Convolutional Networks

CSCI 447/547 MACHINE LEARNING

Slides adapted from Towards Data Science

Outline

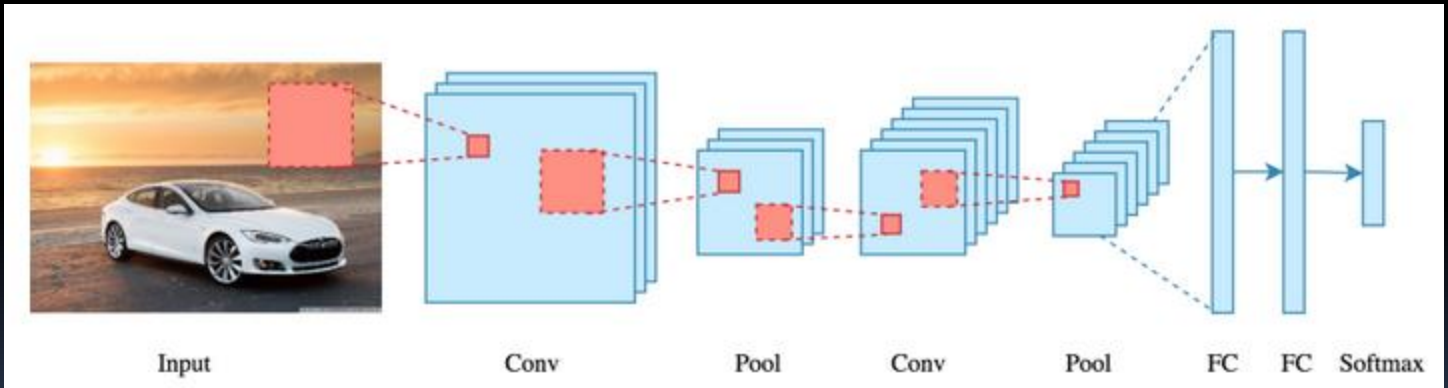
- Overview
- Architecture
- Intuition
- Example
- Visualization

Overview

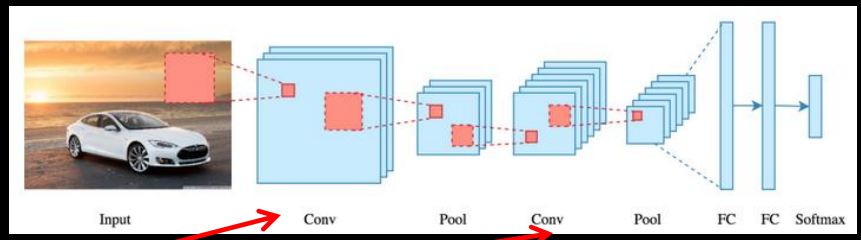
- Detects low level features
  - Uses these to form higher and higher level features
- Computationally efficient
  - Convolution and pooling operations
    - Parameter sharing
- Primarily used on images, but has been successful in other areas as well

Architecture

- "Several" convolutional and pooling layers followed by fully connected neural network layers



