Image Classification using K-Nearest Neighbor and Convolutional Neural Networks

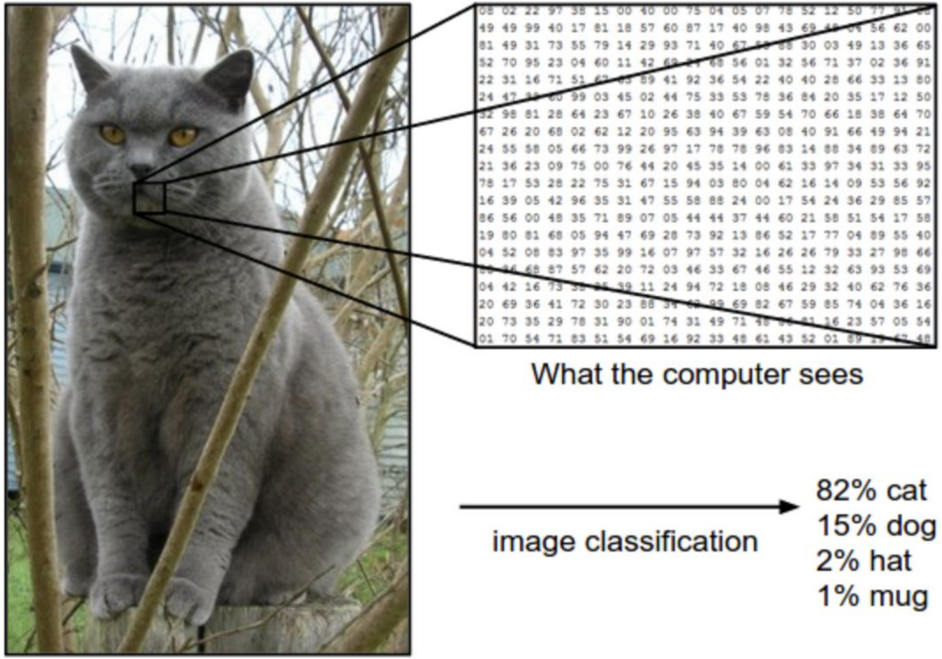
By Rumaisa Abdulhai

Why Image Classification?

Many Applications, including:

- Medical Diagnosis
  - Identify cancers & tumors using CT Scans
- Image Searching
  - Law Enforcement to identify suspects
- Autonomous vehicles such as self-driving cars
  - Obstacle avoidance
  - Navigation
- And much more...

