CNNs : Practical Tips and Applications

Matt Klawonn

CNN Applications

- Image Recognition
- Video analysis
- Drug Discovery
- Misc Signal Processing

Why use a CNN?

Traditional ML Approach

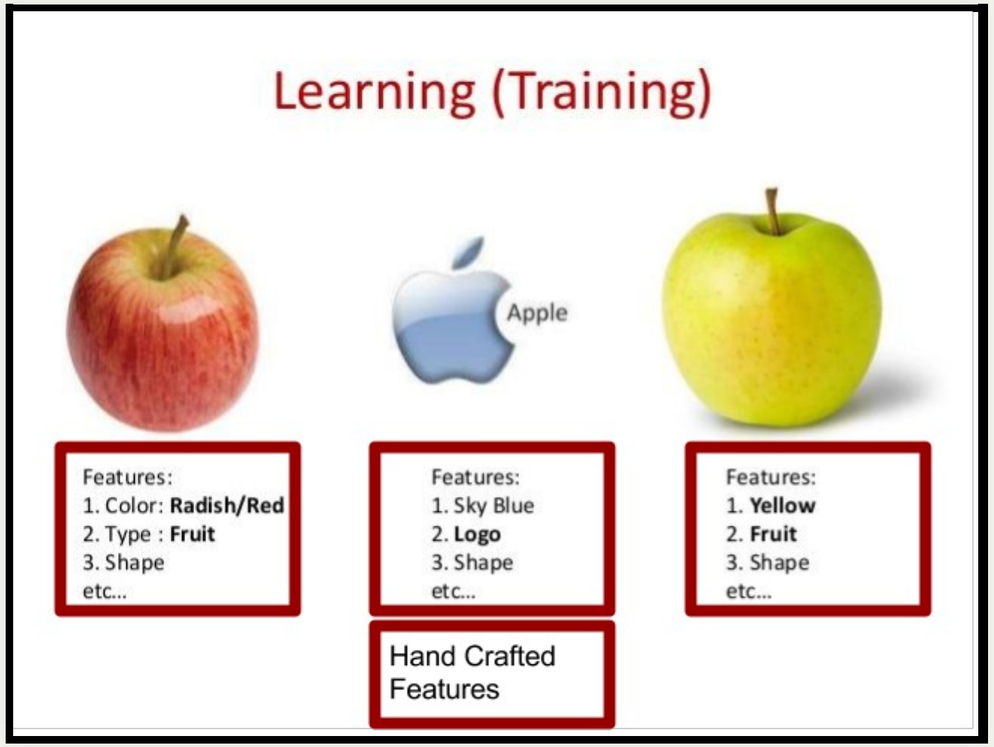


Image courtesy of here

Deep Learning Approach

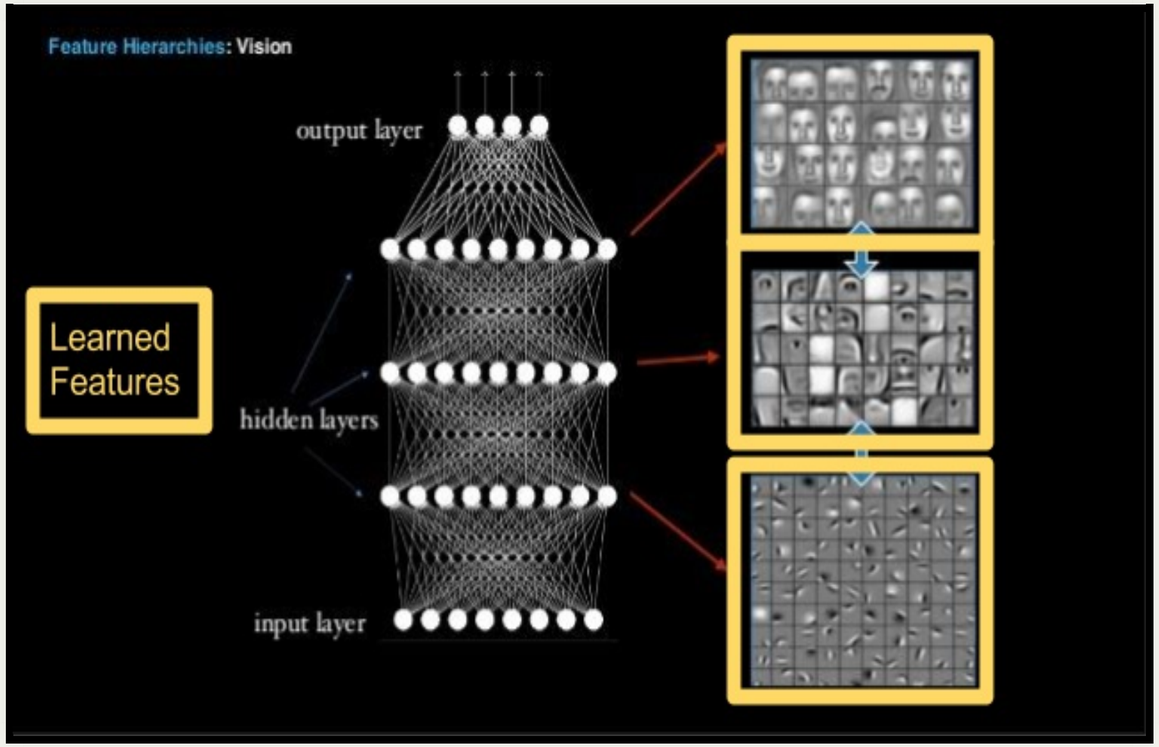


Image courtesy of here

What considerations are made when developing a CNN for a particular task?

- How to design the network (# of layers, filters, filter size).
- What preprocessing to do on the data.
- How to regularize and avoid overfitting.
- How to select hyperparameters.
- How to implement the chosen network.

Lecture Overview

- Network Design Considerations
- Data Tips
- Regularization Techniques
- Hyperparameters
- Implementation Advice
- Applications

