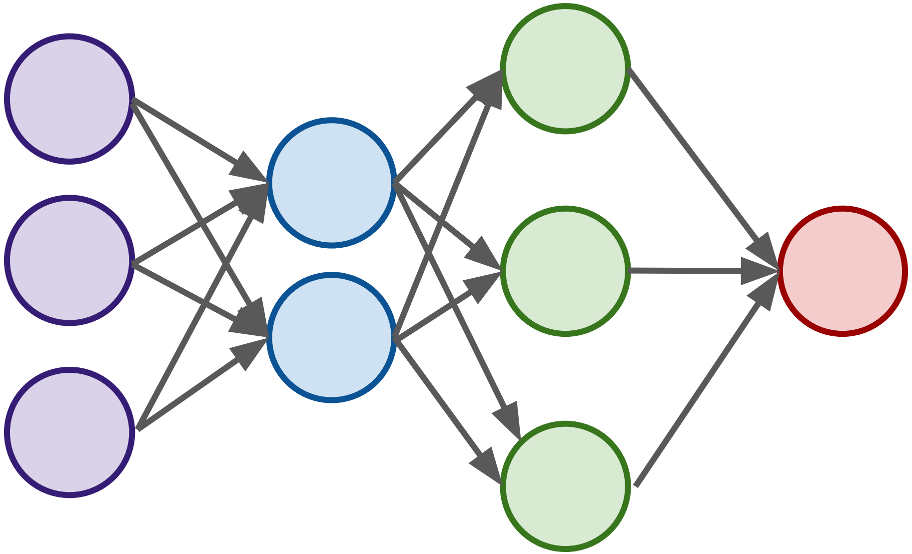
- Single Neuron
- We now understand how to perform a calculation in a neuron
  - $w \cdot x + b = z$
  - $a = \sigma(z)$

Deep Learning

- Activation Functions
  - Perceptrons
  - Sigmoid
  - Tanh
  - ReLU
  - (We'll discuss other functions later on)

Deep Learning

- Connecting Neurons to create a network



Deep Learning

- Neural Network
    - Input Layer
    - Hidden Layers
    - Output Layer
- More layers → More Abstraction

Deep Learning

- To “learn” we need some measurement of error.
- We use a Cost/Loss Function
  - Quadratic
  - Cross-Entropy

Deep Learning

- Once we have the measurement of error, we need to minimize it by choosing the correct weight and bias values.
- We use gradient descent to find the optimal values.

- We can then backpropagate the gradient descent through multiple layers, from the output layer back to the input layer.
- We also know of dense layers, and later on we'll introduce softmax layer.

Deep Learning

- We can then backpropagate the gradient descent through multiple layers, from the output layer back to the input layer.
- We also know of dense layers, and later on we'll introduce softmax layer.

Deep Learning

* Very quick overview of what we know so far!
* We still need to learn a bit more theory before diving into Convolutional Neural Networks!

New Theory Topics

Deep Learning

- We've reviewed the basics, but there are still some theory components we haven't covered yet...

Deep Learning

- Initialization of Weights Options
  - Zeros
    - No Randomness
    - Not a great choice
  - Random Distribution Near Zero
    - Not Optimal
    - Activation Functions Distortion

- Initialization of Weights Options
  - Xavier (Glorot) Initialization
    - Uniform / Normal
  - Draw weights from a distribution with zero mean and a specific variance

$$\mathrm{Var}(W) = \frac{1}{n_{\mathrm{in}}}$$

Deep Learning

- Xavier Initialization

$$Y = W_1 X_1 + W_2 X_2 + \cdots + W_n X_n$$

$$\text{Var}(W_i X_i) = E[X_i]^2 \text{Var}(W_i) + E[W_i]^2 \text{Var}(X_i) + \text{Var}(W_i)\text{Var}(i_i)$$

$$\text{Var}(W_i X_i) = \text{Var}(W_i)\text{Var}(X_i)$$

- Xavier Initialization

$$\mathrm{Var}(Y) = \mathrm{Var}(W_1X_1 + W_2X_2 + \cdots + W_nX_n) = n\mathrm{Var}(W_i)\mathrm{Var}(X_i)$$

$$\mathrm{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{\text{in}}}$$

$$\mathrm{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

Deep Learning - Gradient Descent

- Learning Rate - defines the step size during gradient descent
- Batch Size - batches allow us to use stochastic gradient descent.
  - Smaller → less representative of data
  - Larger → longer training time

Deep Learning - Gradient Descent

- Second-Order Behavior of the gradient descent allows us to adjust our learning rate based off the rate of descent
  - AdaGrad
  - RMSProp
  - Adam

Deep Learning - Gradient Descent

- This allows us to start with larger steps and then eventually go to smaller step sizes.
- Adam allows this change to happen automatically.

Deep Learning

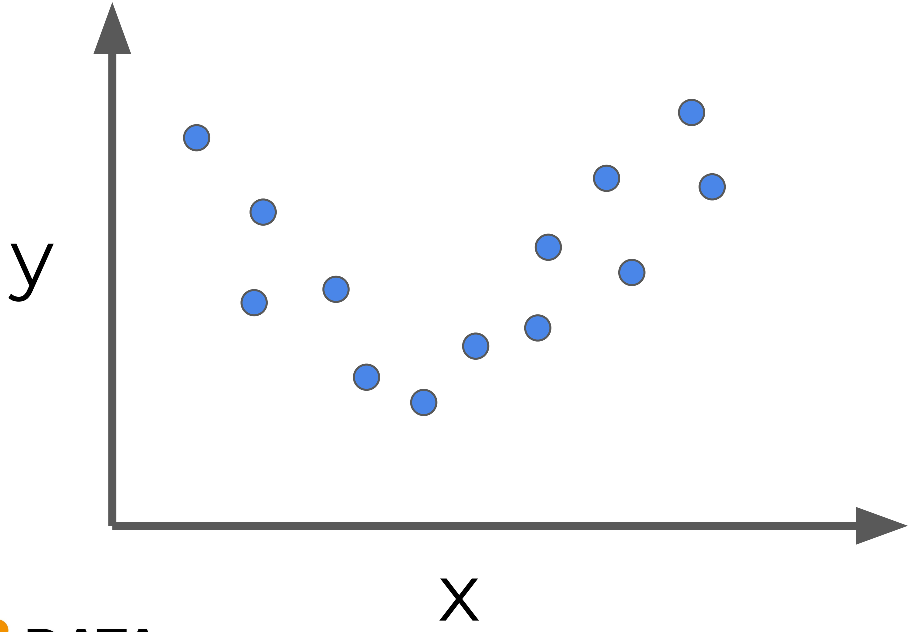
- Unstable / Vanishing Gradients
    - As you increase the number of layers in a network, the layers towards the input will be affected less by the error calculation occurring at the output as you go backwards through the network

Deep Learning

- Unstable / Vanishing Gradients
    - Initialization and Normalization will help us mitigate these issues.
    - We'll discuss vanishing gradients again in more detail when discussing Recurrent Neural Networks.

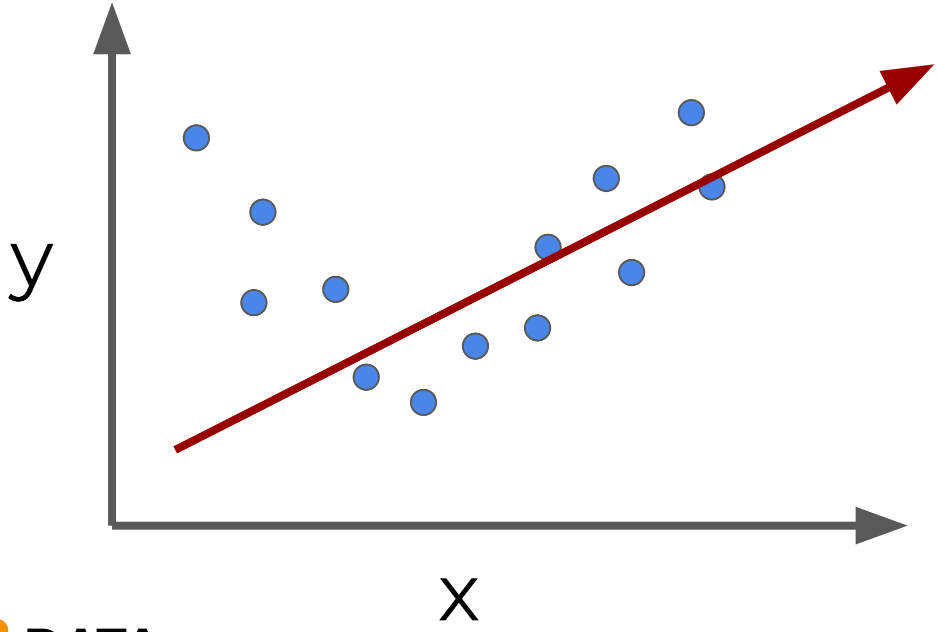
Deep Learning

- Overfitting vs Underfitting a Model



Training Data

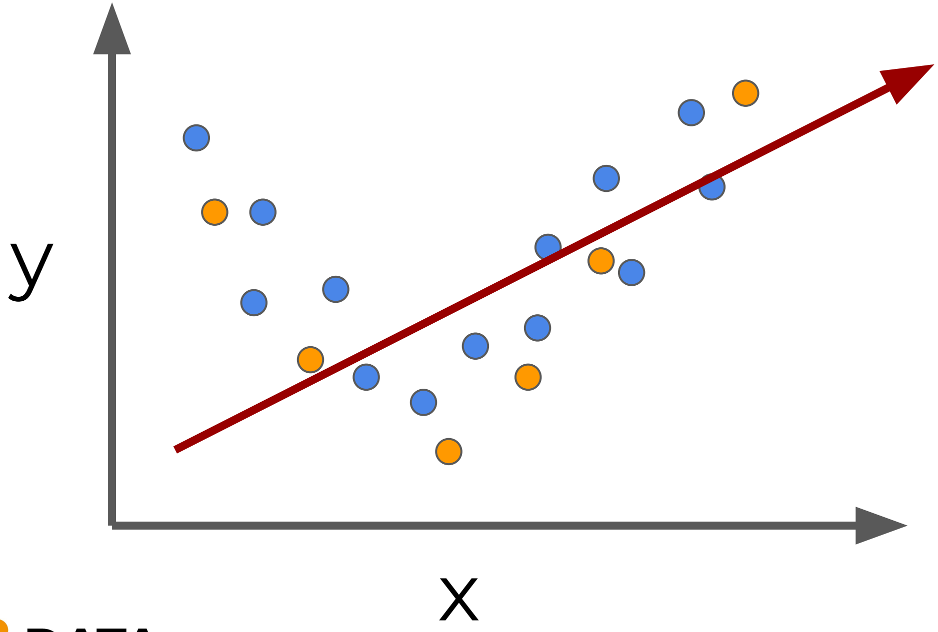
- Model Underfitting



Fitted Model on Training Data

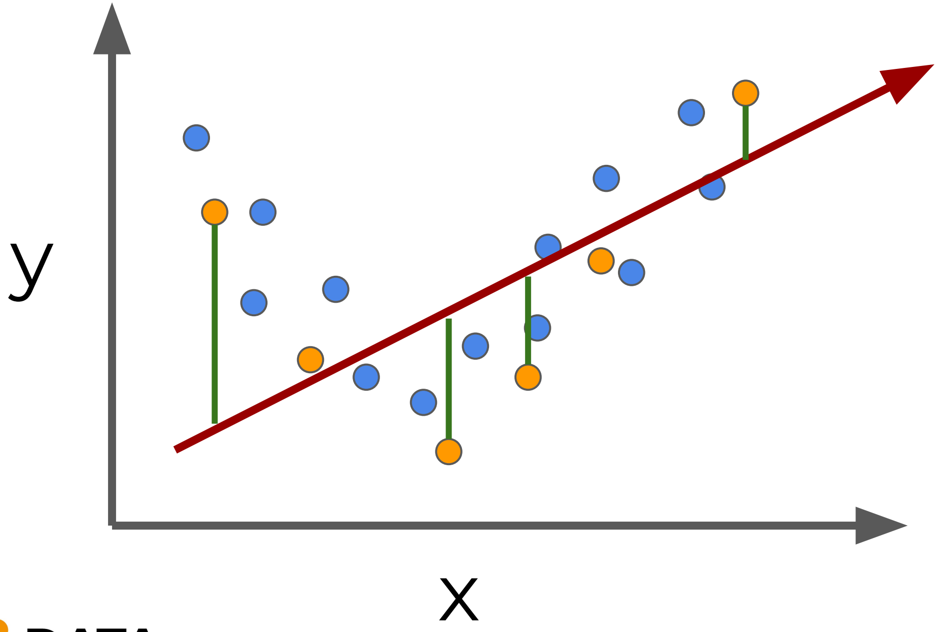
Deep Learning

- Model Underfitting



Get error on test data.

- Model Underfitting

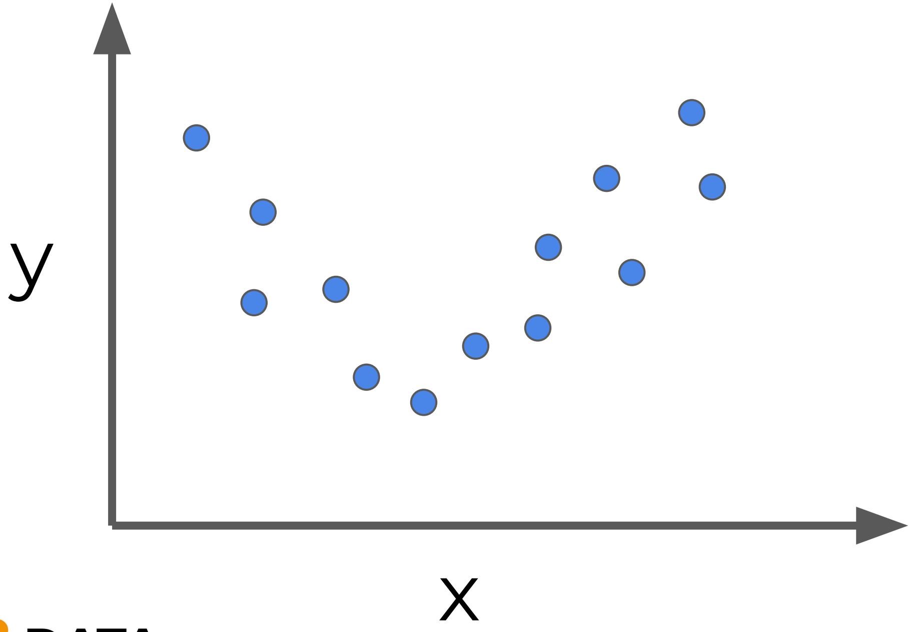


Get error on test data.

High error on training and test data

Deep Learning

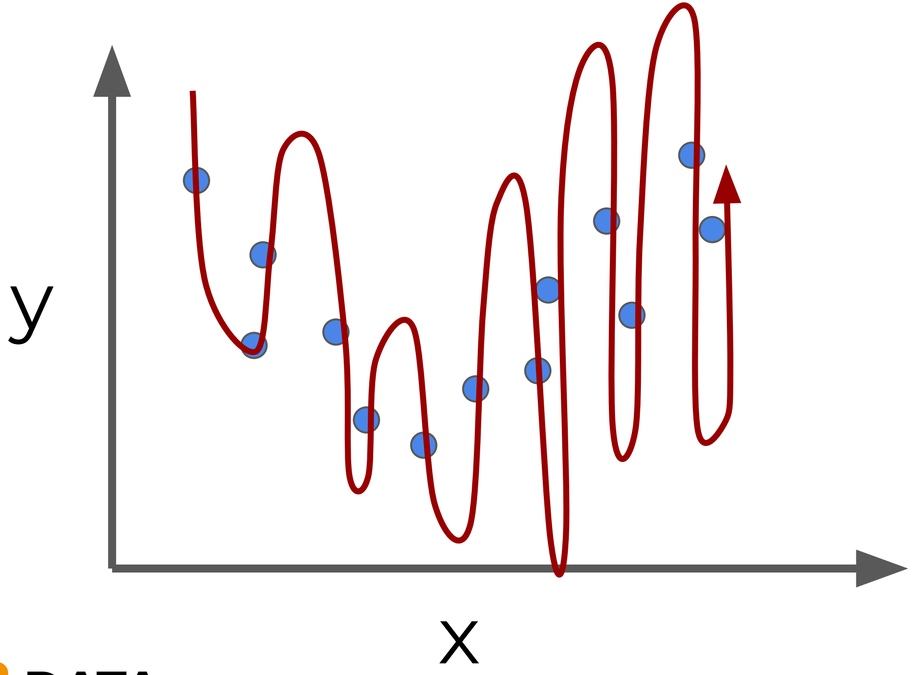
- Overfitting a Model



Training Data

Deep Learning

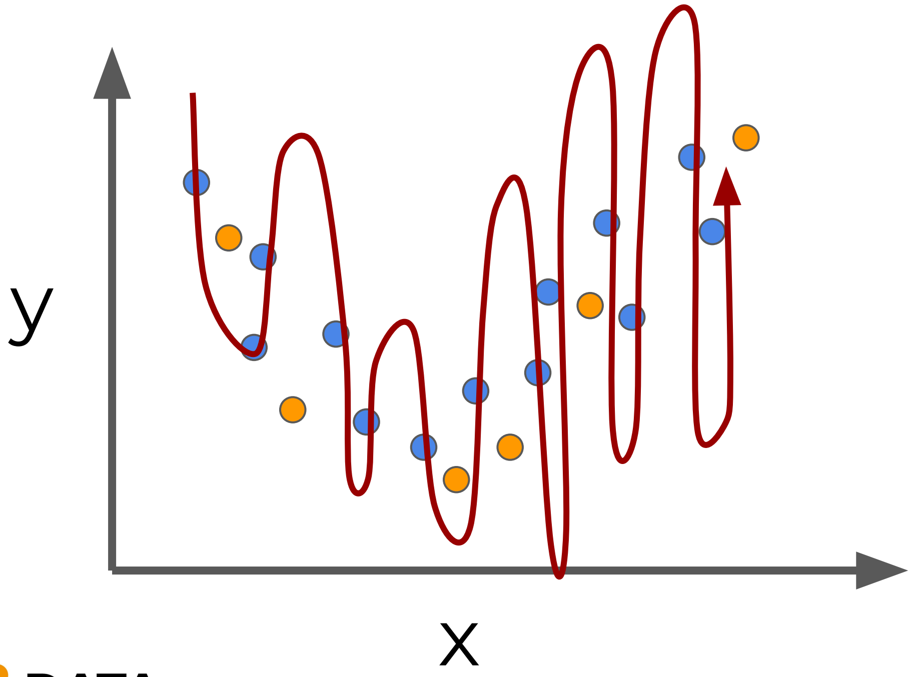
- Overfitting a Model



Very low error on training data!