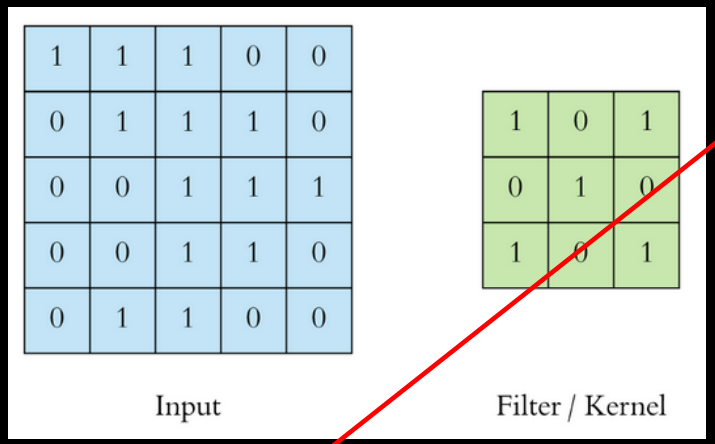
Architecture

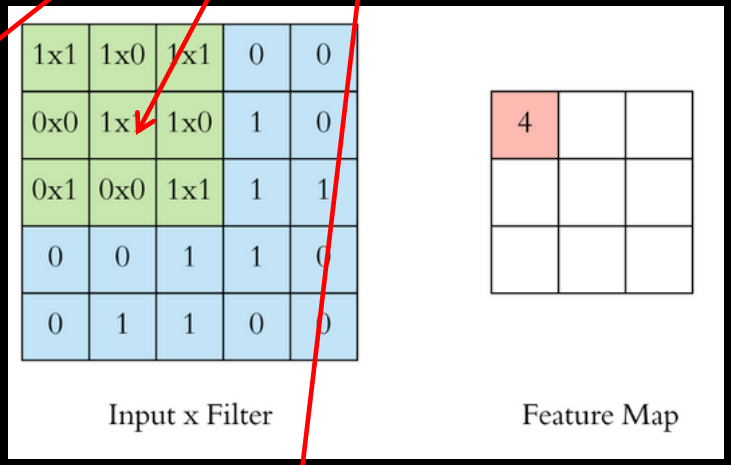


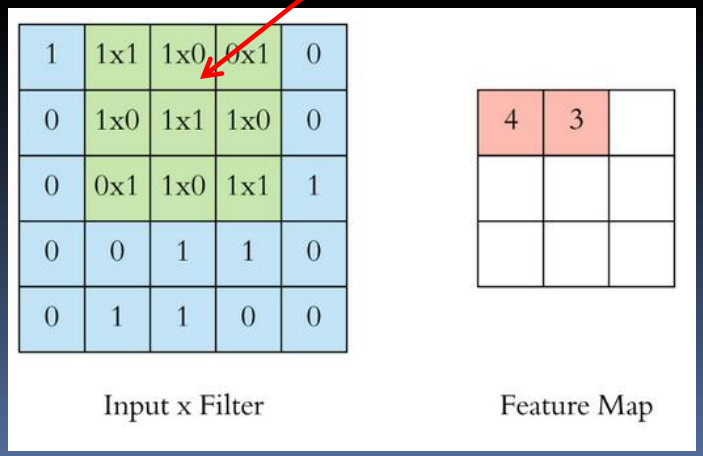
- Convolution
  - Filter or kernel applied to input data
  - Output is a feature map
    - Based on the type of filter used
  - Filter is slid over area of input
    - Values in filter multiplied by values in input and then summed together to produce one output

