Deep Learning Explained

Module 4: Convolution Neural Networks (CNN or Conv Nets)

Sayan D. Pathak, Ph.D., Principal ML Scientist, Microsoft
Roland Fernandez, Senior Researcher, Microsoft

Module Outline

Application:

OCR using MNIST data

Model:

- Recap Multi-Layer Perceptron
- Convolution Network
- Popular Deep Convolution Networks

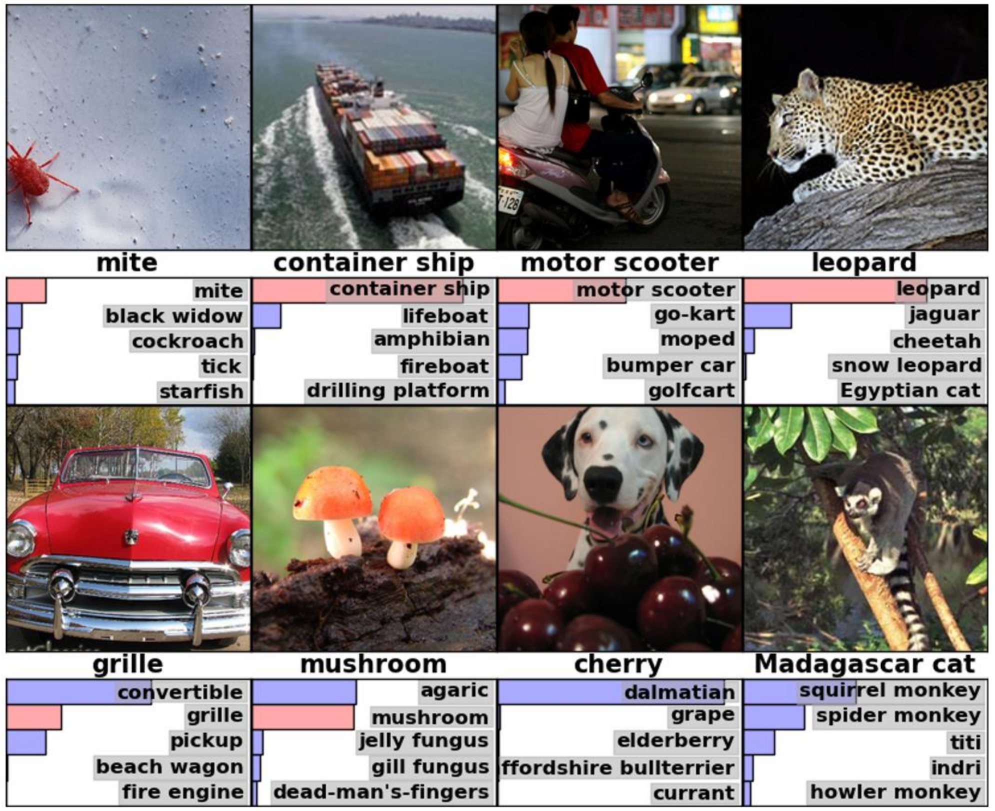
Concepts:

- Convolution
- Pooling

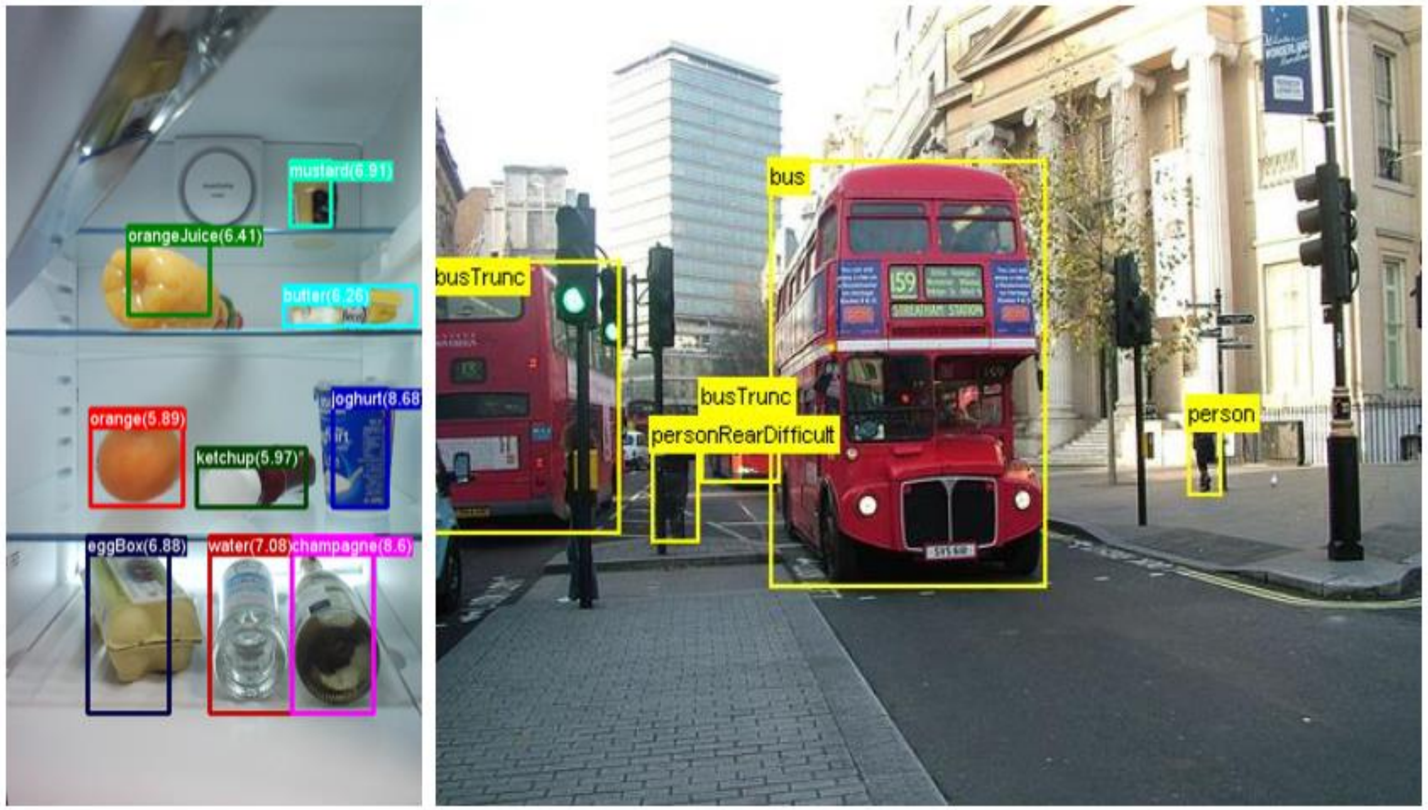
Train-Test-Predict Workflow

Applications of Conv Nets

Image Tagging



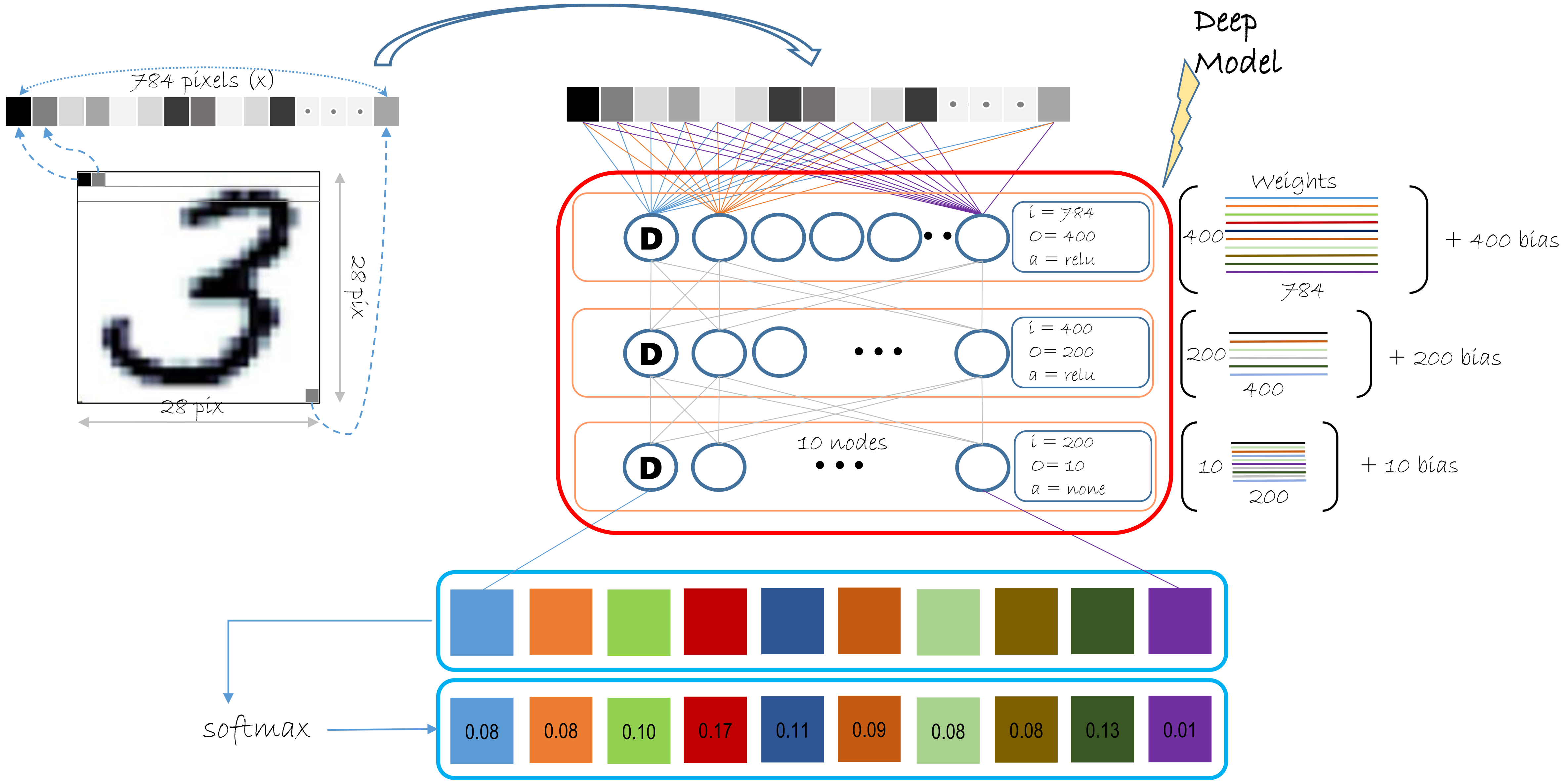
Object Detection



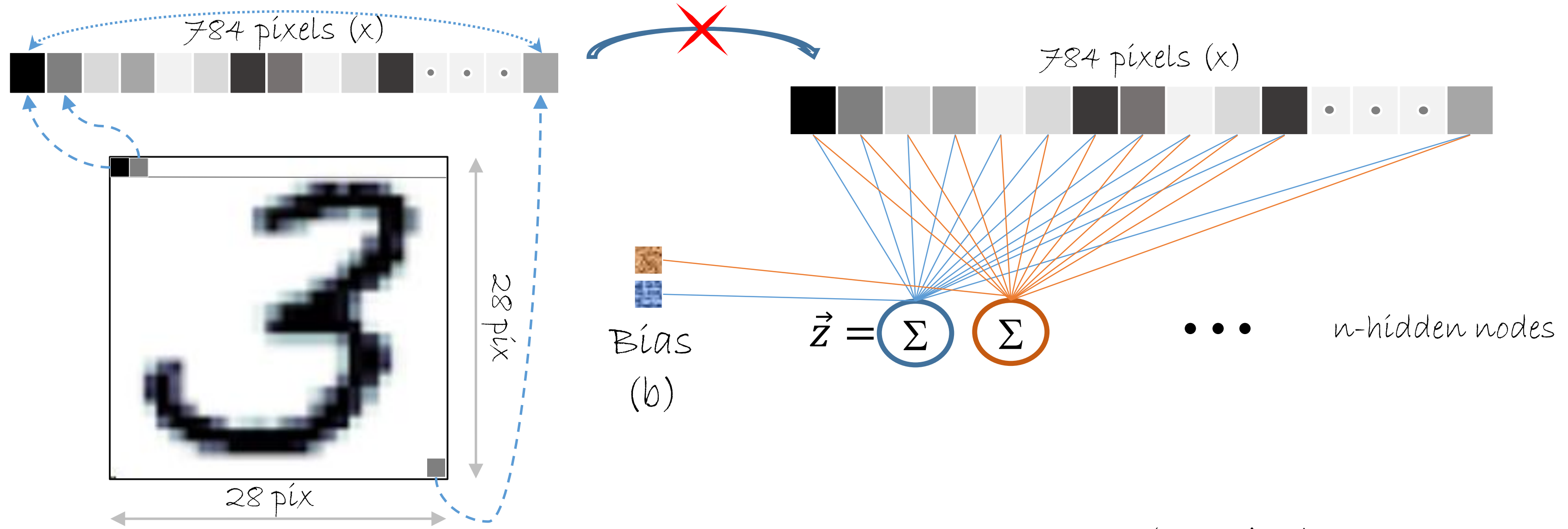
http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

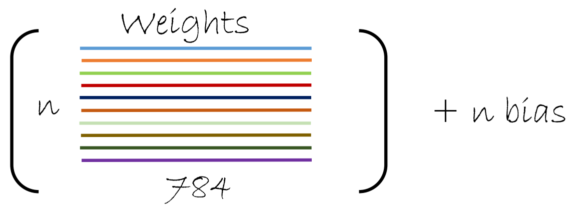
https://github.com/Microsoft/CNTK/wiki/Object-Detection-using-Fast-R-CNN

