W4995 Applied Machine Learning
Fall 2021

Lecture 9

Dr. Vijay Pappu

Announcements

- HW2 grades are released
- Project deliverable 2 grades are released
- Project deliverable 3 does not require a presentation and is due on 12/15
- Midterms are being graded currently
- No class next week (Thanksgiving break)

In today’s lecture, we will cover...

- Convolutional Neural Networks
- Recurrent Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

- Convolutional neural networks (CNNs) have been highly successful in many practical applications
  - Image classification
  - Object detection
  - ...
- The term “Convolutional Neural Network” indicates that the network employs a mathematical operation called **“convolution”**
- “Convolution” is a specialized kind of linear operator
- CNNs are simply neural networks that use convolution instead of general matrix multiplication in at least one of their layers.

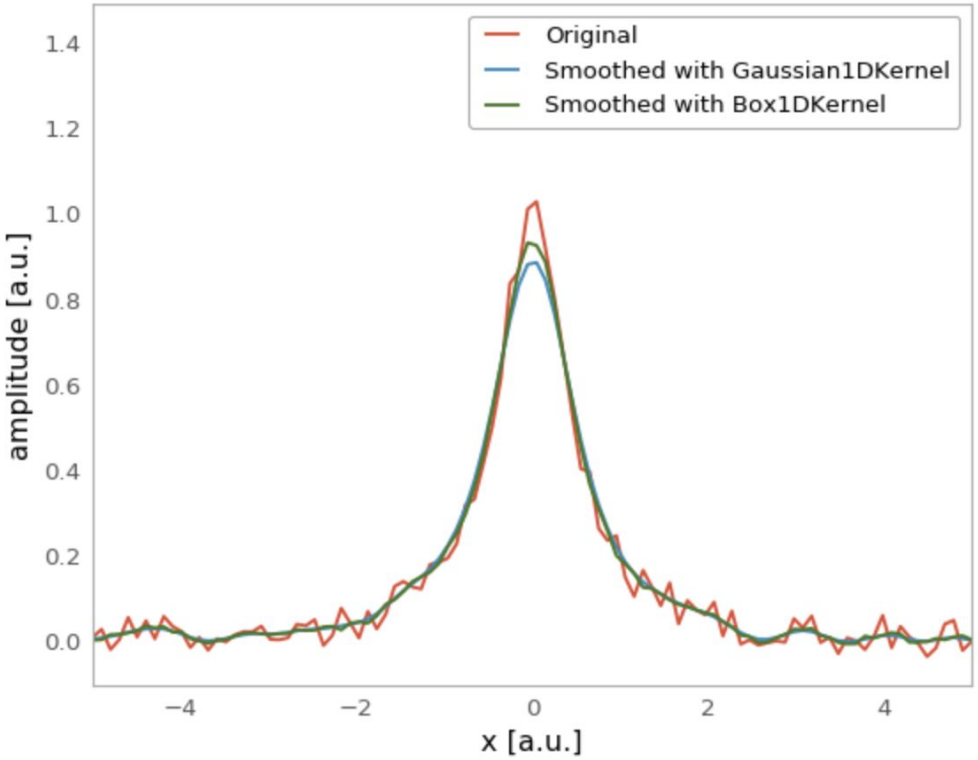
Convolution - 1D function

$$s(t) = \int x(a)w(t-a)da.$$

$$s(t) = (x * w)(t).$$

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a).$$

Convolution - 1D function



https://keflavich-astropy.readthedocs.io/en/latest/convolution/kernels.html

Convolution - 2D function

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

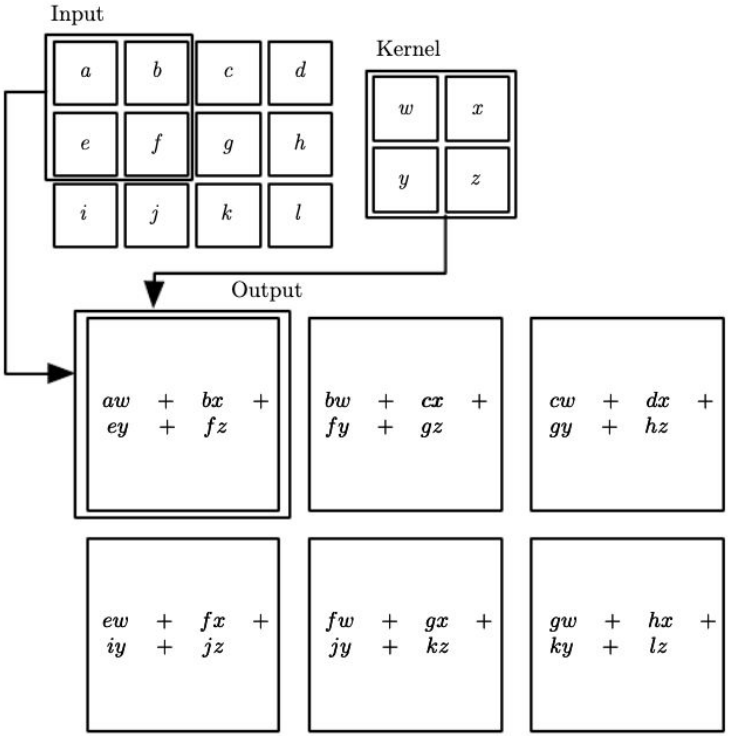
$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$

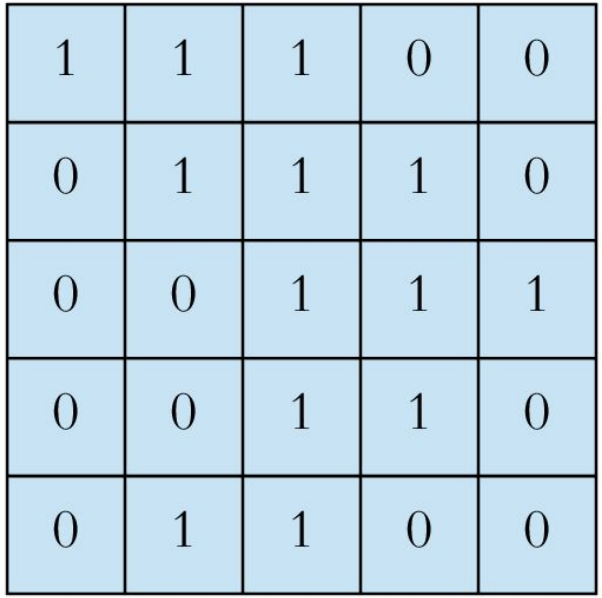
cross-correlation

(2D convolution without kernel flipping)

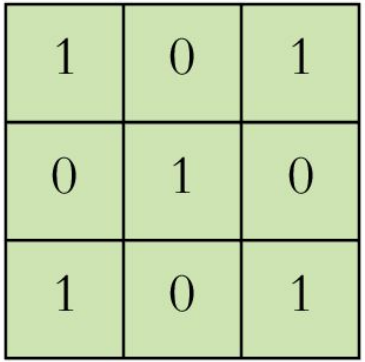
Convolution - 2D function



Convolution - 2D function



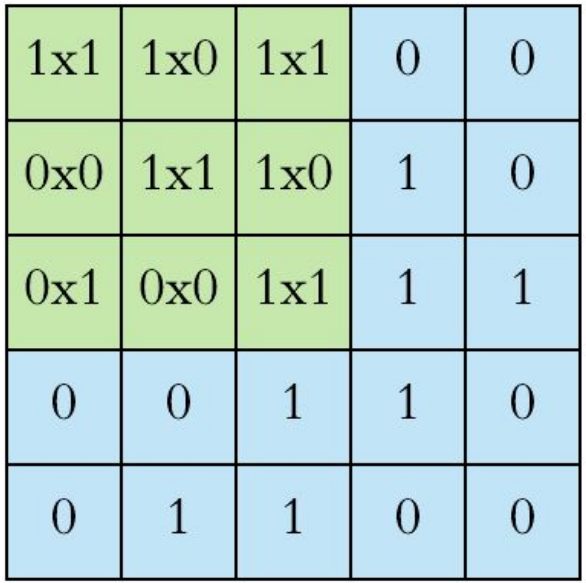
Input



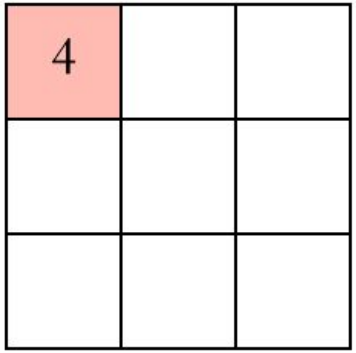
Filter / Kernel

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Convolution - 2D function



Input



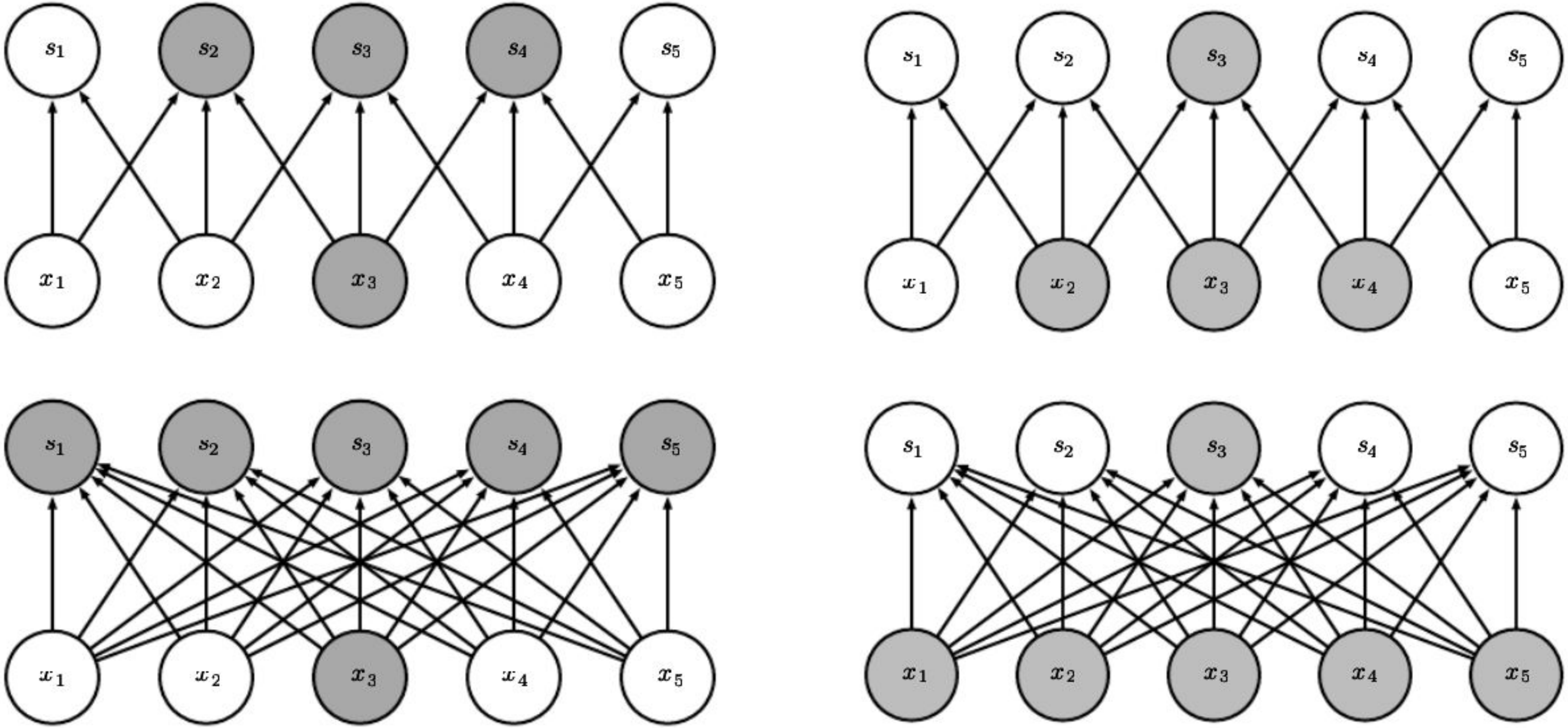
Feature map

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Convolutions in Neural Networks

