Introduction to Deep Learning

Yogesh Kulkarni

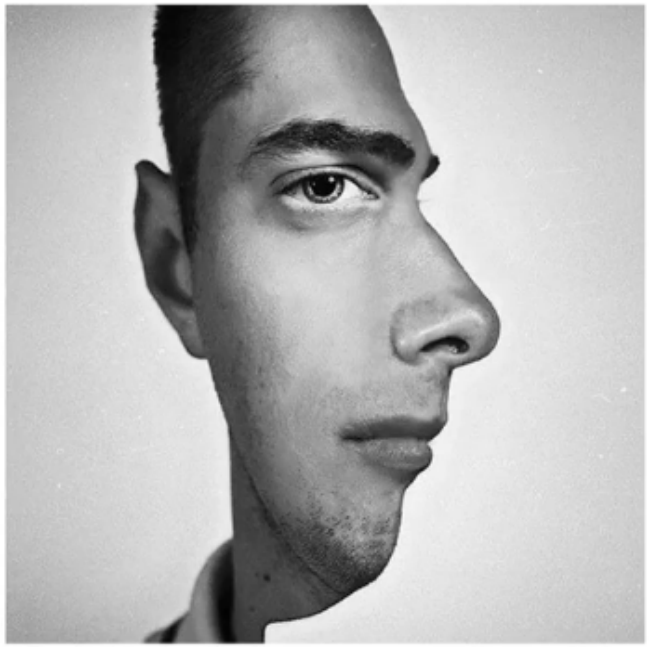
September 19, 2020

Convolutional Neural Network (CNN)

Introduction

- Are images easier to work with? with respect to Machine Learning?
- But, Data is all numeric, is fixed size . . . , still?
- If the image is of a cat, it is of a cat.
- If the image is of a dog, it is of a dog.
- There isn't any ambiguity, right? like languages.
- Then why can't we confidently use it (yet) for health-care images?

Images can be ambiguous too



(Ref: Deep Learning A-Z - Kirill Eremenko)

- Do you see a person looking at you or to the right?
- Brain struggles.
- if you look at right edge, something. If left, something else.
- Brain is trying to detect based on Features.

Images can be ambiguous too, even for humans

(Ref: Deep Learning A-Z - Kirill Eremenko)

- Is it a young girl looking away or an old lady looking downwards.
- Just one or two features are not sufficient at times, need more clues ie more differentiable features.

Human Brain is not a perfect Machine

(Ref: Deep Learning A-Z - Kirill Eremenko)

- Brain can not detect at times.
- We sometimes get confused, Eisher's diagrams, anyone?

Theory of CNN

NN types: ANN, CNN, RNN

- In ANN, rows are independent of each other. None of the neighbor information is used.
- In CNN, spatial (not 'special') neighbors are used to form features, which help in discerning the categories in the classification.
- In RNN, temporal (no, 'temporary') neighbors are used to form features, which help in predictions.

What is CNN?

- Inspired by mammalian visual cortex.
- The visual cortex contains a complex arrangement of cells
- Sensitive to small sub-regions of the visual field,
- Cells act as local filters over the input space.

What is CNN?

Different filter/cell types

- Simple cells respond maximally to specific edge-like patterns.
- Complex cells have larger receptive fields and are locally invariant to position.

Applications of CNN

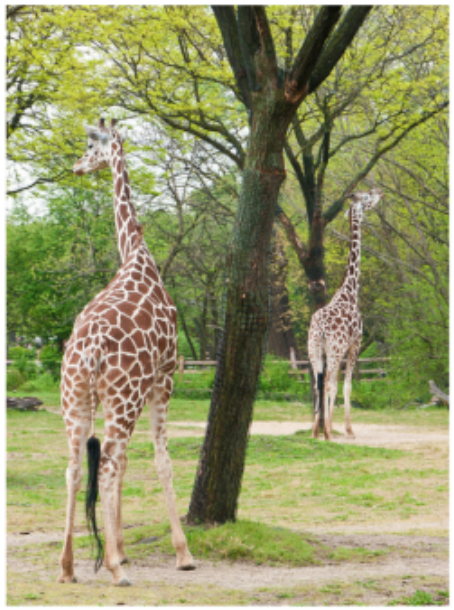
- Very effective in areas such as image recognition and classification.
- Identifying faces, objects and traffic signs apart from powering vision in robots and self driving cars.

Applications of CNN

Even in mixed-modes, ie Image and Language, together.

a soccer player is kicking a soccer ball

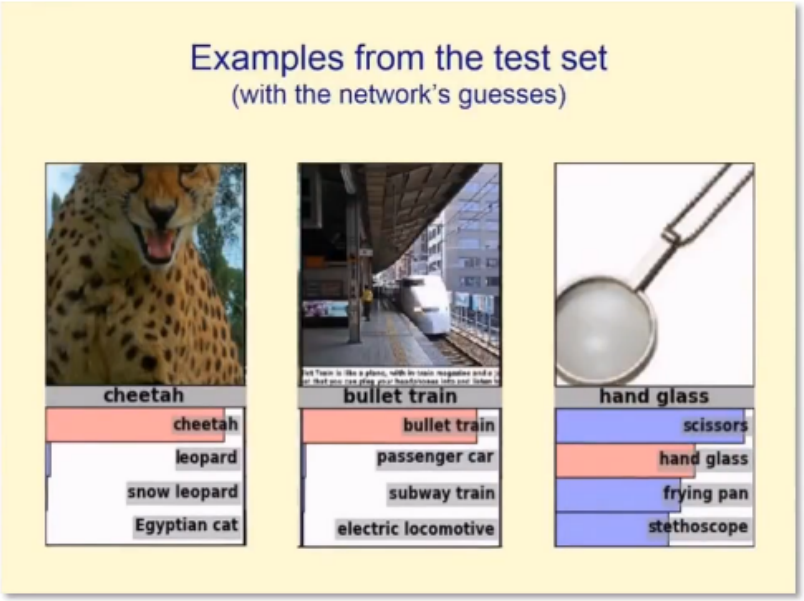
a street sign on a pole in front of a building


a couple of giraffe standing next to each other

- ConvNet is able to recognize scenes and the system is able to suggest relevant captions ("a soccer player is kicking a soccer ball")
- Also, being used for recognizing everyday objects, humans and animals.