Multi-layer Perceptron

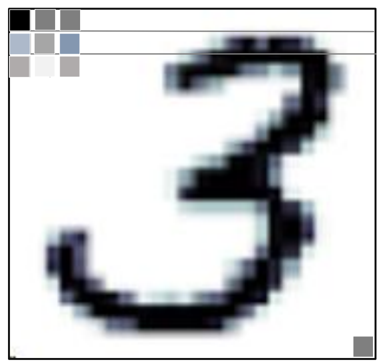


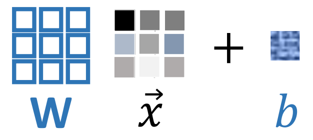
Fully Connected Networks



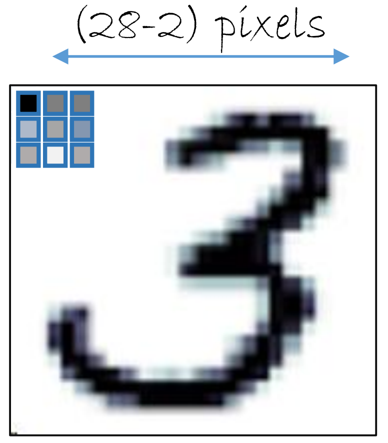


Total parameters: 784n + n





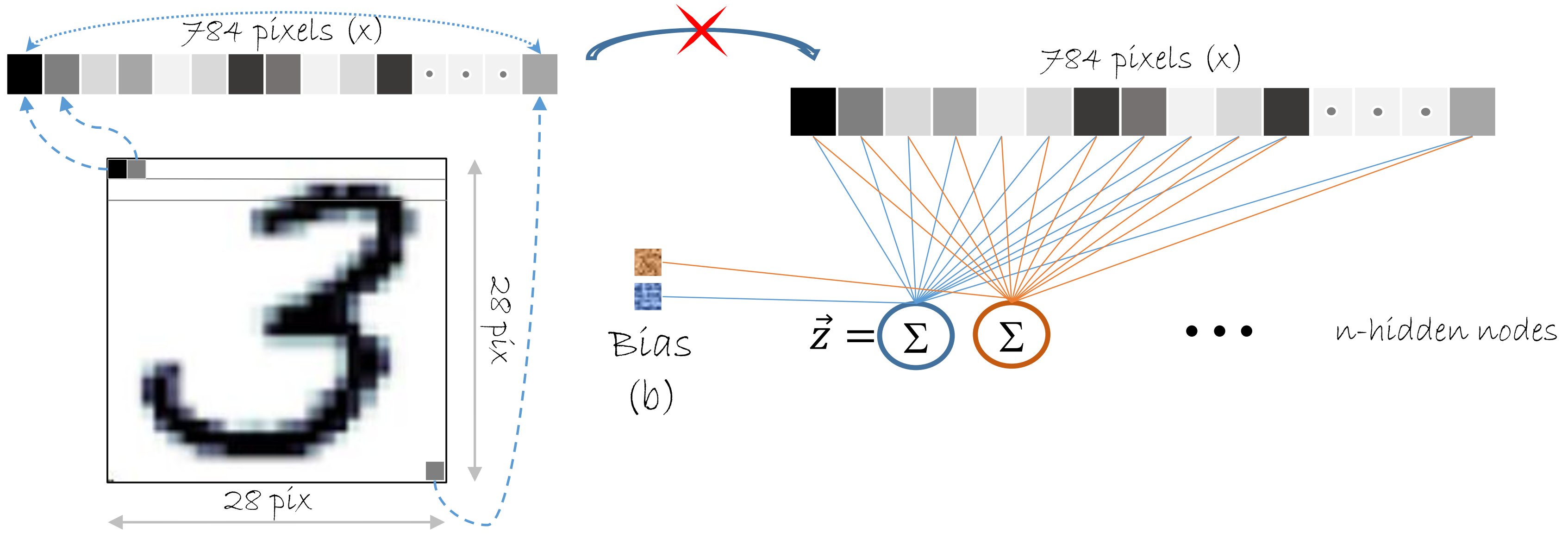
$$\vec{z} = \mathbf{W}\,\vec{x}^{T} + b$$

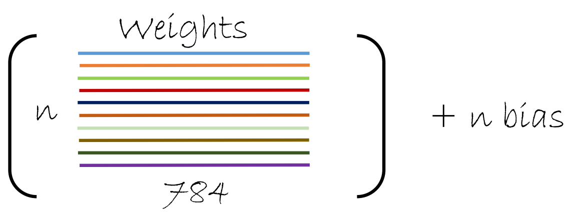


For 1 position: 3 x 3 + 1 = 10 parameters

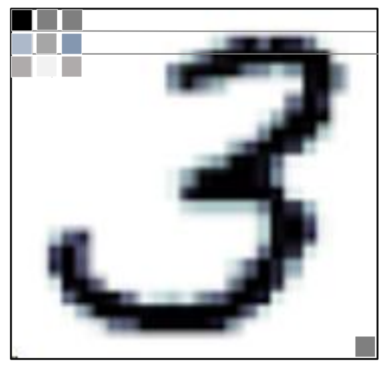
For all positions:
10 x (28-2) x (28-2)
= 6760 parameters

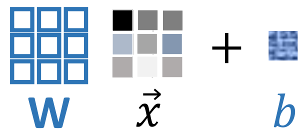
Fully Connected Networks





Total parameters: 784n + n

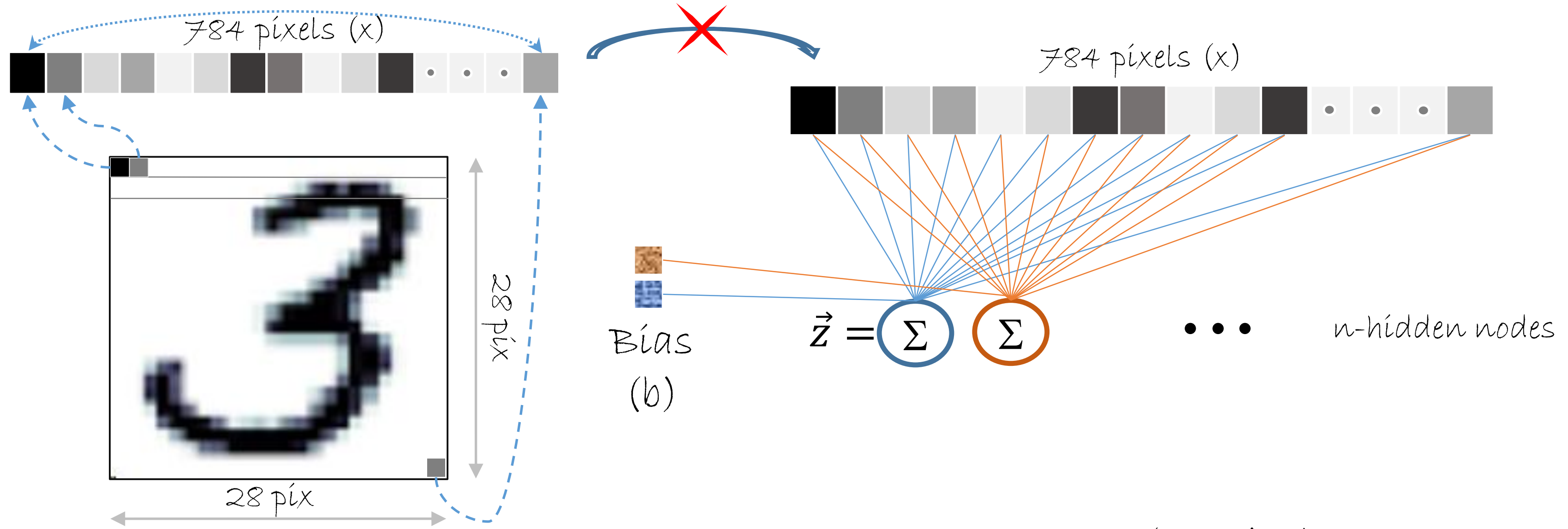


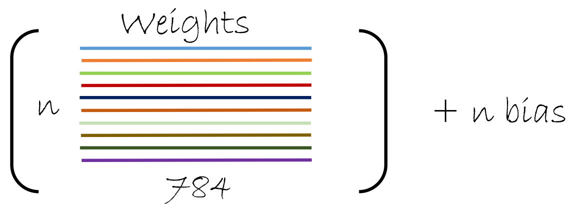


$$\vec{z} = \mathbf{W}\,\vec{x}^{T} + b$$

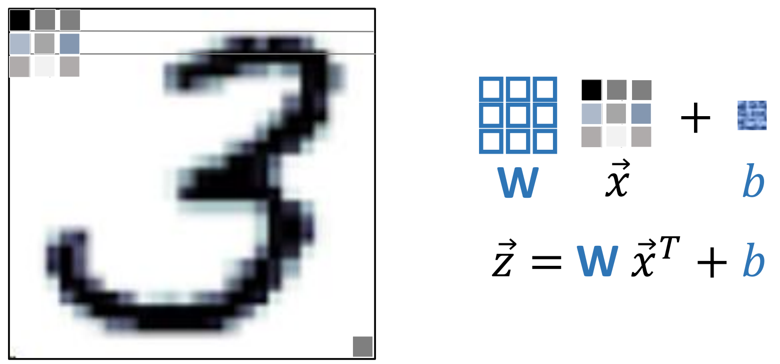
For 1 position: 3 x 3 + 1 = 10 parameters

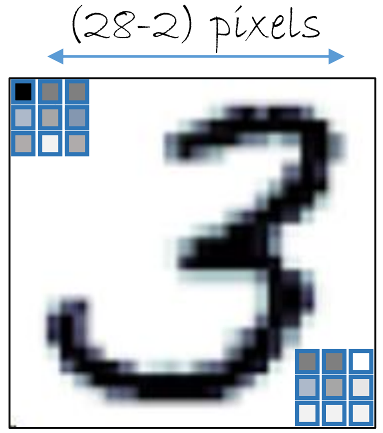
Fully Connected Networks





Total parameters: 784n + n

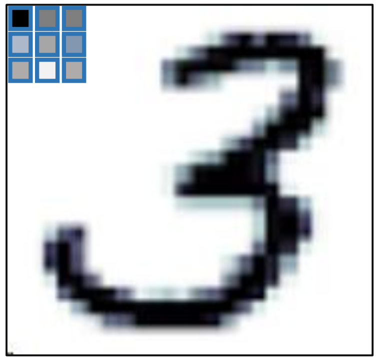


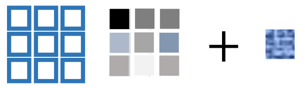


For 1 position: 3 x 3 + 1 = 10 parameters

For all positions with each having individual (W, b) :
10 x (28-2) x (28-2)
= 6760 parameters

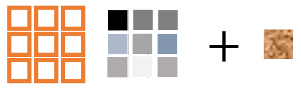
Convolution Networks





$\mathbf{W} \quad x \quad b$

$z = \mathbf{W}x + b$

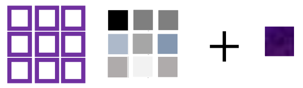


$\mathbf{W} \quad x \quad b$

$z = \mathbf{W}x + b$

. . .

n-filters



$\mathbf{W} \quad x \quad b$

$z = \mathbf{W}x + b$

W is called a filter: shape (3,3)

Total parameters: $9n + n$

With convolution (10 - 3 x 3 filters and 5 layers):

= 500 parameters

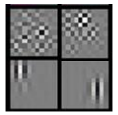
With larger image size:

Image size = 200 x 200 pixels
Filter size = 3 x 3 (**W** , b = 10 values)
Stride = 1
Layers = 5
Number of filters per layer = 20
Number of parameters =
10 x 5 x 20
1000

Allows for:
- ✓ Handling of larger image sizes (512 x 512)
- ✓ Trying larger filter sizes (11 x 11)
- ✓ Learning more filters (128 filters)
- ✓ Deeper architecture (152 layers)

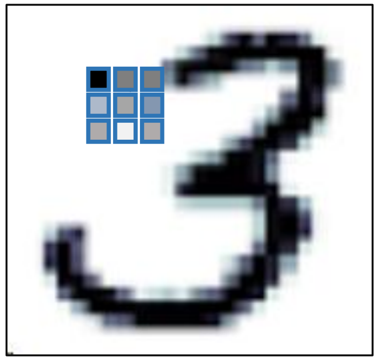
Primitive features such as edges (First few layers)

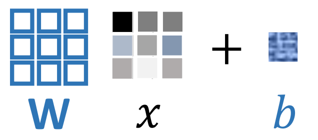
Color features (for color images)

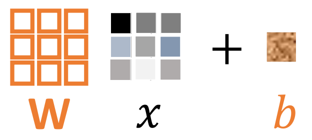


Complex features such as corners (Deeper layers)

Convolution Networks

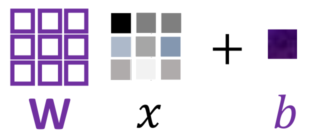



$z = \mathbf{W}\, x \quad + b$


$z = \mathbf{W}\, x \quad + b$

$\bullet\bullet\bullet$

n-filters


$z = \mathbf{W}\, x \quad + b$

W is called a filter: shape (3,3)

Total parameters: 9n + n

With convolution (10 - 3 x 3 filters and 5 layers):
= 500 parameters

With larger image size:
Image size = 200 x 200 pixels
Filter size = 3 x 3 (**W** , b = 10 values)
Stride = 1
Layers = 5
Number of filters per layer = 20
Number of parameters =
10 x 5 x 20
1000

