UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

SCS 3253 Analytics Techniques and Machine Learning

Module 11: Distributing TensorFlow, CNNs & RNNs

Course Plan

Module Titles
Module 1 – Current Focus: Introduction to Machine Learning
Module 2 – End to End Machine Learning Project
Module 3 – Classification
Module 4 – Clustering and Unsupervised Learning
Module 5 – Training Models and Feature Selection
Module 6 – Support Vector Machines
Module 7 – Decision Trees and Ensemble Learning
Module 8 – Dimensionality Reduction
Module 9 – Introduction to Neural Networks & Deep Learning
Module 10 – Introduction to TensorFlow
Current Focus: Module 11 – Distributing TensorFlow, CNNs and RNNs
Module 12 – Final Assignment and Presentations (no content)

Learning Outcomes for this Module

- How to distribute computations across several CPUs and/or GPUs
- Convolutional neural networks
- Recurrent neural networks

Topics in this Module

- **11.1** Distributed Training
- **11.2** Convolutional Neural Networks
- **11.3** CNN Architectures
- **11.4** Recurrent Neural Networks
- **11.5** Wrap-up

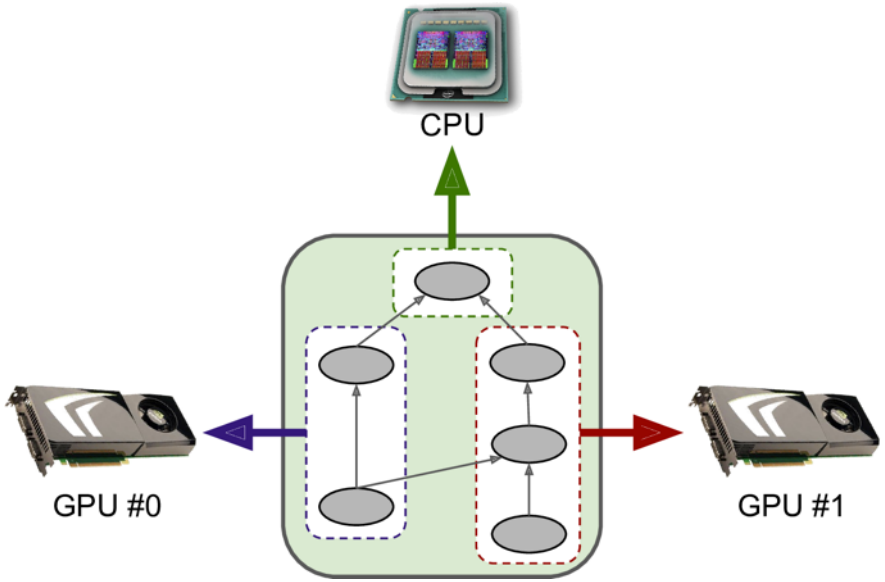
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 11 – Section 1

Distributed Training

Improve Training

- Many options:
  - Better weight initialization,
  - Batch Normalization,
  - Sophisticated optimizers, etc.…
- However, training a large neural network on a single machine with a single CPU can take days or even weeks.

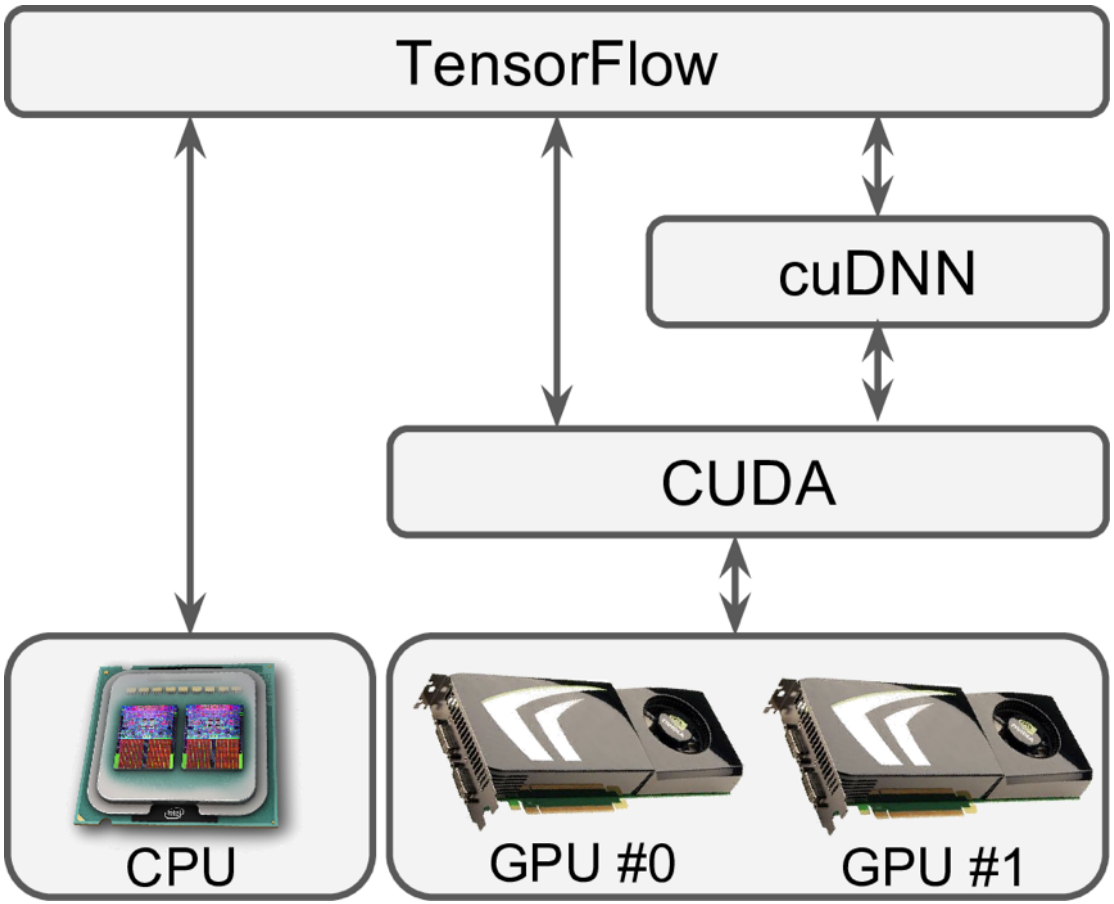


Cloud is Future of AI/ML Training Environment

- If you don't own any GPU cards, you can use a hosting service with GPU capability such as Amazon AWS.
  - https://aws.amazon.com/tensorflow/
  - https://cloud.google.com/solutions/running-distributed-tensorflow-on-compute-engine

- You must then download and install the appropriate version of the CUDA and cuDNN libraries
  - *Compute Unified Device Architecture* library (CUDA) by Nvidia
  - *CUDA Deep Neural Network* library (cuDNN)
  - CUDA as a parallel computing platform enables use of CUDA-enabled GPU for general purpose processing, including ML/AI