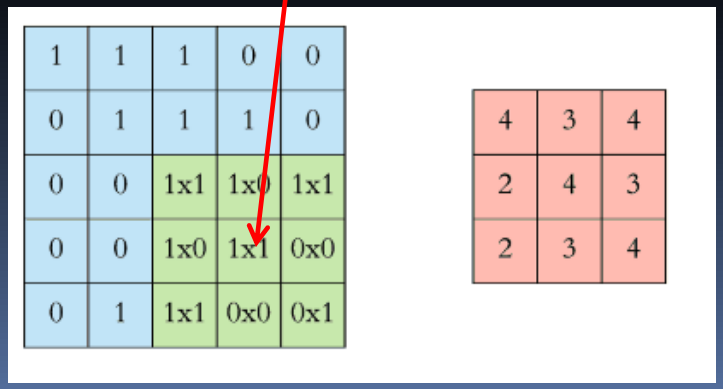
Architecture

- Convolution – 2D



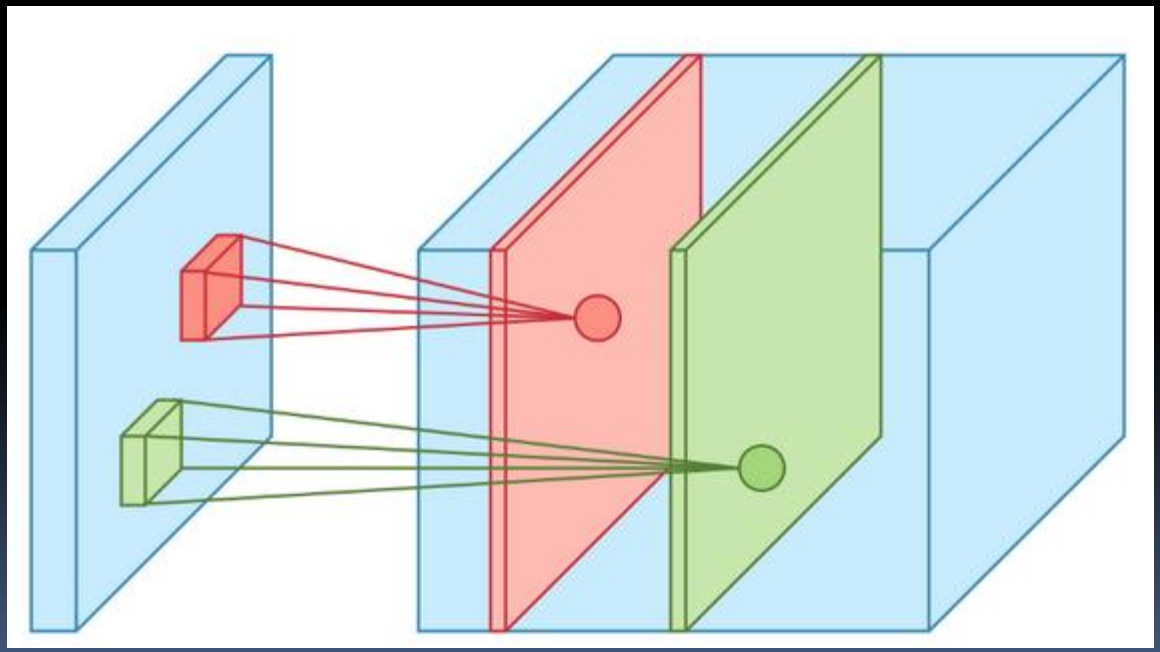






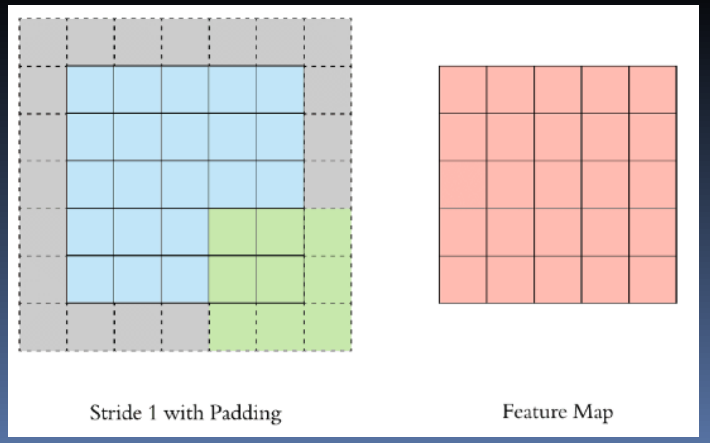
Architecture

- Convolution – 3D

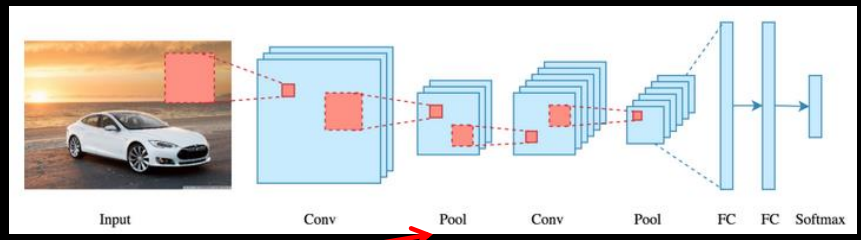


Architecture

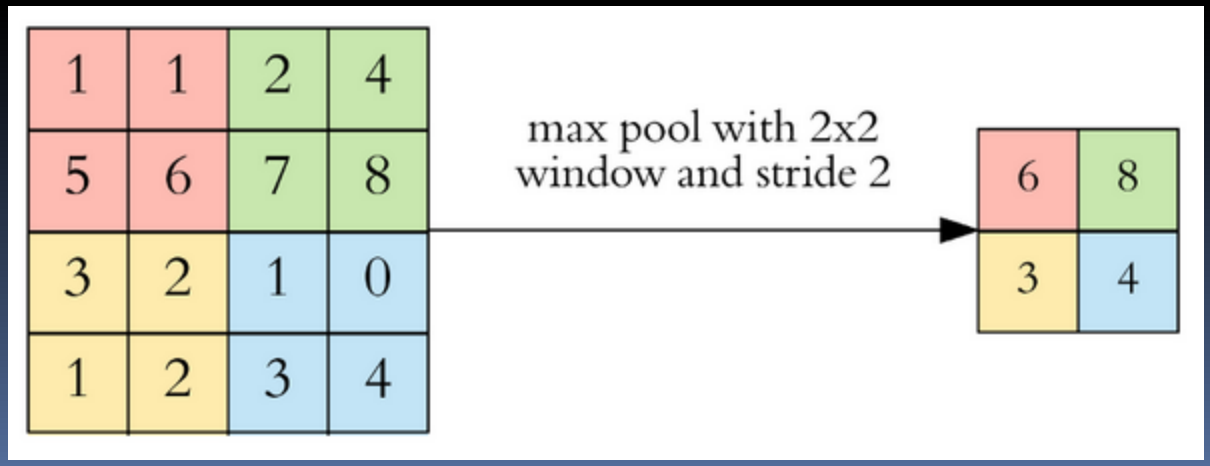
- Non-Linearity
  - Results of convolution operation passed through an activation function
    - e.g. ReLU
- Stride
  - How much the filter is moved at each step
- Padding – or not
  - Fill external boundary with o's or neighboring value



Architecture



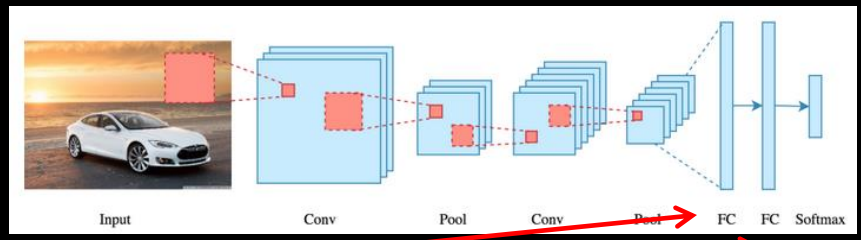
- Pooling
  - Reduces dimensionality
  - Most common is max pooling, can use average pooling also
  - Still specify stride



Architecture

- Hyperparameters
  - Filter size
  - Filter count
  - Stride