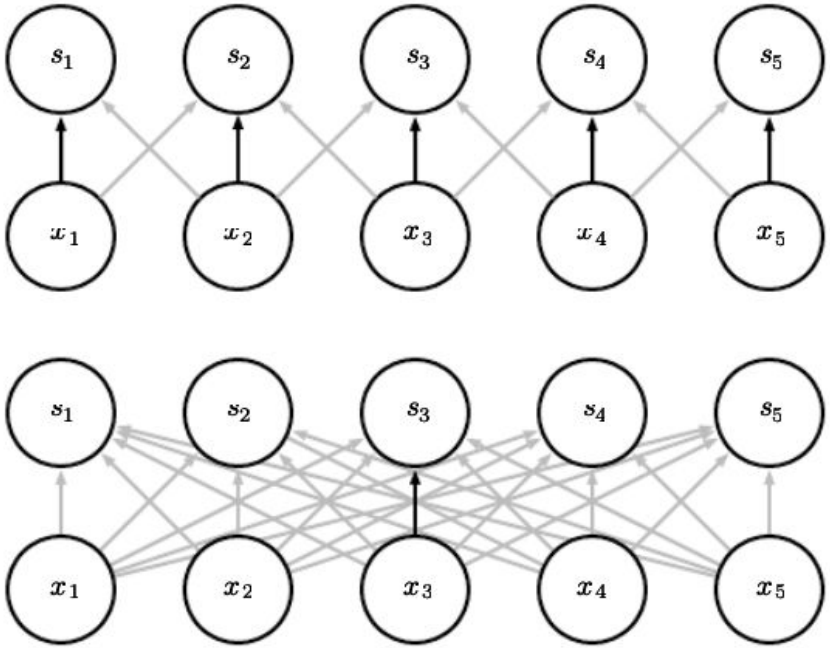
- Sparse interactions
- Parameter sharing
- Equivariant representations

Convolutions in Neural Networks - Sparse Interactions



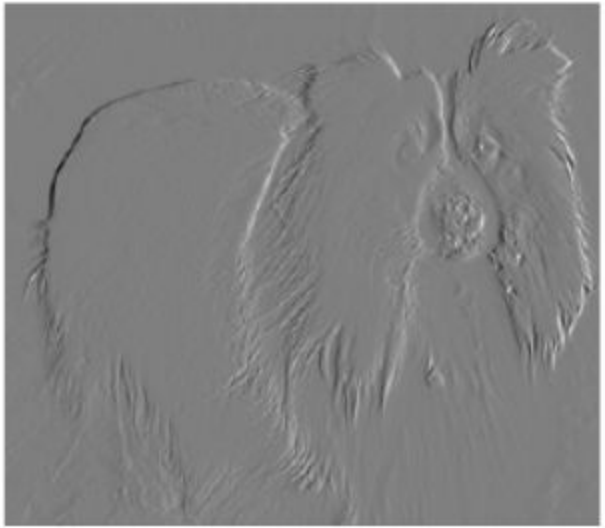
https://www.deeplearningbook.org/contents/convnets.html

Convolutions in Neural Networks - Parameter Sharing



https://www.deeplearningbook.org/contents/convnets.html

Convolutions in Neural Networks

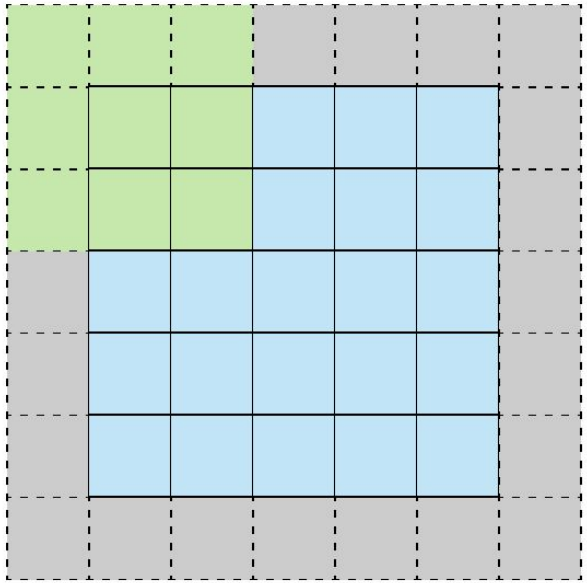


https://www.deeplearningbook.org/contents/convnets.html

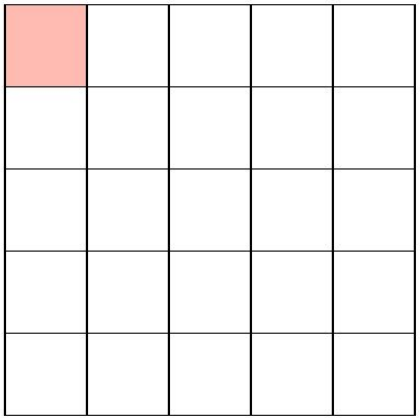
Convolutions in Neural Networks - Stride & Padding

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Convolutions in Neural Networks - Stride & Padding



Stride 1 with Padding



Feature Map

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

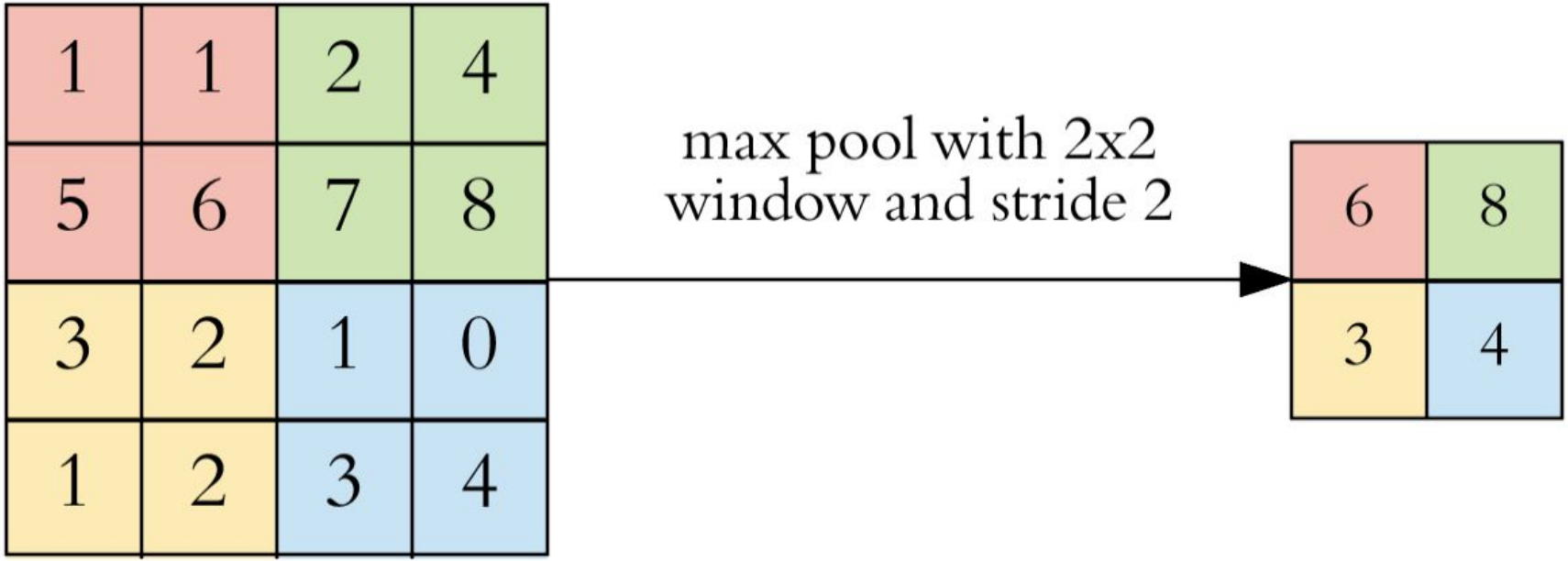
Convolutions in Neural Networks - Activation function

1	1	1	0	0
0	1	1	1	0
0	0	1x1	1x0	1x1
0	0	1x0	1x1	0x0
0	1	1x1	0x0	0x1

4	3	4
2	4	3
2	3	4

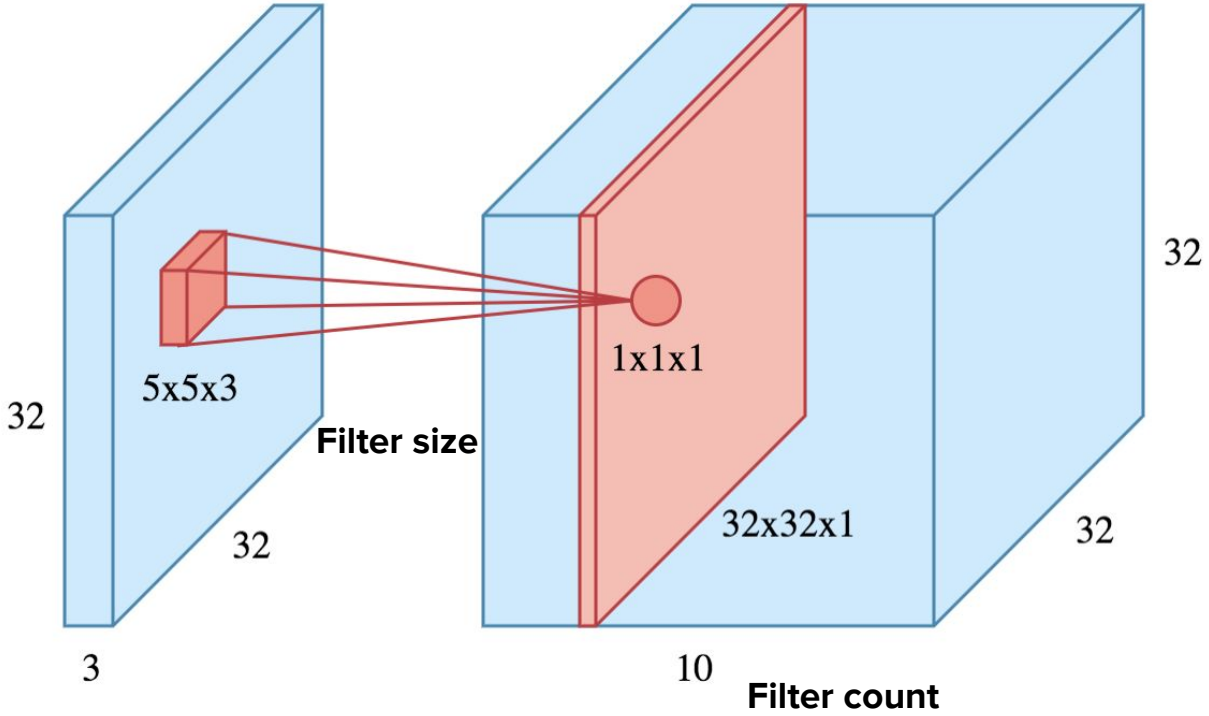
$h(x)$

Convolutional Neural Networks - Pooling



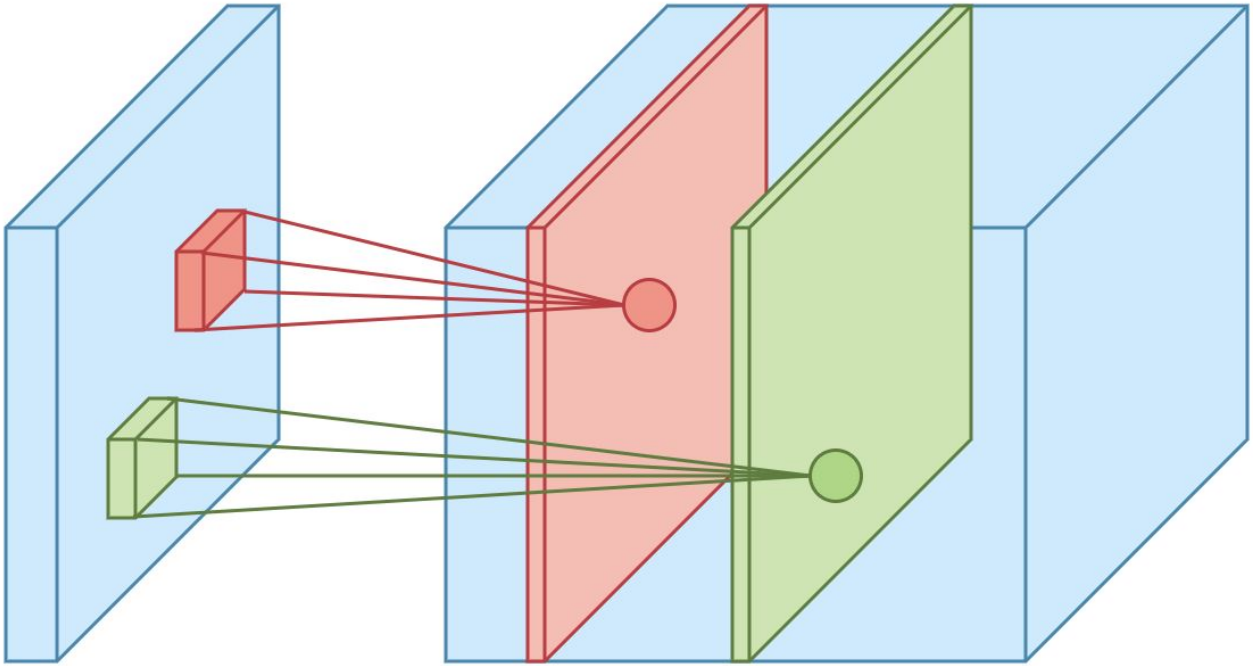
https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