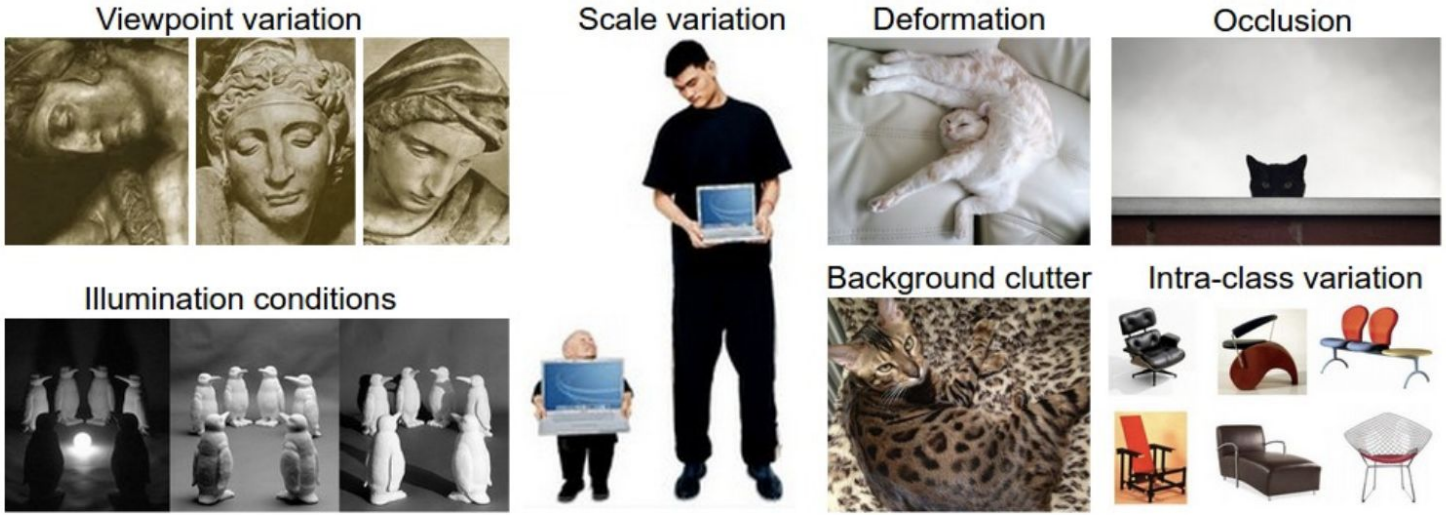
Image Classification Problem



- Turn the matrix of numbers (pixels) into a single label...

CS231n- http://cs231n.github.io/classification/

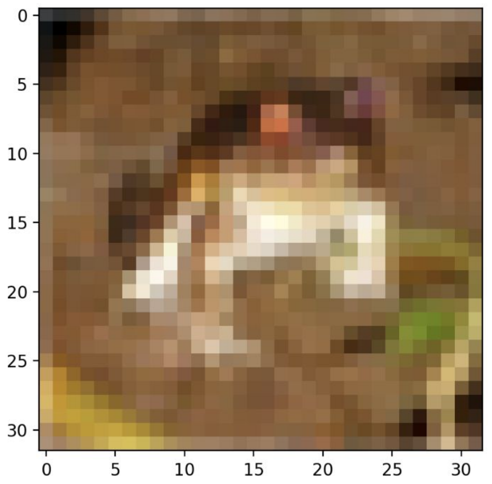
Strength of the Classification Model



CS231n- http://cs231n.github.io/classification/

CIFAR-10 Dataset

- 60,000 32x32 RGB images with labels
- 10 classes of images so 6000 images per class
- 1/6 images reserved for testing



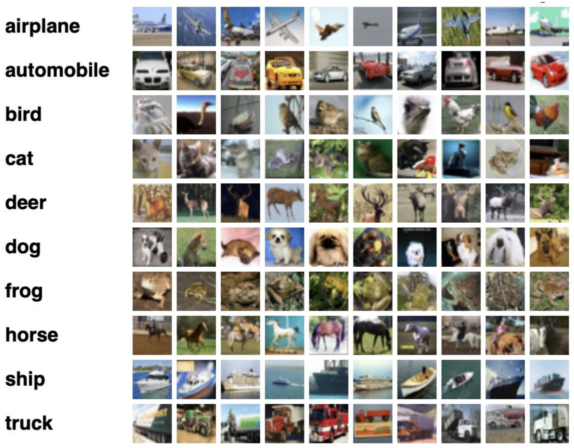


Image Net Dataset

- 2012 Challenge
  - 1.4 Million 256 x 256 RGB images with labels
  - 1000 classes of images so roughly 1000 images per class
  - 1.2 Million Training, 50,000 Validation, 150,000 Testing

These are popular datasets, but there are many more

Machine Learning Techniques for Image Classification

- K-Nearest Neighbor (KNN)
  - Simplest method
- Convolutional Neural Networks (CNN)
  - Dominant method

