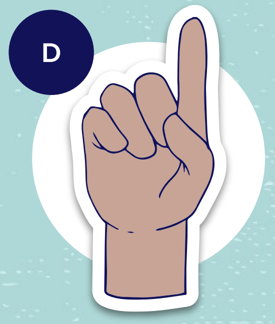
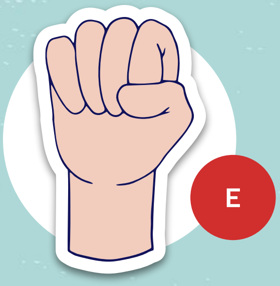


WEEK 2:
Sign Language Translator



Icebreaker!

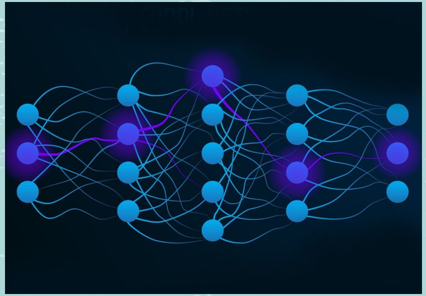
- Find people near you and form small groups of 2-3 people
- Introduce yourself if you don’t know them already!
  - Name, major, grade
  - One place you’ve always wanted to visit!

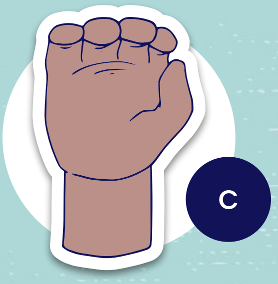
Last Week…

- Talked about why ASL translation is important
- Talked about MediaPipe and OpenCV
- Saw how MediaPipe creates landmarks
- Hardcoded detection functions (is_peace_sign())

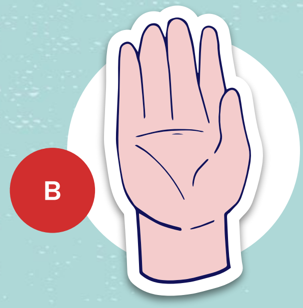
This week...

- Introduction to Neural Networks
- Notebook Activity
- Introduction to CNNs
- Environment Setup



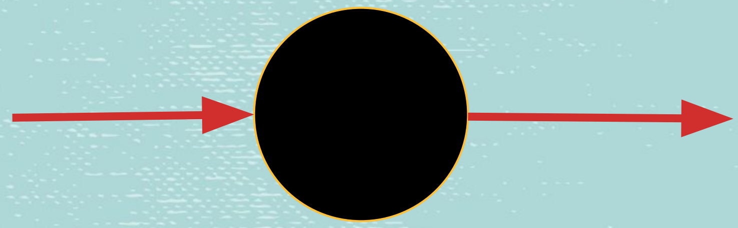


Neural Networks



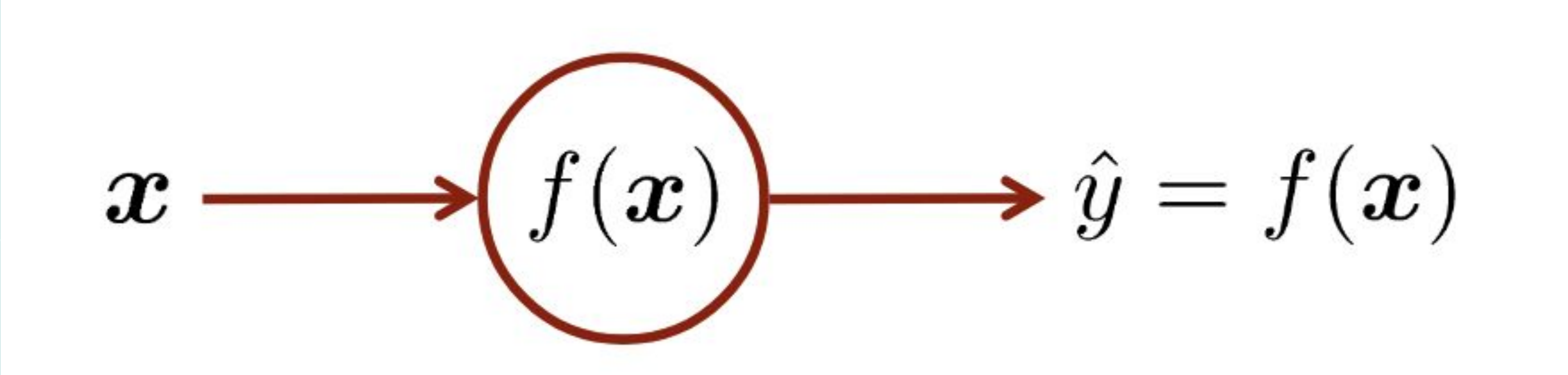
What is a Neuron?

- **In biology,** neurons are fundamental units of the brain
  - Receives input from the external world (through your senses)
  - Transforms and relays signals
  - Sends signals to other neurons and commands to the muscles
- **Artificial Neurons**
  - Receive Inputs
  - Transforms information
  - Creates an output