Allows for:
- ✓ Handling of larger image sizes (512 x 512)
- ✓ Trying larger filter sizes (11 x 11)
- ✓ Learning more filters (128 filters)
- ✓ Deeper architecture (152 layers)

Primitive features such as edges (First few layers)

Color features (for color images)

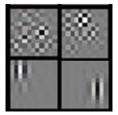
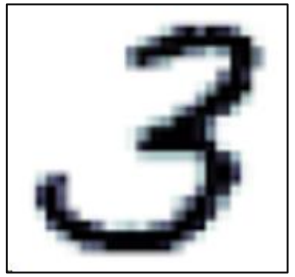
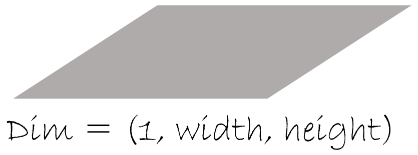

Complex features such as corners (Deeper layers)

Image Data

MNIST data

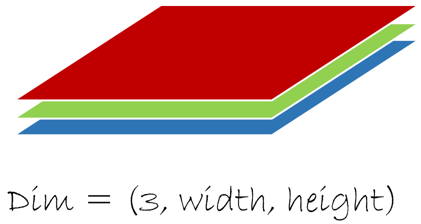
- Matrix of dimensions: 28 (width) x 28 (height) pixels
- Each pixel has 1 integer value

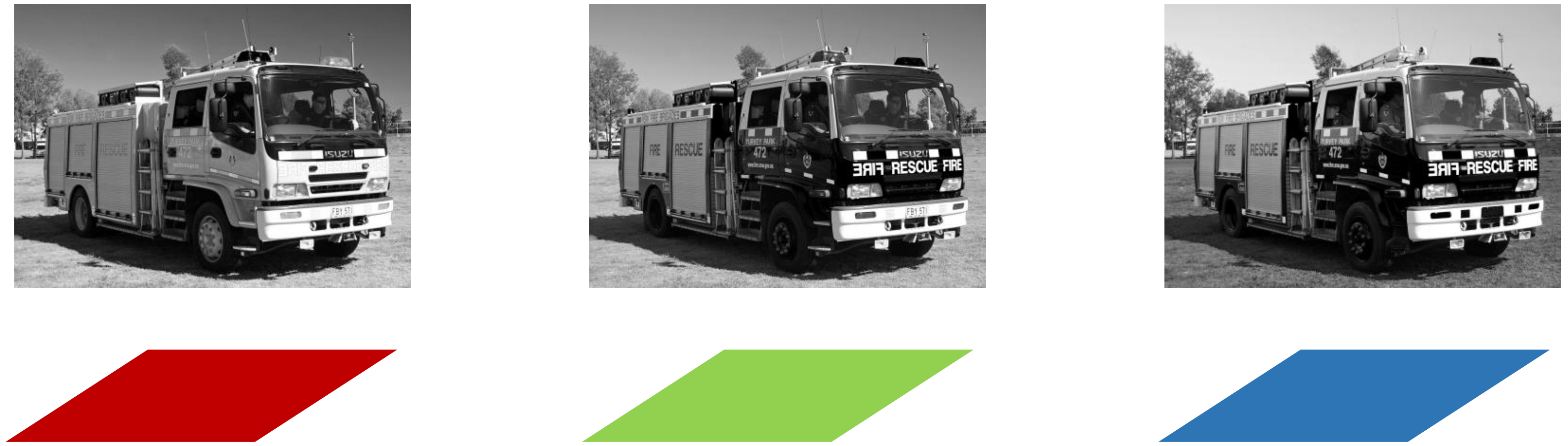




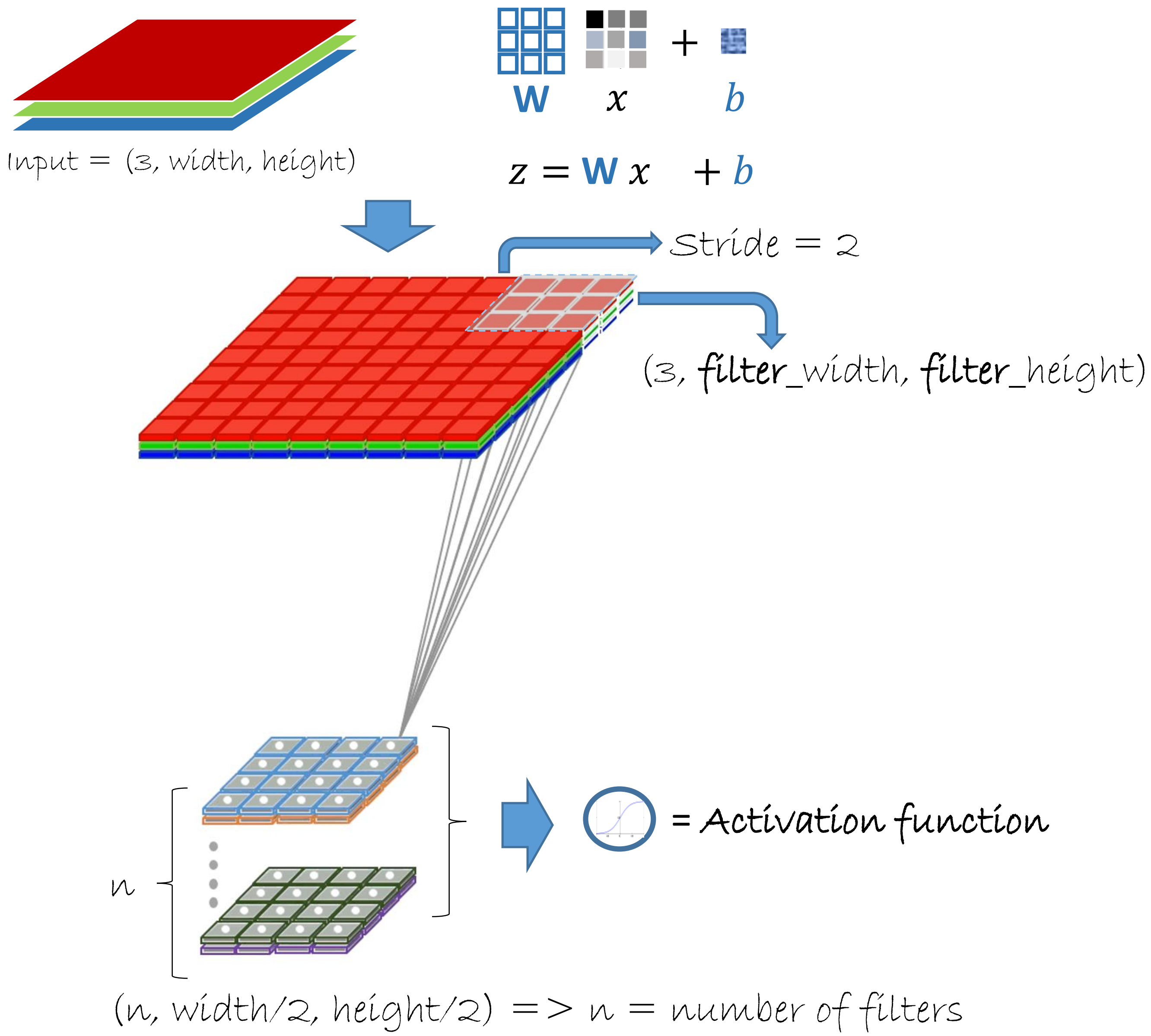
Natural scene images

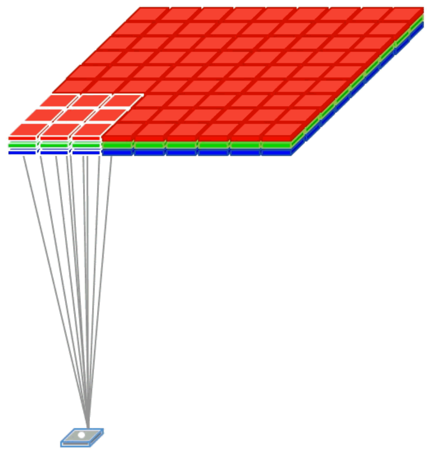
- Matrix of dimensions: width x height pixels
- Each pixel has 3 different integers,
- 1 each for Red, Green and Blue channels

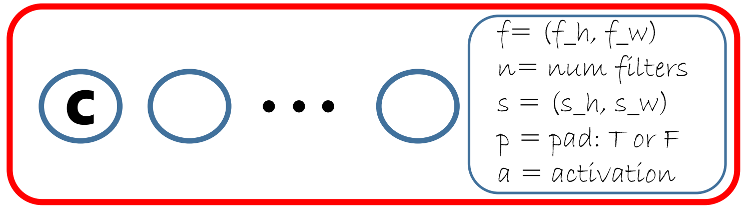




Convolution with Images

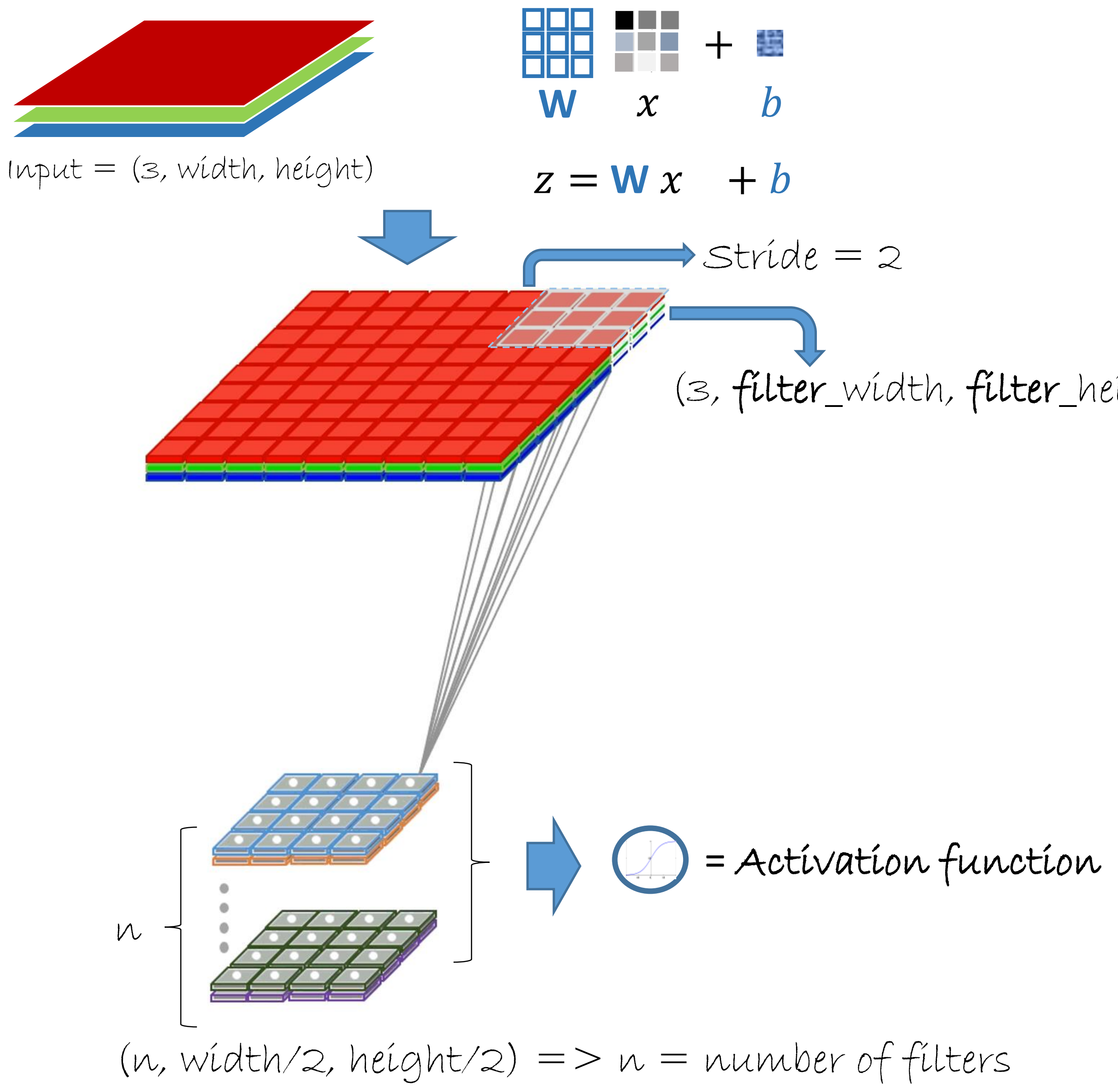

W
x
+ b
Input = (3, width, height)
z = W x + b
Stride = 2
(3, filter_width, filter_height)
= Activation function
n
(n, width/2, height/2) => n = number of filters

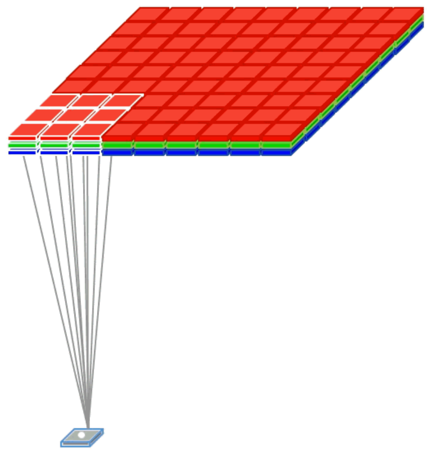


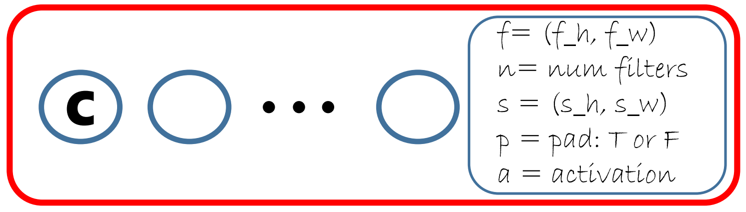


```
Convolution2D(filter_shape=(3,3),
              num_filters=8,
              strides=(2,2),
              pad=True,
              activation=relu)
```