History of CNN

History



(Ref: Deep Learning A-Z - Kirill Eremenko)

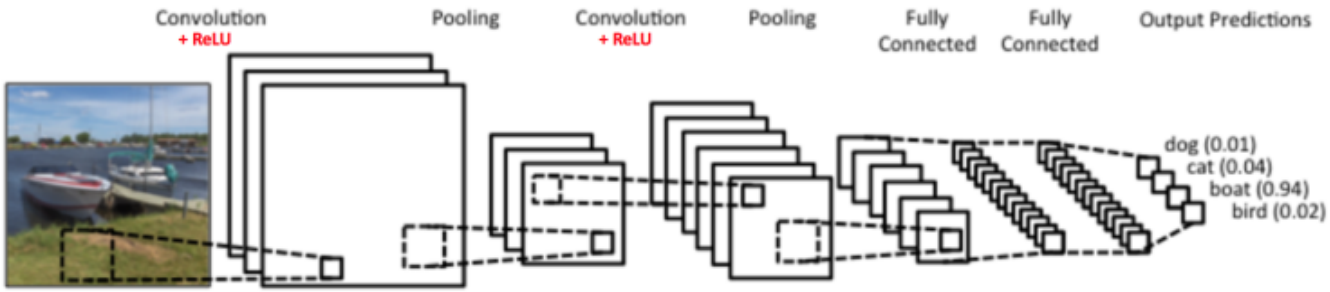
- Geoffrey Hinton, Yann Lecun are one of the pioneers of image related Neural Networks, Convolution Neural Networks (CNN), etc.
- Their results were promising but confusing at times, then.

History

- LeNet Architecture (1990s): Pioneering work by Yann LeCun was named LeNet5 after many previous successful iterations since the year 1988
- Used mainly for character recognition tasks such as reading zip codes, digits, etc.

History

Newer architectures are improvements over the LeNet, but they all use the main concepts from the LeNet and are relatively easier to understand.



On receiving a boat image as input, the network correctly assigns the highest probability for boat (0.94) among all four categories.

Other ConvNet Architectures

Convolutional Neural Networks have been around since early 1990s (LeNet).

- 1990s to 2012: In incubation
- AlexNet (2012): Deeper and much wider version of the LeNet and won by a large margin the difficult ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012
- ZF Net (2013): Improvement on AlexNet by tweaking the architecture hyperparameters. The ILSVRC 2013 winner.
- GoogLeNet (2014): The ILSVRC 2014 winner. Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).

Other ConvNet Architectures

- VGGNet (2014) : The runner-up in ILSVRC 2014. Showed, the depth of the network (number of layers) is a critical component for good performance.
- ResNets (2015): winner of ILSVRC 2015. Are the default choice for using ConvNets in practice.
- DenseNet (August 2016): Significant improvements over previous state-of-the-art architectures on five highly competitive object recognition benchmark tasks.

Concepts of CNN

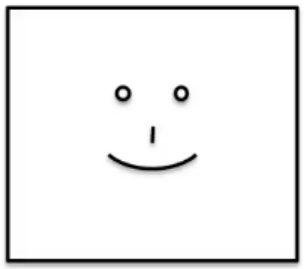
Terms

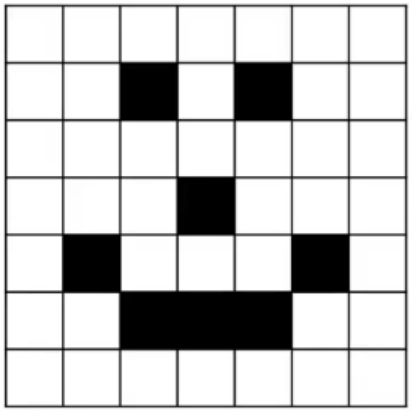
There are four main operations in the ConvNet

- Convolution
- Non Linearity (ReLU)
- Pooling or Sub Sampling
- Classification (Fully Connected Layer)

What is an Image?

Simplification: Assuming only 0 and 1 (black or white) are the values, here is the representation of a smiley image:

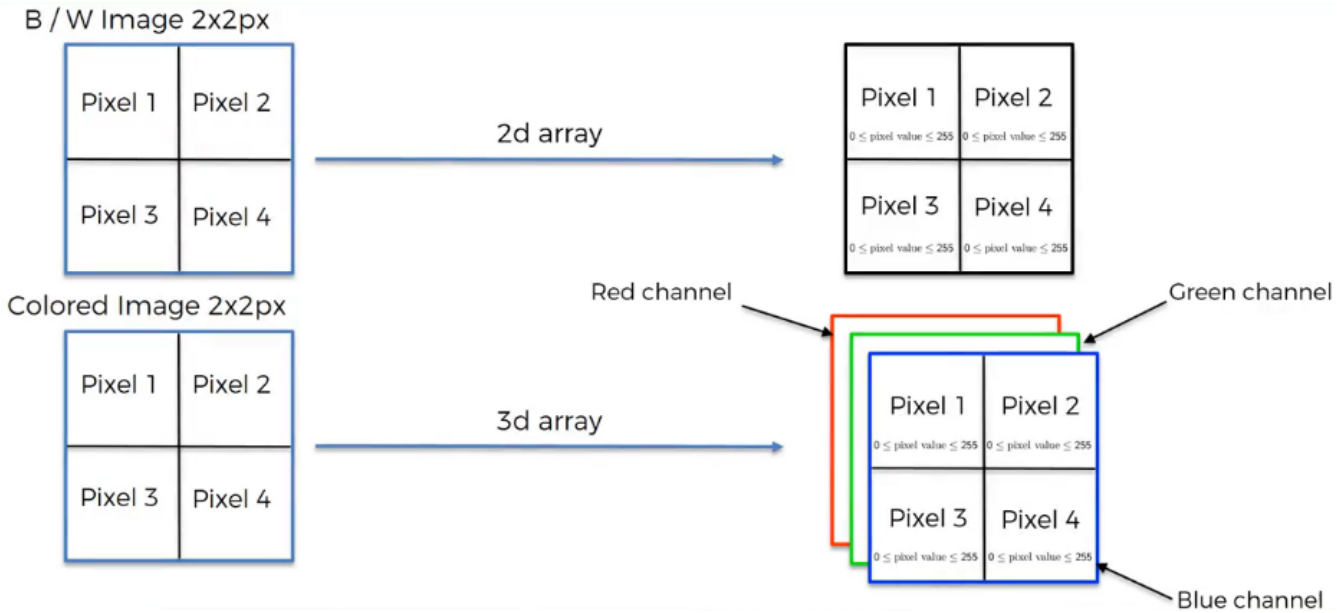




0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

(Ref: Deep Learning A-Z - Kirill Eremenko)