3D Convolutions in Neural Networks



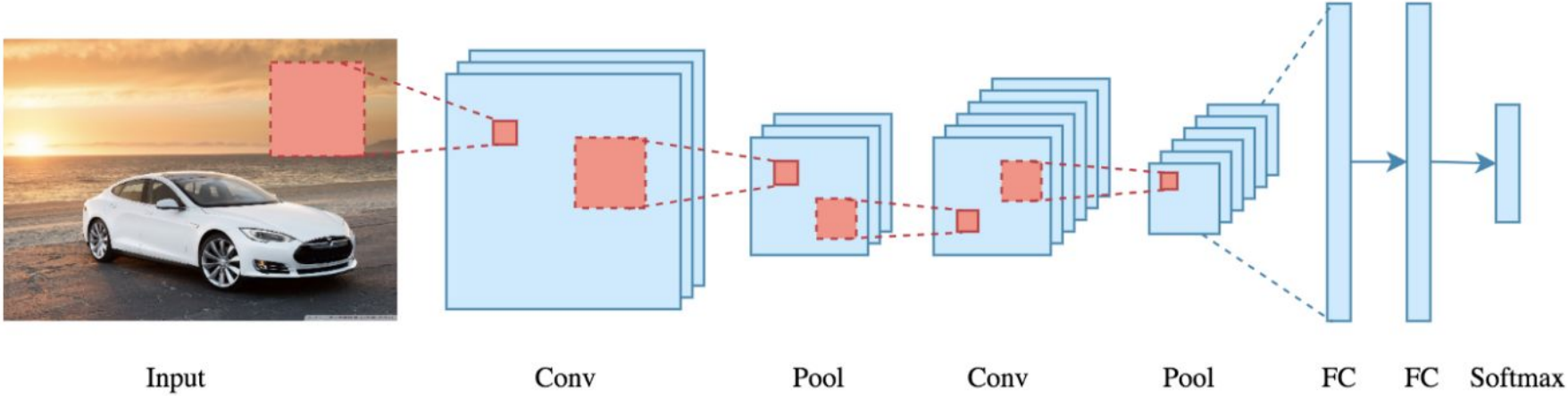
https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

3D Convolutions in Neural Networks



https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Typical CNN architecture



https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

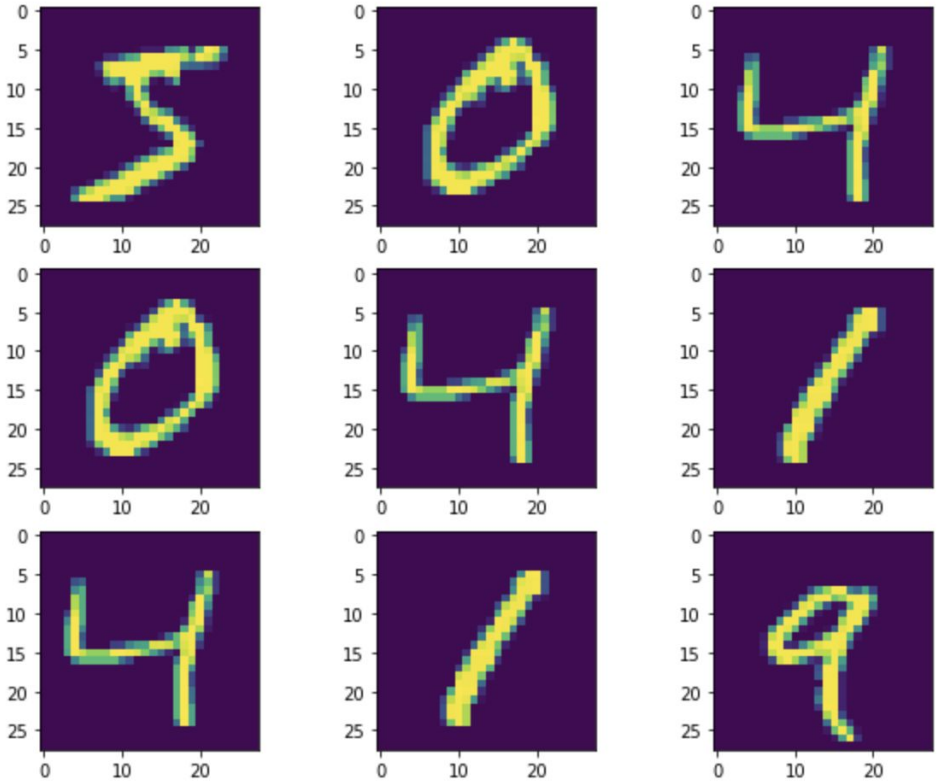
Training Convolutional Neural Networks

- CNNs are generally trained using backpropagation algorithm
- Math gets messy

Convolutional Neural Networks - Hyperparameters

- Network architecture
- Convolutional layer
  - Filter size
  - Filter count
  - Padding (mostly yes)
  - Stride (usually 1)
- Optimization algorithm parameters
- Activation function (ReLU, tanh, sigmoid etc.)
- Weight initialization
- Batch size
- # of epochs

Convolutional Neural Networks - Example



https://keras.io/api/datasets/mnist/

Convolutional Neural Networks - Example

```
batch_size = 128
num_classes = 10
# input image dimensions
img_rows, img_cols = 28, 28
# the data, shuffled and split between train and test sets
(x_dev, y_dev), (x_test, y_test) = mnist.load_data()
X_dev_images = x_dev.reshape(x_dev.shape[0], img_rows, img_cols, 1)
X_test_images = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)
y_dev = np_utils.to_categorical(y_dev, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)
```

Convolutional Neural Networks - Example

```
num_classes = 10
cnn = Sequential()
cnn.add(Conv2D(32, kernel_size=(3, 3),
                activation='relu',
                input_shape=input_shape))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Conv2D(32, (3, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Flatten())
cnn.add(Dense(64, activation='relu'))
cnn.add(Dense(num_classes, activation='softmax'))
```

Convolutional Neural Networks - Example

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 32)         0
 2D)

 flatten_1 (Flatten)         (None, 800)               0

 dense (Dense)               (None, 64)                51264

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 61,482
Trainable params: 61,482
Non-trainable params: 0
_________________________________________________________________
```

Convolutional Neural Networks - Example

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 32)         0
 2D)

 flatten_1 (Flatten)         (None, 800)               0

 dense (Dense)               (None, 64)                51264

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 61,482
Trainable params: 61,482
Non-trainable params: 0
_________________________________________________________________
```

(3*3*1+1)*32 = 320

Convolutional Neural Networks - Example

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 32)         0
 2D)

 flatten_1 (Flatten)         (None, 800)               0

 dense (Dense)               (None, 64)                51264

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 61,482
Trainable params: 61,482
Non-trainable params: 0
_________________________________________________________________
```

(3*3*1+1)*32 = 320

(3*3*32+1)*32 = 9248

Convolutional Neural Networks - Example

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)            (None, 11, 11, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 32)         0
 2D)

 flatten_1 (Flatten)          (None, 800)               0

 dense (Dense)                (None, 64)                51264

 dense_1 (Dense)              (None, 10)                650

=================================================================
Total params: 61,482
Trainable params: 61,482
Non-trainable params: 0
_________________________________________________________________
```

(3*3*1+1)*32 = 320

(3*3*32+1)*32 = 9248

(800+1)*64 = 51264

Convolutional Neural Networks - Example

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)            (None, 11, 11, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 32)         0
 2D)

 flatten_1 (Flatten)          (None, 800)               0

 dense (Dense)                (None, 64)                51264

 dense_1 (Dense)              (None, 10)                650

=================================================================
Total params: 61,482
Trainable params: 61,482
Non-trainable params: 0
_________________________________________________________________
```

(3*3*1+1)*32 = 320

(3*3*32+1)*32 = 9248

(800+1)*64 = 51264

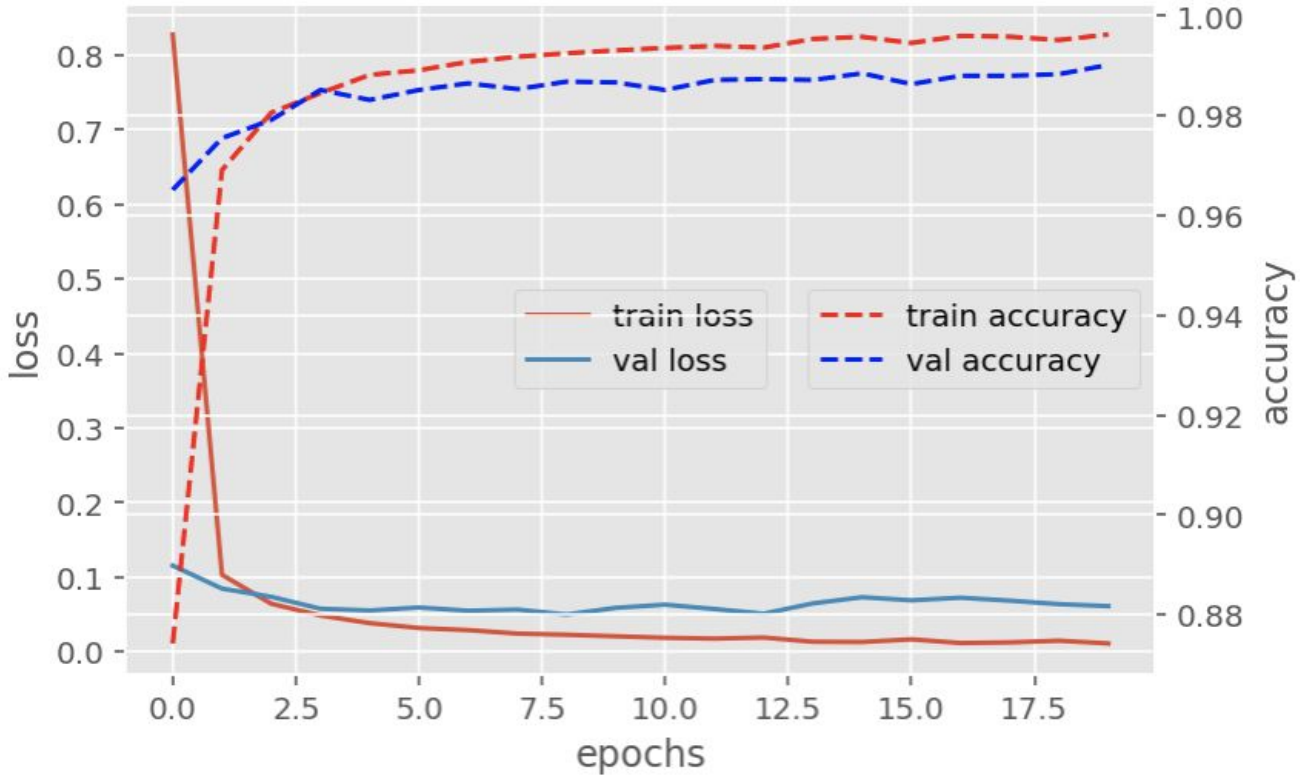
(64+1)*10 = 650

Convolutional Neural Networks - Example