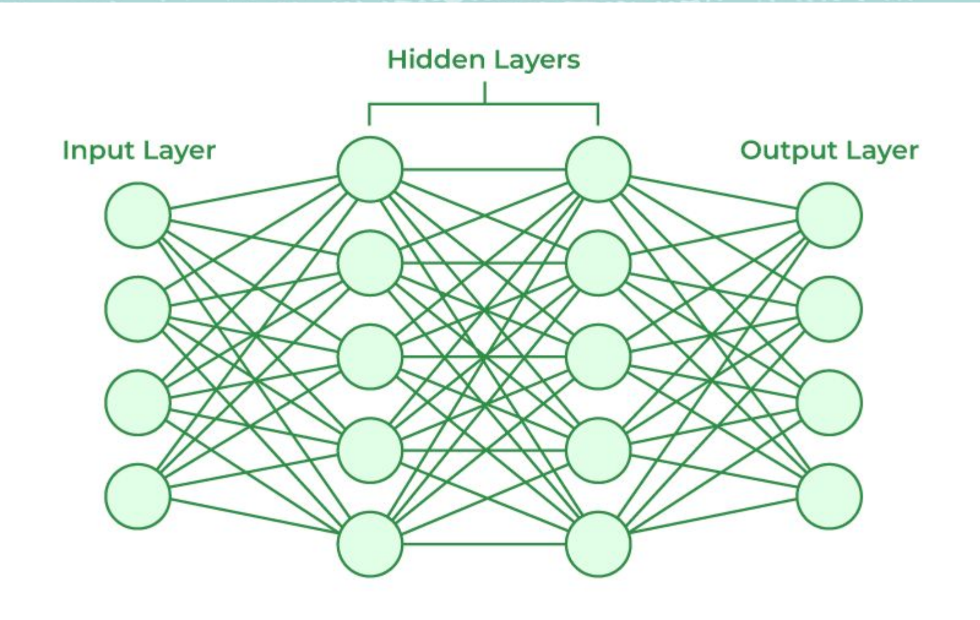
Neurons

- Receive **inputs/activations** from sensors/other neurons
- **Combines/transforms** information
- Creates an **output/activation**



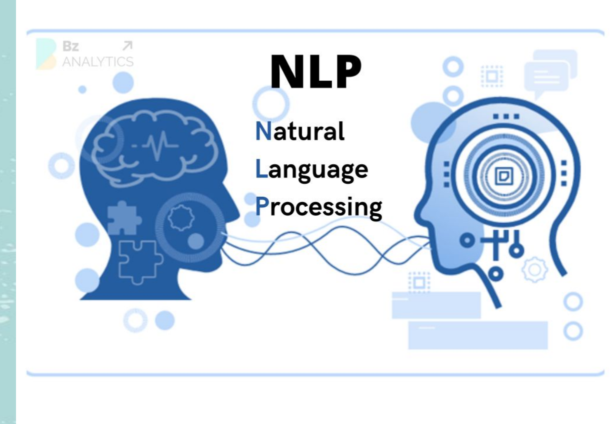
Neural Network

- Each node is a neuron
- Edges are input-output connections

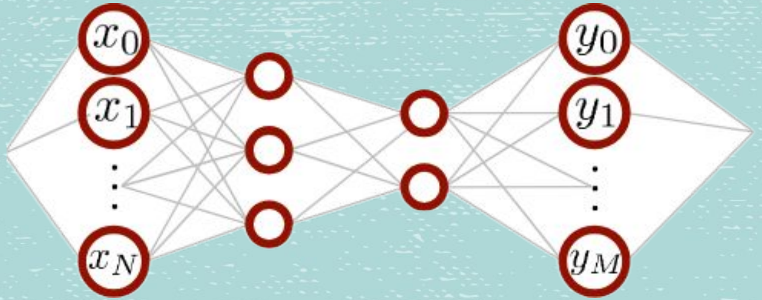


Use Cases

- Image classification
- Natural Language Processing (chatbots)
- Predictive Analytics



Let's see it in Action



"Donkey"

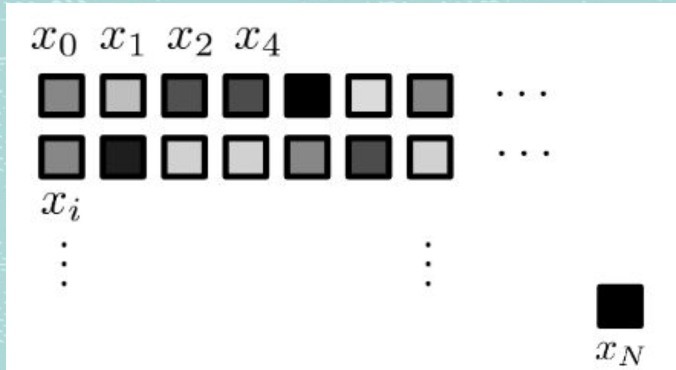
Input
Image

Function
maps
images to
labels

Label

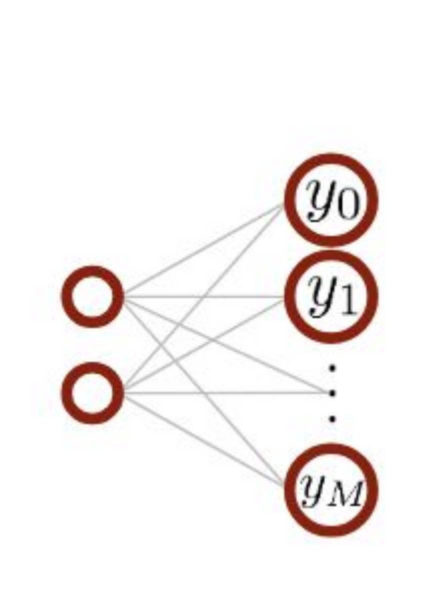
What is our Input Exactly?

- Computers can't really **see** images like we do
- An image consists of many individual pixels
- Each pixel has an intensity value



$$\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix}$$

What is our Output?



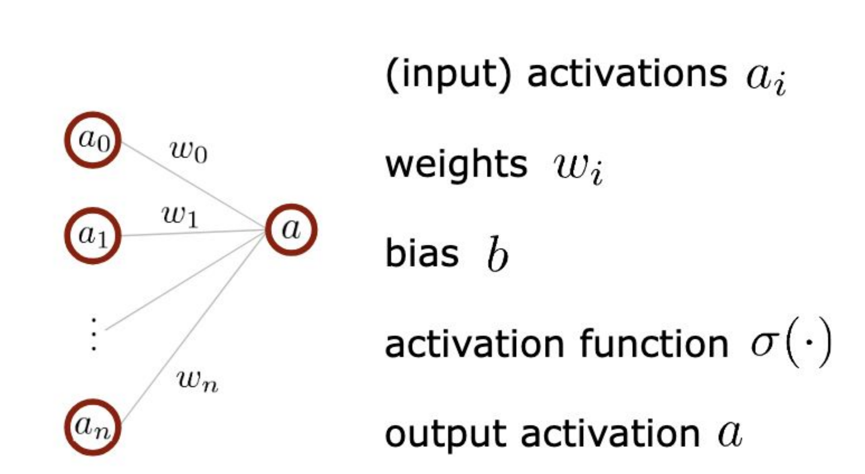
Is it a donkey?
Or a dog? Or a
lizard? Or....

