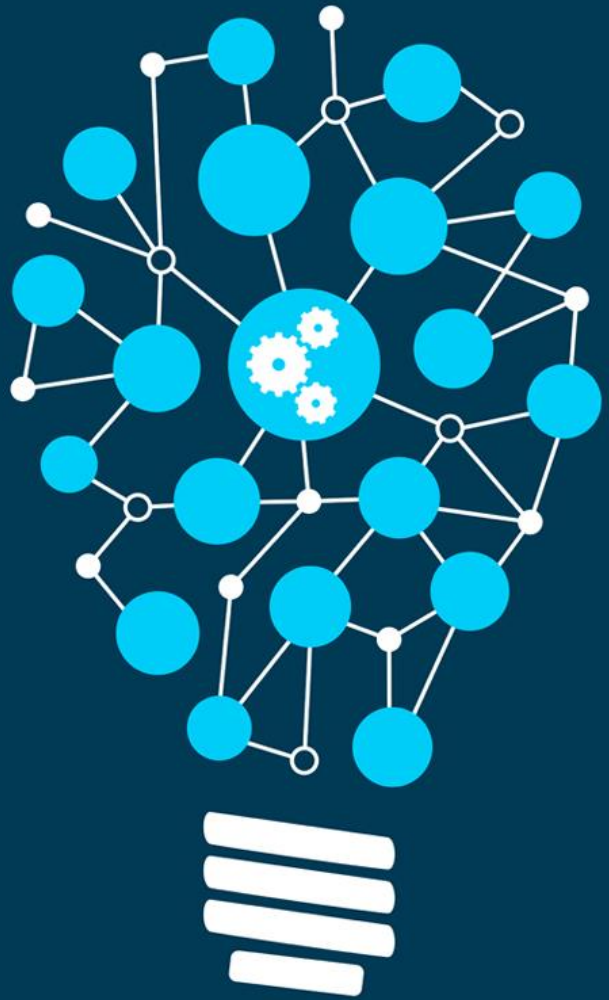
NYU

NYU Summer Machine Learning Program

Presenter Name Here
Date Here



NYU

Convolutional Neural Network

Day 7

Acknowledgement

❑ Some slides in this presentation is from Professor Sundeep Rangan's Machine Learning course

Outline

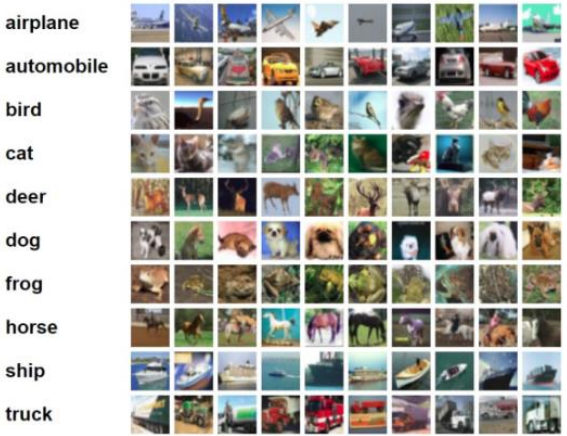
- Motivation
- 1-D Convolution
- Up/Down Sampling
- Images in Computers
- 2-D Convolution
- Pooling
- Deep Network Architecture

Recent History of Deep Neural Networks

Dataset Pre-2009

- Pre-2009, many image recognition systems worked on relatively small datasets
- MNIST:
  - 10 classes, 70,000 examples, 28 x 28 images
- CIFAR 10 (right)
  - 10 classes, 60000 examples, 32x32 color
- CIFAR 100:
  - 100 classes, 600000 examples, 32x32 color
- PASCAL VOC:
  - 20 classes, 11530 examples, variable size images
- Performance saturated
  - Difficult to make significant advancements

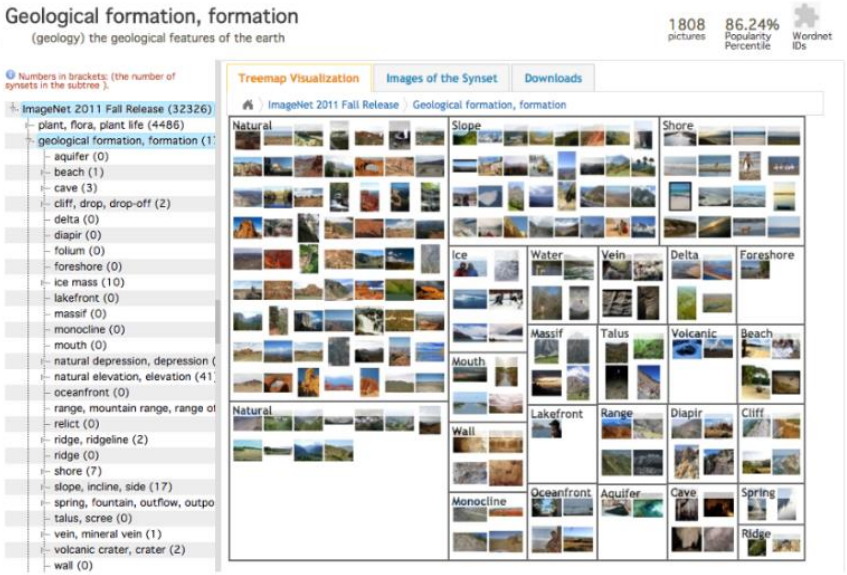
https://www.cs.toronto.edu/~kriz/cifar.html



ImageNet (2009)

- *Better algorithms need better data*

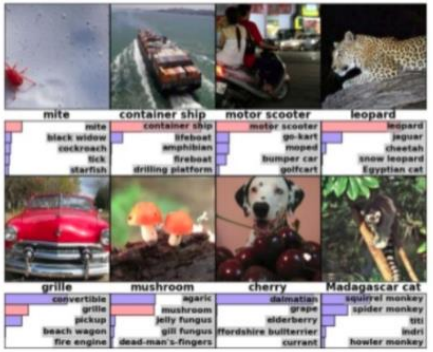
- ImageNet:
  - Goal: “map out the entire world of objects” (see this great article)
  - 3.2 million images
  - Annotated by Amazon mechanical turk
  - Hierarchical categories
  - Details in 2009 CVPR paper



Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (pp. 248-255). IEEE.

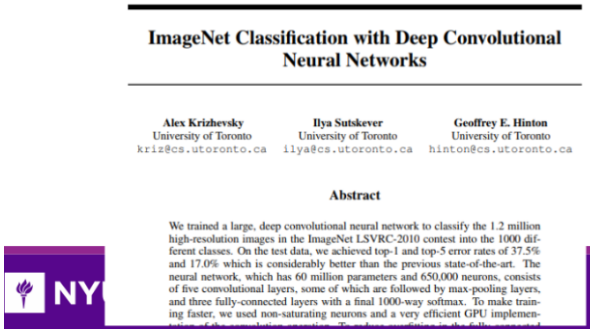
ILSVRC

- ImageNet Large-Scale Visual Recognition Challenge held yearly 2010-2017
- Challenging!
  - Classification: 1000 classes, label 5 objects per image
  - Fine-grained classification: 120 dog categories
  - Objects in many positions, scales, rotations, lightings, occlusions . . .
- Phil...



Deep Networks Enter in 2012

- 2012: Stunning breakthrough by the first deep network
- "AlexNet" from Hinton Group at U Toronto
- Easily won ILSVRC competition
  - Top-5 error rate: 15.3%
  - Second place: 25.6%
- Soon, all competitive methods are deep networks

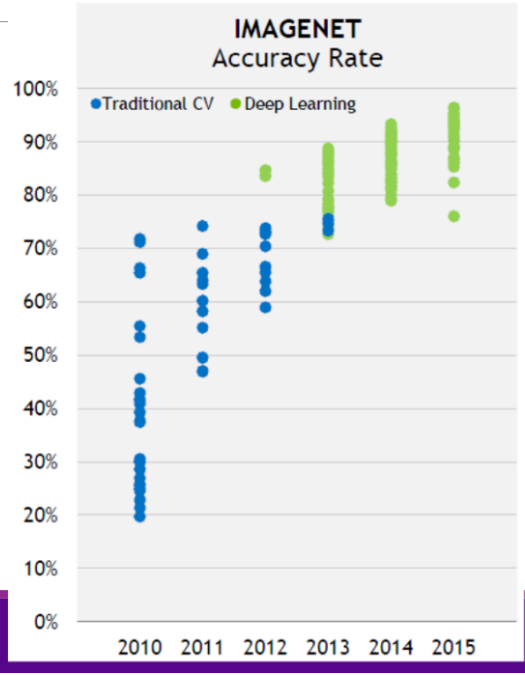


ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky University of Toronto kriz@cs.utoronto.ca
Ilya Sutskever University of Toronto ilya@cs.utoronto.ca
Geoffrey E. Hinton University of Toronto hinton@cs.utoronto.ca