```
cnn.compile("adam", "categorical_crossentropy", metrics=['accuracy'])
history_cnn = cnn.fit(X_dev_images, y_dev,
                      batch_size=128, epochs=20, verbose=1, validation_split=.1)
cnn.evaluate(X_test_images, y_test)
```

```
Epoch 1/20
422/422 [==============================] - 17s 38ms/step - loss: 0.8281 - accuracy: 0.8743 - val_loss: 0.1155 - val_accuracy: 0.
9652
Epoch 2/20
422/422 [==============================] - 16s 37ms/step - loss: 0.1032 - accuracy: 0.9691 - val_loss: 0.0847 - val_accuracy: 0.
9755
Epoch 3/20
422/422 [==============================] - 17s 40ms/step - loss: 0.0644 - accuracy: 0.9806 - val_loss: 0.0737 - val_accuracy: 0.
9792
Epoch 4/20
422/422 [==============================] - 17s 40ms/step - loss: 0.0486 - accuracy: 0.9845 - val_loss: 0.0576 - val_accuracy: 0.
9852
Epoch 5/20
422/422 [==============================] - 18s 42ms/step - loss: 0.0383 - accuracy: 0.9882 - val_loss: 0.0553 - val_accuracy: 0.
9832
Epoch 6/20
422/422 [==============================] - 18s 43ms/step - loss: 0.0318 - accuracy: 0.9891 - val_loss: 0.0593 - val_accuracy: 0.
9852
Epoch 7/20
422/422 [==============================] - 19s 46ms/step - loss: 0.0289 - accuracy: 0.9908 - val_loss: 0.0550 - val_accuracy: 0.
9865
Epoch 8/20
422/422 [==============================] - 21s 49ms/step - loss: 0.0243 - accuracy: 0.9918 - val_loss: 0.0565 - val_accuracy: 0.
9853
```

Convolutional Neural Networks - Example



CNNs v.s. Fully Connected NNs - Example

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 512)               401920

 dense_5 (Dense)             (None, 10)                5130

=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

Fully connected NN

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 32)         0
 2D)

 flatten_1 (Flatten)         (None, 800)               0

 dense (Dense)               (None, 64)                51264

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 61,482
Trainable params: 61,482
Non-trainable params: 0
_________________________________________________________________
```

Convolutional NN

Convolutional Neural Networks - Another Example

https://www.kaggle.com/c/dogs-vs-cats/

Convolutional Neural Networks - Another Example

```
base_dir = 'dogs-vs-cats-small'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validate')
test_dir = os.path.join(base_dir, 'test')

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

Convolutional Neural Networks - Another Example

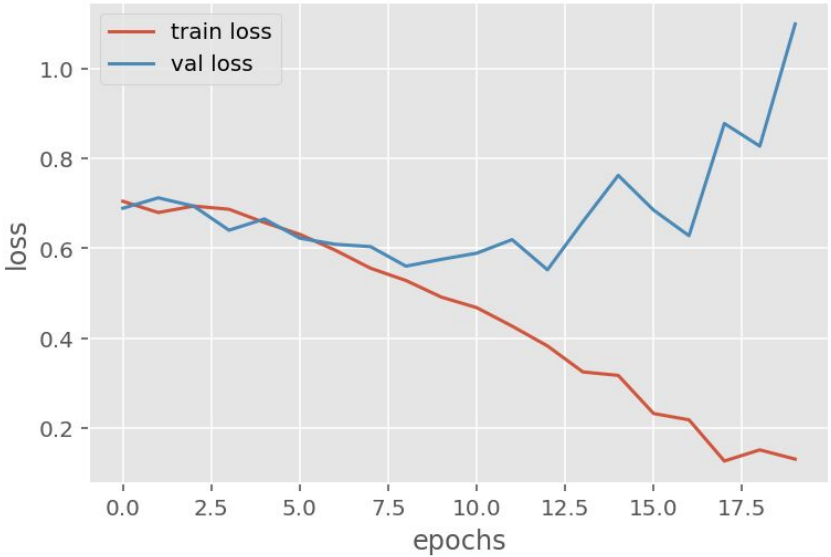
```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1',
                 input_shape=(150, 150, 3)))
model.add(MaxPooling2D((2, 2), name='maxpool_1'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'))
model.add(MaxPooling2D((2, 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(256, activation='relu', name='dense_2'))
model.add(Dense(1, activation='sigmoid', name='output'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

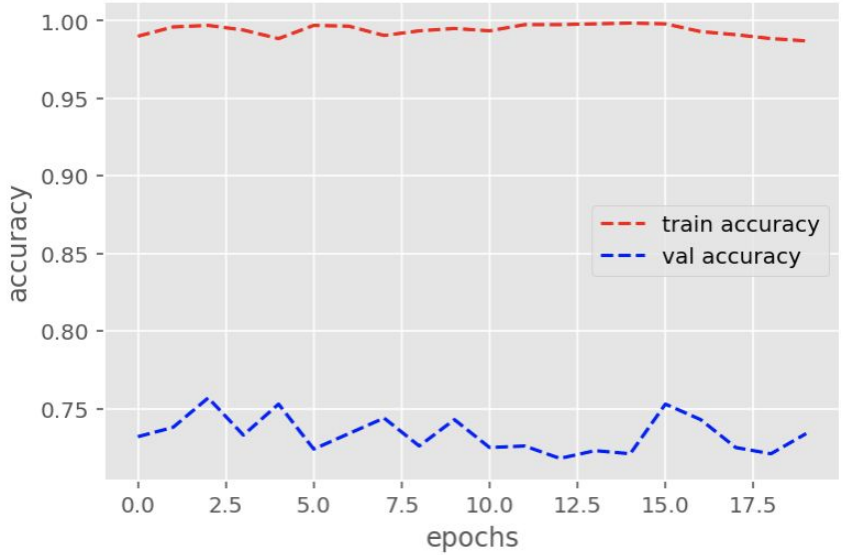
Convolutional Neural Networks - Another Example

```
history = model.fit_generator(train_generator, steps_per_epoch=100, epochs=20,
                              validation_data=validation_generator, validation_steps=50, verbose=1)
```

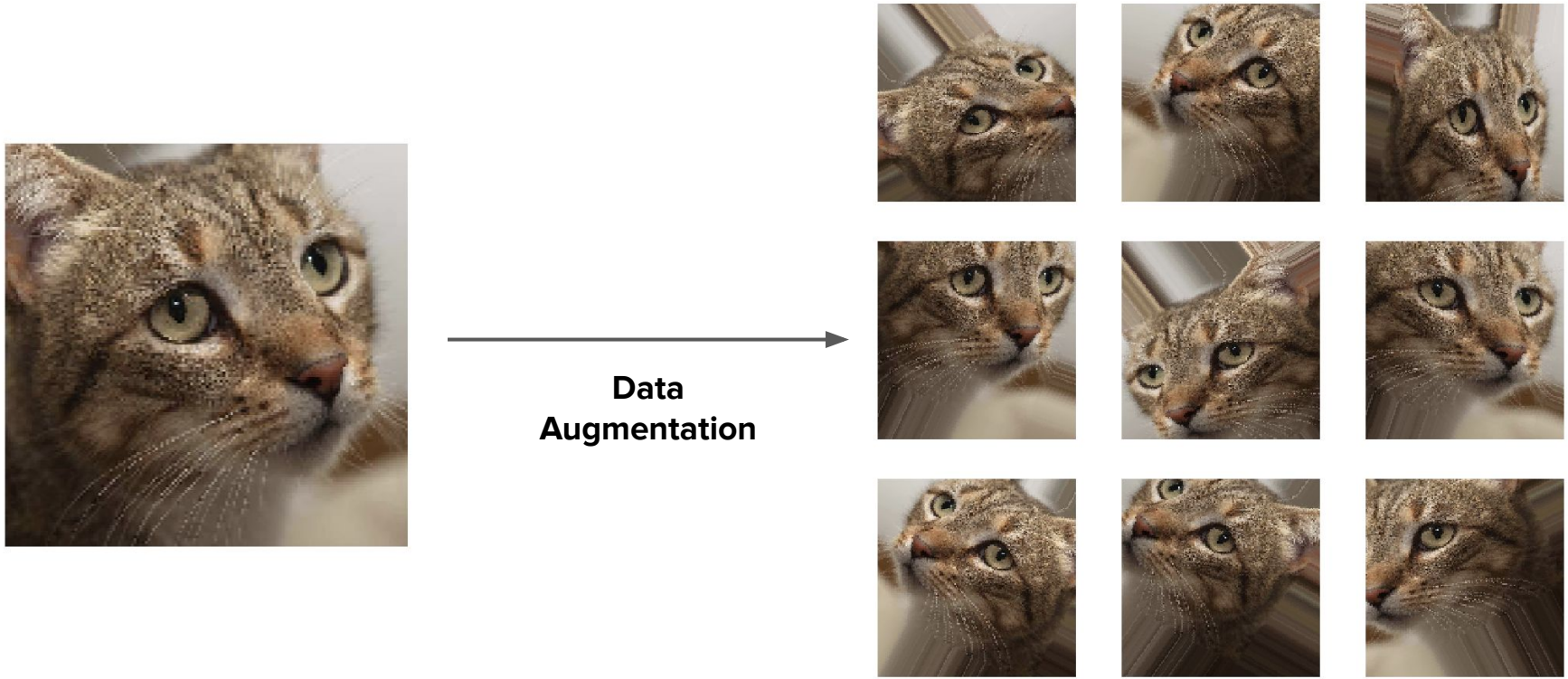
```
Epoch 1/20
100/100 [==============================] - 27s 266ms/step - loss: 0.7045 - accuracy: 0.5100 - val_loss: 0.6891 - val_accu
racy: 0.5800
Epoch 2/20
100/100 [==============================] - 28s 275ms/step - loss: 0.6795 - accuracy: 0.5510 - val_loss: 0.7122 - val_accu
racy: 0.5060
Epoch 3/20
100/100 [==============================] - 29s 292ms/step - loss: 0.6939 - accuracy: 0.5335 - val_loss: 0.6939 - val_accu
racy: 0.4790
Epoch 4/20
100/100 [==============================] - 30s 297ms/step - loss: 0.6868 - accuracy: 0.5590 - val_loss: 0.6401 - val_accu
racy: 0.6160
Epoch 5/20
100/100 [==============================] - 28s 283ms/step - loss: 0.6573 - accuracy: 0.5875 - val_loss: 0.6652 - val_accu
racy: 0.6110
Epoch 6/20
100/100 [==============================] - 27s 274ms/step - loss: 0.6310 - accuracy: 0.6435 - val_loss: 0.6224 - val_accu
racy: 0.6560
Epoch 7/20
100/100 [==============================] - 29s 291ms/step - loss: 0.5957 - accuracy: 0.6855 - val_loss: 0.6088 - val_accu
racy: 0.6610
```

Convolutional Neural Networks - Another Example





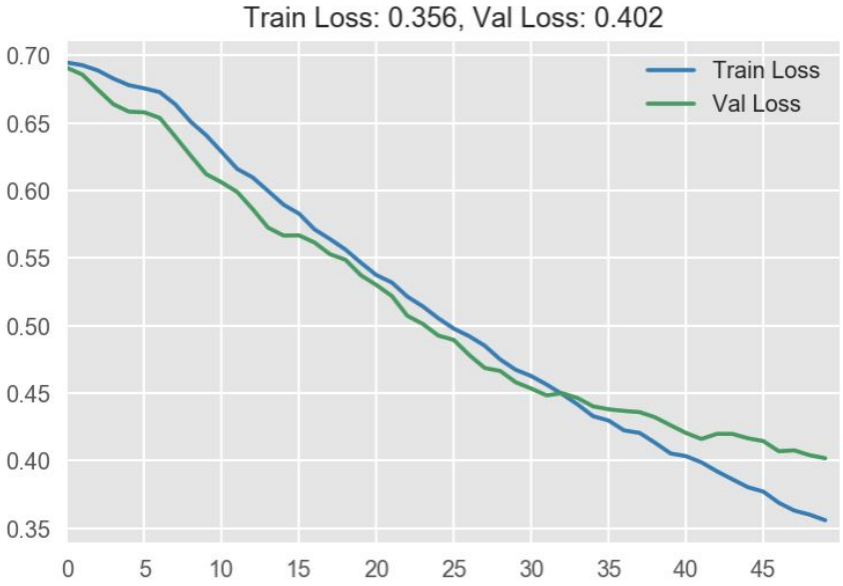
Convolutional Neural Networks - Another Example