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

Indicates activation/
likelihood for each
variable

What's Actually Going On

- Let's look at a **perceptron**:



$$a = \sigma(w_0 a_0 + w_1 a_1 + \ldots + w_n a_n + b)$$

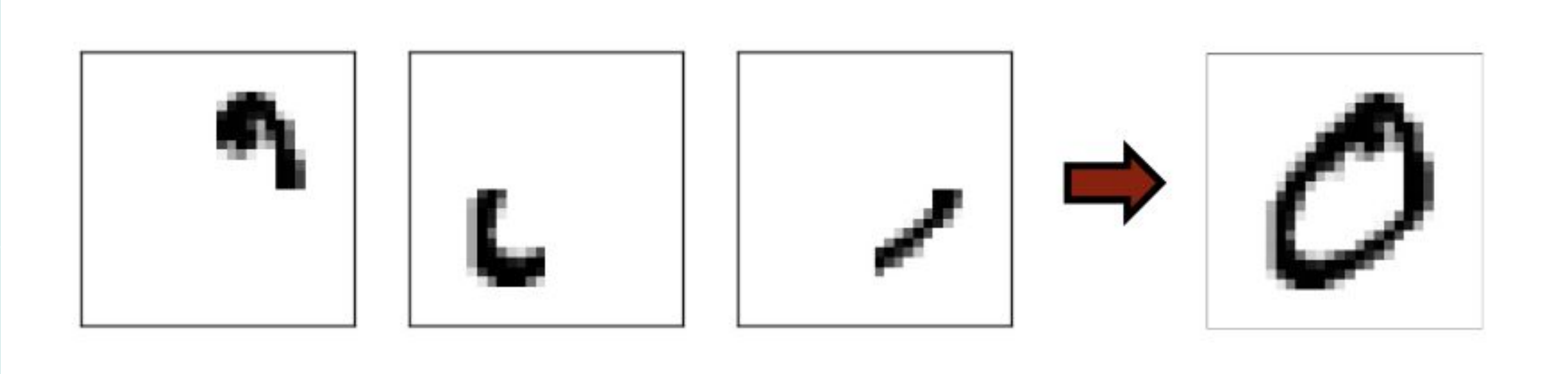
What Do the Parts Mean?

- **Weights** define the patterns to look for in the image
- **Bias** tells us how well the image must match the pattern
- **Activation functions** introduces *non-linearity* to decision boundaries to allow for the learning of complex patterns

$$a = \sigma(w_0 a_0 + w_1 a_1 + \ldots + w_n a_n + b)$$

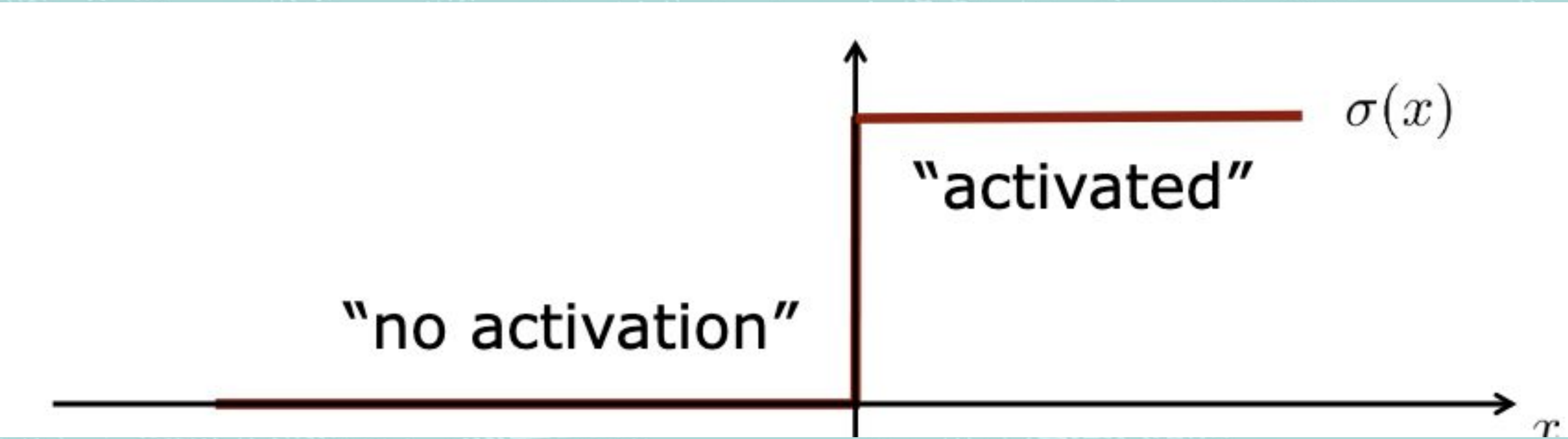
Weights

- The weights in layers 2 onward tell us which previous layer patterns should be combined
- Early layers detect simple patterns; later layers combine them into more complex ones.