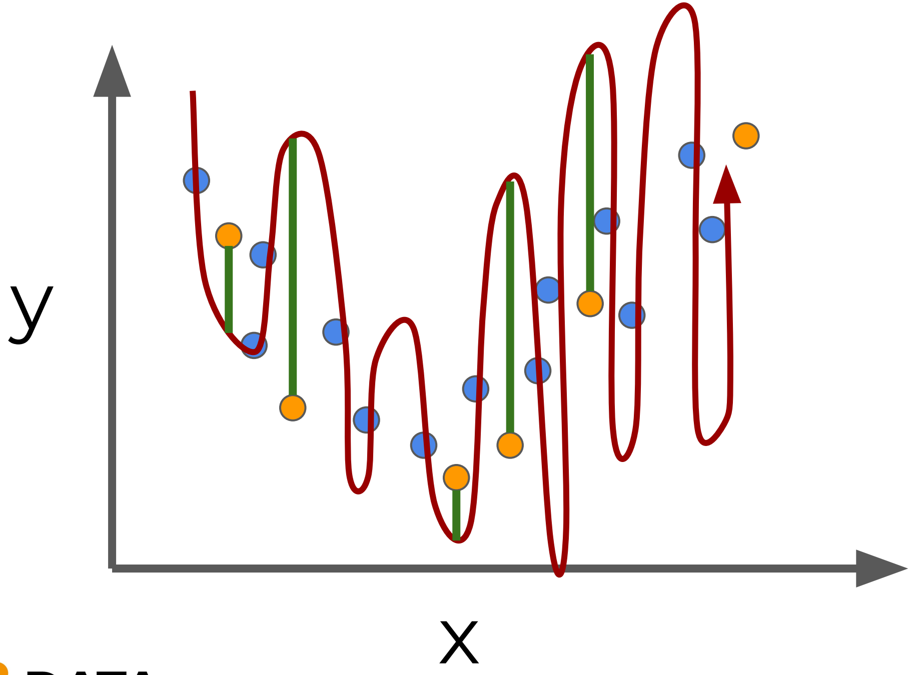
Deep Learning

- Overfitting a Model



But large error on test data.

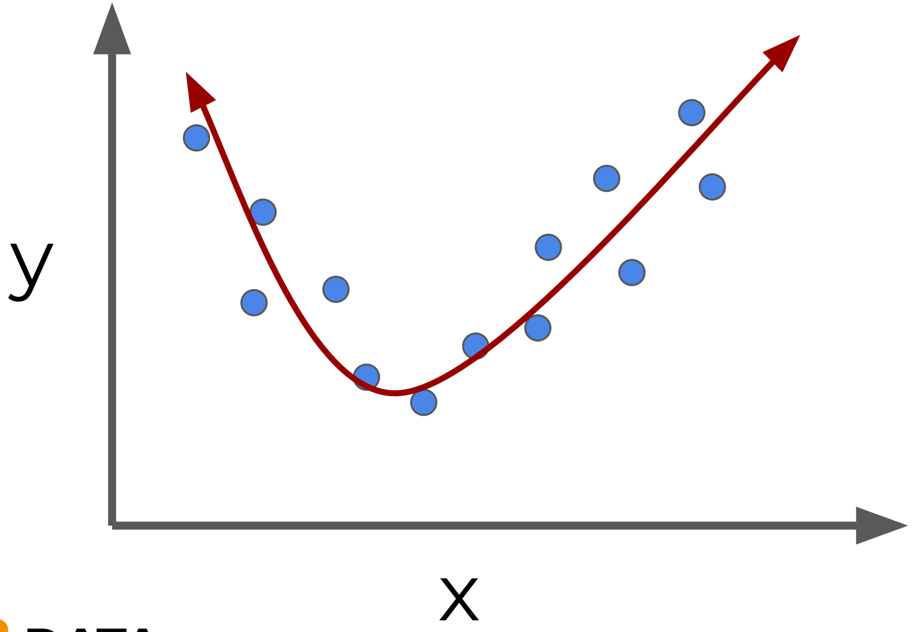
- Overfitting a Model



But large error on test data.

Deep Learning

- Need to strike a balance



Deep Learning

- With potentially hundreds of parameters in a deep learning neural network, the possibility of overfitting is very high!
- There are a few ways to help mitigate this issue.

Deep Learning

- L1/L2 Regularization
  - Adds a penalty for larger weights in the model
  - Not unique to neural networks

- Dropout
  - Unique to neural networks
  - Remove neurons during training randomly
  - Network doesn’t over rely on any particular neuron

Deep Learning

- Expanding Data
  - Artificially expand data by adding noise, tilting images, adding low white noise to sound data, etc...

Deep Learning

- We still have more theory to learn, such as pooling layers, convolutional layers, etc…
- But we'll wait until we begin to build CNNs to cover those!
- Let's explore the famous MNIST data set, a must know for CNN!

MNIST Data

Deep Learning

- A classic data set in Deep Learning is the MNIST data set.
- Let’s quickly cover some basics about it since we’ll be using it quite frequently during this section of the course!

Deep Learning

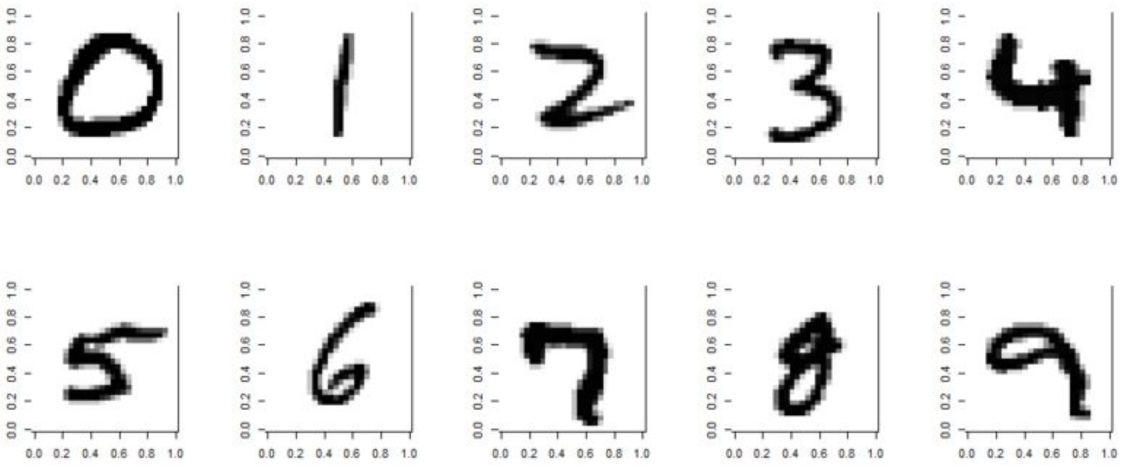
- Fortunately this data is easy to access with TensorFlow. TF has:
  - 55,000 training images
  - 10,000 test images
  - 5,000 validation images

Deep Learning

- The MNIST data set contains handwritten single digits from 0 to 9

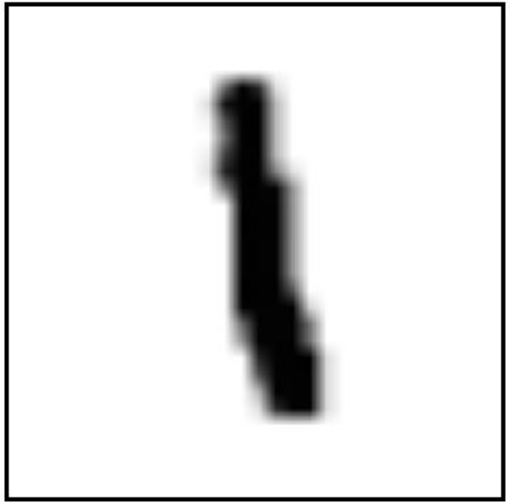
Deep Learning

- A single digit image can be represented as an array



Deep Learning

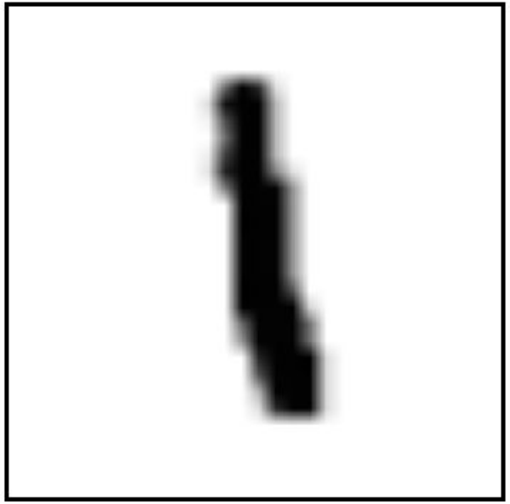
- Specifically, 28 by 28 pixels



$$\cong \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

Deep Learning

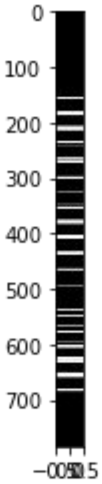
- The values represent the grayscale im



$$\simeq \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

Deep Learning

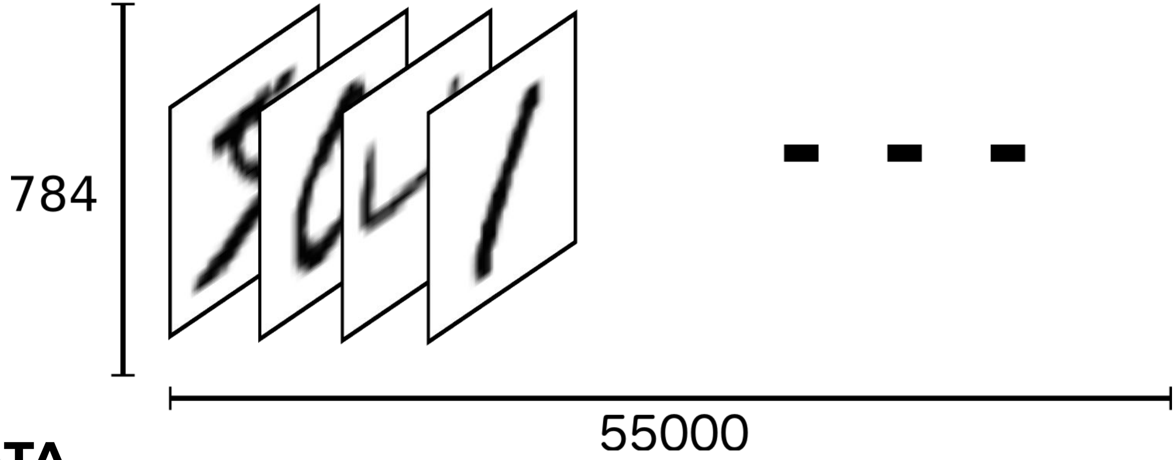
- We can flatten this array to a 1-D vector of 784 numbers. Either (781,1) or (1,781) is fine, as long as the dimensions are consistent.



- Flattening out the image ends up removing some of the 2-D information, such as the relationship of a pixel to its neighboring pixels.
- For now, we'll ignore this, but come back to it later when we discuss CNN in depth!

Deep Learning

- We can think of the entire group of the 55,000 images as a tensor (an n-dimensional array)



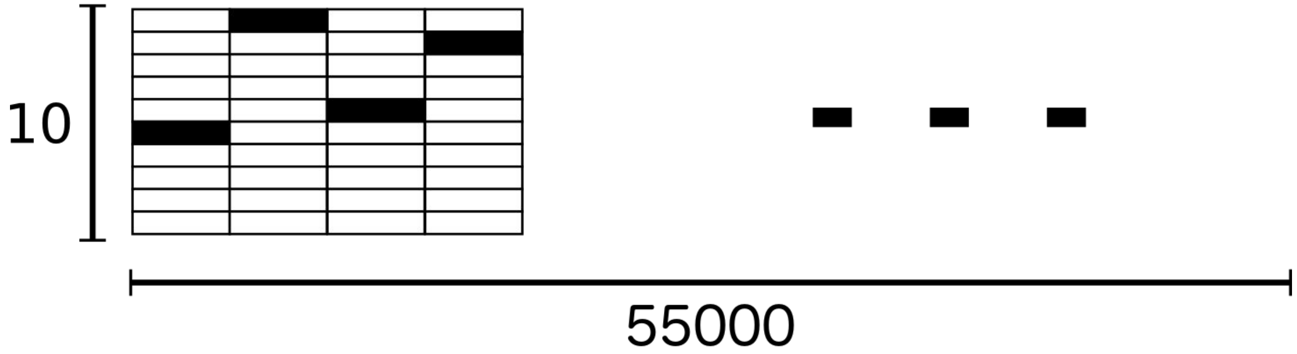
* For the labels we'll use One-Hot Encoding.
* This means that instead of having labels such as "One", "Two", etc... we'll have a single array for each image.

Deep Learning

- The label is represented based off the index position in the label array.
- The corresponding label will be a 1 at the index location and zero every where else.
- For example, 4 would have this label array:
    - [0,0,0,0,1,0,0,0,0,0]

Deep Learning

- As a result, the labels for the training data ends up being a large 2-d array (10,55000):



MNIST Data
“Basic” Approach

Deep Learning

- Before we dive into using CNN on the MNIST data set we'll use a more basic Softmax Regression Approach.
- Let's quickly go over this method (it's very similar to the methods used in the previous sections)

- A Softmax Regression returns a list of values between 0 and 1 that add up to one.
- We can use this as a list of probabilities!

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

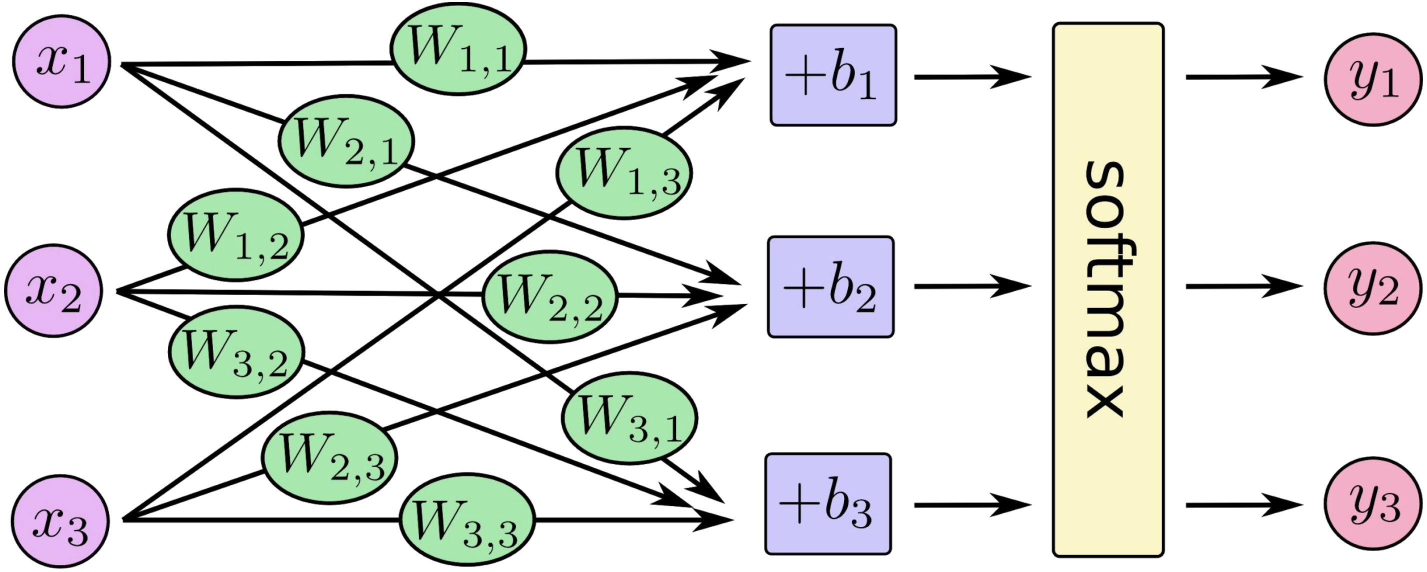
Deep Learning

- We'll use Softmax as our activation function.

$$z_i = \sum_j W_{i,j} x_j + b_i$$

$$y = \text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Our Network


x_1
x_2
x_3
$W_{1,1}$
$W_{2,1}$
$W_{1,3}$
$W_{1,2}$
$W_{2,2}$
$W_{3,2}$
$W_{3,1}$
$W_{2,3}$
$W_{3,3}$
$+b_1$
$+b_2$
$+b_3$
softmax
y_1
y_2
y_3

Deep Learning

As an equation

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

As an equation

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Deep Learning

- Let’s implement this with Python and TensorFlow!