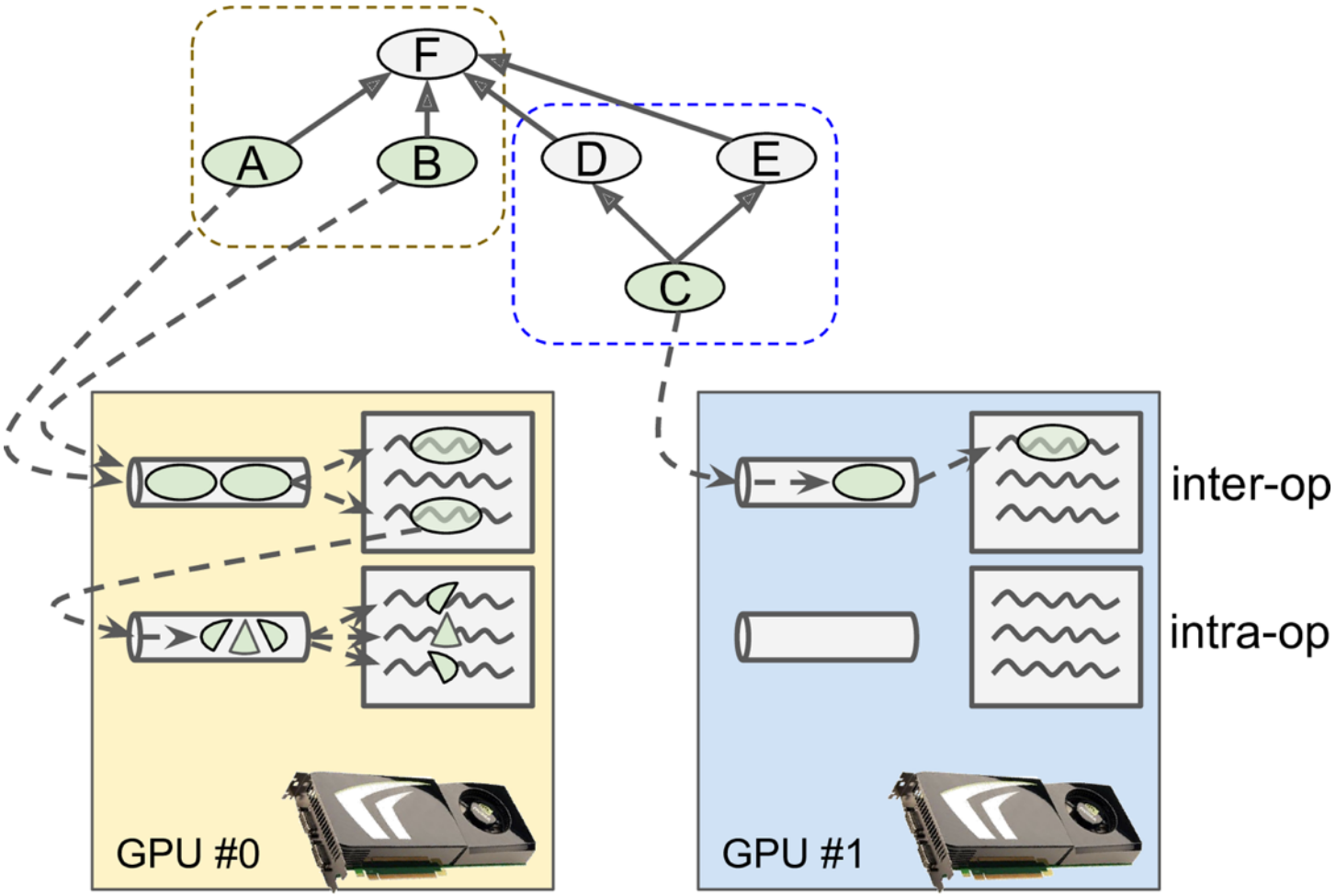
Distributed TF



Distributed TF

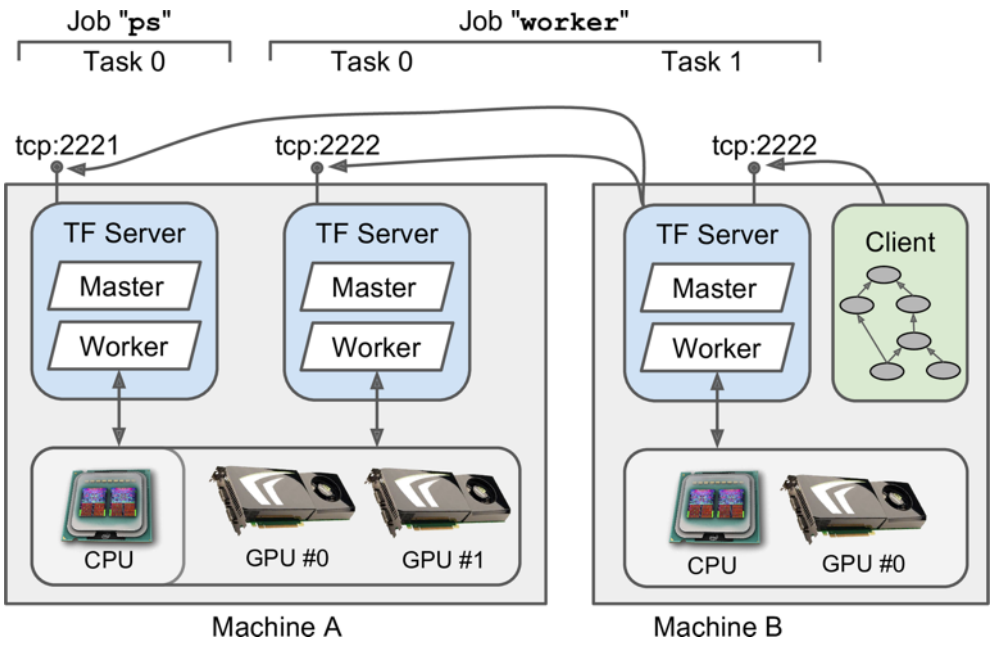
```
$ nvidia-smi
Wed Sep 16 09:50:03 2016
+-----------------------------------------------------+
| NVIDIA-SMI 352.63     Driver Version: 352.63        |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GRID K520           Off  | 0000:00:03.0     Off |                  N/A |
| N/A   27C    P8    17W / 125W |     11MiB /  4095MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Parallel Execution ution *intra-op thread pools*



Distributed Computing: Multiple Devices Across Servers

- A cluster is composed of one or more TensorFlow servers, called *tasks*, typically spread across several machines.
- Each task belongs to a *job*.



If you're curious for more try this notebook:
https://github.com/ageron/handson-ml/blob/master/12_distributed_tensorflow.ipynb

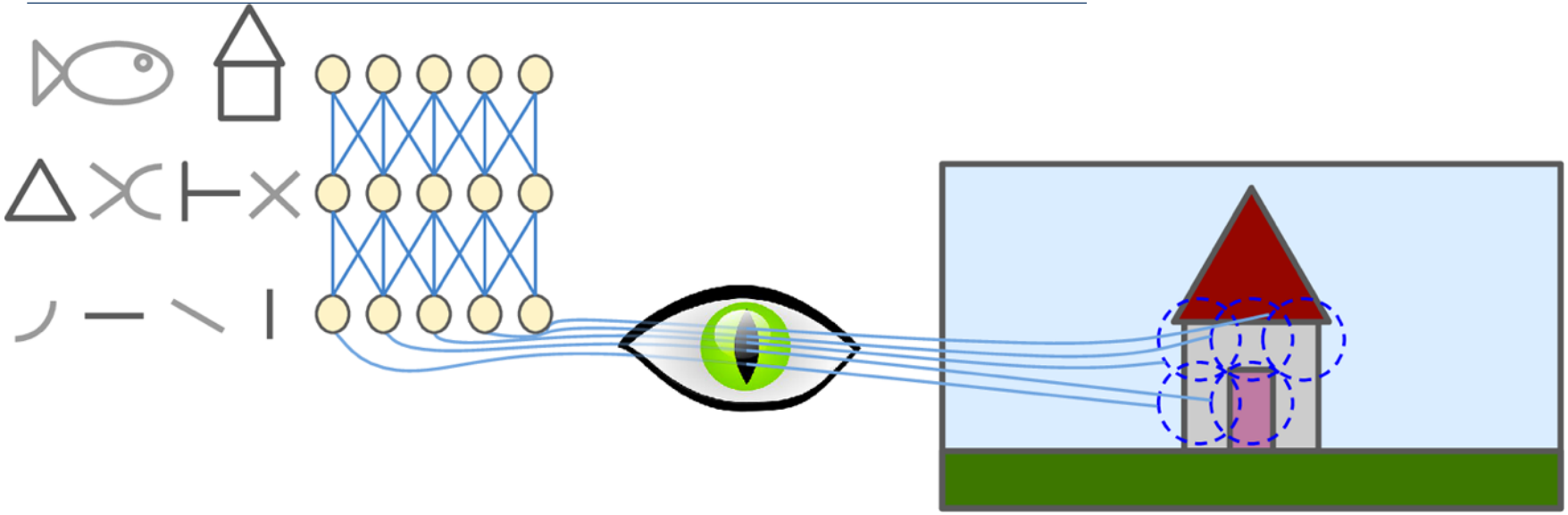
Module 11 – Section 2

Convolutional Neural Networks

Convolutional Neural Networks

- CNNs emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s.
- In the last few years, thanks to the increase in computational power, the amount of available training data, CNNs have managed to achieve superhuman performance on some complex visual tasks.
- They power image search services, self-driving cars, automatic video classification systems, and more.
- Moreover, CNNs are not restricted to visual perception: they are also successful at other tasks, such as *voice recognition* or *natural language processing* (NLP)

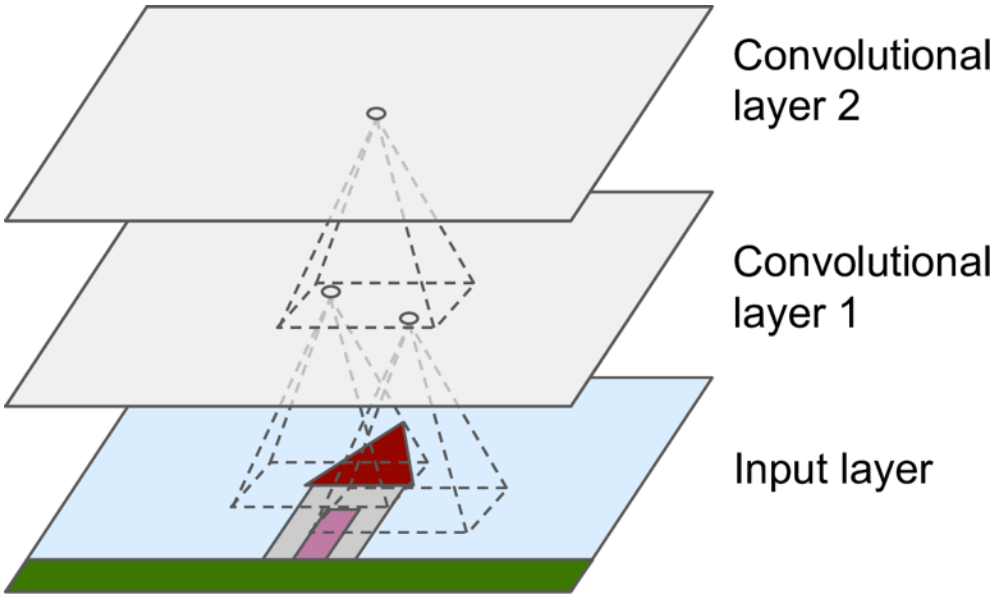
Convolutional Neural Networks



- Some neurons have larger receptive fields, and they react to more complex patterns that are combinations of the lower-level patterns