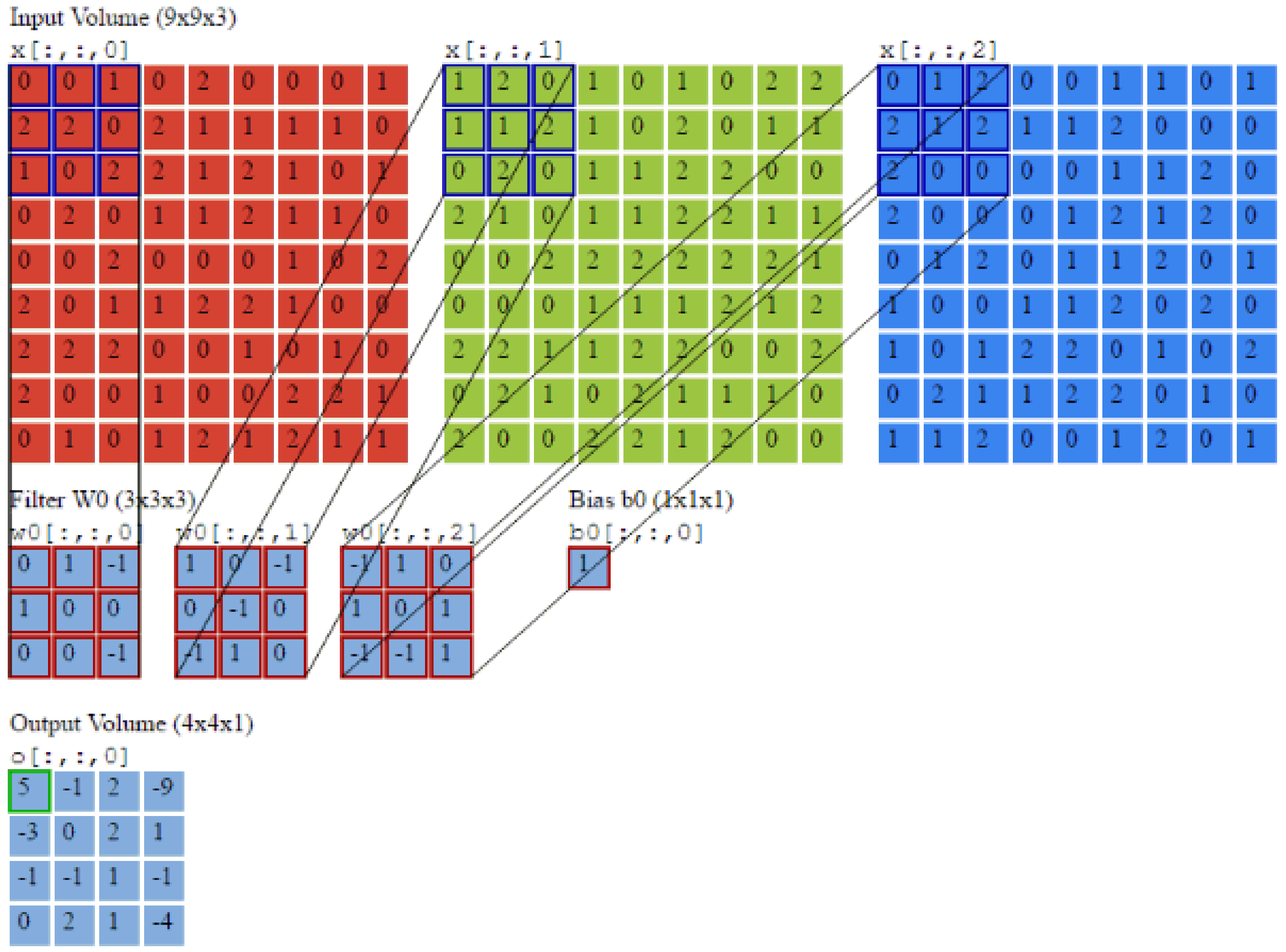
Convolution with Images


W x + b
Input = (3, width, height)
z = W x + b
Stride = 2
(3, filter_width, filter_height)
= Activation function
n
(n, width/2, height/2) => n = number of filters



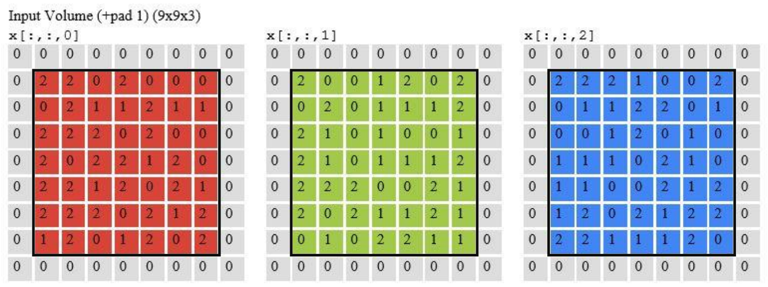


```
Convolution2D(filter_shape=(3,3),
              num_filters=8,
              strides=(2,2),
              pad=True,
              activation=relu)
```



Ref: http://cs231n.github.io/convolutional-networks/

No Padding vs Padding



Input Volume (+pad 1) (9x9x3)

x[:,:,0]

0	0	0	0	0	0	0	0	0
0	2	2	0	2	0	0	0	0
0	0	2	1	1	2	1	1	0
0	2	2	2	0	2	0	0	0
0	2	0	2	2	1	2	0	0
0	2	2	1	2	0	2	1	0
0	2	2	2	0	2	1	2	0
0	1	2	0	1	2	0	2	0
0	0	0	0	0	0	0	0	0

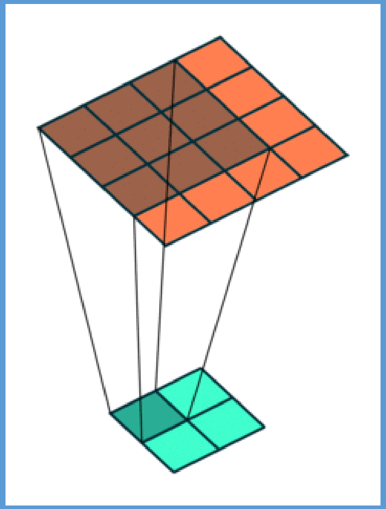
x[:,:,1]

0	0	0	0	0	0	0	0	0
0	2	0	0	1	2	0	2	0
0	0	2	0	1	1	1	2	0
0	2	1	0	1	0	0	1	0
0	2	1	0	1	1	1	2	0
0	2	2	2	0	0	2	1	0
0	2	0	2	1	1	2	1	0
0	0	1	0	2	2	1	1	0
0	0	0	0	0	0	0	0	0

x[:,:,2]

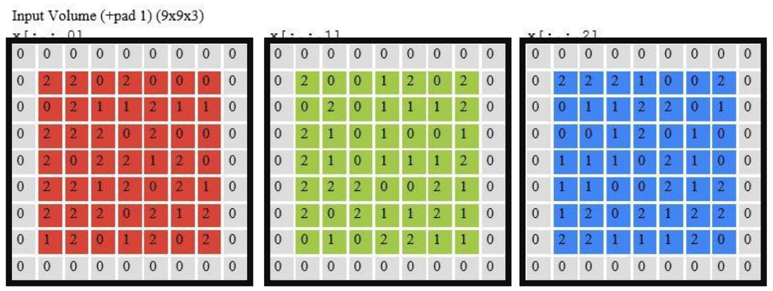
0	0	0	0	0	0	0	0	0
0	2	2	2	1	0	0	2	0
0	0	1	1	2	2	0	1	0
0	0	0	1	2	0	1	0	0
0	1	1	1	0	2	1	0	0
0	1	1	0	0	2	1	2	0
0	1	2	0	2	1	2	2	0
0	2	2	1	1	1	2	0	0
0	0	0	0	0	0	0	0	0

No Padding



Stride = 1

No Padding vs Padding



Input Volume (+pad 1) (9x9x3)

x[:, :, 0]

0	0	0	0	0	0	0	0	0
0	2	2	0	2	0	0	0	0
0	0	2	1	1	2	1	1	0
0	2	2	2	0	2	0	0	0
0	2	0	2	2	1	2	0	0
0	2	2	1	2	0	2	1	0
0	2	2	2	0	2	1	2	0
0	1	2	0	1	2	0	2	0
0	0	0	0	0	0	0	0	0

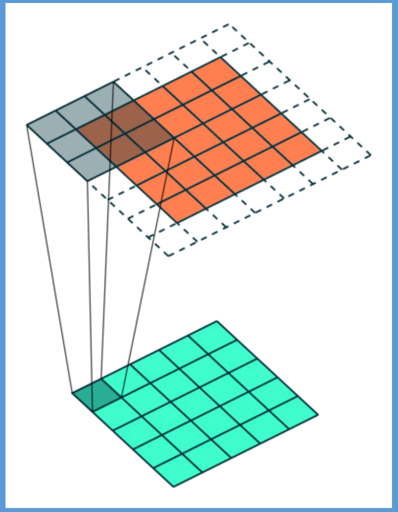
x[:, :, 1]

0	0	0	0	0	0	0	0	0
0	2	0	0	1	2	0	2	0
0	0	2	0	1	1	1	2	0
0	2	1	0	1	0	0	1	0
0	2	1	0	1	1	1	2	0
0	2	2	2	0	0	2	1	0
0	2	0	2	1	1	2	1	0
0	0	1	0	2	2	1	1	0
0	0	0	0	0	0	0	0	0

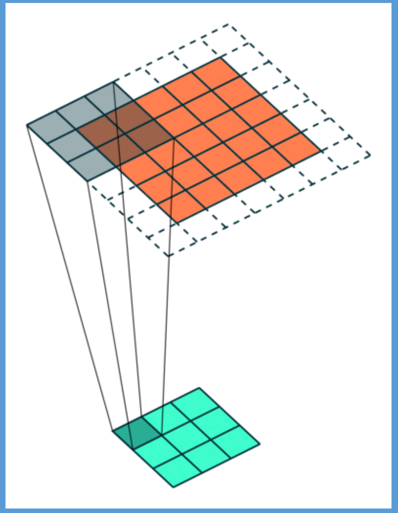
x[:, :, 2]

0	0	0	0	0	0	0	0	0
0	2	2	2	1	0	0	2	0
0	0	1	1	2	2	0	1	0
0	0	0	1	2	0	1	0	0
0	1	1	1	0	2	1	0	0
0	1	1	0	0	2	1	2	0
0	1	2	0	2	1	2	2	0
0	2	2	1	1	1	2	0	0
0	0	0	0	0	0	0	0	0

With Padding



Stride = 1



Stride = 2

Input Volume (+pad 1) (9x9x3)

x[:,:,0]

0	0	0	0	0	0	0	0	0
0	2	2	0	2	0	0	0	0
0	0	2	1	1	2	1	1	0
0	2	2	2	0	2	0	0	0
0	2	0	2	2	1	2	0	0
0	2	2	1	2	0	2	1	0
0	2	2	2	0	2	1	2	0
0	1	2	0	1	2	0	2	0
0	0	0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0	0	0
0	2	0	0	1	2	0	2	0
0	0	2	0	1	1	1	2	0
0	2	1	0	1	0	0	1	0
0	2	1	0	1	1	1	2	0
0	2	2	2	0	0	2	1	0
0	2	0	2	1	1	2	1	0
0	0	1	0	2	2	1	1	0
0	0	0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0	0	0
0	2	2	2	1	0	0	2	0
0	0	1	1	2	2	0	1	0
0	0	0	1	2	0	1	0	0
0	1	1	1	0	2	1	0	0
0	1	1	0	0	2	1	2	0
0	1	2	0	2	1	2	2	0
0	2	2	1	1	1	2	0	0
0	0	0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

0	1	1
0	1	-1
1	0	1

w0[:,:,1]

0	1	1
-1	1	1
-1	0	1

w0[:,:,2]

-1	0	0
0	0	-1
1	1	1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

1	0	0
1	1	1
0	1	1

w1[:,:,1]

-1	0	-1
-1	1	1
1	-1	0

w1[:,:,2]

1	0	0
0	1	0
-1	1	1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (4x4x2)

o[:,:,0]

