MCDB 4110/6440 – Quantitative Microscopy Lab

Lab 4 – 1:

Introduction to deep learning image classification

Lecturer: Jian Wei Tay

Date: 8 November 2021

Introduction to Deep Learning

Artificial Intelligence

Algorithms which allow computers to mimic human behavior

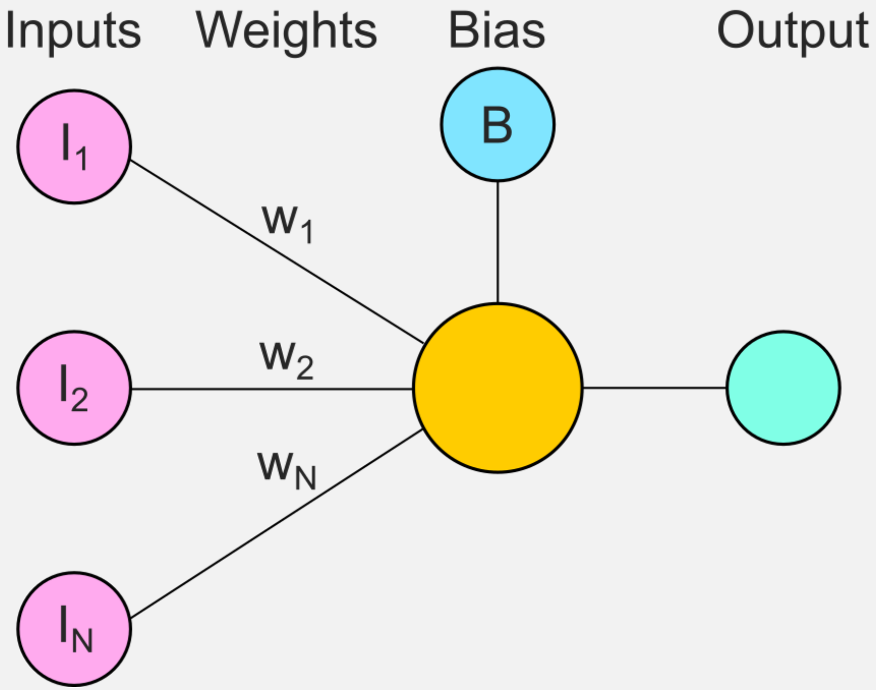
Machine Learning

Algorithms which allow a computer to learn without being explicitly programmed

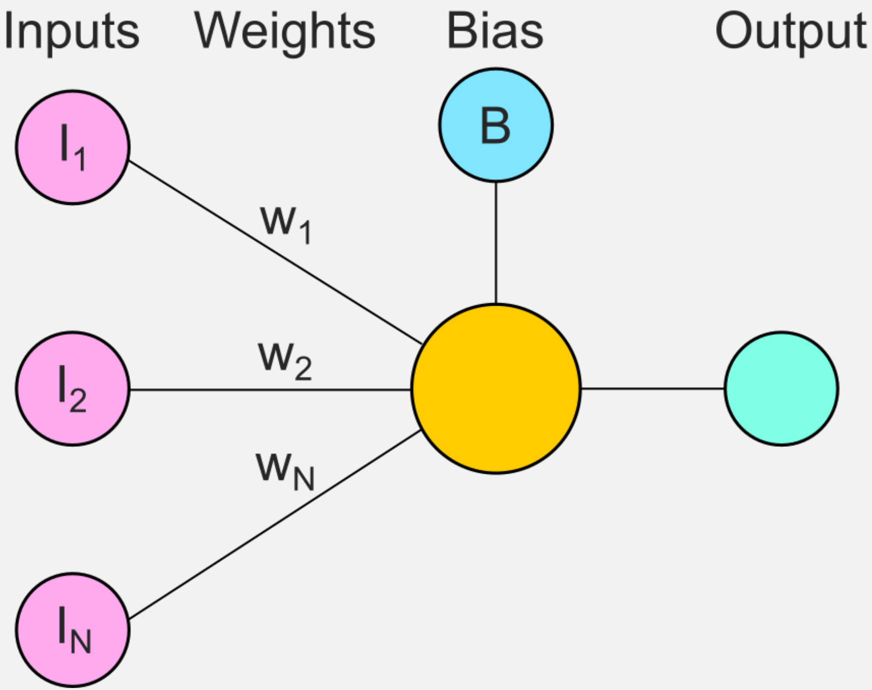
Deep Learning

Algorithms which use neural networks to extract patterns

The perceptron

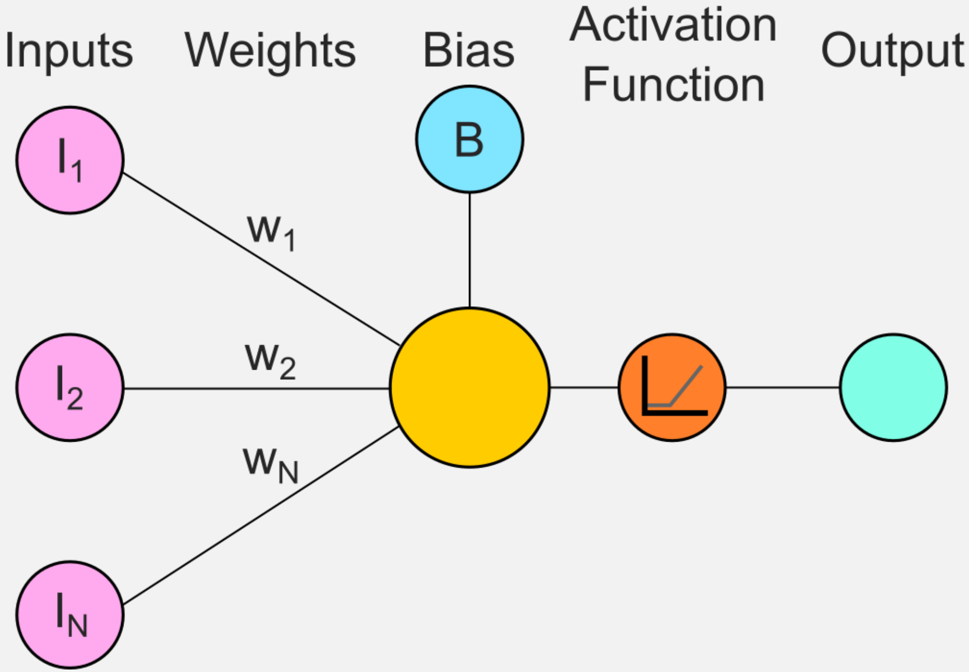


The perceptron



$$\text{Output} = \sum_{N}^{i=1} w_i I_i - B$$

Activation function



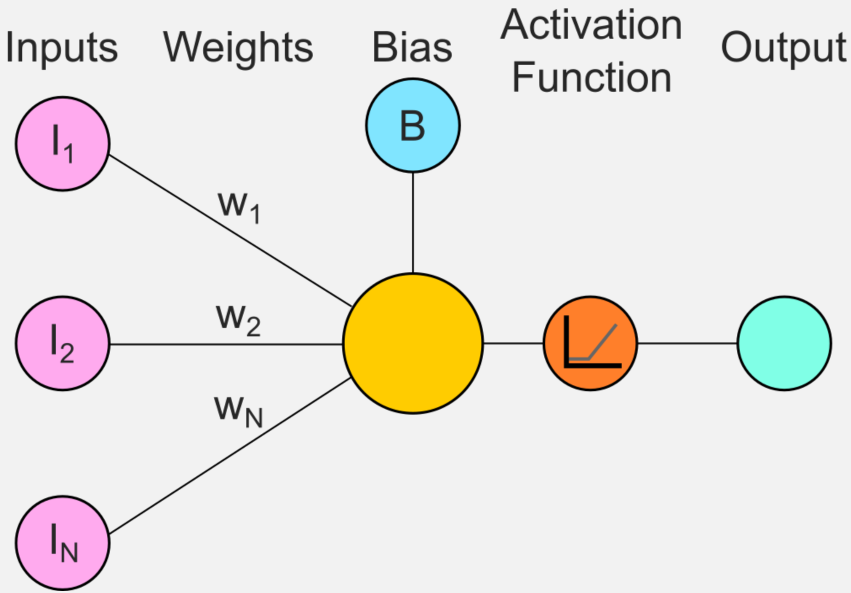
Step activation function

Output:

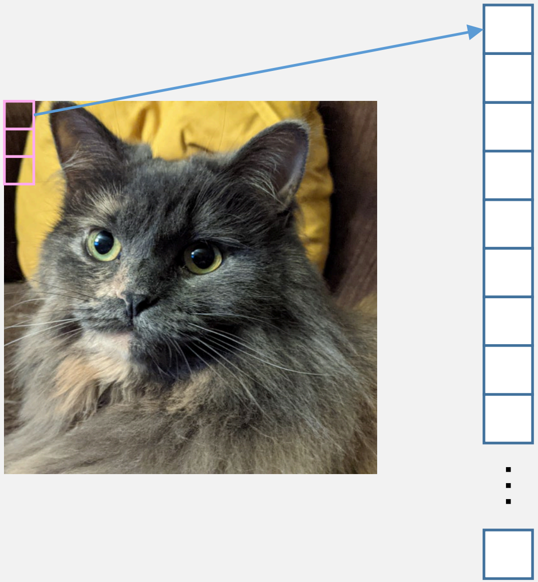
$= 1 \text{ if } \sum_{N}^{i=1} w_i I_i - B > 0$

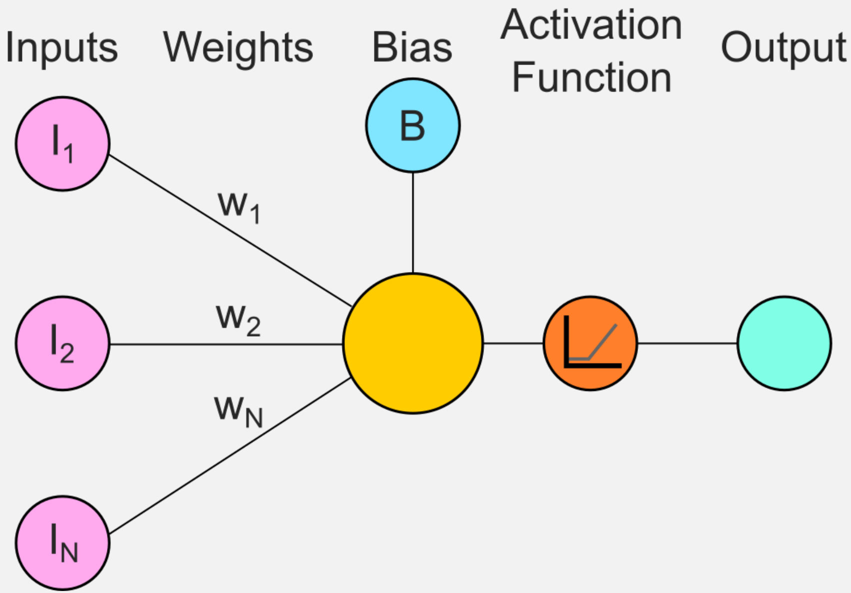
$= 0$ otherwise

Image classification: Is this a cat?