  - Padding

Architecture



- Fully connected layers
  - Same as a deep network
  - Flatten output of convolution and pooling to get vector input
- Training
  - Backpropagation with gradient descent
  - More involved than fully connected networks
    - https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/
    - https://grzegorzgwardys.wordpress.com/2016/04/22/8/
  - Filter values are weights, and are adjusted during backpropagation

Intuition

- Convolution + pooling layers perform feature extraction
  - Earlier layers detect low level features
  - Later layers combine low level into high level features
- Fully connected layers perform classification

Intuition

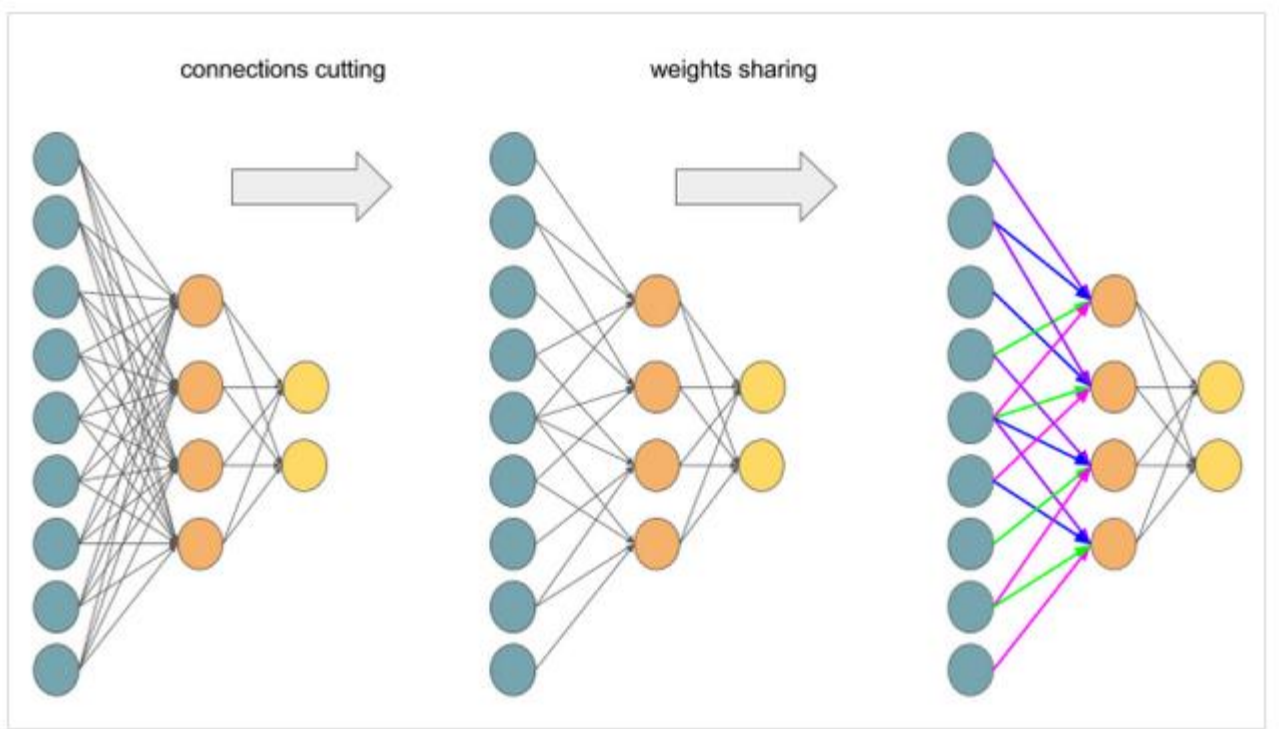
- Perspectives
  - Convolution in Image Processing
  - Weight Sharing in Neural Networks