6	5	8	4
11	7	11	7
14	11	12	5
5	3	6	4

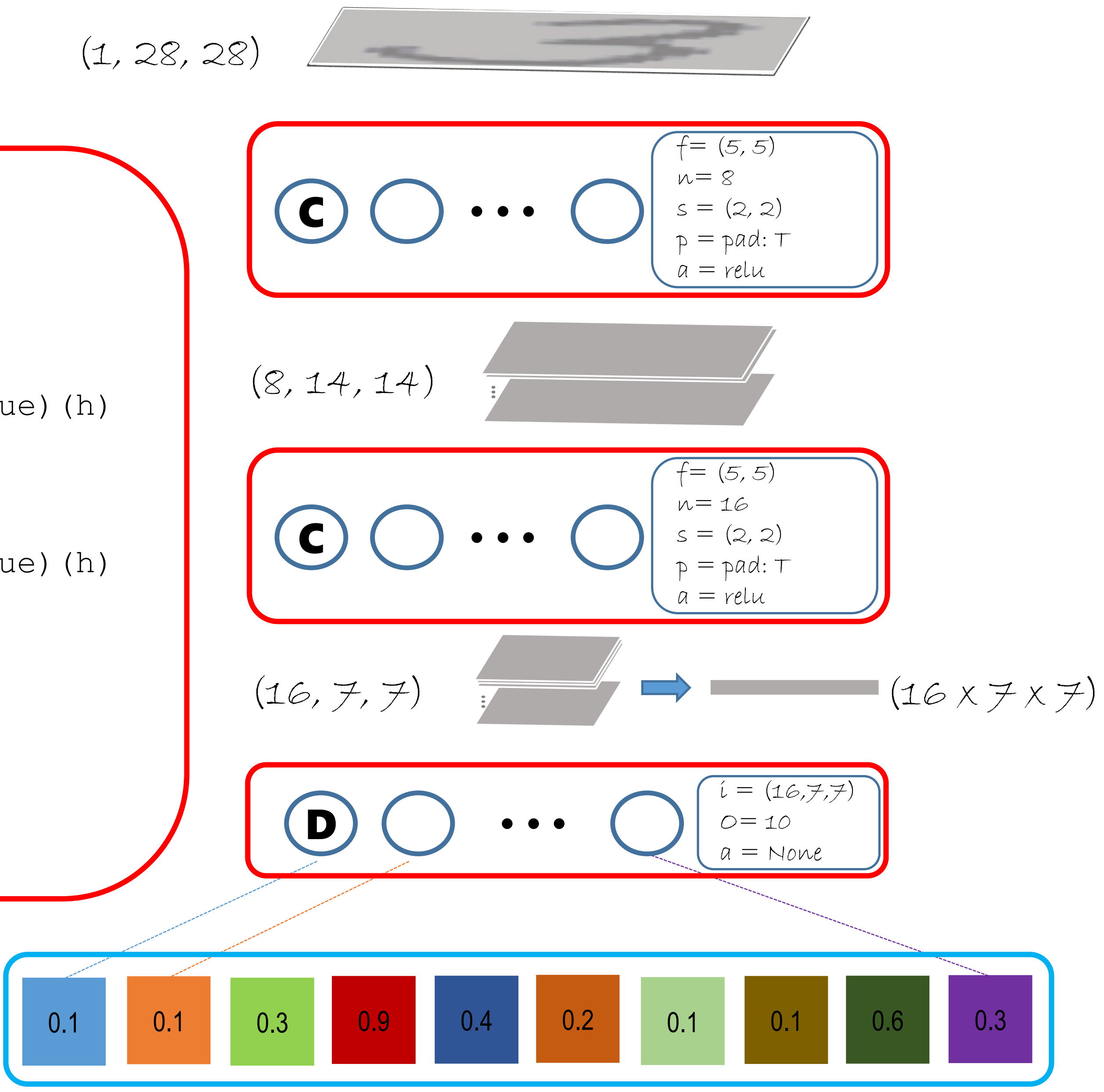
o[:,:,1]

11	13	6	5
7	10	8	-1
13	4	12	9
6	8	4	3

Ref:
http://cs231n.github.io/convolutional-networks/

Pure Convolution Network

```
def create_model(features):
    with default_options(activation = relu):
        h = features
        h = Convolution2D(filter_shape=(5,5),
                          num_filters=8,
                          strides=(2,2), pad=True)(h)

        h = Convolution2D(filter_shape=(5,5),
                          num_filters=16,
                          strides=(2,2), pad=True)(h)

        r = Dense(num_output_classes,
                  activation = None)(h)
        return r

z = create_model(input)
```



% Error with MNIST Data = 1.56%

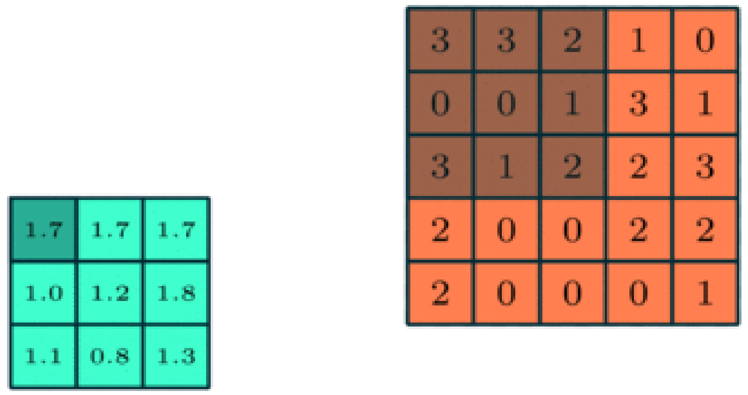
Pooling

Typically inserted in-between successive Convolution layers

Goal is to reduce number of parameters

- ✓ Control overfitting

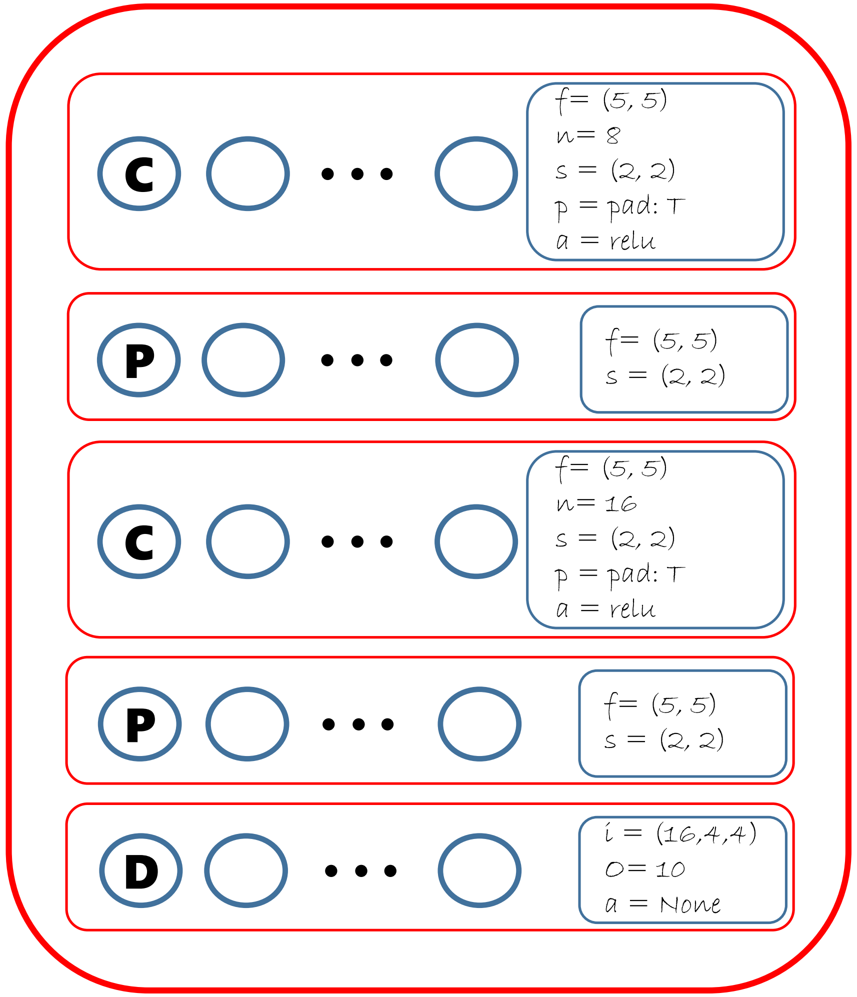
Popular pooling options



Average pooling

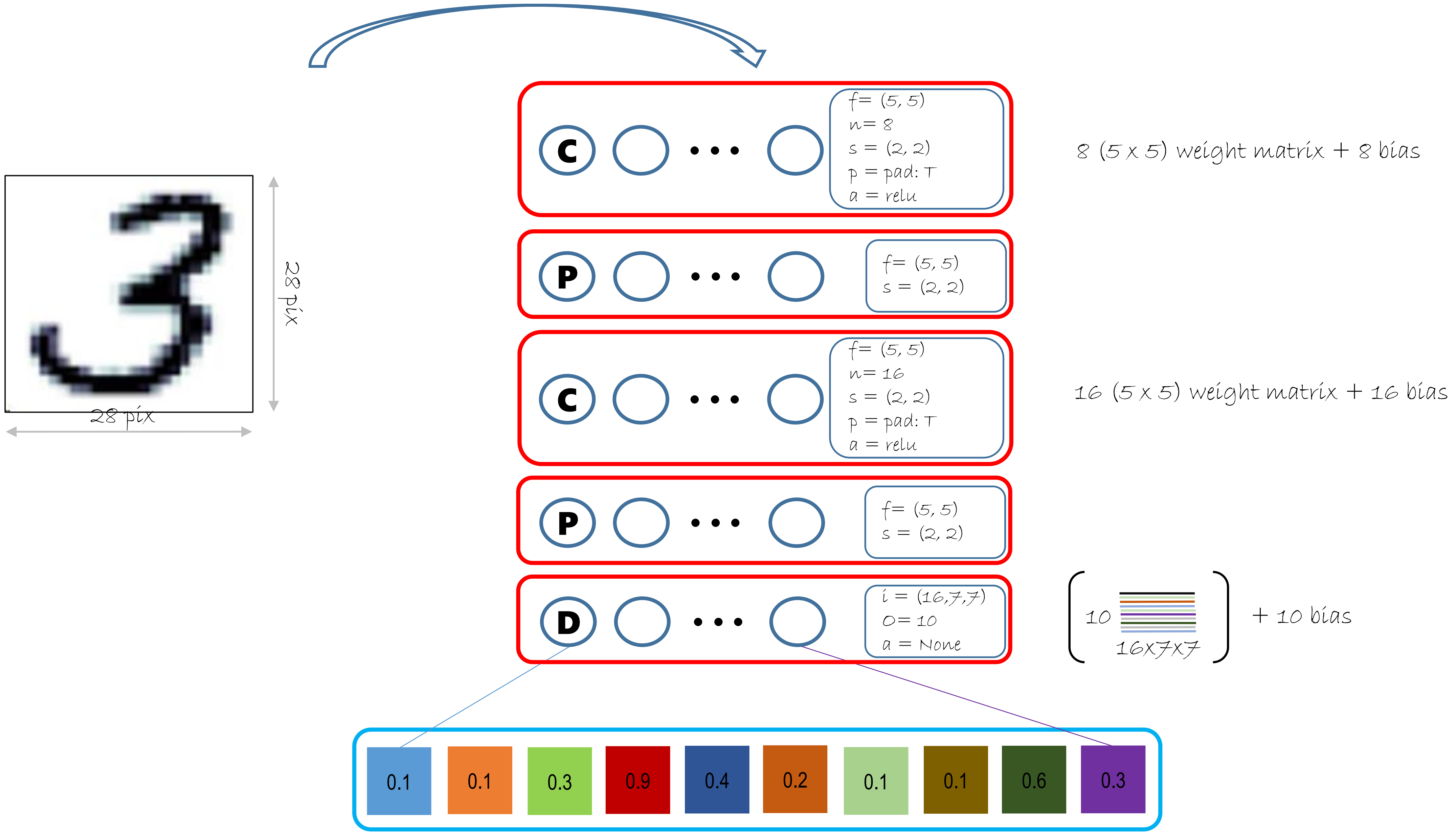
Typical Convolution Network

```
def create_model(features):
    with default_options(activation = relu):
        h = features
        h = Convolution2D(filter_shape=(5,5),
                          num_filters=8,
                          strides=(1,1), pad=True)(h)

        h = MaxPooling(filter_shape=(2,2),
                       strides=(2,2))(h)

        h = Convolution2D(filter_shape=(5,5),
                          num_filters=16,
                          strides=(2,2), pad=True)(h)

        h = MaxPooling(filter_shape=(2,2),
                       strides=(2,2))(h)

        r = Dense(num_output_classes,
                  activation = None)(h)
        return r

z = create_model(input)
```