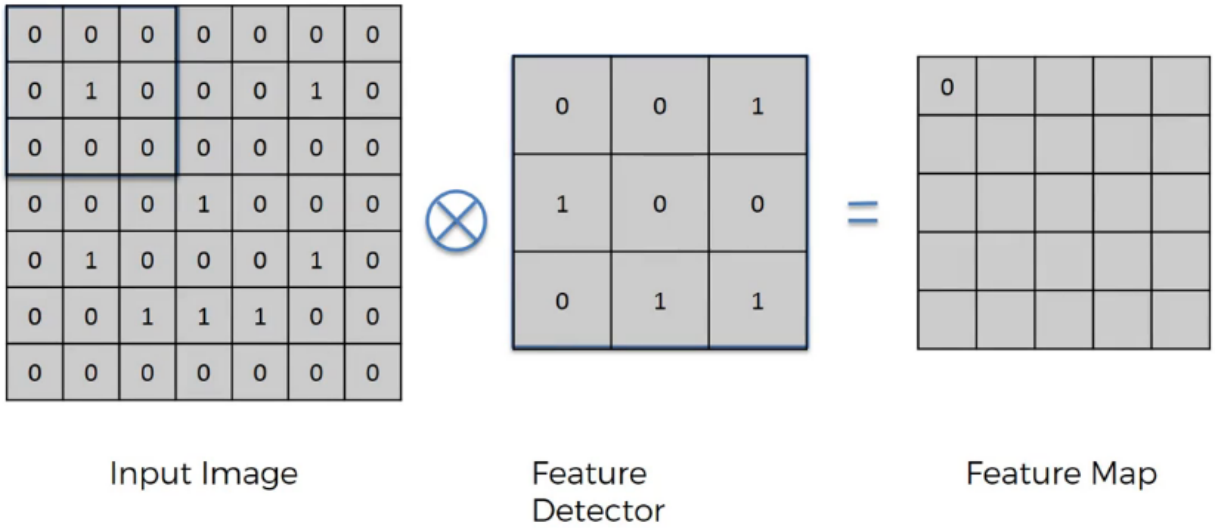
What is an Image?



(Ref: Deep Learning A-Z - Kirill Eremenko)

- For Grey scale image, each pixel has value from 0 (black) to 255 (white), 2^8 as its 8 bit.
- For Color image, there are 3 such matrices, one each for Red, Green and Blue.

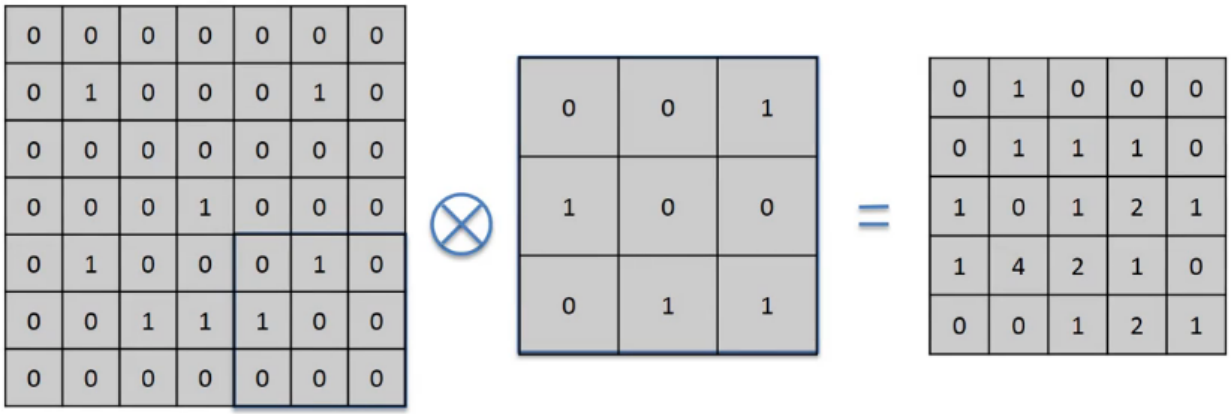
Step 1 : Convolution



(Ref: Deep Learning A-Z - Kirill Eremenko)

- Feature detector, same as filter, or convolution operator.
- Traverse the Input Image using Feature Detector, doing dot product at each position, then storing result in the "feature map".
- The amount of movement is called "stride".

Step 1 : Convolution

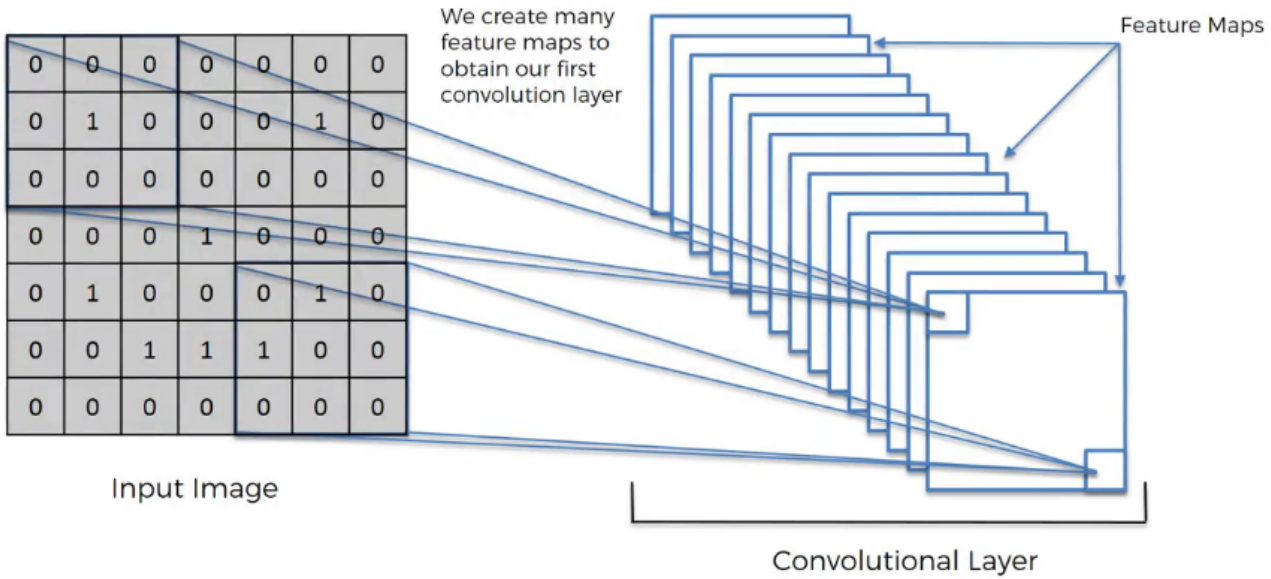


(Ref: Deep Learning A-Z - Kirill Eremenko)

What have we done?:

- Reduced the image size. More with bigger stride.
- Finally we want to go up to few enum values, for classification.
- While doing this dimension reduction, we wish to store the aggregated information.
- Some info is lost, but the relevant info (around features) is kept or even enhanced ("4" in the output).

Step 1 : Convolution



(Ref: Deep Learning A-Z - Kirill Eremenko)

- We as humans do not look at each pixel, but at strong features.
- We use multiple feature maps to accentuate multiple features.

