



Autumn 2023 Week 4

go.osu.edu/aiclubsignup



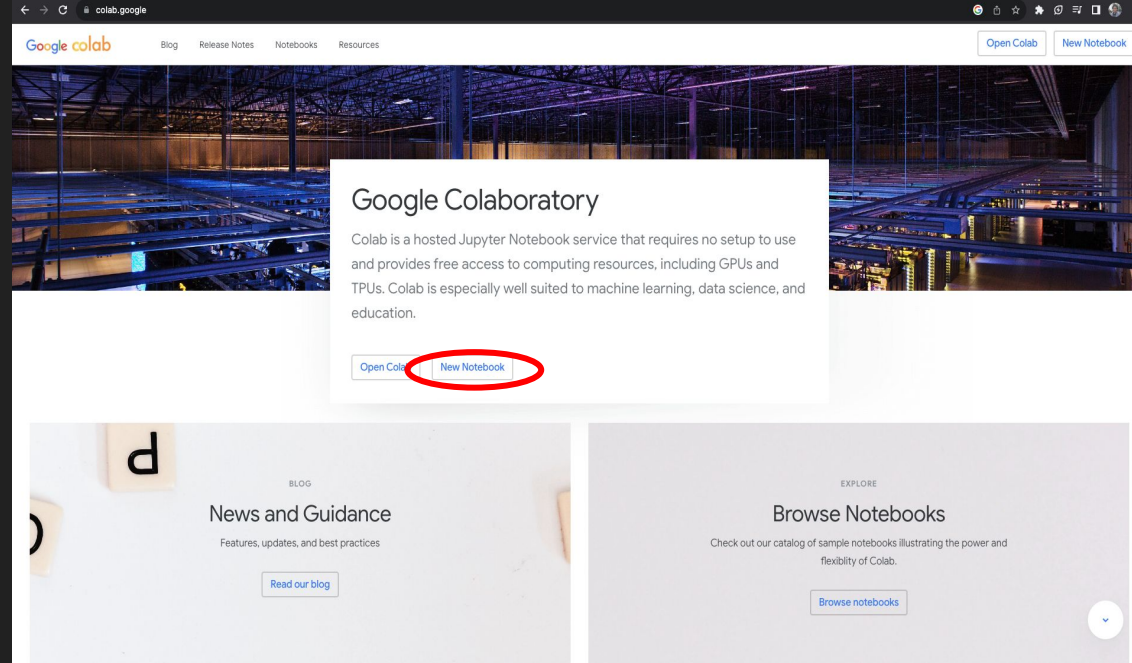
Plan for today

- What is a model?
- What makes up a model?
- How do you make your own model?



Before we start...

- Open up Google Colab
- Select “New Notebook”
- Name the notebook whatever you want
- <https://colab.research.google.com/>



Enter this into your import code block

```
from torch import nn, save, load
from torch.optim import Adam
```



ProjectSeriesLiveLessonipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

+ Code + Text

Getting the dataset into your google colab

```
[ ] !unzip /content/challenges-in-representation-learning-facial-expression-recognition-challenge.zip
```

Importing necessary libraries

```
[ ] import torch
import torchvision
from torchvision import transforms
```

Data Preprocessing & Loading


```
#this is for data preprocessing and loading with train data
def train_pl():
    #the transformation we will apply to the images from the FER2013 dataset
    transform = transforms.Compose([
        transforms.Grayscale(),
        transforms.ToTensor(), # Convert image to tensor
        transforms.Normalize(0.485, 0.229) # Normalize image
    ])

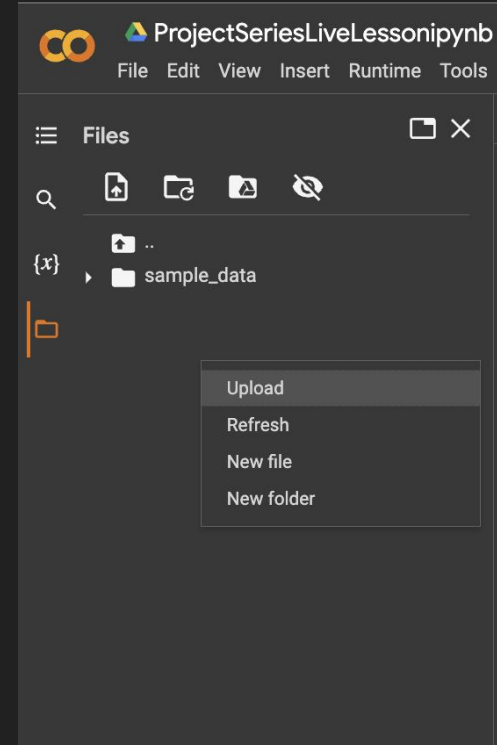
    # loading the data from the directory I have stored the downloaded FER2013 dataset
    train_data = torchvision.datasets.FER2013(root='/content', split = 'train', transform=transform)

    # create dataloaders so that the FER2013 data can be loaded into the model we will implement
    train_loader = torch.utils.data.DataLoader(train_data, batch_size=19, shuffle=True, num_workers=2)

    return train_loader
```