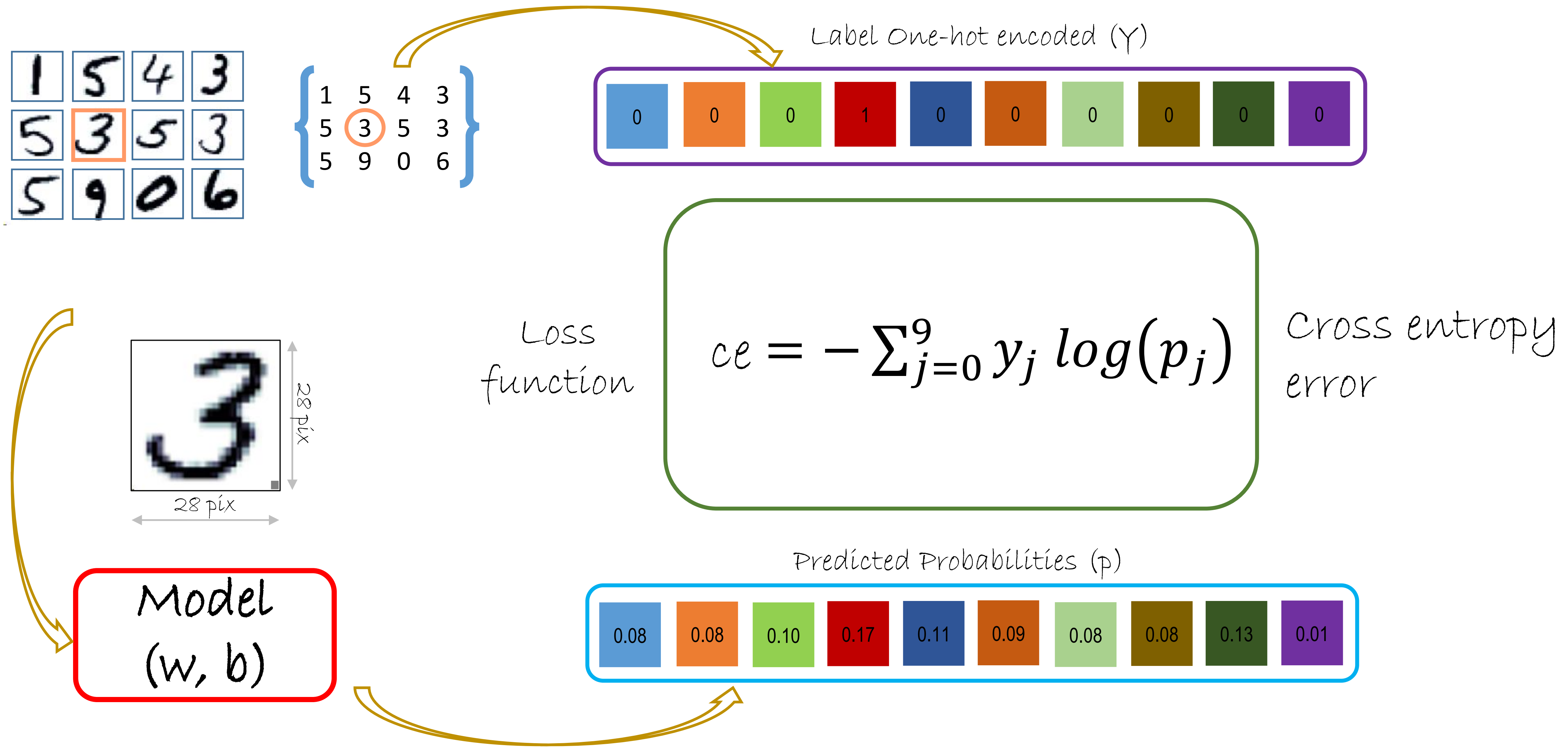


% Error with MNIST Data =~ 1%

Convolution Workflow

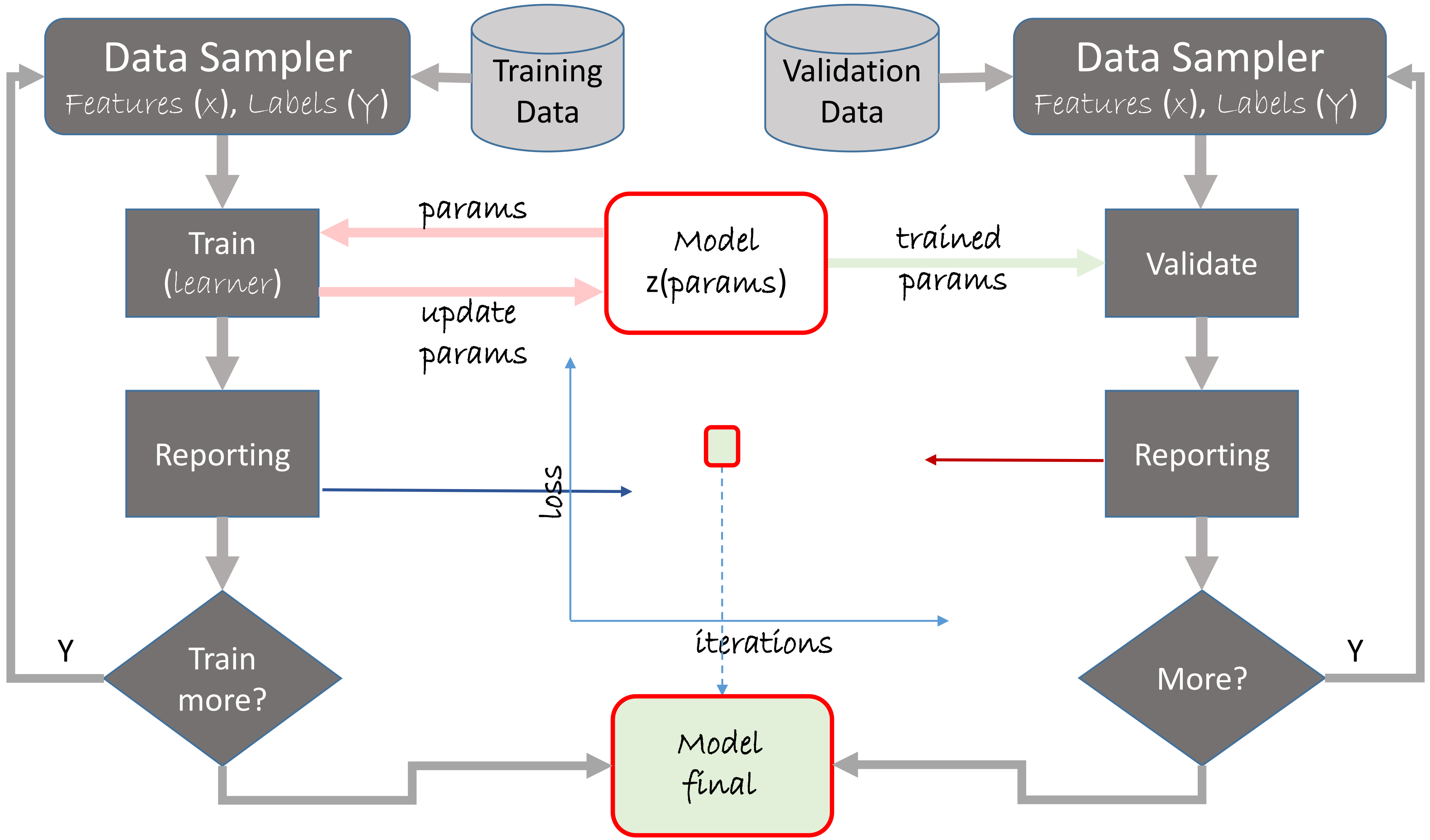


Error or Loss Function



$$ce = -\sum_{j=0}^{9} y_j \, log(p_j)$$

Train / Validation Workflow

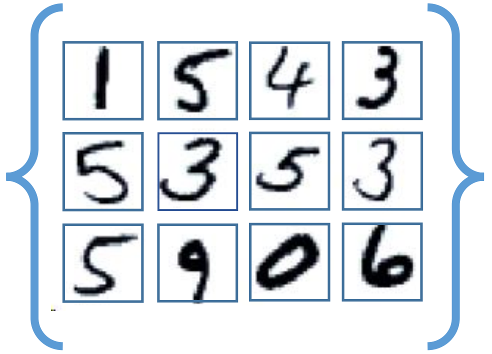


Train Workflow

Weights	8 (5 x 5)	16 (5 x 5)	10 16x7x7
	+	+	+
bias	8	16	10

Model Parameters

Input feature (X: 128 x 1 x 28 x 28)



Model

```
z = model(X):
        h = Convolution2D((5,5),filt=8, …)(X)
        h = MaxPooling(…)(h)
        h = Convolution2D ((5,5),filt=16, …)((h)
        h = MaxPooling(…)(h)
        r = Dense(output_classes, act= None)(h)
        return r
```

128 samples (mini-batch)

0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0
⋮									
⋮									
⋮									
1	0	0	0	0	0	0	0	0	0

One-hot encoded Label (Y: 128 x 10)

Loss `cross_entropy_with_softmax(z,Y)`

Error `classification_error(z,Y)`

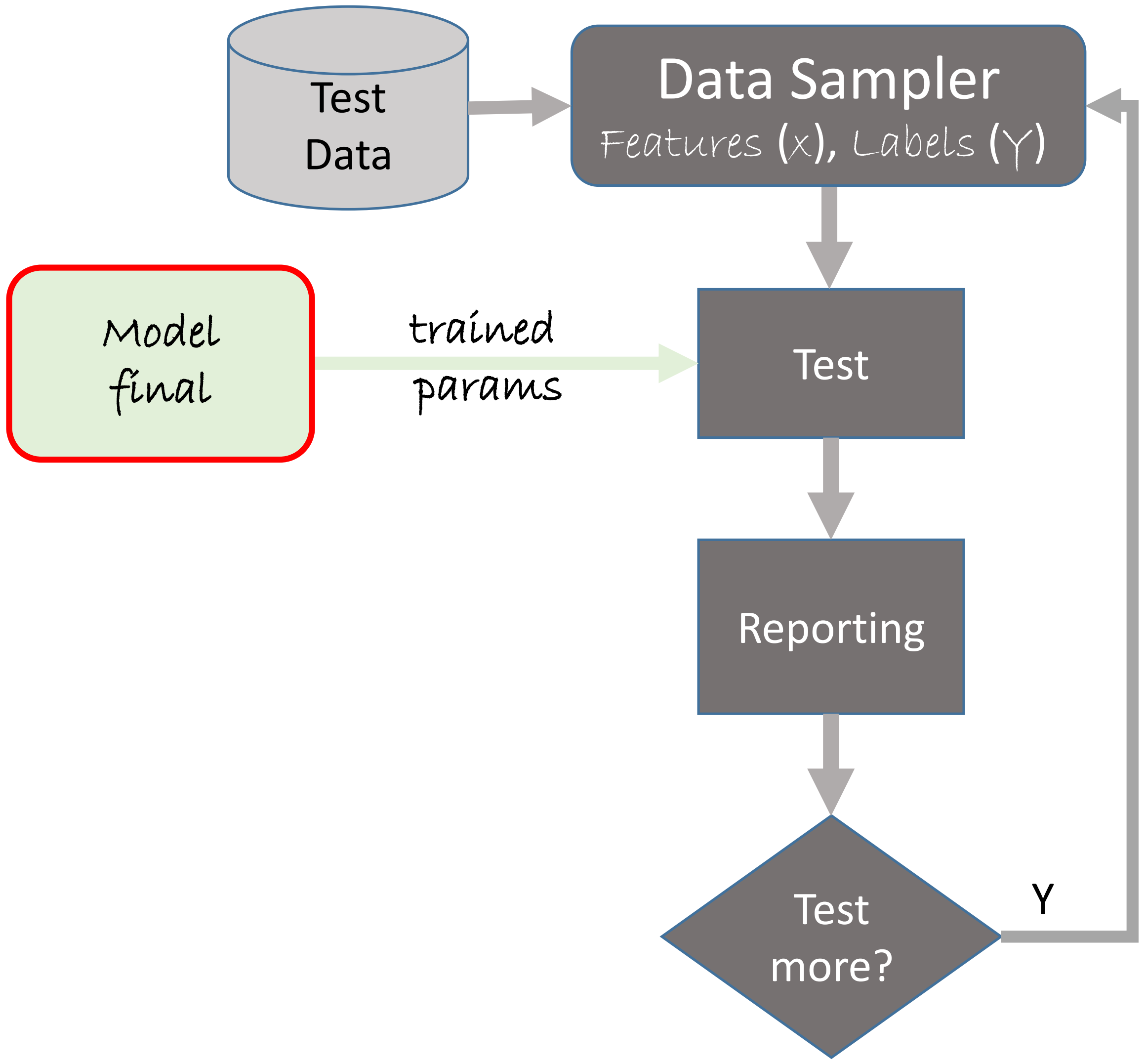
`Trainer(`*model, (loss, error), learner*`)`