Filter for Sharpen

Experimentation: Using GIMP to apply filters on an image of Taj Mahal

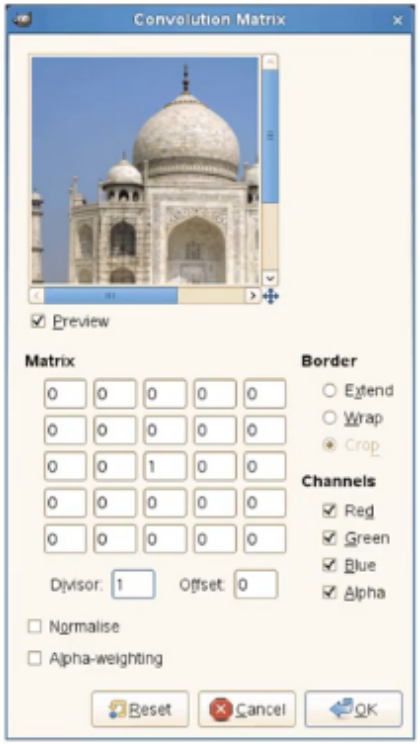


Image Source: docs.gimp.org/en/plug-in-convmatrix.html

(Ref: Deep Learning A-Z - Kirill Eremenko)

Filter for Sharpen

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

(Ref: Deep Learning A-Z - Kirill Eremenko)

"5" is in the middle of the filter and surroundings are "-1". It enhances center and reduces neighbors. Thus sharpens. Imp to note that it does not shuffle the pixels, so, preserves the pattern.

Filter for Blurr

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

(Ref: Deep Learning A-Z - Kirill Eremenko)

Filter for Edge Detect

	0	1	0	
	1	-4	1	
	0	1	0	

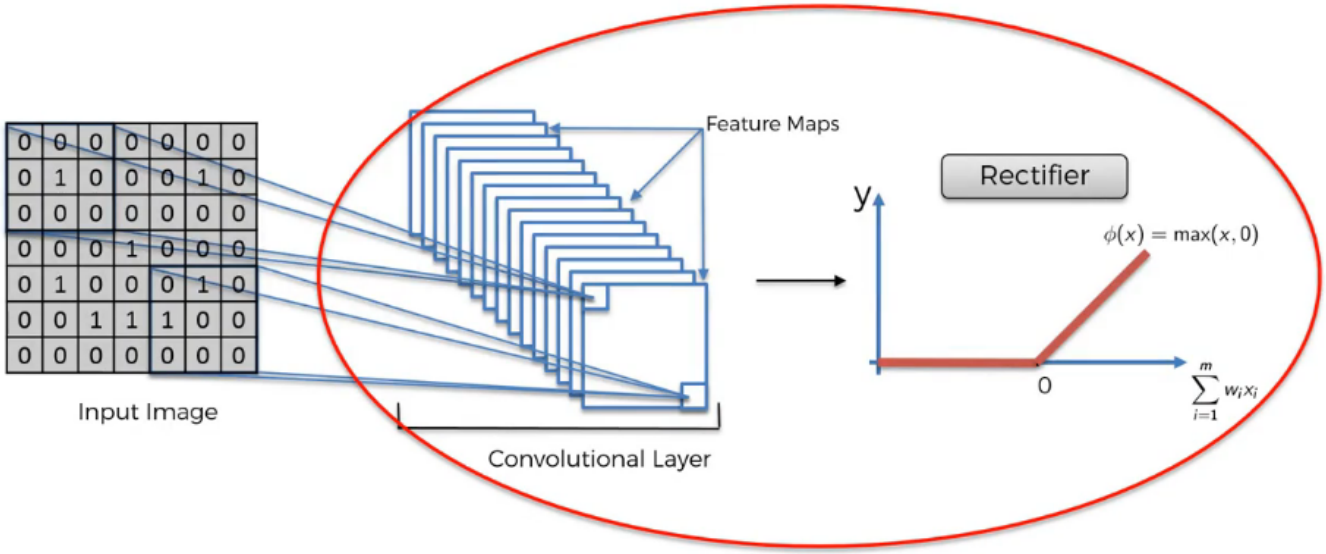
(Ref: Deep Learning A-Z - Kirill Eremenko)

Filter for Emboss

	-2	-1	0	
	-1	1	1	
	0	1	2	

(Ref: Deep Learning A-Z - Kirill Eremenko)

Step 1B : Relu



(Ref: Deep Learning A-Z - Kirill Eremenko)

Increases non linearity (as image itself are not linearly progressing).

Step 1B : Relu

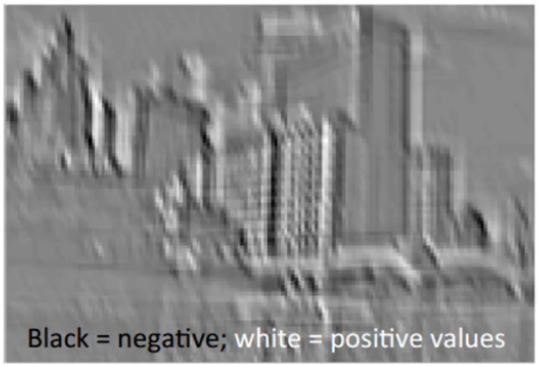
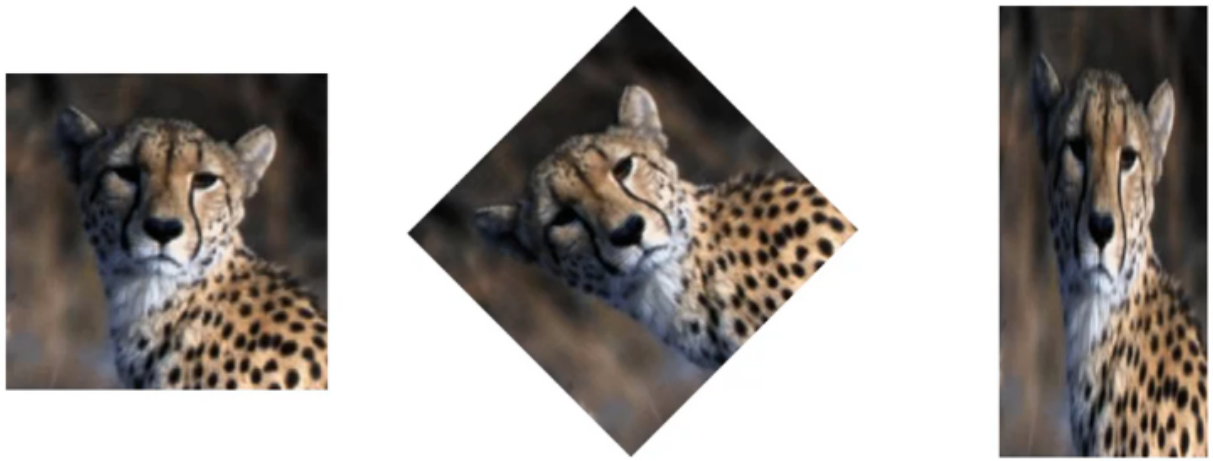


Image Source: http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

- Just using Feature Detector gives above output.
- ReLu makes it non negative, as shown below. Introduces non-linearity.