Convolutional Neural Networks

Deep Learning

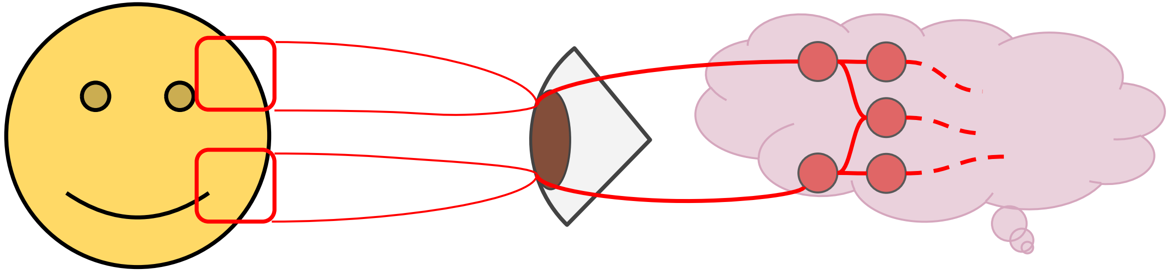
- We just solved the MNIST task with a very simple linear approach.
- Let’s explore a much better approach using Convolutional Neural Network.

Deep Learning

- Just like the simple perceptron, CNNs also have their origins in biological research.
- Hubel and Wiesel studied the structure of the visual cortex in mammals, winning a Nobel Prize in 1981.

Deep Learning

- Their research revealed that neurons in the visual cortex had a small local receptive field.

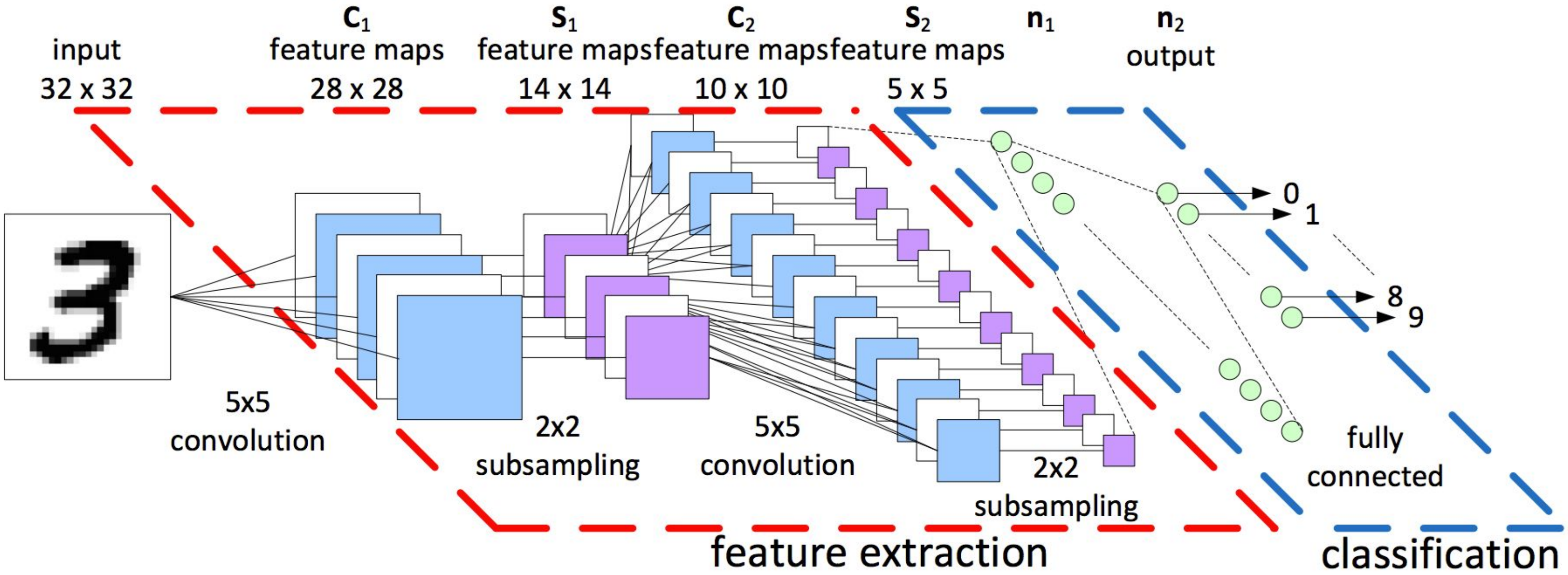


Deep Learning

- This idea then inspired an ANN architecture that would become CNN
- Famously implemented in the 1998 paper by Yann LeCun et al.
- The LeNet-5 architecture was first used to classify the MNIST data set.

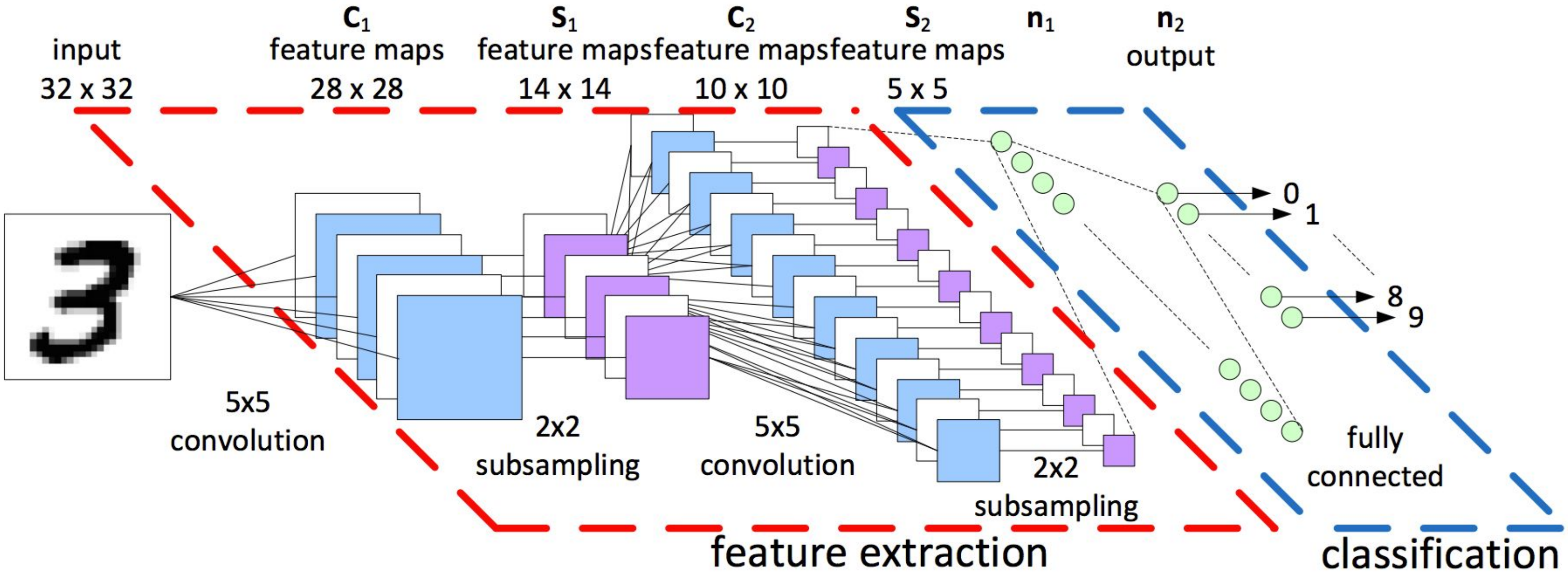
Deep Learning

- When learning about CNNs you'll often see a diagram like this:



Deep Learning

- Let's break down the various aspects of a CNN seen here:



Deep Learning

- Tensors
- DNN vs CNN
- Convolutions and Filters
- Padding
- Pooling Layers
- Review Dropout

Deep Learning

- Recall that Tensors are N-Dimensional Arrays that we build up to:
  - Scalar - 3
  - Vector - [3,4,5]
  - Matrix - [[3,4] , [5,6] , [7,8]]
  - Tensor - [[[1, 2] , [3, 4]],
    [[5, 6] , [7, 8]]]

Deep Learning

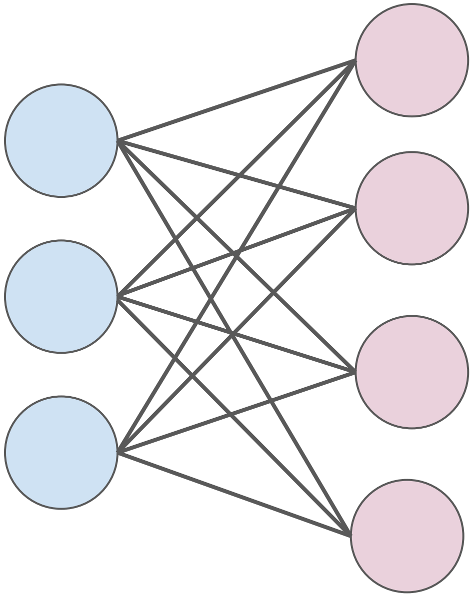
- Tensors make it very convenient to feed in sets of images into our model - (I,H,W,C)
    - I : Images
    - H: Height of Image in Pixels
    - W: Width of Image in Pixels
    - C: Color Channels: 1-Grayscale, 3-RGB

Deep Learning

- Now let’s explore the difference between a Densely Connected Neural Network and a Convolutional Neural Network.
- Recall that we’ve already been able to create DNNs with tf.estimator API.

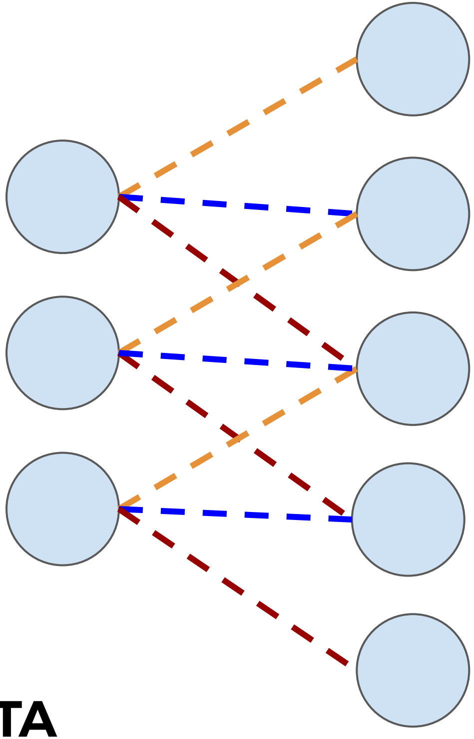
Deep Learning

- Densely Connected layer



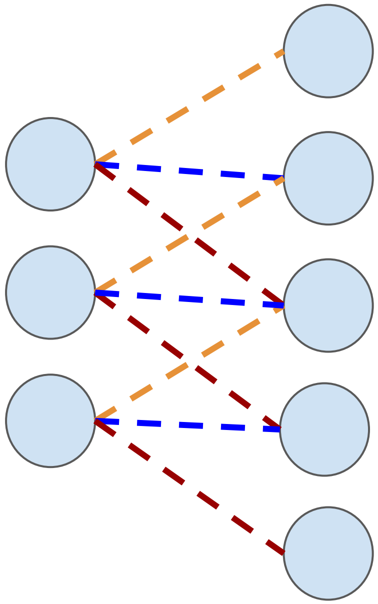
Deep Learning

- Convolutional Layer



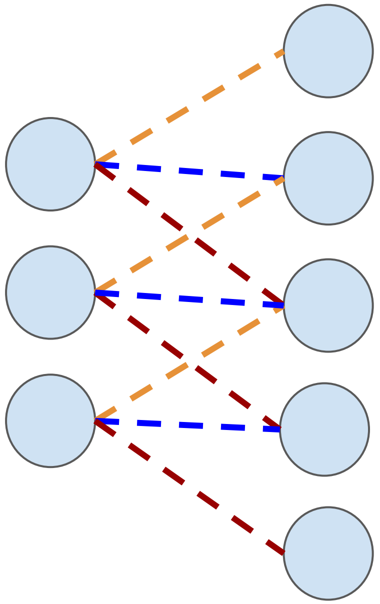
Deep Learning

- Each unit is connected to a smaller number of nearby units in next layer.



Deep Learning

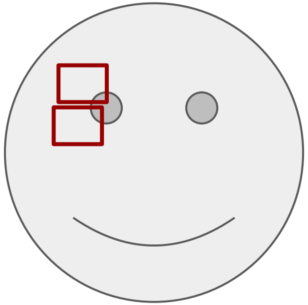
- So why bother with a CNN instead of a DNN?



Deep Learning

- The MNIST dataset was 28 by 28 pixels (784 total)
- But most images are at least 256 by 256 or greater, (<56k total)!
- This leads to too many parameters, unscalable to new images.

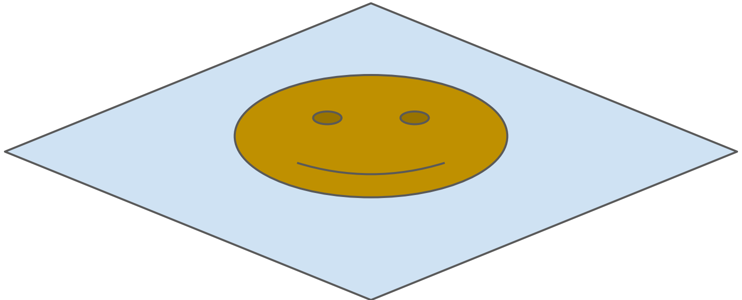
- Convolutions also have a major advantage for image processing, where pixels nearby to each other are much more correlated to each other for image detection.



- Each CNN layer looks at an increasingly larger part of the image.
- Having units only connected to nearby units also aids in *invariance*.
- CNN also helps with regularization, limiting the search of weights to the size of the convolution.

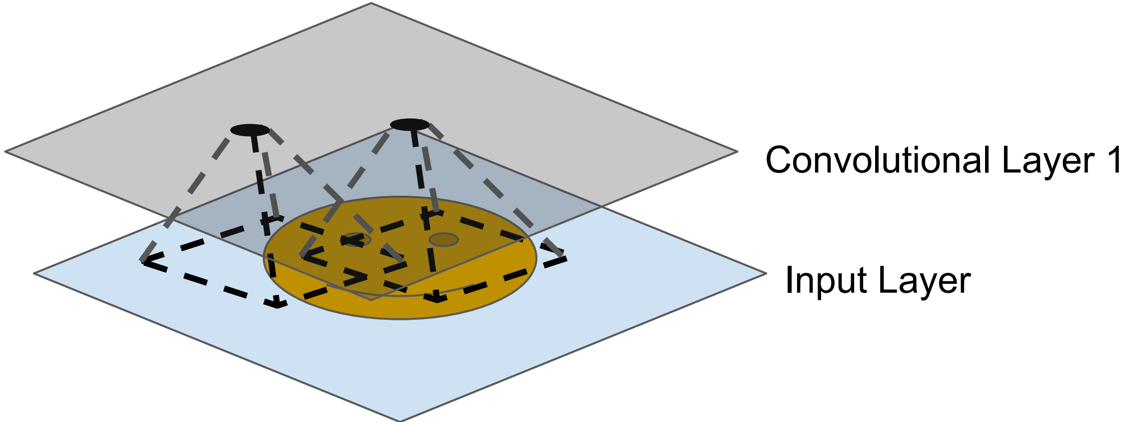
Deep Learning

- Let's explore how the convolutional neural network relates to image recognition!
- We start with the input layer, the image itself.



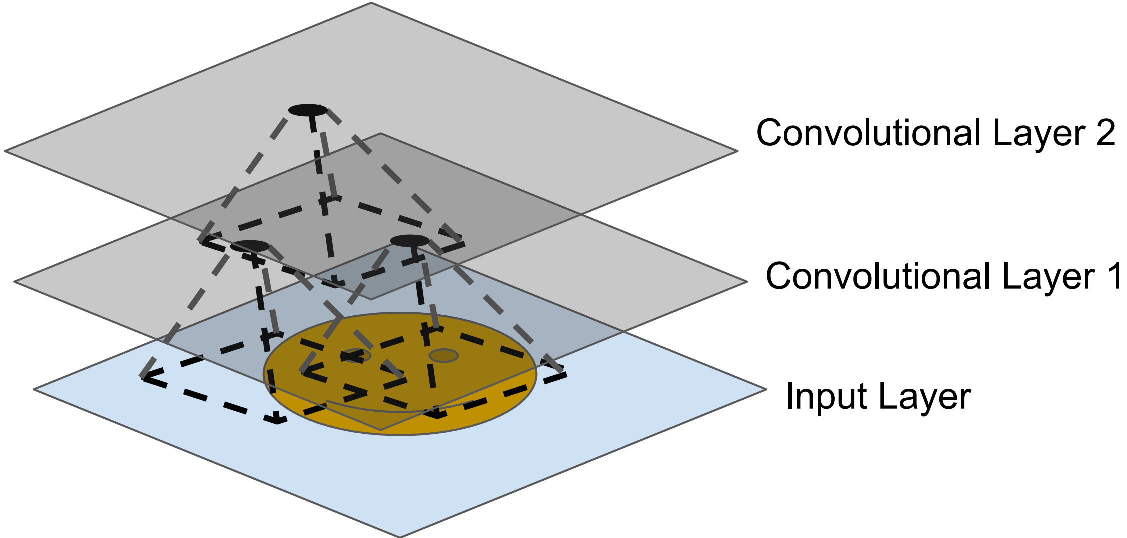
Deep Learning

- Convolutional layers are only connected to pixels in their respective fields.



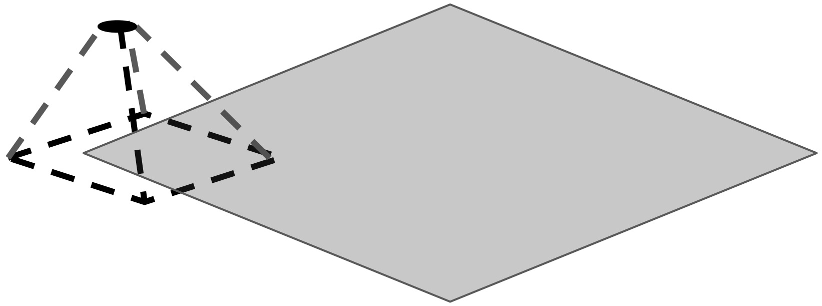
Deep Learning

- Convolutional layers are only connected to pixels in their respective fields.