Intuition: Image Processing

- Convolution Operators

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
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 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Intuition: Weight Sharing

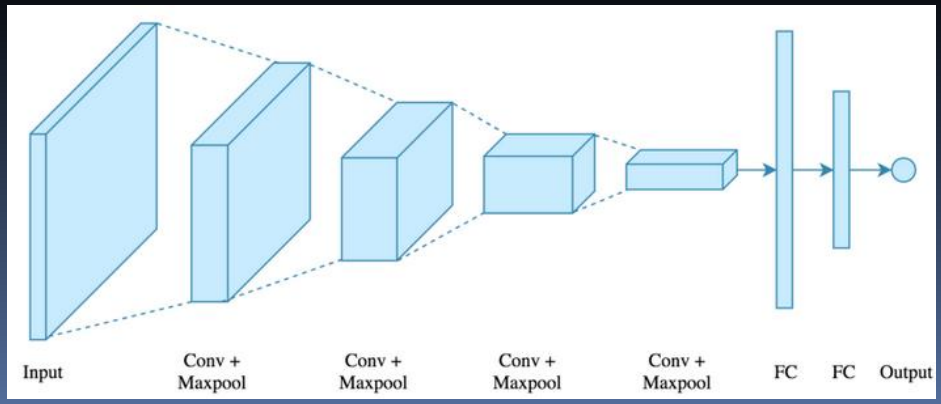


Transforming Multilayer Perceptron to Convolutional Neural Network

Example

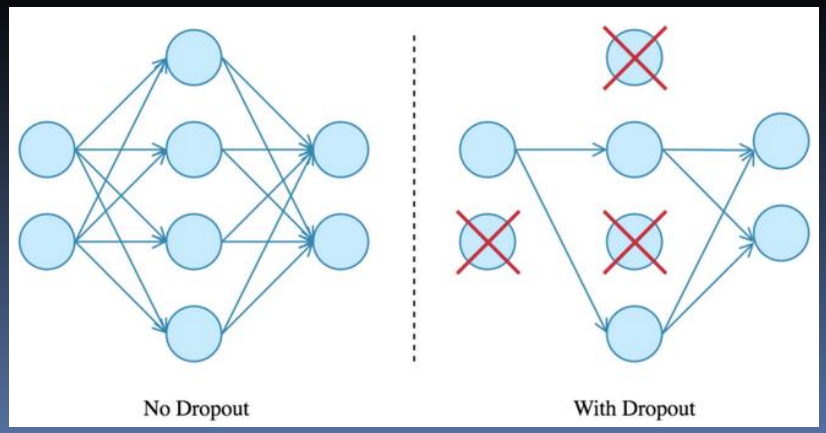
```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1',
                 input_shape=(150, 150, 3)))
model.add(MaxPooling2D((2, 2), name='maxpool_1'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'))
model.add(MaxPooling2D((2, 2), name='maxpool_2'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_3'))
model.add(MaxPooling2D((2, 2), name='maxpool_3'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_4'))
model.add(MaxPooling2D((2, 2), name='maxpool_4'))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu', name='dense_1'))
model.add(Dense(128, activation='relu', name='dense_2'))
model.add(Dense(1, activation='sigmoid', name='output'))
```