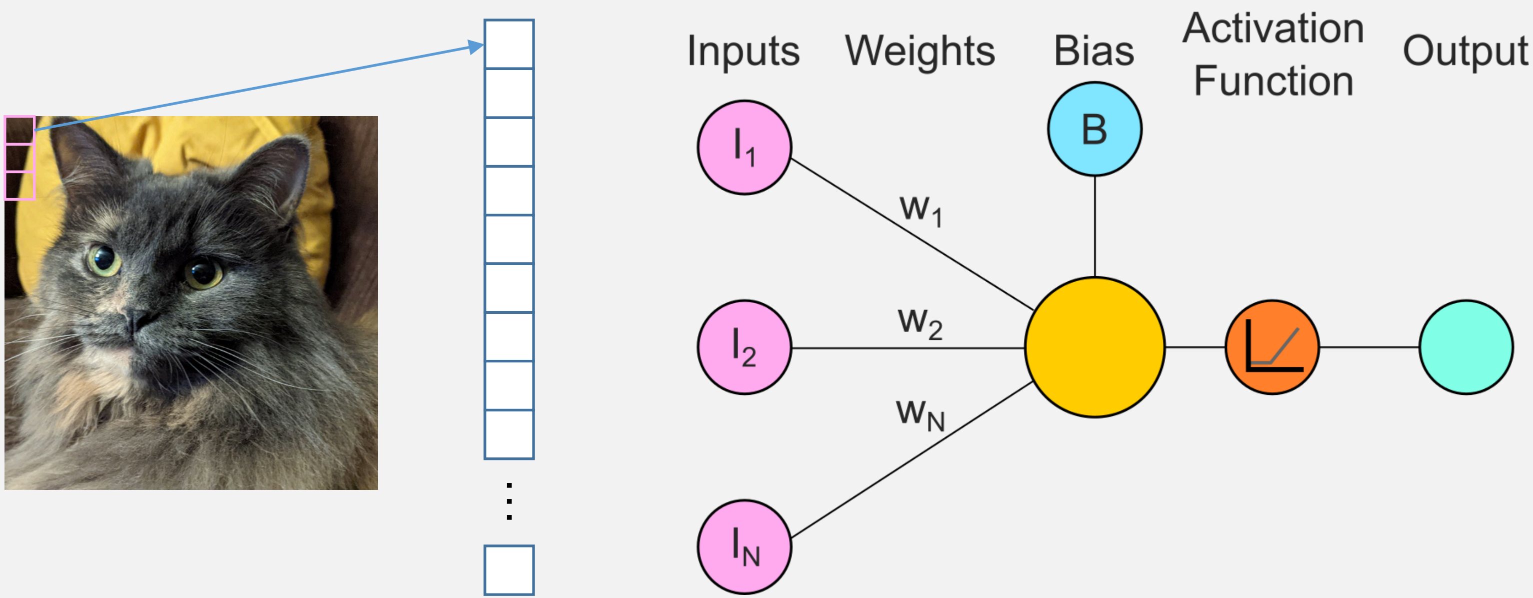
Each input = a pixel in the image



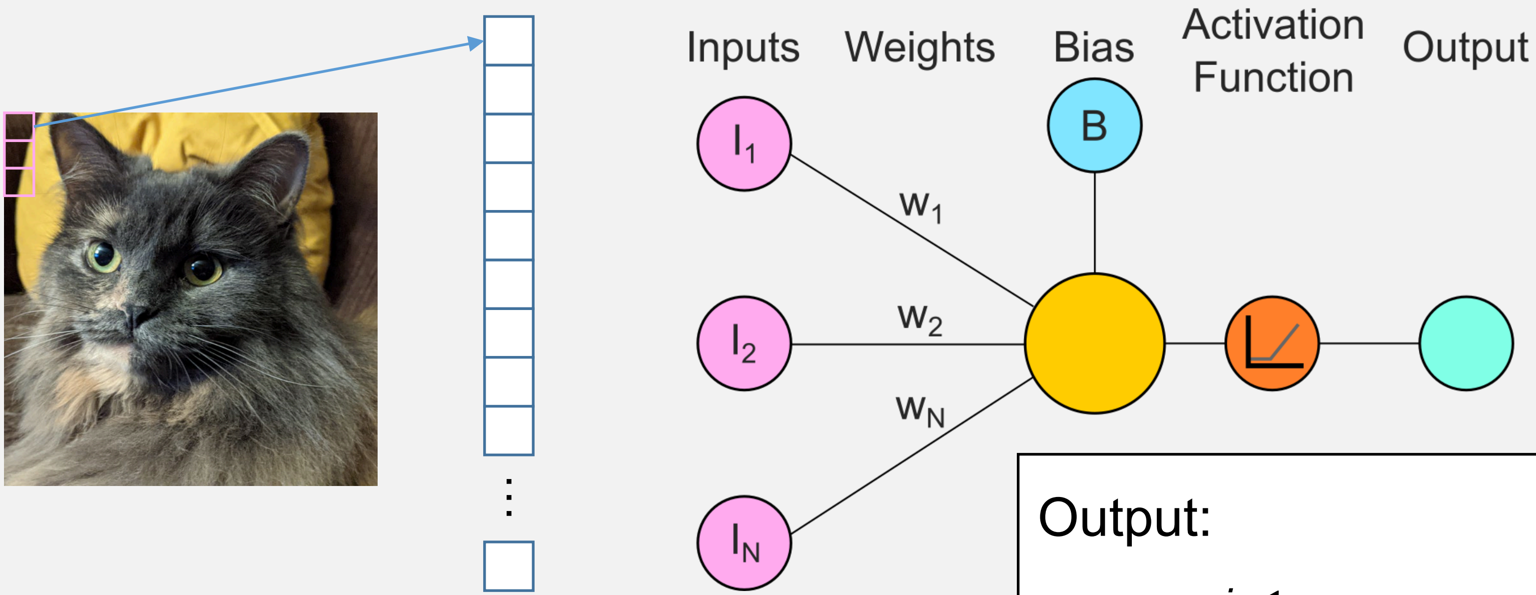


"Linearizing" or "flattening" the image = reshaping the 2D image matrix into a (column) vector

The output should be binary (1 = yes, 0 = no)



The output should be binary (1 = yes, 0 = no)



Output:

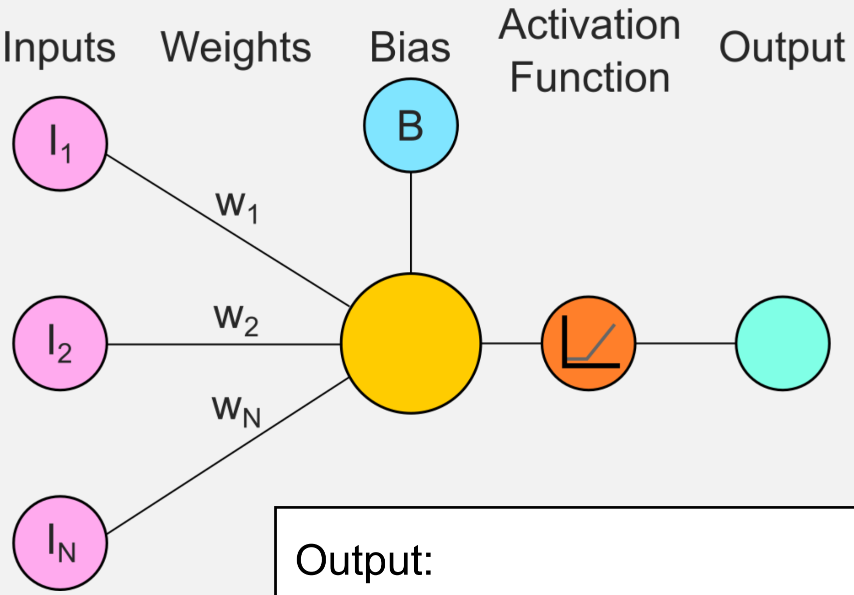
= 1 if $\sum_{N}^{i=1} w_i I_i - B > 0$

= 0 otherwise

Training a deep learning network

is the process of finding the weights and bias which gives you the correct output

Initial weights and bias are randomized

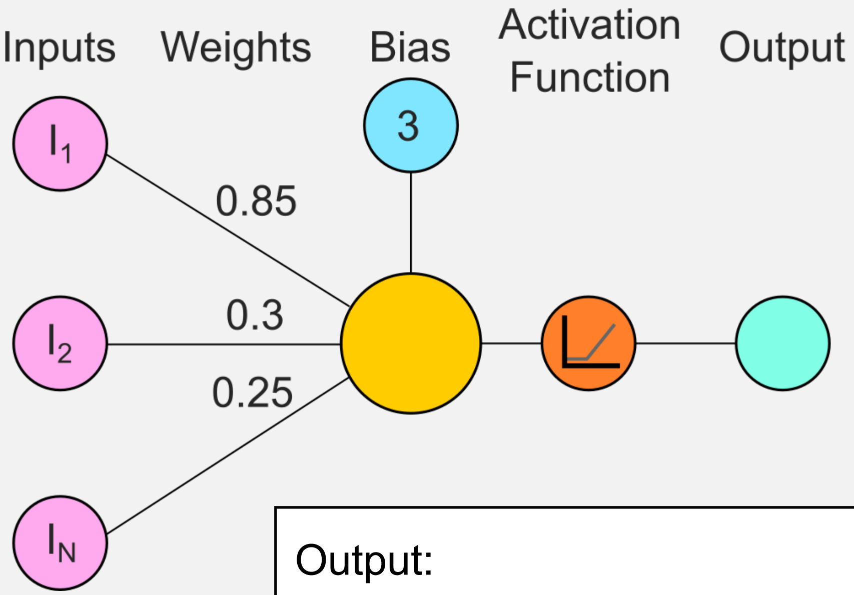


Output:

= 1 if $\sum_{N}^{i=1} w_i I_i - B > 0$

= 0 otherwise

Initial weights and bias are randomized

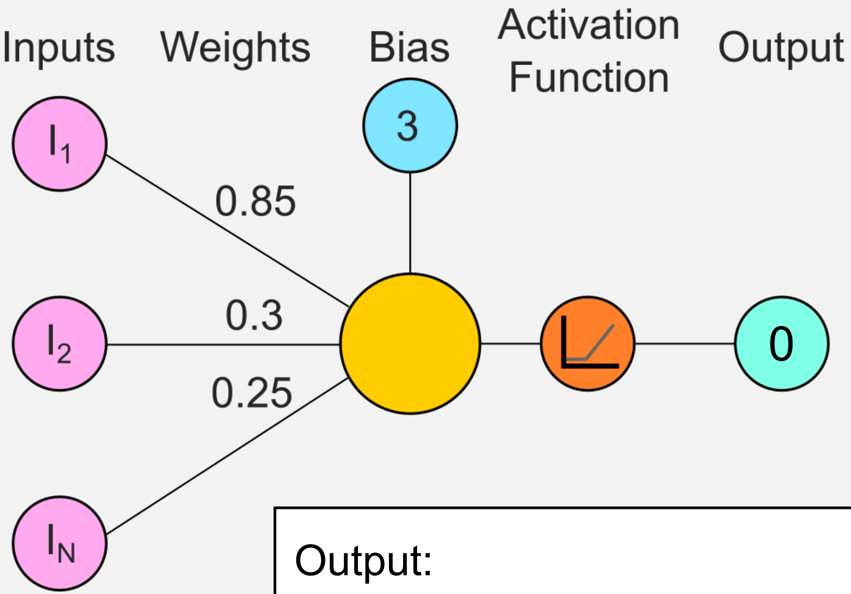


Output:

= 1 if $\sum_{N}^{i=1} w_i I_i - B > 0$

= 0 otherwise

First training picture



Expected output = 1

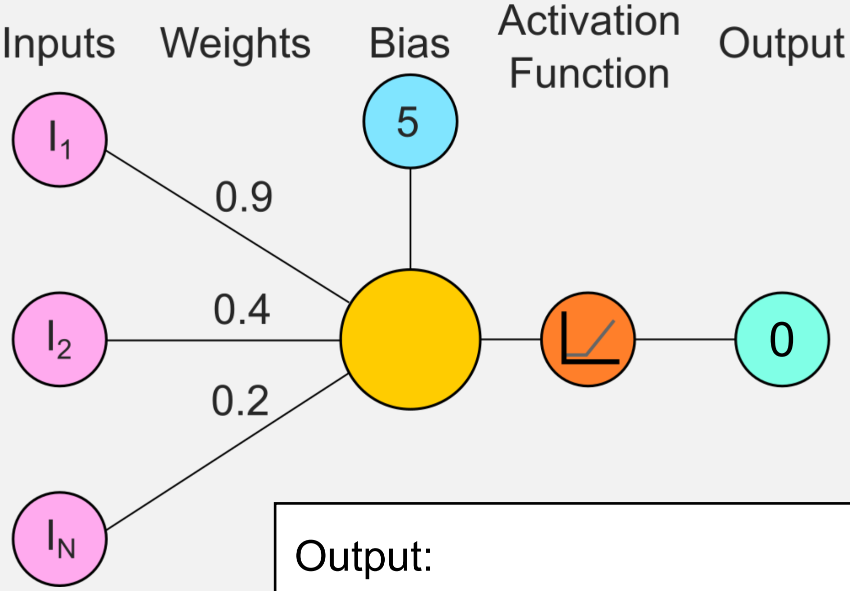
Wrong: Change weights and bias

Output:

= 1 if $\sum_{N}^{i=1} w_i I_i - B > 0$

= 0 otherwise

First training picture



Expected output = 1

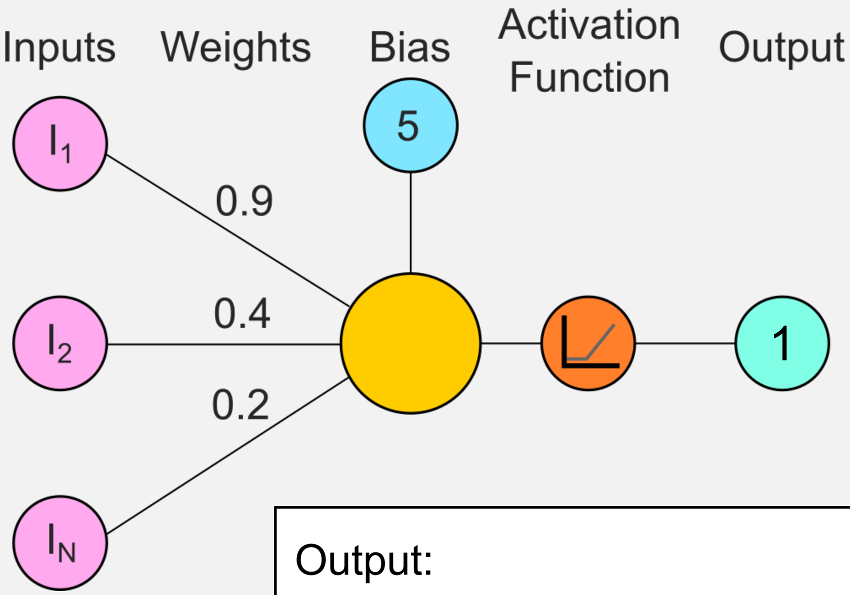
Wrong: Change weights and bias

Output:

= 1 if $\sum_{N}^{i=1} w_i I_i - B > 0$

= 0 otherwise

Second training picture



Output:

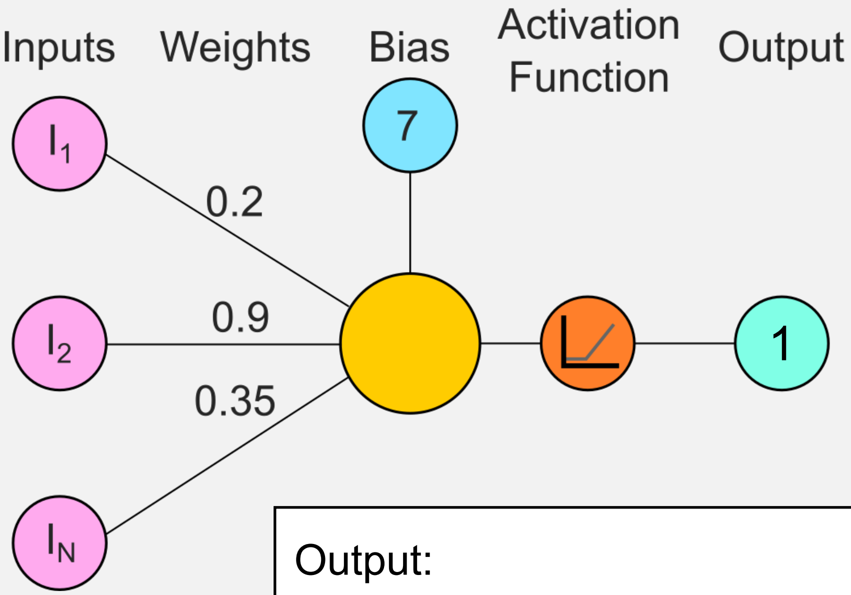
= 1 if $\sum_{N}^{i=1} w_i I_i - B > 0$

= 0 otherwise

Expected output = 0

Wrong: Change weights and bias

Second training picture



Expected output = 0

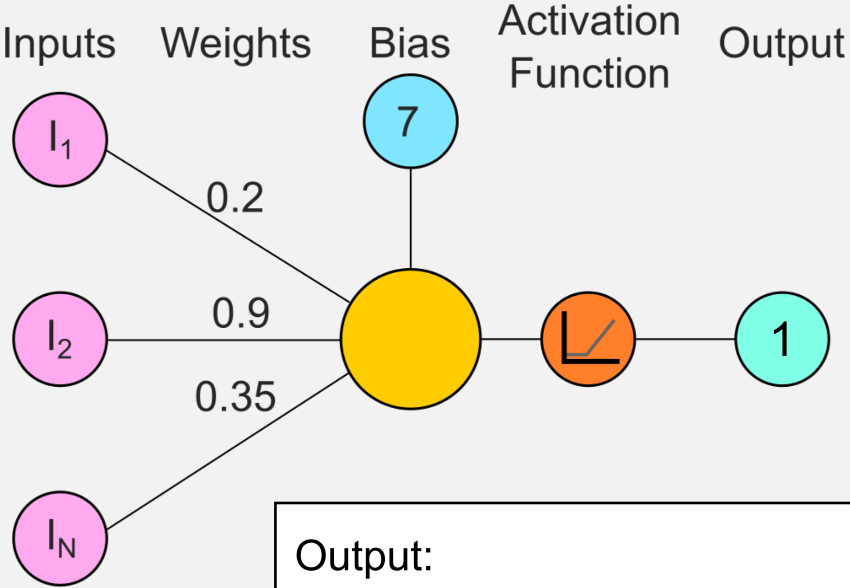
Wrong: Change weights and bias

Output:

= 1 if $\sum_{N}^{i=1} w_i I_i - B > 0$

= 0 otherwise

Third training picture



Expected output = 1

Correct: Do nothing

Output:

$= 1$ if $\sum_{N}^{i=1} w_i I_i - B > 0$

$= 0$ otherwise

A typical training cycle

Input a training image

Compare the output to the expected label

If wrong, update weights and bias by a small amount

Repeat for entire set of training images
= **1 epoch**

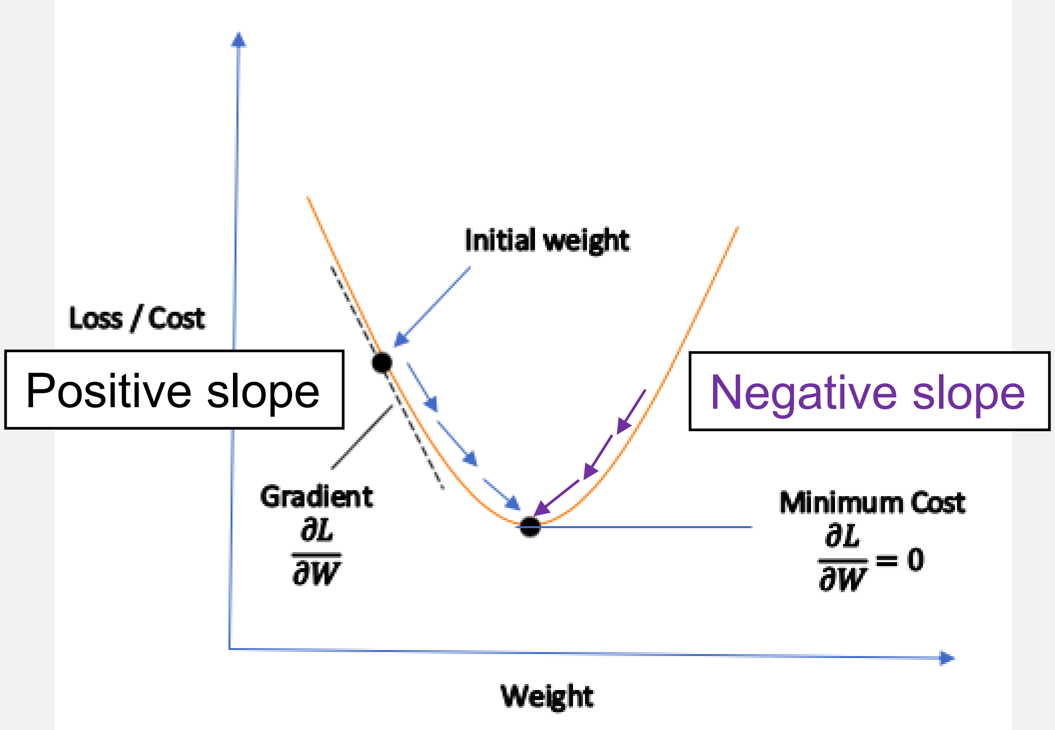
Training runs for a set number of epochs decided by the user

The Stochastic Gradient Descent with Momentum (SGDM) training algorithm

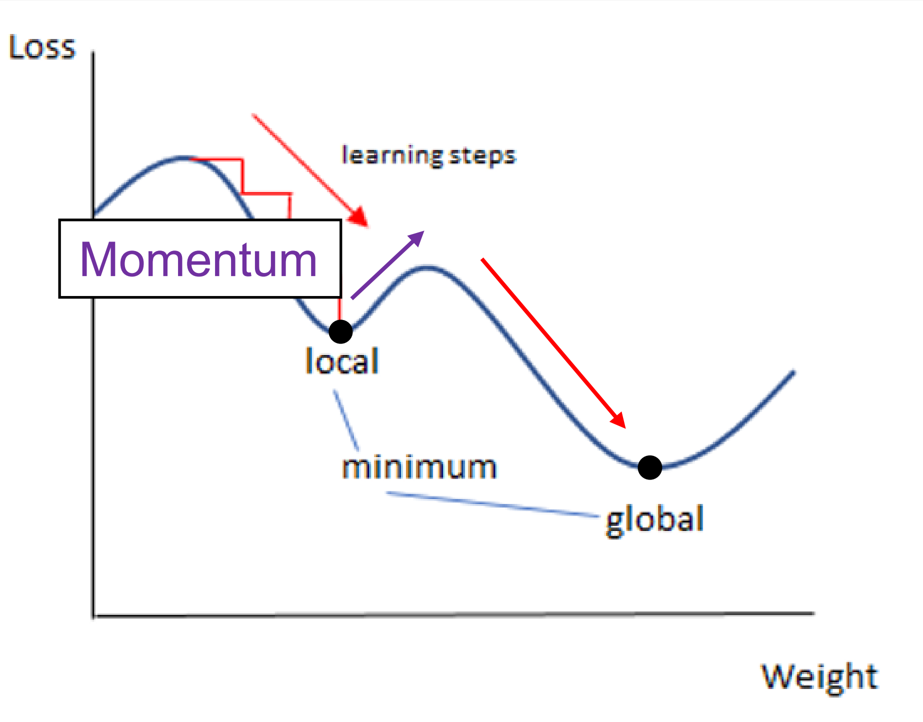
The goal of any training algorithm is to minimize error (aka loss/cost)

$$\text{Error} = |\ \text{Output value} - \text{Expected value}\ |^2$$

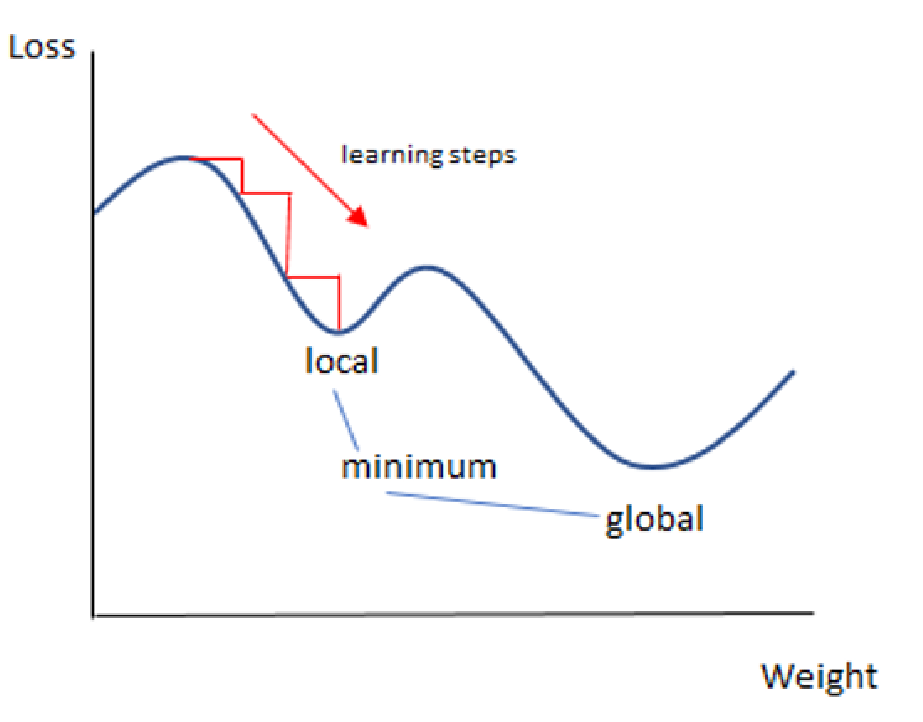
SGDM computes the derivative of the error, then changes the weights based on this slope