Image Source: http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

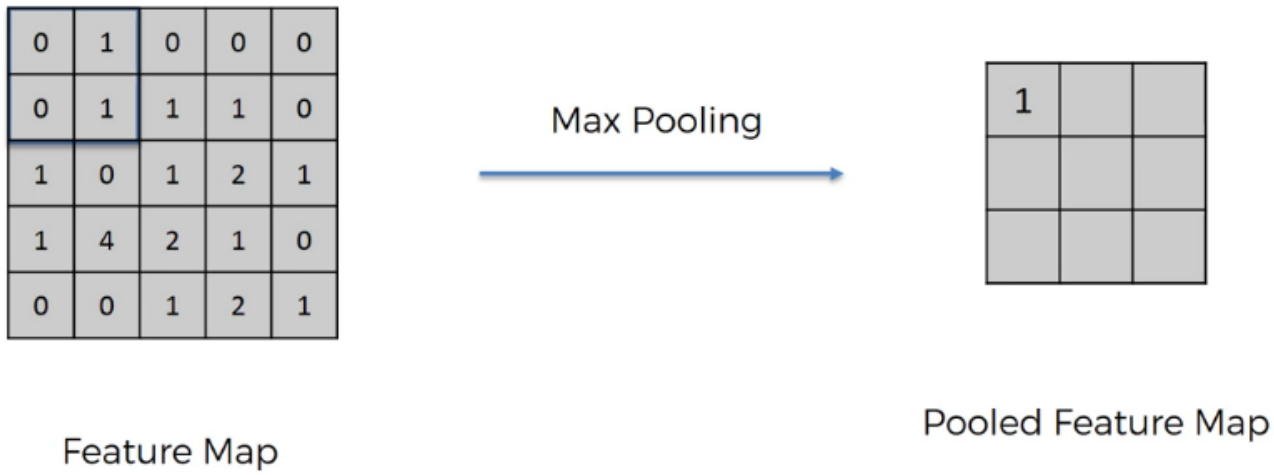
Step 2 : Max Pooling



(Ref: Deep Learning A-Z - Kirill Eremenko)

- Images can be transformed: scaled, rotated.
- But you need to identify each as the same.
- So, if you take MAX values, it does not matter its location in the image.

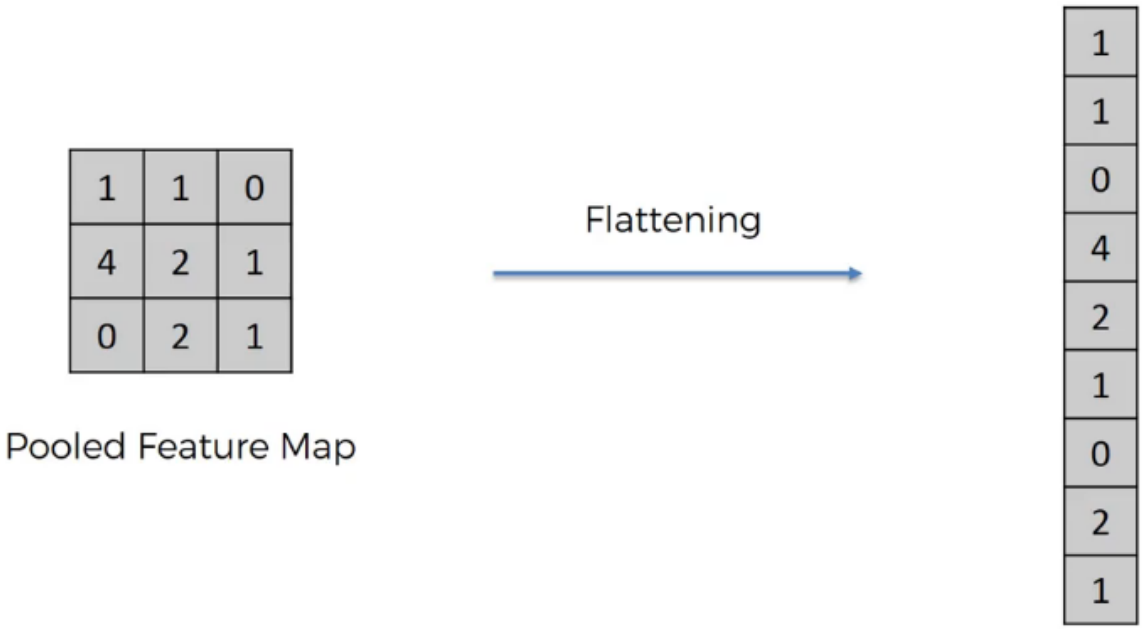
Step 2 : Max Pooling



(Ref: Deep Learning A-Z - Kirill Eremenko)

- A smaller window (not filter), use MAX of the window and dis regard other 3.
- No DOT product here.
- Good reduction in dimension.
- There are other types of pooling to "reduce', like average pooling.

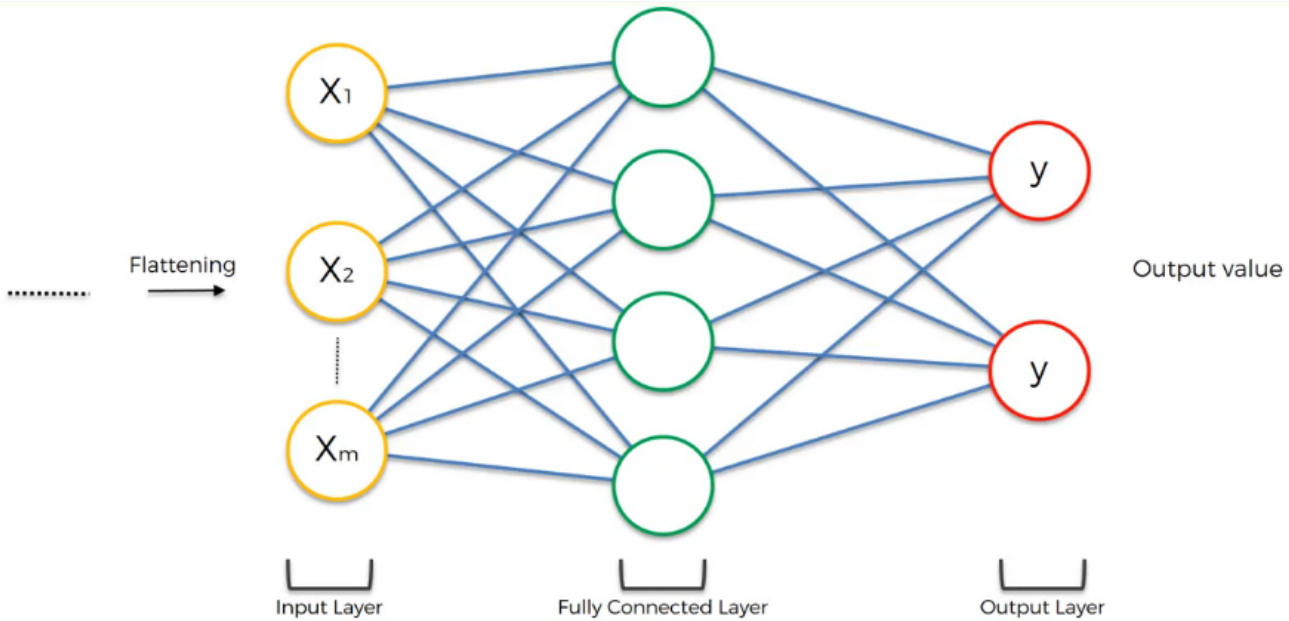
Step 3 : Flattening



(Ref: Deep Learning A-Z - Kirill Eremenko)

- Just making a vector out of matrix, so that it becomes in put to the new layer.
- Each filter will give its own vector, all such vectors can be clubbed together in a single column vector.