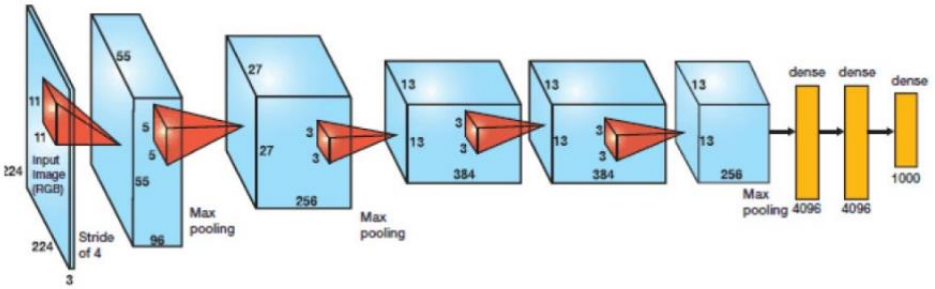
Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implemen-

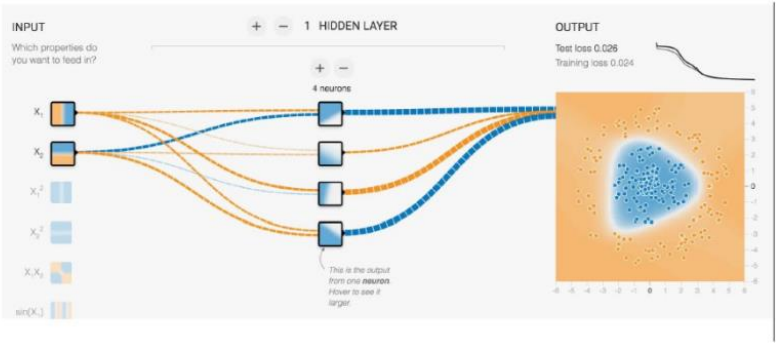


Alex Net

- Key idea: Build a very deep neural network
- 60 million parameters, 650000 neurons
- 5 conv layers + 3 FC layers
- Final is 1000-way softmax



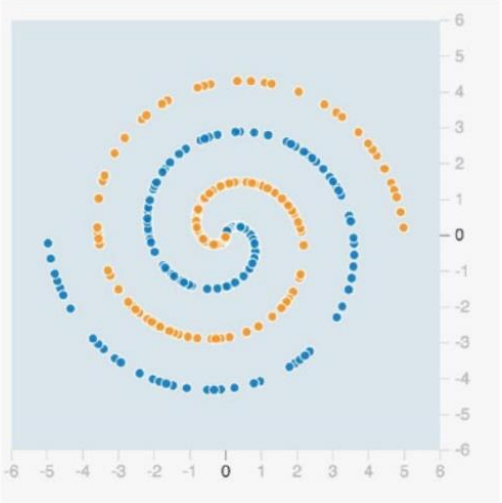
Review of Neural Networks: Output Layer



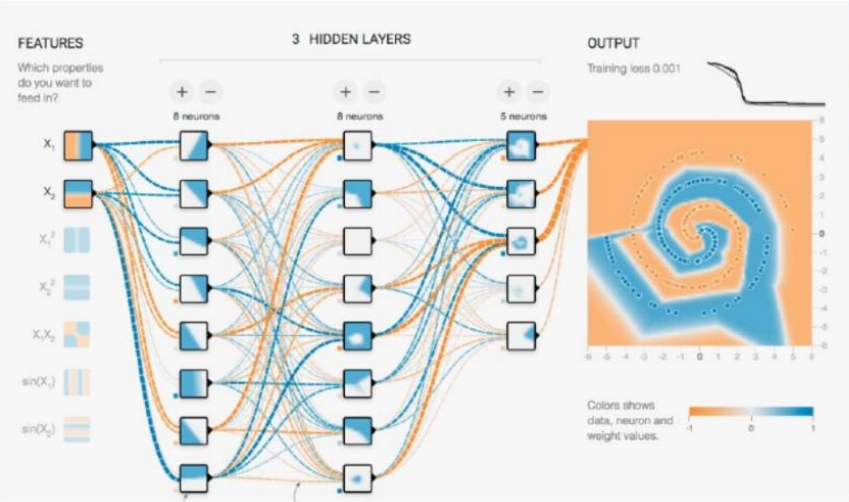
- Two-layer neural networks:
  - Implement nonlinear boundaries
  - Built from intersections of linear regions
- Picture to left:
  - Output of Tensorboard
  - Tool in TensorFlow
  - Provided for visualizing neural nets

From Kaz Sato, "Google Cloud Platform Empowers TensorFlow and Machine Learning"

What about a More Complicated Region?



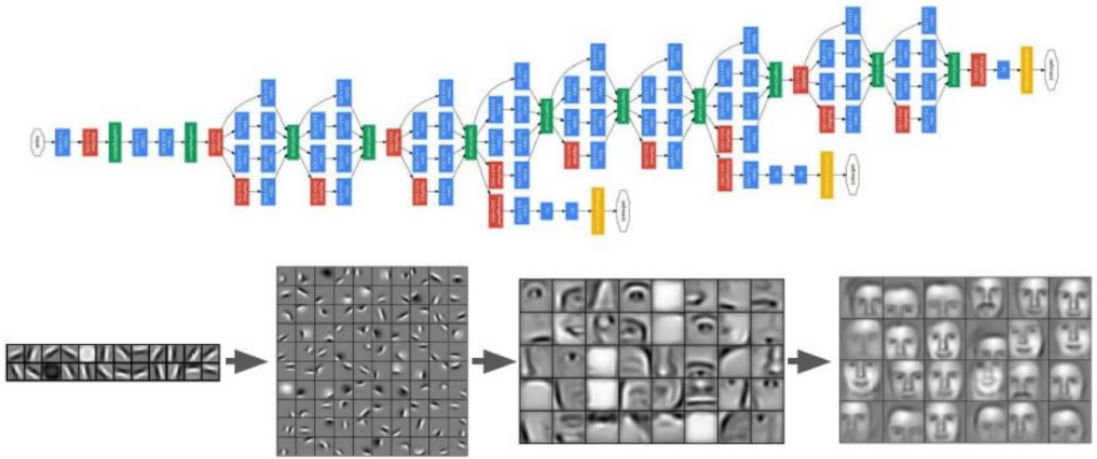
Use Multiple Layers



- More hidden layers
- Hierarchies of features
- Generate very complex shapes

Can you Classify This?

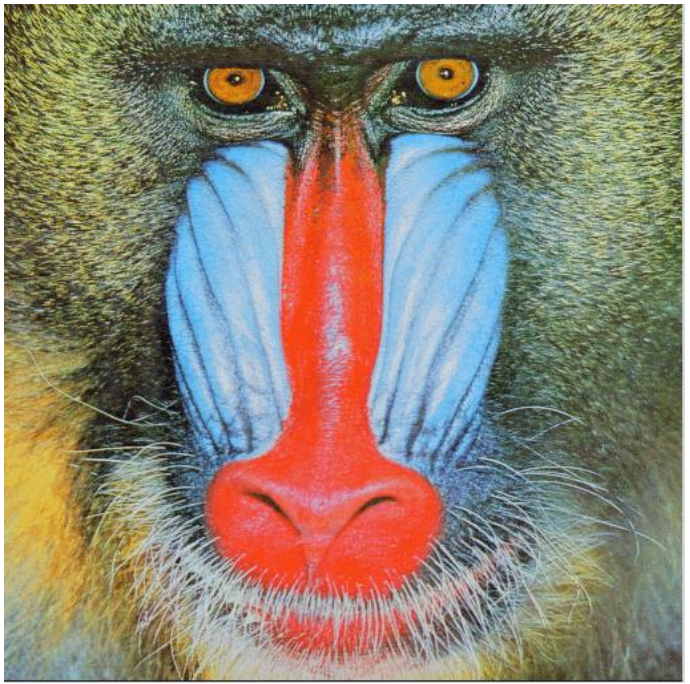
Build a Deep Neural Network



From: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations, Honglak Lee et al.

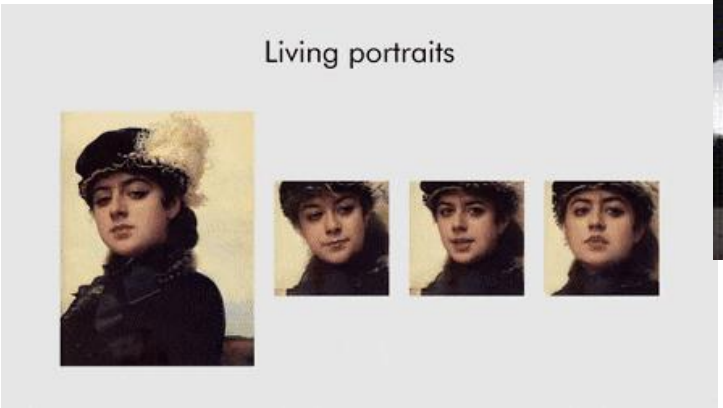
CNN Motivation: Locality & Feature Extraction

- Things closer together tend to be more related (red)
  - Feature extraction: A face has eyes, mouth, nose
- Fully connected NN layer considers all areas equally related (green)
  - Must we consider whiskers and eyes together?
- Maybe we can learn locality? Can we “bake it in” to the NN?