Momentum helps to avoid being stuck in a local minimum



Typically want to reduce the size of learning steps as time goes on to help the network settle at the minima



Parameters for SGDM

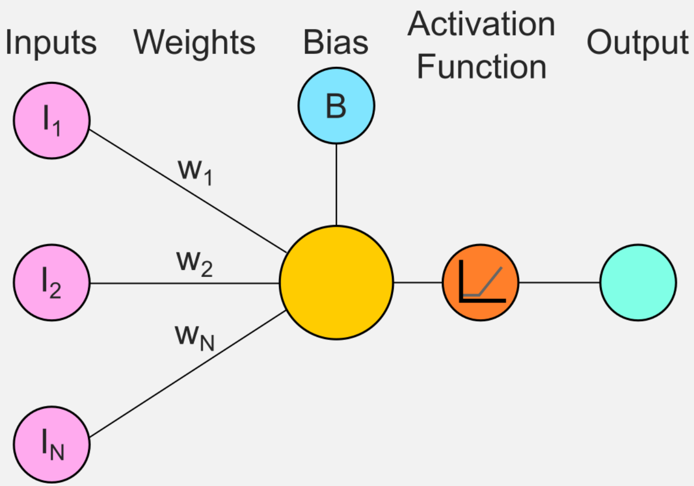
- Learning Rate – How big a step to change the weights when output is wrong

- Learning Rate Drop Factor – Factor to reduce learning rate after each epoch

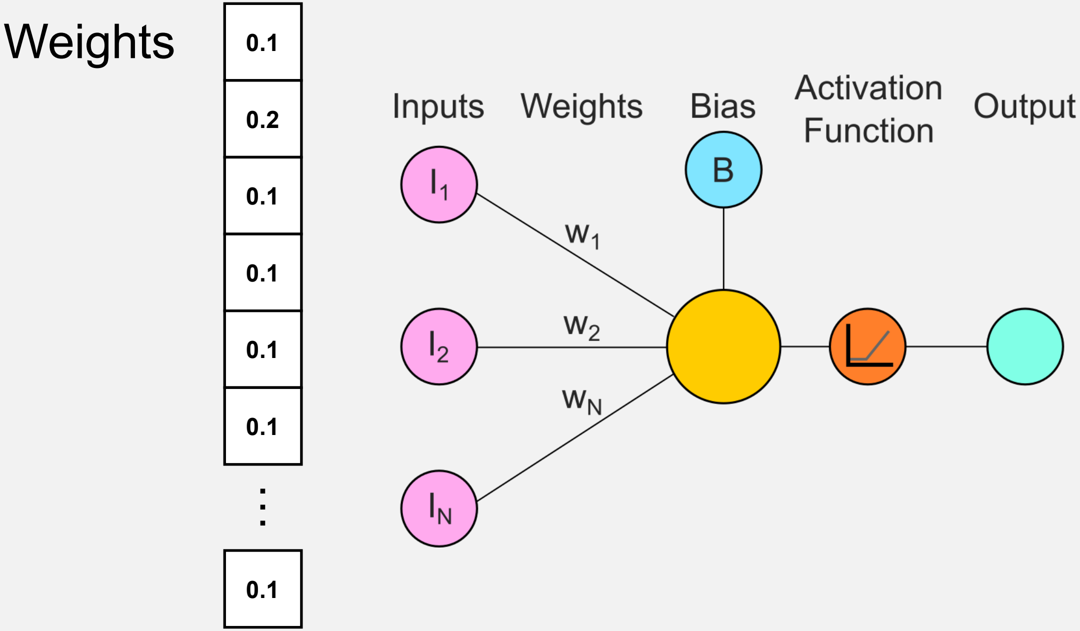
Understanding how the perceptron works after training

Weights

0.1
0.2
0.1
0.1
0.1
0.1
⋮
0.1



The weights indicate how important an input is in getting the classification correct



Reshape weights into a matrix for visualization

0.1	0.2	0.1	0.1	0.1	0.1	0.1
0.1	1	3	5	2	1	0.3
0.3	2	1	2	8	2	0.3
0.1	1	6	5	3	3	0.1
0.2	3	8	7	3	1	0.1
0.1	2	1	2	3	2	0.2
0.1	0.2	0.1	0.2	0.1	0.3	0.1

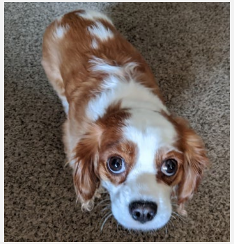
Weights are higher at the center of the matrix = the center of the image is more important

The weights indicate how important an input is in getting the classification correct

0.1	0.2	0.1	0.1	0.1	0.1	0.1
0.1	1	3	5	2	1	0.3
0.3	2	1	2	8	2	0.3
0.1	1	6	5	3	3	0.1
0.2	3	8	7	3	1	0.1
0.1	2	1	2	3	2	0.2
0.1	0.2	0.1	0.2	0.1	0.3	0.1

Weights are higher at the center of the matrix = the center of the image is more important

Final accuracy is dependent on training images



0.1	0.2	0.1	0.1	0.1	0.1	0.1
0.1	1	3	5	2	1	0.3
0.3	2	1	2	8	2	0.3
0.1	1	6	5	3	3	0.1
0.2	3	8	7	3	1	0.1
0.1	2	1	2	3	2	0.2
0.1	0.2	0.1	0.2	0.1	0.3	0.1

Training images are centered on the object we are trying to classify

This perceptron will work well for images that are centered on the object

Will likely misclassify this image

The bias indicates how likely the perceptron will classify the object as a cat

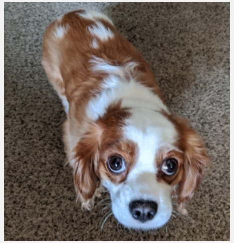
Output:

$= 1$ if $\sum_{N}^{i=1} w_i I_i - B > 0$

$= 0$ otherwise

- If B is high, the perceptron will be less likely to classify an image as a cat because the summation part needs to have a higher value to meet the threshold

The bias is influenced by the ratio of cat vs non-cat images in the training set



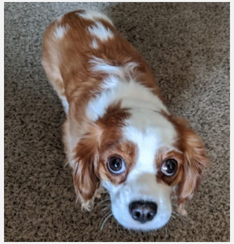
More cats in training images will likely result in a lower bias

Why? Because the perceptron will learn that the image it's shown is more likely to be a cat then not

Need to have **balanced classes** when training

From perceptron to network

Final accuracy is dependent on training images



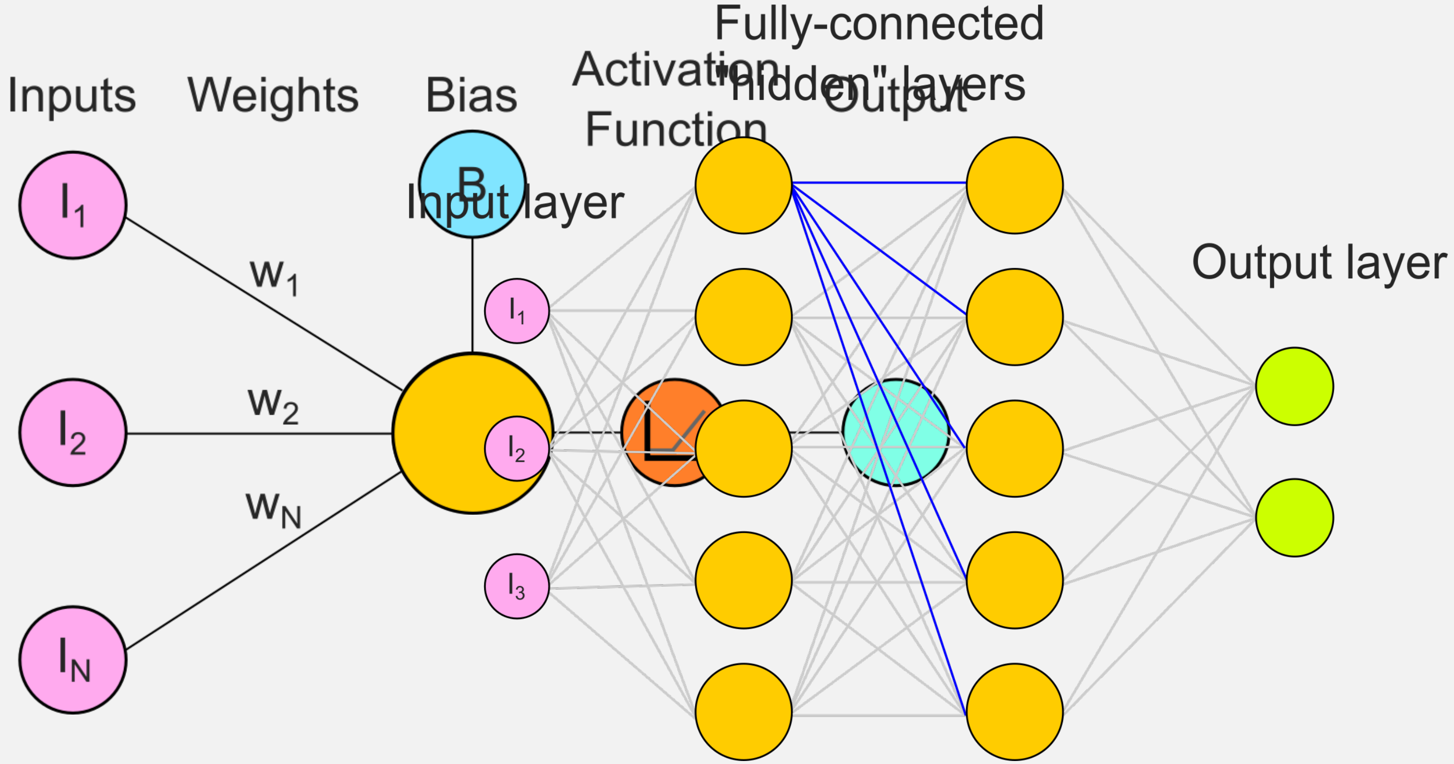
0.1	0.2	0.1	0.1	0.1	0.1	0.1
0.1	1	3	5	2	1	0.3
0.3	2	1	2	8	2	0.3
0.1	1	6	5	3	3	0.1
0.2	3	8	7	3	1	0.1
0.1	2	1	2	3	2	0.2
0.1	0.2	0.1	0.2	0.1	0.3	0.1