- Example is for Dogs vs Cats data from Kaggle



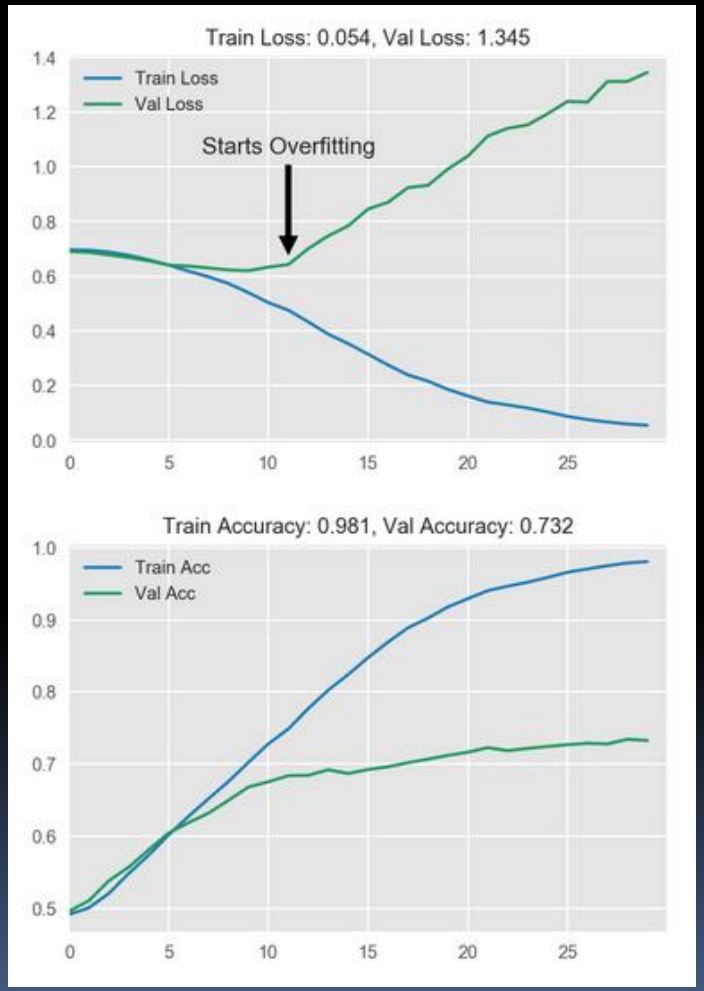
Example

- Dropout
  - Prevent overfitting
  - Temporarily disable a node with probability p
    - Can become active at the next pass
    - p is the "dropout rate" – 0.5 is a typical starting point
  - Can be applied to input or hidden layer nodes