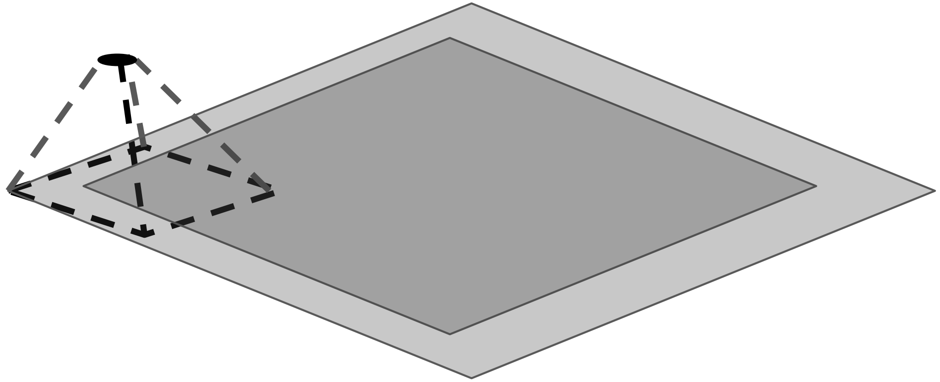
Deep Learning

- We run into a possible issue for edge neurons! There may not be an input there for them.



Deep Learning

* We can fix this by adding a “padding” of zeros around the image.

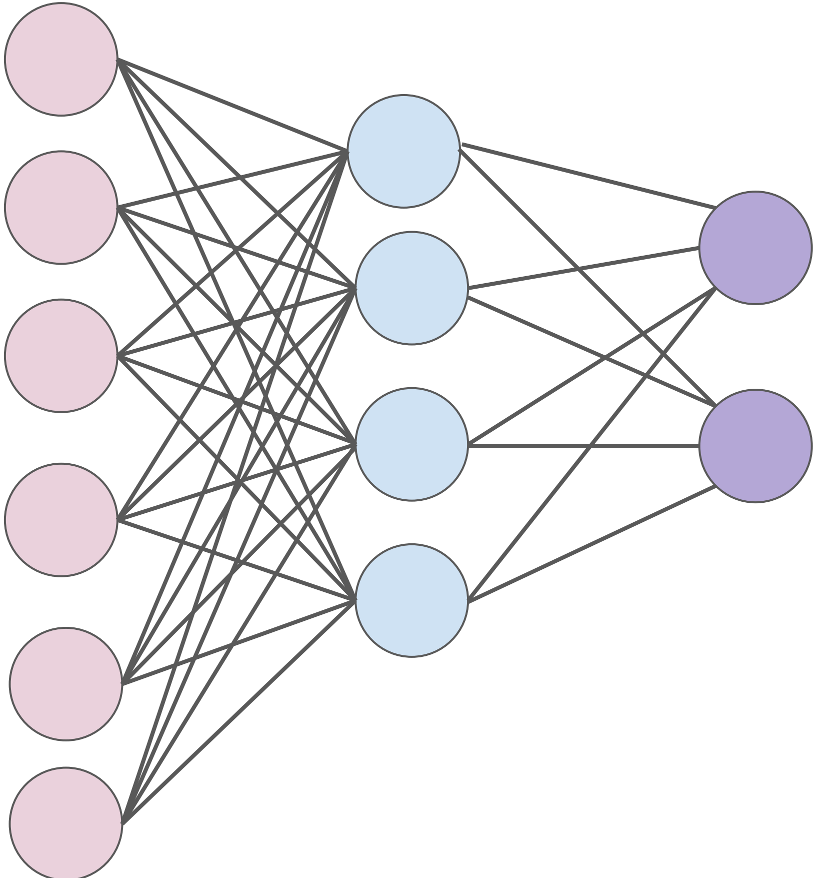


Deep Learning

- Let's walk through 1-D Convolution in more detail, then expand this idea to 2-D Convolution.
- Let's revisit our DNN and convert it to a CNN.

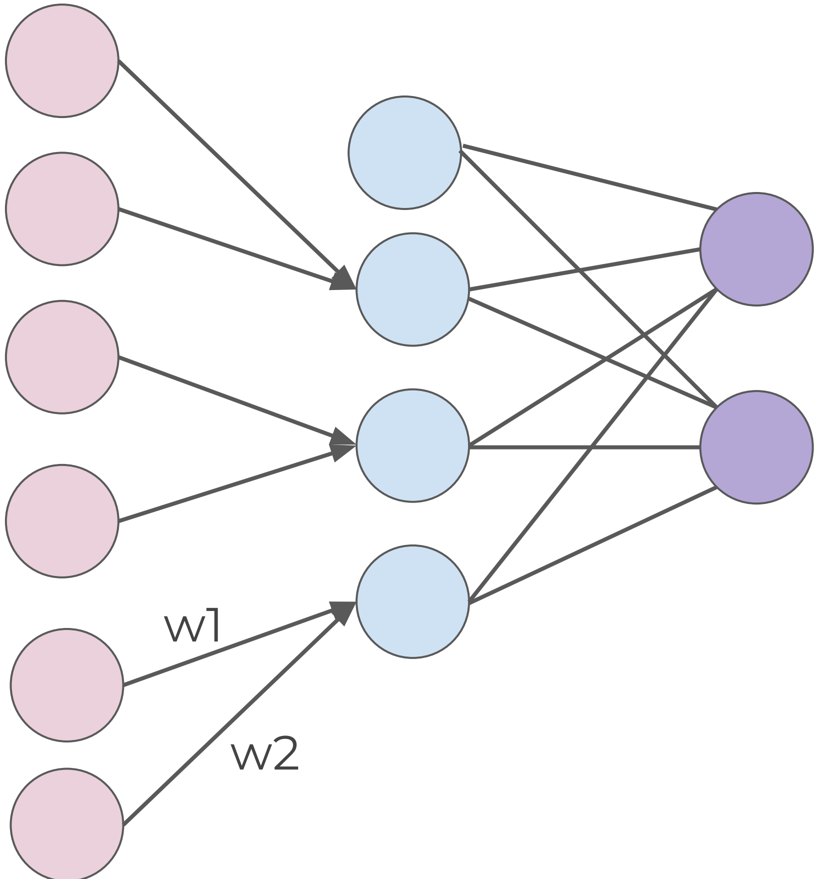
Deep Learning

- A DNN



Deep Learning

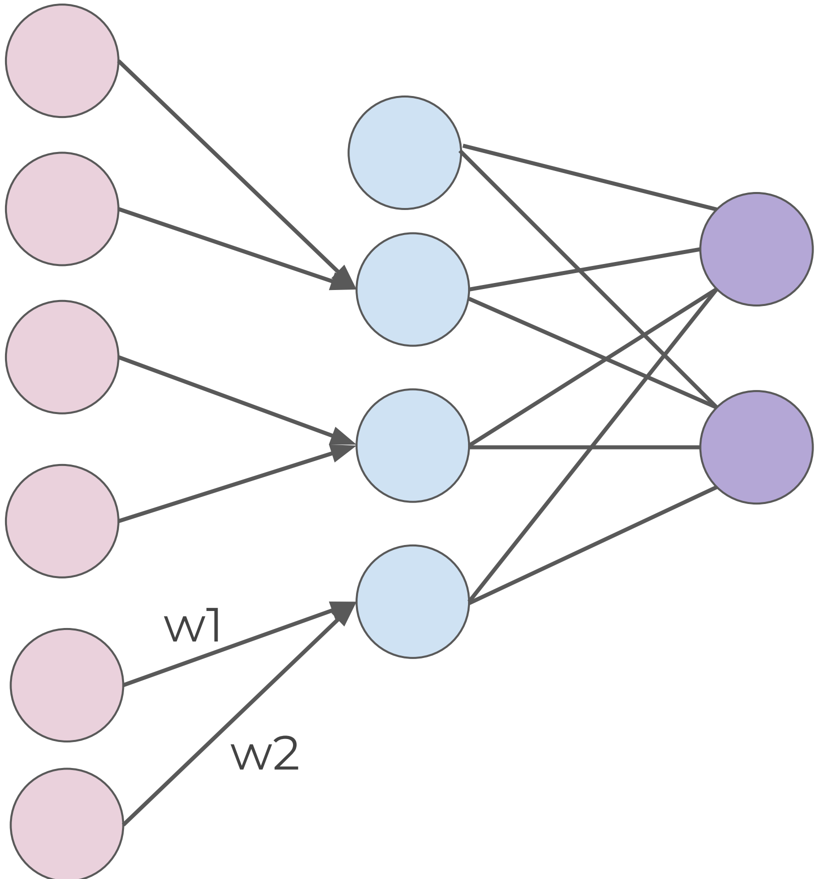
1-D Convolution



PIERIAN DATA

1-D Convolution

We can treat these weights as a filter.



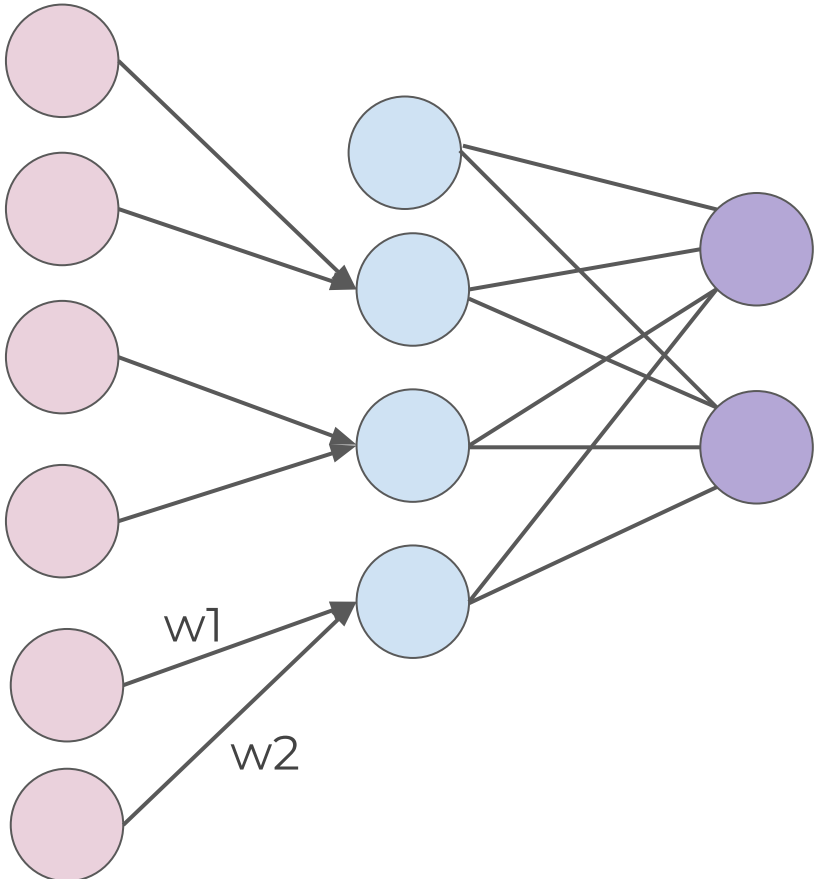
1-D Convolution

$$y = w_1 x_1 + w_2 x_2$$

If $(w_1, w_2) = (1,-1)$ Then
$$y = x_1 - x_2$$

When is y at a maximum?

$$(x_1, x_2) = (1,0)$$



Deep Learning

1-D Convolution

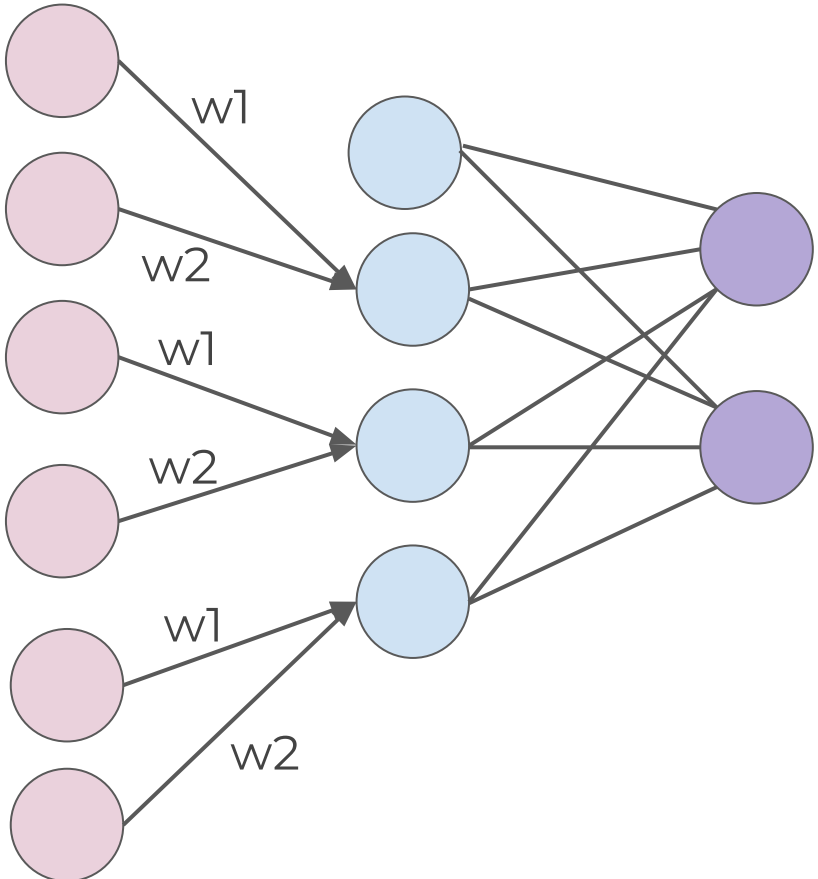
$$y = w_1 x_1 + w_2 x_2$$

If $(w_1, w_2) = (1,-1)$ Then

$$y = x_1 - x_2$$

When is y at a maximum?

$$(x_1, x_2) = (1,0)$$

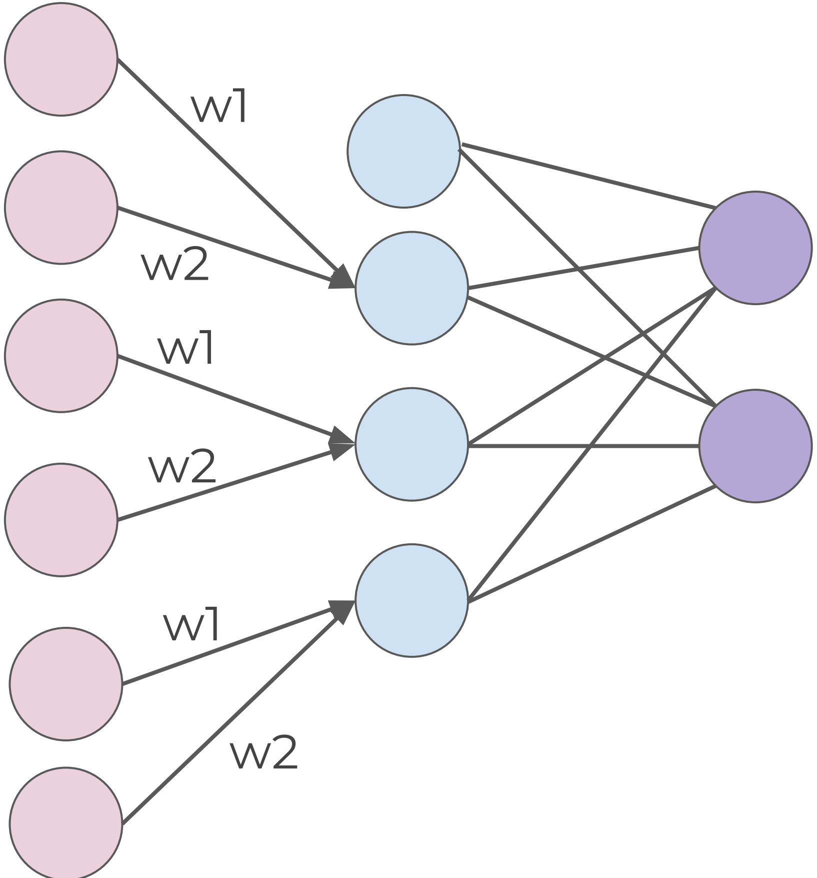


Deep Learning

1-D Convolution

We now have a set of weights that can act as a filter for edge detection!

We can then expand this idea to multiple filters.