Activation Functions

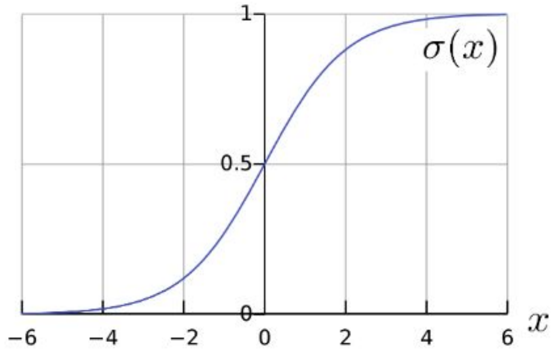
- In biology, neurons are either **active** or **not active**
- This is shown as a step function
- Bias is what tells us **where** the activation happens



Common Activation Functions: Sigmoid

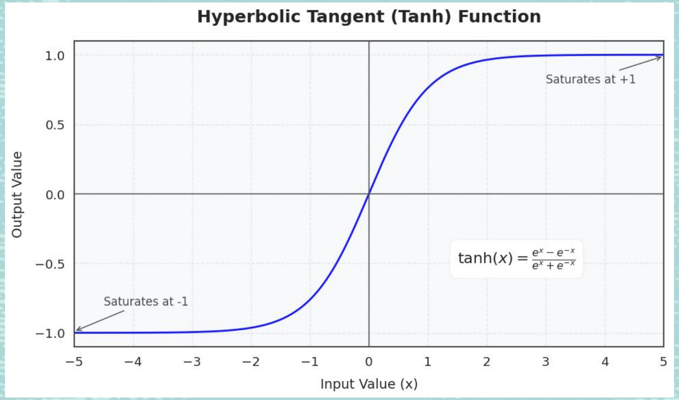
- AKA logistic function
- Squeezes values to [0,1]
- motivation: can interpret as a probability

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$



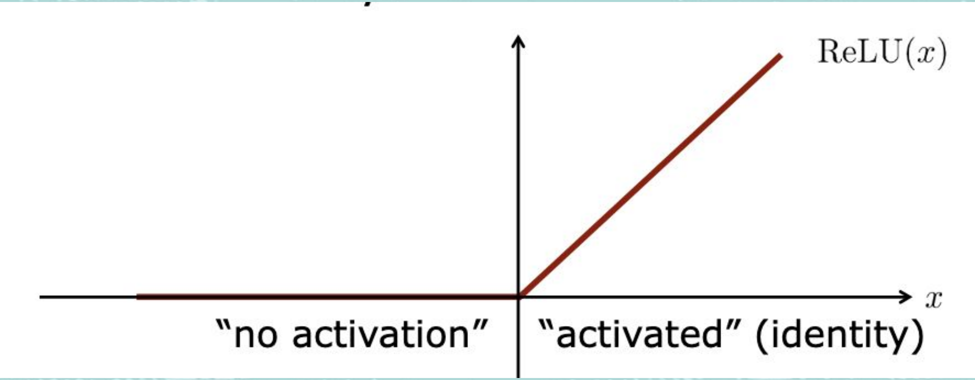
Common Activation Functions: Tanh

- AKA hyperbolic tangent
- Squeezes values to [-1, 1]
- motivation: outputs have mean=0, steeper gradient at x=0 which allows for larger updates



ReLU

- Most commonly used – Rectified Linear Unit
- Neuron only activated if x > 0
- motivation: avoids "saturation" seen in other activation functions



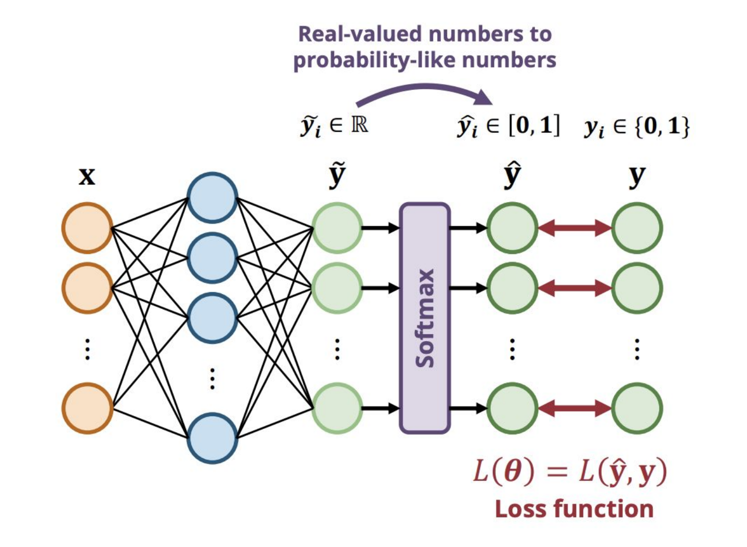
Which Activation Function to Use?

- Intuition of ReLU: Neurons only get activated if the **weighted sum** of input activates is larger than the negative bias
- This helps mitigate a common issue with model training:

Vanishing Gradient Problem: as updates propagate backwards in a deep neural network, they become small due to repeated multiplication of small derivatives. This results in the model not being able to effectively learn parameters that exist in earlier layers.

Loss Function

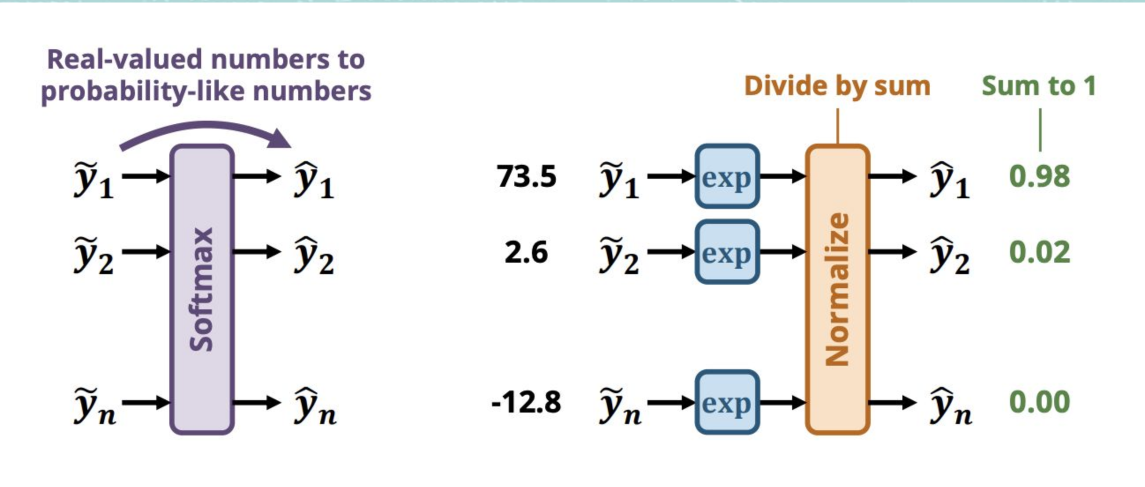
The loss function defines what we want to optimize for. In our case, we want to make **multiclass classifications**, so we will use cross-entropy.