Convolutional Neural Networks

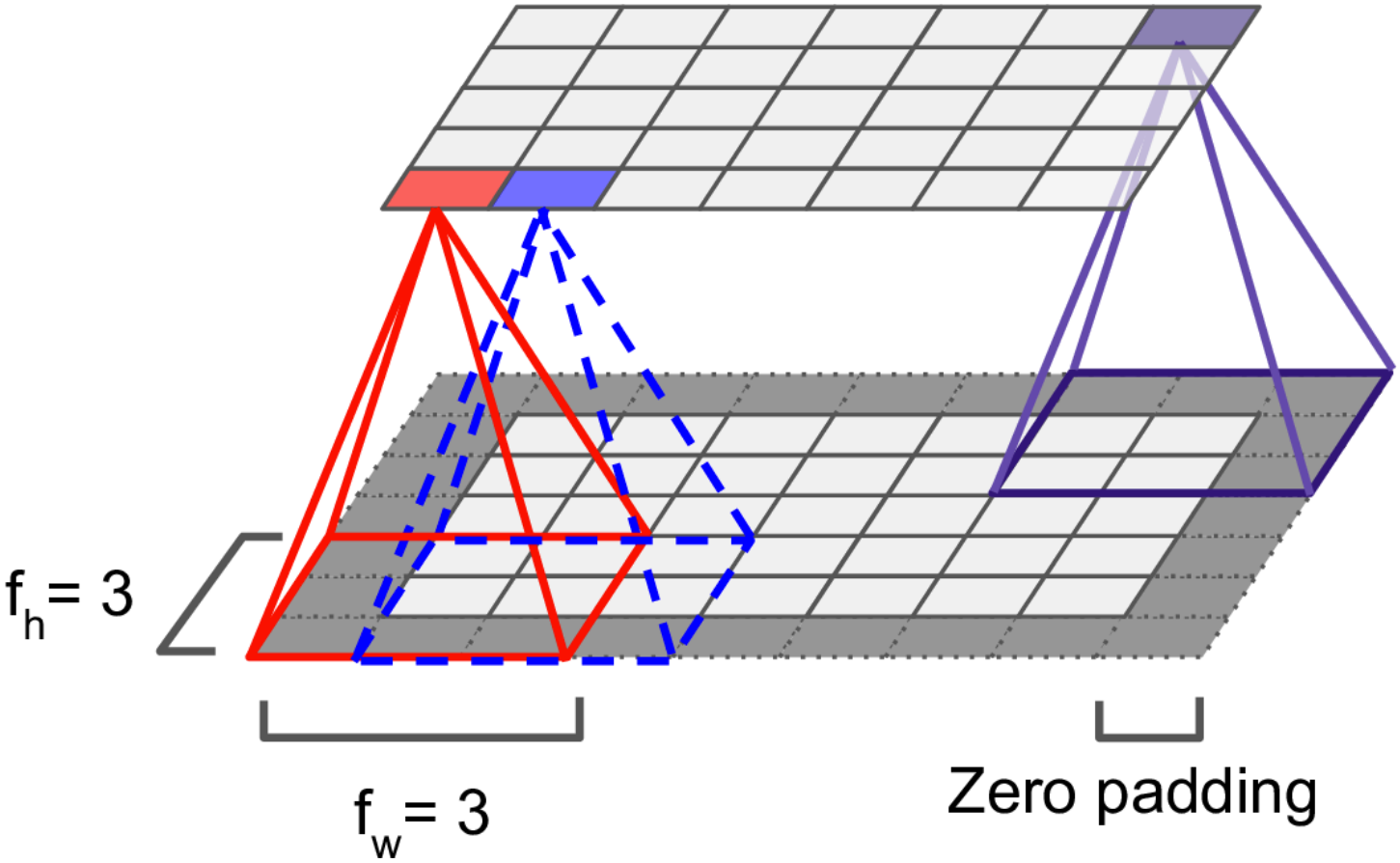
- Neurons in the first convolutional layer are not connected to every single pixel in the input image
- This architecture allows the network to concentrate on low-level features in the first hidden layer, then assemble them into higher-level features in the next hidden layer, and so on..



Without convolutions:
https://scs.ryerson.ca/~aharley/vis/fc/

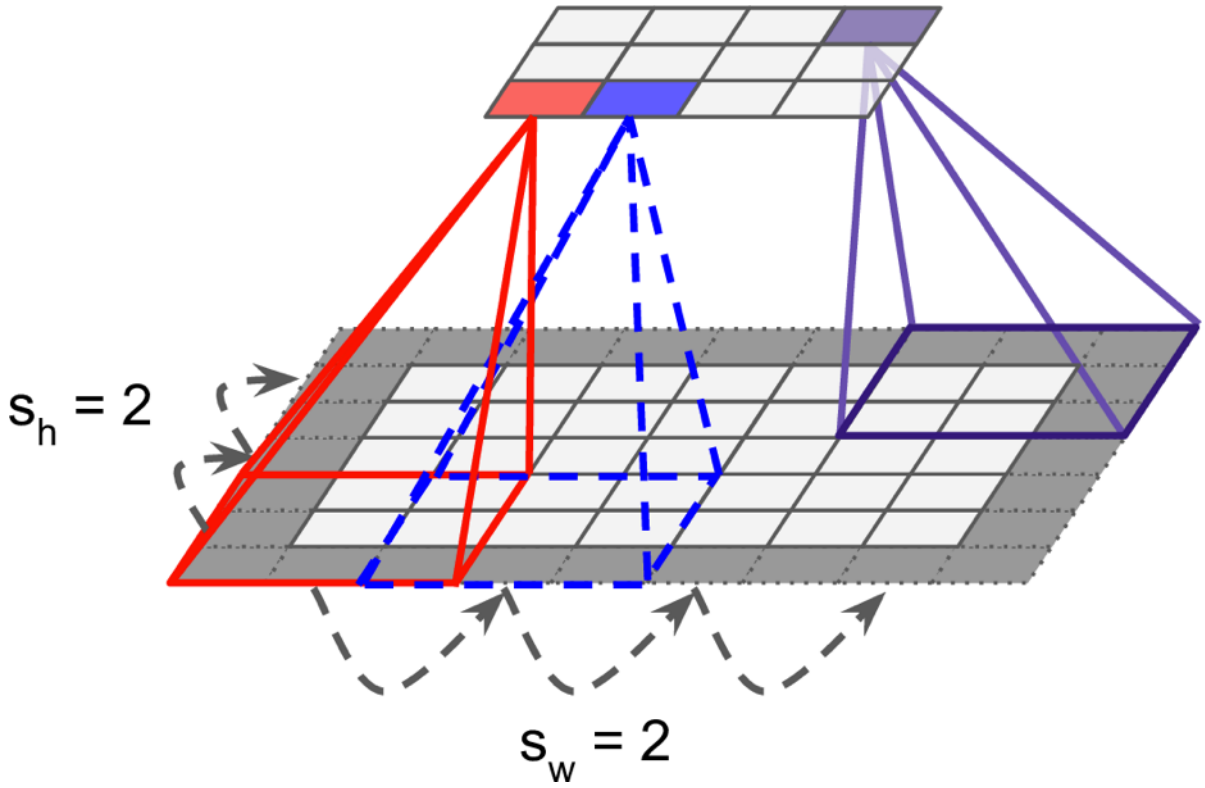
With convolutions:
https://scs.ryerson.ca/~aharley/vis/conv/flat.html

Convolutional Neural Networks



- Layer connections and receptive fields

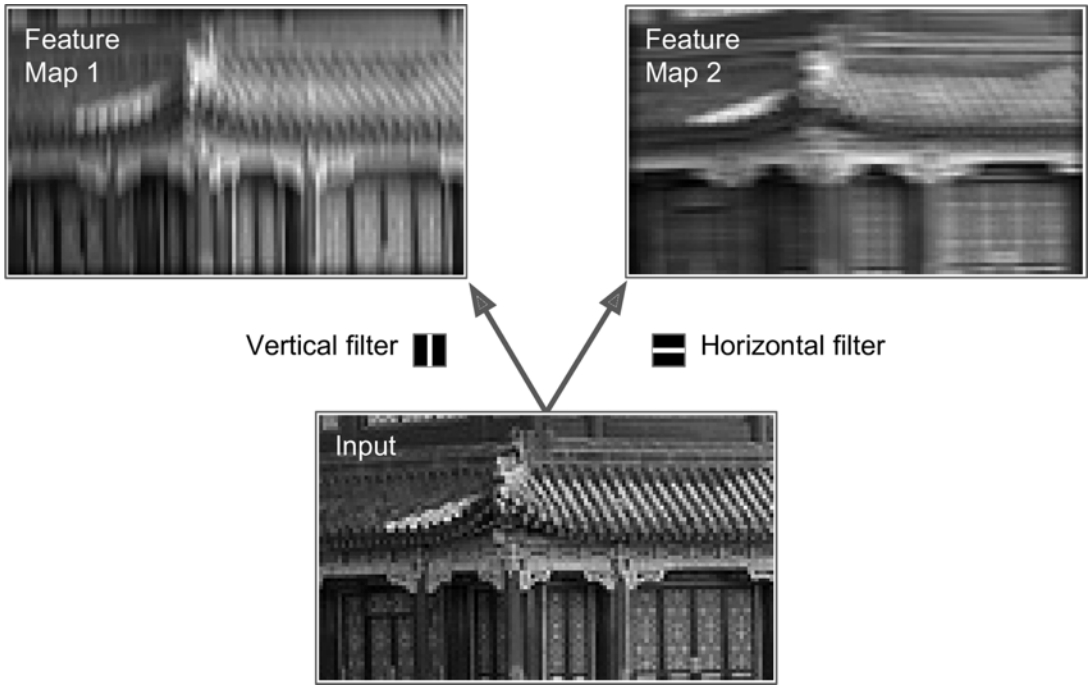
Convolutional Neural Networks



- If stride size > 1 the dimensionality of the convolutional layer is reduced

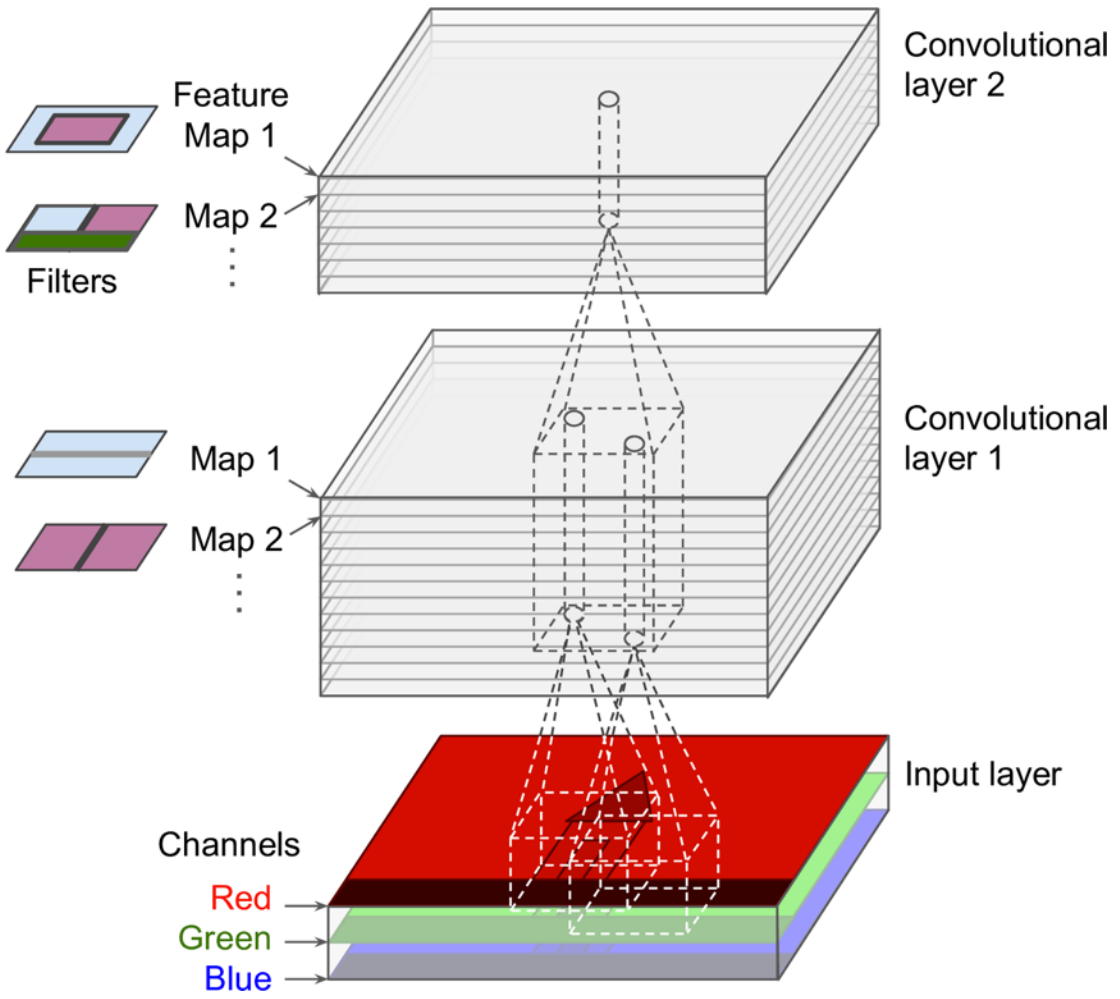
Filters or Receptive Fields

- A neuron's weights can be represented as a small image the size of the receptive field (or *convolution kernels*)
- During training, a CNN finds the most useful filters for its task, and it learns to combine them into more complex patterns



Filters or Receptive Fields

- Each feature map share the same parameters which in turn reduces the number of parameters in the model
- Once the CNN has learned to recognize a pattern in one location, it can recognize it in any other location.
- A regular DNN can recognize only recognize pattern in the region they learned



Filters

- If your receptive field is 3X3, filters are matrixes like below and if you apply each different filter on pictures, a specific aspect of the image will be highlighted

1	0	0
0	1	0
0	0	1

1	1	1
0	0	0
1	1	1

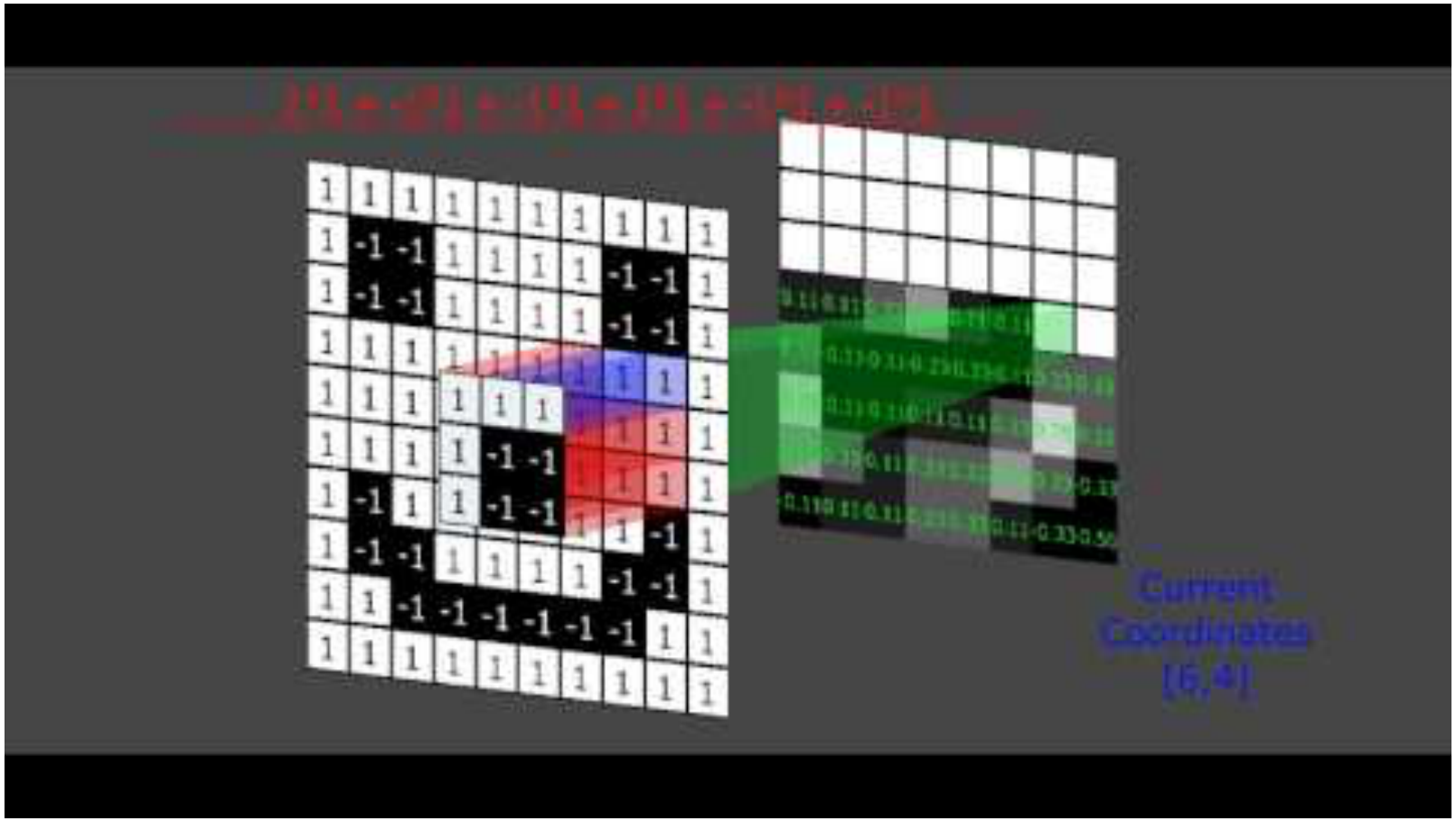