Training images are centered on the object we are trying to classify

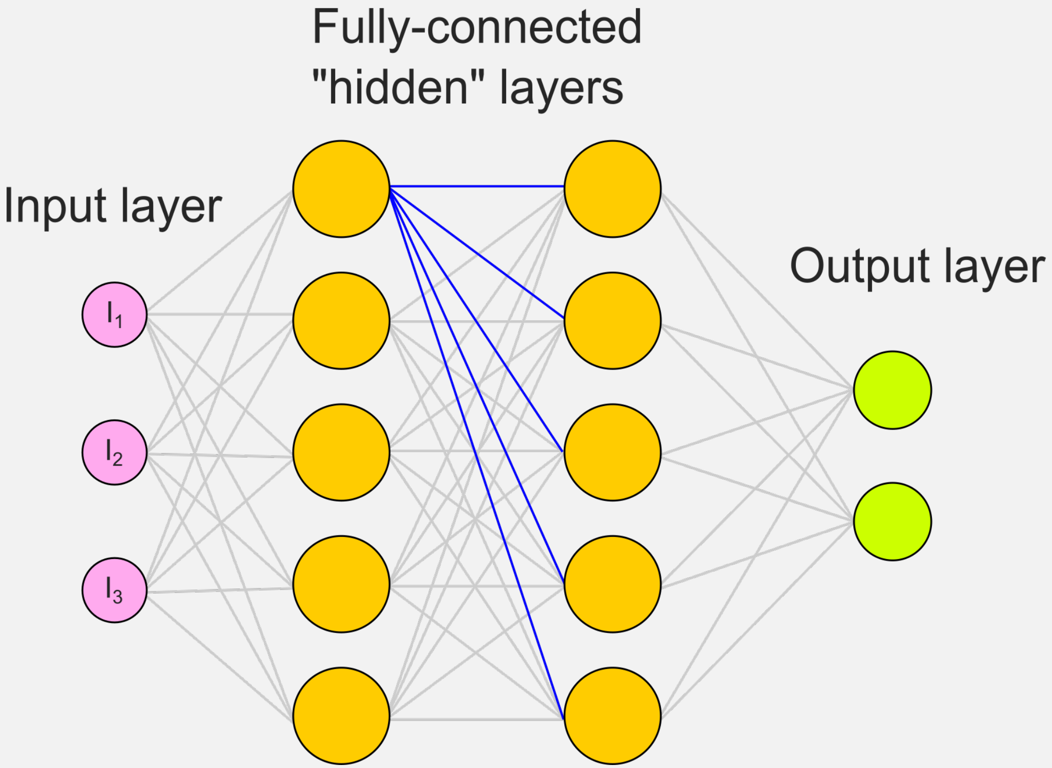
This perceptron will work well for images that are centered on the object

How do we improve the perceptron model to handle other images?

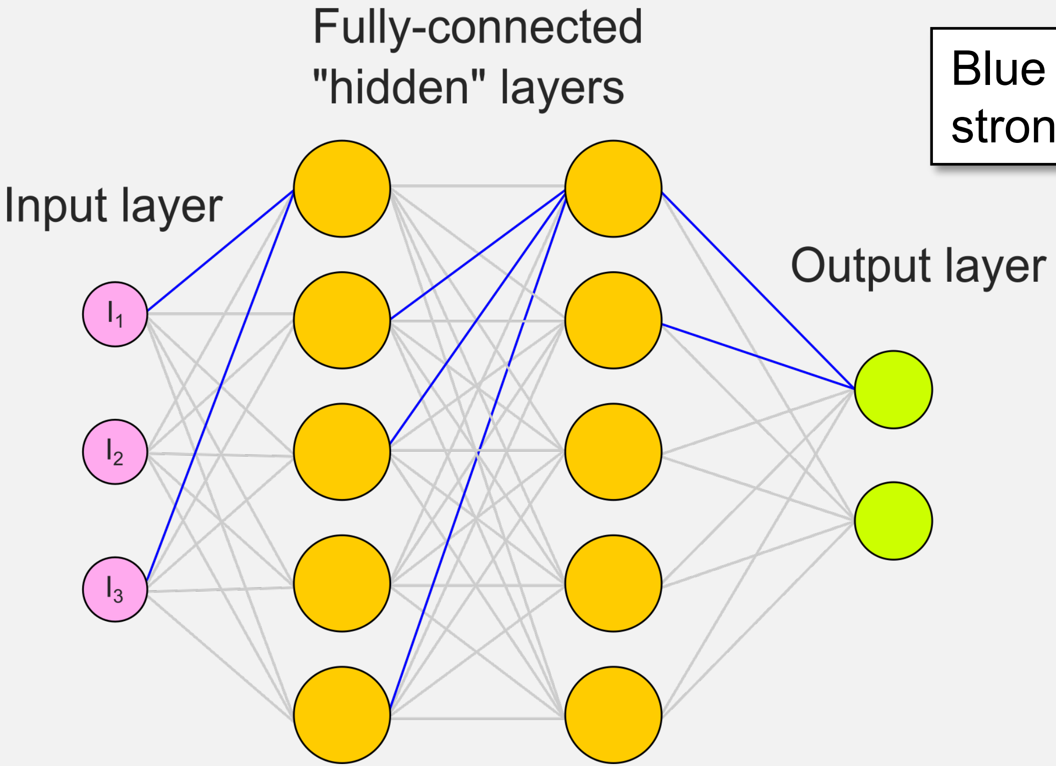
Neural networks have connected layers of perceptrons



Neural networks have connected layers of perceptrons

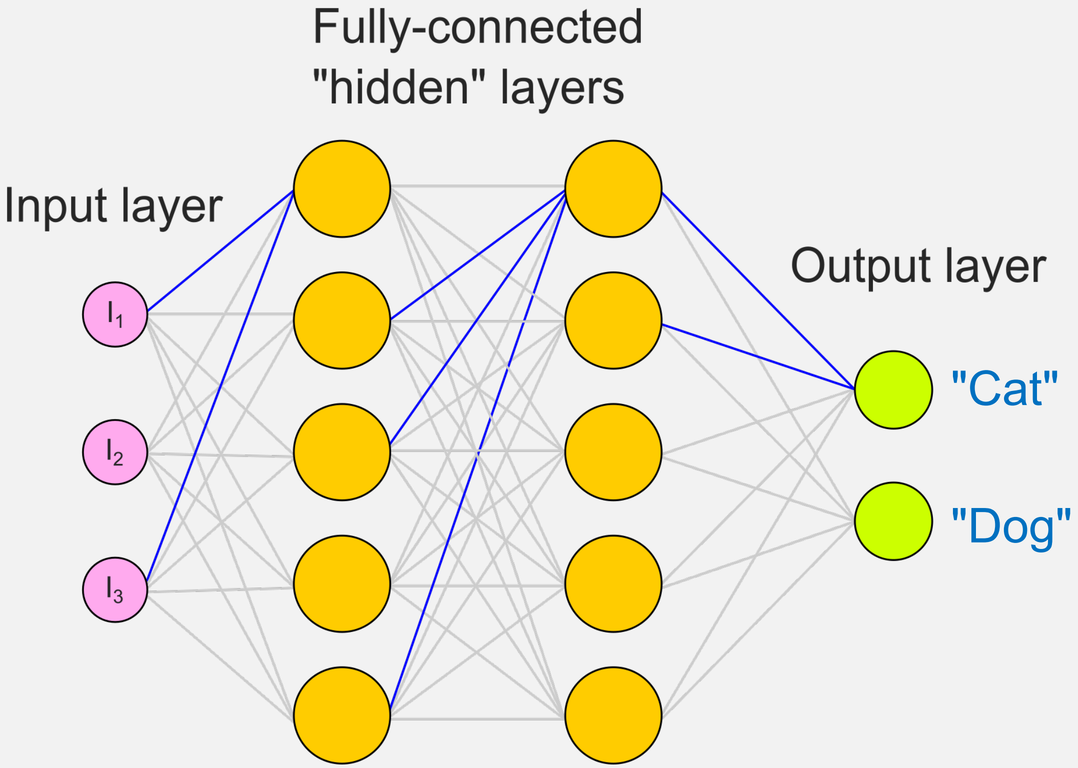


Each layer "mixes" pixels/features from the previous layer



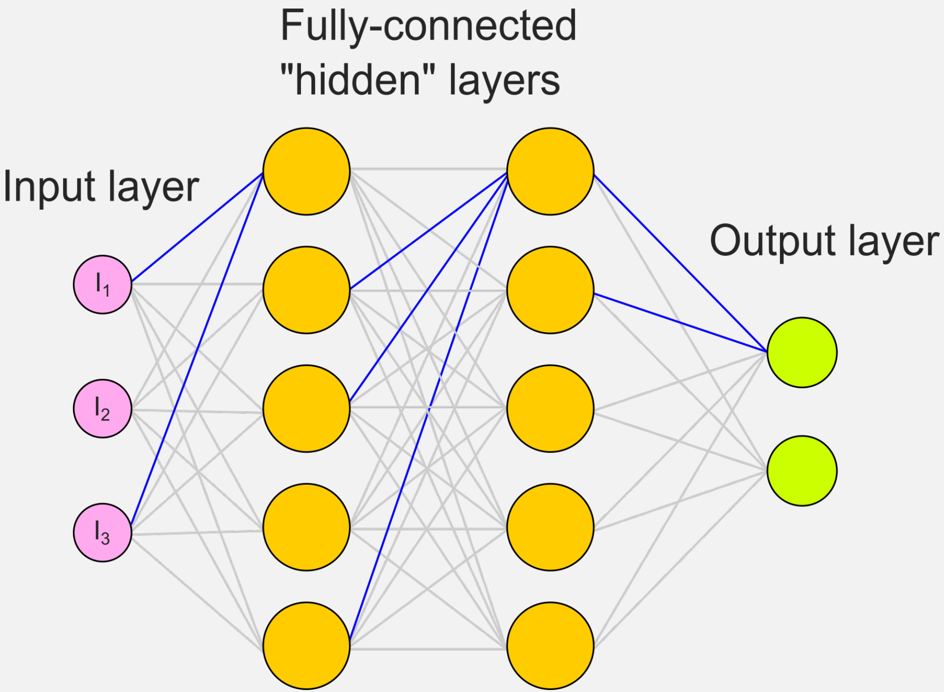
Blue lines indicate strongest weights

The output layer allows different classifications

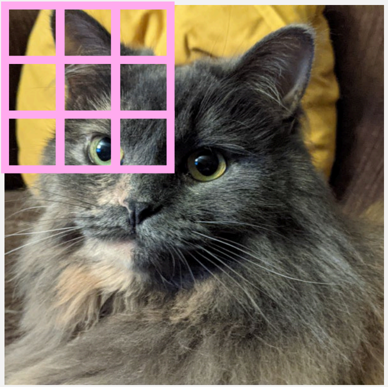


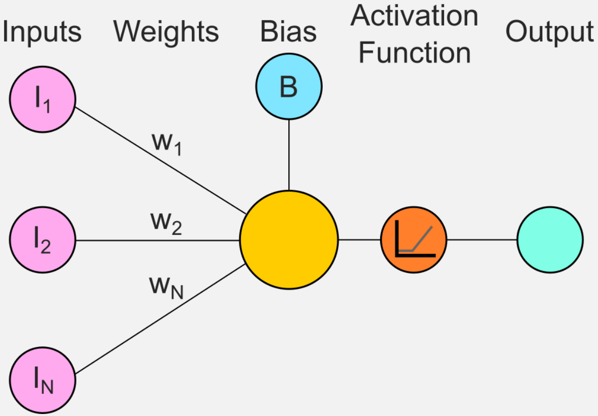
Convolutional Neural Networks

In a fully connected layer, every input is connected to the perceptrons in the next layer



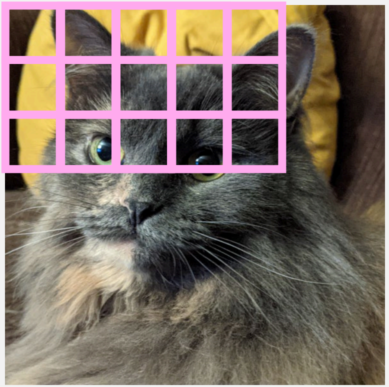
In a convolutional layer, only a patch of an image is used as input

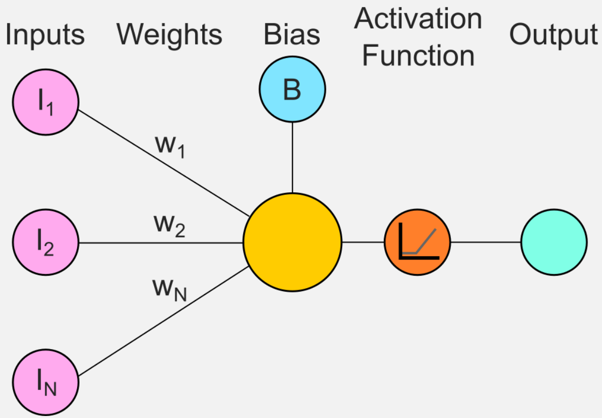




Each perceptron is called a "filter"

