DS 4400

Machine Learning and Data Mining I
Spring 2021

Alina Oprea
Associate Professor
Khoury College of Computer Science
Northeastern University

March 30 2021

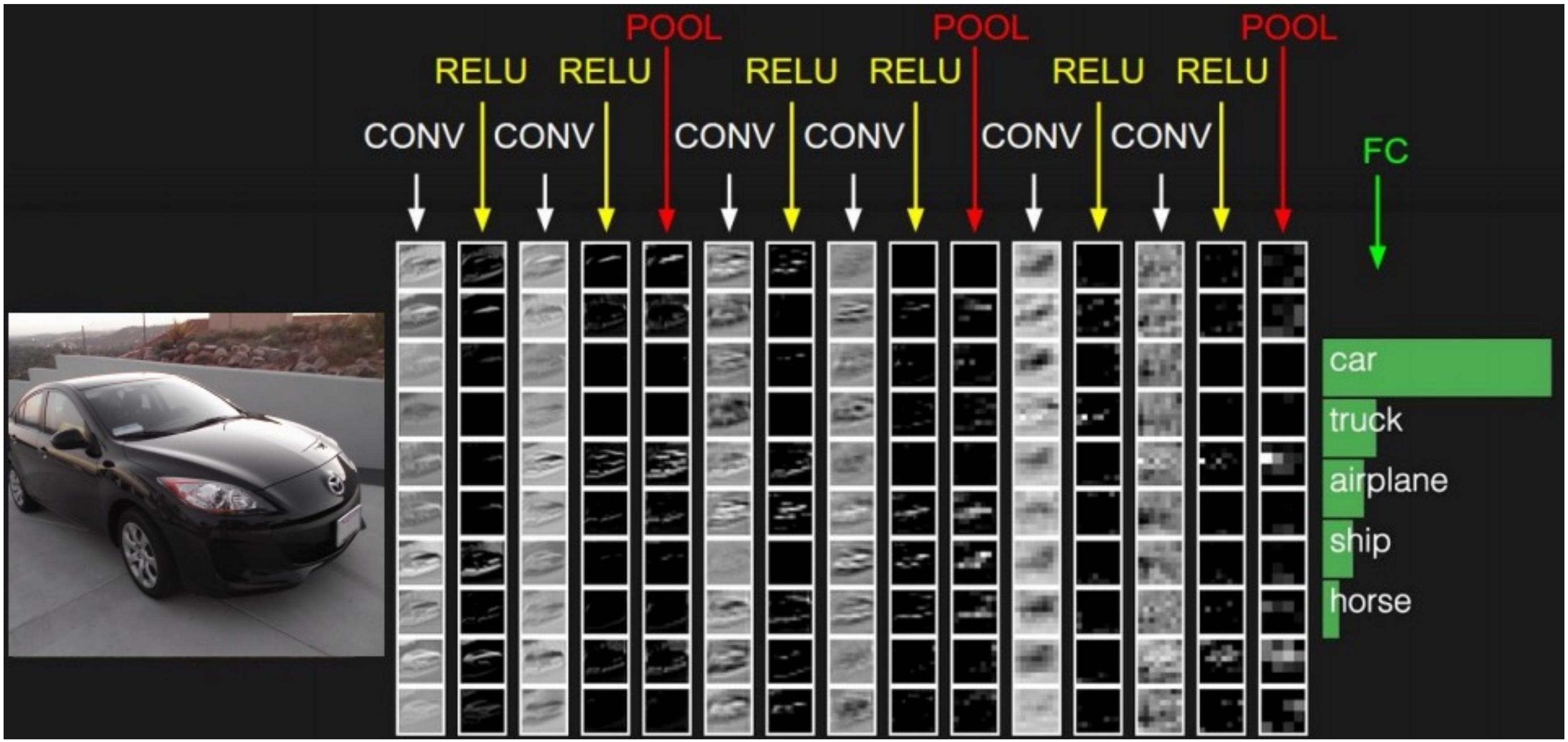
Outline

- Convolutional neural networks
  - Max pooling
  - Estimating parameters
- Architectures for convolutional networks
- Lab in Keras on convolutional networks
- Representing Boolean functions
- Regularization
- Transfer learning

Convolutional Nets

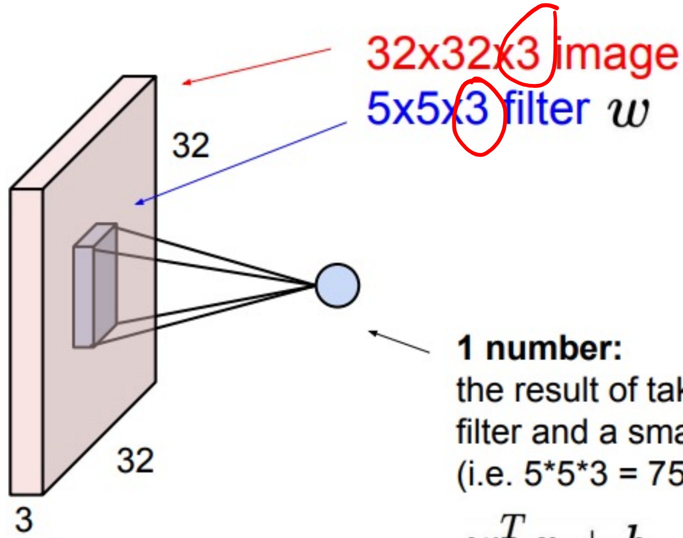
- Particular type of Feed-Forward Neural Nets
  - Invented by [LeCun 89]
- Applicable to data with natural grid topology
  - Time series
  - Images
- Use convolutions on at least one layer
  - Convolution is a linear operation that uses local information
  - Also use pooling operation
  - Used for dimensionality reduction and learning hierarchical feature representations