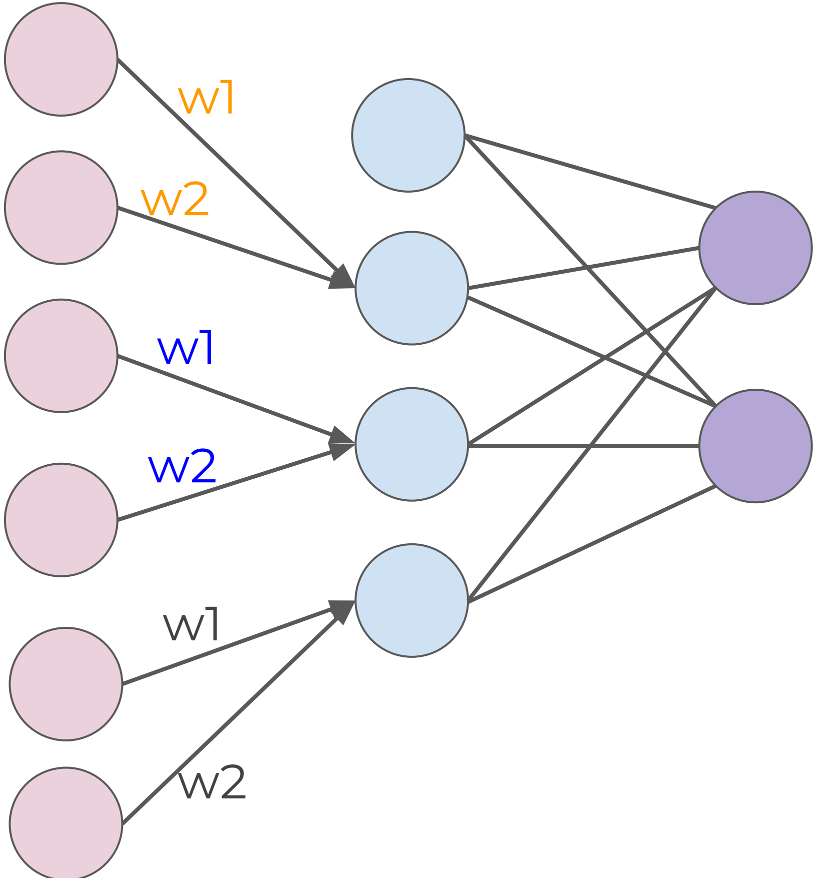
PIERIAN DATA

1-D Convolution

Filters: 1

Filter Size: 2

Stride: 2



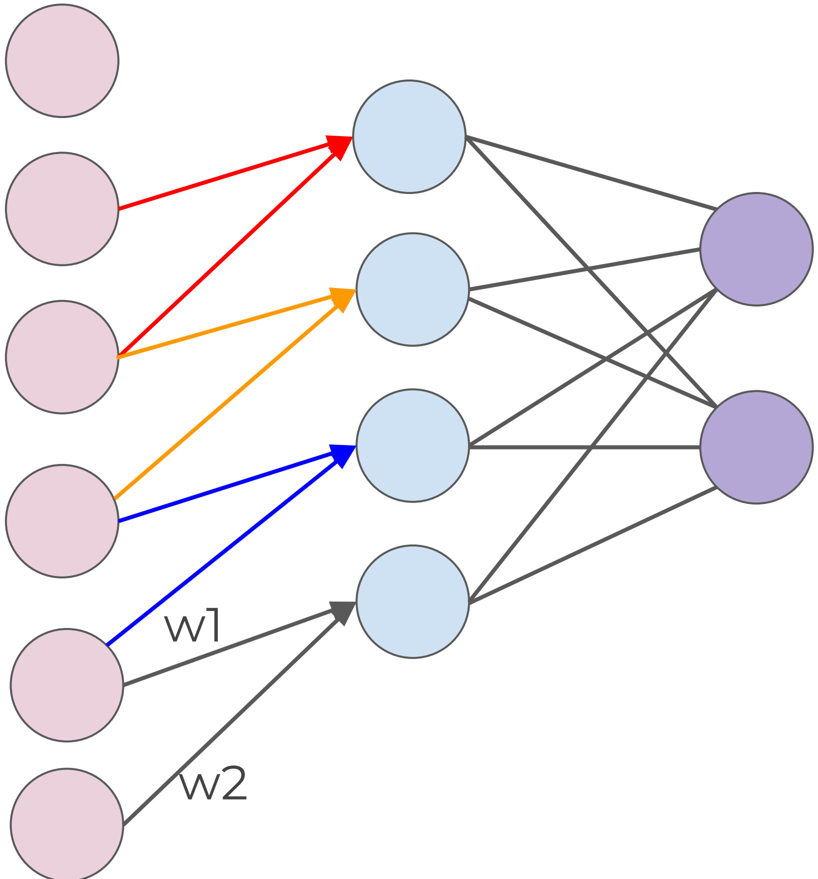
Deep Learning

1-D Convolution

Filters: 1

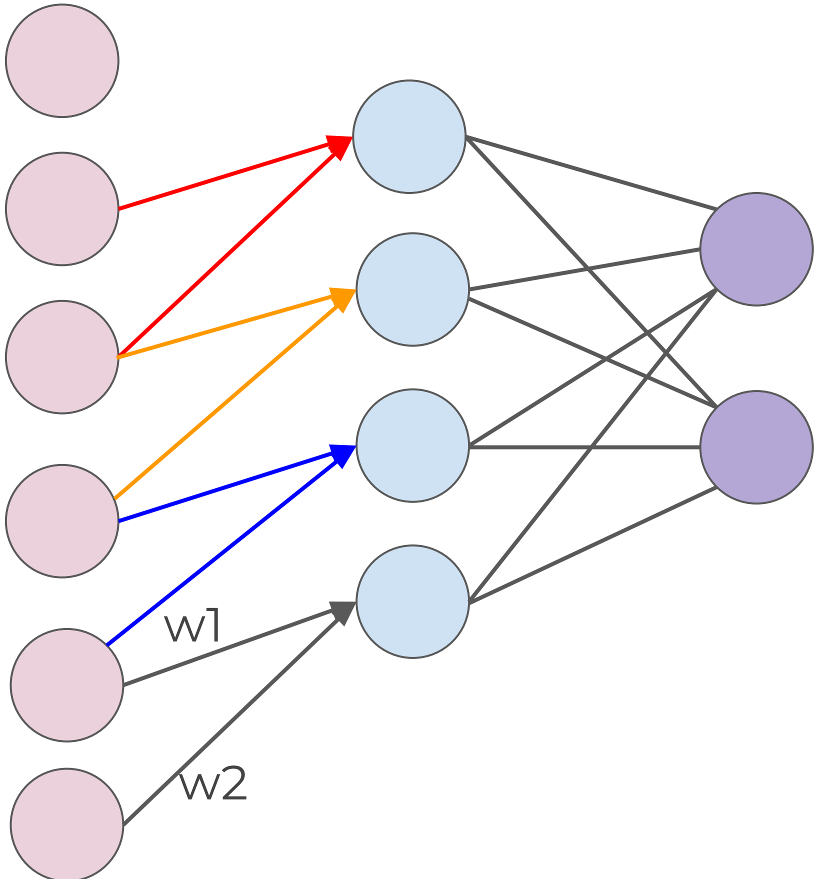
Filter Size: 2

Stride: 1 (1 Unit at a time)



1-D Convolution

Remember that we can add zero padding to include more edge pixels.



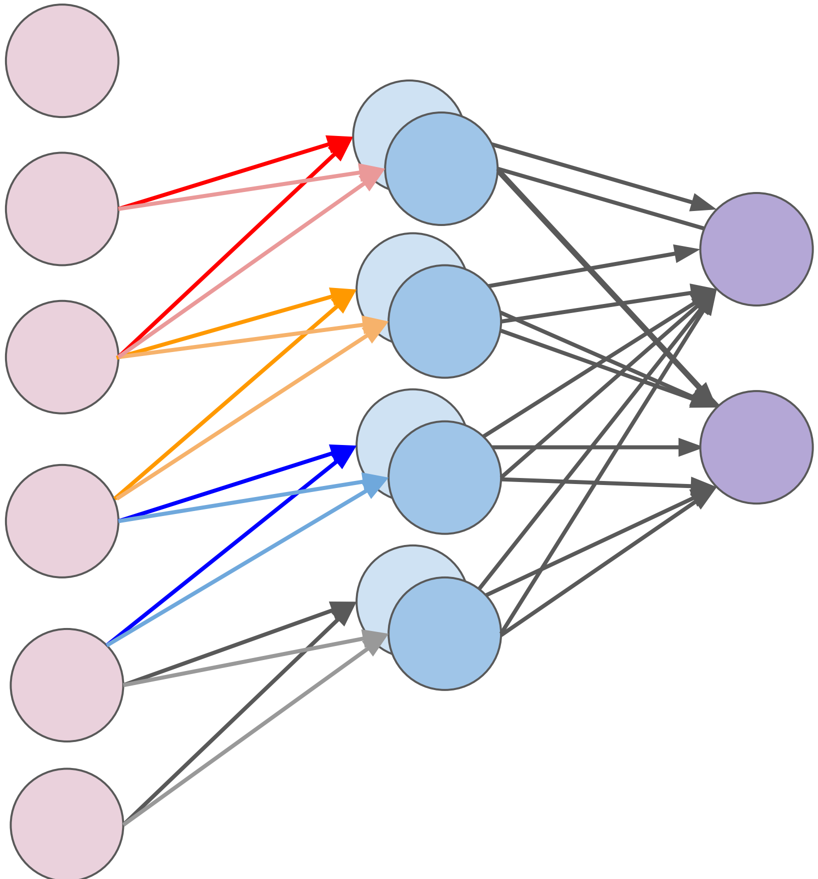
Deep Learning

1-D Convolution

Filters: 2

Filter Size: 2

Stride: 1 (1 Unit at a time)

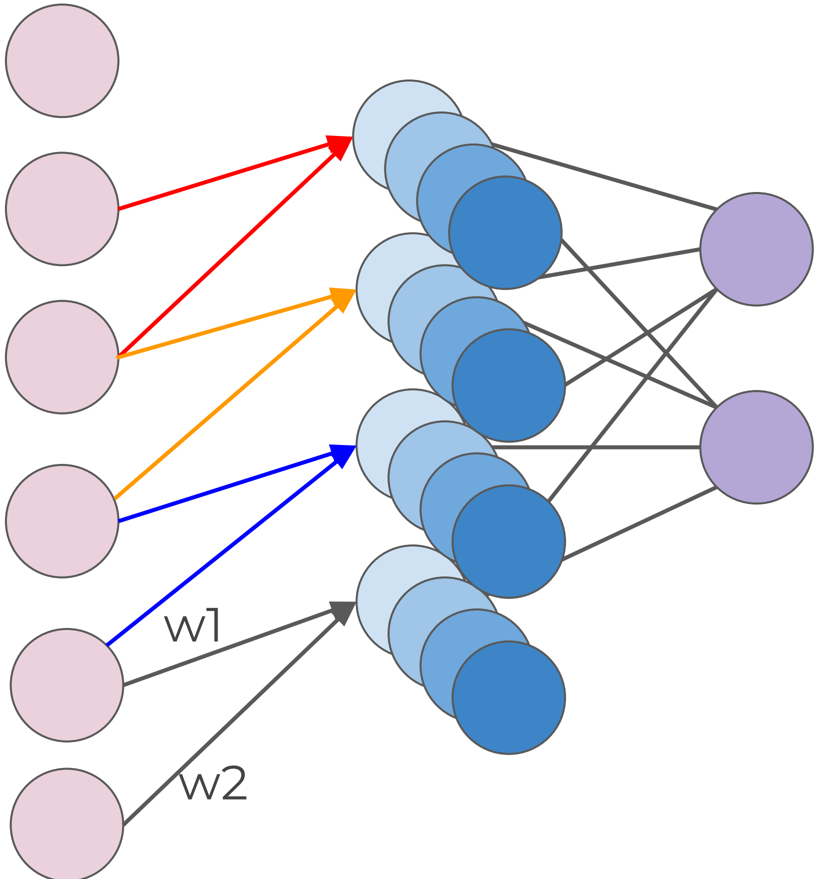


1-D Convolution

Filters: 4

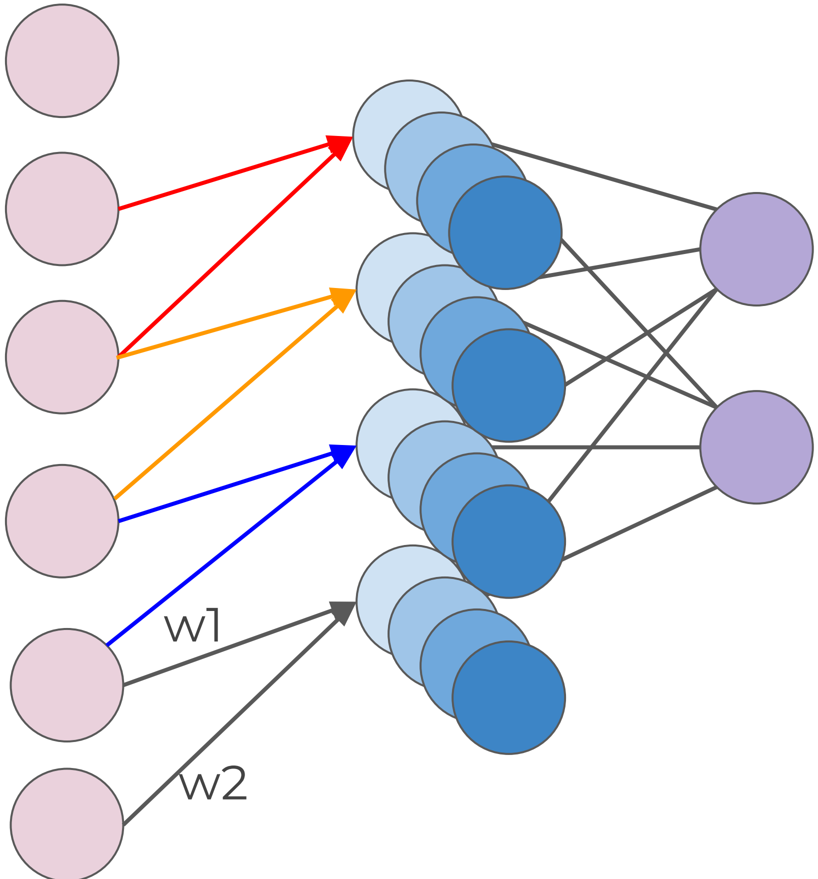
Filter Size: 2

Stride: 1 (1 Unit at a time)



1-D Convolution

Each filter is detecting a different feature



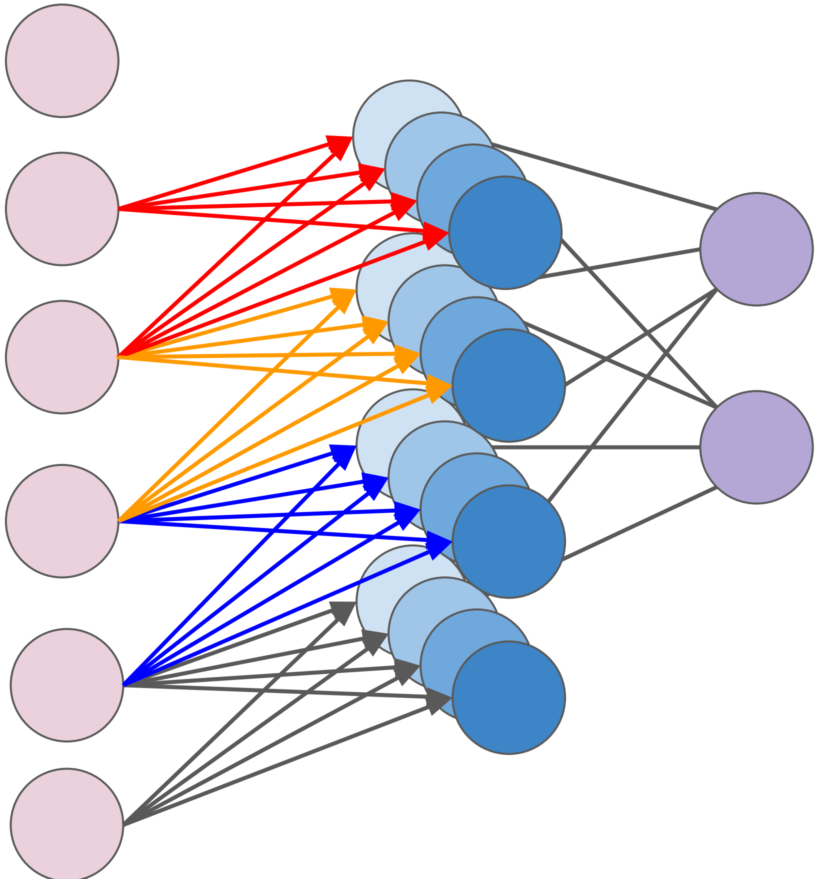
Deep Learning

1-D Convolution

Filters: 4

Filter Size: 2

Stride: 1 (1 Unit at a time)