Overview

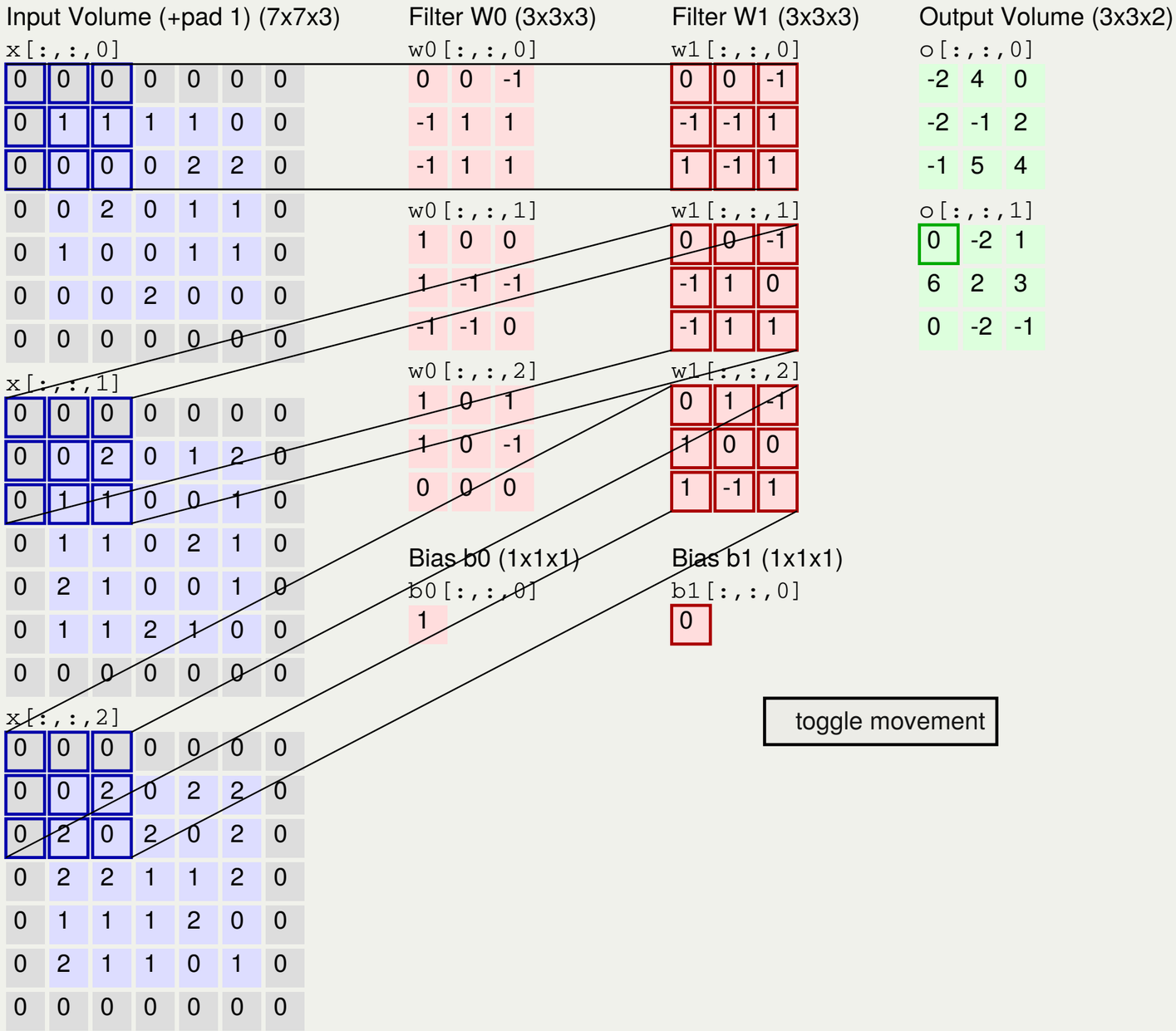
- Network Design Considerations
- Data Tips
- Regularization Techniques
- Hyperparameters
- Implementation Advice
- Applications

Recall Convolutional Layers

Convolutional layers compute feature maps via the following equation

$$C(r,c) = I * W(r,c) = \sum_{i=1}^{K} \sum_{j=1}^{K} I(r+i-1, c+j-1) W(i,j)$$

where C(r,c) is a feature map, I is an image, W is a matrix filter, and (r,c) refers to a pixel location in the image I.



Animation courtesy of here

Filters Correspond to Learned Features

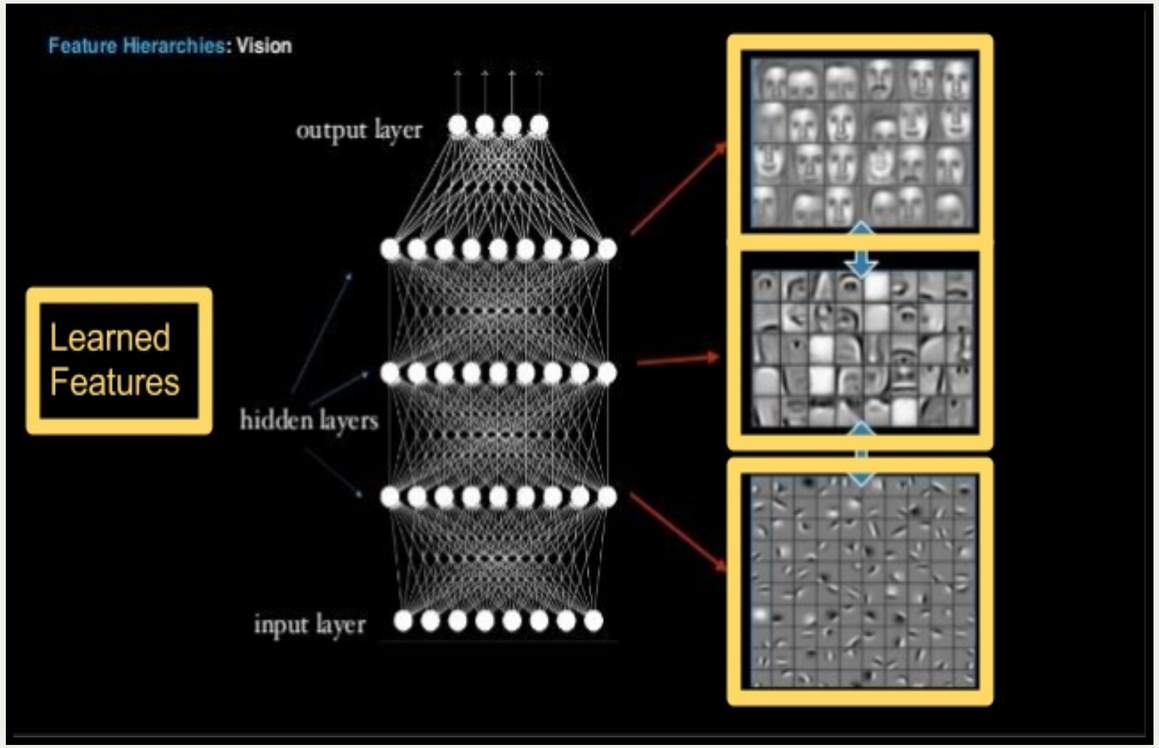
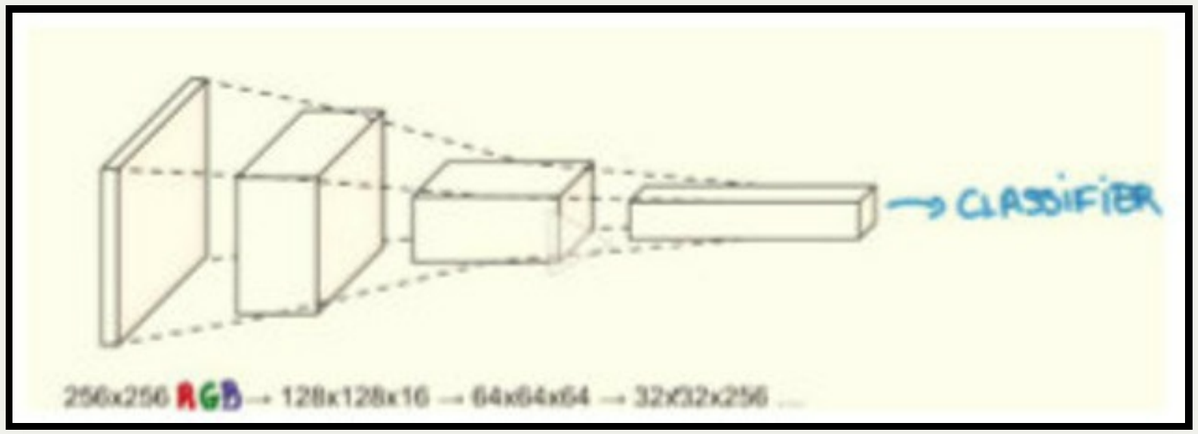


Image courtesy of here

Selecting Network Depth

Network depth will be affected by the parameters of convolutional layers and pooling layers. Once enough filters have been applied, i.e enough features have been generated, the features are fed into a classifier (fully connected layers).