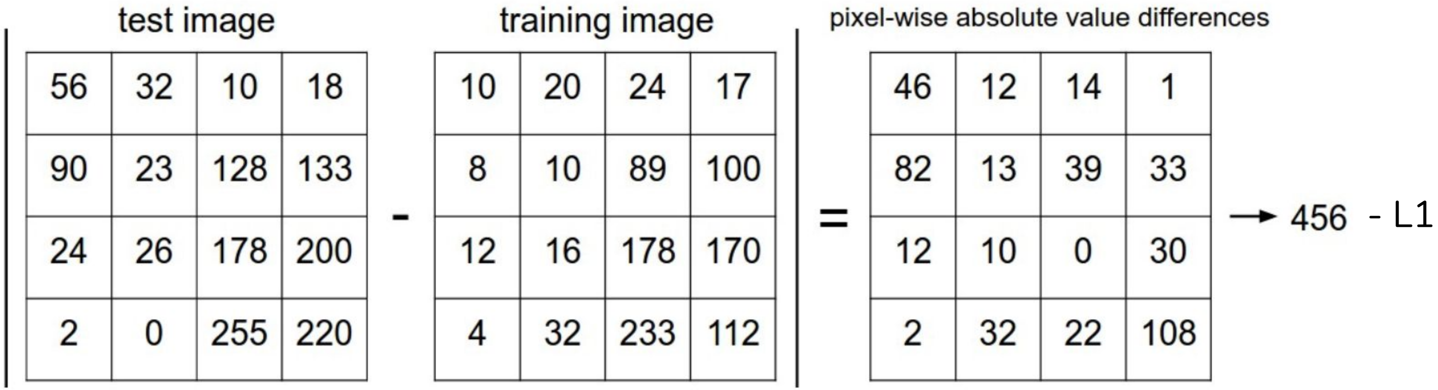
ML: Training, Validation, & Testing Sets

- Dataset divided into Training and Testing (~80% Training & 20% Testing)
- Within Training dataset, a small portion is reserved for validation
- Training set: Used to form the weights/params of the model
- Validation set: Used in between runs of training to further tune parameters
- Test set: Used at the very end after model is completed to evaluate performance

K-Nearest Neighbor (KNN)

L1 & L2 Distances- Definitions

- Calculates distances between two vectors (images)
  - Numerical measure of proximity between two images
- L1: sum of absolute value differences between corresponding pixel values
- L2: square root of the sum of the squares of the pixel-wise differences



CS231n- http://cs231n.github.io/classification/

Nearest Neighbor for Classifying Images

- Take the L2 distance between every training image and desired test image
- The label of the training image with the smallest L2 dist with the test image is assigned to test image
- Repeat for all desired test images...

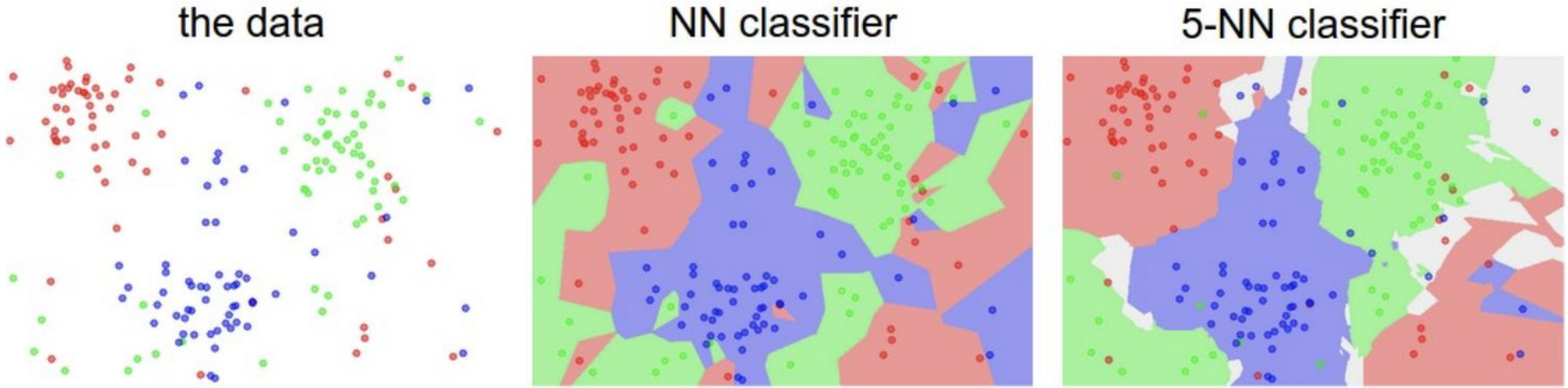
In [14]:
```
from sklearn.metrics import accuracy_score
# making predictions
predictions = k_nearest_neighbor(x_train, y_train, x_test, 7)

# evaluating accuracy
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: {}".format(100*accuracy))
```

```
Accuracy: 27.400000000000002
```