Convolutional Nets



Convolution Layer

w, b = PARAMS LEARNED DURING TRAINING

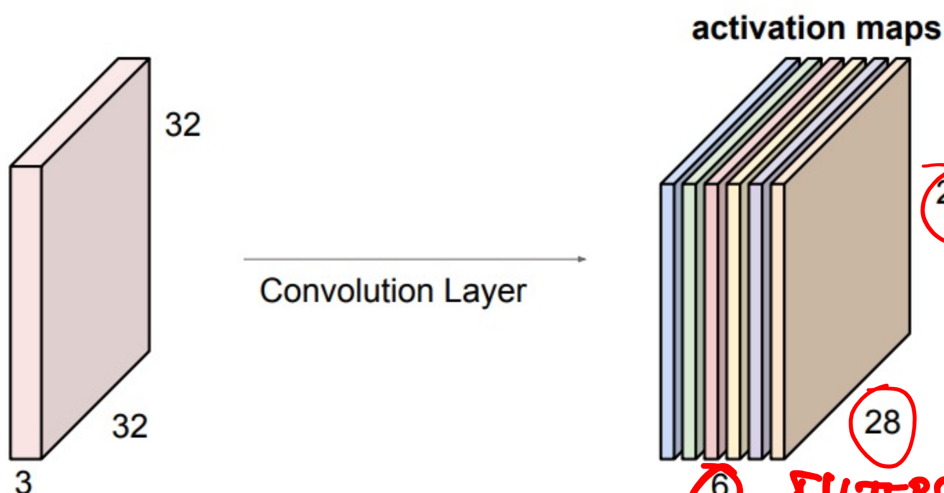


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

76

$$w^T x + b$$

LINEAR

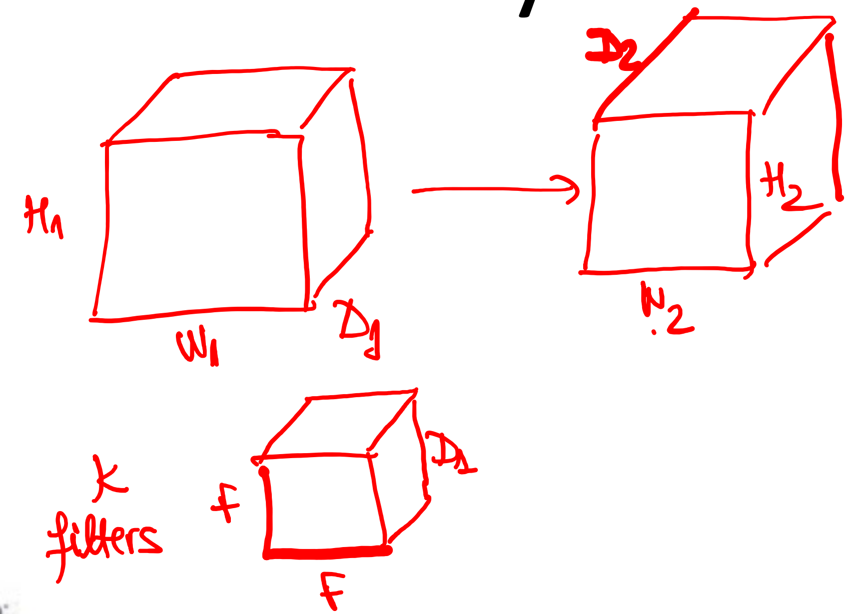


CONV

stride=1

FILTERS

Summary: Convolution Layer



Summary. To summarize, the Conv Layer: INPUT

- Accepts a volume of size $\boxed{W_1 \times H_1 \times D_1}$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $\boxed{W_2 \times H_2 \times D_2}$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$ # FILTERS
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

EACH FILTER: $F \times F \times D_1$ 1 bias

K FILTERS: $K \cdot (F \times F \times D_1)$ K biases

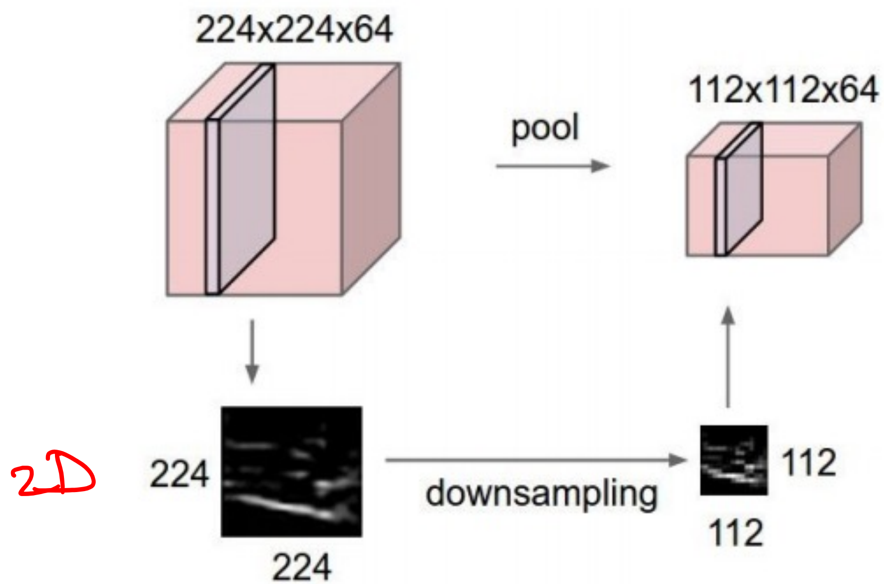
Convolution layer: Takeaways

- Convolution is a linear operation
  - Reduces parameter space of Feed-Forward Neural Network considerably
  - Capture locality of pixels in images
  - Smaller filters need less parameters
  - Multiple filters in each layer (computation can be done in parallel)
- Convolutions are followed by activation functions
  - Typically ReLU $(x) = \max(0, x)$