Softmax

Intuition: Map several numbers to [0,1] while **keeping their relative magnitude**

- Softmax is like the multivariate version of sigmoid



Cross Entropy

Lower values correspond to low “surprise” or error

Binary Cross Entropy

Only one of them will be one!

$$L(\hat{y}, y) = -\boxed{y}\log\hat{y} - \boxed{(1-y)}\log(1-\hat{y})$$

Cross Entropy

Only one of them will be one!

$$L(\hat{\mathbf{y}}, \mathbf{y}) = -\boxed{y_1}\log\hat{y}_1 - \boxed{y_2}\log\hat{y}_2 - \cdots - \boxed{y_i}\log\hat{y}_n$$

$$= -\boxed{\sum_{i}^{n} y_i \log\hat{y}_i}$$

Log likelihood

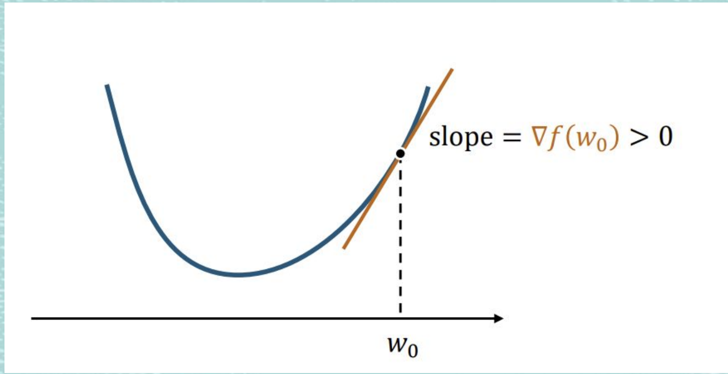
How Does it Learn?

- **Training:**
  - Show the model an input
  - It makes a prediction
  - Compare the prediction to the true label
  - Adjust weights
  - Repeat

Gradient Descent

Intuition: Gradient can suggest a good direction to tune the parameters

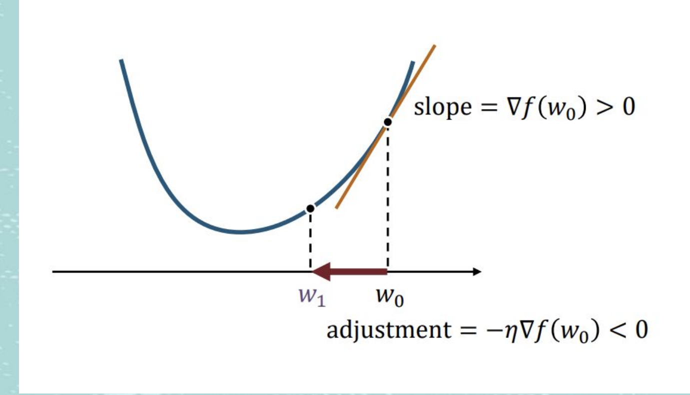
Gradient: Derivative for a vector, matrix or tensor

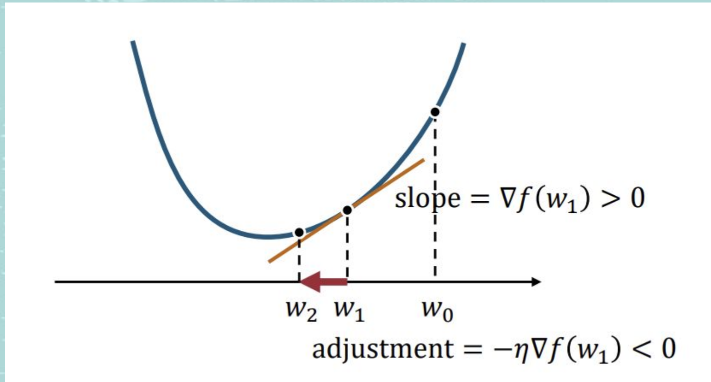


Gradient Descent

- Pick an initial **weight vector** w0 and **learning rate** η
- Repeat until convergence:

$$w_{t+1} = w_t - \eta \boxed{\nabla f(w_t)} \longrightarrow \text{Gradient of function } f \text{ with respect to weight } w$$