The filter is moved over the entire image and the output is stored in a matrix



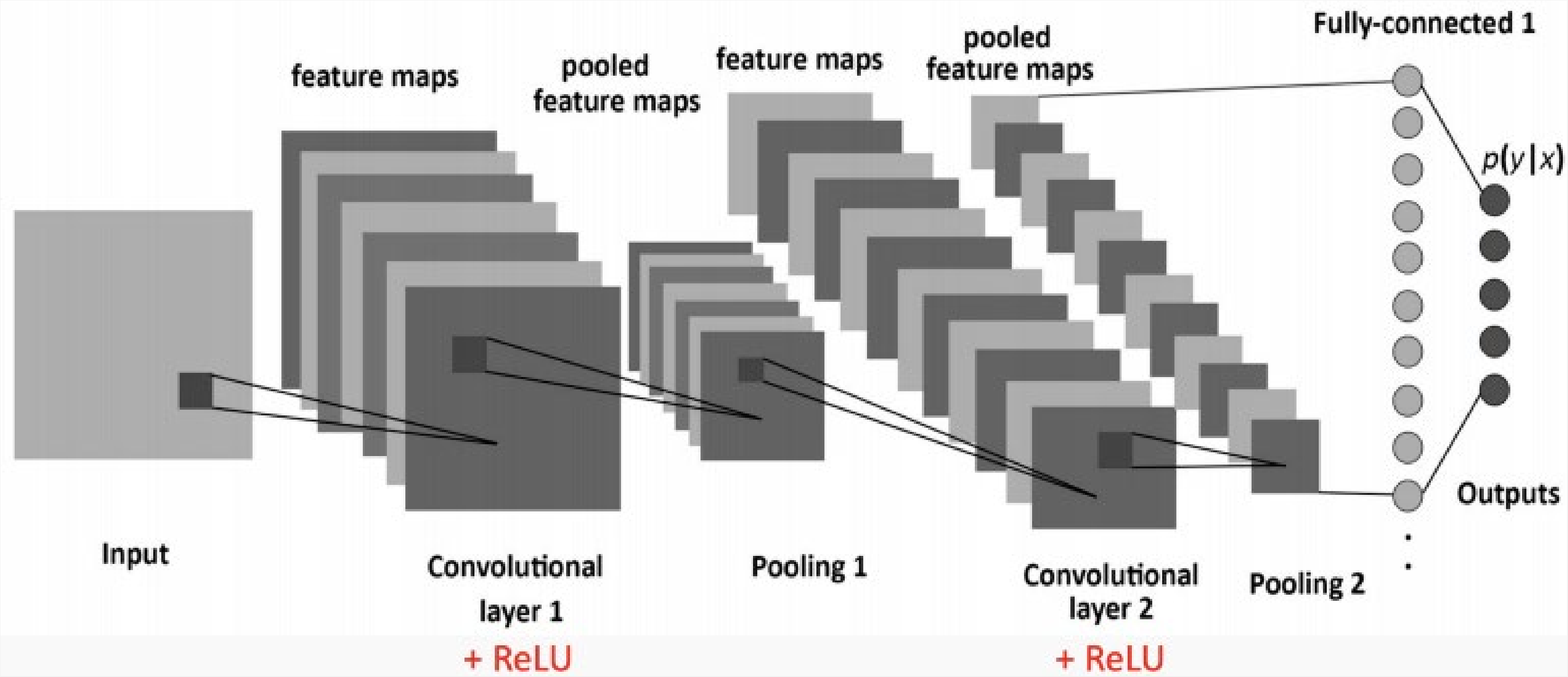


-1	5	2		

The process of sliding the filter over the entire image is called a convolution operation

This matrix is called a "feature map" or "activation map"

Pooling is used to combine pixels in the feature maps (e.g., combine 2x2 pixels into 1)



Example: Edge detection (related method)

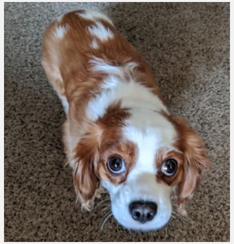
Edge detection

Kernel

$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

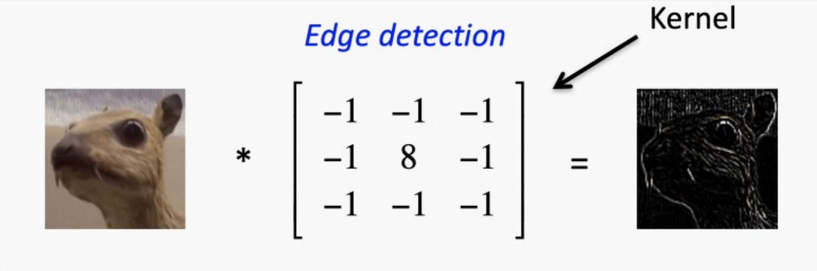
Fully-connected networks vs Convolutional networks

FCNs detect features based on <u>position</u> in an image

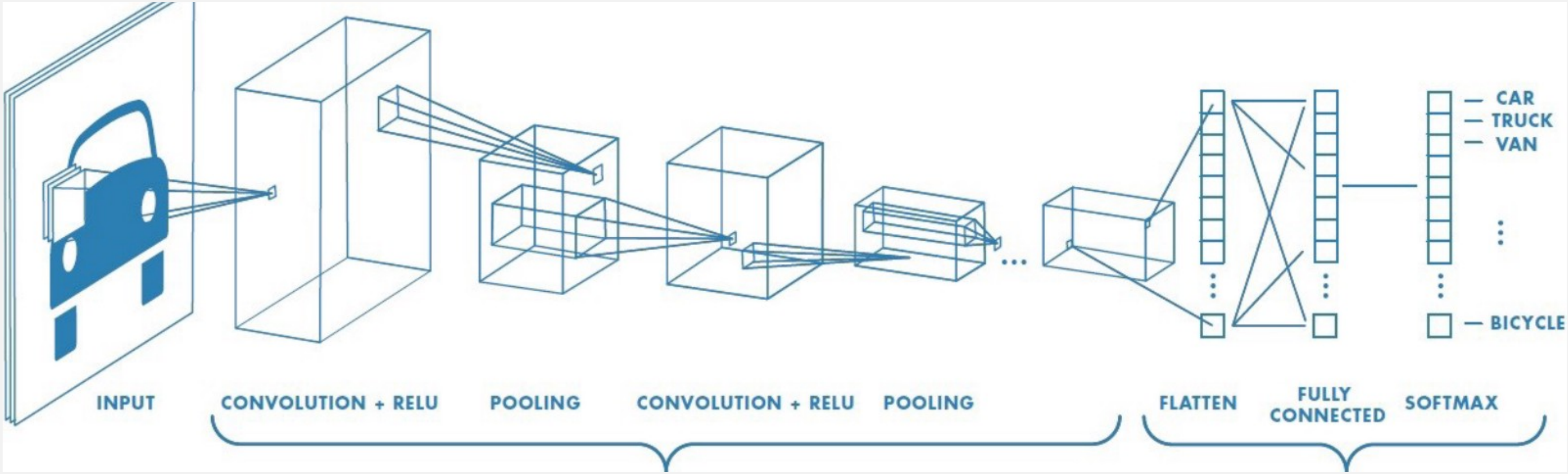


0.1	0.2	0.1	0.1	0.1	0.1	0.1
0.1	1	3	5	2	1	0.3
0.3	2	1	2	8	2	0.3
0.1	1	6	5	3	3	0.1
0.2	3	8	7	3	1	0.1
0.1	2	1	2	3	2	0.2
0.1	0.2	0.1	0.2	0.1	0.3	0.1

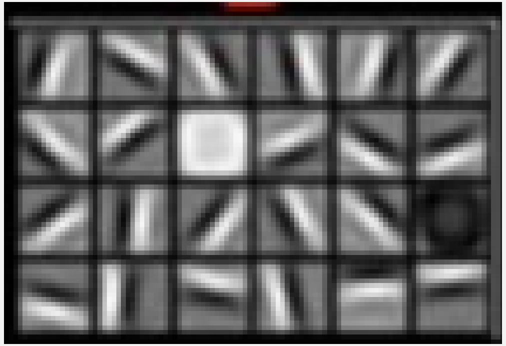
By sliding a filter around, a convolutional network detects features <u>anywhere</u> in an image



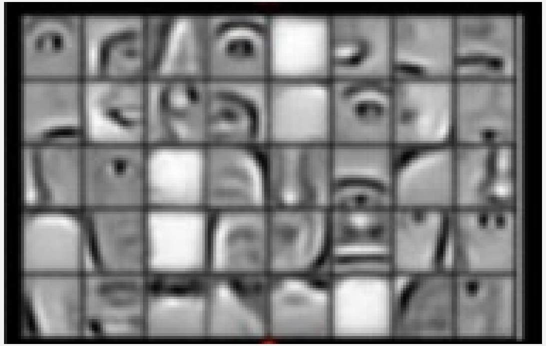
Convolutional neural networks have several layers of convolutional layers



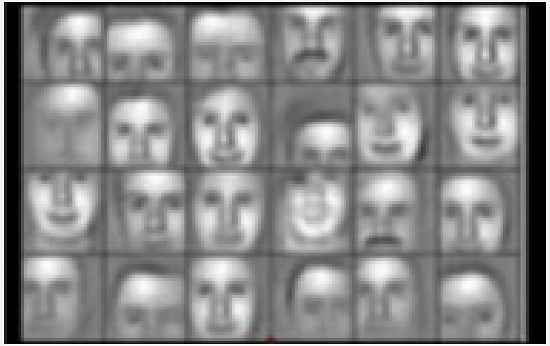
Each convolutional layer combines features from the previous



Layer 1
Detect lines and edges



Layer 2
Combine lines and edges to detect eyes, ears, noses



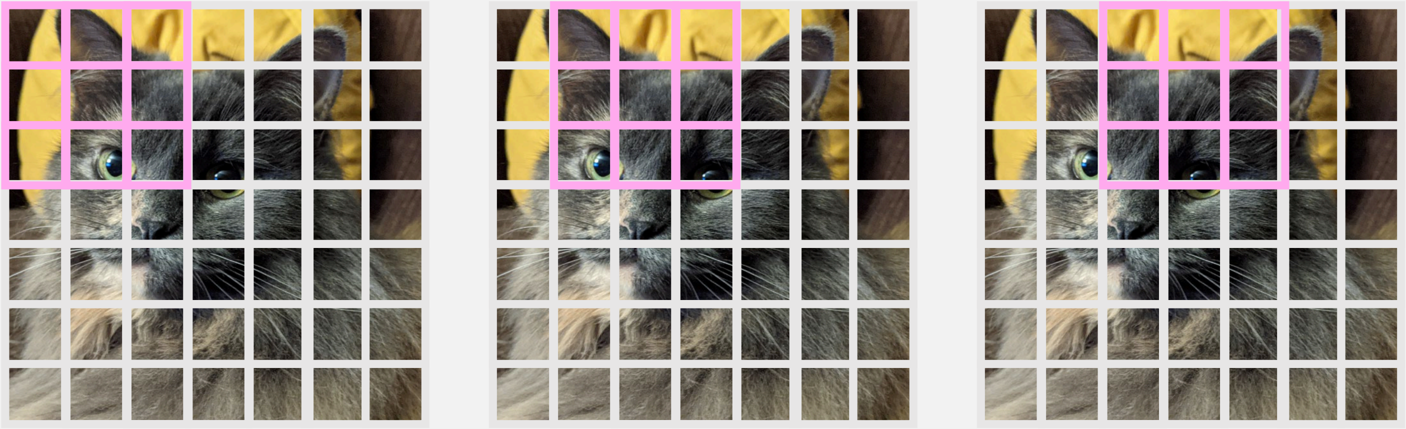
Layer 3
Combine eyes, ears, noses to detect faces