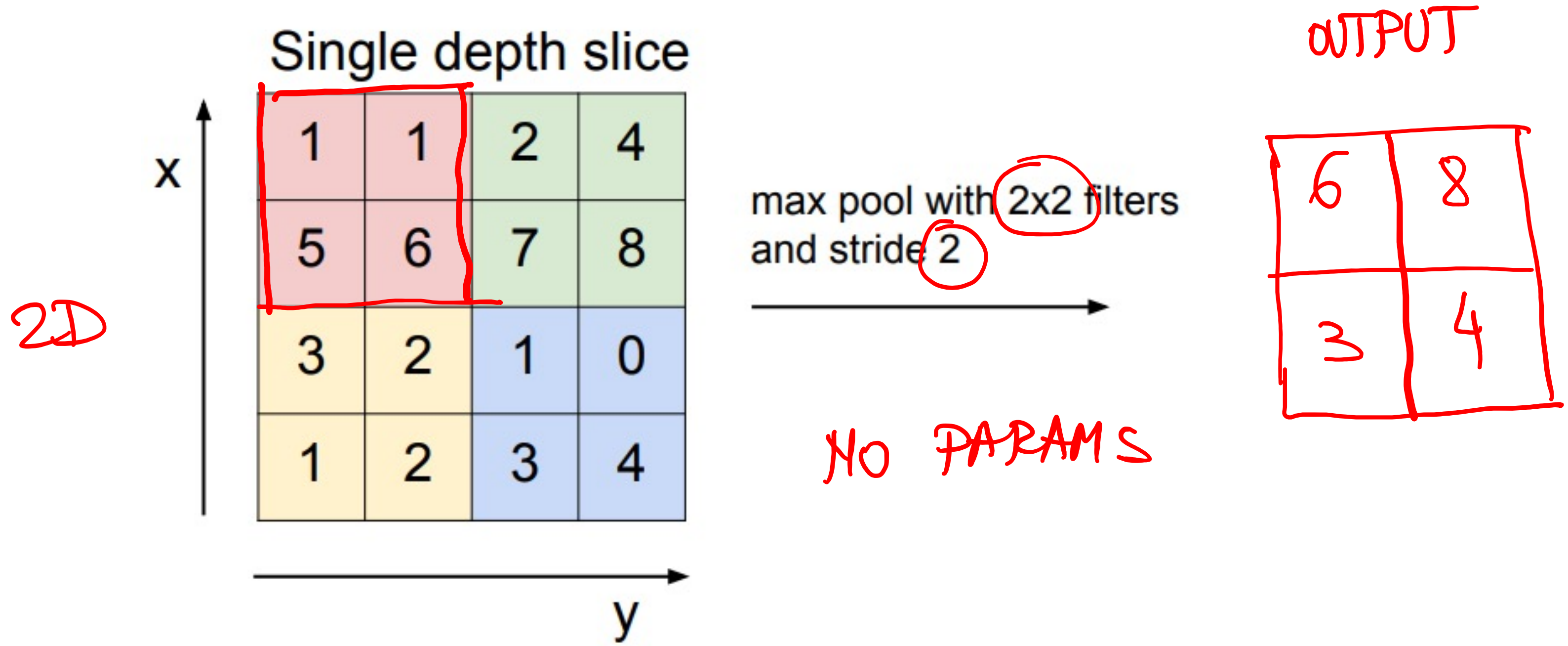
Pooling layer

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



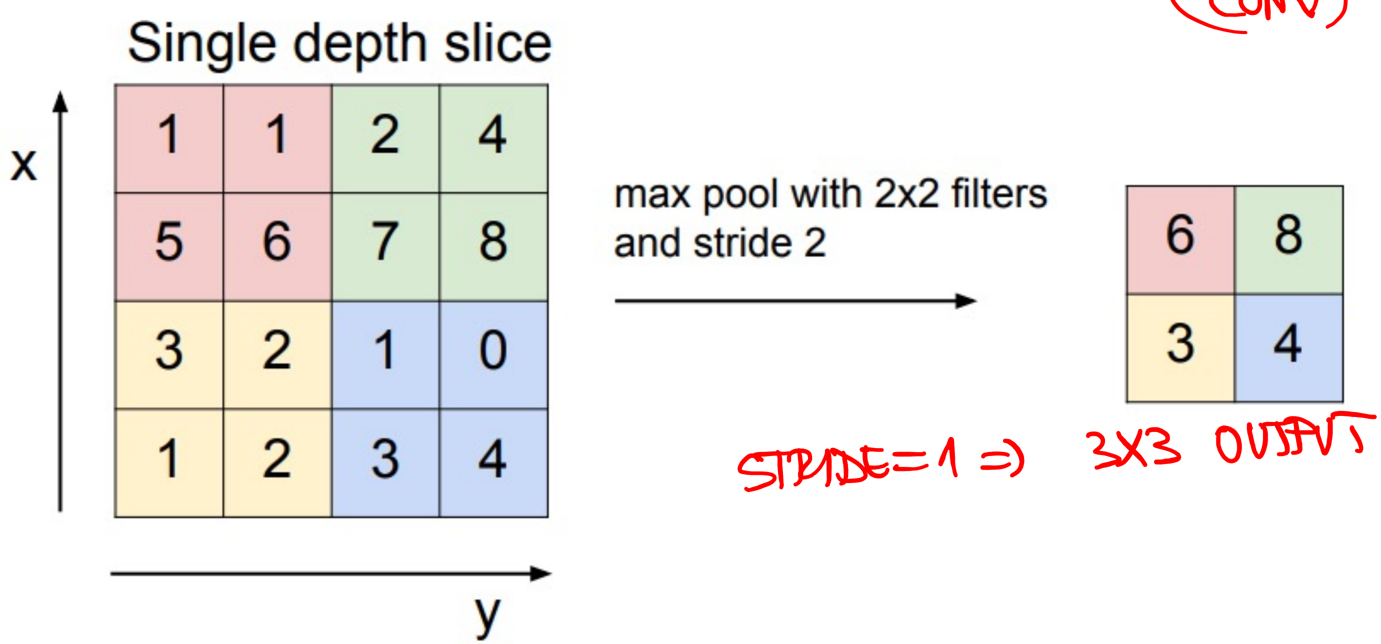
Max Pooling


OUTPUT
Single depth slice
x
y
2D
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4
NO PARAMS

Max Pooling

$(\text{CONV-ACT})^{*}\text{-POOL}$

$(\text{CONV})^{*}\text{-POOL}$