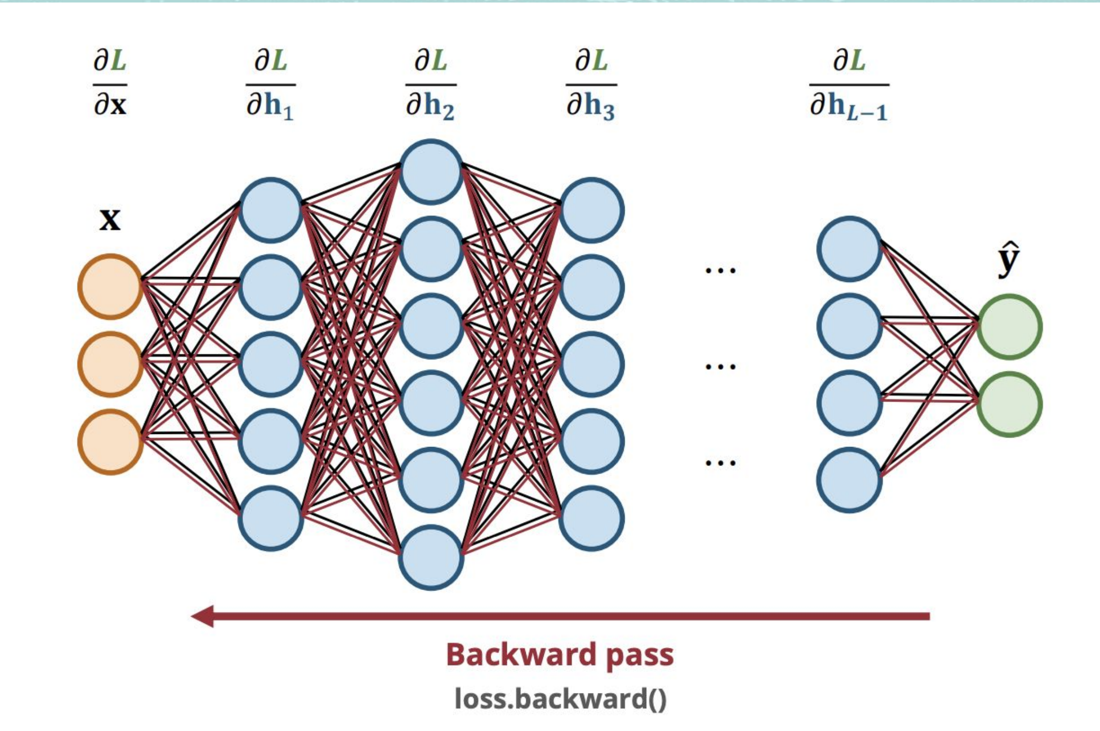
Gradient Descent

Batch Gradient Descent: Averages updates over multiple examples (batches)

Stochastic Gradient Descent: Iterates over training examples in a random order for each epoch (pass over entire dataset)

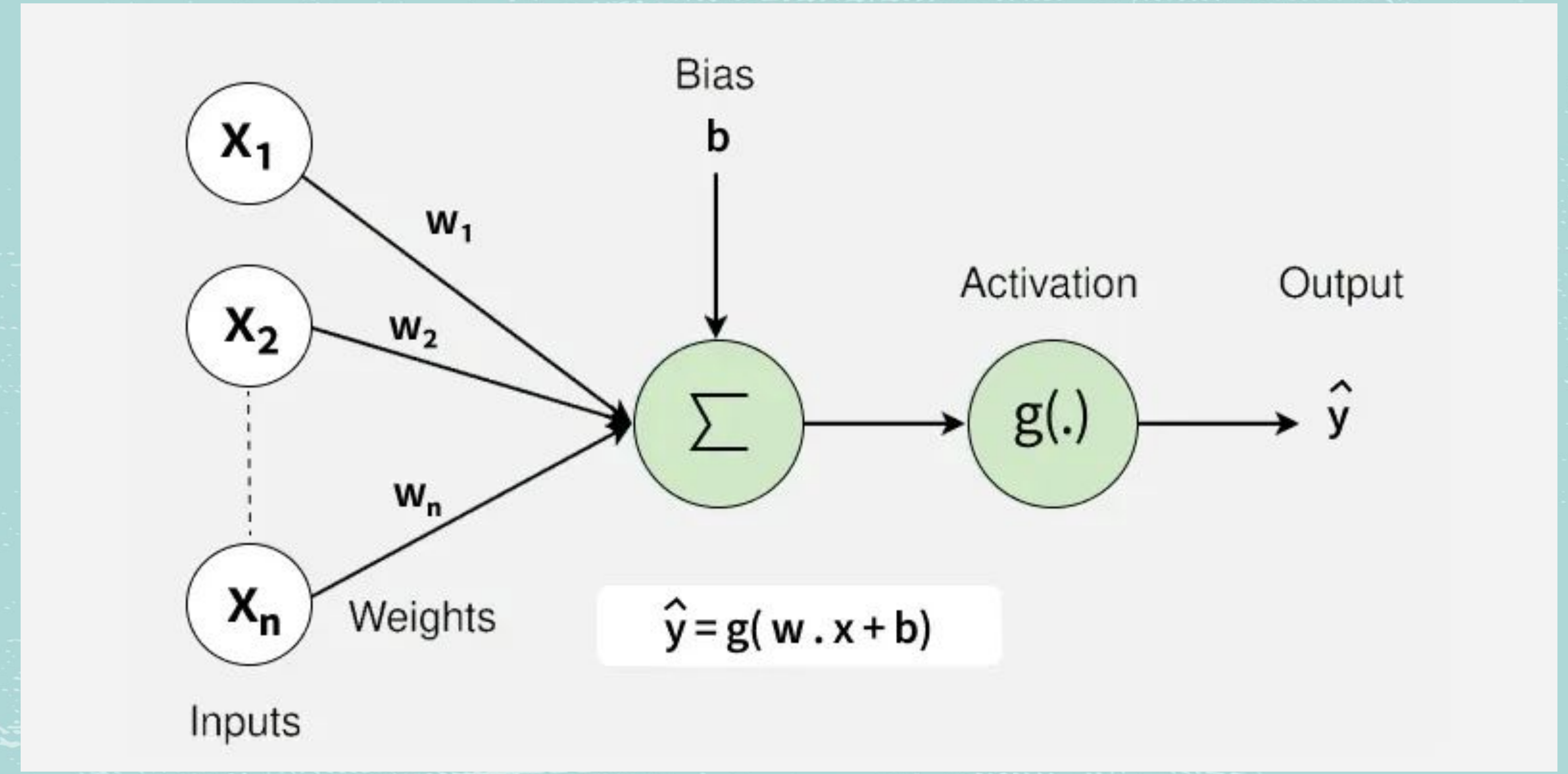
Mini-batch SGD: Combines both ideas by using small, random samples for each update (what we'll use)

Backpropagation



Testing

- **Testing/Validation:**
  - Show data that the model has never seen
  - Don’t update the weights
  - Measure performance
  - Ensures model isn’t just memorizing training data



Bias
b
X1
X2
Xn
w1
w2
wn
Weights
Inputs
Activation
Output
g(.)
ŷ
ŷ = g(w . x + b)

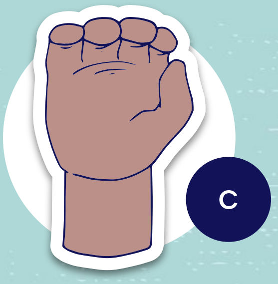
Environment Setup

- Clone the repository to your local machine.
- Run the following commands in your terminal:

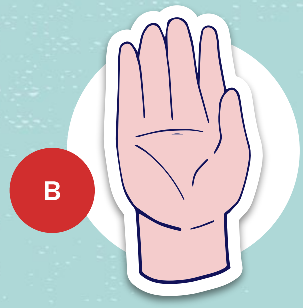
```
cd W26-Sign-Language-Translator/Week\ 2
python3 -m venv .venv
source ./.venv/bin/activate
pip install -r requirements.txt
# <see speaker notes for hand_landmarker.task file>
```

Try It Yourself!