Downloading Dataset into Google Colab (renewed)

- Click on the  icon on the left side bar.
- Then right click in the file area and click “Upload” (as shown in the picture to the right)
- Then upload the zip file that we downloaded in the previous slide



Downloading Dataset into Google Colab (renewed)

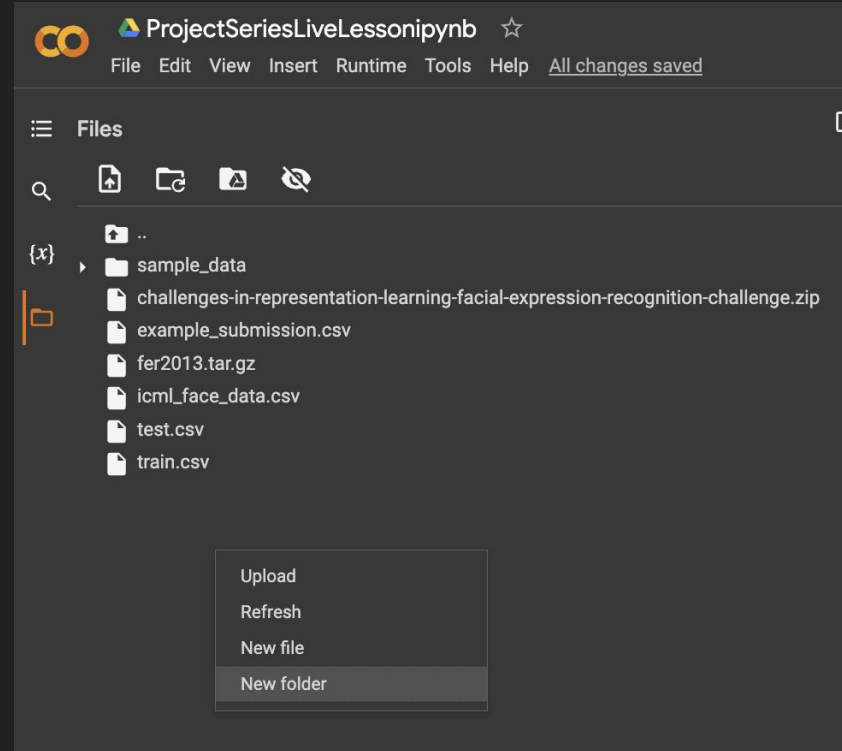
- Create a new code block with the button that says “ + Code ” at the top.
- Type in this line into that code block and click run (if your zip file is called something else, replace the “challenges-in-representation-learning-facial-expression-recognition-challenge” part with the name of your zip file
- Then wait a minute or two for the files to show up on your colab files.
- DELETE this code block from your notebook once you’ve done this!!!



```
!unzip /content/challenges-in-representation-learning-facial-expression-recognition-challenge.zip
```

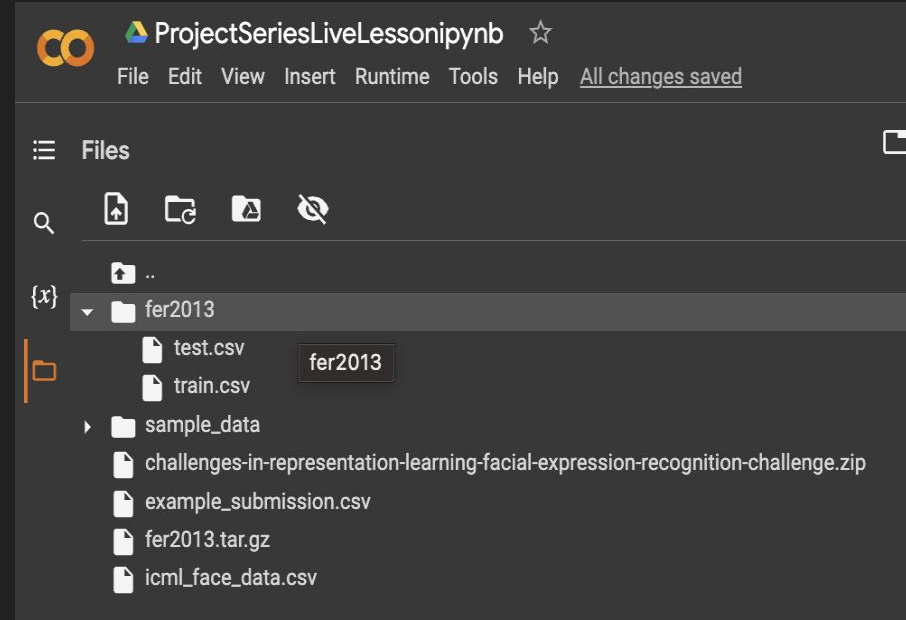
Downloading Dataset into Google Colab (renewed)