STRIDE=1 ⇒ 3X3 OUTPUT

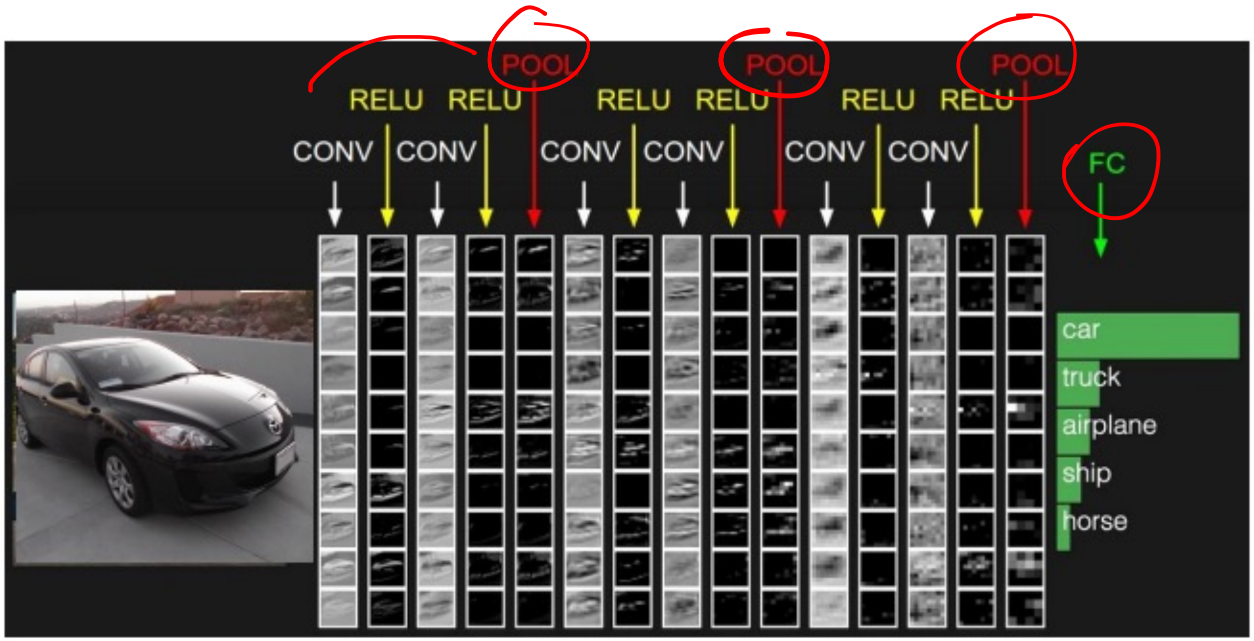
NO PADDING.

- Accepts a volume of size $\boxed{W_1 \times H_1 \times D_1}$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $\boxed{W_2 \times H_2 \times D_2}$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Convolutional Nets

Fully Connected Layer (FC layer)