Number of Filters

The number of filters generally increases as you go further into the network. This is because early filters are computing basic features. There are only so many edge detectors that are useful.

Fully Connected Classifier

This is typically a few "normal" (fully connected) neural network layers, though people have used SVMs and other classification algorithms with the convolutional features as inputs.

Existing Models

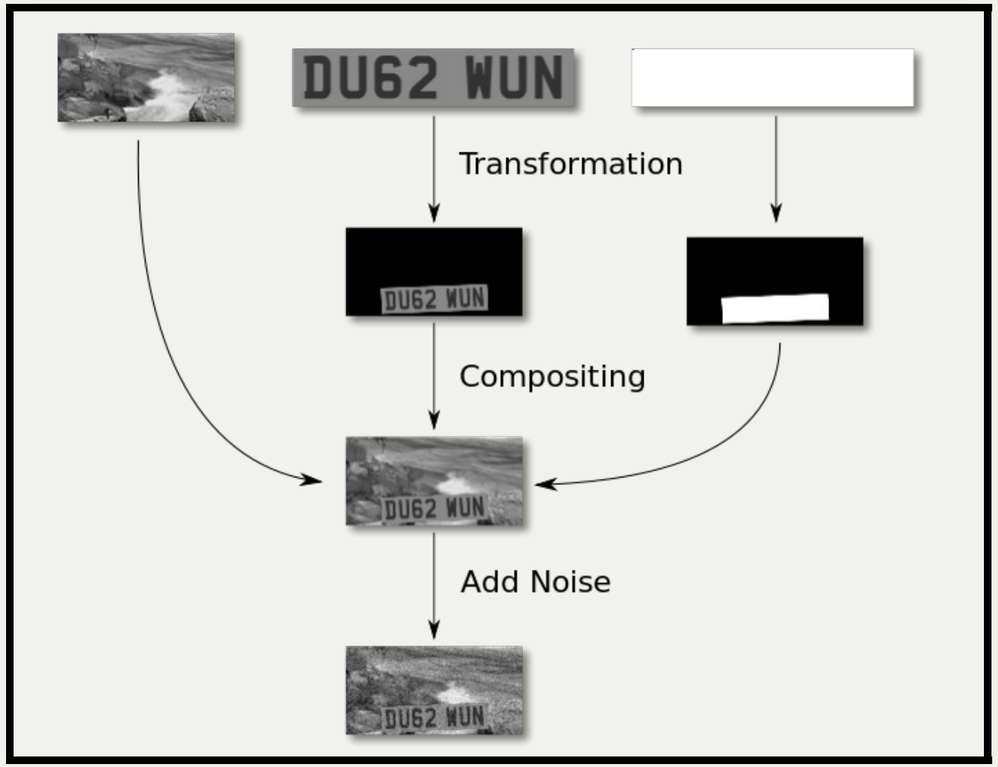
Deep learning tends to be a field which adopts practices "because they work." Try out choices with which previous models have had success.

Overview

- Network Design Considerations
- Data Tips
- Regularization Techniques
- Hyperparameters
- Implementation Advice
- Applications

Data Augmentation

Deep models require lots of data.



Example of generating image for data augmentation. Image courtesy of here

More Data Augmentation

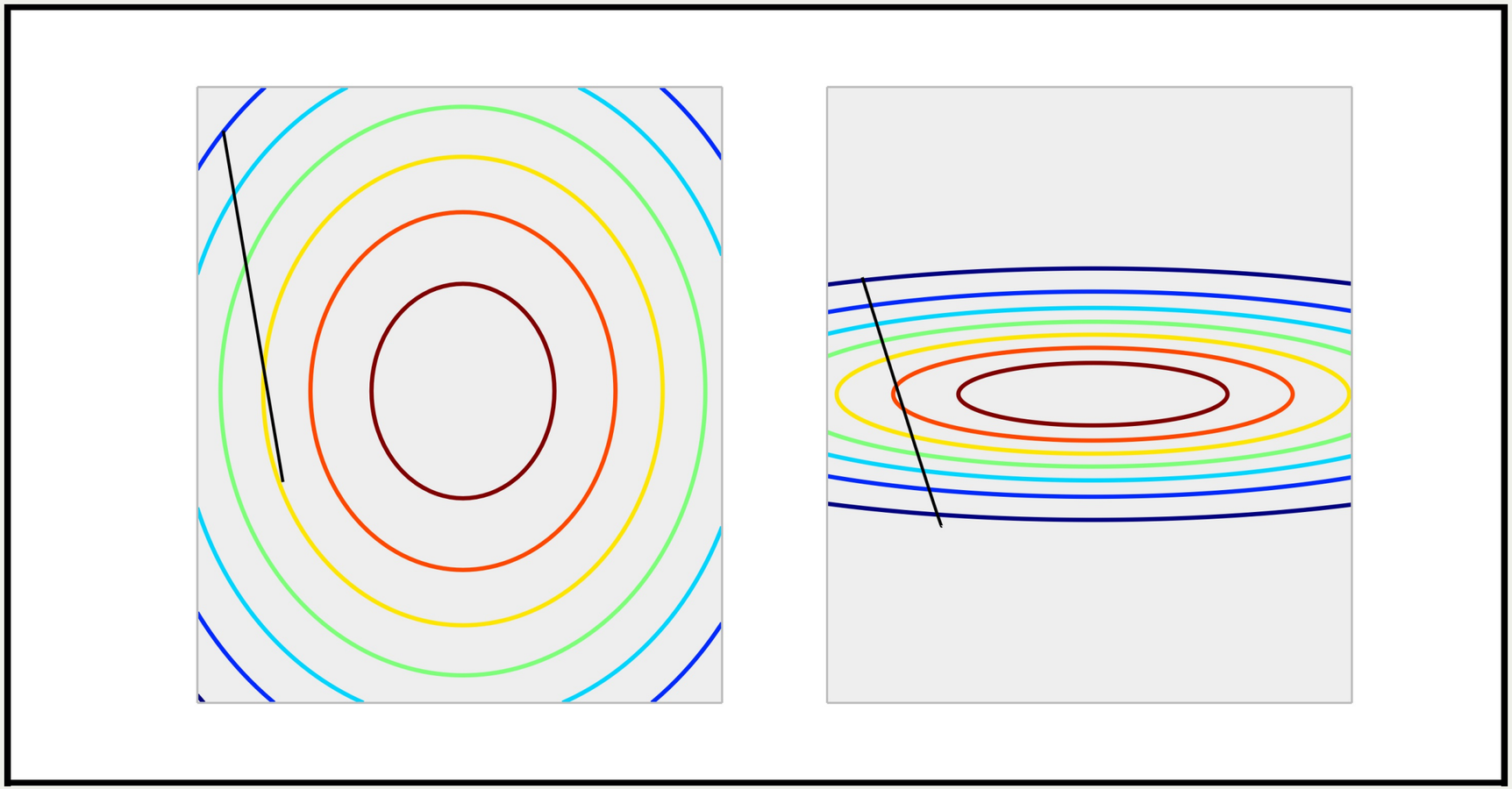
- Random Cropping
- Horizontally Flipping

Data Imbalances

Data imbalances can have negative impacts on training (see here.) One can solve this by cutting down on the number of more abundant classes, artificially enhancing the number of less abundant classes, or first training on abundant classes and then finetuning on rarer ones.