- Once all the files unzipped, right click in the file area and click “New folder” (as shown in the picture to the right)
- Title this folder “fer2013” (MAKE SURE TO COPY PASTE THIS EXACT)



Downloading Dataset into Google Colab (renewed)

- Move the “train.csv” and “test.csv” files into the “fer2013” folder you just made.
- You should have something that looks like the picture to the right.

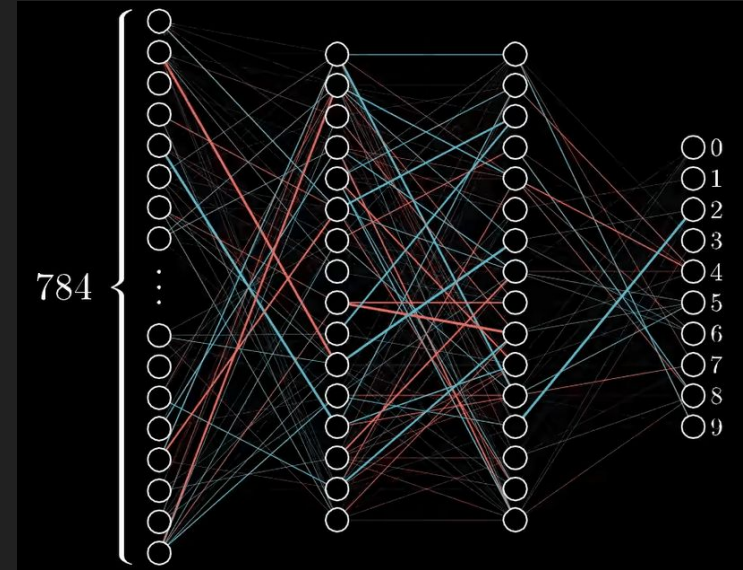


What is a Model?

Deep Learning Model: An algorithm/mathematical (often “black box”) model using deep learning that is capable of recognizing patterns in image, text, audio and other data to produce accurate insights.

- Often involves a lot of statistics, calculus, and linear algebra that we won't be going over in this course.
- Black Box: When you give a certain object something, you expect it to do something with it and give you an expected result without a care for how it does it. (ex: When you use `System.out.println()`, you know that if you give it a string, it will print it to the console. You don't care how it does it)

The algorithm is like the steps in the recipe, the model is the dish itself



[More info on what a neural network actually is](#)

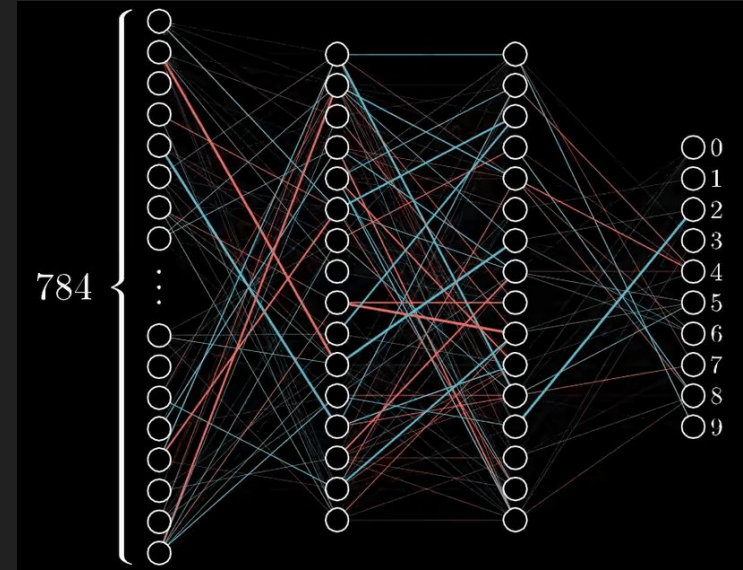
What is a Model?

Deep Learning Model: the resulting neural network of putting a deep learning algorithm through multiple rounds of training.

- A black box. You can give it an input and you can expect a certain range of outputs.
- Can use it with a variety of input like video, audio, and text

Deep Learning Algorithm: a series of steps that uses math to recognize patterns.