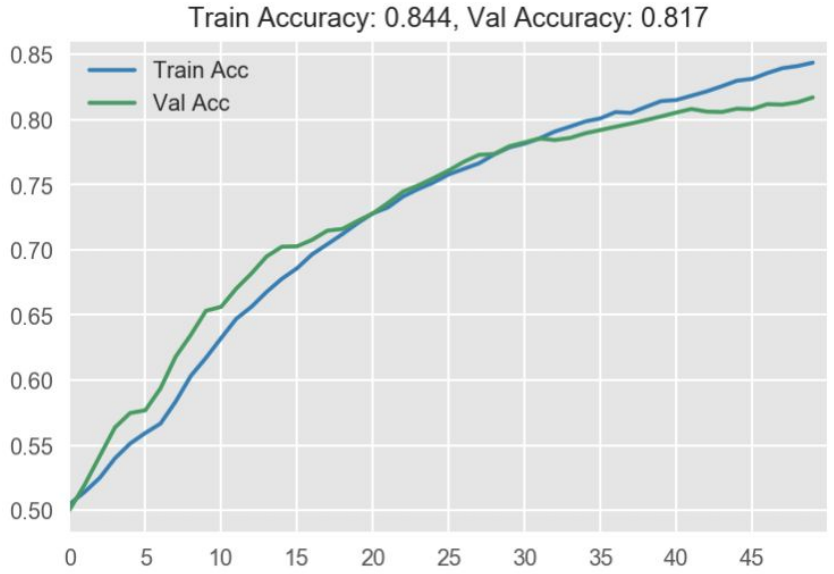


Convolutional Neural Networks - Another Example

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

# early_stop = EarlyStopping(monitor='val_loss', patience=6, verbose=1)
history_aug = model_aug.fit_generator(train_generator, steps_per_epoch=100, epochs=60,
                                      validation_data=validation_generator, validation_steps=50, verbose=1)
```

Convolutional Neural Networks - Another Example



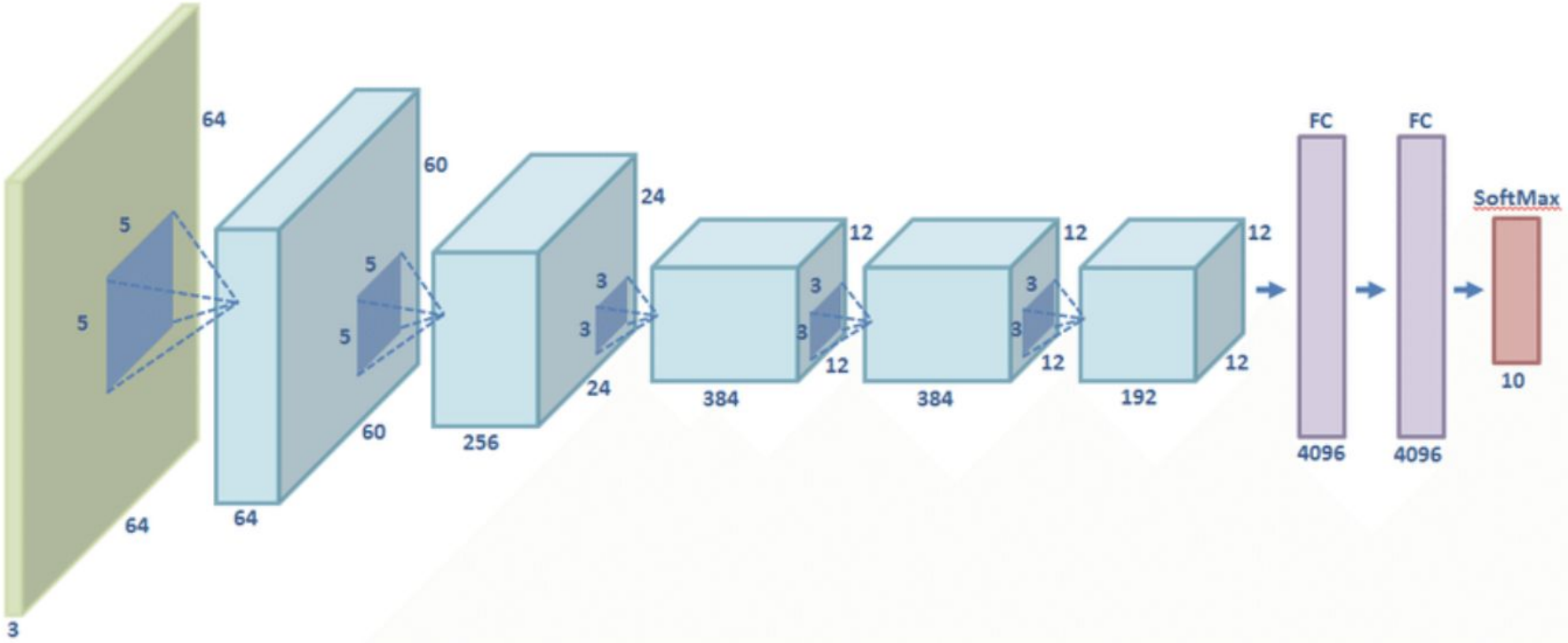


Questions?

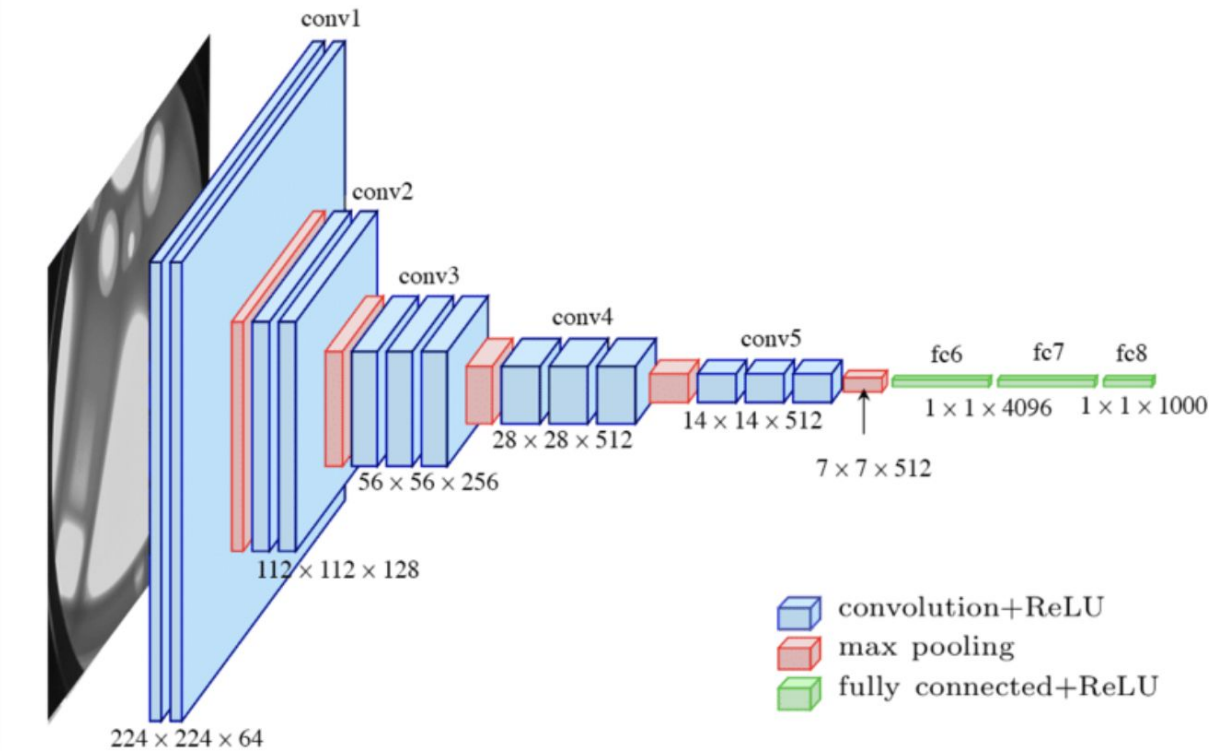
Let’s take a 10 min break!

CNN architectures

Convolutional Neural Networks - AlexNet



Convolutional Neural Networks - VGG16



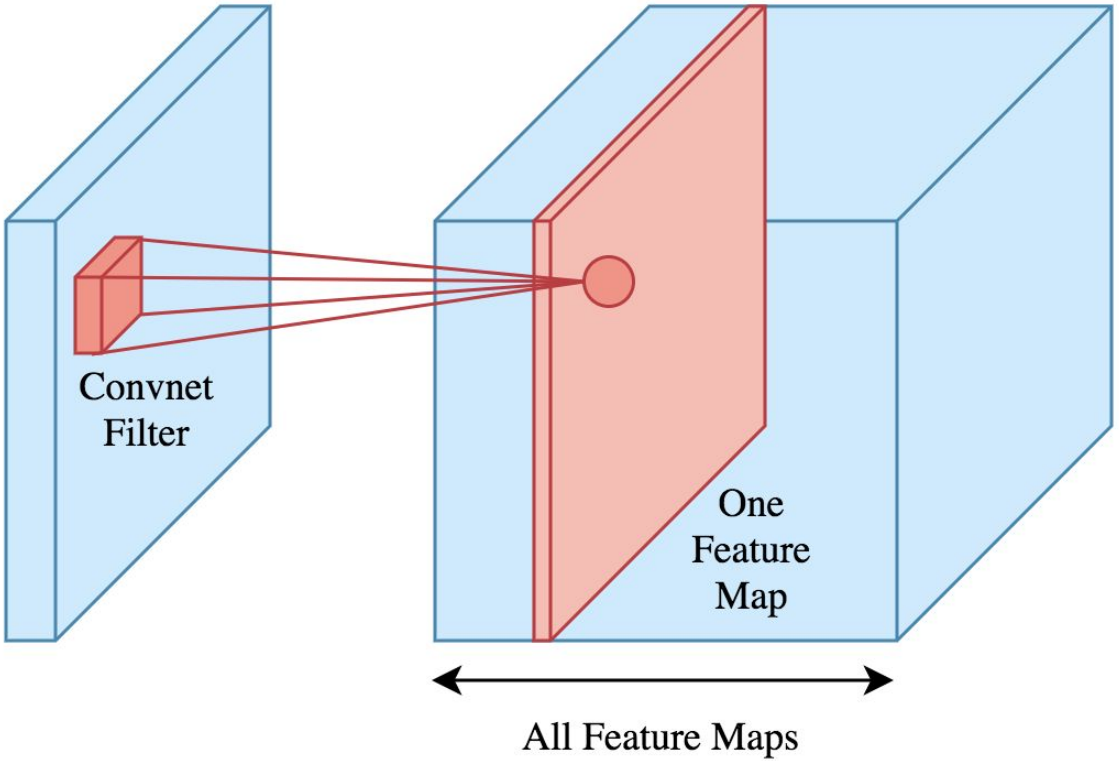
Convolutional Neural Networks - VGG16

- VGG16 uses only 3x3 convolutions