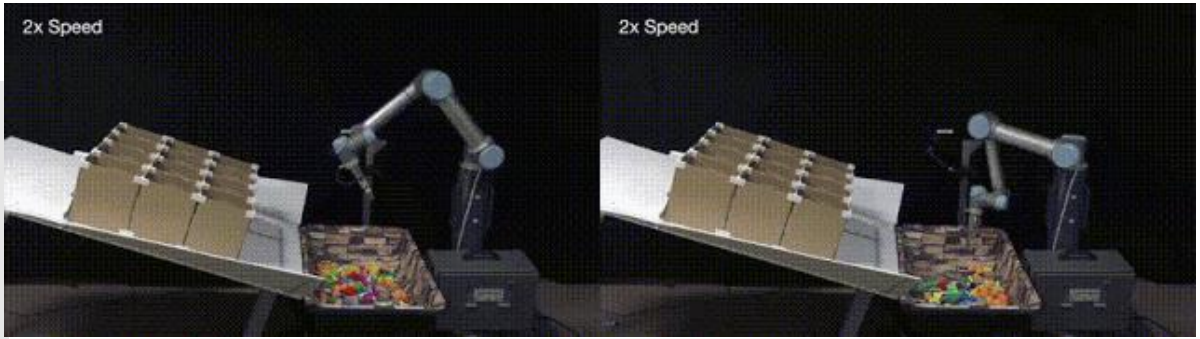


CNN Motivation: Locality & Feature Extraction

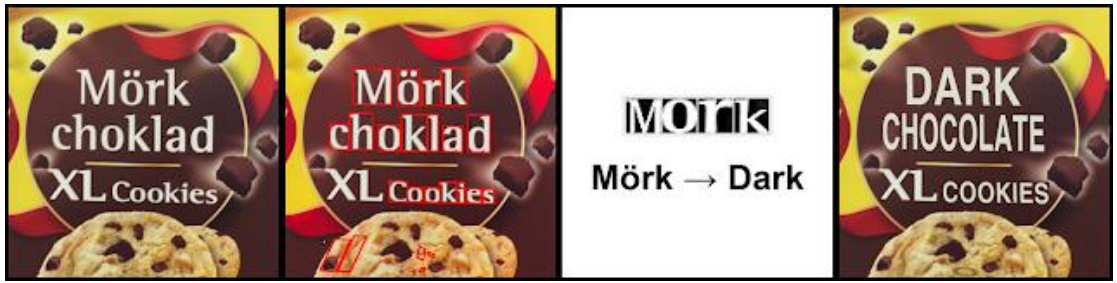
- Ted Talk: the wonderful and terrifying implications of computers that can learn
- https://ai.googleblog.com/



Few Shot Learning of Realisitic Neural Talking Head Models



Robots Learning to Toss



Optical Character Recognition and Translation

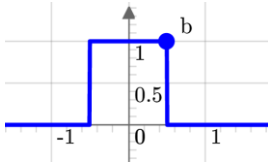
Introduction to Convolution (1-Dimensional)

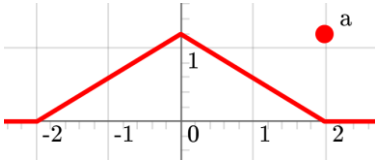
- What is convolution?
  - An operation that computes the amount of “overlap” of a kernel g as it shifts over another function f
  - Mathematical definition:
    - $(f * g)(n) = \sum_{k=-\infty}^{\infty} g[k-n]f[k]$
    - Interpretation: $g[k]$ is the kernel, $g[k-n]$ slides the function to position n

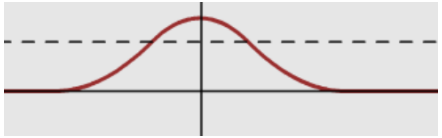
      $g[k-n]$ is then multiplied with $f[k]$ and we sum over all k to get the output for position n

Convolution Illustrated

- $g[k]$ is a rectangular function, $f[k]$ is a triangular function

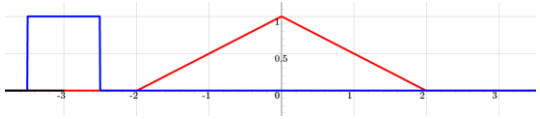




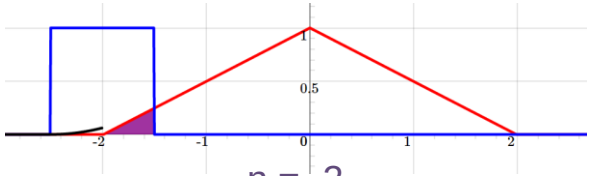


Output

- Positions of f and g for different n



n = -3



n = -2

n = -0

n = 1

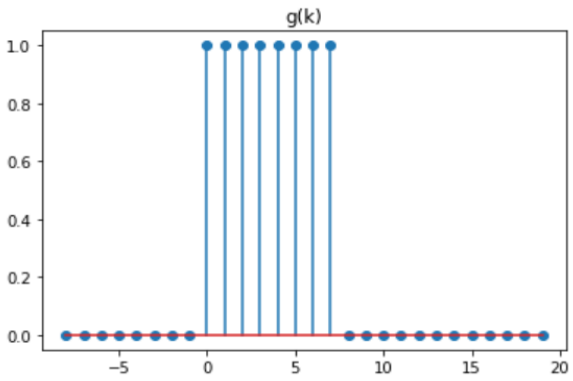
n = 2

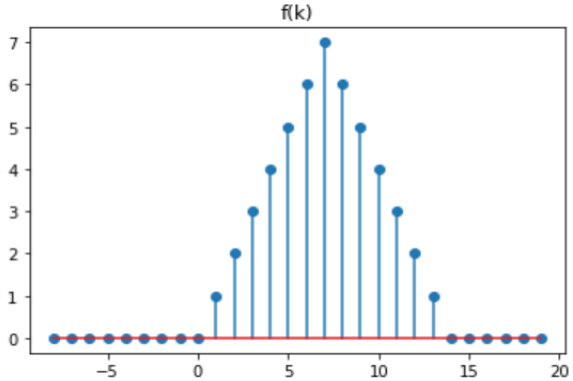
n = 3

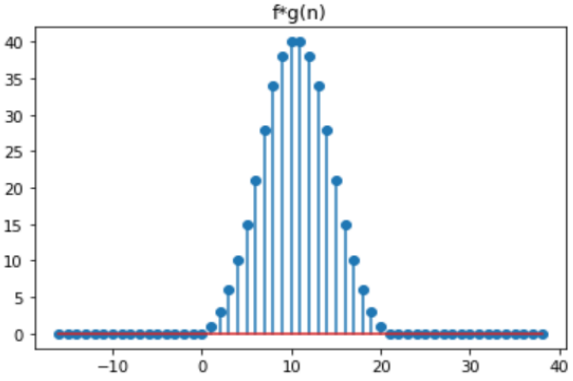
Demo: https://phiresky.github.io/convolution-demo/

Convolution in Discrete Time

- $g[k]$ is a rectangular function, $f[k]$ is a triangular function

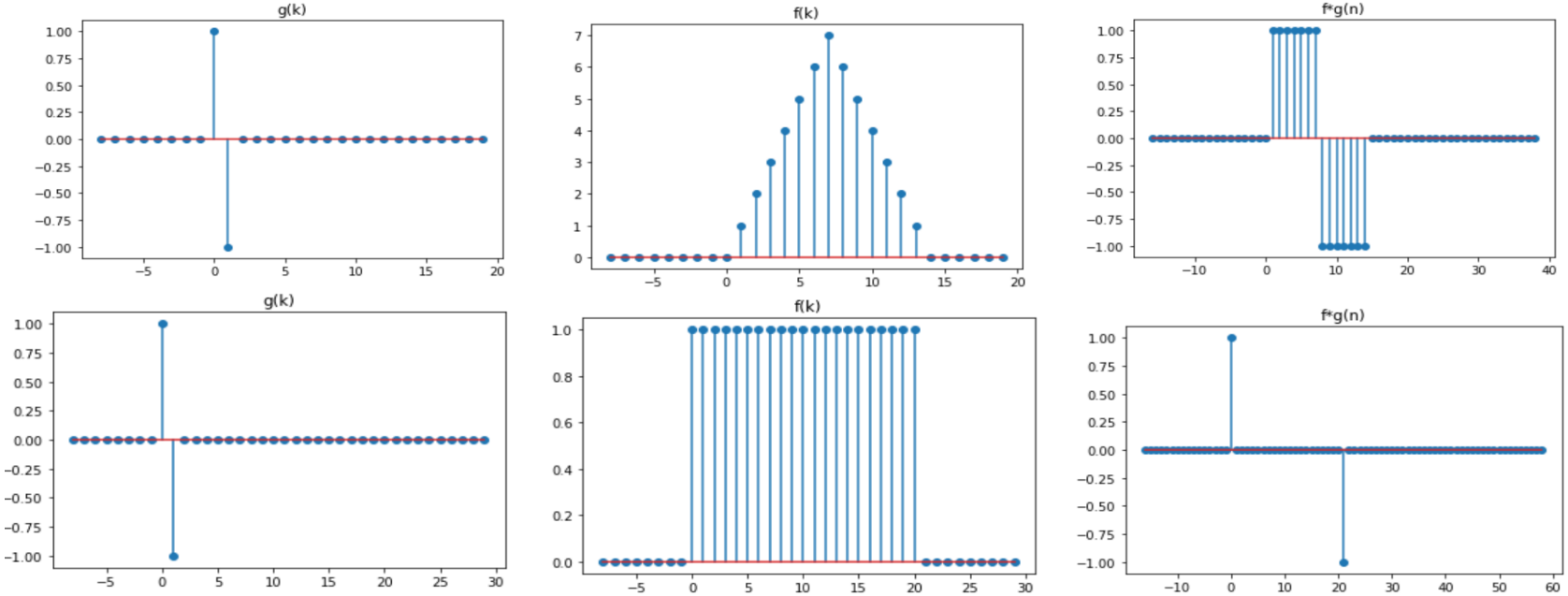






- Can you tell what does convolution with the rectangular function do to the triangular signal?