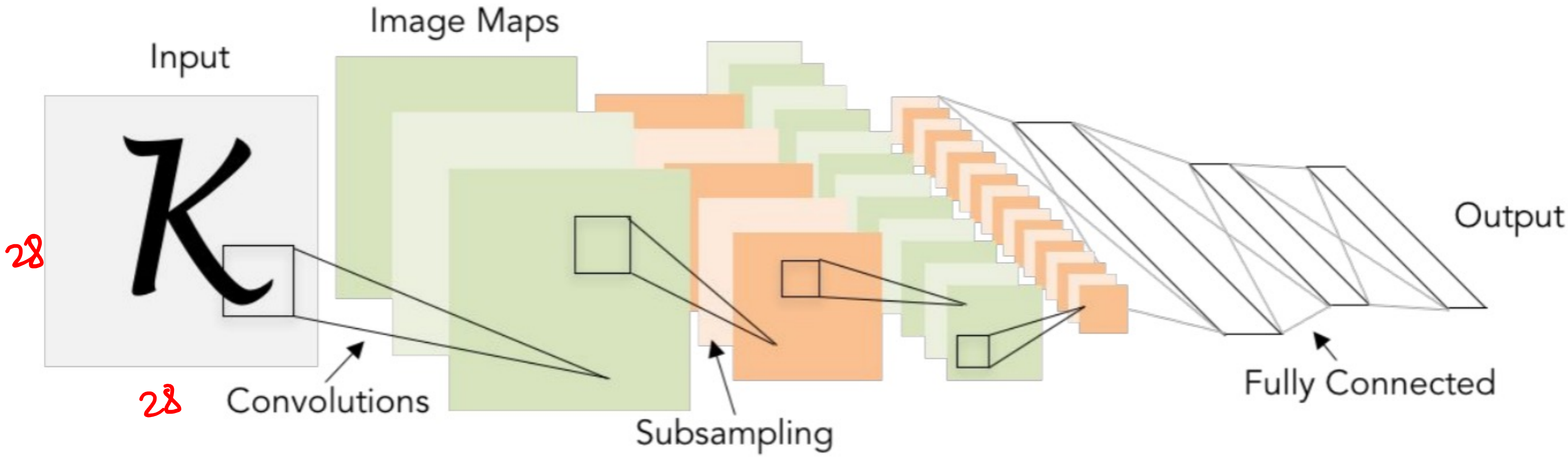
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



- FC layers are usually at the end, after several Convolutions and Pooling layers

LeNet 5

[LeCun et al., 1998]



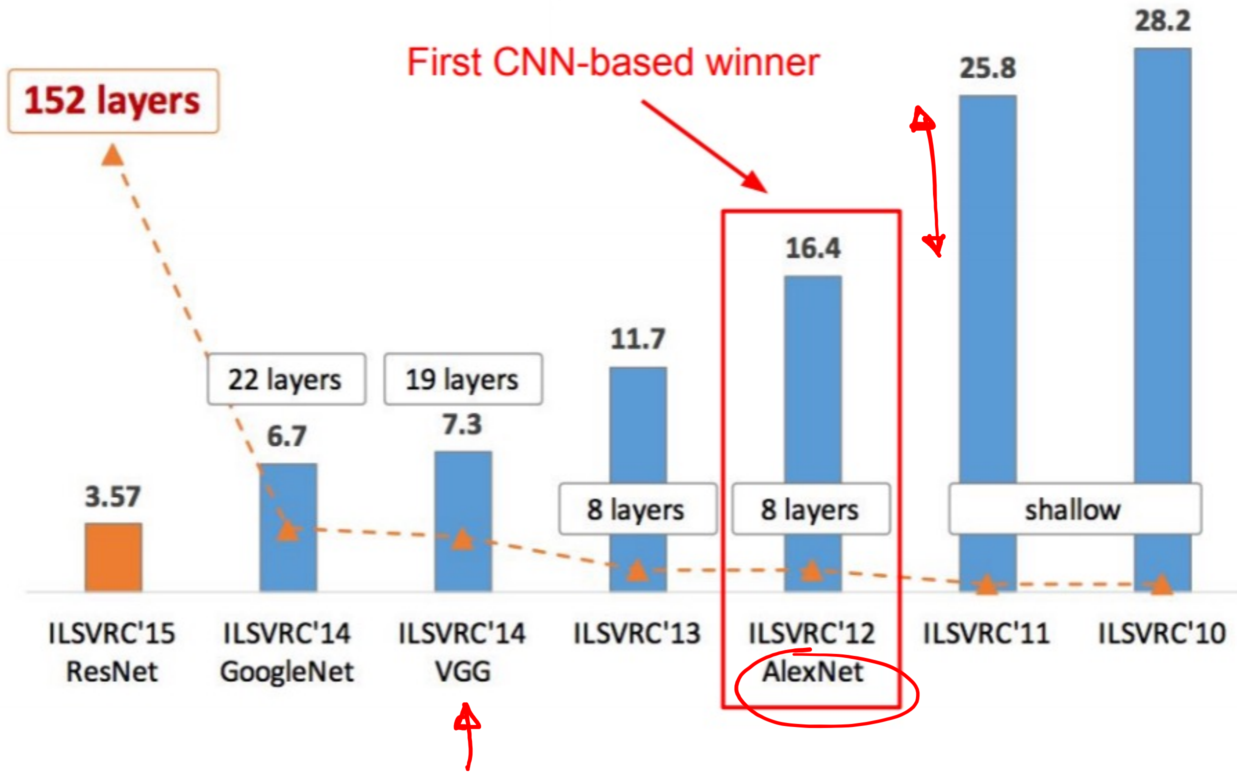
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