Normalization

In previous assignments, dividing MNIST images by 255 has yielded normalized data. Normalized data is nicer for optimization.



Zero Centering and Normalization

The most common type of preprocessing is to compute a mean over the training data, and subtract this mean from each data point to zero center the data. Normalization is then performed on the centered data. This is done to give lower importance to inputs which are "average" as they will be near zero.

Overview

- Network Design Considerations
- Data Tips
- Regularization Techniques
- Hyperparameters
- Implementation Advice
- Applications

Regularization Helps with Overfitting

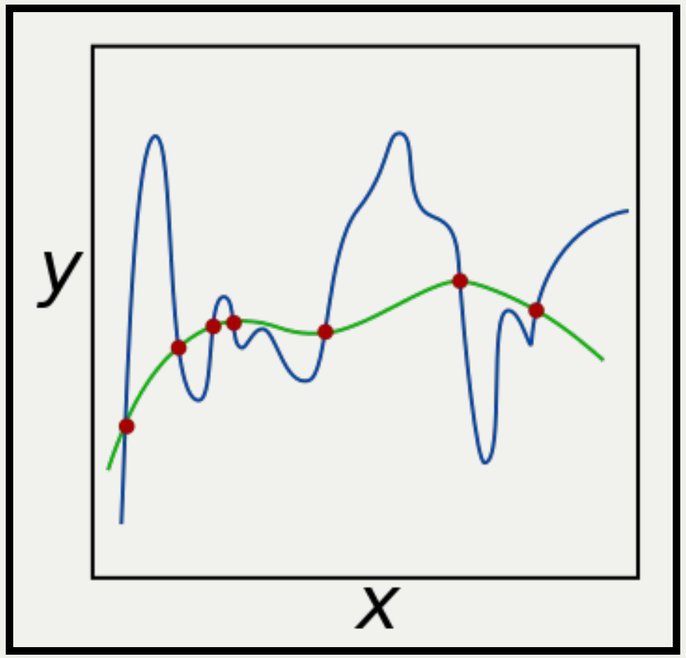


Image courtesy of here

L1 and L2 Regularization

$$L(D:\Theta) = 1/M \sum_{m=1}^{M} l(X[m], y[m], \Theta) + \lambda R(\Theta)$$

$$\text{L1}: R(\Theta) = \|\Theta\|_1$$

$$\text{L2}: R(\Theta) = \|\Theta\|_2^2$$

Max Norm Constraints

Enforce an absolute upper bound on weights going from one neuron to another.

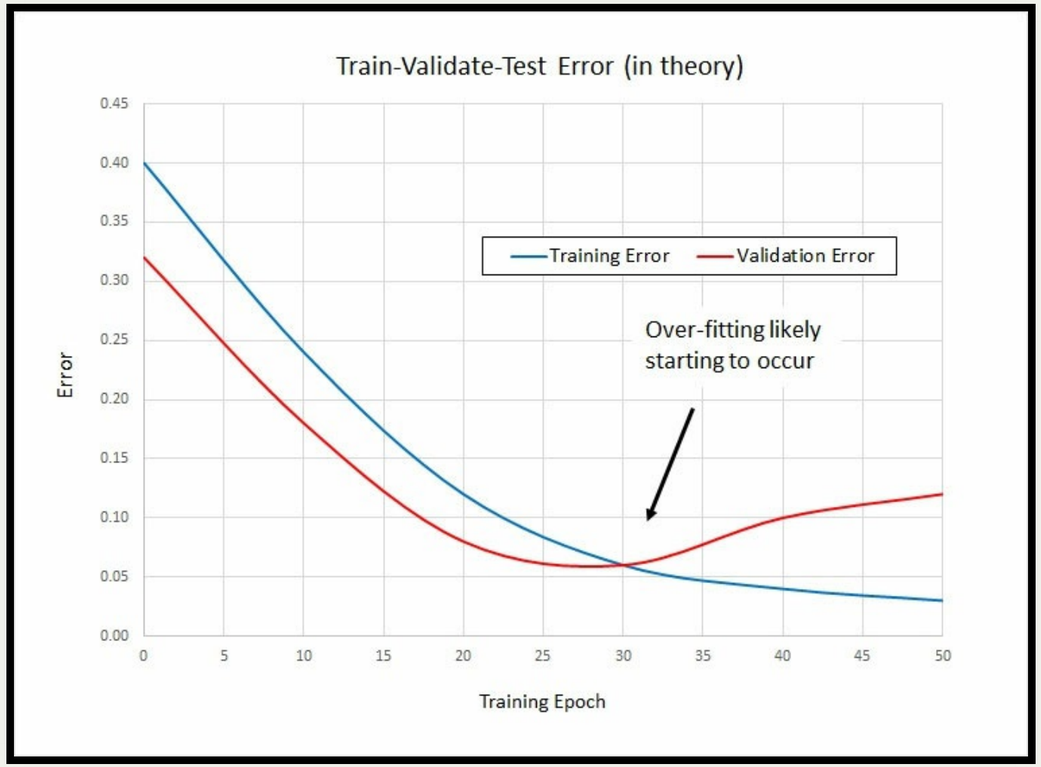
$$\overset{L}{W_{ij}} \leq c$$

Max Norm in TensorFlow

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
grads = optimizer.compute_gradients(loss, [w1, w2, ...])
#Grads is a list of tuples (gradient, associated_variable)
capped =
[(tf.clip_by_norm(grad_tuple[0], clip_norm=1.0, axes=0), grad_tuple[1])
                for grad_tuple in grads]
optimizer = optimizer.apply_gradients(capped)
```

Early Stopping

Stop when validation loss stops decreasing.

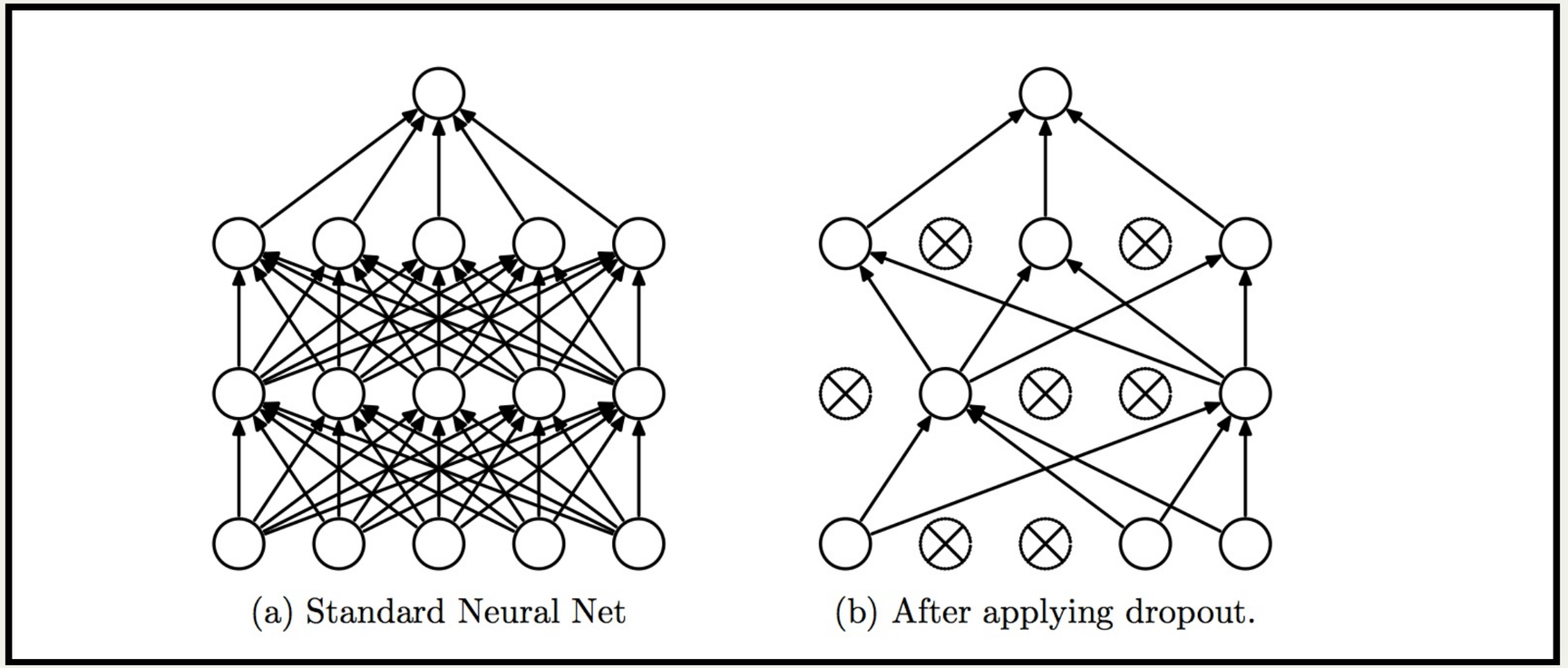


Early stopping illustration. Image courtesy of here

Early Stopping in TensorFlow