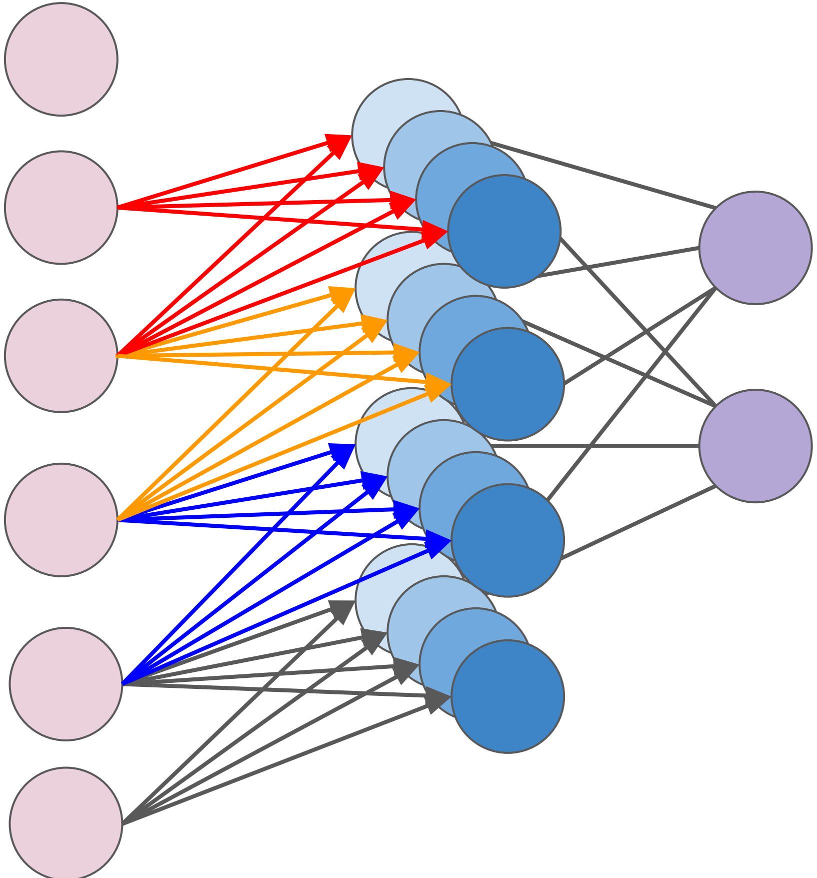


PIERIAN DATA

Deep Learning

1-D Convolution

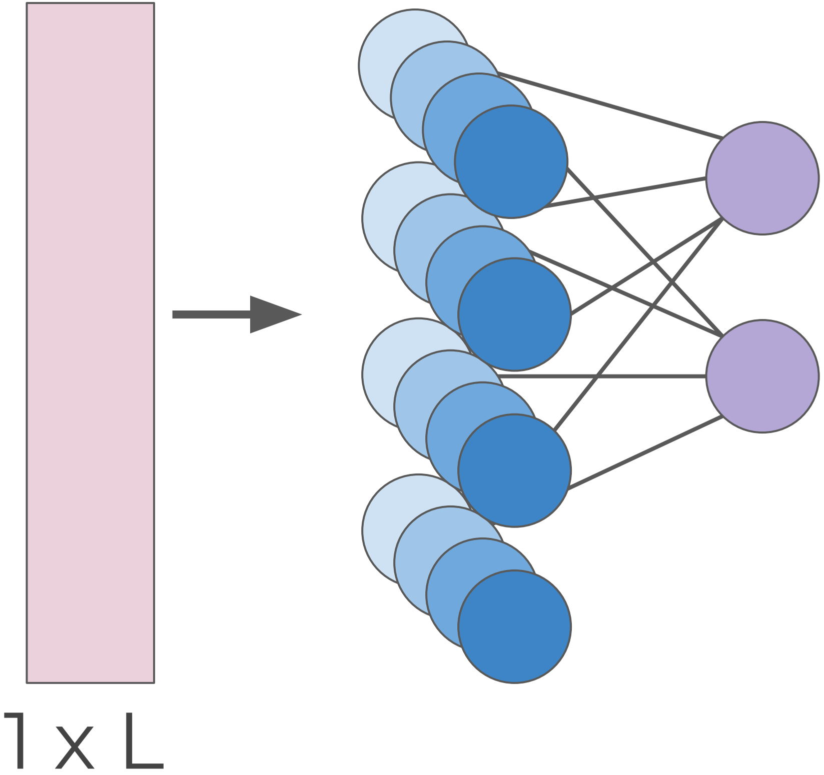
For simplicity, we begin to describe and visualize these sets of neurons as blocks instead



Deep Learning

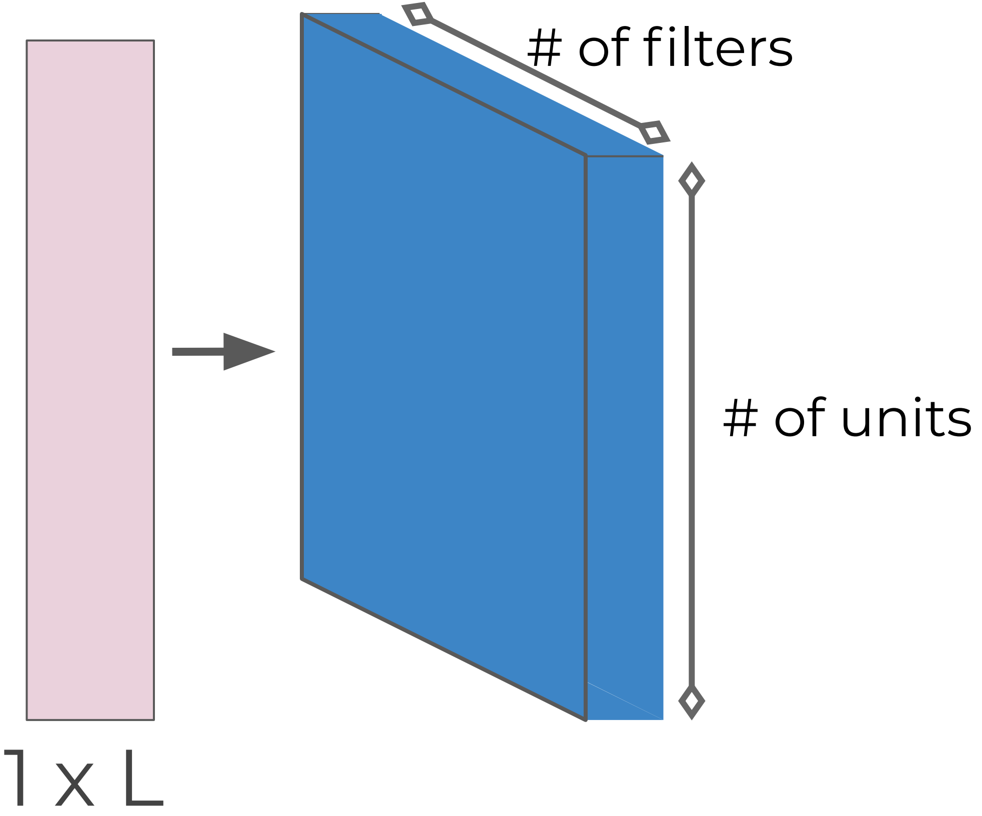
1-D Convolution

For simplicity, we begin to describe and visualize these sets of neurons as blocks instead



1-D Convolution

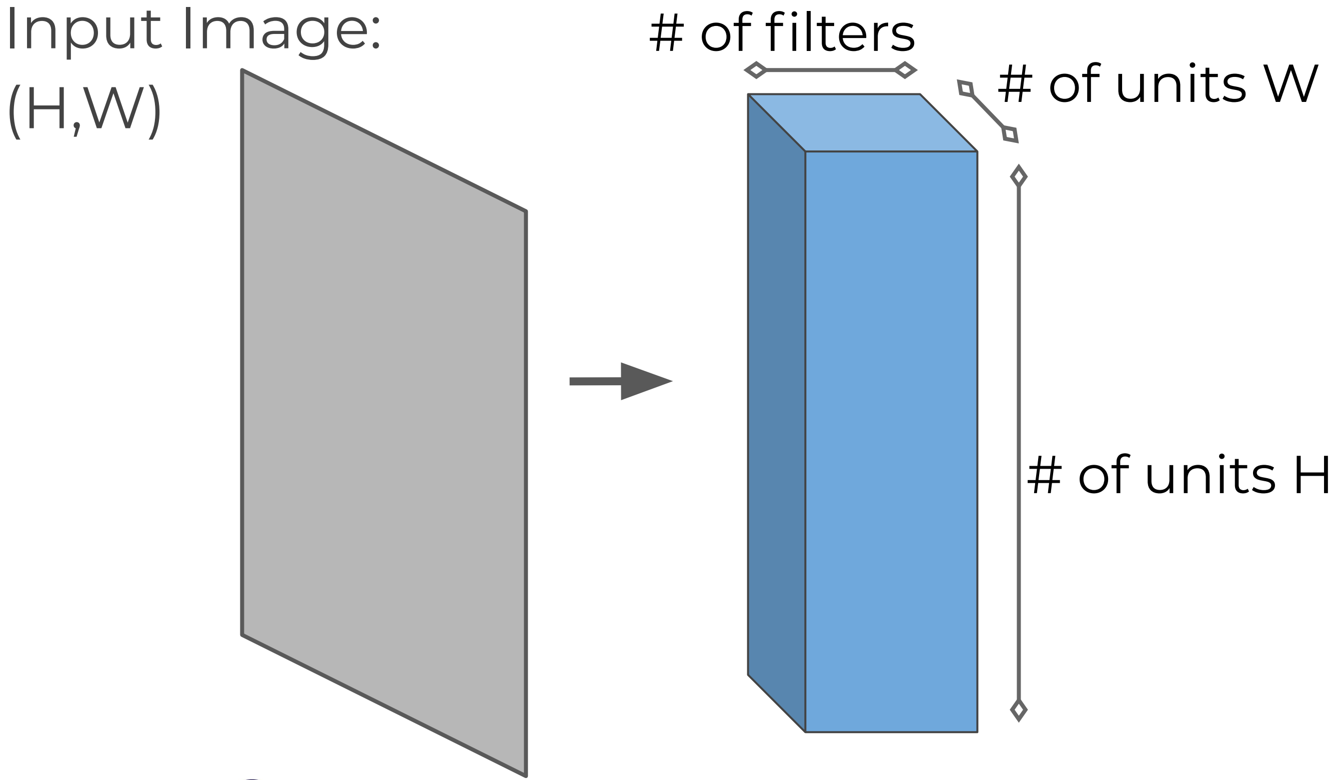
For simplicity, we begin to describe and visualize these sets of neurons as blocks instead



Deep Learning

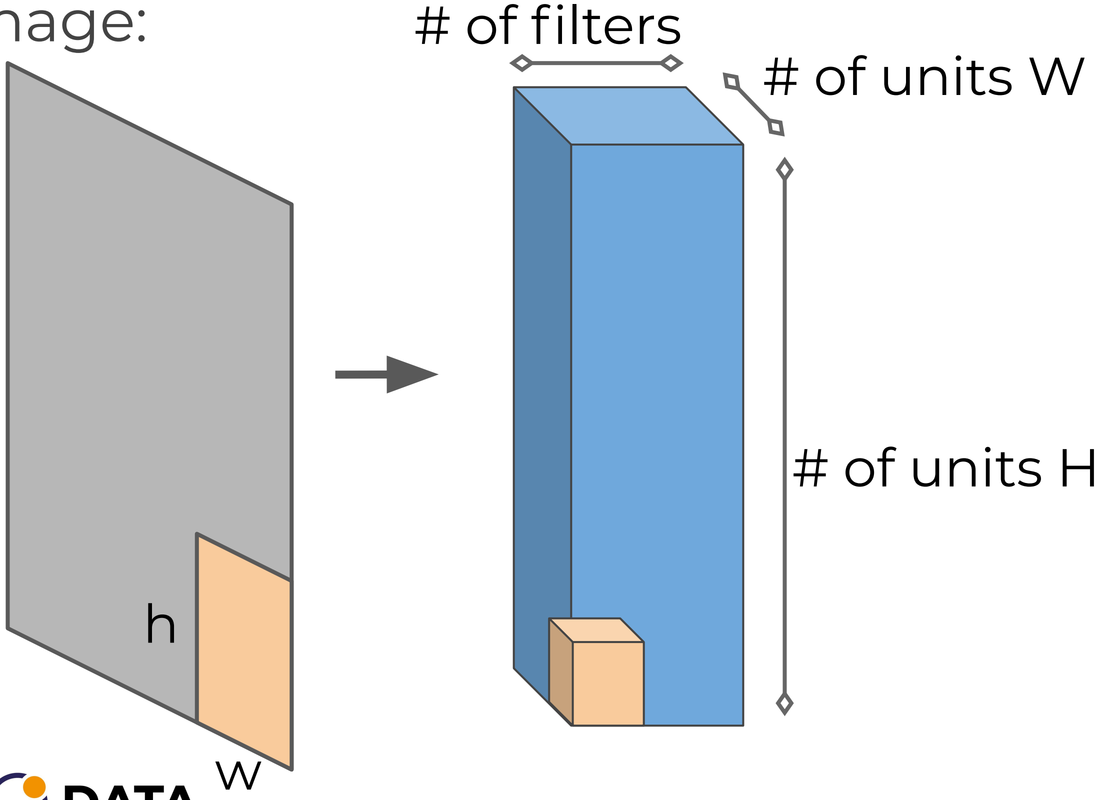
- Let's now expand these concepts to 2-D Convolution, since we'll mainly be dealing with images.

Deep Learning - 2D Convolution

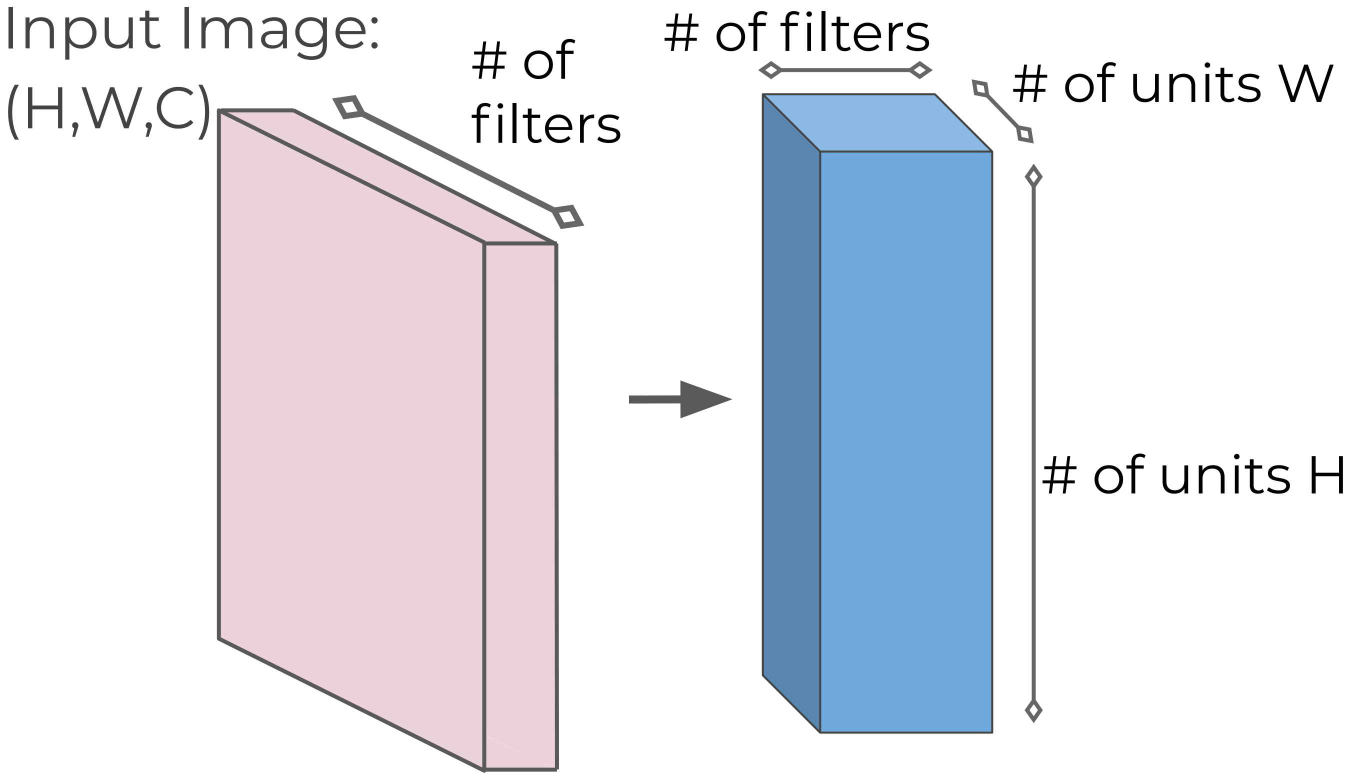


Deep Learning - 2D Convolution

Input Image:
(H,W)

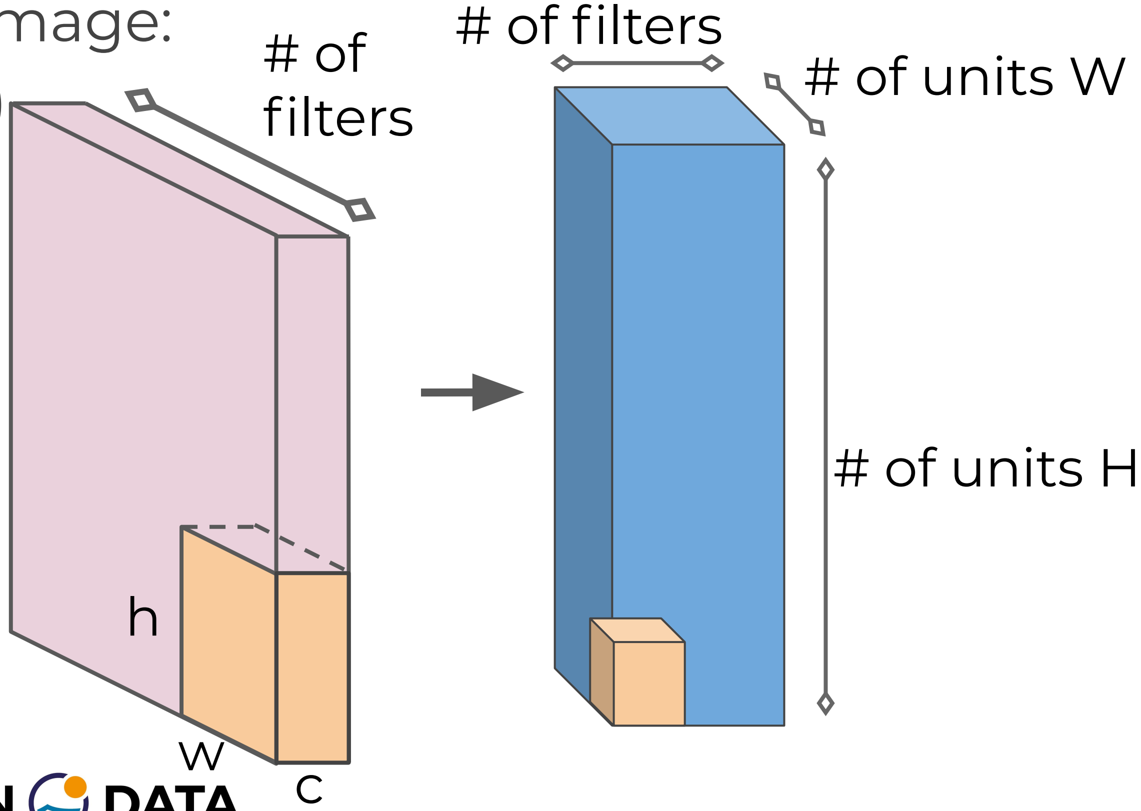


Deep Learning - 2D Color Images



Deep Learning- 2D Color Images

Input Image:
(H,W,C)