- Often involves a lot of statistics, calculus, and linear algebra that we won't be going over (that much) in this course.



[More info on what a neural network actually is](#)

Difference between Model and Algorithm

Model: Batman

Algorithm: The training regimen to become Batman

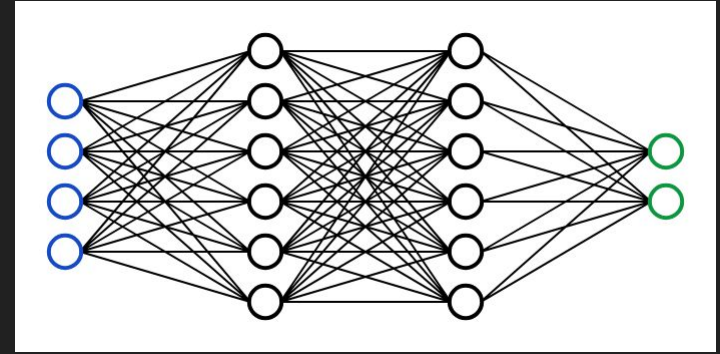


vs.

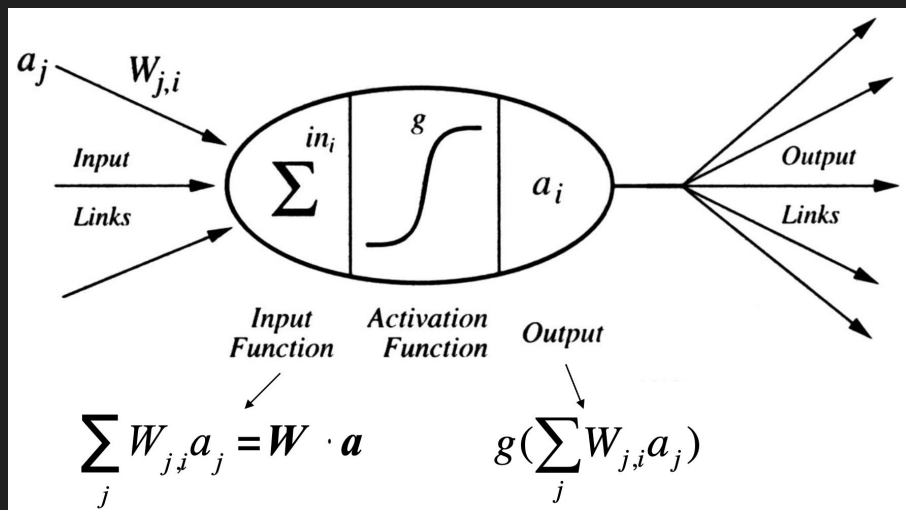


What makes up a model?

- **Neurons**: A node in the network that holds a number
- **Layers**: Applied transformations to the data to make it easier for the computer to detect patterns
- **Weighted Connections**: The neurons in each layer are connected with a weight
- **Activations**: Applied function after one or more transformations to determine findings from the data (AKA the significance of the results of transformations)
 - The more significant the finding is, the higher value the activation will return



Neurons

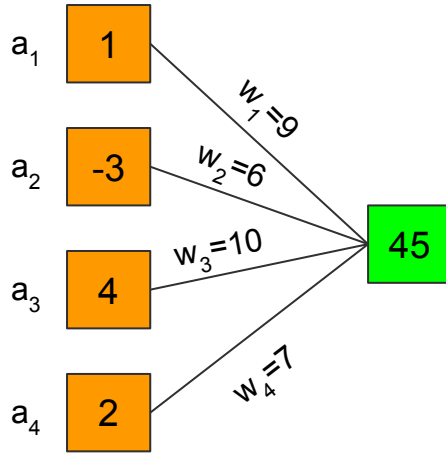


Anatomy of a Neuron (got this from CSE 3521 slides with Professor Joe Barker)

IF YOU DON'T UNDERSTAND THE MATH,
DON'T WORRY

- Takes number as input (calculated by adding the weights * raw input)
- Uses an activation function to determine if the number should proceed to the next layer (more on this later)
- If the activation function says so, the number will move onto the next layer as part of that layer's input
- This type of anatomy only applies for hidden layers
- For input neurons, it is simply however your data is represented as a number (in our case, pixel values)
- For output neurons, the output isn't passed to any other layer