```
with tf.Session() as sess:
    for _ in xrange(max_iterations):
        sess.run(training_op)
        val_loss = sess.run(loss, feed_dict =
                    {data_placeholder : validation_data,
                    label_placeholder : validation_labels})
        if val_loss > last_val_loss:
            counter += 1
        if counter == early_stop_threshold:
            break
        last_val_loss = val_loss
```

Dropout



Dropout illustration. Image courtesy of here

Dropout in TensorFlow

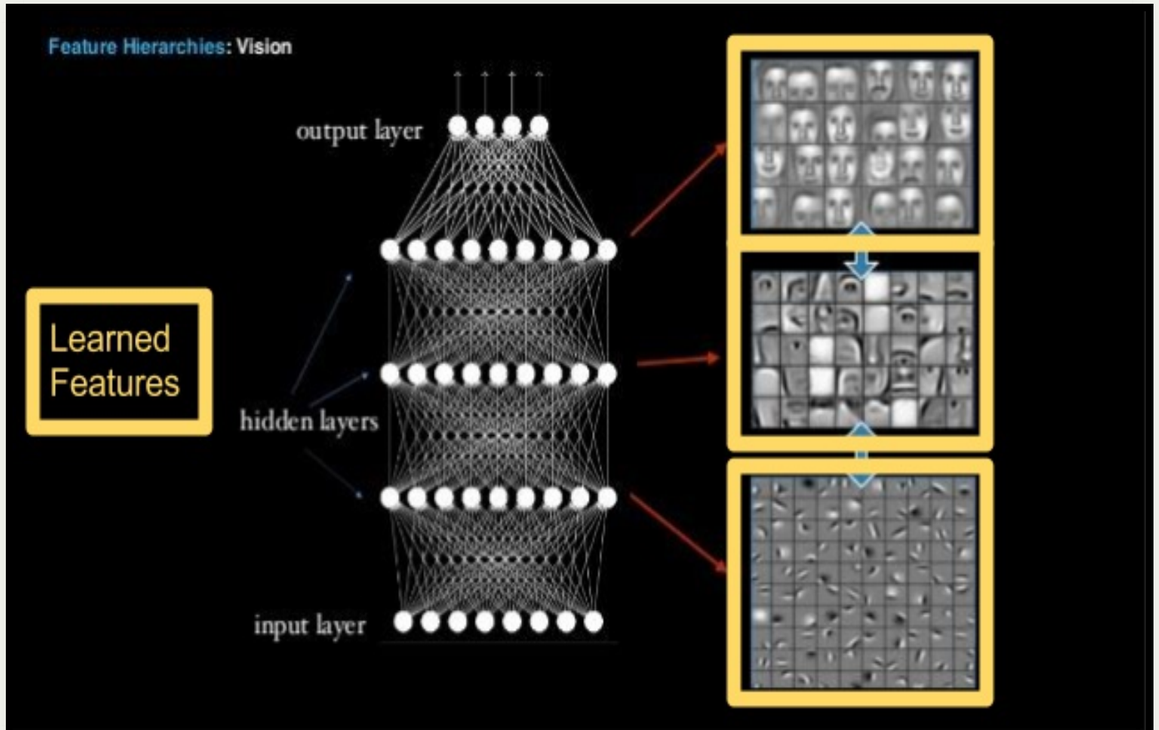
```
#x is the input (i.e a layer's activation)
#keep_prob is the probability that each entry in the input is not set
#to zero
d = tf.nn.dropout(x, keep_prob, noise_shape=None, seed=None, name=None)
#Pass d along to the next layer
```

Transfer Learning

Many times it is unnecessary to train a CNN from scratch! Models which have been trained on large image classification tasks are often a good starting point for other image recognition tasks.

Feature Hierarchy and Transfer Learning

When finetuning a pre-trained network to a new task, consider how similar the tasks are.



Overview

- Network Design Considerations
- Data Tips
- Regularization Techniques
- Hyperparameters
- Implementation Advice
- Applications

Validation Data

Validation data can be used to select hyperparameters in a rigorous way. A common practice is to set aside 10% of the training data for use as validation data.

Activation Function Choices

Recall the purpose of an activation function is to introduce a non-linear transform into the learning algorithm.

Motivations of ReLUs

ReLUs avoid the saturation problem of sigmoids.