PIERIAN DATA

Deep Learning

- Filters are commonly visualized with grids

0	0	0	0	0	0
0	1	1	1	1	0
0	1	-1	-1	1	0
0	1	-1	-1	1	0
0	1	1	1	1	0
0	0	0	0	0	0

Deep Learning

- Filters are commonly visualized with grids

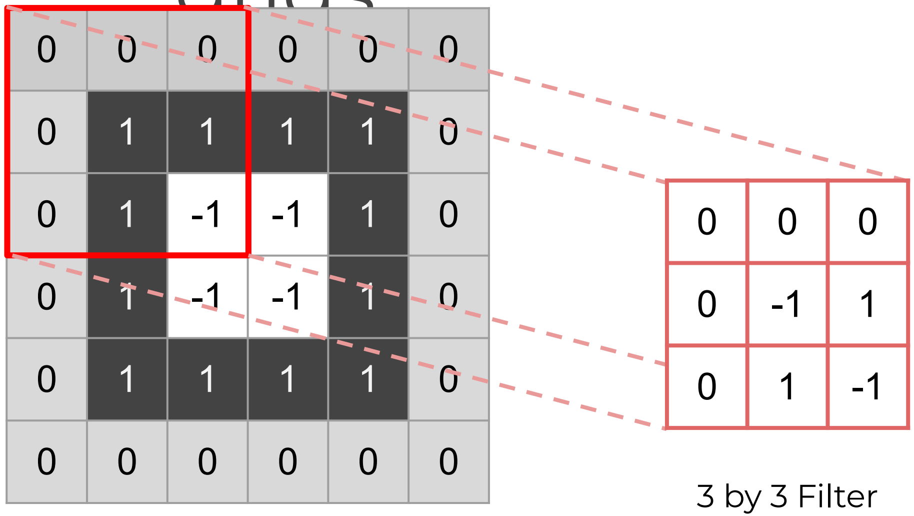
0	0	0	0	0	0
0	1	1	1	1	0
0	1	-1	-1	1	0
0	1	-1	-1	1	0
0	1	1	1	1	0
0	0	0	0	0	0

0	0	0
0	-1	1
0	1	-1

3 by 3 Filter

Deep Learning

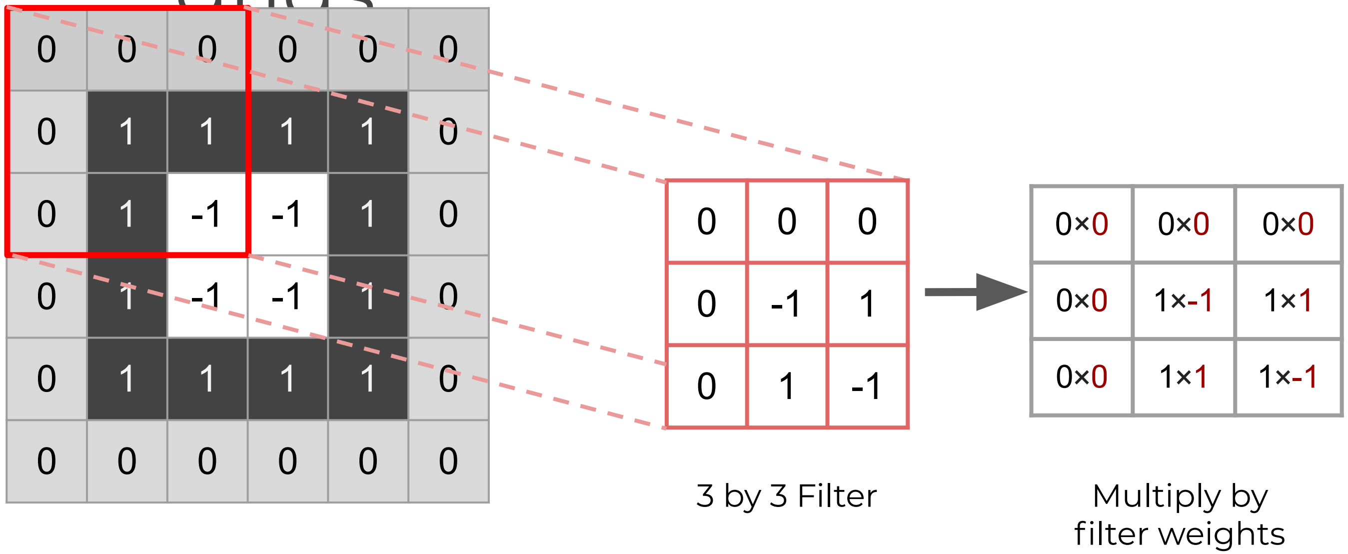
- Filters are commonly visualized with grids



3 by 3 Filter

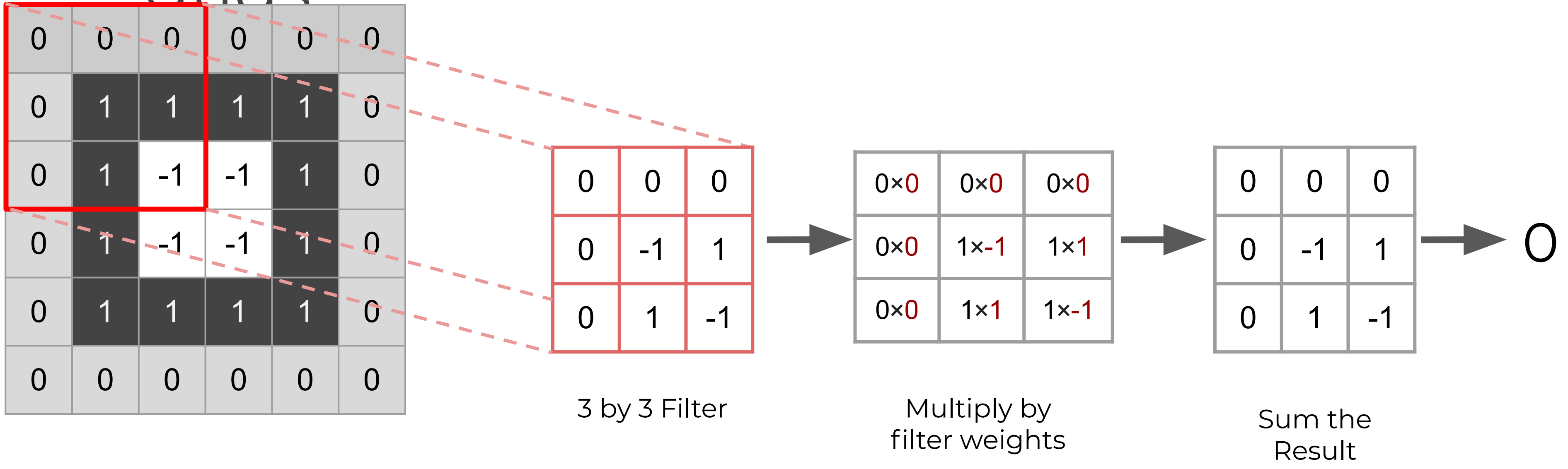
Deep Learning

- Filters are commonly visualized with grids



Deep Learning

- Filters are commonly visualized with grids



Deep Learning

- Stride Distance of 1 Example

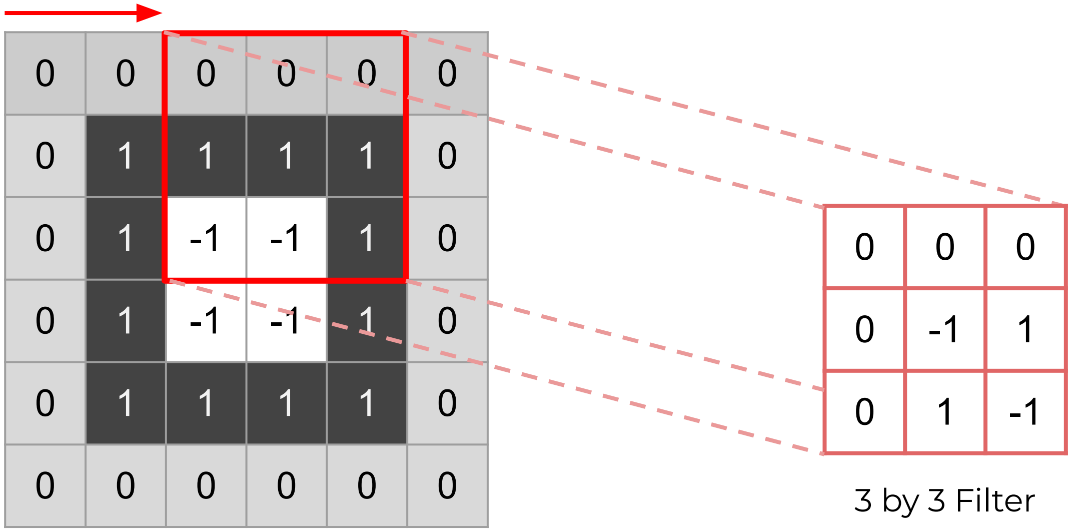
0	0	0	0	0	0
0	1	1	1	1	0
0	1	-1	-1	1	0
0	1	-1	-1	1	0
0	1	1	1	1	0
0	0	0	0	0	0

0	0	0
0	-1	1
0	1	-1

3 by 3 Filter

Deep Learning

- Stride Distance of 2 Example

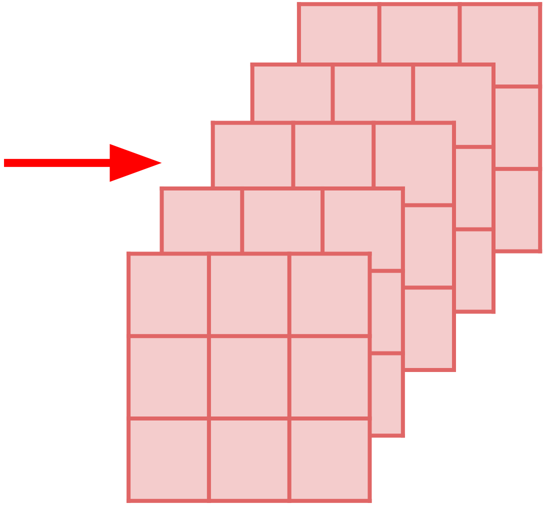


3 by 3 Filter

Deep Learning

- Representation of Multiple Filters

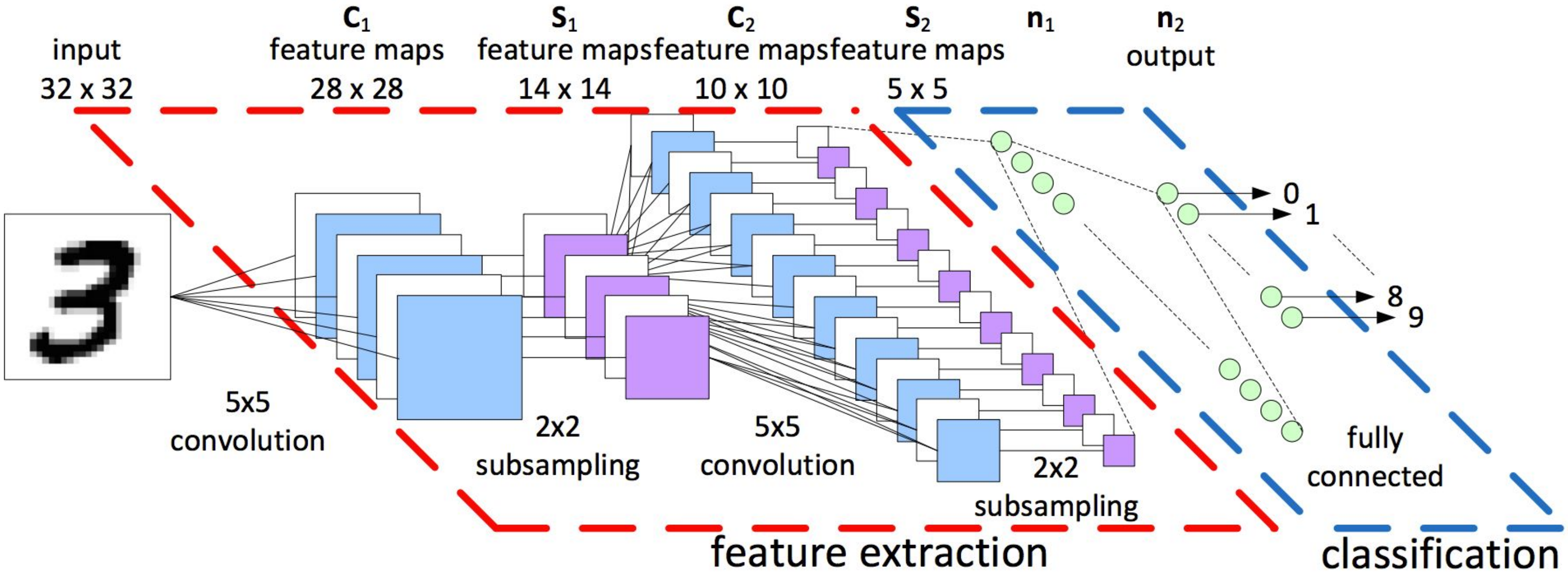
0	0	0	0	0	0
0	1	1	1	1	0
0	1	-1	-1	1	0
0	1	-1	-1	1	0
0	1	1	1	1	0
0	0	0	0	0	0



3 by 3
Convolution

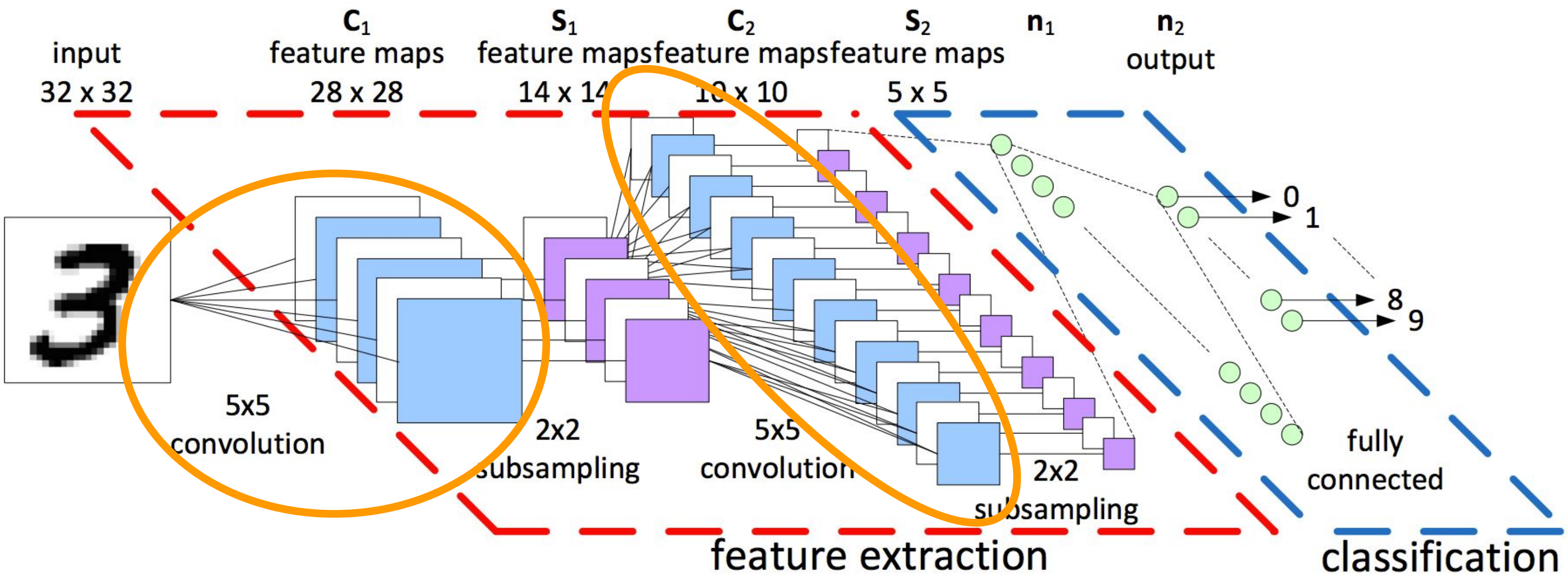
Deep Learning

- At the original CNN diagram



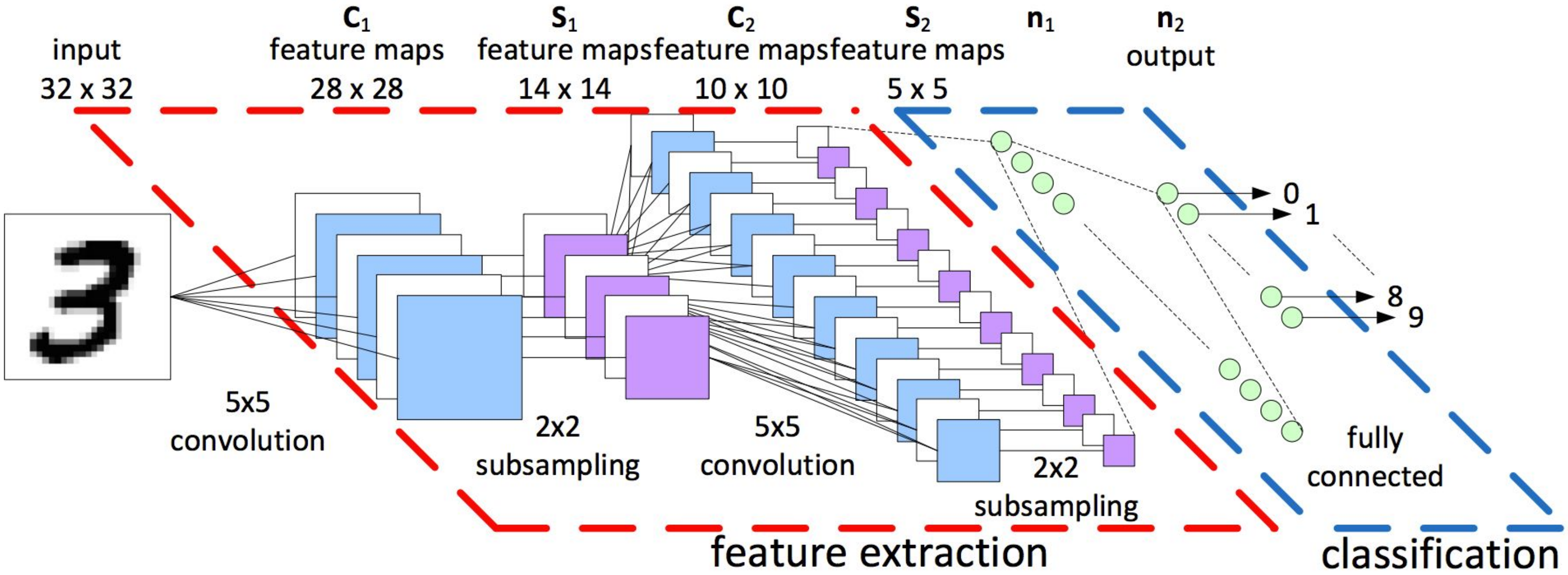
Deep Learning

- Exactly what we saw here:



Deep Learning

- Now it is time to discuss subsampling (pooling)

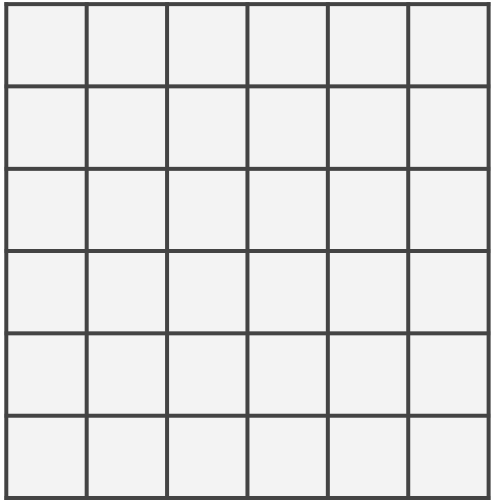


Convolutional Neural Networks Part Two

- Now that we understand convolutional layers, let's discuss **pooling layers.**
- Pooling layers will subsample the input image, which reduces the memory use and computer load as well as reducing the number of parameters.