Example



a_1 , a_2 , a_3 , a_4 are random input values

w_1 , w_2 , w_3 , w_4 are the weights for each input value

The weighted sum = $(a_1)(w_1) + (a_2)(w_2) + (a_3)(w_3) + (a_4)(w_4)$

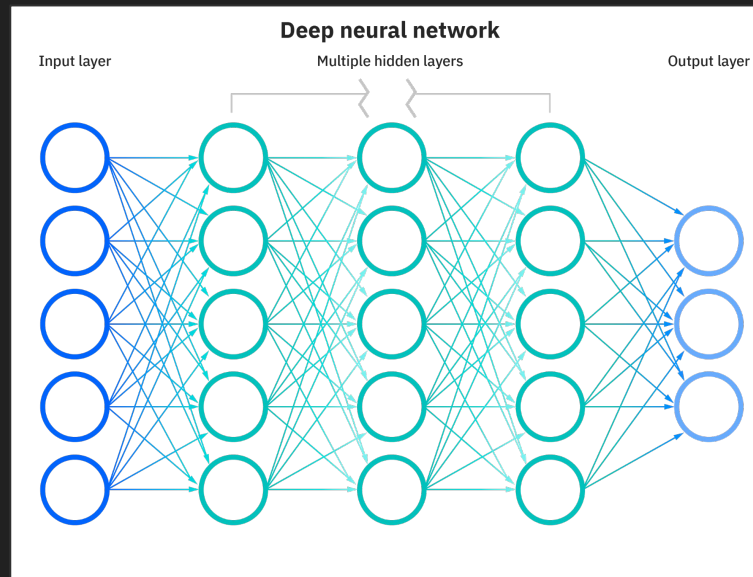
$$= (1)(9) + (-3)(6) + (4)(10) + (2)(7) = 45$$

So the value that green neuron holds is 45, and that's also the value it will pass off to whatever layer is after it



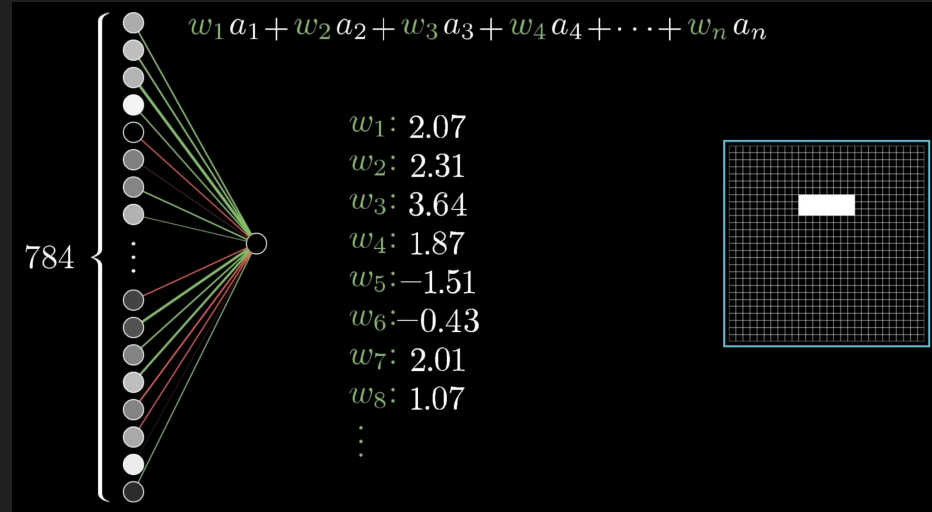
Layers

- Just a group of neurons.
- Neurons in one layer are connected to neurons in the next layer
- 3 types of layers:
 - Input layer: how a piece of data is represented as neurons (in our case, the pixel value of each pixel in a picture)
 - Hidden layer: group of neurons that transform the input image in some way to pick up on features (more on this later)
 - Output layer: a group of neurons that indicate the likelihood of each possible output value



Weighted Connections

- A way to describe connections between layers.
- Consider example to the right.
- Single neuron picks up when an image has those pixels shaded in white
- We want that neuron to be “bright” when the image does have that pattern
- Pixels in that region should have a higher weight, and those not in that region should have a lower weight
- When we take the weighted sum, the result should be higher if that pattern exists and vice versa.



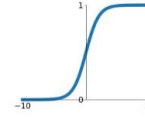
Activation Function

- Non linear function
- The higher the input, the more likely it will pass through
- Lot of different activation functions you can use, but we will be using one called “**ReLU**”
- It essentially says, if the weighted sum is less than or equal to 0, make the value of the neuron 0. Otherwise, make the value of the neuron the weighted sum

Activation Functions

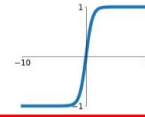
Sigmoid

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



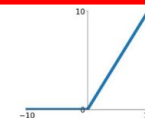
tanh

$$\tanh(x)$$



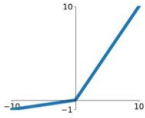
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