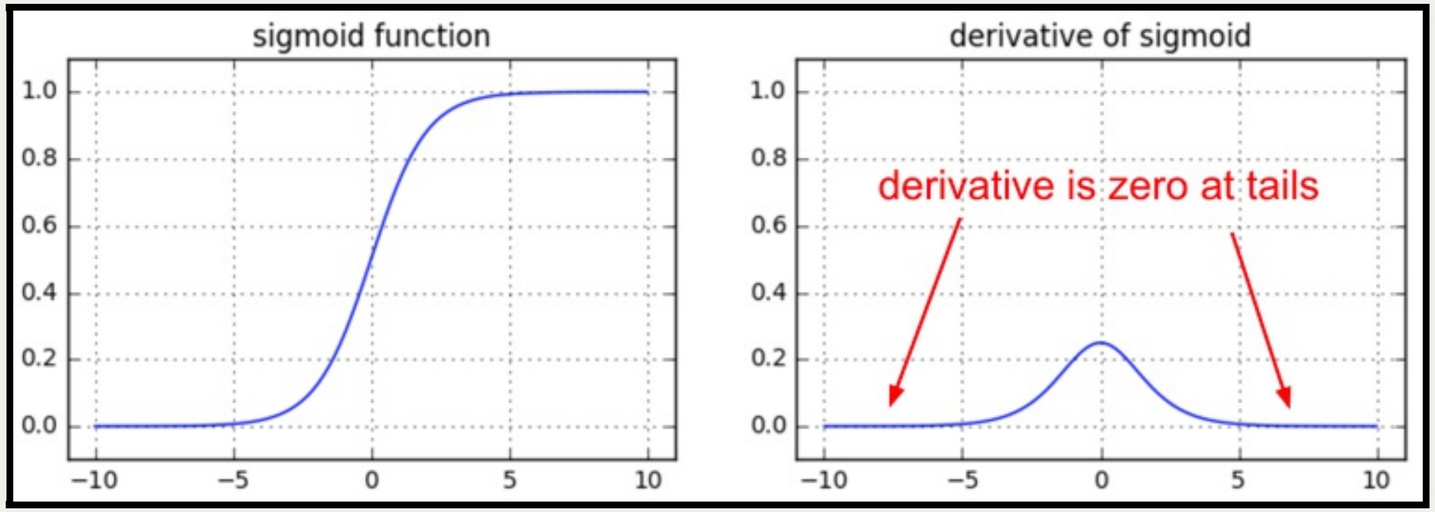


Image courtesy of here

Motivations of Leaky ReLU and Variants

Leaky ReLUs avoid "dead neurons" which can occur when using ReLU.

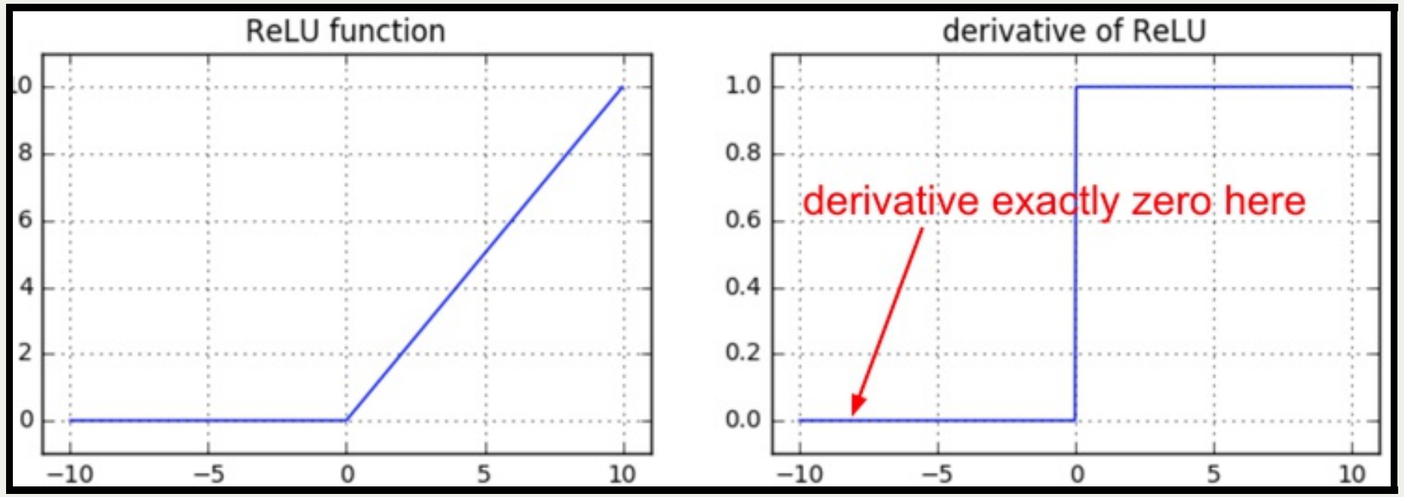


Image courtesy of here

Initialization Choices

Initialzation matters due to non-convexity of deep models. The choice is motivated by activation function choice and network depth. A basic strategy is to use small random weights, with the randomness helping to break symmetry, e.g $W \sim 0.001 \times N(0, 1)$.

Xavier (Glorot) Initialization

One problem with simple small random initialization is that the variance of a neuron's output grows with its fan-in. Xavier initialization proposes to normalize the variance of the output of each neuron in a layer to 1. This is done by dividing each random weight by the fan-in for that neuron.

Xavier Initialization Derivation

The variable "s" represents a neuron's unactivated output. Note that the derivation assumes the weights and inputs have a mean of zero, which is not the case for ReLU activations.

$$
\begin{aligned}
\mathrm{Var}(s) &= \mathrm{Var}\left(\sum_i^n w_i x_i\right) \\
&= \sum_i^n \mathrm{Var}(w_i x_i) \\
&= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i) \\
&= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i) \\
&= (n\mathrm{Var}(w))\,\mathrm{Var}(x)
\end{aligned}
$$

He Initialization

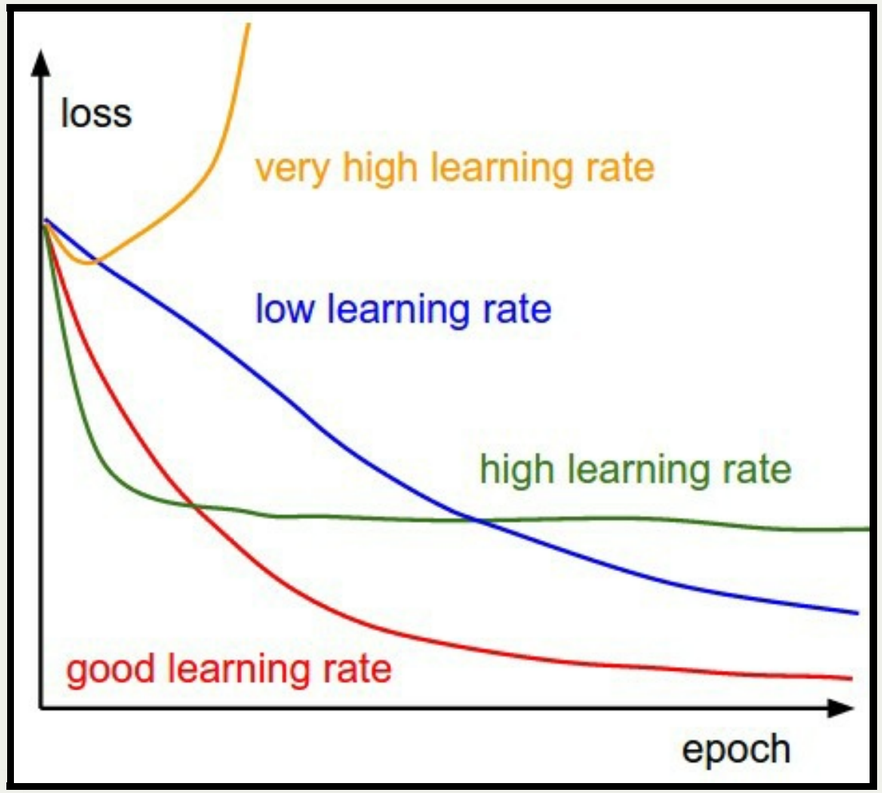
He initialization uses a similar idea (normalizing the variance) which is more appropriate for ReLU activations. This paper provides the derivation, but the recommend multiplying each weight by $\sqrt{2/n}$ where "n" is the fan-in.

Xavier and He in TensorFlow