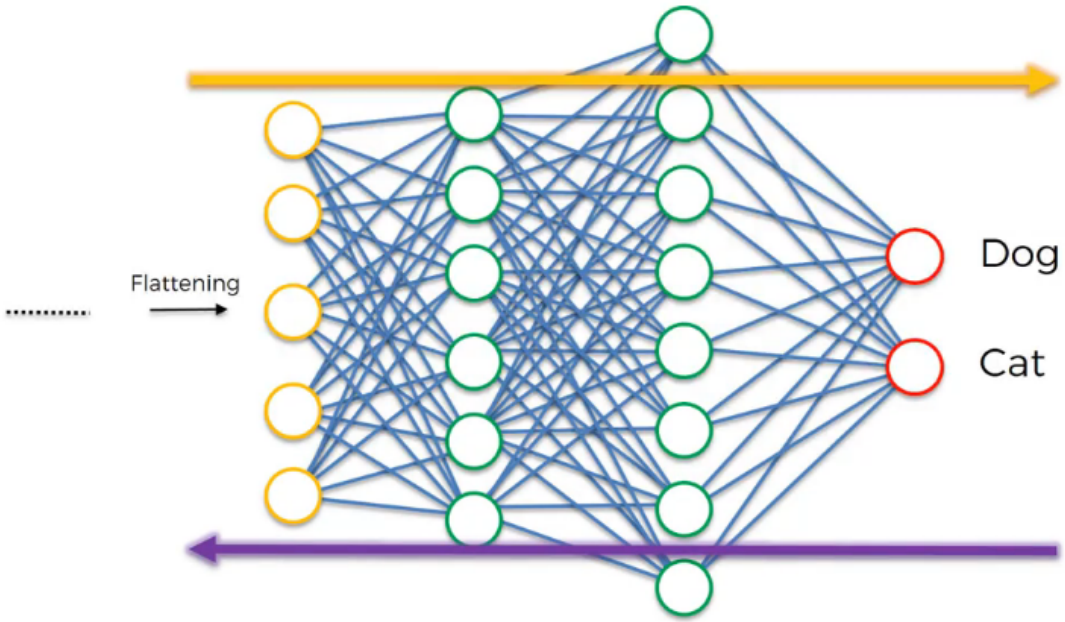
Step 4 : Full Connection



(Ref: Deep Learning A-Z - Kirill Eremenko)

- Adding NN after inputs are ready from previous step
- Fully connected network for classification

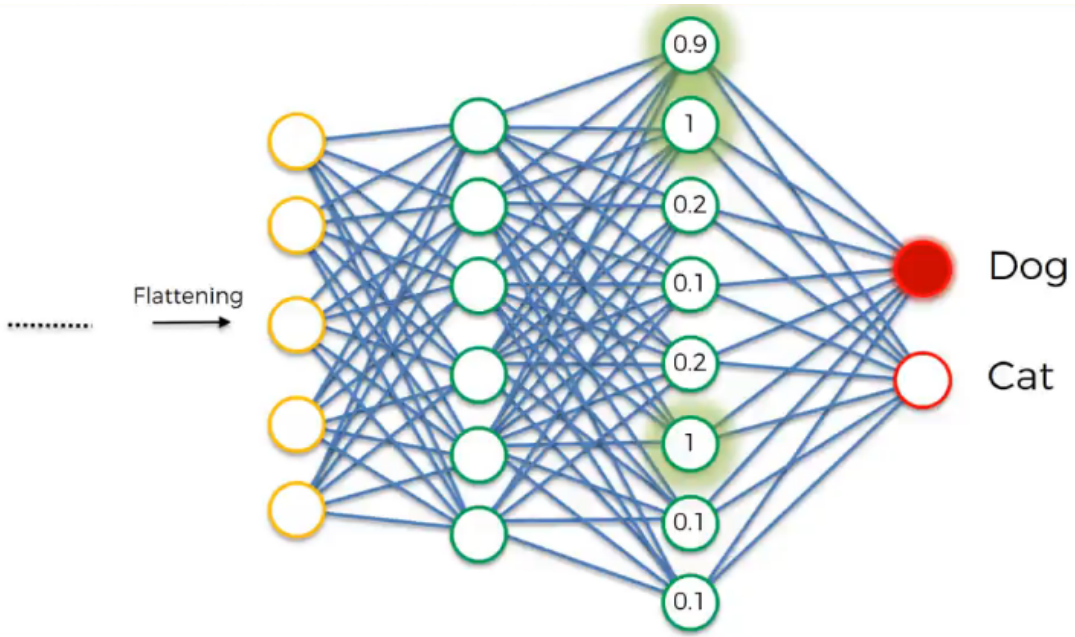
Step 4 : Full Connection



(Ref: Deep Learning A-Z - Kirill Eremenko)

- For binary output, single output is ok, but for multi class , those many output nodes are needed.
- In back propagation both, NN weights as well as Feature detector values are adjusted.