Deep Learning

- Let’s imagine a layer of pixels in our input image:



Deep Learning

- For our MNIST digits set, each pixel had a value representing “darkness”

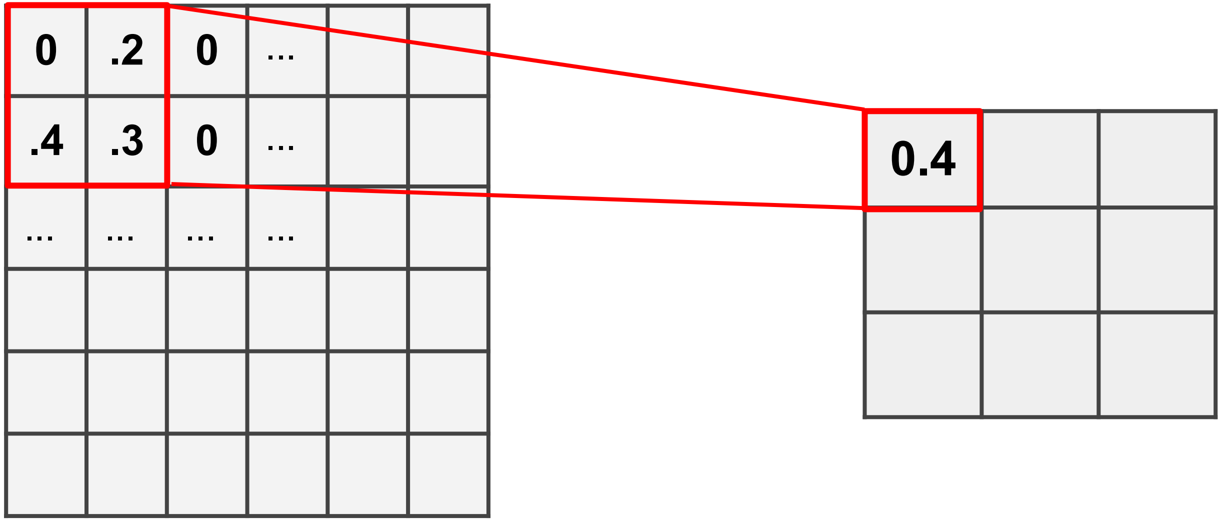
0	.2	0	...		
.4	.3	0	...		
...	...	...	...		

- We create a 2 by 2 pool of pixels and evaluate the maximum value.

0	.2	0	...		
.4	.3	0	...		
...	...	...	...		

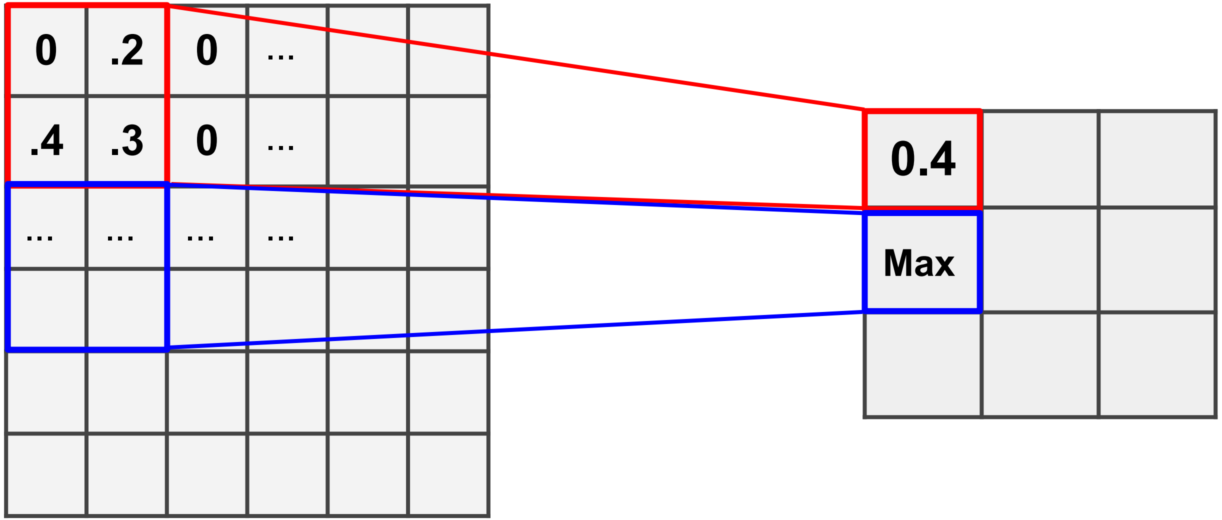
Deep Learning

- Only the max value makes it to the next layer.



Deep Learning

- We then move over by a "stride", in this case, our stride is two.



- This pooling layer will end up removing a lot of information, even a small pooling "kernel" of 2 by 2 with a stride of 2 will remove 75% of the input data.

* Another common technique deployed with CNN is called “Dropout”
* Dropout can be thought of as a form of regularization to help prevent overfitting.
* During training, units are randomly dropped, along with their connections.

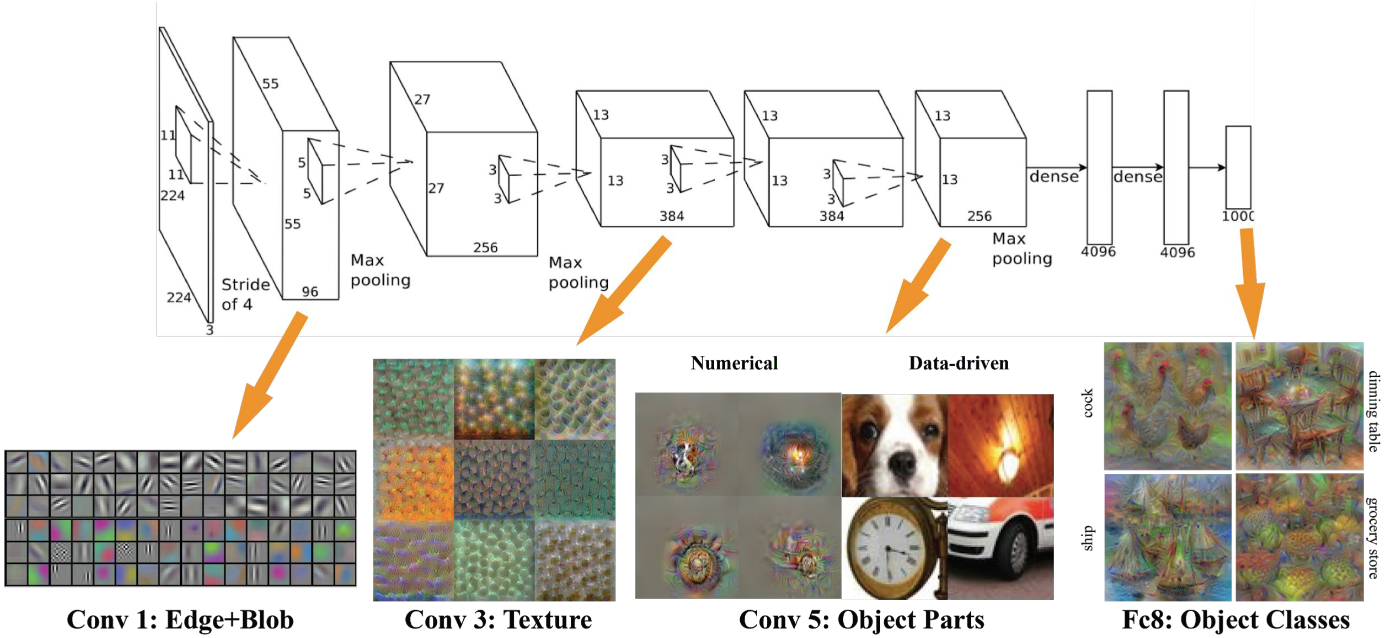
- This helps prevent units from “co-adapting” too much.
- Let’s also quickly point out some famous CNN architectures!

Deep Learning

- LeNet-5 by Yann LeCun
- AlexNet by Alex Krizhevsky et al.
- GoogLeNet by Szegedy at Google Research
- ResNet by Kaiming He et al.
- Check out the resource links to the papers discussing these architectures!

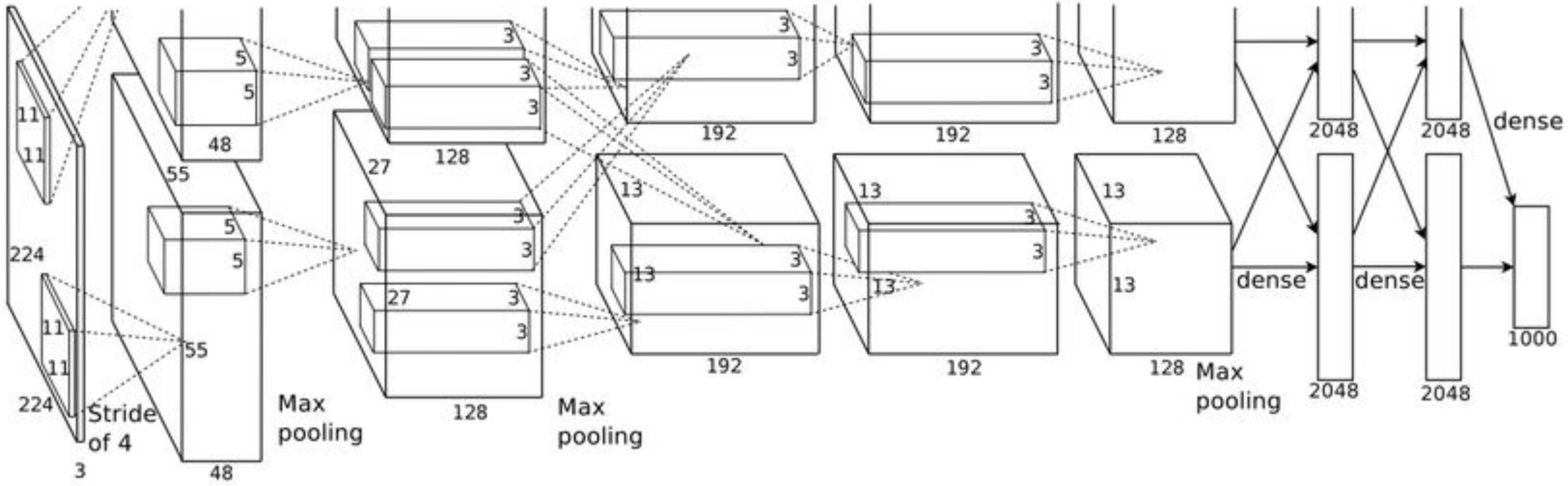
Deep Learning

- AlexNet



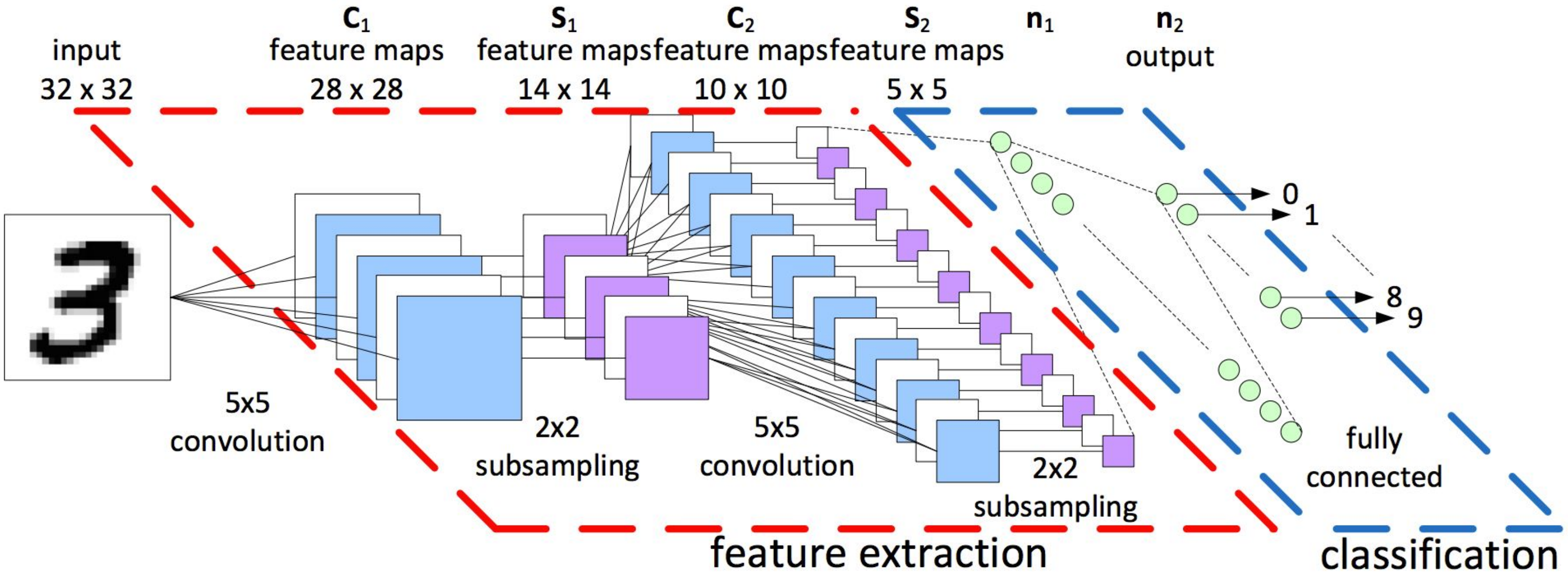
Deep Learning

- AlexNet



Deep Learning

- Convolutional Neural Network



Deep Learning

- Check out the various supplementary resources!
- Now that we’ve covered the basics of CNN, let’s explore how to implement one in TensorFlow!

MNIST Data
CNN Code Along

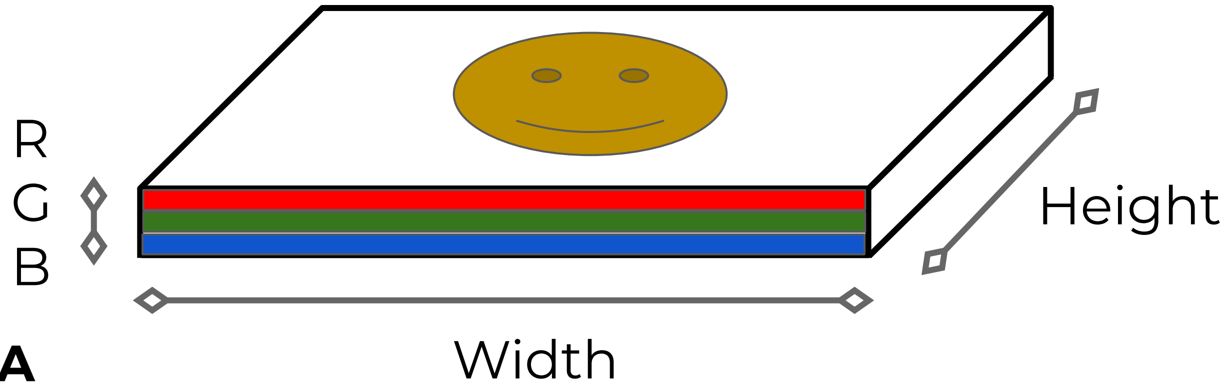
CNN CIFAR-10 Exercise

CNN CIFAR-10 Exercise Solutions Part Two

* Optional CNN Exercise with the CIFAR-10 data set.
* Main challenge is dealing with data and creating Tensor batches and sizing.
* Let’s go through the exercise notebook!

Deep Learning

- We have our color image represented as a 3-D prism. Width, Height, and Depth (3 values R, G, B)



Deep Learning

- Run a neural network on a portion