MNIST

AVG POOLING

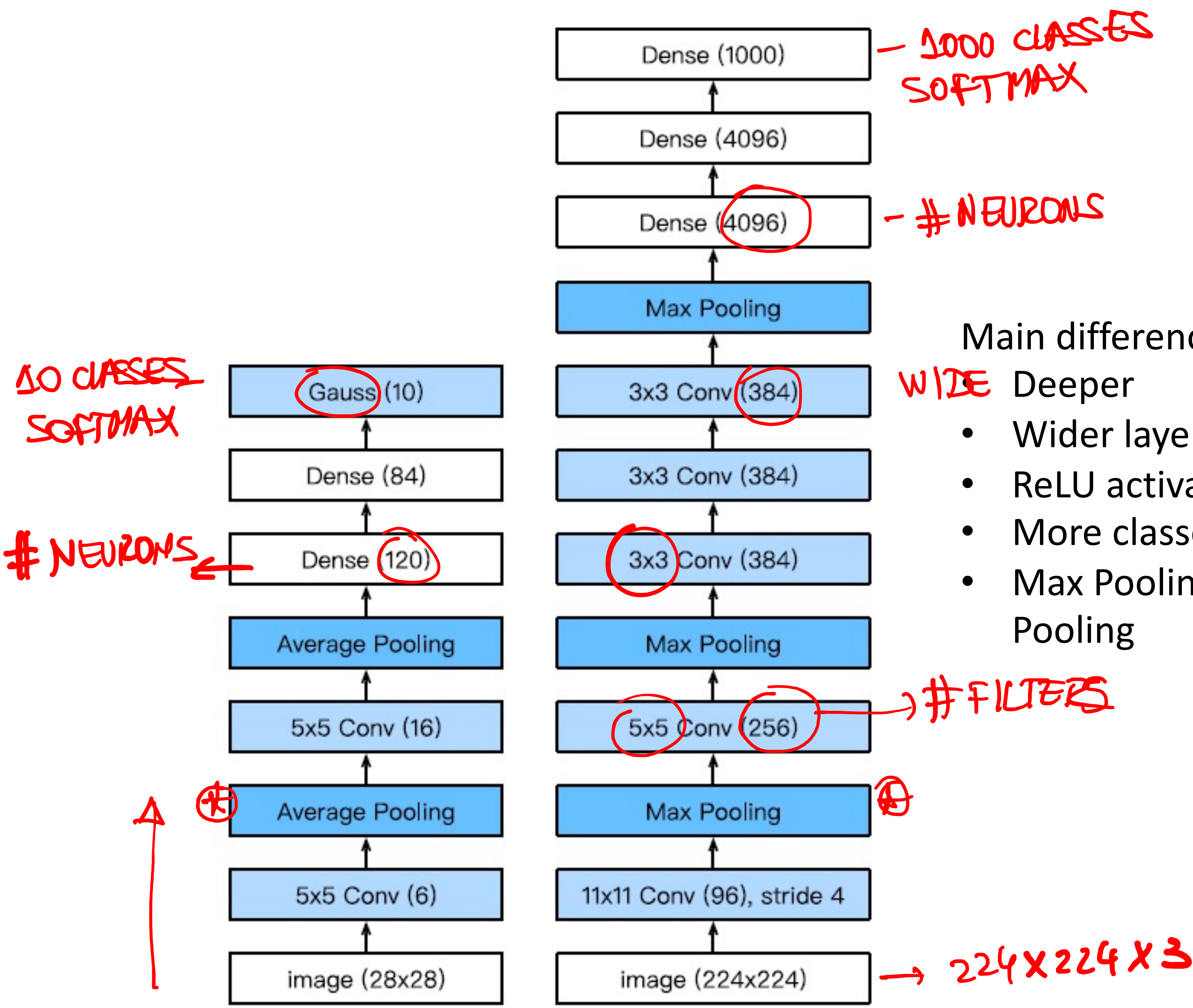
History

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ERROR TOP 5 CLASSES OUT OF 1000 CLASSES

LeNet (left) and AlexNet (right)



Main differences

WIDE Deeper

- Wider layers
- ReLU activation
- More classes in output layer
- Max Pooling instead of Avg Pooling

VGGNet

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

138 million parameters

Lab: Load Data

```
def load_data_matrix():
    print("Loading data")
    (X_train, y_train), (X_test, y_test) = mnist.load_data()

    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')

    # Normalize
    X_train /= 255
    X_test /= 255

    y_train = np_utils.to_categorical(y_train, 10)
    y_test = np_utils.to_categorical(y_test, 10)

    X_train = np.reshape(X_train, (60000, 28, 28, 1))
    X_test = np.reshape(X_test, (10000, 28, 28, 1))

    print("Data loaded")
    return [X_train, X_test, y_train, y_test]
```

Matrix form

28x28x1

Model Architecture

#FILTERS

SIZE OF FILTER

```
def init_model_cnn():
    print("Compiling Model")
    model = Sequential()

    model.add(layers.Conv2D(10, (3,3), input_shape=(28, 28, 1)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(layers.Conv2D(5, (3,3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(layers.Flatten())

    model.add(layers.Dense(64))
    model.add(Activation('relu'))

    model.add(layers.Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    return model
```

NO DEPTH

VECTOR

FC

SOFTMAX

model.add(layers.Dense(1))
(Activation (' sigmoid')

Number of Parameters

INPUT: 28X28X1

STRIDE=2

- CONV 1 → FILTER: 3X3, 10 FILTERS
  - OUTPUT SIZE: 26X26X10
  - PARAMS: $(3 \times 3 + 1) \cdot 10 = 100$
- POOL 1: 2X2, OUTPUT SIZE: 13X13X$\boxed{10}$
  - PARAMS: 0
- CONV 2: 5 FILTERS, 3X3X10
  - OUTPUT SIZE: 11X11X5
  - PARAMS: $(\underbrace{3 \times 3 \times 10}_{\text{FILTER}} + \underbrace{1}_{\text{BIAS}}) \times \underbrace{5}_{\text{\#FILTERS}} = 455$
- POOL 2: OUTPUT SIZE: 5X5X5 ↓ VECTOR
  - PARAMS: 0
- FLATTEN: OUTPUT SIZE 125
- FC 64: OUTPUT SIZE 64
  - PARAMS $125 \times 64 + 64 = 8064$
- FC 10: OUTPUT 10
  - PARAMS $64 \times 10 + 10 = 650$