Convolutional layer parameters

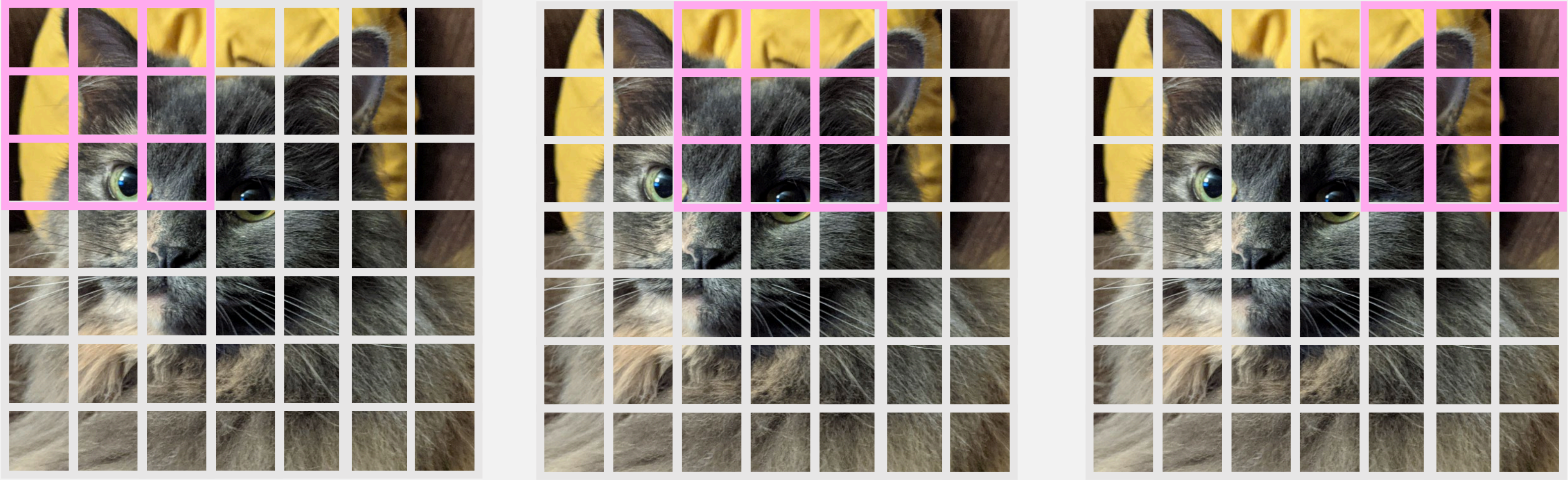
- **Filter size** – specified as [M, N] (usually square). Number of rows and columns for each filter

- **Stride** – step size for traversing the input (usually 1, 1)

Stride of [1, 1]



Stride if [2, 2]



Reference

- MIT 6.S191: Convolutional Neural Networks (Alex Amini) [Youtube]

Training a convolutional network in MATLAB to classify images of food

What we'll do for this lab

- Load image data into MATLAB using an **imageDatastore**
- Use the DeepNetworkDesigner to build a simple network
- Train the network
- Test that it works

For this example, delete *_salad and sashimi folders to reduce number of classes to train

Download the training data

```
fprintf("Downloading Example Food Image data set (77 MB)... ")
filename = matlab.internal.examples.downloadSupportFile('nnet', ...
    'data/ExampleFoodImageDataset.zip');
fprintf("Done.\n")

filepath = fileparts(filename);
dataFolder = fullfile(filepath,'ExampleFoodImageDataset');
unzip(filename,dataFolder);
```

Load the training data

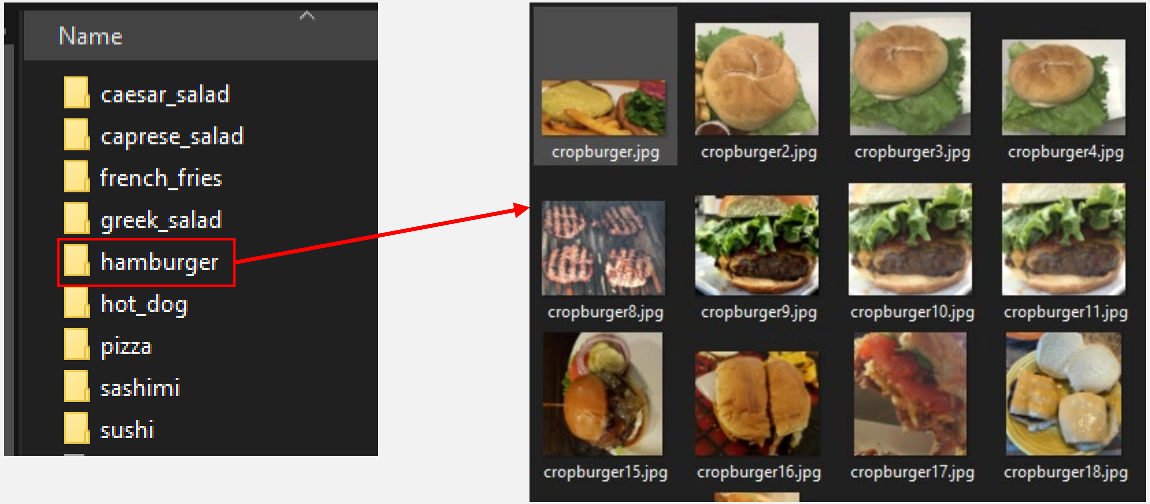
- We are going to use a dataset of images of food



- MATLAB has done all the hard work of preparing the data for us

Folder structure of image data

- The folder `ExampleFoodImageDataset` (created when you unzipped the files) contains 9 folders
- The folders are named with a label that describes each image within it



Load the images into an imageDatastore

```
imds = imageDatastore(dataFolder, ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
```

The `imageDatastore`

- Last week, we talked about how we cannot simply load an entire dataset into memory due to RAM
- The `imageDatastore` object will index all the images in a directory and load them into memory when they are needed

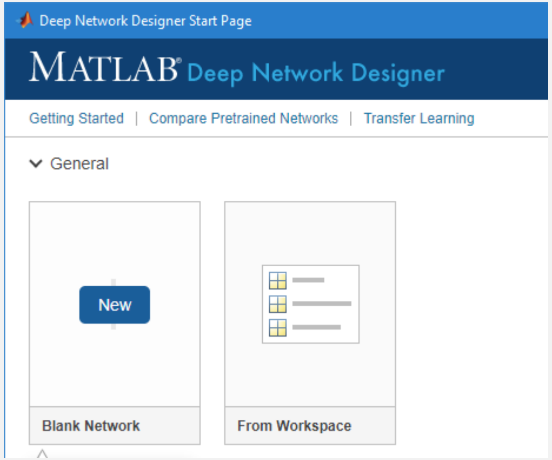
Display some images from the datastore

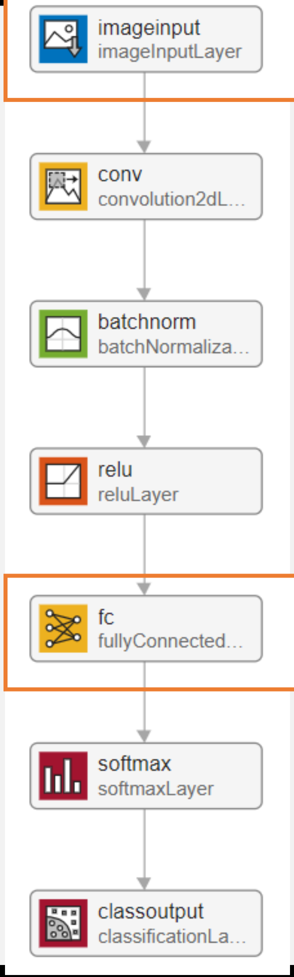
```
figure;
perm = randperm(976,20);
for i = 1:20
    subplot(4,5,i);
    imshow(imds.Files{perm(i)});
end
```

Open the Deep Network Designer

```
>> deepNetworkDesigner
```

When the designer app opens, select **Blank Network**





Set `InputSize` 227, 227, 3

Set `OutputSize` 9

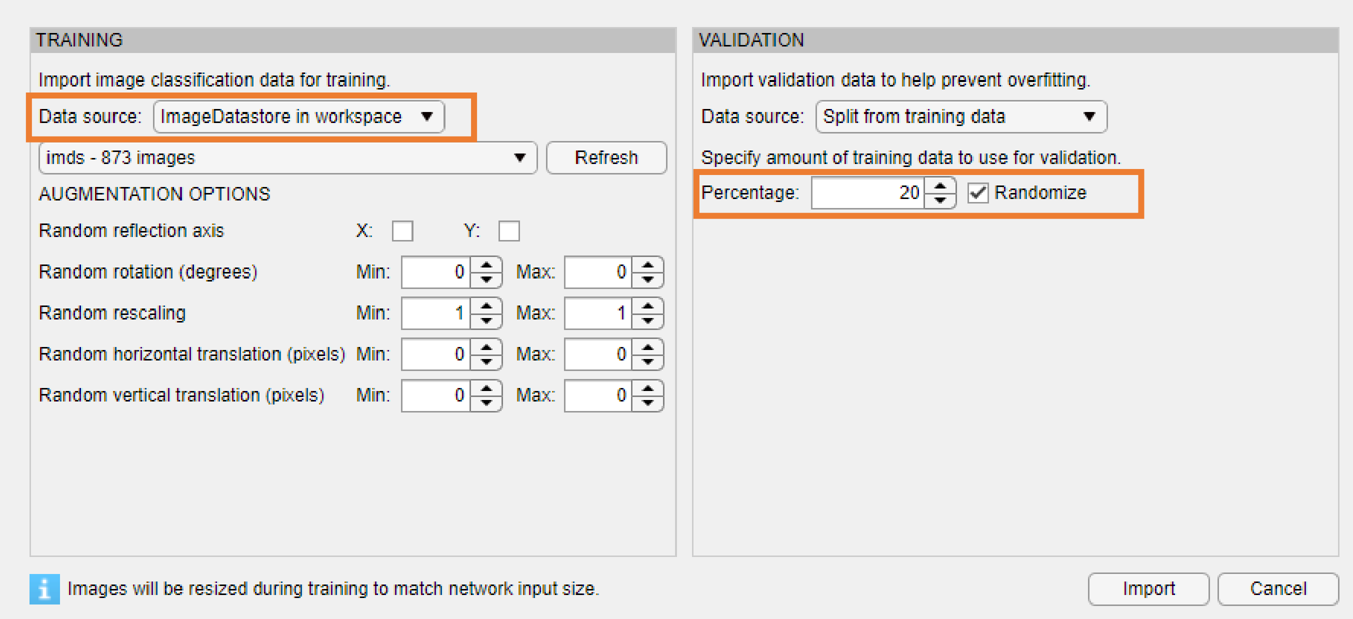
Create this network by dragging layers from the Layer Library panel

Connect them by dragging the output of a layer to the input of the next

Note: MATLAB resizes images to match the input size (227x227 pixels)