K Nearest Neighbor

- Find K Smallest L2 Distances and take most common label
- As K increases, more bias and less variance (smooth boundaries)
- K > 1 always better than k = 1



CS231n- http://cs231n.github.io/classification/

K-NN: Accuracies with Different Training Data

K	Accuracy
1	31.97
3	31.21
5	32.43
7	32.01
9	32.42

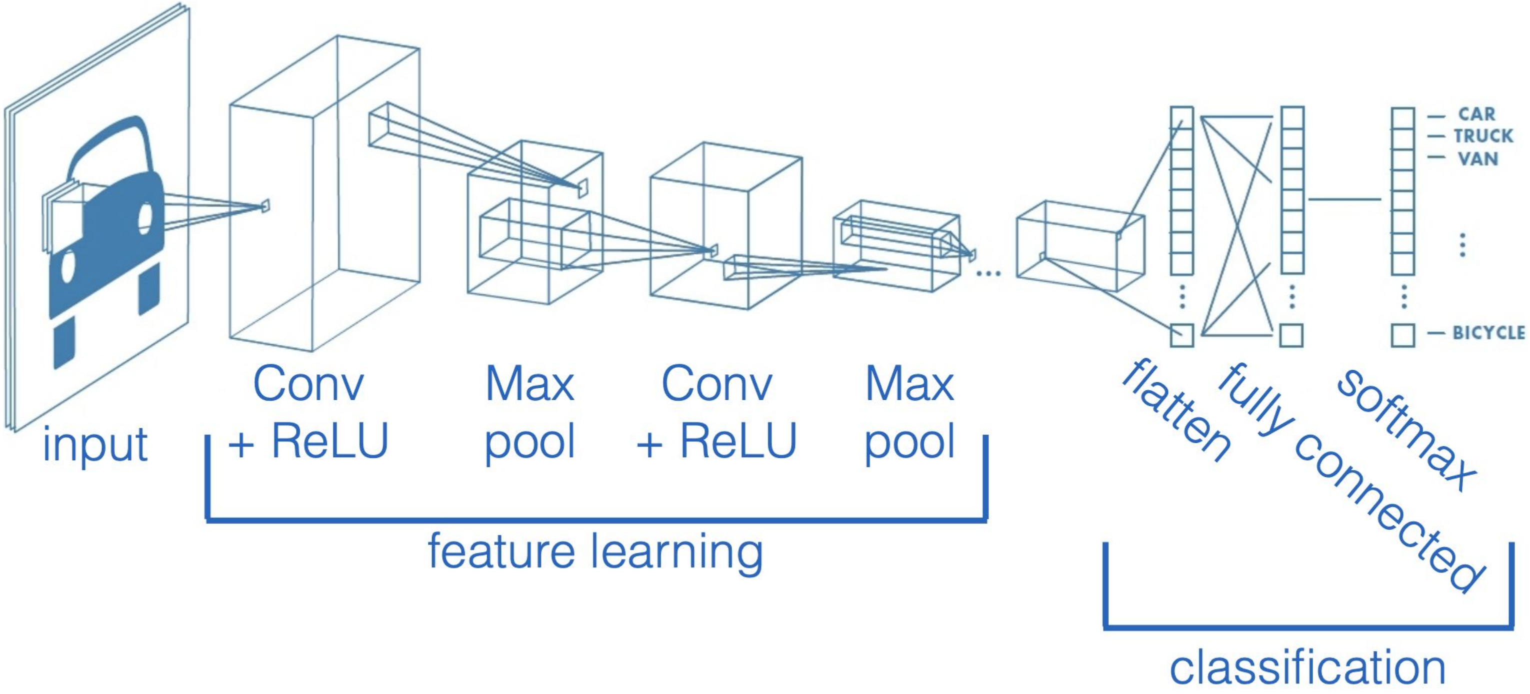
25,000 Training
10,000 Test

K	Accuracy
1	35.39
3	33.02
5	33.98
7	33.58
9	33.98

50,000 Training
10,000 Test

Convolutional Neural Networks (CNNs)