Example Filters and their impact on image

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Example Filters and their impact on image

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Filters or Receptive Fields



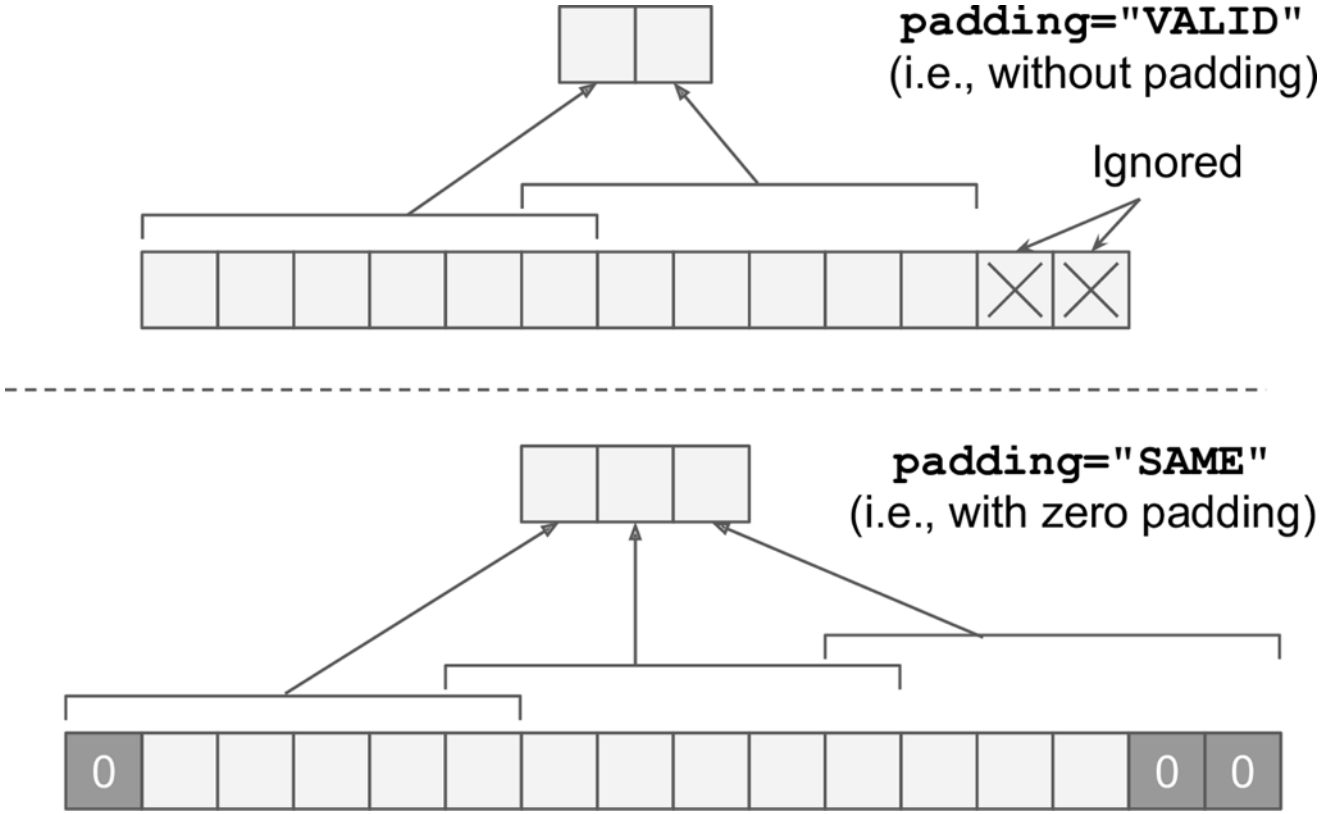
Short URL to the Original YourTube Video ==> shorturl.at/achiN

Convolutional Layer Output

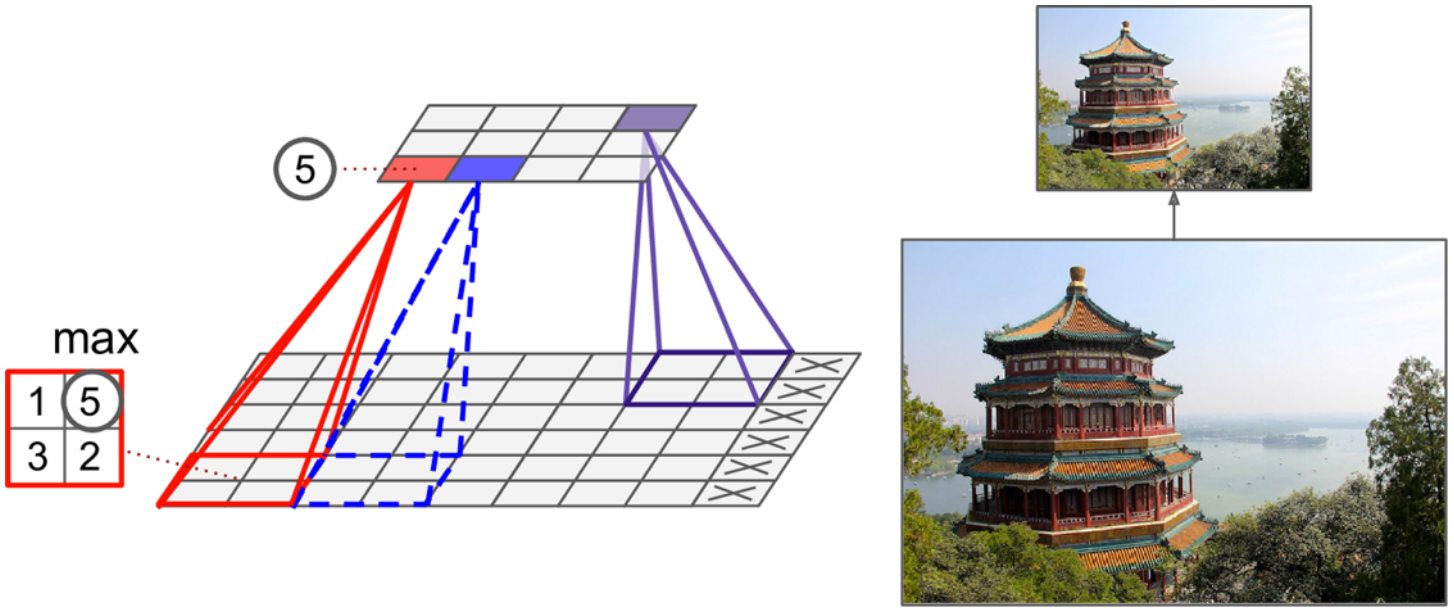
$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

- $z_{i,j,k}$ is the output of the neuron located in row i, column j in feature map k of the convolutional layer (layer l).
- s_h and s_w are the vertical and horizontal strides, f_h and f_w are the height and width of the receptive field, and $f_{n'}$ is the number of feature maps in the previous layer (layer $l - 1$).
- $x_{i',j',k'}$ is the output of the neuron located in layer $l - 1$, row i', column j', feature map k' (or channel k' if the previous layer is the input layer).
- b_k is the bias term for feature map k (in layer l). You can think of it as a knob that tweaks the overall brightness of the feature map k.
- $w_{u,v,k',k}$ is the connection weight between any neuron in feature map k of the layer l and its input located at row u, column v and feature map k'.

Padding


padding="VALID"
(i.e., without padding)
Ignored
padding="SAME"
(i.e., with zero padding)
0
0
0

Pooling Layer



- Their main goal is to subsample the input image
- A pooling neuron has no weights; all it does is aggregate the inputs using an aggregation function such as the max or mean

Pooling Layer

- With Keras:

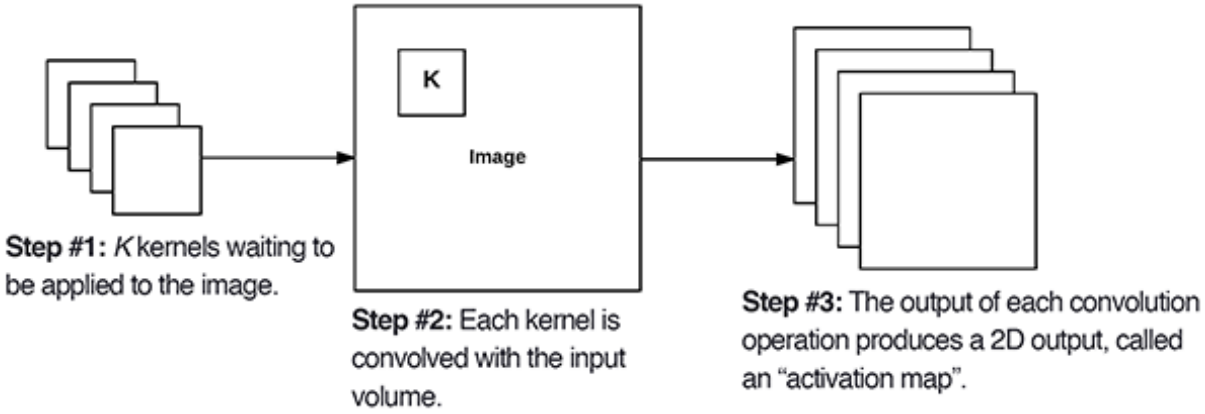
```
max_pool = keras.layers.MaxPool2D(pool_size=2)
```

- With low level TF:

```
output = tf.nn.max_pool(images,
                        ksize=(1, 1, 1, 3),
                        strides=(1, 1, 1, 3),
                        padding="valid")
depth_pool = keras.layers.Lambda(
    lambda X: tf.nn.max_pool(X, ksize=(1, 1, 1, 3), strides=(1, 1, 1, 3),
                             padding="valid"))
```

- The ksize argument contains the kernel shape along all four dimensions of the input tensor: [batch size, height, width, channels]
- This lets you convolve over non-spatial dimensions (channel)
- To create an *average pooling layer*, just use the avg_pool() function instead of max_pool()

Putting It All Together



0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

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

114				

https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

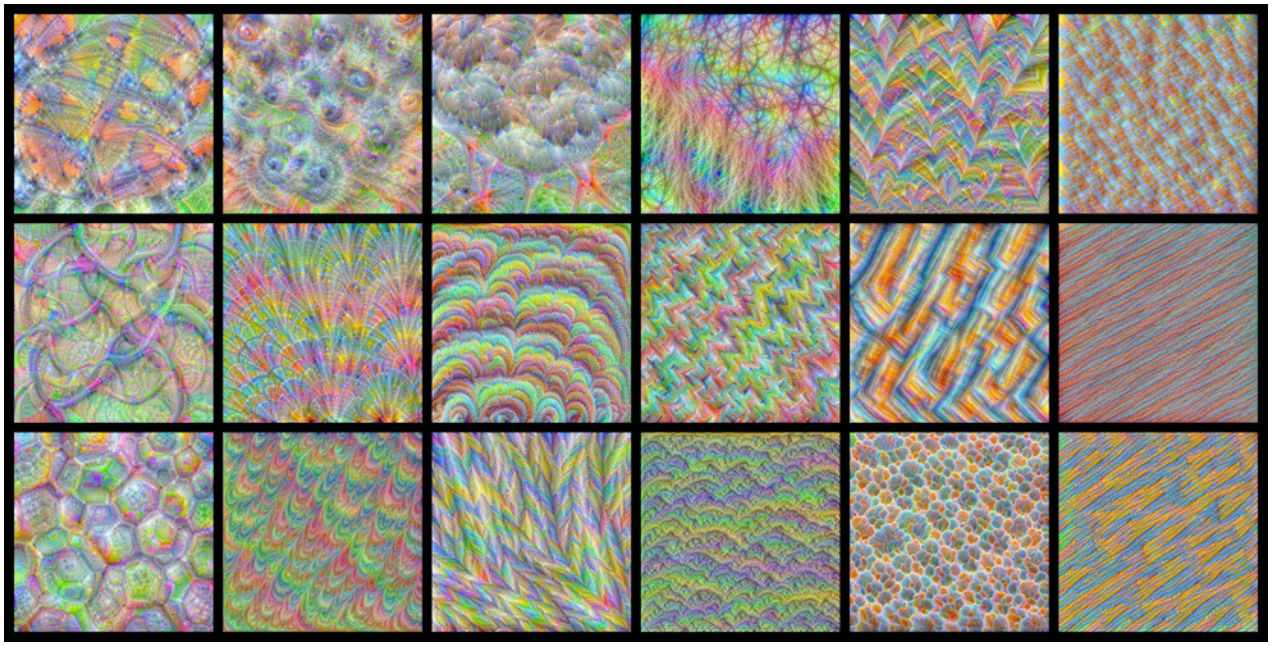
TF CNNs

- Key elements:
  - 64 filters, 7x7 size, no stride, padding maintains size
  - 128 filters, 3x3 size, then 256 filters, 3x3 size, repeated
  - Max pooling by 2, repeated 3 times, plus dropout