Model Summary

```
model_cnn.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 26, 26, 10)        100
_________________________________________________________________
activation_49 (Activation)   (None, 26, 26, 10)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 10)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 11, 11, 5)         455
_________________________________________________________________
activation_50 (Activation)   (None, 11, 11, 5)         0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 5)           0
_________________________________________________________________
flatten_6 (Flatten)          (None, 125)               0
_________________________________________________________________
dense_43 (Dense)             (None, 64)                8064
_________________________________________________________________
activation_51 (Activation)   (None, 64)                0
_________________________________________________________________
dense_44 (Dense)             (None, 10)                650
_________________________________________________________________
activation_52 (Activation)   (None, 10)                0
=================================================================
Total params: 9,269
Trainable params: 9,269
Non-trainable params: 0
_________________________________________________________________
```

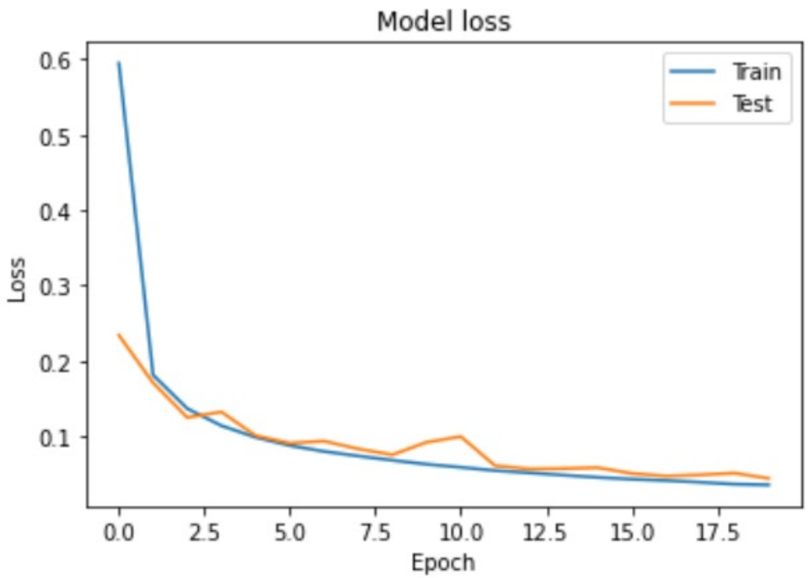
Results

```
model_cnn = init_model_cnn()

hist_cnn = run_network(model = model_cnn, epochs=20, cnn=True)

plot_losses(hist_cnn)
```

MODEL INIT

TRAIN

```
Epoch 18/20
235/235 - 5s - loss: 0.0394 - accuracy: 0.9874 - val_loss: 0.0492 - val_accuracy: 0.9843
Epoch 19/20
235/235 - 6s - loss: 0.0369 - accuracy: 0.9884 - val_loss: 0.0515 - val_accuracy: 0.9839
Epoch 20/20
235/235 - 6s - loss: 0.0358 - accuracy: 0.9887 - val_loss: 0.0445 - val_accuracy: 0.9866
Training duration:112.49477505683899
625/625 [==============================] - 1s 2ms/step - loss: 0.0445 - accuracy: 0.9866

Network's test loss and accuracy:[0.04450253024697304, 0.9865999817848206]
```



NO OVERFITTING

Summary CNNs

- Convolutional Nets are Feed-Forward Networks with at least one convolution layer and optionally max pooling layers
- Convolutions enable dimensionality reduction, are translation invariant and exploit locality
- Much fewer parameters relative to Feed-Forward Neural Networks
  - Deeper networks with multiple small filters at each layer is a trend
- Fully connected layer at the end (fewer parameters)
- Learn hierarchical feature representations
  - Data with natural grid topology (images, maps)
- Reached human-level performance in ImageNet in 2014

Overfitting

RIDGE : L2 NORM OF θ (MODEL PARAMS)
LASSO : L1

- The larger the network, the higher the capacity (more model parameters)
- But also more prone to overfitting!

Regularization

CROSS-ENTROPY LOSS

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \underbrace{\lambda R(W)}_{\text{Regularization: Prevent the model from doing too well on training data}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

λ = regularization strength (hyperparameter)