- Jupyter Notebooks exercise…
- Guided activity with PyTorch + Neural Networks



CNNs

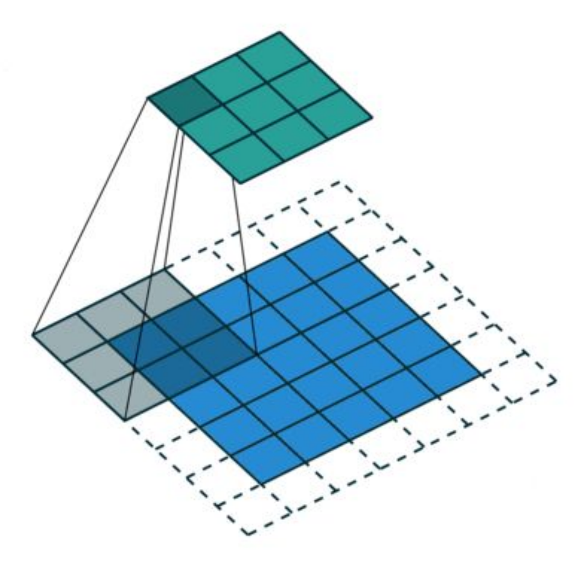


What’s the Issue With What We Had?

- The problem is that we don’t retain any spatial information!
- The location of the pixels in the image are ignored
- CNNs maintain the 2D image structure
- Network layers can learn features that ALSO encode spatial info

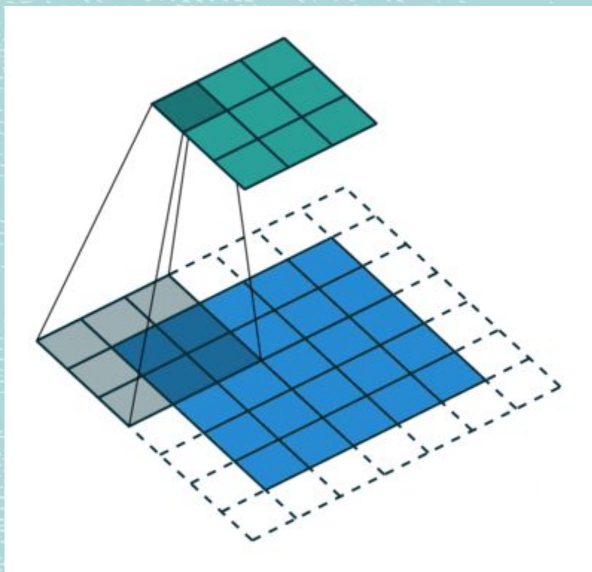
What is Convolution?

- Slide a **filter** over an image
- We take a small filter
- Slide it across the image
- At each position, we multiply and sum
- Result: a feature map that shows where the pattern appears



CNNs

- The **blue square** is the **input**
- The **gray square** is a **filter**
  - the pattern that we want to find in the input
- The **green square** is the **output**
  - tracks where the pattern was found in the input

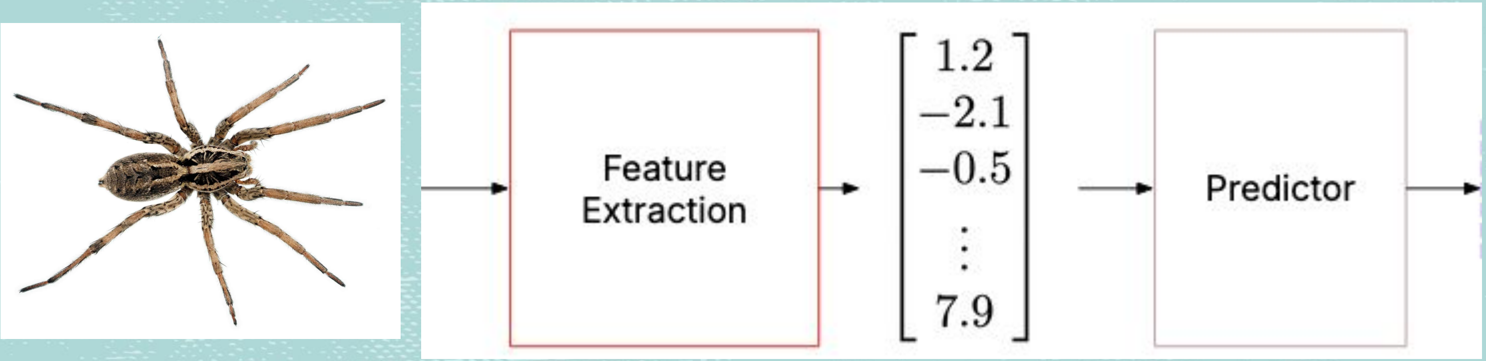


Convolution Hyperparameters:

- **Filter size** - pattern resolution
- **Stride** - how far we jump between windows
- **Padding** - extra space around edges (keeps output size same as input)