```
model = keras.models.Sequential([
    keras.layers.Conv2D(64, 7, activation="relu", padding="same",
                        input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation="softmax")
```

Visualizing Features

- The filters turn the image pixels into higher level features
- These are learned and can include:
  - Edges, colors, textures, patterns, objects
  - See for yourself: https://distill.pub/2017/feature-visualization/



UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 11 – Section 3

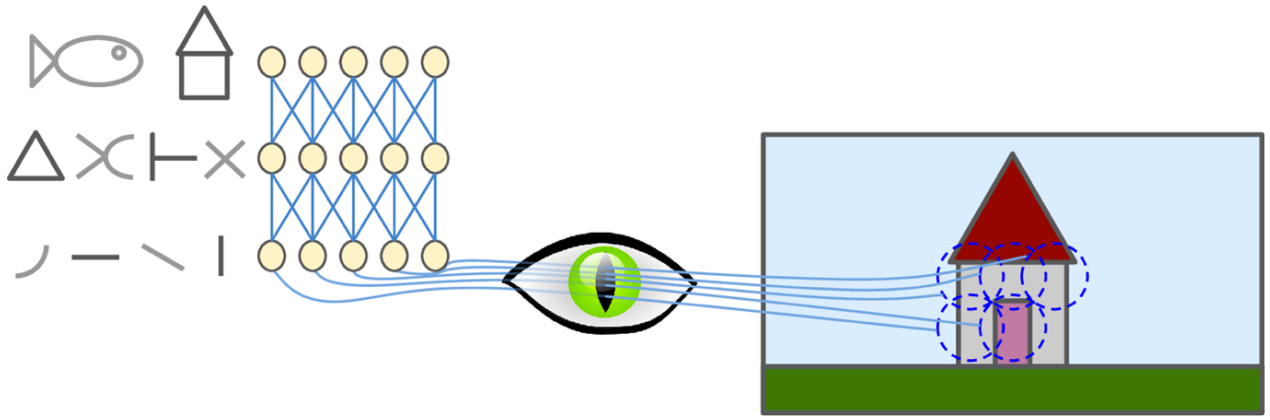
CNN Architectures

What is Convolution Layer?

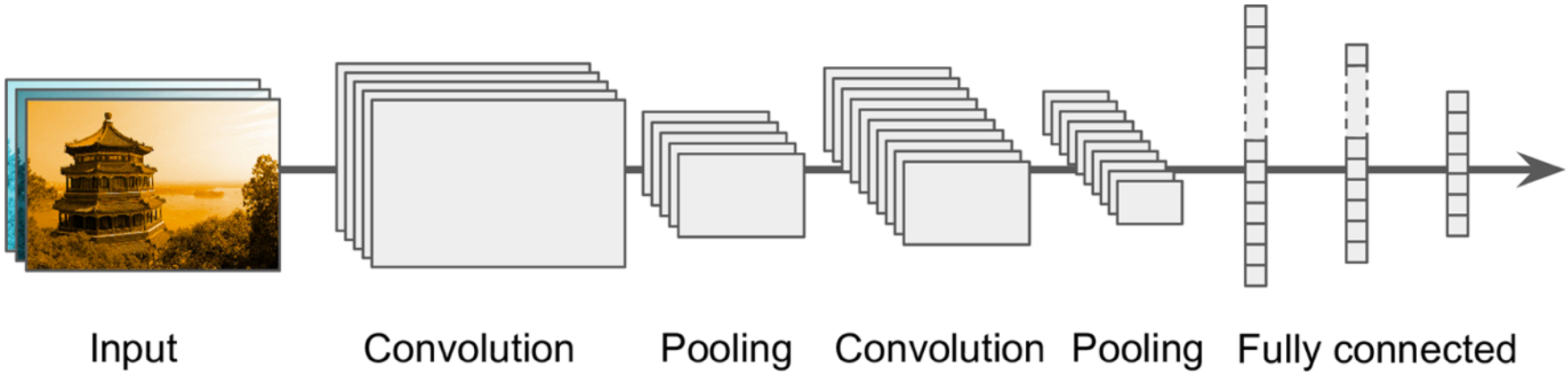
- A convolution is

What is Convolution Layer?

- A convolution is the application of a filter to an input that results in an activation.
- Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image.



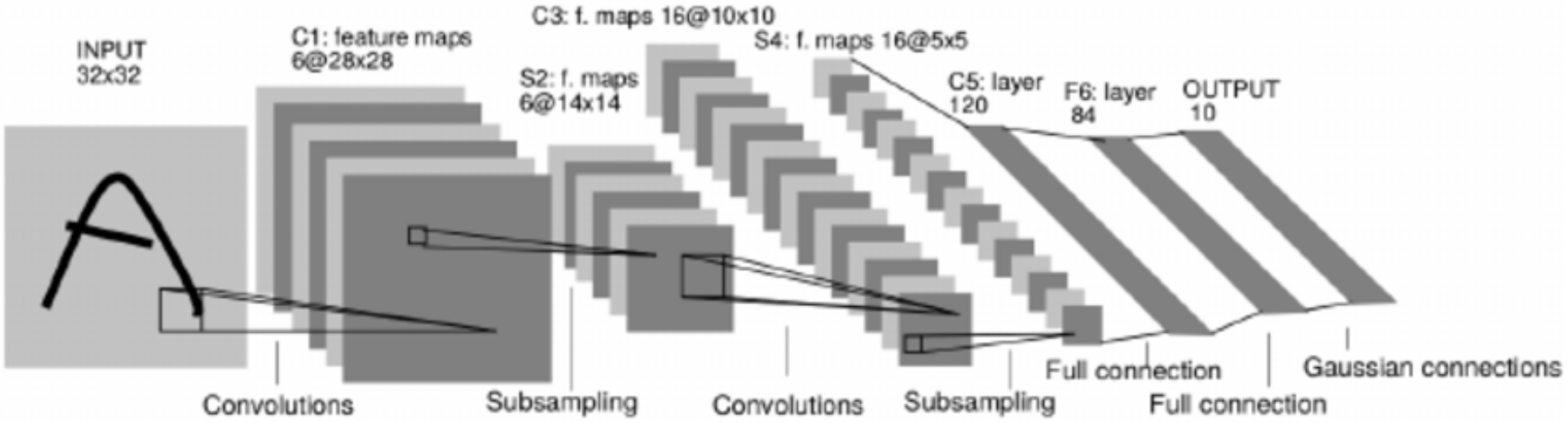
CNN Architectures



- CNN architectures stack a few convolutional layers, then a pooling layer, then another few convolutional layers (+ReLU), then another pooling layer, and so on.
- The image gets smaller and smaller as it progresses through the network, but it also typically gets deeper and deeper.
- At the top of the stack, a regular feedforward neural network (still has backpropagation algorithm to optimize cost) is added, composed of a few fully connected layers (+ReLUs), and the final layer outputs the prediction

LeNet-5

1998
Early!



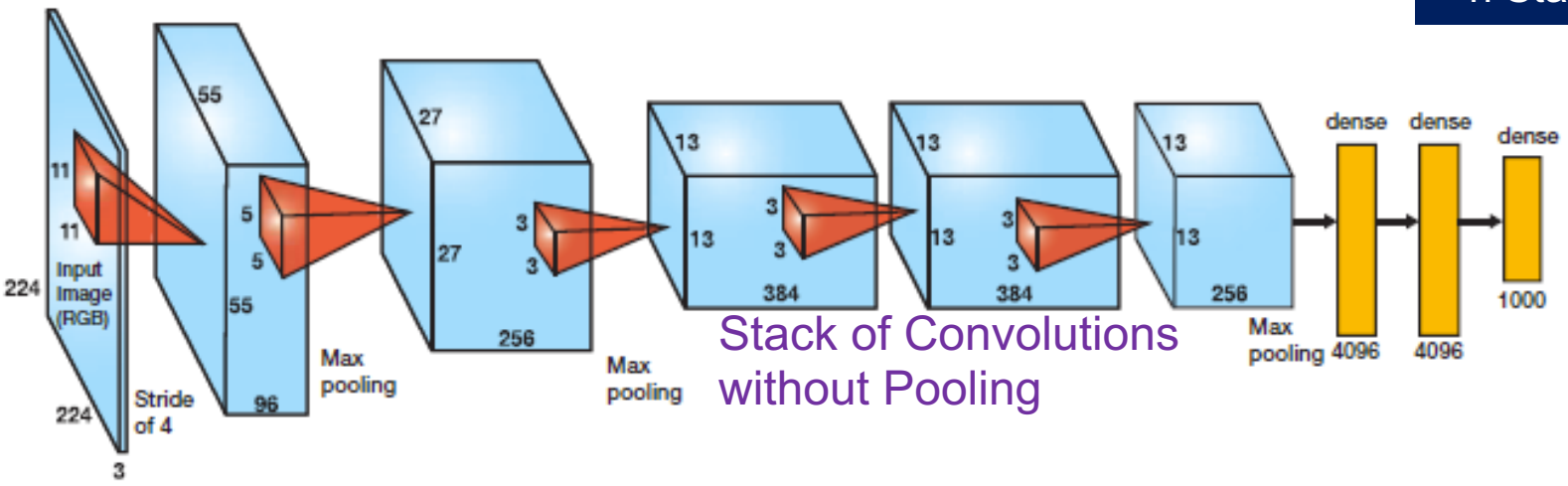
- The LeNet-5 architecture is perhaps the most widely known CNN architecture. It was created by Yann LeCun in 1998 and widely used for handwritten digit recognition (MNIST). Subsampling is pooling.
  - http://yann.lecun.com/exdb/lenet/

- MNIST images are 28 × 28 pixels, but they are zero-padded to 32 × 32 pixels and normalized before being fed to the network.

Q - Why normalizing input before feeding?

AlexNet

2012
Convolutio
n Stack



- Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton
- Won 2012 Image-Net ILSVRC Challenge Award!
- It is quite similar to LeNet-5, only much larger and deeper, and it was the first to stack convolutional layers directly on top of each other, instead of stacking a pooling layer on top
- AlexNet also uses a **competitive normalization** step immediately after the ReLU step of layers C1 and C3, called ***local response normalization*.**

AlexNet Layers

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	–	1,000	–	–	–	Softmax