Import the training images to the designer

- Click on the **Data** tab
- Click on **Import Data** > **Import Image Data**



Set up the training parameters

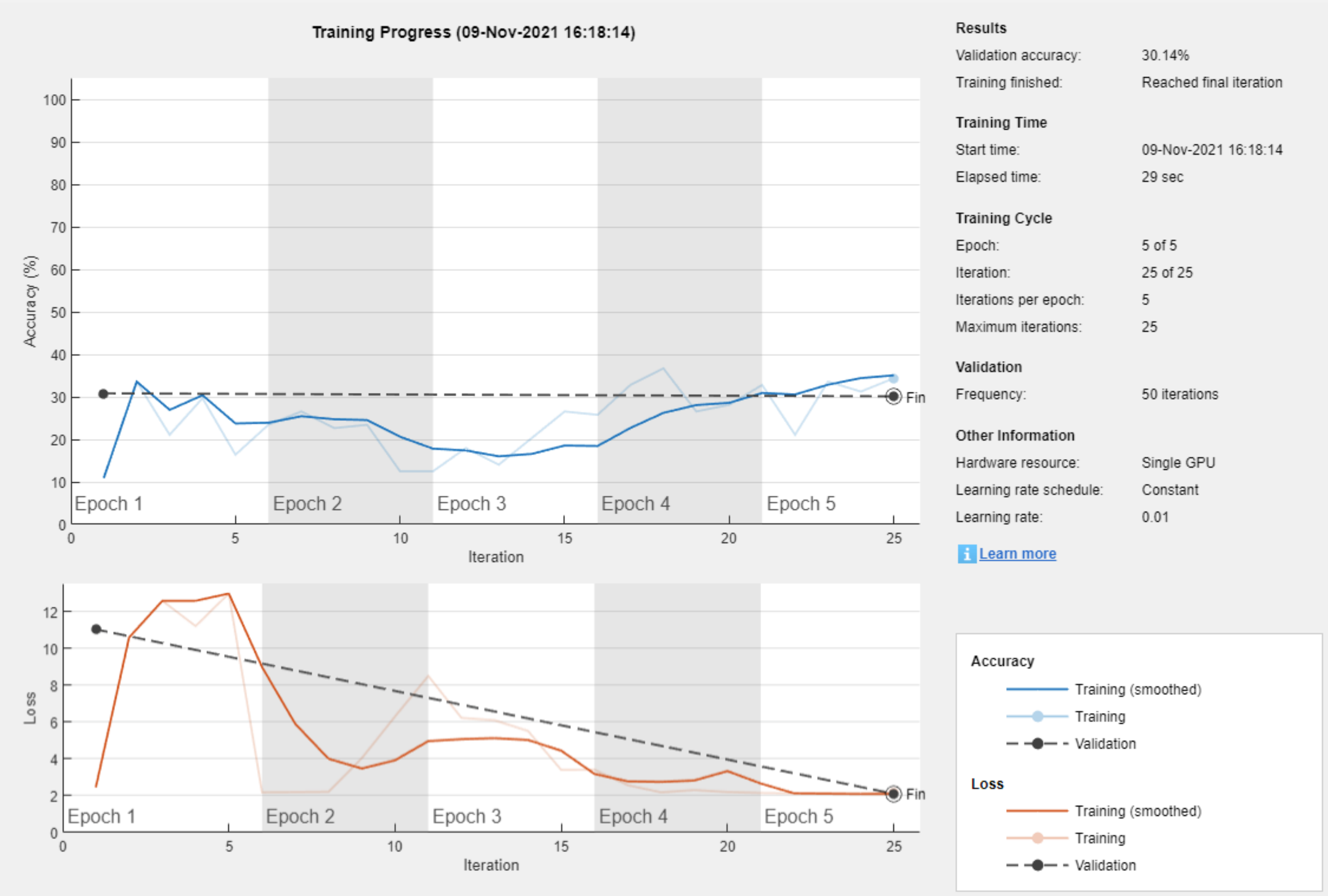
- Click on the **Training** tab, then click **Training Options**

- **Set** the following parameters:
  - InitialLearnRate = 0.001
  - MaxEpochs = 5;
  - MiniBatchSize = 16

- Click **Train**

- When training is complete, select **Export Trained Network and Results**

Testing the trained network


Training Progress (09-Nov-2021 16:18:14)
Accuracy (%)
Iteration
Loss
Iteration
Epoch 1
Epoch 2
Epoch 3
Epoch 4
Epoch 5
Fin
Results
Validation accuracy: 30.14%
Training finished: Reached final iteration
Training Time
Start time: 09-Nov-2021 16:18:14
Elapsed time: 29 sec
Training Cycle
Epoch: 5 of 5
Iteration: 25 of 25
Iterations per epoch: 5
Maximum iterations: 25
Validation
Frequency: 50 iterations
Other Information
Hardware resource: Single GPU
Learning rate schedule: Constant
Learning rate: 0.01
Learn more
Accuracy
Training (smoothed)
Training
Validation
Loss
Training (smoothed)
Training
Validation

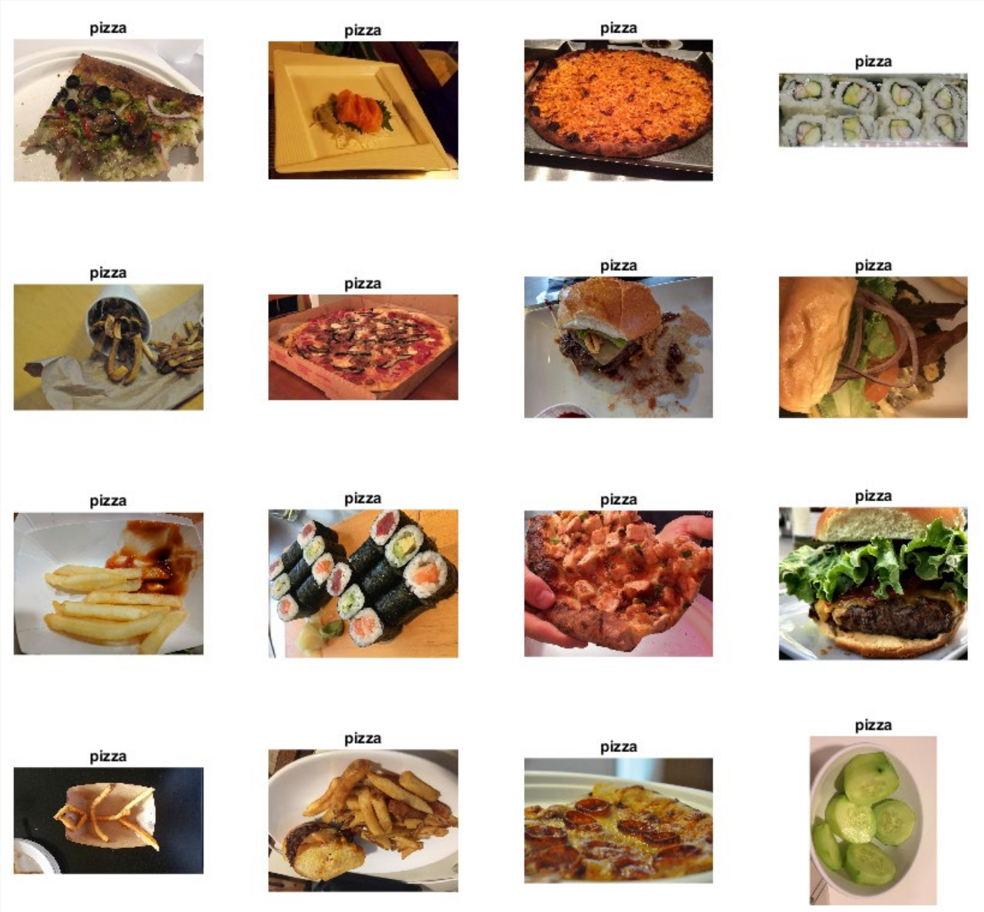
Classifying individual images

```
testImage = imread(imds.Files{1});
classify(trainedNetwork_1, testImage)
```

```
testImage = imread(imds.Files{540});
classify(trainedNetwork_1, testImage)
```

Randomly select and display images

```
%Select 16 random images
imgIdxs = randperm(978, 16);

figure;
for ii = 1:numel(imgIdxs)

    I = imread(imds.Files{imgIdxs(ii)});

    %Resize I to match the input layer size
    Ires = imresize(I, [227 227]);

    classification = classify(trainedNetwork_1, Ires);

    subplot(4, 4, ii);
    imshow(I)
    title(classification, 'Interpreter', 'none')

end
```


pizza
pizza
pizza
pizza
pizza
pizza
pizza
pizza
pizza
pizza
pizza
pizza
pizza
pizza
pizza
pizza

Saving the network

- To save the trained network, you can simply save the variable in the Workspace

```
>> save('savedNetwork.mat', ...
'trainedNetwork_1', 'trainInfoStruct_1')
```

Speeding up training with transfer learning

Transfer learning

uses a network that was previously trained and retrains it on a different dataset (as opposed to starting with random weights)

Load pretrained network

Early layers that learned basic features (edges, blobs...)

⋮

Last layers that learned more specific features (pizza, fries)

Replace final layers

⋮

New layers that will learn features in new data set

Train the network

Training images

Training options

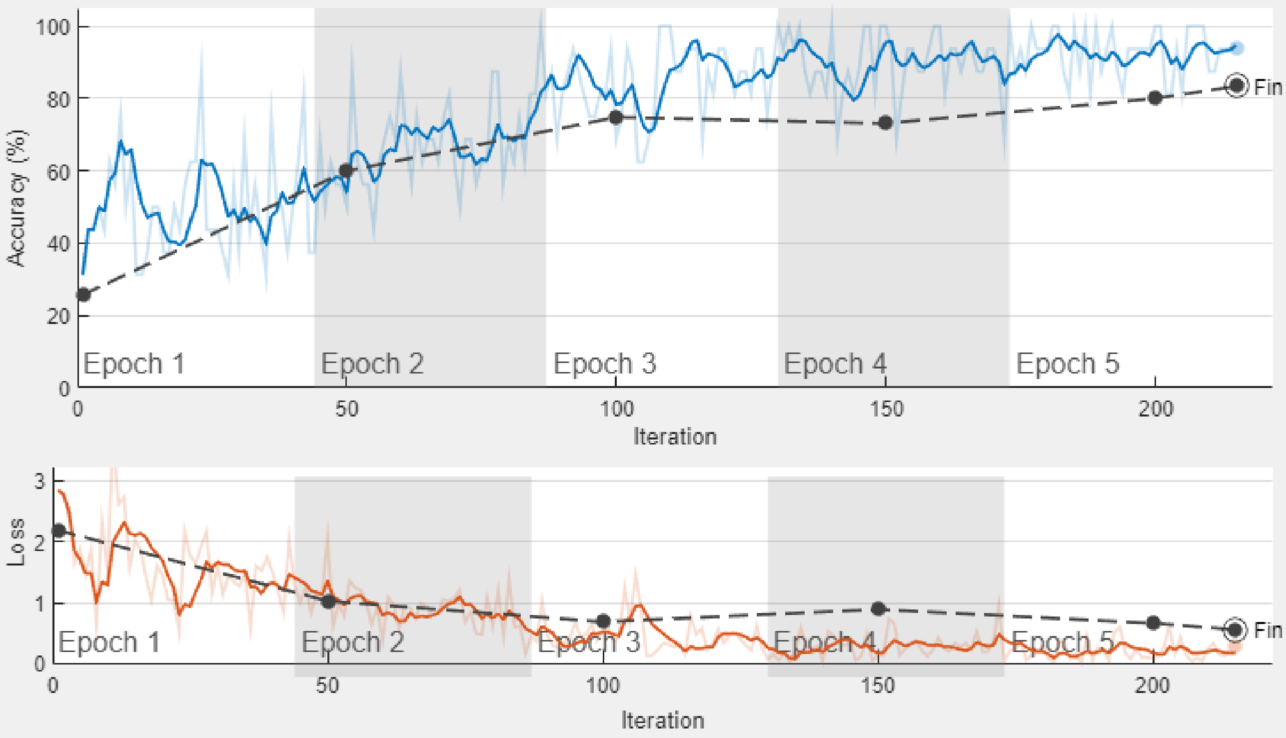
Load a pretrained network

- In the Deep Network Designer, click on **New**
- Install **AlexNet** (if using for the first time)
- Close the dialog box and click on New again to refresh the list
- Select AlexNet

Make the following modifications

- **Replace (delete and drag new copies)** the last three layers:
  - Fully Connected Layer,
  - Softmax
  - Classification layer
- Set the `OutputSize` of the fully connected layer to 9
- Train the network using the same options as before

Training Progress (09-Nov-2021 19:15:42)



Results

Validation accuracy:	83.43%
Training finished:	Reached final iteration

Training Time

Start time:	09-Nov-2021 19:15:42
Elapsed time:	55 sec

Training Cycle

Epoch:	5 of 5
Iteration:	215 of 215
Iterations per epoch:	43
Maximum iterations:	215

Validation

Frequency:	50 iterations

Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.001

Learn more

pizza

hamburger

pizza

hamburger

pizza

pizza

hamburger

pizza

pizza

french_fries

pizza

pizza

hamburger

french_fries

hamburger

pizza

For this lab (Part 1)

- Train a network to classify images of cells:
  - **BPAE (Bovine Pulmonary Arterial Endothelial) cells** – single mammalian cells
  - **Mouse kidney** - tissue
  - **E. coli** – single bacteria cells

- Write code to prepare your own training data from these images

- More details in handout