Convolution in Discrete Time

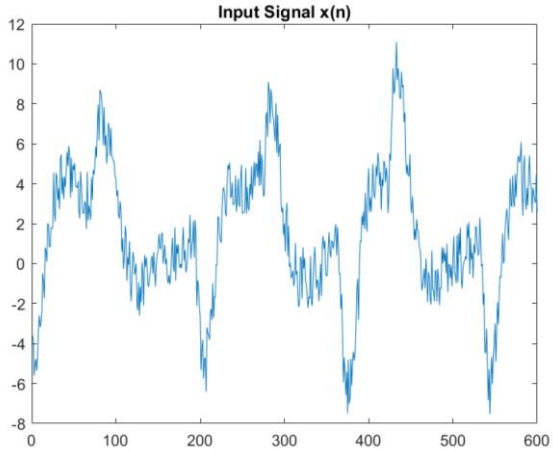


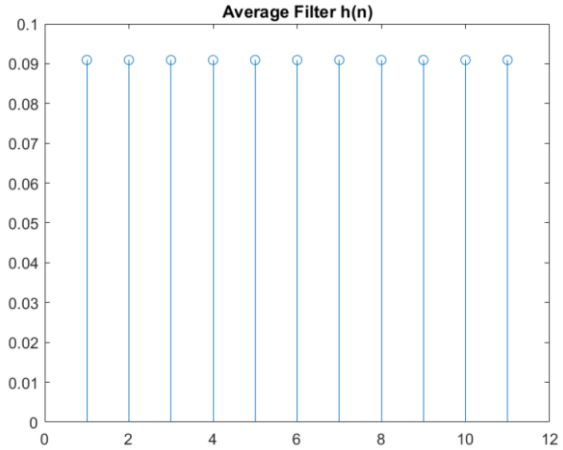
- Can you tell what is the affect of this filter?

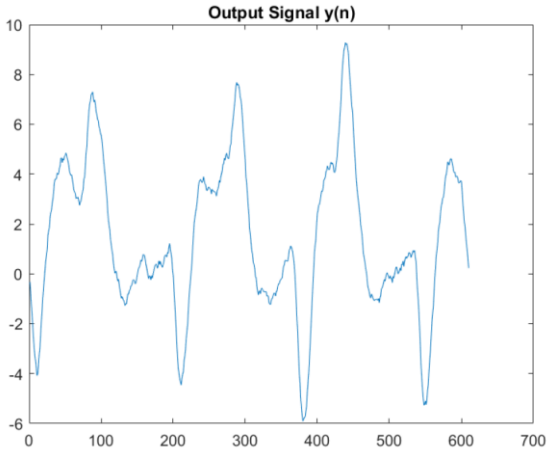
Why Convolution?

- Convolution gives an output that is a weighted sum of several neighboring positions in the input, allowing us to learn local relation
- Only a small number of weights need to be learned, as oppose to learning one weight for each position of a long series
- This will be useful for processing data like images or audio, where there is a huge number of data points
- Different kernels will have different effects on the input: for example, smoothing, sharpening, edge detection, feature detection

Example: Convolution of a Noisy ECG signal





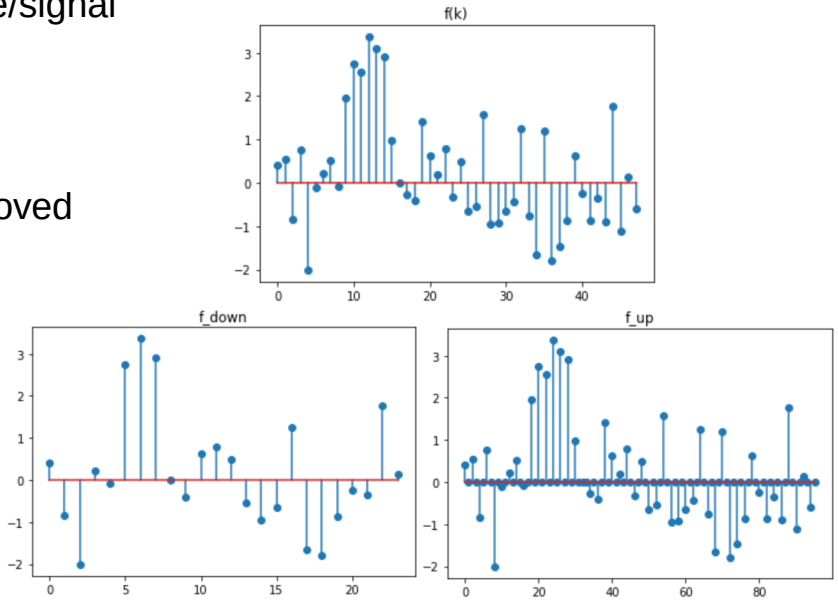


- Convolution with a simple averaging filter smooths the signal and removes the noise

Down sampling and Up sampling

- Data such as image and audio can come in very large size when they have high resolution
- It is not necessary to have fine details when you want to detect features that are large in size
- Down sampling works by removing points in an image/signal
- Down sampling by a factor of 2:
- $f_{down}(n) = f(2n)$
  - Take every other two data point, the rest are removed
- Up sampling: Fill in zeros to increase the size
- Can interpolate to fill in the zeros with values

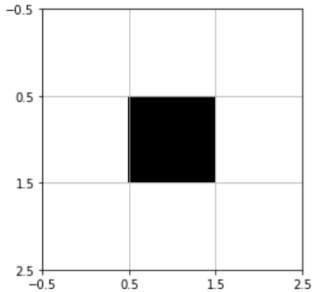
- $f_{up}(n) = \begin{cases} f\left(\frac{n}{2}\right), & n = even \\ 0, & n = odd \end{cases}$



Images in Computer

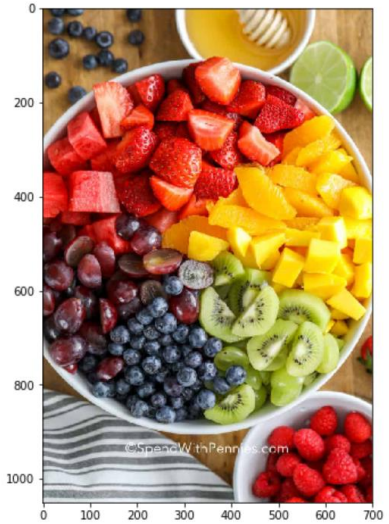
- Images are stored as arrays of quantized numbers in computers
- Gray scale image: 2D matrices with each entry specifying the intensity (brightness) of a pixel
  - Pixel values range from 0 to 255, 0 being the darkest, 255 being the brightest

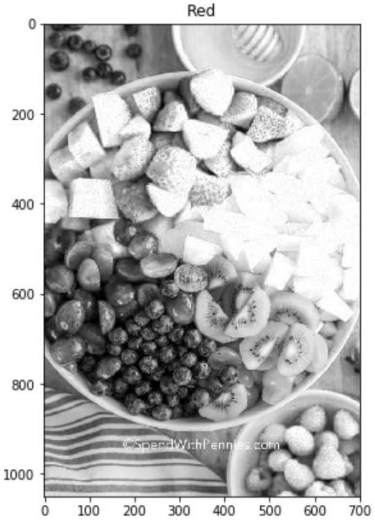
```
[[255 255 255]
 [255   0 255]
 [255 255 255]]
```

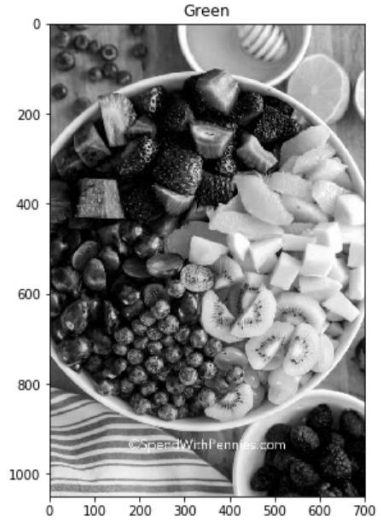


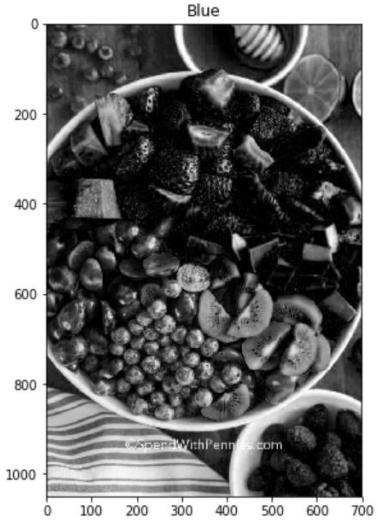
- Color image: 3D array, 2 dimensions for space, 1 dimension for color
  - Can be thought of as three 2D matrices stacked together into a cube, each 2D matrix specify the R,G,B value at each pixel