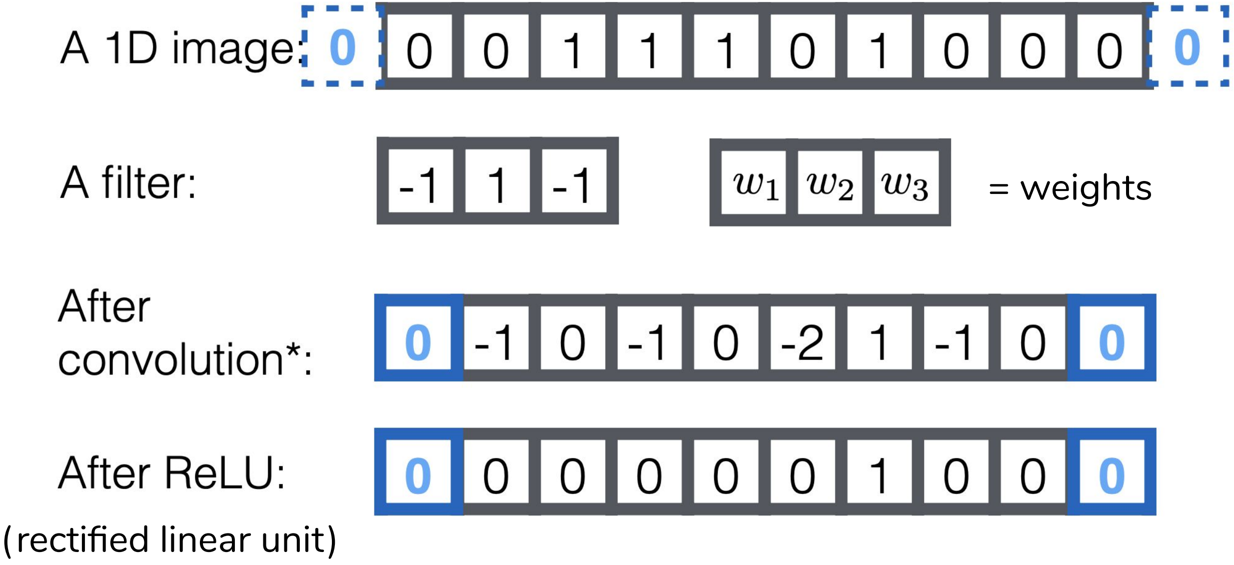
Components of a CNN Model



Courtesy Tamara Broderick, MIT

What is a Convolution Layer?

- Convolution is the scalar/dot product of two vectors/matrices
- Used to detect patterns in target images where pattern is specified by filter
- Example:



ReLU = activation function where pos #s are kept & neg #s =0

Courtesy Tamara Broderick, MIT

2D Convolution Example (B&W Image)

A 2D image:

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

A filter:

-1	-1	-1
-1	1	-1
-1	-1	-1

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

After convolution:

0	-4	0	-3	-1
-2	-7	-2	-4	1
-2	-5	-2	-5	-2
-2	-7	-2	-5	0
0	-4	0	-4	0

After convolution & ReLU:

0	0	0	0	0
0	0	0	0	1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Courtesy Tamara Broderick, MIT

What is a Max Pooling Layer?