`Trainer.`**`train`**`_minibatch({X, Y})`

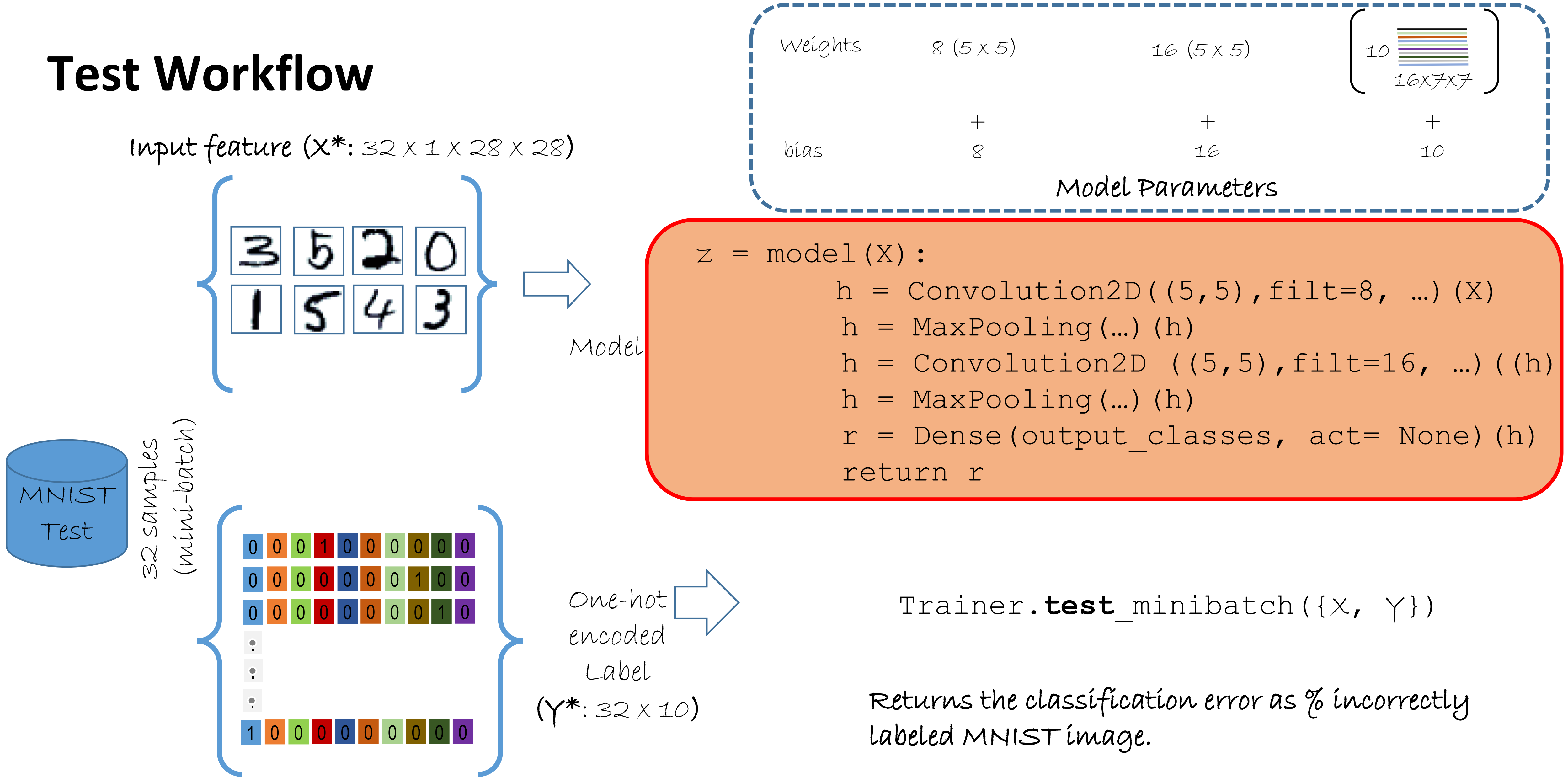
Learner

sgd, adagrad etc, are solvers to estimate – W & b

Test Workflow



Test Workflow

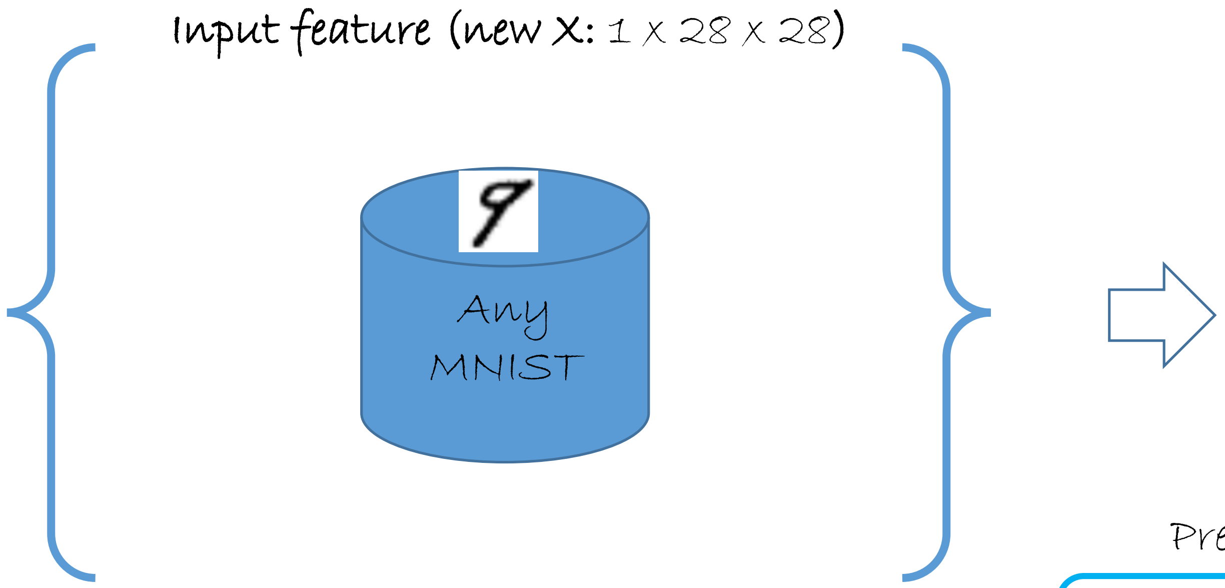


```
z = model(X):
       h = Convolution2D((5,5),filt=8, …)(X)
       h = MaxPooling(…)(h)
       h = Convolution2D ((5,5),filt=16, …)((h)
       h = MaxPooling(…)(h)
       r = Dense(output_classes, act= None)(h)
       return r
```

Trainer.**test**_minibatch({X, Y})

Returns the classification error as % incorrectly labeled MNIST image.

Prediction Workflow



Model
(w, b)

Model.eval(new X)

Predicted Softmax Probabilities (predicted_label)

0.02	0.09	0.03	0.03	0.01	0.02	0.02	0.06	0.02	0.70

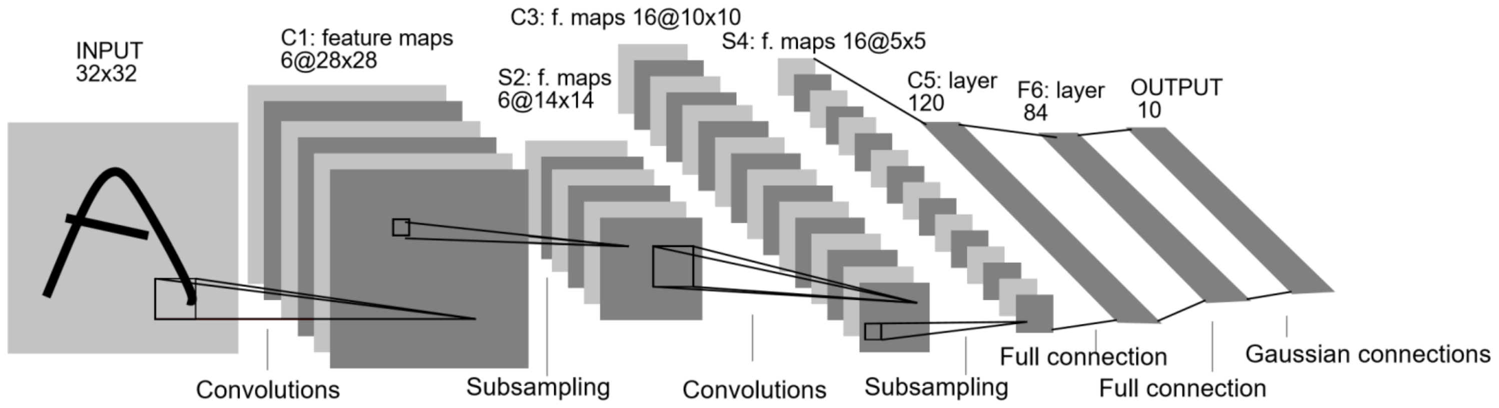
[numpy.argmax(predicted_label) for predicted_label in predicted_labels]

[9]

Popular Convolution Networks

LeNet

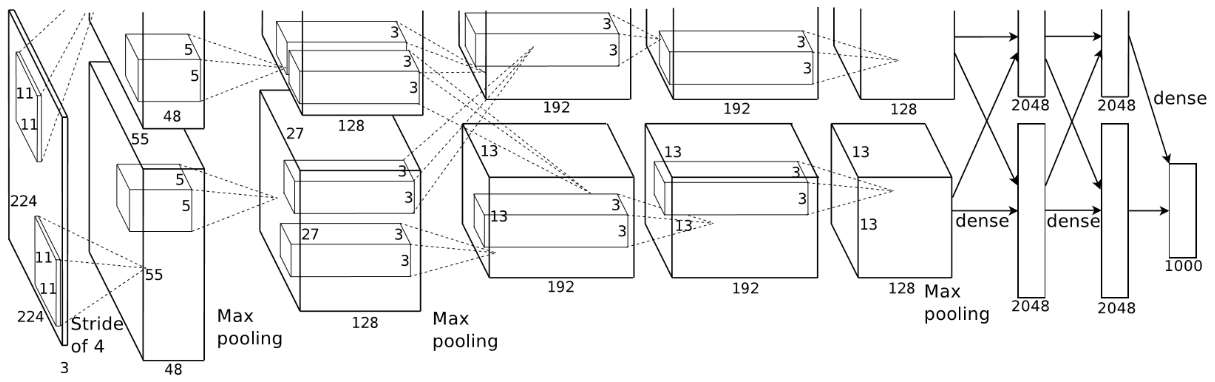
- First successful CNN by Yann LeCun in 1990
- Used to read zip codes / digits

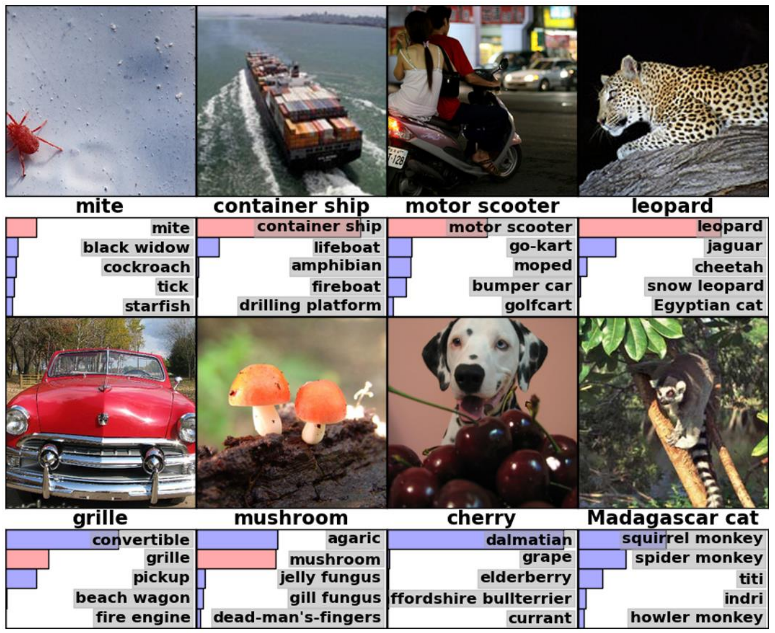


Popular Convolution Networks

AlexNet

- Popularized conv nets by Alex Krishevsky, Ilya Sutskever and Geoff Hinton
- In 2012 ImageNet ISLVRC challenge:
  - outperformed then state-of-the-art by reducing the error from 26% to 16%
  - First introduced the use of deeper, bigger stacked convolutional layers

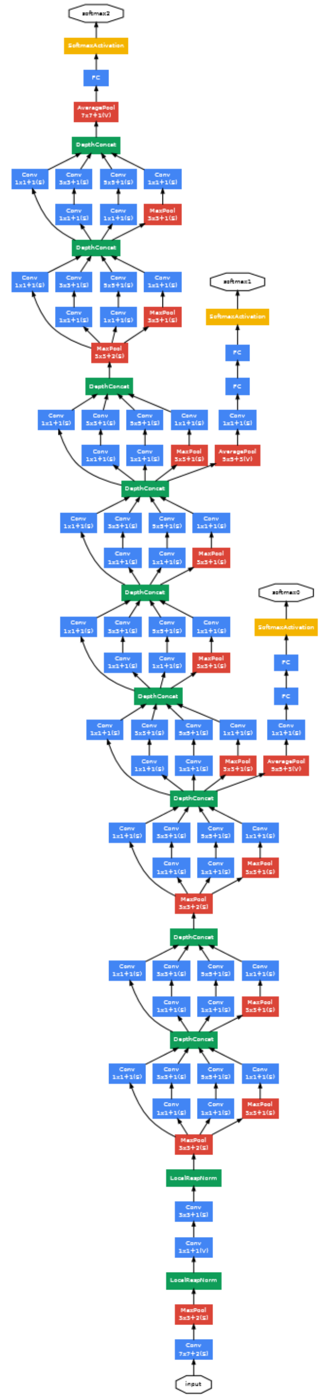




Popular Convolution Networks

GoogLeNet

- ILSVRC 2014 winner by Szegedy et al from Google
- Introduced the inception module
- Reduced the parameters dramatically from 60M in AlexNet to 4M
- Uses Average-pooling instead of fully connected layers



https://arxiv.org/abs/1409.4842

Popular Convolution Networks

VGGNet

- ILSVRC 2014 runner-up by Simoyan and Zisserman
  - Showed depth of network is key to performance
  - 16 CONV/FC layers and extremely homogeneous architecture (with end-to-end having only 3x3 convolutions and 2x2 pooling)
  - It is more expensive to evaluate and requires large memory
    - It has 140M compared to 60M AlexNet
    - Most parameters are in fully connected layers (which when removed do not cause significant performance drop