Color Images







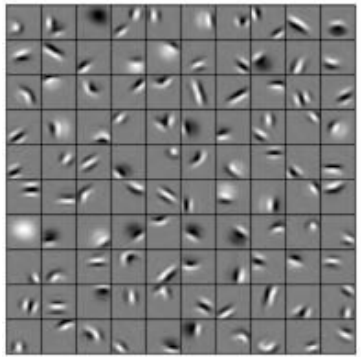
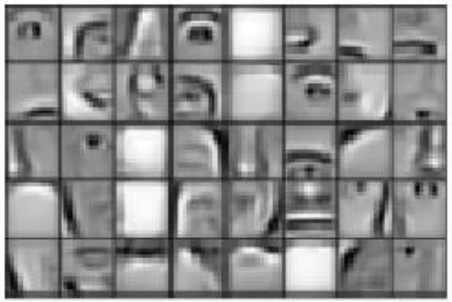
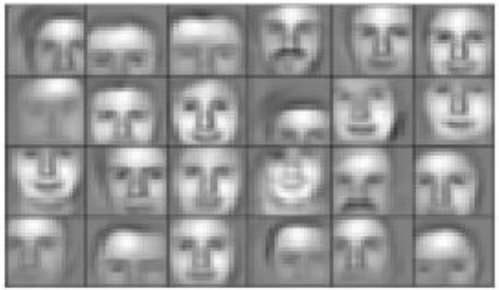


Shape of the image array: (1050,700,3)

-> There are 1050x700 pixels, 3 channels: R,G,B

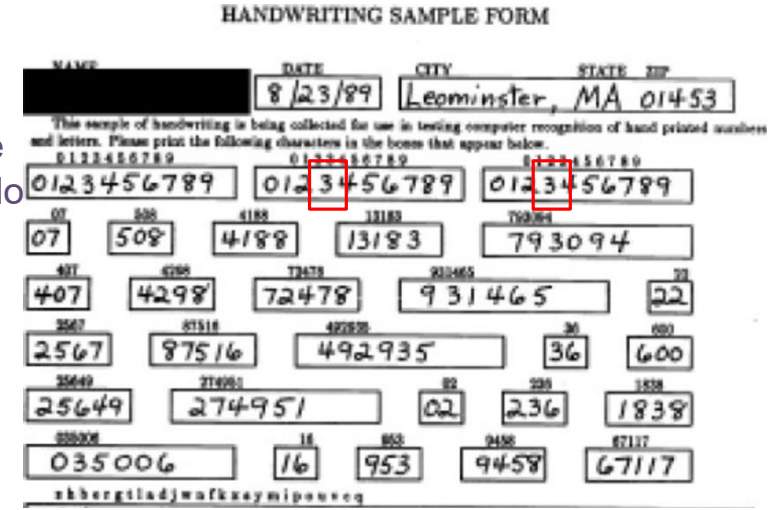
2D Convolutions: Local Features

- As we observed from the 1D examples, the purpose of performing convolution is to extract/highlight useful features/information from the input.
- The initial layers in a Deep Neural Network are often used to detect local features
- These layers detect small patterns in larger images examples: small lines, curves, edges
- The subsequent layers combine local features to create more complex features

How do we find local features?

- How do we find local features?
- Let's revisit the handwritten numbers problem. If we have to find the digit "3" in the form, how would you do it?



Localization via Correlation

- Correlation measures the similarity between two elements. We can find the local feature (in our case 4) by sliding a window over the data and finding the highest correlated output
- Say our image is of the size $N_1 \times N_2$ (e.g. 512 x 512)
- A small filter of the size $K_1 \times K_2$ (e.g. 8 x 8)

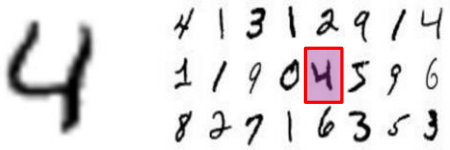
At each offset (j_1, j_2) compute:

$$Z[j_1, j_2] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} W[k_1, k_2] X[j_1 + k_1, j_2 + k_2]$$

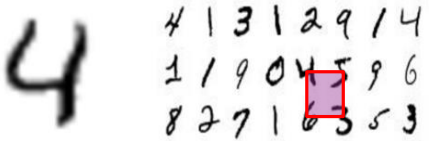
Correlation of W with image box starting at (j_1, j_2)

$Z[j_1, j_2]$ is large if feature is present around (j_1, j_2)

Filter Image $Z[j_1, j_2]$



High



Low

Terminology used for convolution

- In signal processing and mathematics, convolution includes flipping:

$$z[n_1, n_2] = \sum_{k_2=0}^{K_2-1} \sum_{k_1=0}^{K_2-1} w[k_1, k_2] x[n_1 - k_1, n_2 - k_2]$$

  - Implemented in python using scipy.signal.convolve2d
  - However, we will call this convolution with reversal (for convenience)

- But, in many neural network packages (including Keras), convolution does not include flipping:

$$z[n_1, n_2] = \sum_{k_2=0}^{K_2-1} \sum_{k_1=0}^{K_2-1} w[k_1, k_2] x[n_1 + k_1, n_2 + k_2]$$

  - In signal processing, this is called correlation.
  - Implemented in python using scipy.signal.correlate2d
  - In Deep learning, we will call this convolution.

Boundary Conditions

- Suppose inputs are:

x, size $N_1 \times N_2$, w: size $K_1 \times K_2$, $K_1 \le N_1, K_2 \le N_2$

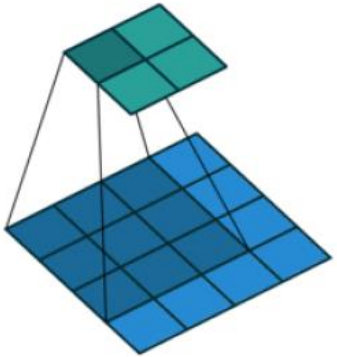
$z = x * w$ (without reversal)

$$z[n_1, n_2] = \sum_{k_2=0}^{K_2-1} \sum_{k_1=0}^{K_2-1} w[k_1, k_2] x[n_1 + k_1, n_2 + k_2]$$

- The following are different ways to define outputs:
- Valid mode: $0 \le n_1 < N_1 - K_1 + 1,\ 0 \le n_2 < N_2 - K_2 + 1$
  - Requires no zero padding
- Same mode: Output size $N_1 \times N_2$
  - Usually use zero padding for neural networks
- Full mode: Output size $(N_1+K_1 - 1) \times (N_2+K_2 - 1)$
  - Not used often in neural networks

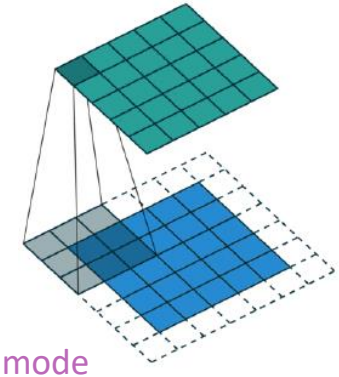
Boundary Conditions Illustrated

- As we discussed above, here are three ways to handle boundary conditions
- See excellent github with animated gifs
-



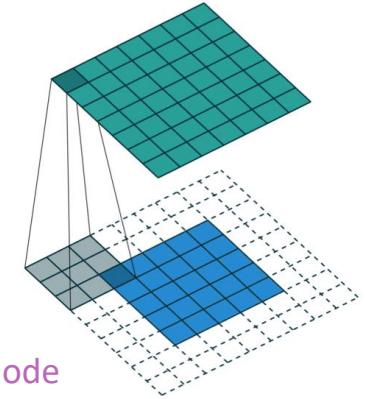
Valid mode
(no zero padding)

Output shape:
$(N_1 - K_1 + 1) \times (N_2 - K_2 + 1)$



Same mode
(half zero padding)