  - Two stacked 3x3 convolutions -> a single 5x5 convolution.
  - Three stacked 3x3 convolutions -> a single 7x7 convolution.
- Stacking two convolutions uses two relu operations, and thus capture more non-linear interactions
- The number of filters increase as we go deeper into the network.
- Trained on 4 GPUs for 3 weeks

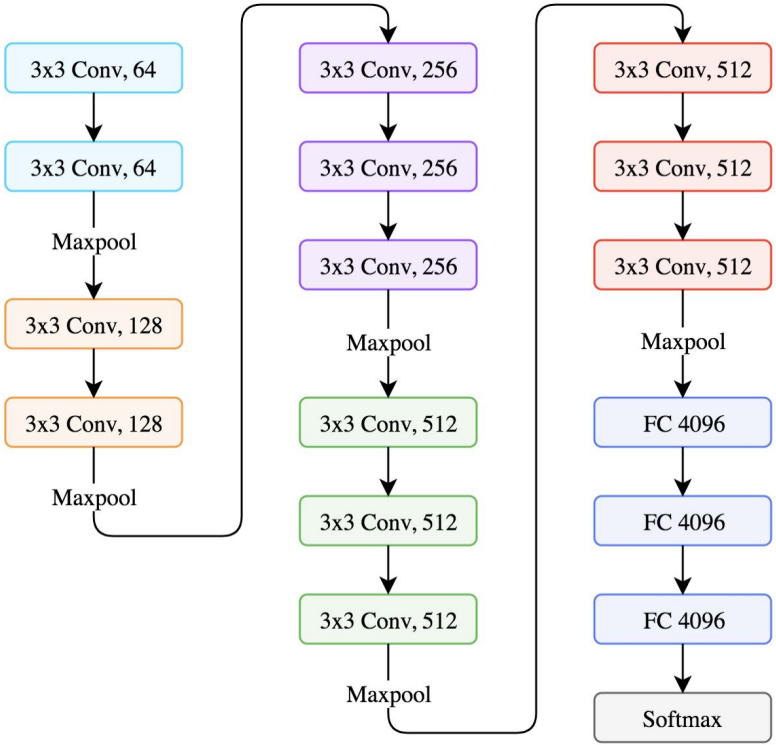
Convolutional Neural Networks - Visualizing VGG16

- Feature maps
- Convnet filters
- Class outputs

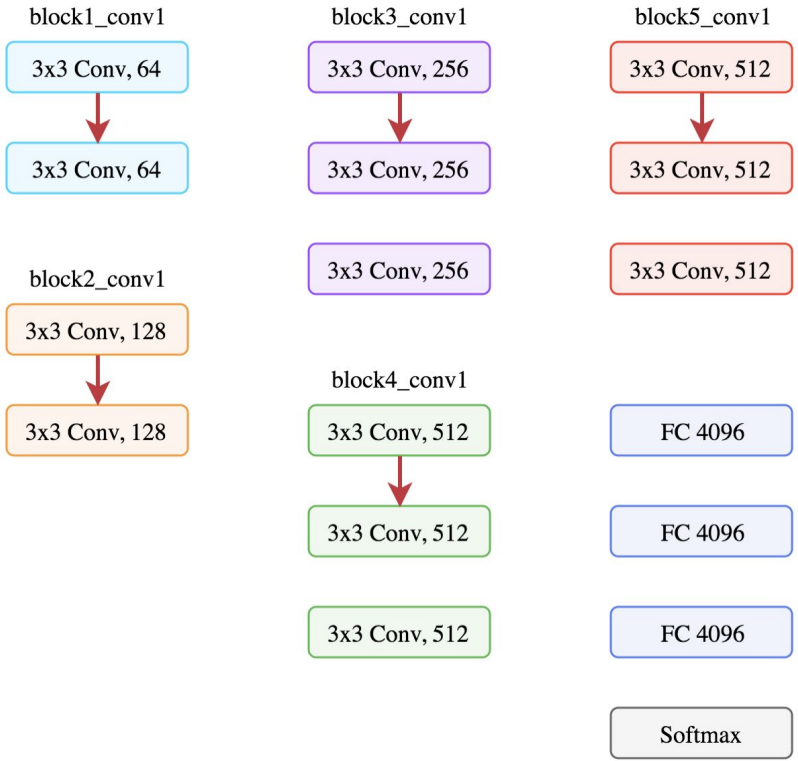
Convolutional Neural Networks - Visualizing VGG16



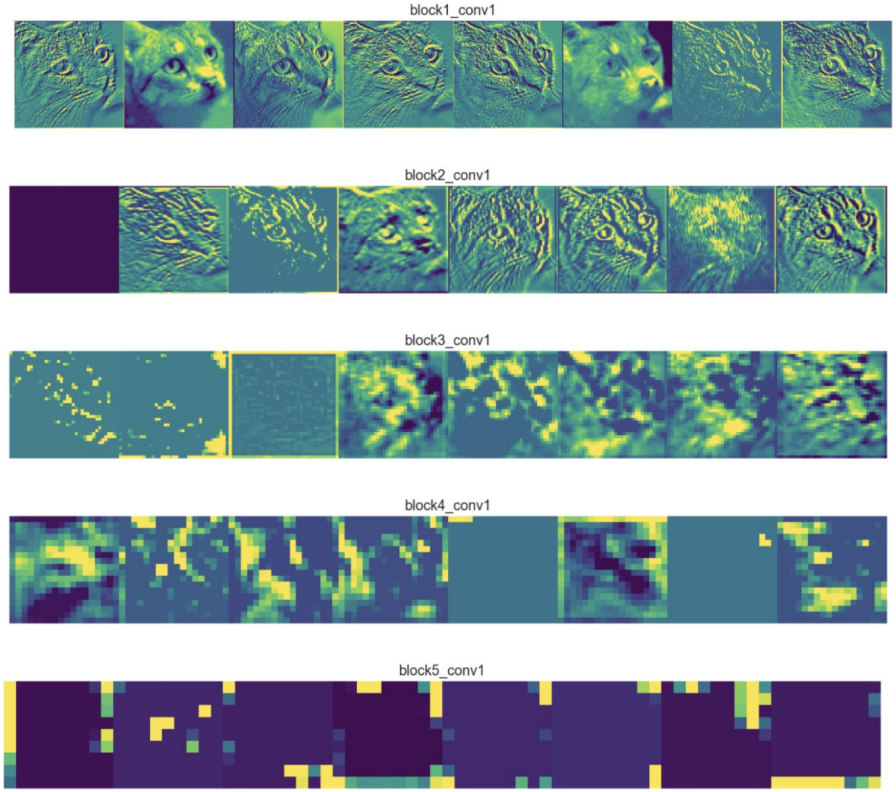
Convolutional Neural Networks - Visualizing VGG16



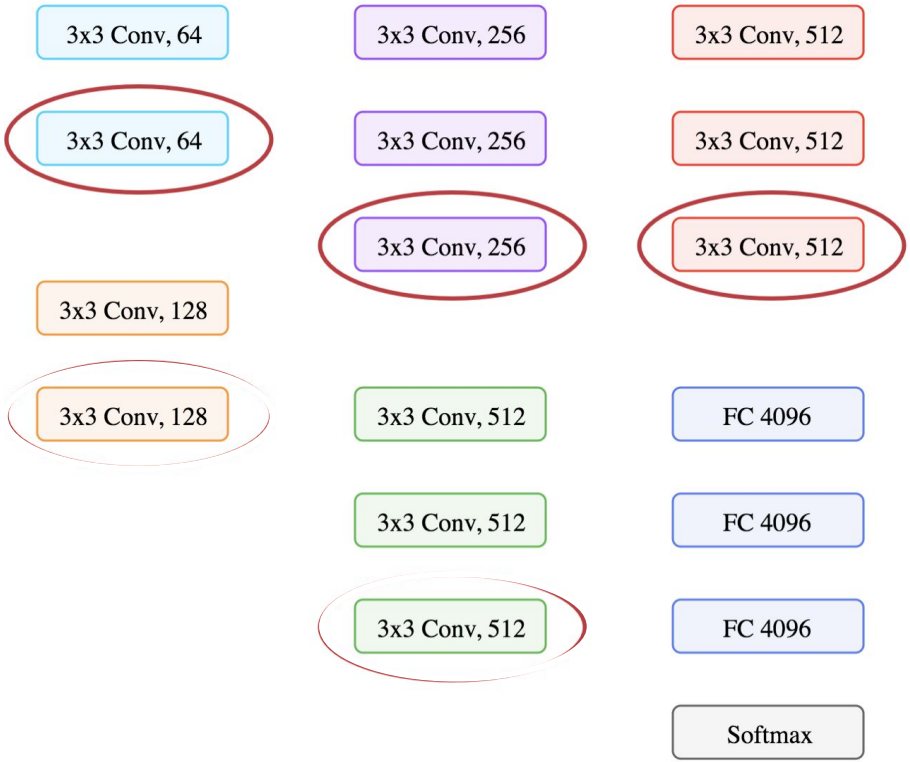
Convolutional Neural Networks - Visualizing Feature Maps



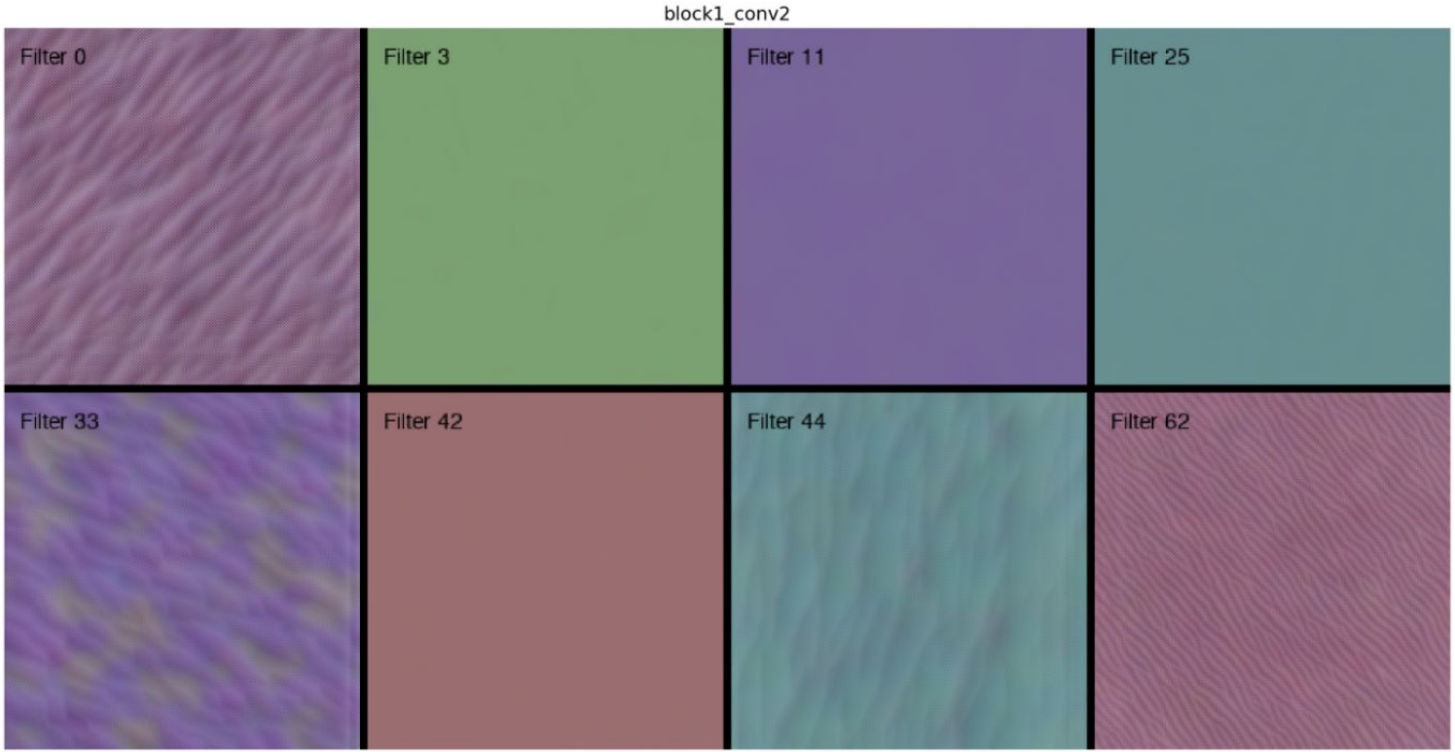