- Max Pooling is the reduction in the size of an image (matrix) by taking maximum pixel values from certain areas of the image
- Goal: Summarize the patterns it discovered during convolution + relu layer
- Use a kernel for specifying the part of the image you take the maximum of
- Stride specifies how your kernel slides over the image, can be from 1 to length of the kernel
  - If stride = kernel length, then image size reduces by the factor of the kernel's length

2D Max Pooling Example

Output from the convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

6x6 B&W Image

Max pooling: returns max of its arguments

- size 3x3 (“size 3”)
- stride 3

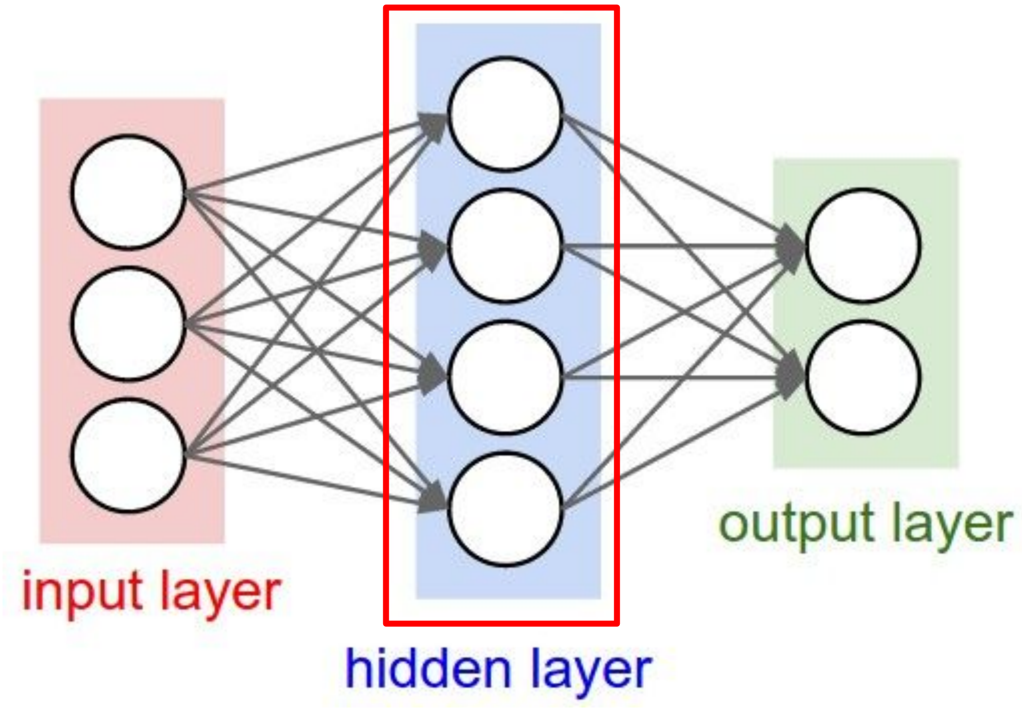
After max pooling:

0	1
1	0

2x2 B&W Image

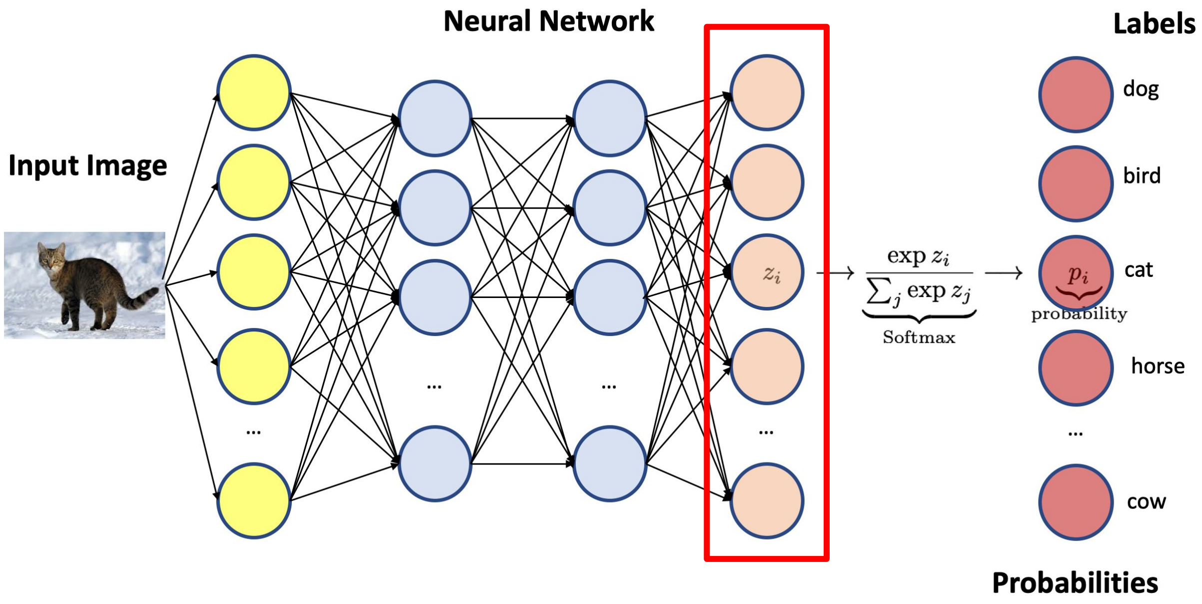
Courtesy Tamara Broderick, MIT

What is a Fully-Connected Layer?



- Performs the classification part
- Ultimately needed for ML
- Known as Hidden Layer
- Where all inputs are connected to each node at the next level with a seperate weight.
- Images must be flattened or converted to 1D array before passed into hidden layer

What is a Softmax Layer?



- An activation function for multicategory classification
- Converts the output of the final fully-connected layer into probabilities of the test image falling into a certain category
- These probabilities are compared against one-hot-encoded actual label

Preliminary Results on CIFAR-10 with CNN Models

	Layers	Parameters	% Test Error with 5 Epochs	% Test Accuracy with 5 Epochs	% Test Error with 10 Epochs	% Test Accuracy with 10 Epochs
Model 1	1 conv, 1 maxpool, 1 hidden, 1 softmax	806,666	37.42	62.58	34.41	65.59
Model 2	2 conv, 1 maxpool, 1 hidden, 1 softmax	619,306	34.58	**65.42**	31.39	68.61
Model 3	3 conv, 1 maxpool, 1 hidden, 1 softmax	464,714	35.59	64.41	30.86	**69.14**
Model 4	4 conv, 1 maxpool, 1 hidden, 1 softmax	342,890	40.12	59.88	32.45	67.55

Future Work

- Fine tune the model further to achieve a higher accuracy
  - Save best weights throughout training that minimize classification error between training and test sets
  - Train for more epochs
- Use a different, larger dataset such as ImageNet and develop an accurate model if time permits

Extras

Model 1- Convolutional Layer

```
model.add(Conv2D(32, (5, 5), activation='relu', input_shape=(32,32,3))
```

Input Image- 32x32x3

For every input image, make 32 copies of image and apply a different filter each of size 5x5 for each copy of the image

Output Image: 28x28x32

Total weights/parameters needed: 5x5 x 32 x 3 + 32 = <u>2432</u>

No padding is used, so images are reduced by 2 pixels on each side

Model 1- Max Pooling Layer

```
model.add(MaxPooling2D(pool_size=(2, 2)), data_format='channels_last')
```

Input Image: 28x28x32

Apply Max Pooling with a size 2x2 kernel with stride 2, which divides the row and column of each image in half

Result Image: 14x14x32

Model 1: Hidden Layer

```
model.add(Flatten())

model.add(Dense(128, activation='relu'))
```

Flatten Image First: 14x14x32 = 6272 pixels

Hidden Layer has 6272 inputs, and 128 outputs

Total weights/parameters required: 6272 x 128 + 128 = 802,944

Model 1: Final Softmax Layer

```
model.add(Dense(num_classes, activation='softmax'))
```

Now Hidden Layer to Softmax

Softmax has 128 inputs, and 10 outputs (for the 10 classes)

Total weights/parameters needed: 128 * 10 + 10 = 1290

Total Tunable Parameters: 6272 + 802,944 + 1290 = **806,666**