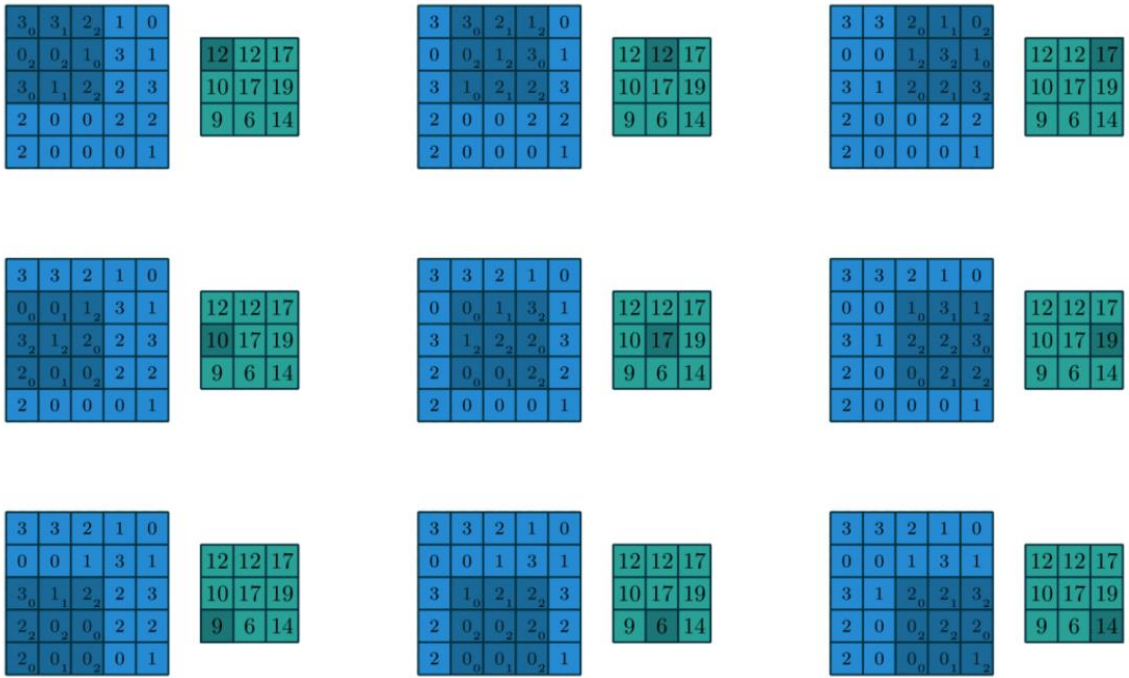
Output shape:
$N_1 \times N_2$



Full mode
(full zero padding)

Output shape:
$(N_1 + K_1 + 1) \times (N_2 + K_2 + 1)$

2D Convolution Example with Valid Mode



Kernel

$$W = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

Convolution in Python

- Our demo illustrates 2D convolution in Python.
- We use the following packages:
  - skimage: For image processing
  - scipy signal: For signal processing
- We use the “cameraman” image, which is a built-in image (skimage.data).
- We perform two convolutions:
  - Local averaging / blurring
  - Edge detection

```
im = skimage.data.camera()
disp_image(im)
```

Convolution for Local Averaging

- First, consider convolving with a uniform kernel
- Each output is a local average of the input
- Visually, this blurs the image
- Amount of blurring increases with kernel size

$$\boldsymbol{G} = \frac{1}{K_x K_y}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \begin{matrix} \updownarrow K_y \\ \\ \leftrightarrow \\ K_x \end{matrix}$$

```
kx = 9
ky = 9
sig = 3
G_unif = np.ones((kx,ky))/(kx*ky)
im_unif_full = scipy.signal.correlate2d(im, G_unif, mode='full')
im_unif_same = scipy.signal.correlate2d(im, G_unif, mode='same')
im_unif_valid = scipy.signal.correlate2d(im, G_unif, mode='valid')
```

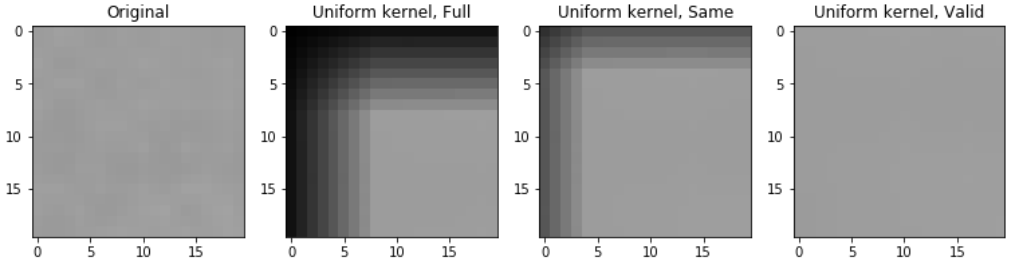
Original

Uniform kernel, 9x9

Uniform kernel, 15x15

Illustration of Boundary Conditions


```
kx = 9
ky = 9
sig = 3
G_unif = np.ones((kx,ky))/(kx*ky)
im_unif_full = scipy.signal.correlate2d(im, G_unif, mode='full')
im_unif_same = scipy.signal.correlate2d(im, G_unif, mode='same')
im_unif_valid = scipy.signal.correlate2d(im, G_unif, mode='valid')
```




```
print("Input shape = " + str(im.shape))
print("Output shape (Full) = " + str(im_unif_full.shape))
print("Output shape (Same) = " + str(im_unif_same.shape))
print("Output shape (valid) = " + str(im_unif_valid.shape))
```

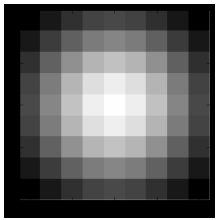
```
Input shape = (512, 512)
Output shape (Full) = (520, 520)
Output shape (Same) = (512, 512)
Output shape (valid) = (504, 504)
```

- Notice the black pixels at the boundaries when we use same or full mode, these arise from zero padding

Averaging vs. Gaussian Filtering

- We illustrated the output using a uniform kernel
- Now we try a Gaussian kernel.
- We can see that both kernels blur the images, but differently.

9x9 Gaussian blur kernel



Convolution for Edge Detection

- We can do edge detection by convolving with a gradient filter

  For example, use Sobel filters:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \qquad G_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

  Define $Z_x = G_x * X$, $Z_y = G_y * X$ (without reversal)

- Called gradient filters since:

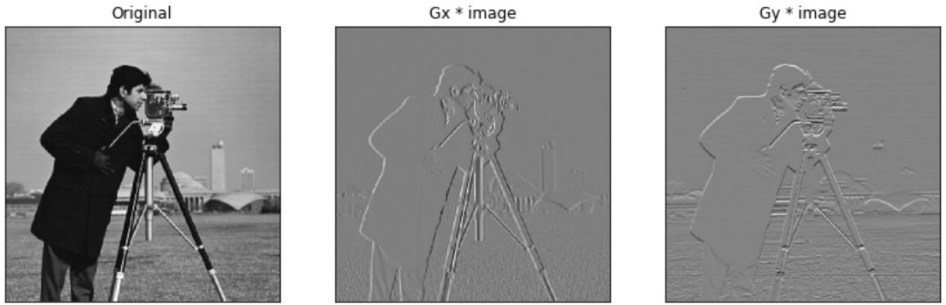
  $Z_x[i,j] = Z_y[i,j] = 0$ in areas where $X[i,j]$ is constant

  $Z_x[i,j]$ = large positive on strong decrease in x-direction = vertical edge white → black

  $Z_x[i,j]$ = large negative on strong increase in x-direction = vertical edge black → white

  $Z_y[i,j]$ is similarly sensitive to horizontal edges

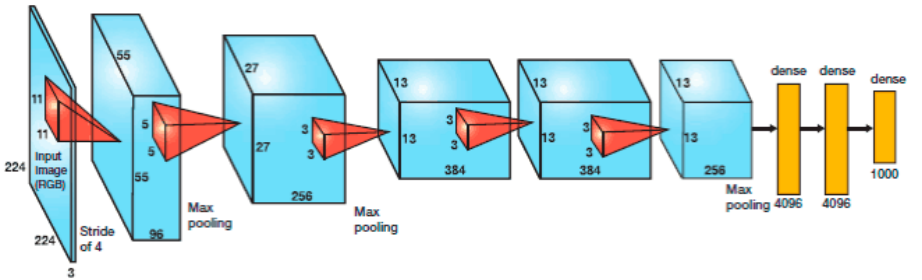
Edge Detection using Sobel Filter




```
Gx = np.array([[1,0,-1],[2,0,-2],[1,0,-1]])  # Gradient operator in the x-direction
Gy = np.array([[1,2,1],[0,0,0],[-1,-2,-1]])  # Gradient operator in the y-direction
```

```
# Perform the convolutions
imx = scipy.signal.correlate2d(im, Gx, mode='valid')
imy = scipy.signal.correlate2d(im, Gy, mode='valid')
```

Classic CNN Structure



Convolutional layers

2D convolution with Activation and pooling / sub-sampling

Fully connected layers

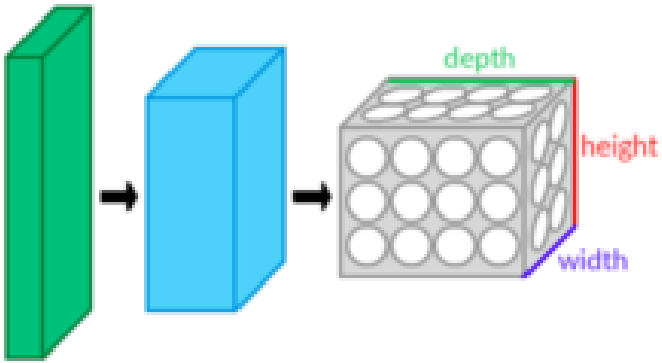
Matrix multiplication & activation

- ❑ Starts with convolutional layers.
- ❑Each layer does:
  - ❑ 2D convolution with several kernels
  - ❑ Activation (e.g., ReLU)
  - ❑ Sub-sampling or pooling
- ❑ Finish with fully connected (or dense) layers.
- ❑Each layer does:
  - ❑Matrix multiplication
  - ❑Activation

Tensors

- Input and output of each layer is a tensor
  - A multidimensional array
- Examples of tensors
  - Grayscale image: *(Height, Width)*
  - Color image: *(Height, Width, Chan)*

    Chan: {Red, Blue, Green}
  - Batch of images: *(Sample, Height, Width, Chan)*
- Example: A batch of 100 color images with pixels has shape:
- The number of dimensions is called the order or rank
  - Note that rank has a different meaning in linear algebra. Hence, we will use ‘order’ for convenience.