F10	Fully connected	–	4,096	–	–	–	ReLU
F9	Fully connected	–	4,096	–	–	–	ReLU
S8	Max pooling	256	6 × 6	3 × 3	2	valid	–
C7	Convolution	256	13 × 13	3 × 3	1	same	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	same	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	same	ReLU
S4	Max pooling	256	13 × 13	3 × 3	2	valid	–
C3	Convolution	256	27 × 27	5 × 5	1	same	ReLU
S2	Max pooling	96	27 × 27	3 × 3	2	valid	–
C1	Convolution	96	55 × 55	11 × 11	4	valid	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

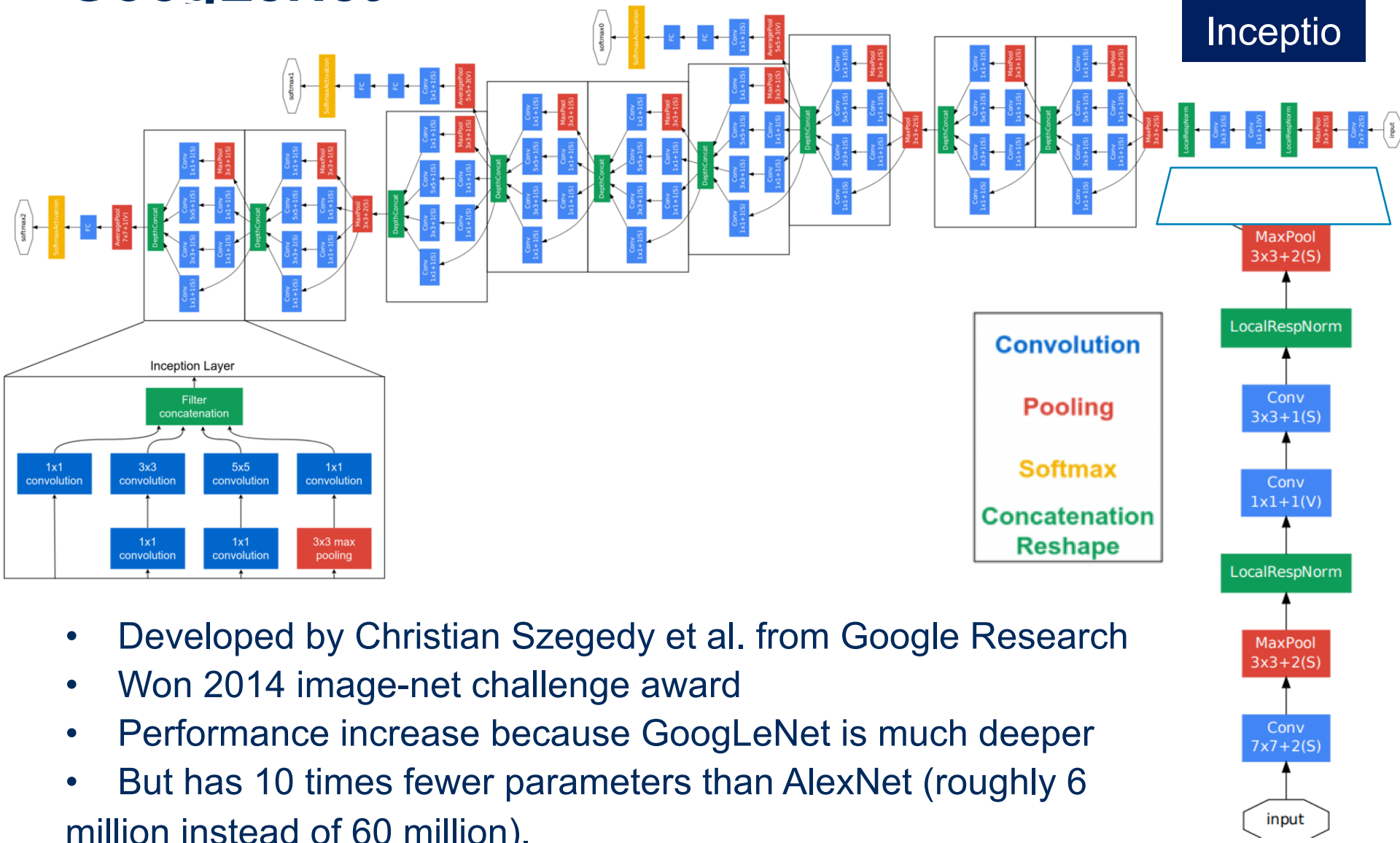
To address overfitting:

- **50% Dropout** is added at the outputs of F8 and F9

- Perform **image augmentation** (randomly shifting training images by different offset, rotating, and changing lighting)

GoogLeNet

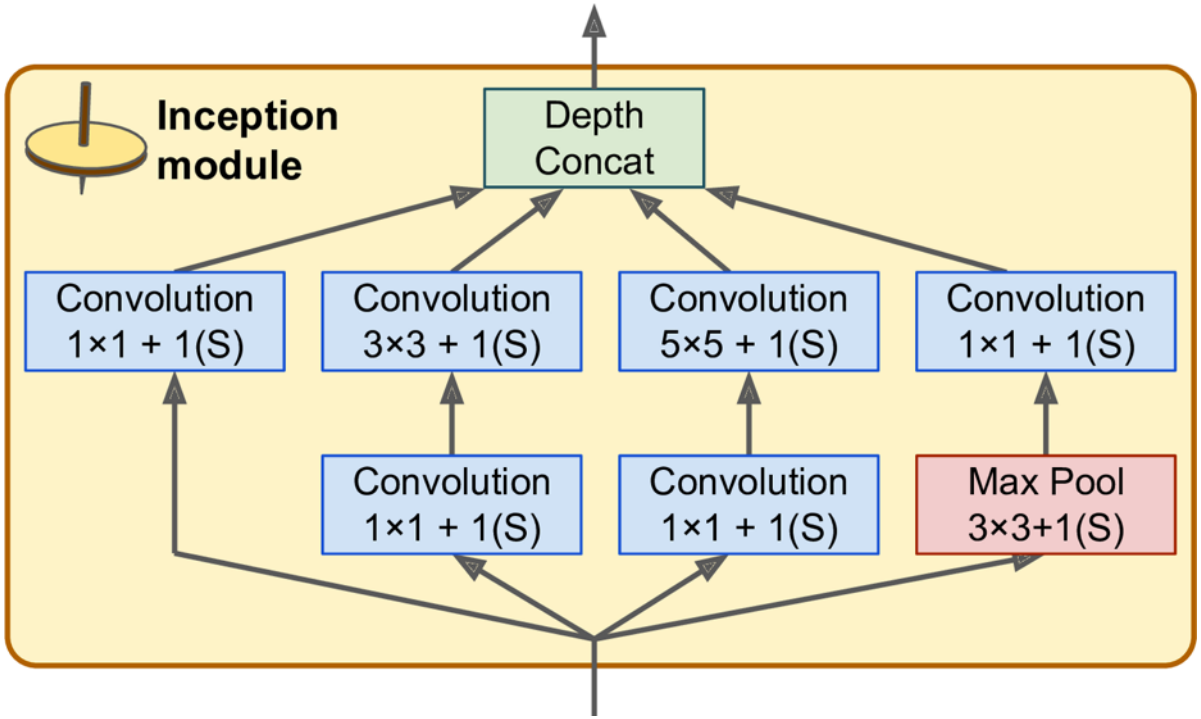
2015
Inceptio

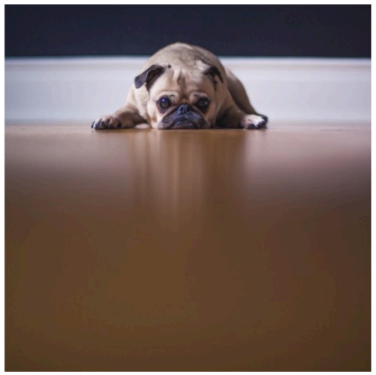


- Developed by Christian Szegedy et al. from Google Research
- Won 2014 image-net challenge award
- Performance increase because GoogLeNet is much deeper
- But has 10 times fewer parameters than AlexNet (roughly 6 million instead of 60 million).

Paper: https://arxiv.org/pdf/1409.4842v1.pdf

GoogLeNet – Inception Modules



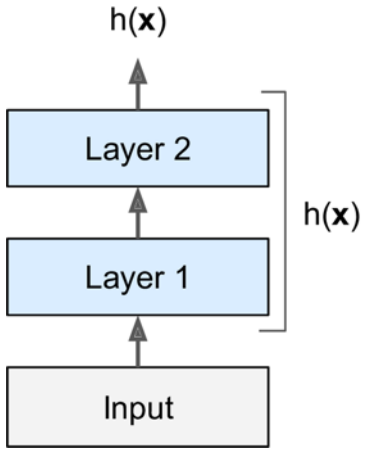


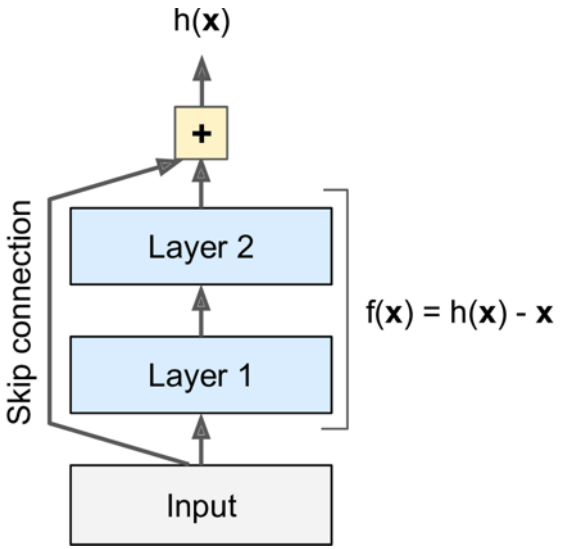
- Dogs are unequally occupying space. The second set of convolutional layers uses different kernel/filter sizes (1×1, 3×3, and 5×5), allowing them to capture patterns at different scales.
- The input signal is first copied and fed to four different layers.
- All convolutional layers use the **ReLU** activation function.

Residual Learning

2015
Residual Learning

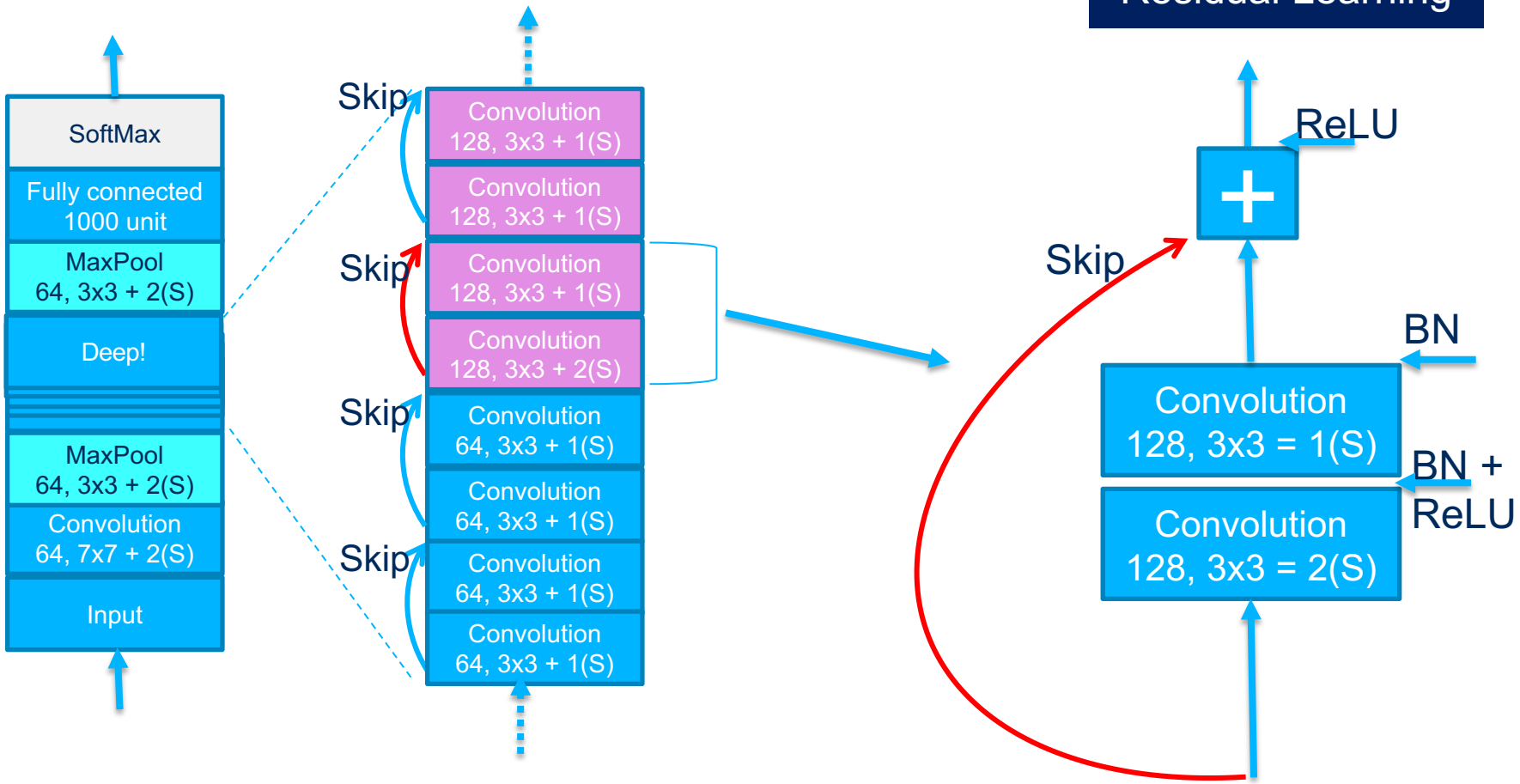
- When training a neural network, the goal is to make it model a target function $h(\mathbf{x})$
- If you add the input **x** to the output of the network (i.e., you add a skip connection), then the network will be forced to model $f(\mathbf{x}) = h(\mathbf{x}) - \mathbf{x}$ rather than $h(\mathbf{x})$
- This is called *residual learning*





ResNet

2015
Residual Learning

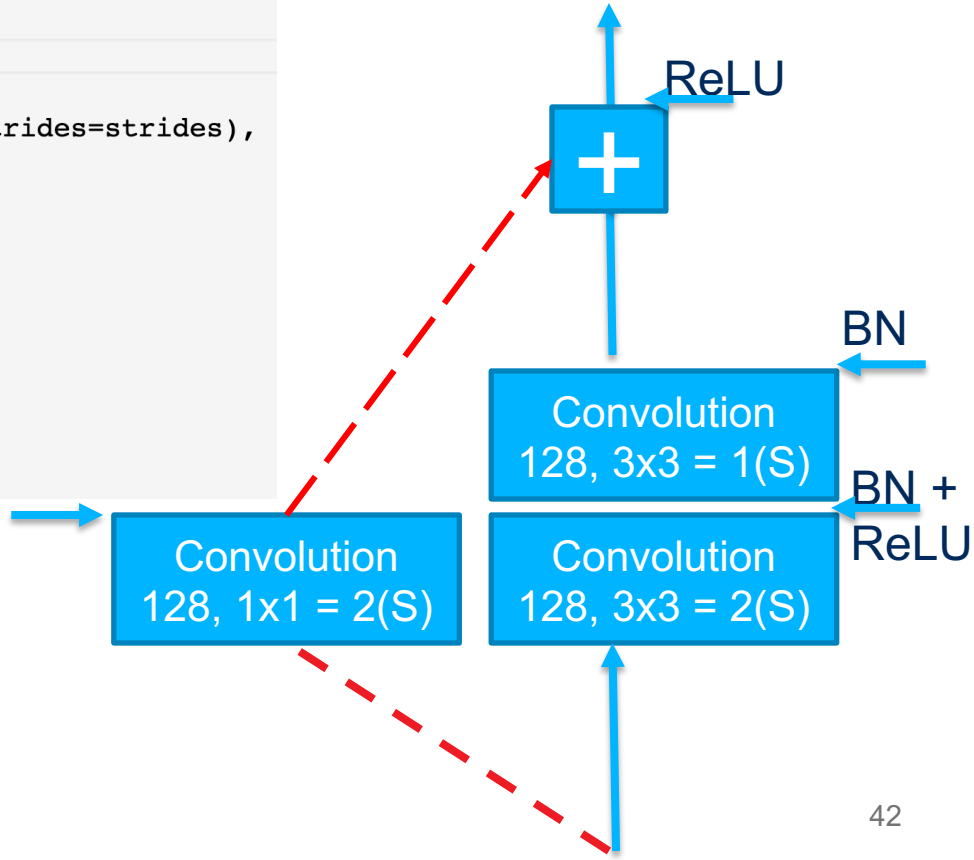


152 layers compared to other architectures of 32

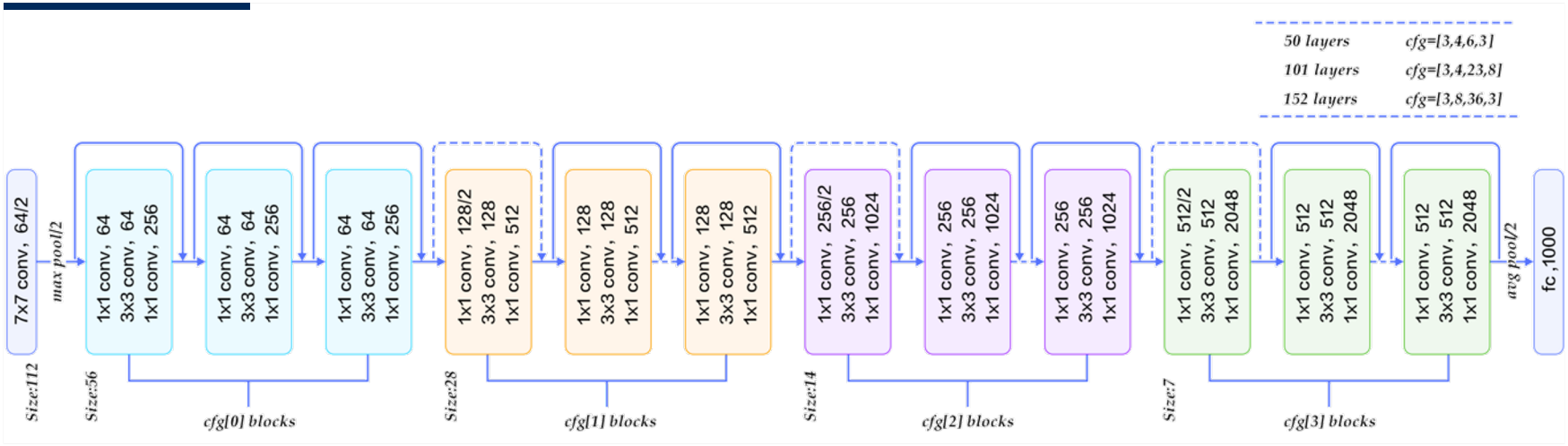
- Residual Network (or *ResNet*), developed by Kaiming He et al.
- Via skip connections, the signal can easily travel across the whole network
- The deep residual network can be seen as a stack of *residual units*

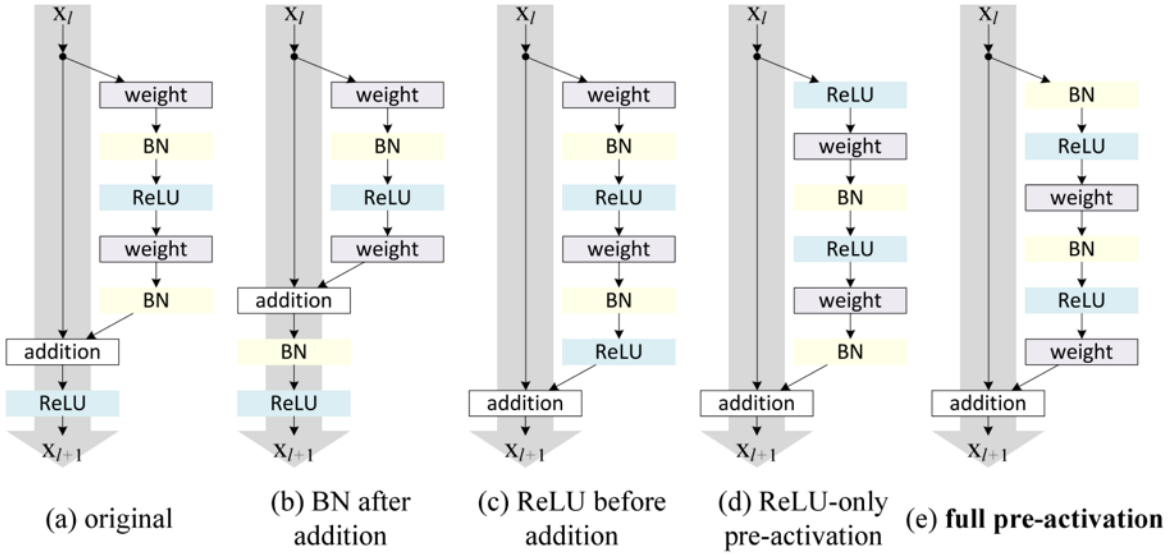
Skip connection - ResNet