Convolutional Neural Networks - VGG16 Feature Maps



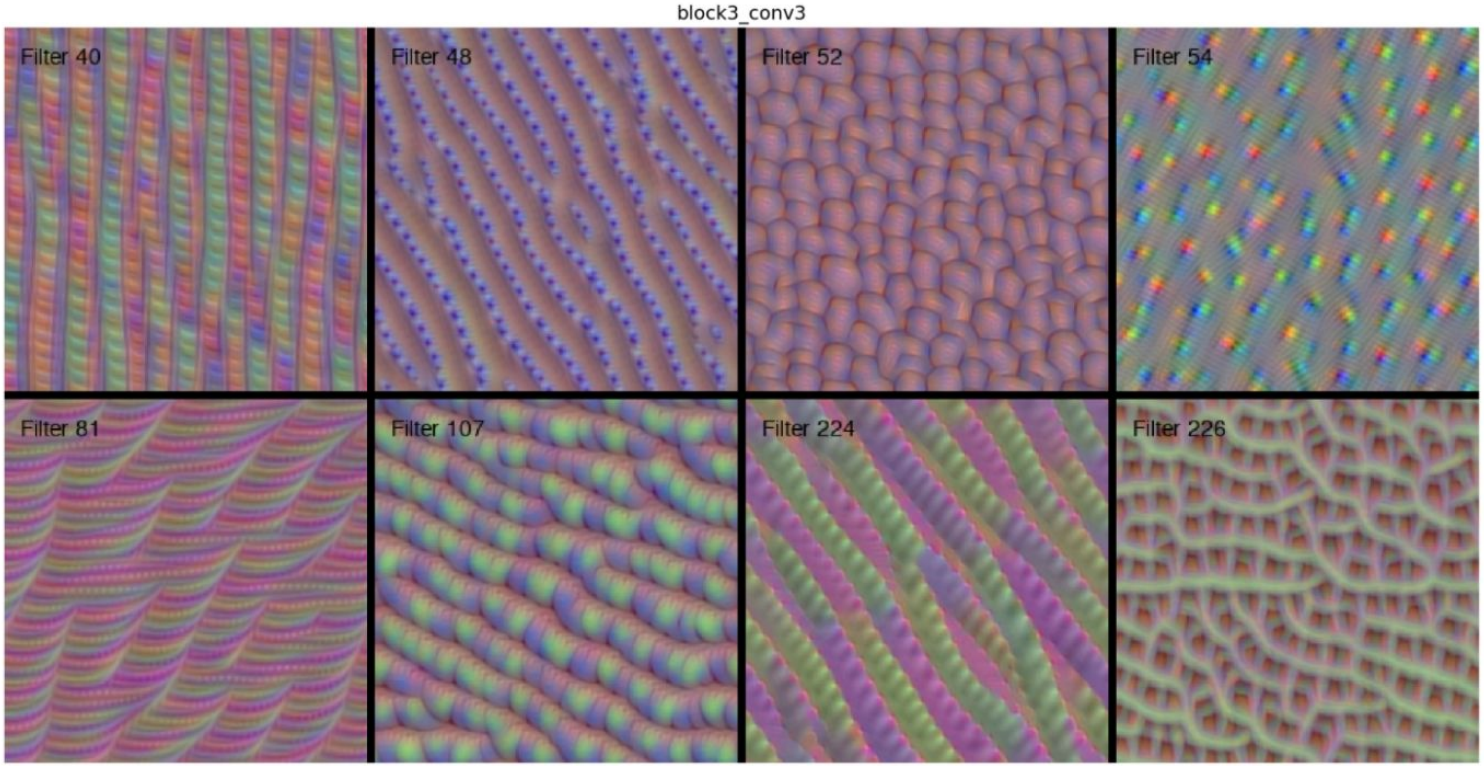
Convolutional Neural Networks - Visualizing Convnet Filters



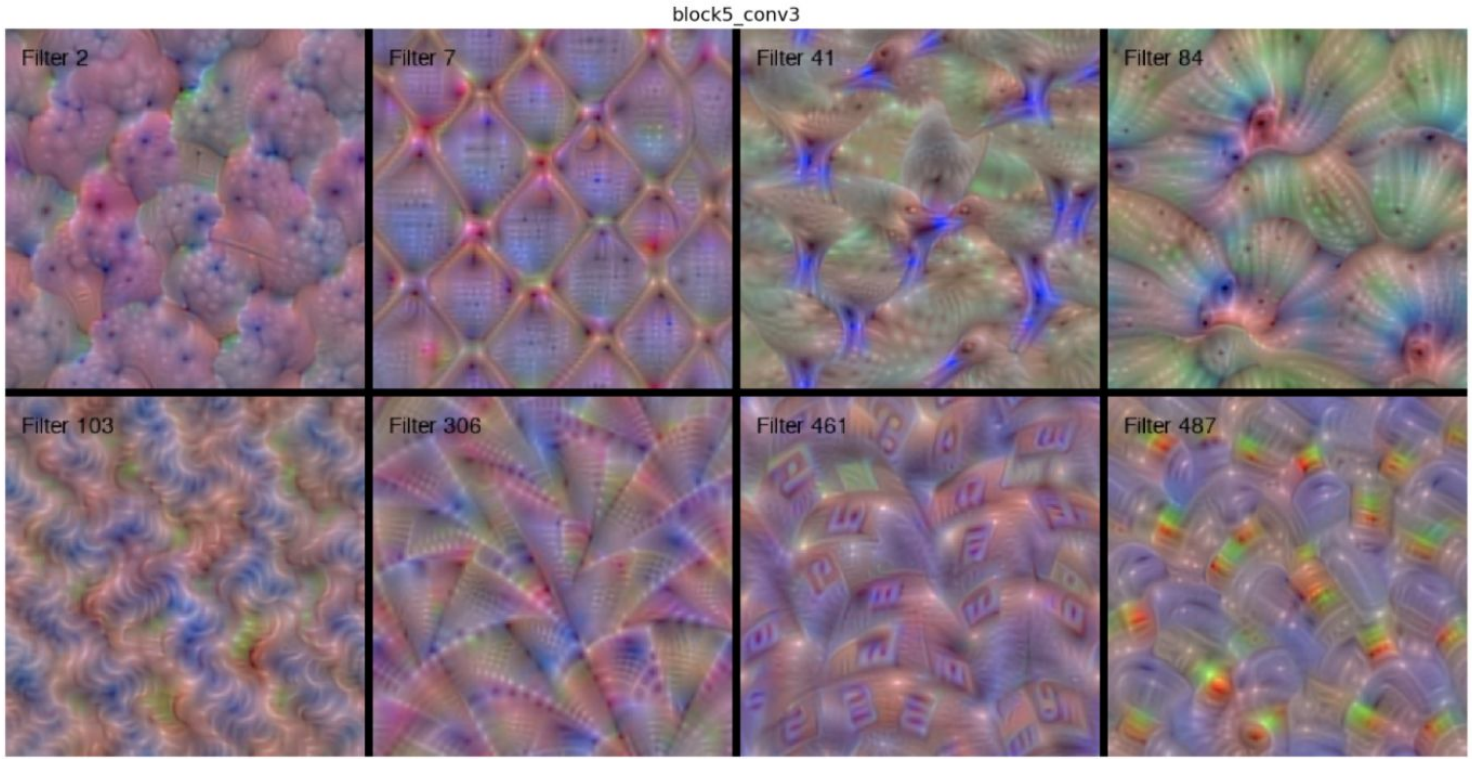
Convolutional Neural Networks - Visualizing Convnet Filters



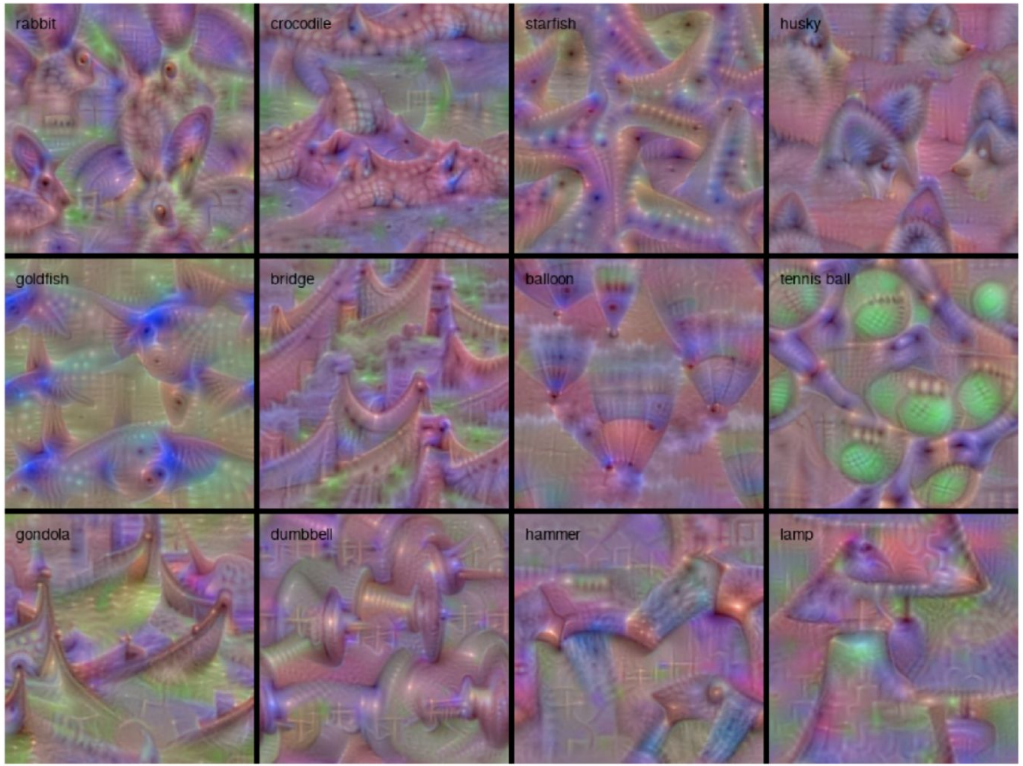
Convolutional Neural Networks - Visualizing Convnet Filters



Convolutional Neural Networks - Visualizing Convnet Filters



Convolutional Neural Networks - Visualizing Output Layers



Recurrent Neural Networks

Recurrent Neural Networks

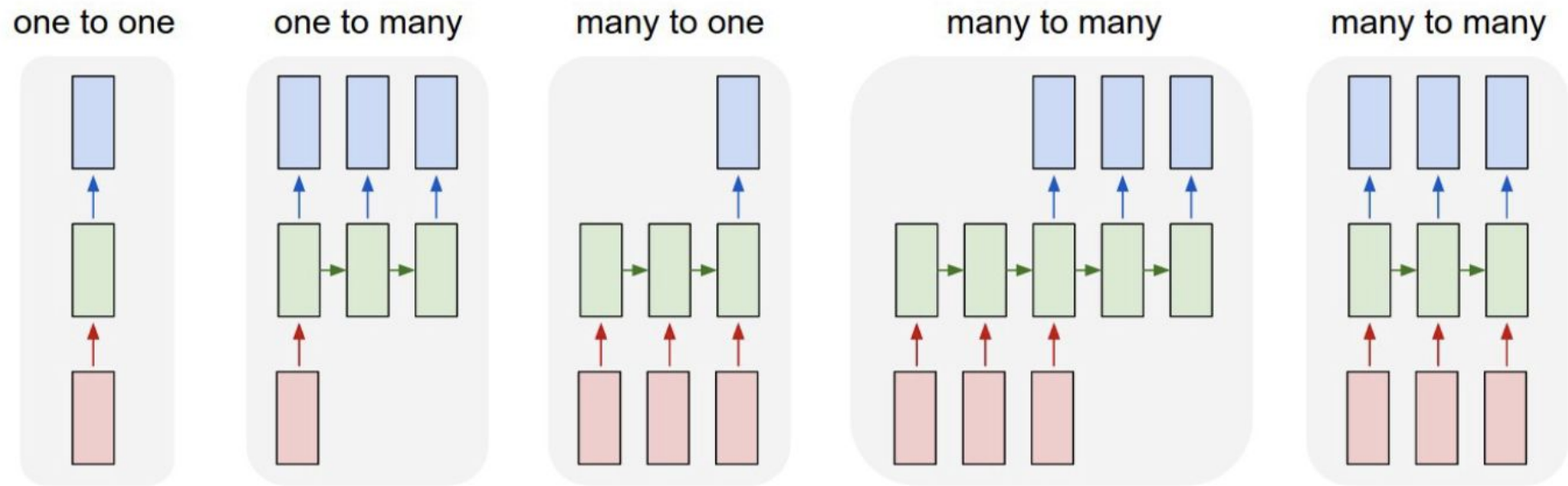
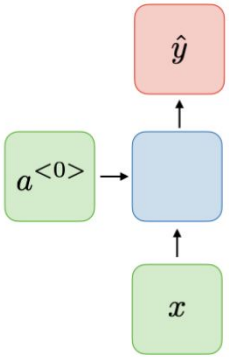
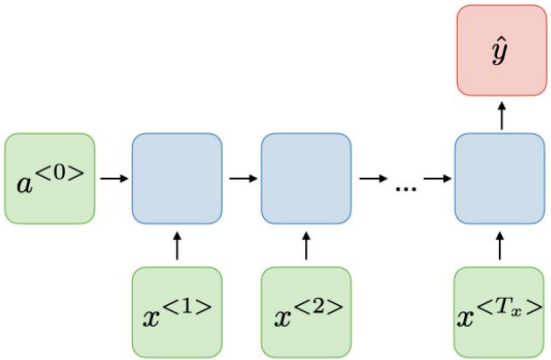


Image Classification | **Image Captioning** | **Sentiment Analysis** | **Machine Translation** | **Video Classification**

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