What Do Convolutional Layers Do?

- Each convolutional layer has:
  - Weight tensor: W siz $(K_1, K_2, N_{in}, N_{out})$
  - Bias vector: b siz N_{out}
- Takes input tensor U creates output tensor
- Convolutions performed over space and added over channels

$$Z[i_1, i_2, m] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} \sum_{n=0}^{N_{in}-1} W[k_1, k_2, n, m] X[i_1 + k_1, i_2 + k_2, n] + b[m]$$

- For each output channel m, input channel n
  - Computes 2D convolution with W[:,:,n,m] (2D filters of size K1 X K2)
  - Sums results over n
  - Different 2D filter for each input channel and output channel pair

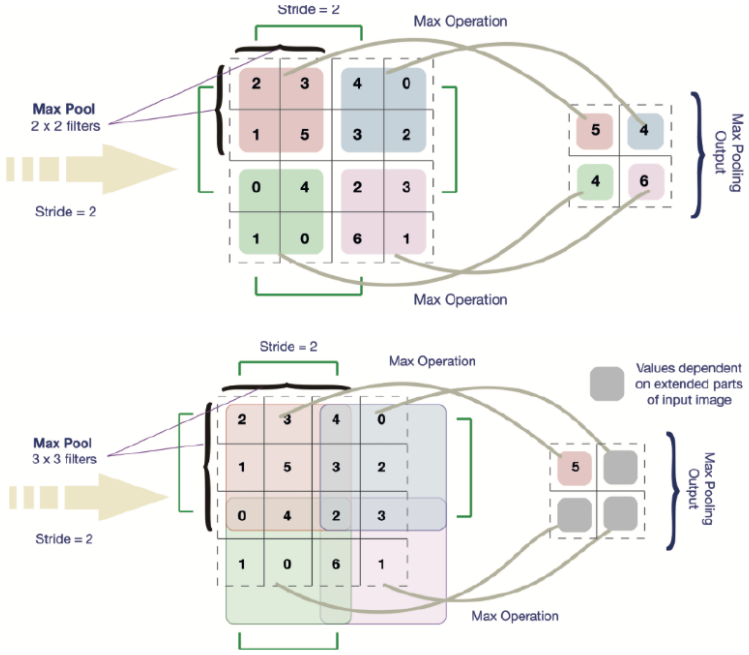
Subsampling and Pooling

- After convolution and activation, there is often a data-reduction stage
- There are many options here. Some popular ones are:
- Subsampling:
  - keep the top-left pixel from every S X S region, which is called the stride
  - Implemented as part of convolution (has no wasted computations!)
  - Also referred to as “downsampling” in signal processing
- Max pooling:
  - Keep the largest value in each K X K region
  - Then shift the region by stride S horizontally & vertically
- Average pooling:
  - Keep the largest value in each K X K region
  - Shift the region by stride S horizontally & vertically
  - Called ”decimation” in signal processing
- The above is performed independently on every channel and batch item

Max Pooling Illustrated

An example Image Portion
for Max Pooling
Numbers represent
the pixel values

2	3	4	0
1	5	3	2
0	4	2	3
1	0	6	1



What Dense Layers Do?

- Say that the last convolutional layer produced (after pooling) a tensor of shape (B, N_1, N_2, C)
- Just before the first dense layer, we flatten (i.e., reshape) into matrix **U**
  - Shape is $(B, N_{in}),\ N_{in} = N_1 N_2 C$
- Then output is performed with matrix multiplication:

$$Z[i,k] = \sum_{j=1}^{N_{in}} W[j,k]\, U[i,j] + b[k], \qquad k = 0, \ldots, N_{out}$$

  - Weights W: shape (N_{in}, N_{out})
  - Bias b: Shape $(N_{out},)$

- Same as the linear stages of the 2-layer neural network from the last unit!

Convolution vs Fully Connected

- Using convolution layers greatly reduces number of parameters
- Ex: Suppose input is $(*, N_1, N_2, N_{in})$ output is $(*, M_1, M_2, N_{out})$
  - Ex: AlexNet 2nd layer $(*, 55, 55, 96) \rightarrow (*, 55, 55, 256)$
- Convolutional network with (K_1, K_2) size filters

  Requires $K_1 K_2 N_{in} N_{out}$ weights and N_{out} biases

  Example: AlexNet 2nd layer with $K_1 = K_2 = 5$ filters has $6.1(10)^5$ weights and 25 biases

- But, a fully-connected layer with same size inputs and outputs:

  Would require $N_1 N_2 N_{in} M_1 M_2 N_{out}$ weights and $N_1 N_2 N_{out}$ biases

  Example: AlexNet 2nd layer would need $2.2(10)^{11}$ weights and $7.7(10)^5$ biases

- Convolutional layers exploit translation invariance
  - Local features are small and could be located

Creating Convolutional Layers in Keras

- Done easily with Conv2d
  - Specify input_shape (if first layer), kernel size and number of output channels

- To illustrate:
  - We create a network with a single convolutional layer
  - Set the weights and biases (normally these would be learned)
  - Run input through the layer (using the predict command)
  - Look at the output

```
# Create network
K.clear_session()
model = Sequential()
model.add(Conv2D(input_shape=input_shape,filters=nchan_out,
                 kernel_size=kernel_size,name='conv2d'))
```

Example 1: Gradients of a BW image

- Create simple convolutional layer
- Input: BW image, input channel
- Two output channels: x- and y-gradient,

$$* \; G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \qquad G_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \Rightarrow \; ?$$

Input.
One channel
Shape = (512,512,1)

Filters
Two gradient

Create a Layer in Keras

```
K.clear_session()
model = Sequential()
kernel_size = Gx.shape
nchan_out = 2
model.add(Conv2D(input_shape=input_shape,filters=nchan_out,
                 kernel_size=kernel_size,name='conv2d'))
```

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 510, 510, 2)       20
=================================================================
Total params: 20
Trainable params: 20
Non-trainable params: 0
_________________________________________________________________
```

- Create a single layer model
- Use the Conv2D layer
- Specify
  - Kernel size
  - Number of output channels
  - Input shape
- Why do we have 20 parameters?

Set the Weights

```
layer = model.get_layer('conv2d')
W, b = layer.get_weights()
print("W shape = " + str(W.shape))
print("b shape = " + str(b.shape))
```

```
W shape = (3, 3, 1, 2)
b shape = (2,)
```

```
W[:,:,0,0] = Gx
W[:,:,0,1] = Gy
b = np.zeros(nchan_out)
layer.set_weights((W,b))
```

- Read the weights and the shapes
- Set the weights to the two filters
- Normally, these would be trained
- Run the input through the network