```
class ResidualUnit(keras.layers.Layer):
    def __init__(self, filters, strides=1, activation="relu", **kwargs):
        super().__init__(**kwargs)
        self.activation = keras.activations.get(activation)
        self.main_layers = [
            DefaultConv2D(filters, strides=strides),
            keras.layers.BatchNormalization(),
            self.activation,
            DefaultConv2D(filters),
            keras.layers.BatchNormalization()]
        self.skip_layers = []
        if strides > 1:
            self.skip_layers = [
                DefaultConv2D(filters, kernel_size=1, strides=strides),
                keras.layers.BatchNormalization()]

    def call(self, inputs):
        Z = inputs
        for layer in self.main_layers:
            Z = layer(Z)
        skip_Z = inputs
        for layer in self.skip_layers:
            skip_Z = layer(skip_Z)
        return self.activation(Z + skip_Z)
```



ResNet



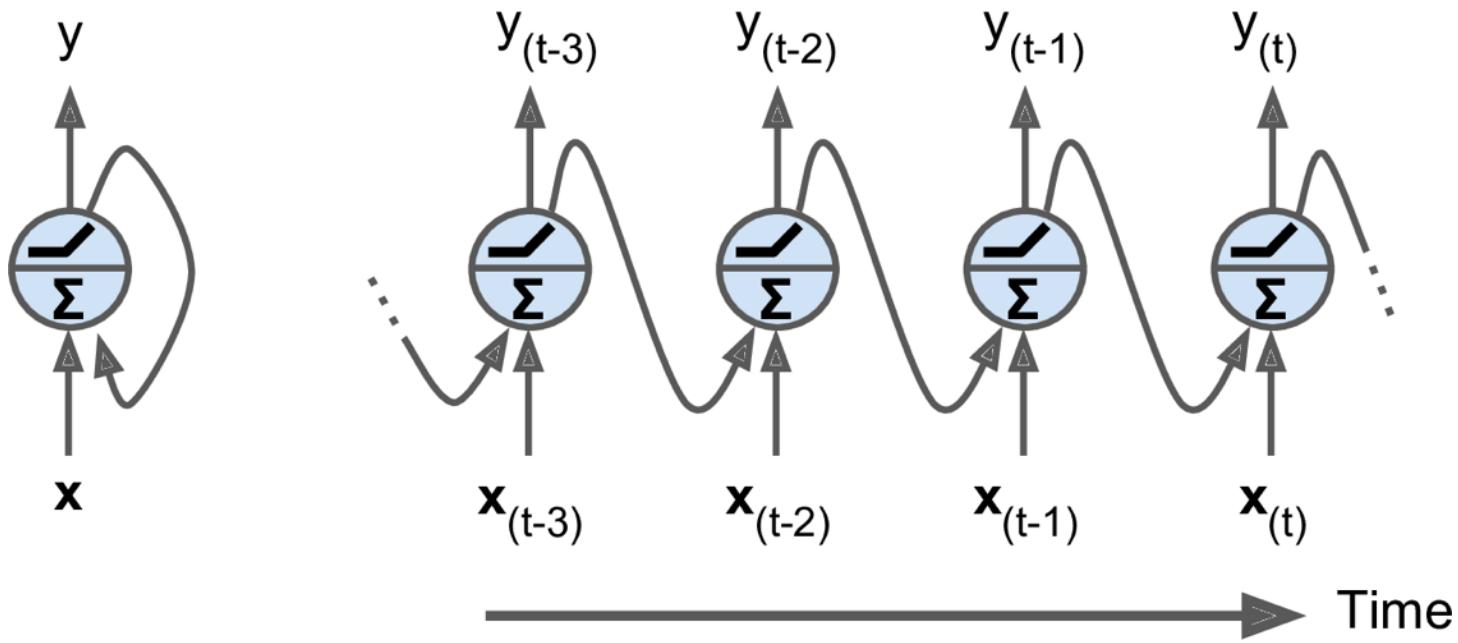


https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

Module 11 – Section 4

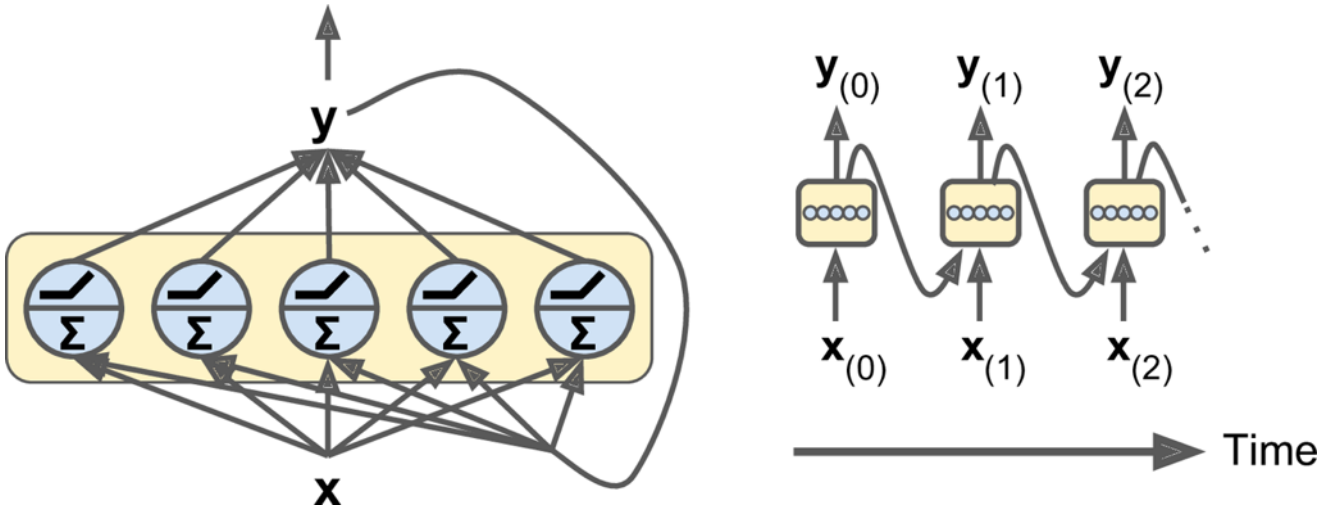
Recurrent Neural Networks

Recurrent Neural Networks (RNN)



- RNN are particular suited to work with sequence data
- They can work on *sequences* of arbitrary lengths
- A recurrent neural network looks very much like a feedforward neural network, except it also has connections pointing backward

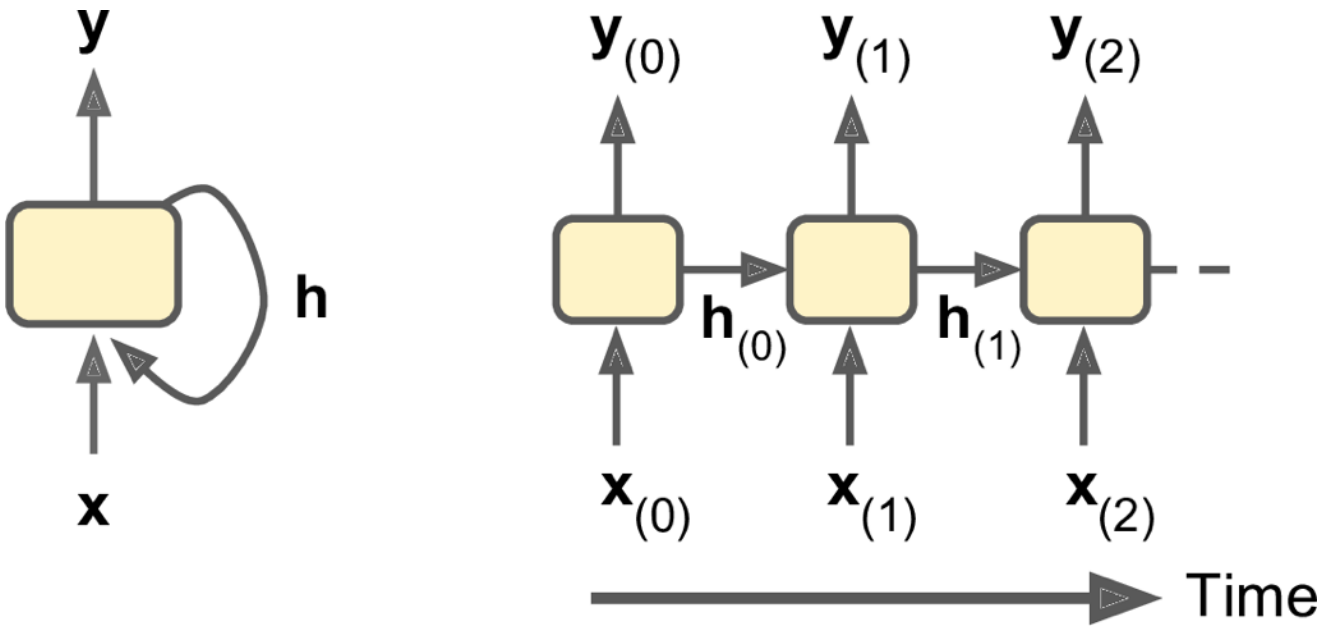
Recurrent Neural Networks (RNN)



- Each recurrent neuron has two sets of weights: one for the inputs $\mathbf{x}_{(t)}$ and the other for the outputs of the previous time step, $\mathbf{y}_{(t-1)}$. Let's call these weight vectors $\mathbf{w}_x$ and $\mathbf{w}_y$.

$$\mathbf{y}_{(t)} = \phi\left(\mathbf{W}_x^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_y^T \cdot \mathbf{y}_{(t-1)} + \mathbf{b}\right)$$

Memory Cells



- Since the output of a recurrent neuron at time step *t* is a function of all the inputs from previous time steps, you could say it has a form of *memory*.
- $\mathbf{h}_{(t)} = f(\mathbf{h}_{(t-1)}, \mathbf{x}_{(t)})$.

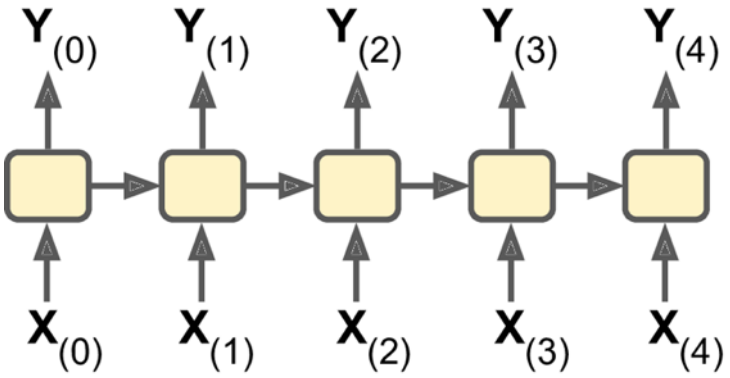
Input and Output Sequences

Many RNN variations are possible:

1. Sequence of inputs to sequence of outputs
   - E.g. predicting time series such as stock prices (forecasting)
2. Sequence of inputs to vector output
   - E.g. words corresponding to a movie review, and output a sentiment score
3. Single vector input and sequence of outputs
   - E.g. input an image and output a caption for that image
4. Sequence to vector *encoder* followed by a vector to sequence *decoder*
   - E.g. translating a sentence from one language to another

Input and Output Sequences

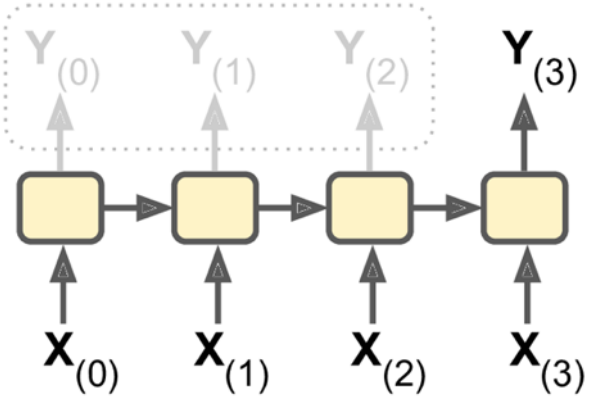
1



Weather Forecast

Ignored outputs

Review to rating



2

3

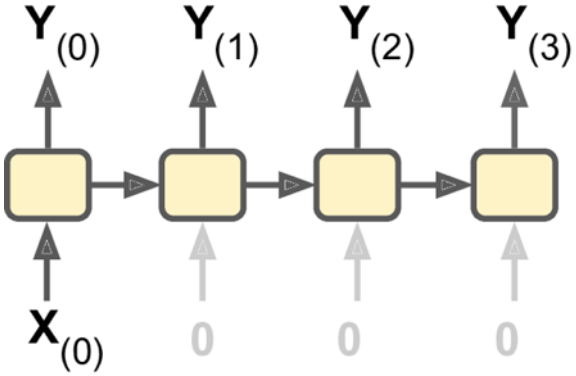
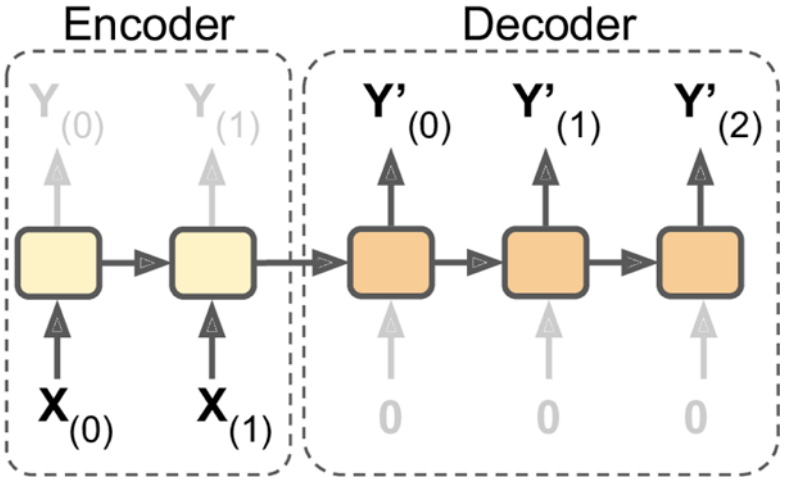


Image to Caption



Sentence Translation

4

RNN TF

- Make sure to set *return_sequences=True* for all RNN layers except the last one, otherwise the layer won't return a sequence of data over time

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(1)
])
```

- If you're curious for more: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Module 11 – Section 5

Wrap-up

Homework

- Term Project is due next week!

Next Class

- Group Presentations!

Any questions?

Join the conversation with us online:

facebook.com/uoftscs

@uoftscs

linkedin.com/company/university-of-toronto-school-of-continuing-studies

@uoftscs

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Thank You

Thank you for choosing the University of Toronto School of Continuing Studies