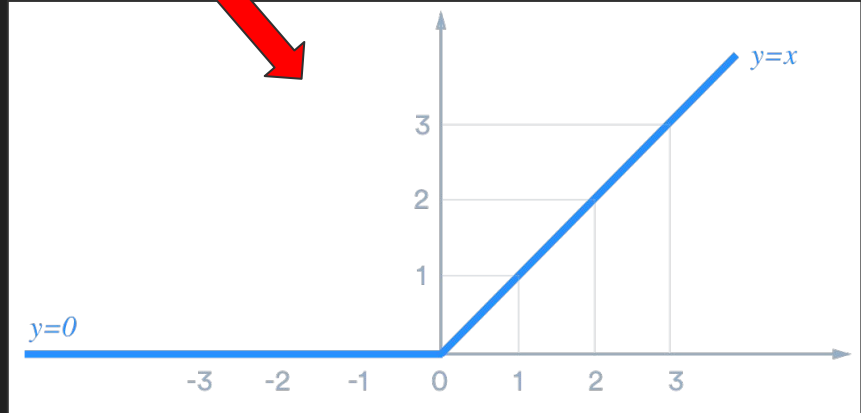
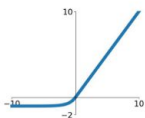


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

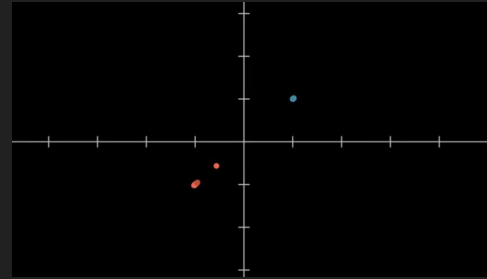
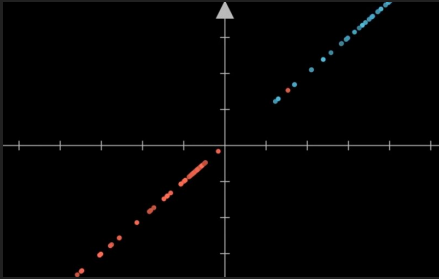
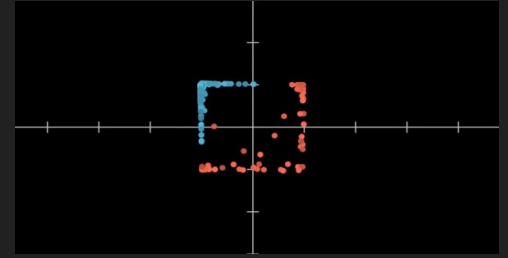
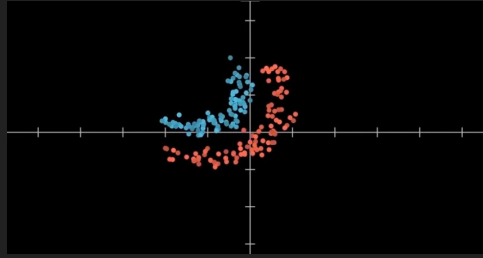
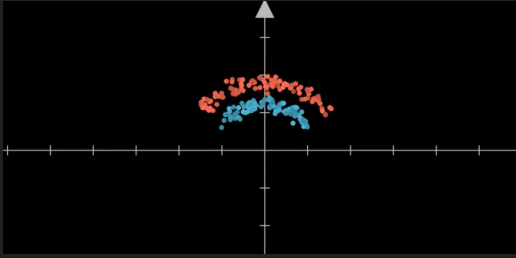
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



What makes up a model? (cont.)

Layers and Activations: Example in a 2D space



The model for our project (Convolutional Neural Network)

▾ Designing the model

```
class EmotionModel(nn.Module):  
    def __init__(self):  
        super(EmotionModel, self).__init__()  
        self.conv_layers = nn.Sequential(  
            nn.Conv2d(1, 32, kernel_size=3, padding=1),  
            nn.ReLU(),  
            nn.MaxPool2d(kernel_size=2, stride=2),  
            nn.Conv2d(32, 64, kernel_size=3, padding=1),  
            nn.ReLU(),  
            nn.MaxPool2d(kernel_size=2, stride=2),  
            nn.Conv2d(64, 128, kernel_size=3, padding=1),  
            nn.ReLU(),  
            nn.MaxPool2d(kernel_size=2, stride=2)  
        )  
        self.fc_layers = nn.Sequential(  
            nn.Linear(128 * 6 * 6, 128),  
            nn.ReLU(),  
            nn.Dropout(0.5),  
            nn.Linear(128, 7) # 7 classes for different emotions  
        )  
  
    def forward(self, x):  
        x = self.conv_layers(x)  
        x = x.view(x.size(0), -1)  
        x = self.fc_layers(x)  
        return x
```

Convolutional Neural Network: a neural network that specializes in image classification.