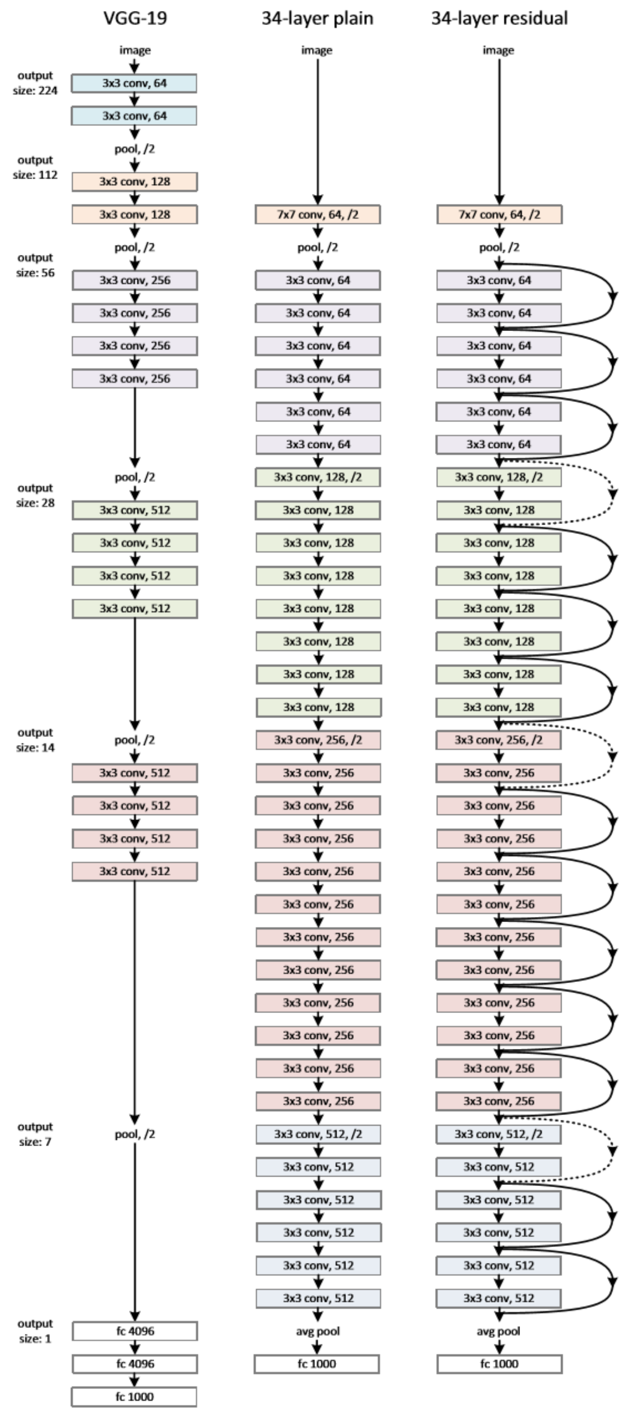
http://www.robots.ox.ac.uk/~vgg/research/very_deep/

Popular Convolution Networks

ResNet

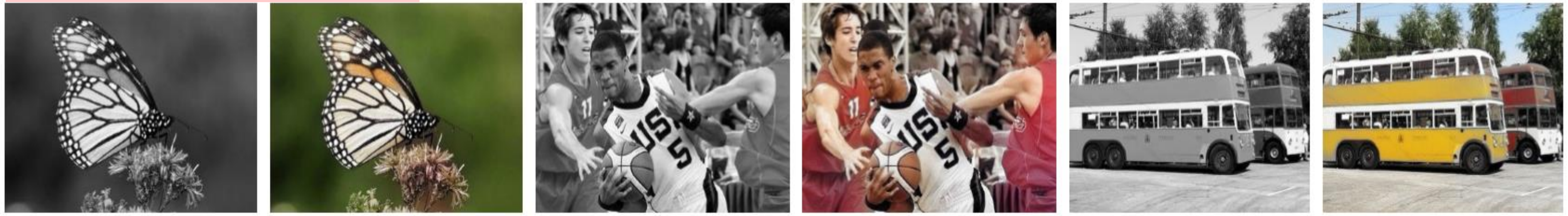
- ILSVRC 2015 winner by Kaiming He et al from Microsoft
- State-of-the-art (May 2016) and default choice
- Original implementation has 152 layers
- Introduces the concept of residual learning

https://arxiv.org/abs/1512.03385



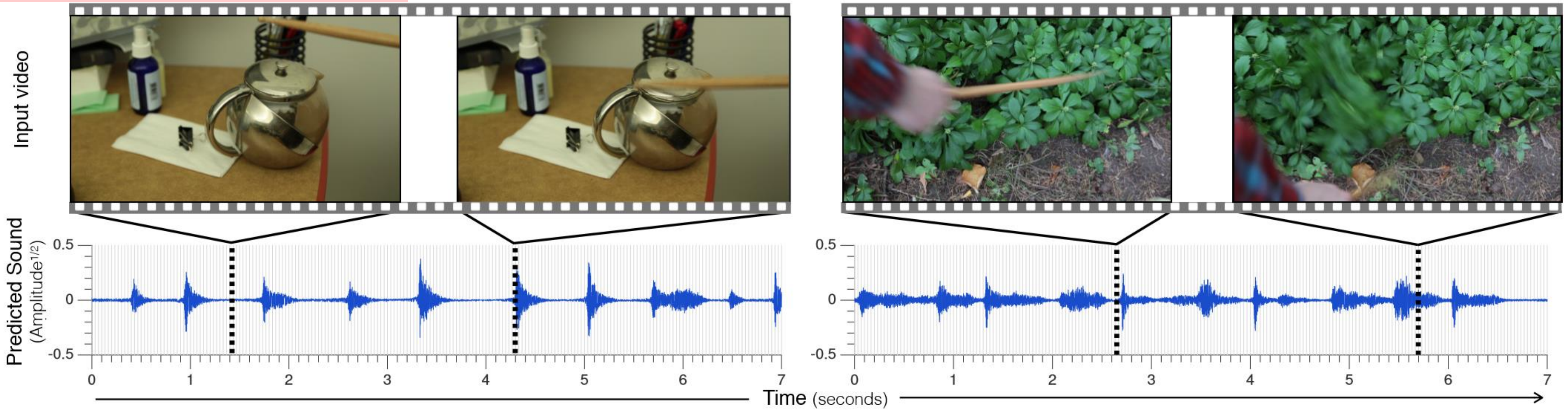
Applications of Conv Nets

Coloring grayscale images



https://arxiv.org/pdf/1603.06668.pdf

Visually indicated sounds



https://arxiv.org/pdf/1512.08512.pdf

Applications of Conv Nets

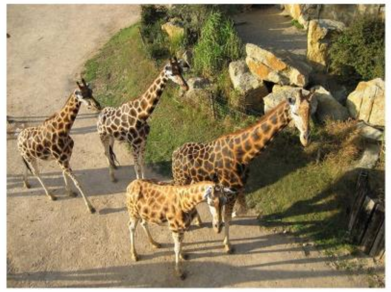
Automated image captioning

A person on a beach flying a kite.

A black and white photo of a train on a train track.

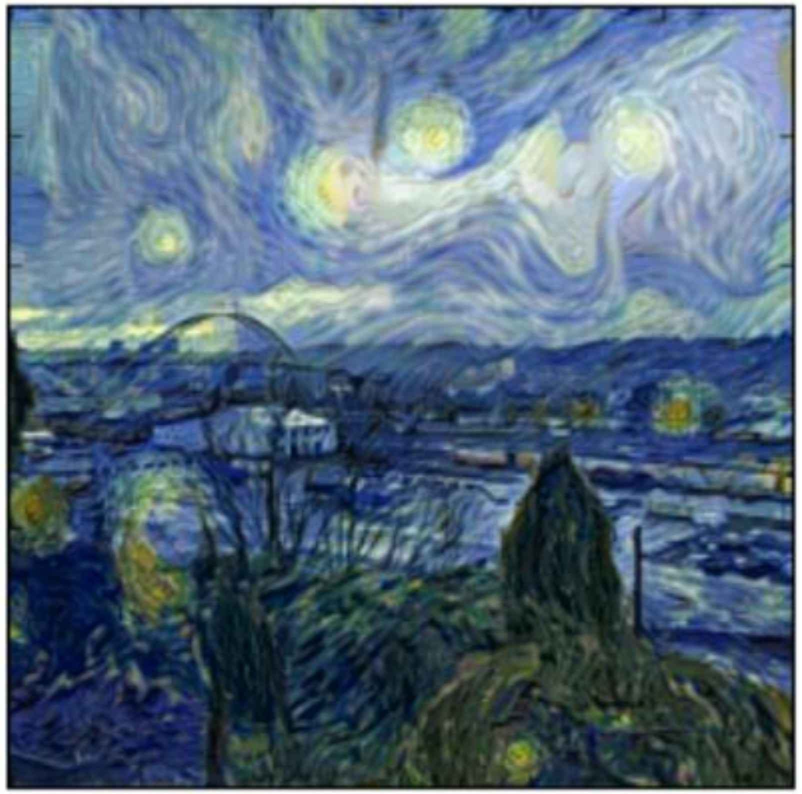
A person skiing down a snow covered slope.

A group of giraffe standing next to each other.



https://arxiv.org/abs/1411.4555

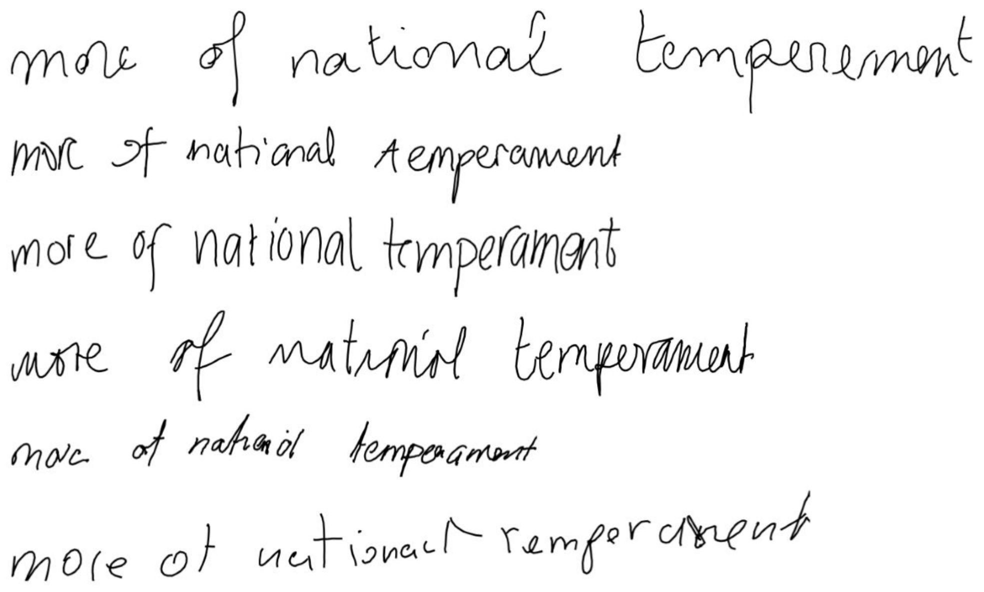
Neuro artistic painting



https://github.com/Microsoft/CNTK/blob/master/Tutorials/CNTK_205_Artistic_Style_Transfer.ipynb

Applications of Conv Nets

Handwriting generation


more of national temperement

morc of national temperament

more of national temperament

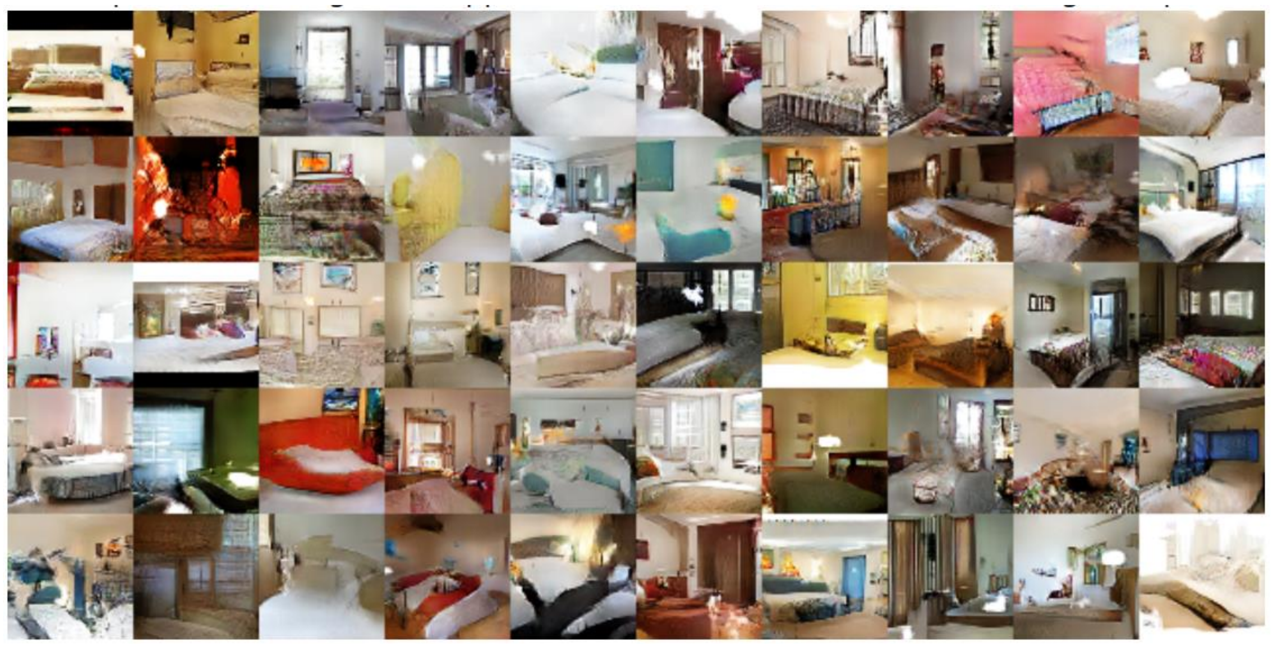
more of national temperament

more of nahonil temperament

more of national remperament

https://arxiv.org/pdf/1308.0850v5.pdf

Image generation



https://github.com/Newmu/dcgan_code

Conclusion

CNNs are widely used in computer vision with increasing popularity for text processing

Convolutions allow for deeper architectures that affects performance

Different CNNs of varying complexity can be easily be built using Cognitive Toolkit (layers library)