Example

- Model Performance
  - Overfitting, despite using dropout

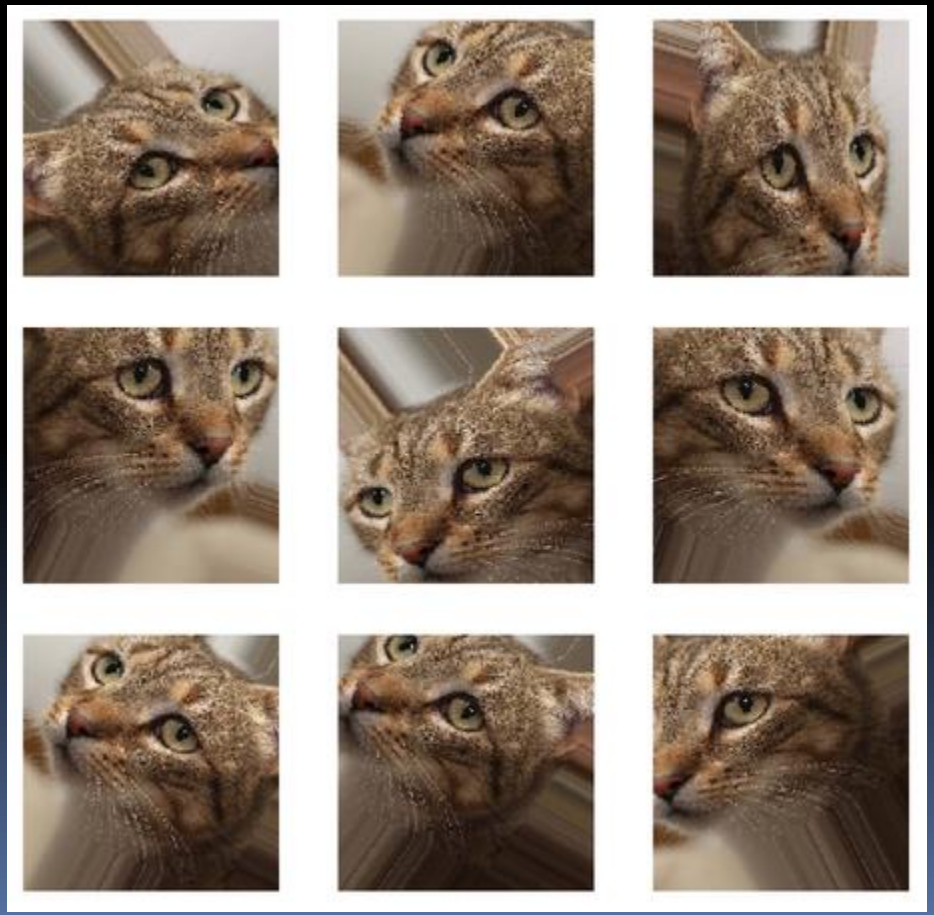


Example

- Data Augmentation
  - Using existing examples to create additional ones
  - Done dynamically during training
  - Transformations should be learnable
    - Rotation, translation, scale, exposure adjustment, contrast change, etc.