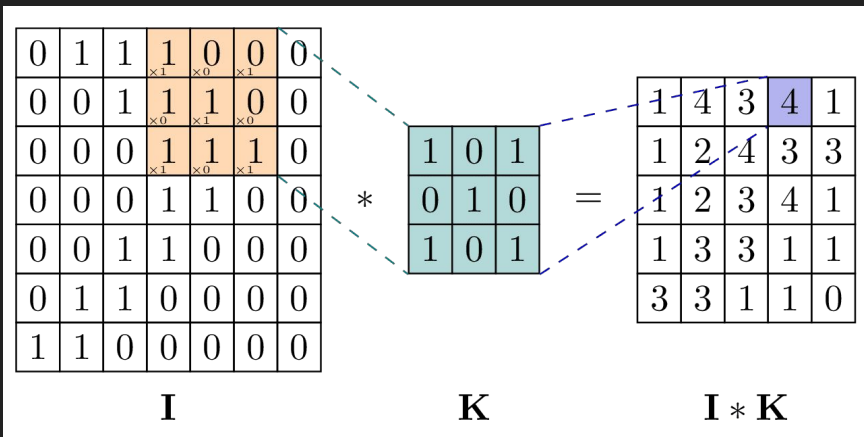
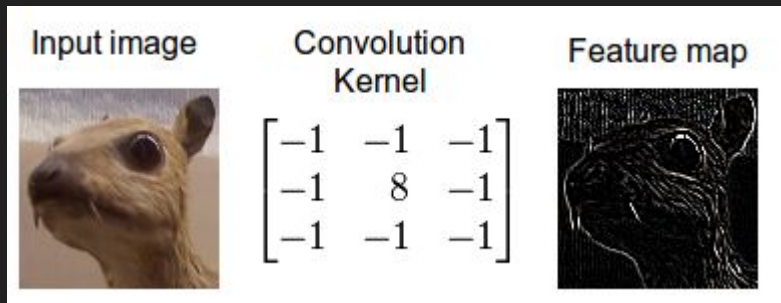
- Achieves image classification by having at least one of the hidden layers do a “convolution” transformation.
- These layers are called “convolution layers”
- More on this later...

Defining the layers in the model:

- Convolutional layers
- Fully connected layers

Defining how a data sample would pass through the layers in the model

The Convolution Layers



```
self.conv_layers = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(32, 64, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(64, 128, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2)
)
```

Convolution

- We define an $m \times n$ matrix that we call a “filter”.
- We slide this filter across every $m \times n$ group in our image.
- We take the “dot product” of these two. If you don’t know what a dot product is, don’t worry. We just wanted to show some of the conceptual stuff.
- Just know convolution picks up on edges

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

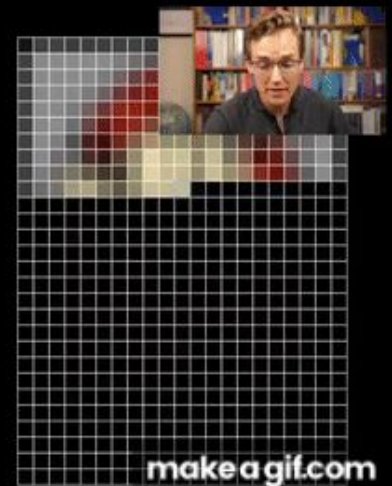
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

Dot Product

$$\begin{bmatrix} a & b \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = [ax + by]$$

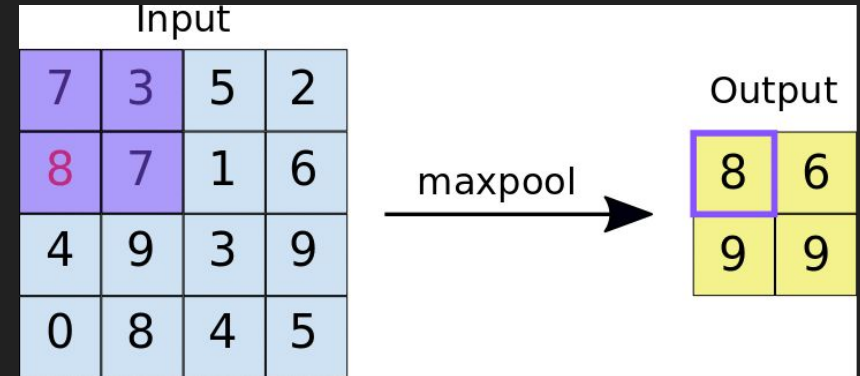
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} aw + by & ax + bz \\ cw + dy & cx + dz \end{bmatrix}$$