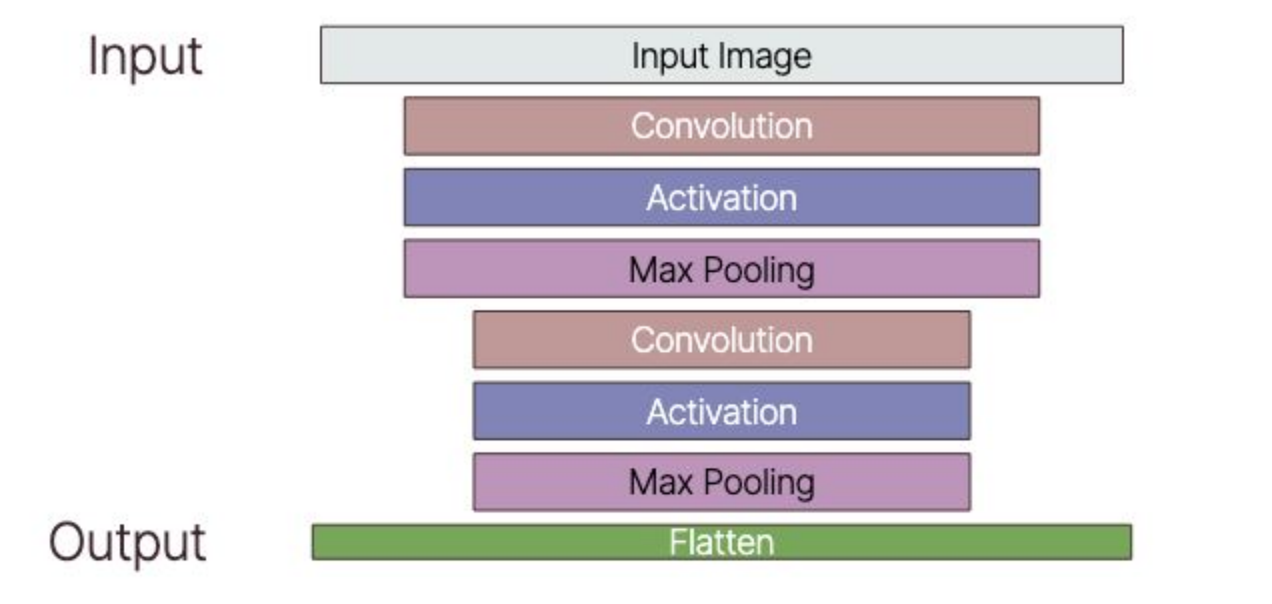
CNNs

- Feature Extraction: converting image to a vector
- Predictor - convert a vector to a predicted class



SPIDER

Feature Extraction



Convolutional Layers

- **Input**: Tensor (3D array) from previous layer (D x H x W). Image or a feature map.
- **Process**: Convolutional layers include n number of filters, which slide over the image, looking for a specific pattern
- **Output**: Each filter creates a feature map that shows where the pattern was. We stack them all together to get an output tensor (n x new Height x new Width)
    - Output size shrinks depending on size of the filter and the stride

How Does It Recognize Images?

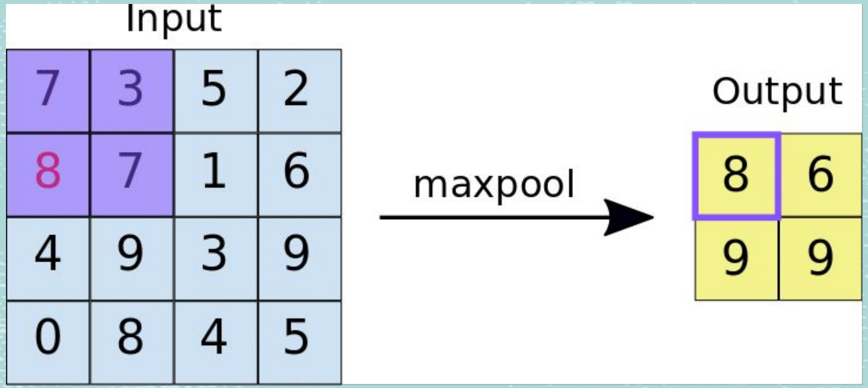
- Layers upon layers of convolutions!
- Layer 1: Edges, colors (Horizontal line, red blob)
- Layer 2: Textures, simple shapes (Checkerboard, circles)
- Layer 3: Object parts (Fingers, palm, knuckles)
- Layer 4: Full objects (Hand making "peace" sign)

Activation Layers

- We’ve seen this before!
- The same process:
    - **Input:** Tensor from previous layer ***(D x H x W)***
    - **Process**: Apply a function to each number in tensor
    - **Output:** Output tensor after activation ***(D x H x W)***

Pooling Layers

- Shrinks the image down!
- This reduces information, but keeps what's important
- For example, keeping the biggest number



Flatten Layers

- Finally, we take the tensor and flatten down into a single dimension vector
- This vector can now be used to predict!

Predictor

- Dense layers:
  - Transform a vector from one size to another
  - Our input vector has all the features flattened
  - Output vector should be one number per class (like 26 letters)
- Activation layers:
  - **Input:** Vector from previous layer
  - **Output:** Activated output of same size

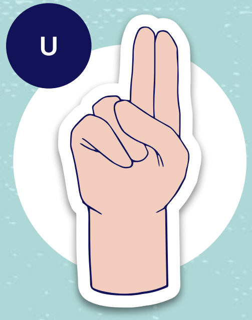
Why This Matters For Us

- Makes detecting patterns more accurate
- Can spot edges, curves, and finger positions
- Preserves spatial relationships
- We use hand landmarks, which makes the process easier! For us, the positions matter a lot

So Are We Done?

- Nope!
- ASL isn't just static images, it's moving gestures
- We need to be able to **remember** past frames to detect relationships between keypoints
- CNNs are a good starting point
- Later, we'll learn about RNNs + LSTMs

Next Week: Feature Representation and the WLASL Dataset



Further Reading...

- Introduction to Convolution Neural Network – GeeksforGeeks
- Introduction to Neural Networks – University of Toronto
- Lecture 7: Convolutional Neural Networks – Stanford
- Image Classification using CNN – GeeksforGeeks
- Sign Language Recognition with Convolutional Neural Networks | CS231n