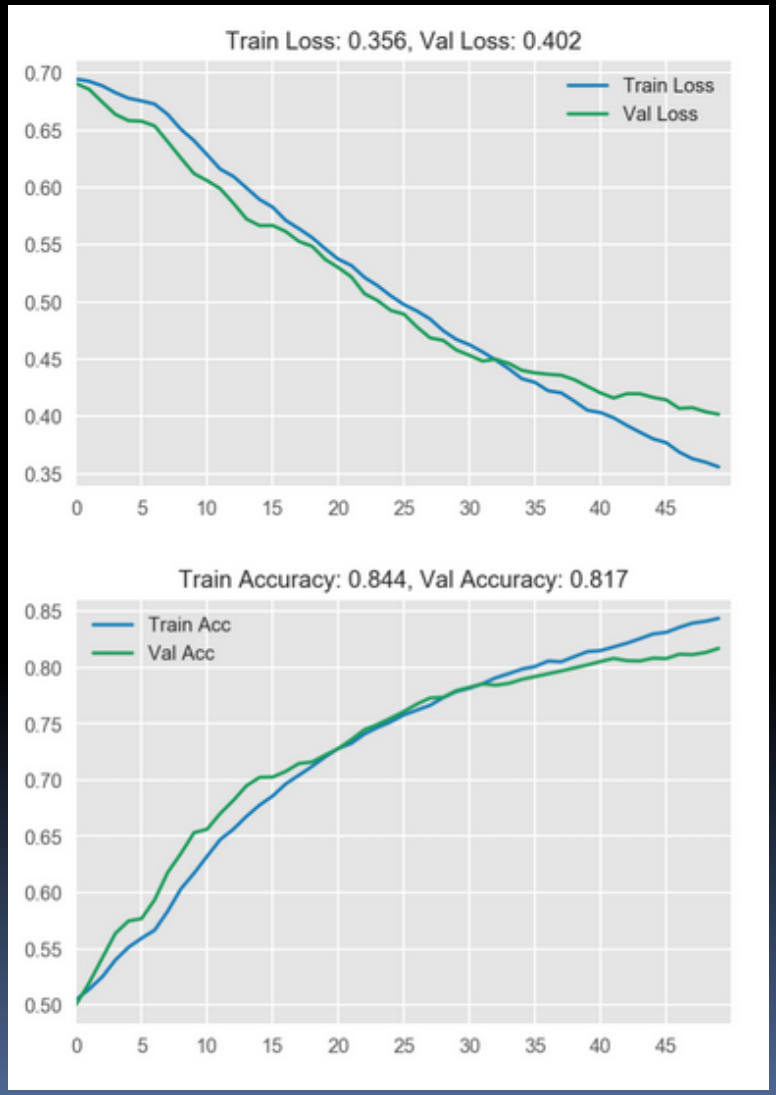
Example

- Data Augmentation

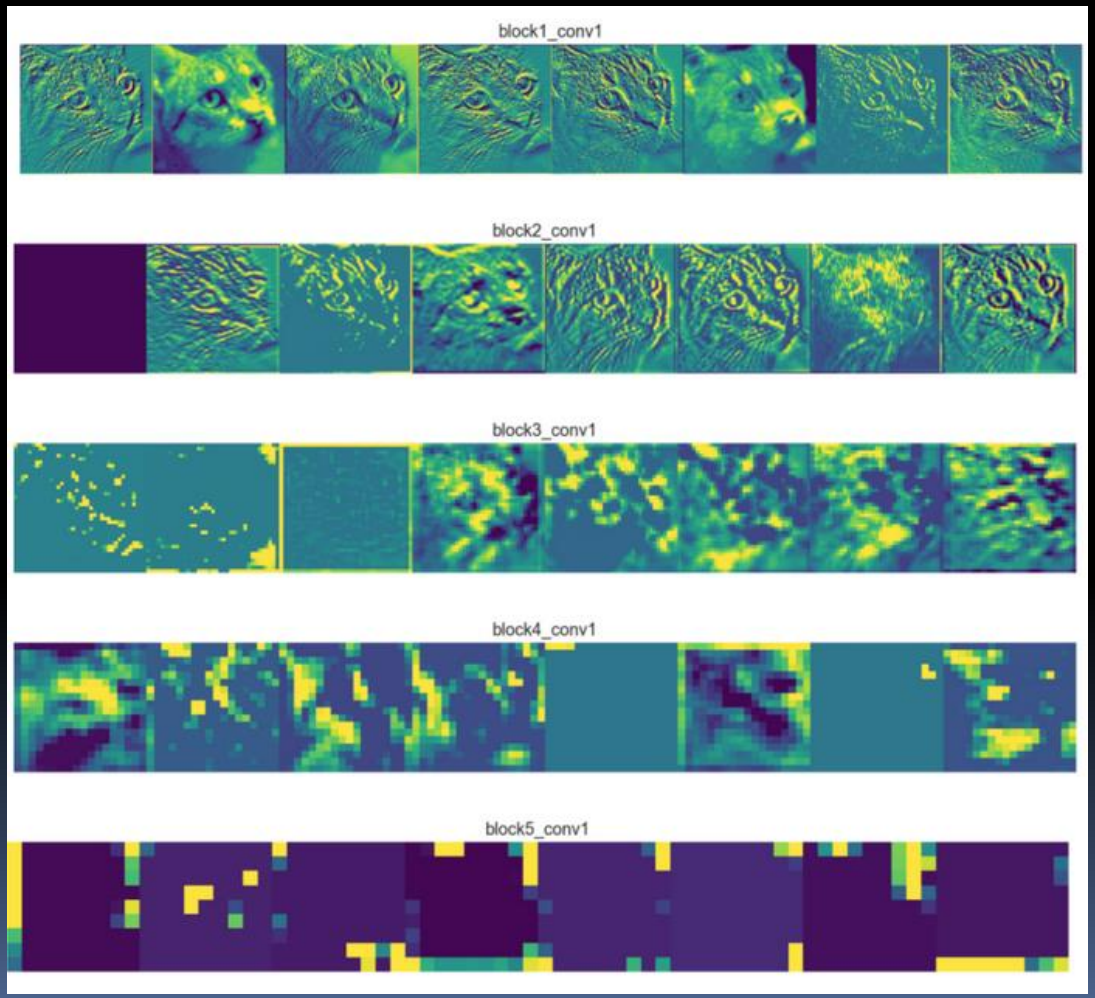


Example

- Updated Model Performance



Visualization



Summary

- Overview
- Architecture
- Intuition
- Example
- Visualization