RIDGE

LASSO

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$ → Weight decay

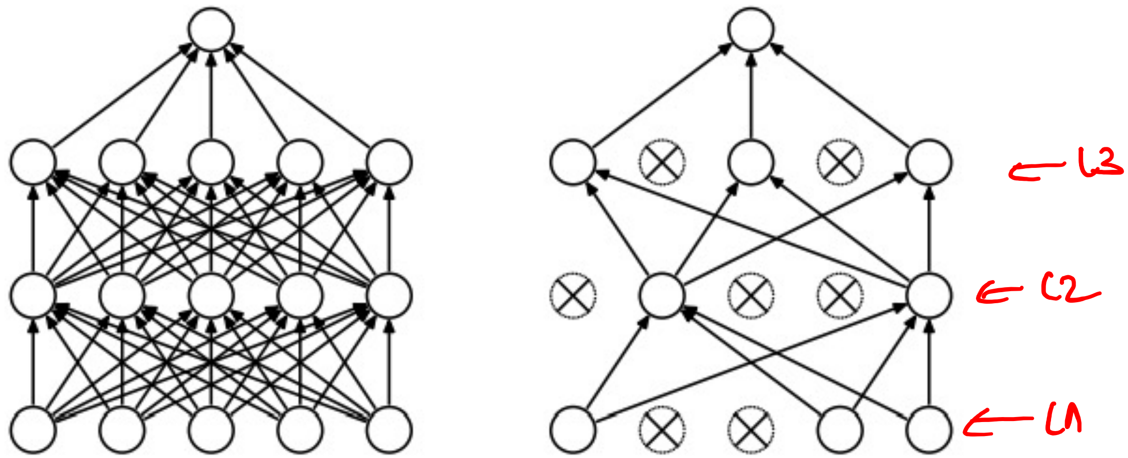
L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

- When computing gradients of loss function, regularization term needs to be taken into account

Dropout

ORIGINAL NET



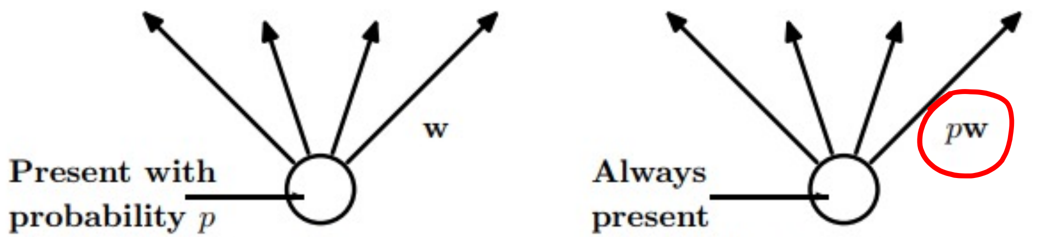
(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- Regularization technique that has proven very effective for deep learning
- Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15, 2014

Dropout

ALL NEURONS



(a) At training time (b) At test time

Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

fraction p of neurons at layer participate in training; ↑p drop out

- At training time, sample a sub-network per epoch (batch) and learn weights
  - Keep each neuron with probability p
- At testing time, all neurons are there, but multiply weight by a factor of p

Dropout Implementation

```python
def init_model():
    start_time = time.time()

    print("Compiling Model")
    model = Sequential()
```

$1-p$

```python
    # Hidden Layer 1
    model.add(Dense(500, input_dim=784))
    model.add(Dropout(0.3))
    model.add(Activation('relu'))

    # Hidden Layer 2
    model.add(Dense(300))
    model.add(Dropout(0.3))
    model.add(Activation('relu'))

    model.add(Dense(10))
    model.add(Activation('softmax'))

    rms = RMSprop()
    model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=['accuracy'])

    print("Model finished"+format(time.time() - start_time))
    return model
```

Dropout regularization

Results on MNIST

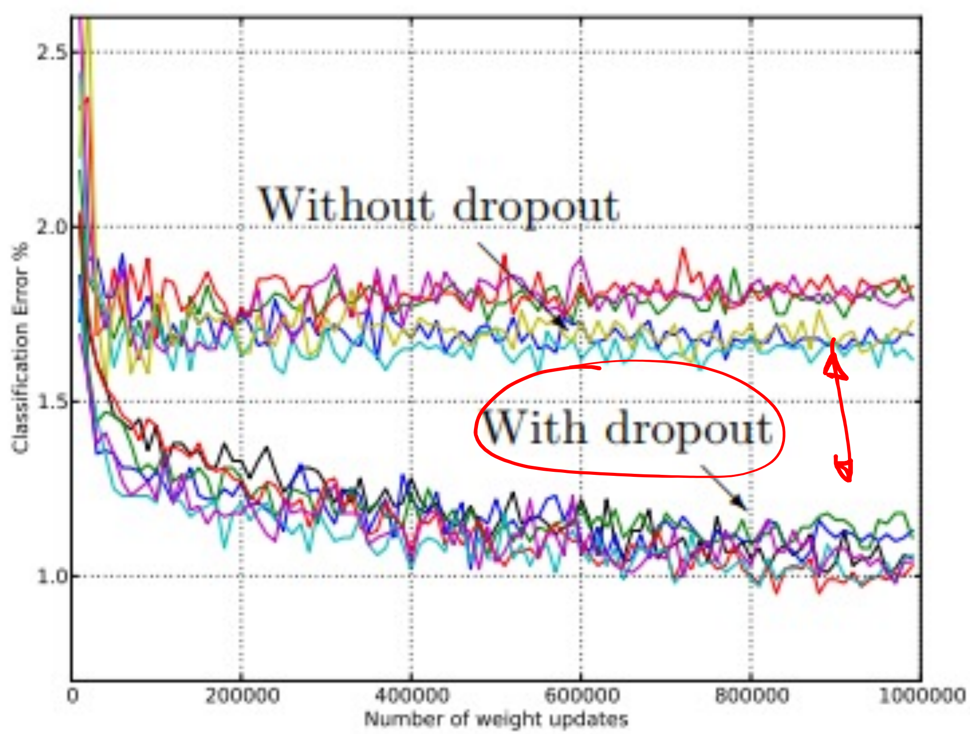


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Acknowledgements

- Slides made using resources from:
  - Andrew Ng
  - Eric Eaton
  - David Sontag
  - Andrew Moore
  - Yann LeCun
- Thanks!