```
W_Xavier = tf.get_variable("W",
       shape=[prev_hidden_shape, next_hidden_shape],
       initializer=tf.contrib.layers.xavier_initializer())
W_He = tf.get_variable("W",
       shape=[prev_hidden_shape, next_hidden_shape],
       initializer=tf.contrib.layers.variance_scaling_initializer(
           factor=2.0, mode="FAN_IN"))
```

Learning Rate

The learning rate is probably the most important hyperparameter to cross-validate. Monitoring the training and validation loss output during training helps to see if the learning rate is appropriate.



More Optimization Techniques

- Learning Rate Schedule
- Momentum
- Nesterov Momentum
- Adaptive Learning Rate Optimizers

Learning Rate Schedule

```
learning_rate = tf.placeholder(tf.float32, shape=[])
# ...
train_step = tf.train.GradientDescentOptimizer(
    learning_rate=learning_rate).minimize(loss)

sess = tf.Session()

# Feed different values for learning rate to each training step.
sess.run(train_step, feed_dict={learning_rate: 0.1})
sess.run(train_step, feed_dict={learning_rate: 0.1})
sess.run(train_step, feed_dict={learning_rate: 0.01})
sess.run(train_step, feed_dict={learning_rate: 0.01})
```

Code courtesy of Stackoverflow

CNN Specific Parameters

- Filter Size
- Stride
- Padding
- Pooling Size

Filter Size

Generally stacks of layers with smaller filters are preferable to one larger filter. This is because the stack of smaller filters can compute more powerful features due to the nonlinearities between layers. The smaller filters also require fewer parameters. A 3x3 filter size is commonly used, and the number of filters usually increases with each layer.

Stride

Small strides allow filters to be applied across more patches of an image. A stride size of 1 or 2 is common. Note that the filter size and image size limits the stride size.

Padding

Padding allows filters to be applied to edges of the input. 1 or 2 are common choices. Bigger filters mean more padding may be necessary.

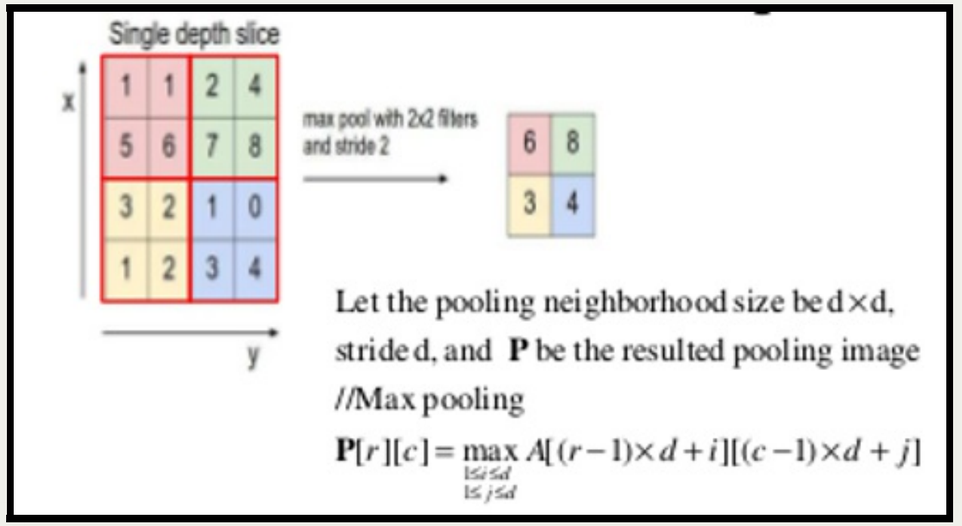
Convolutional Layer Output Volume

Filter size, stride size, and amount of padding all contribute to the output volume dimensions. They must work out such that each output dimension is an integer. In the following formulae, assume square inputs and square filters. Let the filter be FxF, the number of filters be K, the stride be S, and padding be P.

- $W_2 = H_2 = (W_1 - F + 2P)/S + 1$
- $D_2 = K$

Pooling Size

Pooling layers are determined by their receptive field size and stride. In practice two combinations are used, one with a field size of 3x3 and stride of 2, another with a field size of 2x2 and stride of 2. Larger receptive fields than 3x3 tend to downsample too aggressively.



Overview

- Network Design Considerations
- Data Tips
- Regularization Techniques
- Hyperparameters
- Implementation Advice
- Applications

Implementation of Convolutions

Consider an input of shape [28, 28, 1], padding size of 1, and a conv layer with 16 filters of shape [3, 3, 1] with a stride of 1. The output volume will be of width and height (28 -3 + 2)/1 + 1 = 28.

```
filter_size = 3
num_channels = 1
num_filters = 16
W_conv = tf.get_variable("W",
        shape=[filter_size,filter_size,num_channels,num_filters],
        initializer=tf.contrib.layers.variance_scaling_initializer(
            factor=2.0, mode="FAN_IN"))
b_conv = tf.Variable(tf.ones([32]))
conv_out = tf.nn.conv2d(x, W_conv, strides=[1, 1, 1, 1], padding='SAME')
```

Same vs Valid Padding in TF

- `"VALID"` = without padding:

```
inputs:         1  2  3  4  5  6  7  8  9  10 11 (12 13)
                |_________________|                dropped
                         |_________________|
```

- `"SAME"` = with zero padding:

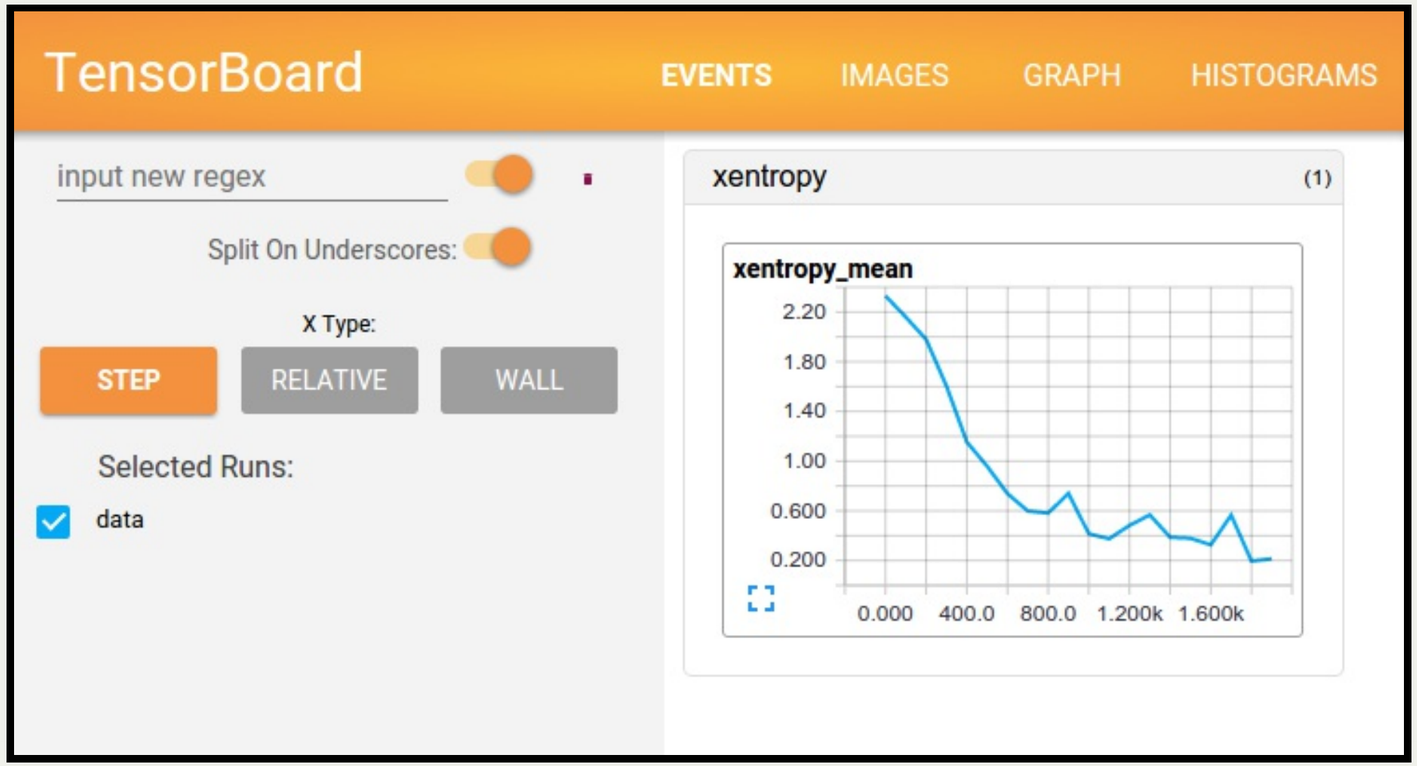
```
                pad|                                      |pad
inputs:          0 |1  2  3  4  5  6  7  8  9  10 11 12 13|0  0
                 |_________________|
                          |_________________|
                                   |_________________|
```

Image courtesy of here

Approximating Memory Use

Often deep learning computations are performed on a GPU. The memory of a GPU tends to be the limiting factor (high-end models are about 12GB). To approximate how much memory your model needs, the memory requirements of parameters, activations, and batch data must be considered.

Monitor Training with TensorBoard



TensorBoard summary:

Overview

- Network Design Considerations
- Data Tips
- Regularization Techniques
- Hyperparameters
- Implementation Advice
- Applications

Image Classification

Caffe Demos

The Caffe neural network library makes implementing state-of-the-art computer vision systems easy.

Classification

Click for a Quick Example

Provide an image URL

Classify URL

Or upload an image:

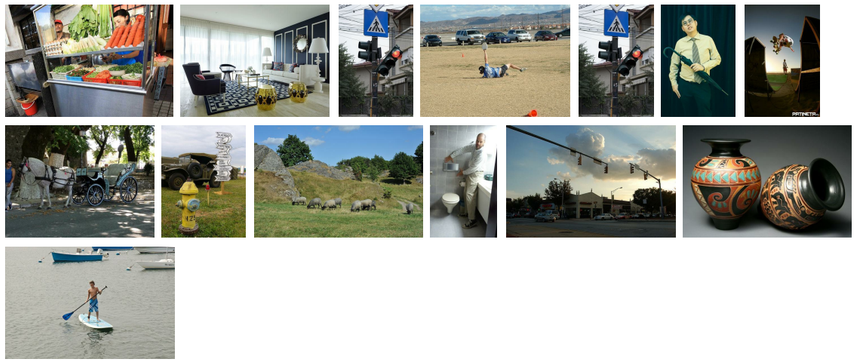
Choose File No file selected

Visual Question Answering

Visual Question Answering Demo

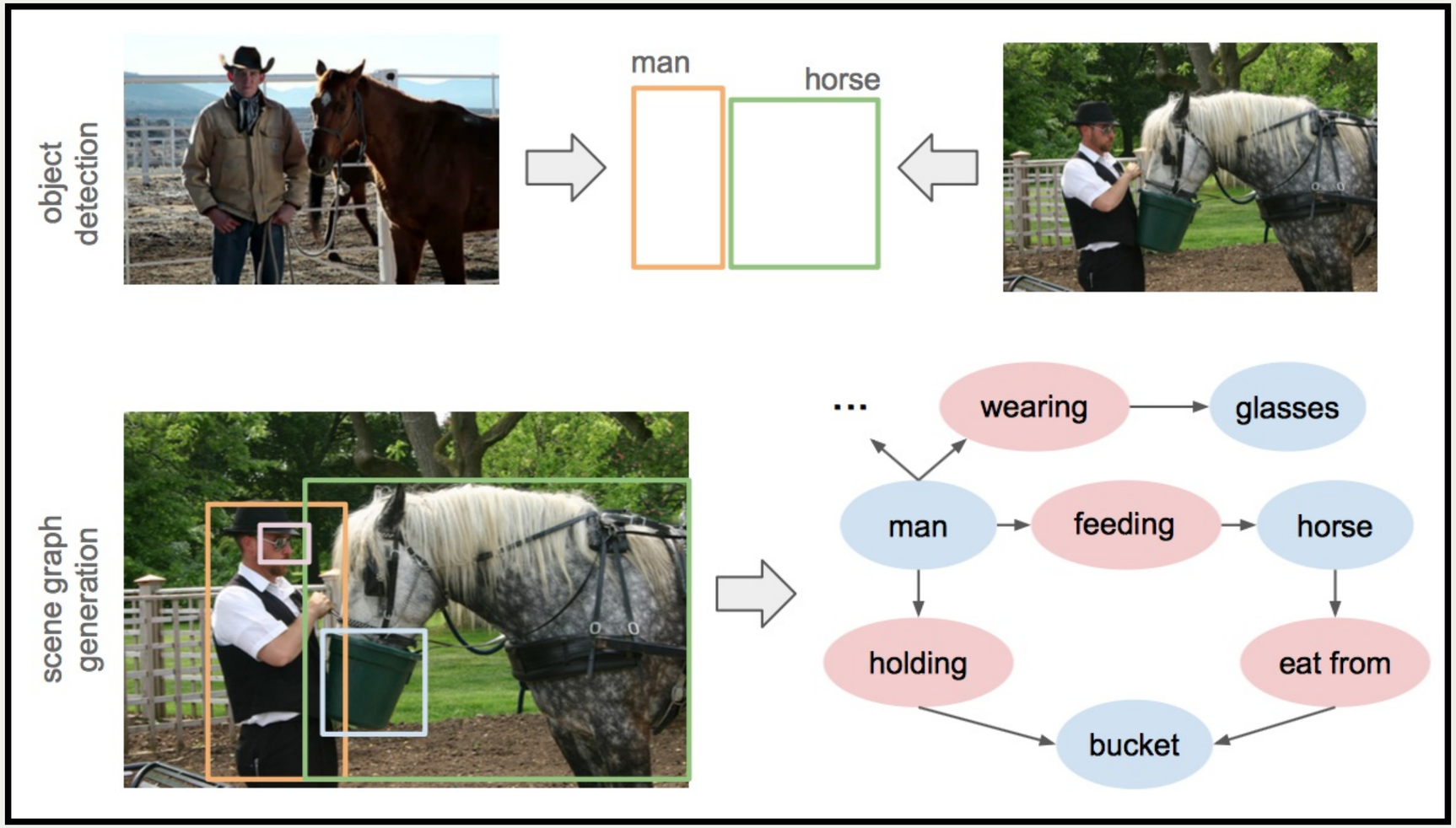
This demo could answer questions relevant to the selected image. You could click one image below (refresh this page to get more images) then type question you would like to ask about this image. This demo is developed by Bolei Zhou. More details are in report and code.

1.Click One:



2.Type Question: For example: where is the place

Scene Graph Generation



Questions?

- Network Design Considerations
- Data Tips
- Regularization Techniques
- Hyperparameters
- Implementation Advice
- Applications