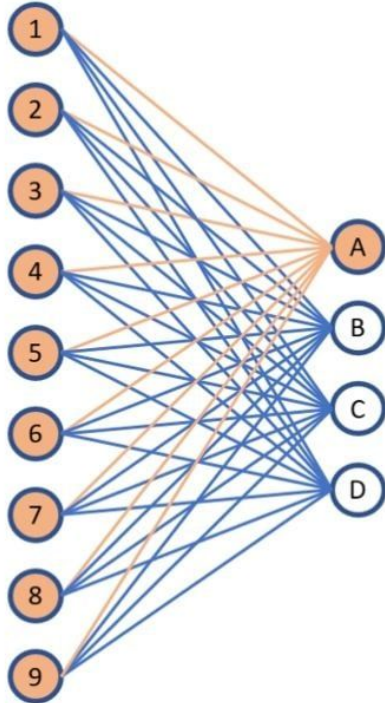


Max Pooling

- We define an $m \times n$ window that we call a “filter”.
- We slide this filter across every $m \times n$ group in our image.
- We take the maximum value from each group
- Reduces size of feature map by filtering out less important features
- Helps generalize (reduce overfitting)
- Makes computation easier for the computer
- In our use of max pooling, we are reducing the size of the feature map by half every time



The Fully Connected Layers



```
self.fc_layers = nn.Sequential(  
    nn.Linear(128 * 6 * 6, 128),  
    nn.ReLU(),  
    nn.Dropout(0.5),  
    nn.Linear(128, 7) # 7 classes for different emotions  
)
```

nn.Linear

- Creates a “fully connected layer” (the stuff we talked about earlier with general NNs)
- First parameter is input size
- Second parameter is the number of neurons in the layer
- The input size is $128 * 6 * 6$:
 - The 128 comes from the amount channels outputted by the convolutional layers (number of feature maps)
 - $6 * 6$ is the size of the feature maps, and it comes from the fact that we used max pooling 3 times ($48/2/2/2 = 6$)

```
nn.Linear(128 * 6 * 6, 128),
```


nn.Dropout

- First (and only) parameter is the likelihood for a neuron to experience drop out
- Dropout is when we decide to make the output of a neuron 0
- In this case, half the neuron's will output 0
- Why?:
 - Model is now less likely to rely heavily on certain neurons (which can lead to overfitting)

```
nn.Dropout(0.5),
```

How will data flow through the model?

The forward function defines this:

- First have the data go through the convolutional layers
- Then flatten the result of all that (to translate the result of the convolutional layers into something the fully connected layers can understand)
- Then pass that to the fully connected layers
- Return the overall result

```
def forward(self, x):  
    x = self.conv_layers(x)  
    x = x.view(x.size(0), -1)  
    x = self.fc_layers(x)  
    return x
```

How do you make your own?

Extend the base class (remember OOP?): [torch.nn.Module](#)

Override the `__init__` method:

- Initialize the superclass (`torch.nn.Module`)
- Declare your “layers”

Override the forward method:

- Call your layers declared in `__init__` with input data
- Return the processed data (in our case, the emotion prediction)

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

How do I train a model?

- Next week :)

But here's some word vomit:

- Loss function
- Optimizer
- Gradient descent
- Backpropagation
- Epochs
- Adam

Your model next week:



For next week:

- At the very minimum, have the data set up from last week.
- There will be code in the GitHub resources repository to prepare to train our models.

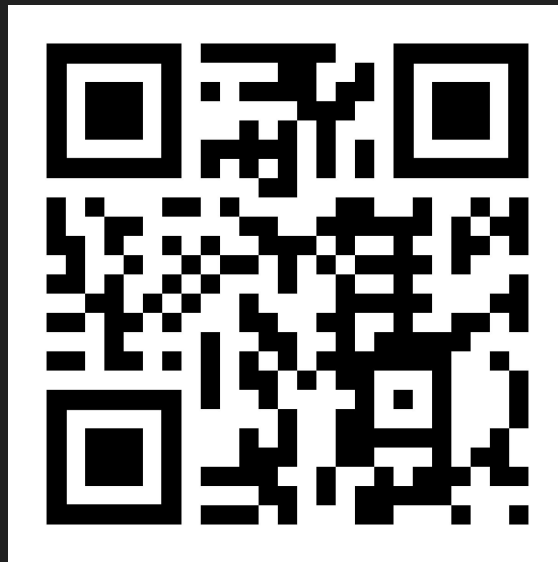
See you next week!



First Time Sign up



AI Club Website





Enjoy your week!

go.osu.edu/aiclubsignup