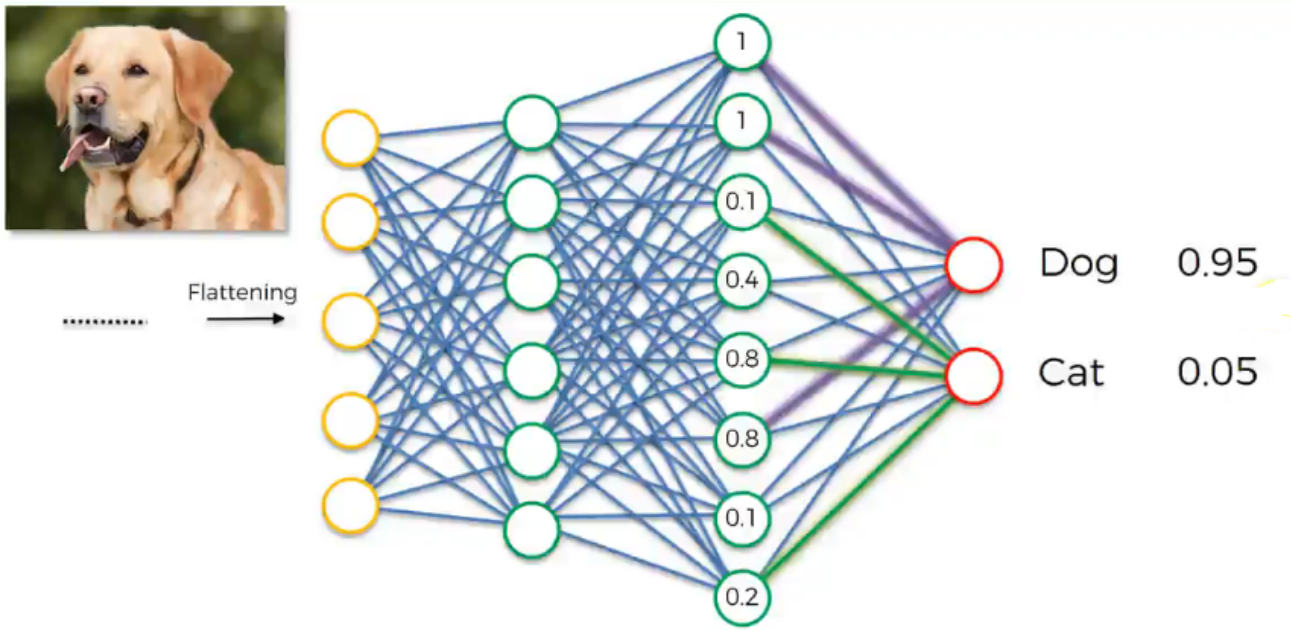
Step 4 : Full Connection



(Ref: Deep Learning A-Z - Kirill Eremenko)

- At the last hidden layer, we have float values.
- All the values are common to both nodes.
- The blue line weights, during training, get adjusted so that probability of specific output is adjusted properly.

Step 4 : Full Connection

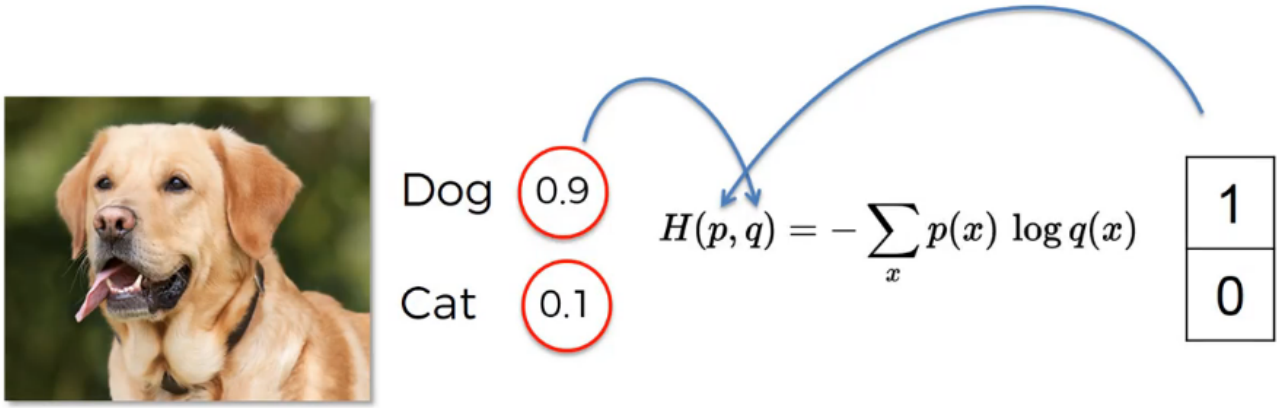


(Ref: Deep Learning A-Z - Kirill Eremenko)

Trained model can then be used for predictions.

Loss Function

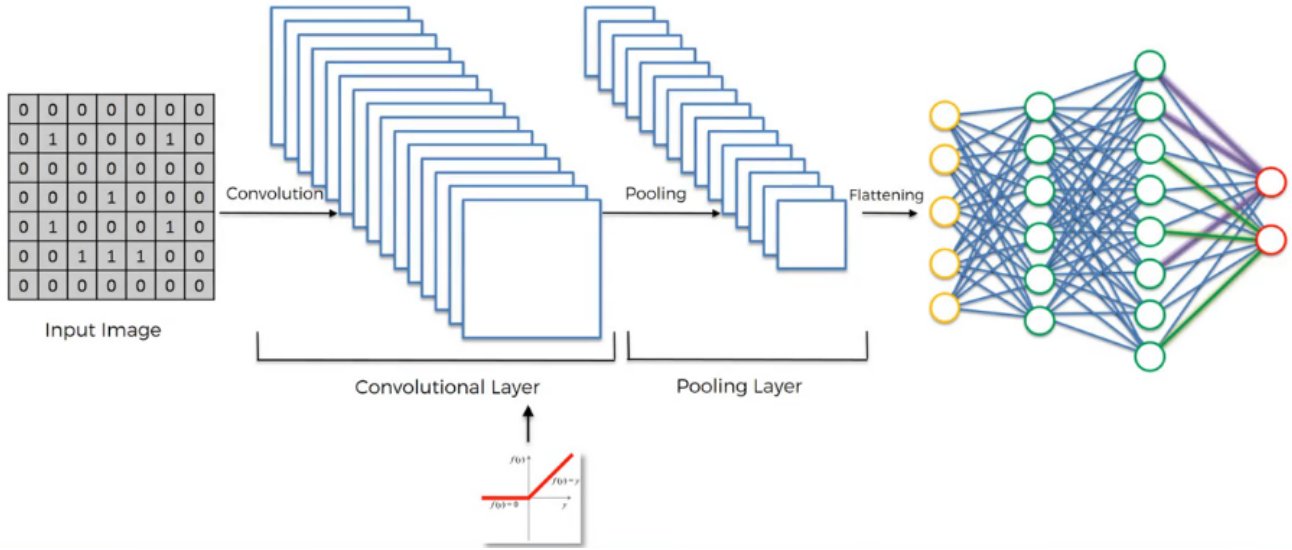
During training, loss can be calculated. p is actual value, whereas q is predicted value.



(Ref: Deep Learning A-Z - Kirill Eremenko)

Loss is minimized by adusting weights in NN as well as filter values.

Summary



(Ref: Deep Learning A-Z - Kirill Eremenko)

Are filters also optimized during Back-propagation?

- We don't know upfront what the filters are needed for any given problem.
- They get set automatically through the back-propagation procedure.
- Values inside each filter get updated along with other weights and biases.

(Ref: How filters are made in a CNN? - Stack Overflow)

CNN Architecture (Recap)

- A CNN is a list of layers that transform the input data into an output class/prediction.
- There are a few distinct types of layers:
  - Convolutional layer
  - Non-linear layer
  - Pooling layer

CNN Architecture (Recap)

- The Convolutional layer consists of a set of filters.
- Each filter covers a spatially small portion of the input data.
- Each filter is convolved across the dimensions of the input data, producing a multidimensional feature map.
- As we convolve the filter, we are computing the dot product between the parameters of the filter and the input.