```
x = im.reshape(batch_shape)
y = model.predict(x)
```

Perform Convolution in Keras

- Create input x
  - Need to reshape
- Use predict command to compute output
- Generates two output channels y

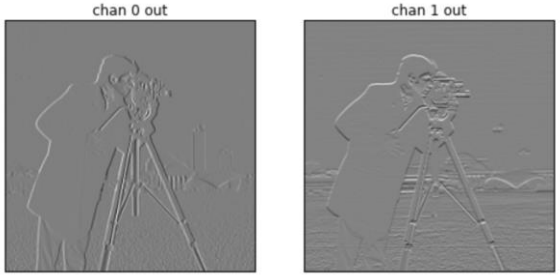
```
x = im.reshape(batch_shape)
y = model.predict(x)
```

Input x

*

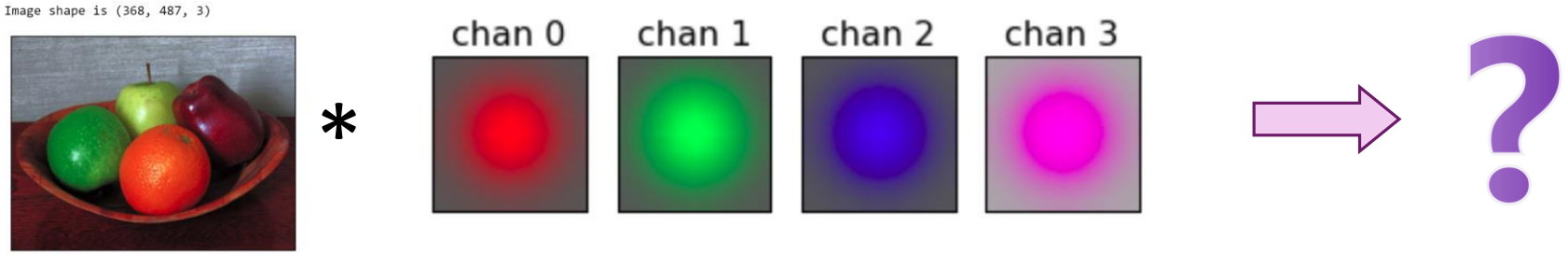
Filters
Two gradients

y[:,:,0] y[:,:,1]



Example 2: Color Input

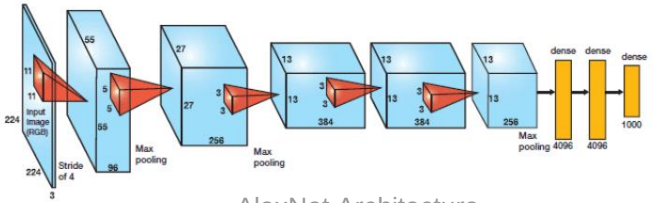
- Input: Single color input
  - input channels
  - Input size per sample = 368 x 487 x 3
- Output: Filter with four different color filters
  - Each kernel is 9 x 9
  - output channels



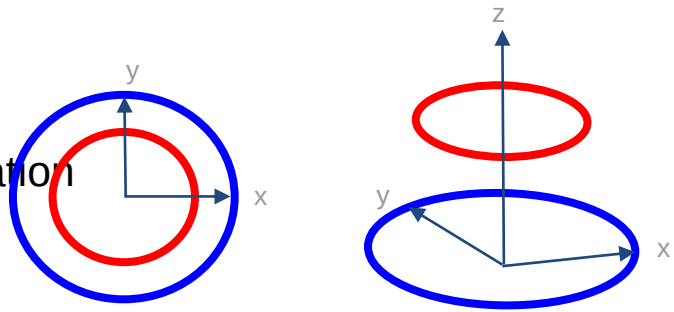
Deep Networks: Using our Tools

What we have at our disposal: Loosely Defined “Layers”

- ❑ Fully Connected Layers
  - Linear Transform with activation, $f(Wx + b)$
- ❑ Convolutional Layers
  - Conv. Kernels
  - Pooling
- ❑ Dimensionality Expansion and Reduction
  - Important way to intuit the function of layers
  - Expansion: can gain new insights into data
  - Reduction: can aggregate and combine information
- ❑ Conv2D.xlsx short demo



AlexNet Architecture



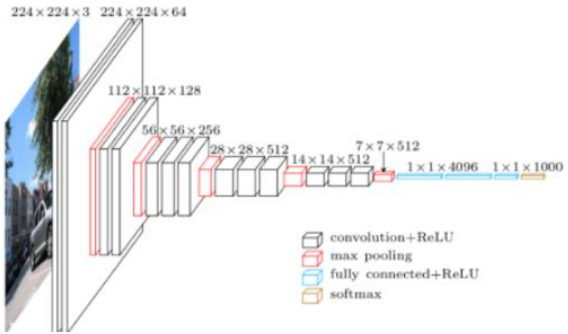
Use of Dimensionality Expansion

VGG16

- From the Visual Geometry Group
  - Oxford, UK
- Won ImageNet ILSVRC-2014
- Based on small early filters
  - But more layers
- Remains a repectable network
- Lower lower layers are often used as feature extraction layers for other tasks

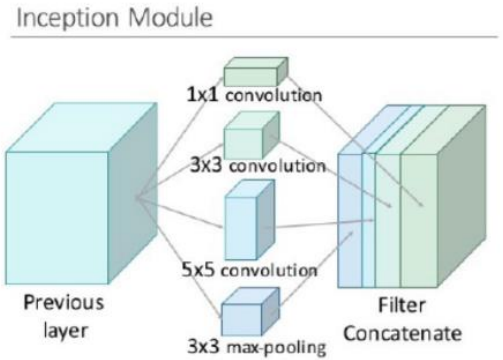
Model	top-5 classification error on ILSVRC-2012 (%)	
	validation set	test set
16-layer	7.5%	7.4%
19-layer	7.5%	7.3%
model fusion	7.1%	7.0%

K. Simonyan, A. Zisserman
Very Deep Convolutional Networks for Large-Scale Image Recognitio
arXiv technical report, 2014



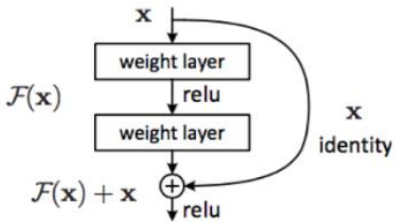
GoogleLeNet / Inception-v1

- From 2012 (AlexNet) to 2014 (VGG-16), performance got better as networks got deeper
- But, the number of parameters was getting out of hand (e.g., 60 to 180 million!)
- So, Google researchers worked to design a network that was deeper, but much more efficient
  - From the film Inception (2010)
  - Used only 5 million parameters!
- Won some contests in ImageNet ILSVRC-2014
  - Achieved top-5 error of 6.7% (VGG got 7.3%)
  - Used 22 layers with "inception modules"
- Several conv layers in parallel, with different kernel sizes
  - Key idea: for bigger kernels, use fewer channels
  - Multi-size kernels also help make scale-invariant
  - Good overview here



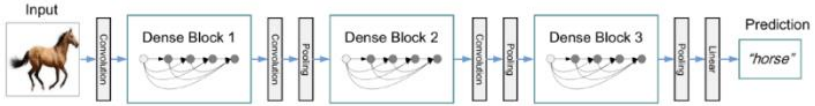
Residual Networks

- Through 2015, the best networks got deeper. But deep nets are hard to train:
  - vanishing gradients (largely solved by ReLU & batchnorm)
  - degradation problem: as networks get deeper, even training error increaes!
- A deeper network should be able to perform at least as good as a shallower network,
  - Making some layers act like identity
  - But difficult to make layers act like identity!
- Idea: Make identity easy via residual blocks:
  - Handle dimension changes w/ learned dense layer
- ResNet, introduced by Microsoft Research, Dec 2015
  - Solves degradation problem; allows super deep networks! (100s of layers)
  - Won ImageNet ILSVRC-2015 with 3.6% error rate! (GoogLeNet got 6.7%)
  - Slightly lower complexity than VGG



Other Networks in the Deep Learning Zoo

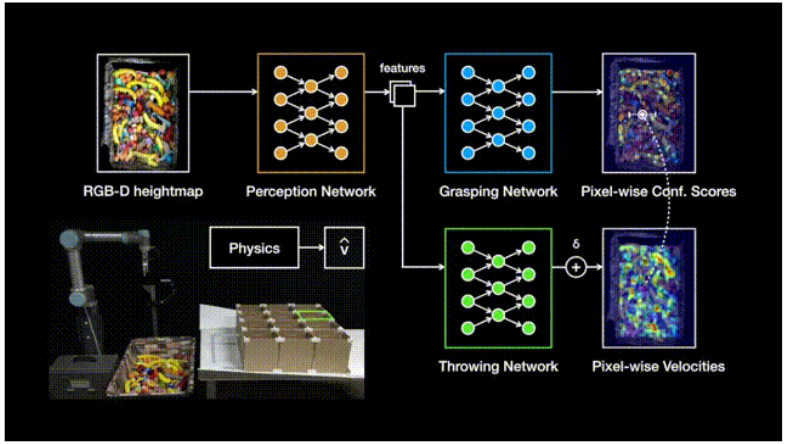
- Inception-v2, Inception-v3 (2015), Inception-v4, Inception-ResNet (2016):
  - Evolutions with more efficient inception modules, auxiliary outputs, etc. (see here)
  - Inception-v3 took 2nd place in ILSVRC-2015
- DenseNet (2016):
  - Each layer connected to all previous layers. Outperforms ResNet (see overview)

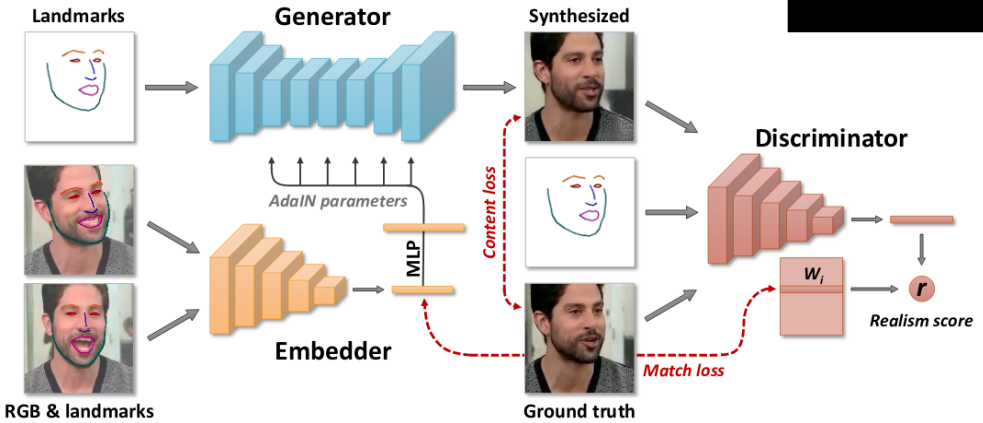


- SqueezeNet (2016):
  - Motivated by limited-memory (e.g., mobile) implementations
  - Achieved AlexNet-like performance with 50x fewer parameters
- MobileNet (2017):
  - Very efficient in memory & computation, yet high accuracy (see overview)

Application Specific Models

State of the Art, revisited





Thank You!