CNN Architecture (Recap)

- Intuition: the network will learn filters that activate when they see some specific type of feature at some spatial position in the input.
  - Stacking multiple layers of feature extractors
  - Low-level layers extract local features.
  - High-level layers extract learn global patterns.
- The key architectural characteristics of the convolutional layer is local connectivity and shared weights.

Conclusion

Pros:

- Deep Convolutional Neural Networks represent current state-of-the-art techniques in image classification, object detection and localization
- Powerful CNN models are like AlexNet, InceptionNet, Deep Residual Networks
- Open-source libraries for deploying applications with CNNs very fast
- Convolutional Neural Networks can share pre-trained weights, which is the basis for transfer learning

Conclusion

Cons:

- The interpretation and mechanism of CNNs are not clear, we don't know why they work better than previous models
- Large number of training data and annotations are needed, which may not be practical in some problems.

CNN with TensorFlow 2.0

MNIST: Handwritten Digits Classification

Ref: The Ultimate Beginner's Guide to Deep Learning in Python - EliteDataScience

Import Libraries

- Importing numpy and setting a seed for the computer's pseudo-random number generator.

```
1 import numpy as np
  np.random.seed(123)  # for reproducibility
```

- Keras libs

```
  from tf.keras.models import Sequential
2 from tf.keras.layers import Dense, Dropout, Activation, Flatten
  from tf.keras.layers import Conv2D, MaxPooling2D
```

Load Data

- MNIST is included in Keras
- It's a big enough challenge to warrant neural networks, but it's manageable on a single computer.

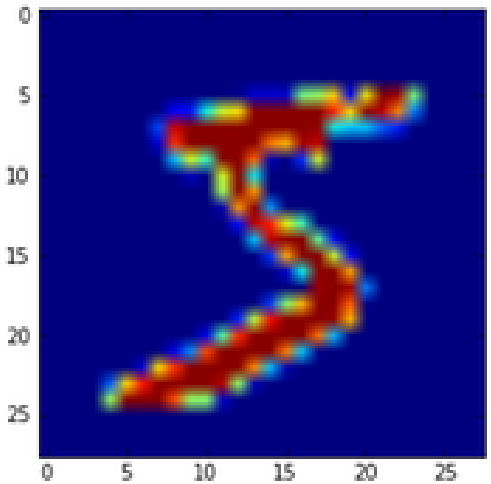
```
from tf.keras.datasets import mnist

# Load pre-shuffled MNIST data into train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

print(X_train.shape)
# (60000, 28, 28)
print(X_train[0].shape)
# (26,28)
```

We have 60,000 samples in our training set, and the images are 28 pixels x 28 pixels each.

Load Data

- Check by plotting a sample image

```
from matplotlib import pyplot as plt
plt.imshow(X_train[0])
plt.show()
```



Input Shape in MNIST

- A full-color image with all 3 RGB channels will have a depth of 3.
- Our MNIST images only have a depth of 1
- We want to transform our dataset from having shape (n, width, height) to (n, width, height, depth) in `data_format= "channels_last"`.

```
keras.backend.set_image_data_format('channels_last')
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
print(X_train.shape)
# (60000, 1, 28, 28)
```

Pre-process Data

- Convert our data type to float32
- Normalize our data values to the range [0, 1].

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

Now, our input data Xs are ready for model training.

Pre-process Labels

- Let's take a look at the shape of our class label data:

```
print(y_train.shape)
# (60000,)
```

- That may be problematic.
- We should have 10 different classes, one for each digit, but it looks like we only have a 1-dimensional array.
- Let's take a look at the labels for the first 10 training samples:

```
print(y_train[:10])
# [5 0 4 1 9 2 1 3 1 4]
```

Pre-process Labels

- And there's the problem.
- The y_train and y_test data are not split into 10 distinct class labels, but rather are represented as a single array with the class values.
- We can fix this easily:

```
from tf.keras.utils import to_categorical

# Convert 1-dimensional class arrays to 10-dimensional class matrices
Y_train = to_categorical(y_train)
Y_test = to_categorical(y_test)
```

- See now:

```
print(Y_train.shape)
# (60000, 10)
```

Define Model

- The input shape parameter should be the shape of 1 sample.
- In this case, it's the same (28, 28, 1) that corresponds to the (width, height,depth) of each digit image.
- But what do the first 3 parameters represent?
- They correspond to the number of convolution filters to use, the number of rows in each convolution kernel, and the number of columns in each convolution kernel, respectively.
- We can confirm this by printing the shape of the current model output:

```
model = Sequential()
model.add(Conv2D(32,  kernel_size=3, activation='relu', input_shape=(28,28,1)))
print(model.output_shape)
# (None, 32, 26, 26)
```

*Note: The step size is (1,1) by default, and it can be tuned using the 'subsample' parameter.

Conv2D output size

- Tensor size or shape: (width $= 28$, height $= 28$)
- Convolution filter size (F): (F_width $= 3$, F_height $= 3$)
- Padding (P): 0
- Stride (S): 1
- Using the equation:
- output width$=((W - F + 2 * P)/S) + 1 = ((28 - 3 + 2 * 0)/1) + 1 = 26$

Pooling

- We can simply add more layers to our model like we're building legos.
- Dropout is a method for regularizing our model in order to prevent overfitting.
- Let also remember that a pooling is a form of "filter" and thus the $((W - F + 2 * P)/S) + 1$ equation is a applicable.
- So after max pooling of 2x2 with Stride 2, it will be $= (((W - F + 2 * P)/S) + 1) = (((26 - 2 + 2 * 0)/2) + 1 = 13$

```
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
```

Define Model

- For Dense layers, the first parameter is the output size of the layer. Keras automatically handles the connections between layers.
- Note that the final layer has an output size of 10, corresponding to the 10 classes of digits.
- Also note that the weights from the Convolution layers must be flattened (made 1-dimensional) before passing them to the fully connected Dense layer.

```
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

Compile Model

- We just need to compile the model and we'll be ready to train it.
- When we compile the model, we declare the loss function and the optimizer (SGD, Adam, etc.).
- Keras has a variety of loss functions and out-of-the-box optimizers to choose from.

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Fit Model

- To fit the model, all we have to do is declare the batch size and number of epochs to train for,
- Then pass in our training data.

```
model.fit(X_train, Y_train,
          batch_size=32, nb_epoch=10, verbose=1)
```

You can also use a variety of callbacks to set early-stopping rules, save model weights along the way, or log the history of each training epoch.

Evaluate Model

We can evaluate our model on the test data.

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(score)
0.9855
```

References

Many publicly available resources have been refereed for making this presentation. Some of the notable ones are:

- TensorFlow 2 Tutorial: Get Started in Deep Learning With tf.keras - Jason Brownlee
- Deep Learning using Keras- Alyosamah
- Introduction to Keras - Francois Chollet
- Michael Nielsen's Neural Networks and Deep Learning: `http://neuralnetworksanddeeplearning.com/`

Thanks ... yogeshkulkarni@yahoo.com