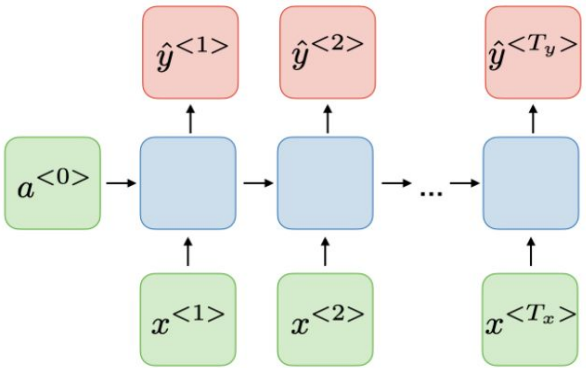
Recurrent Neural Networks

One-to-one $T_x = T_y = 1$		Traditional neural network
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

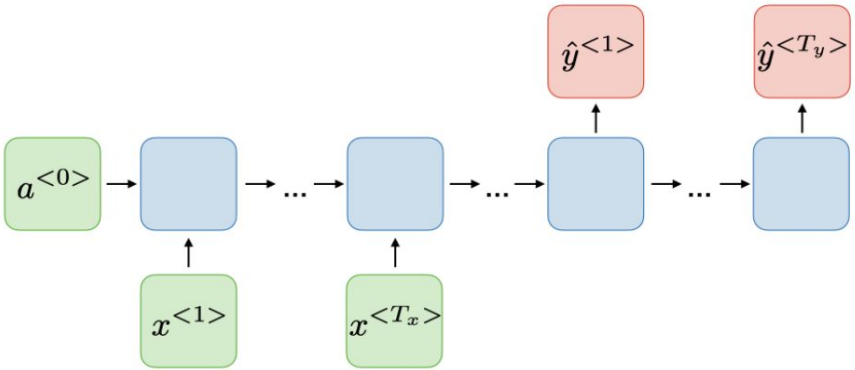
Recurrent Neural Networks

Many-to-many
$T_x = T_y$



Name entity recognition

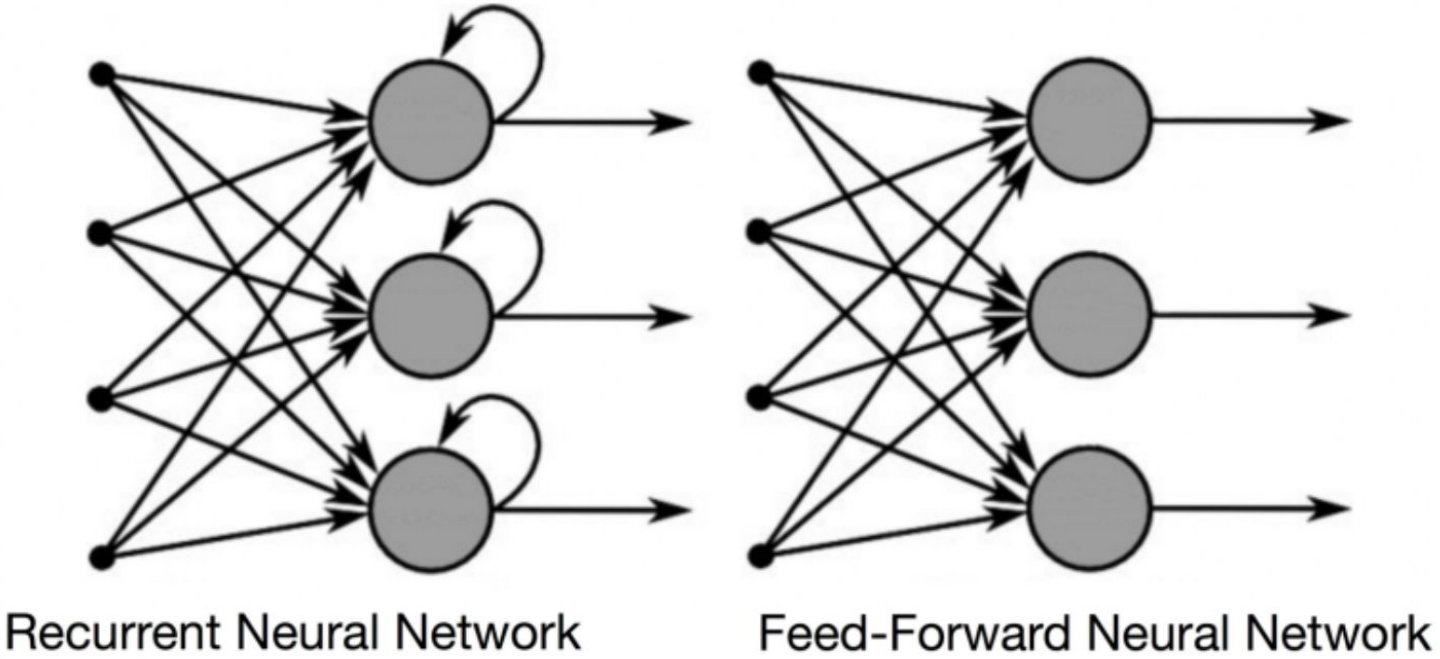
Many-to-many
$T_x \neq T_y$



Machine translation

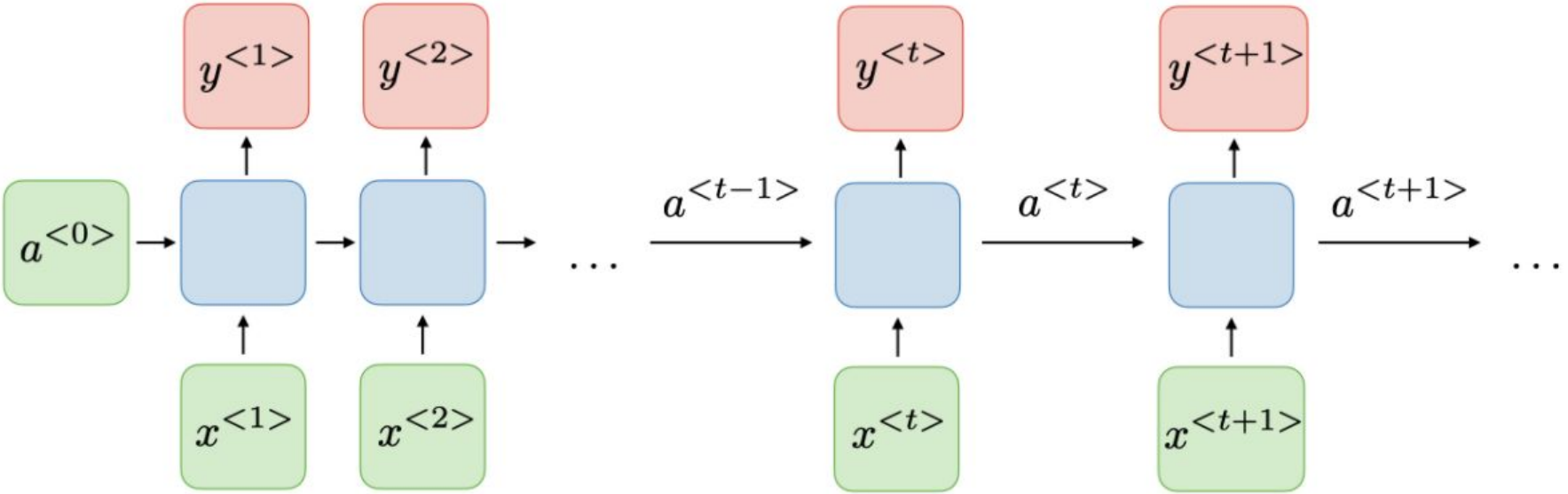
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Recurrent Neural Networks



https://builtin.com/data-science/recurrent-neural-networks-and-lstm

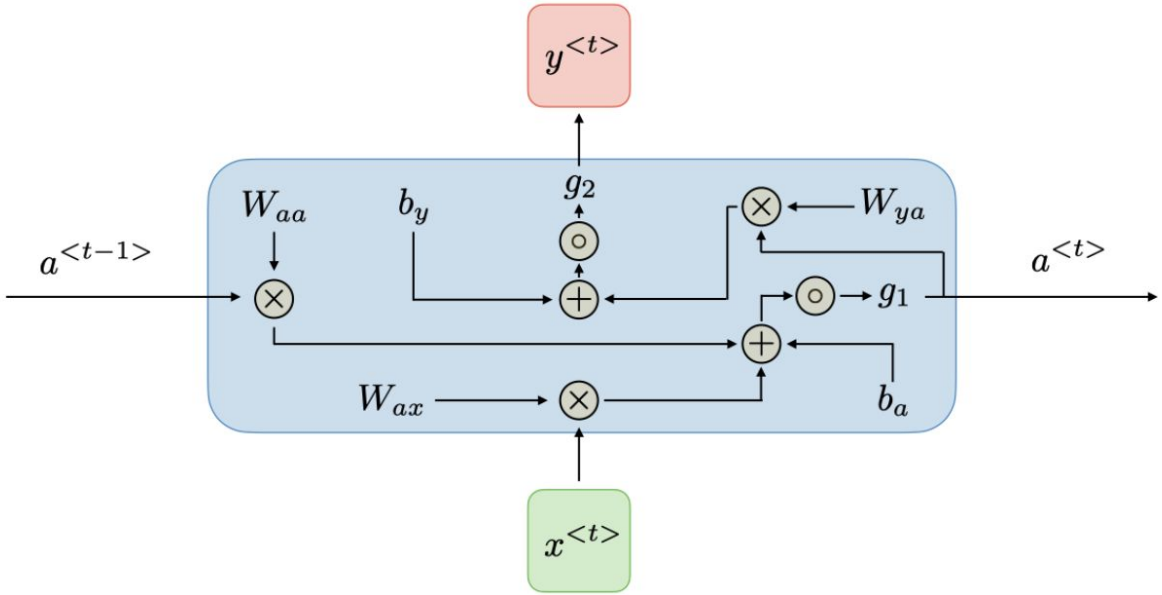
Recurrent Neural Networks



https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

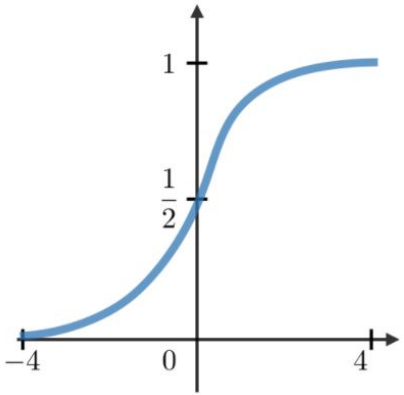
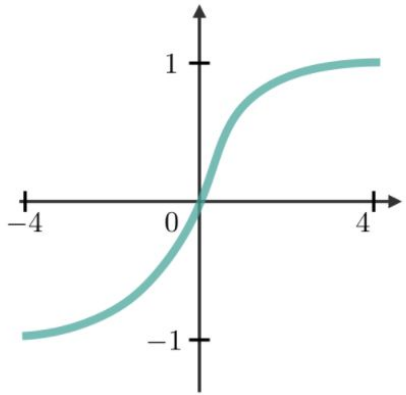
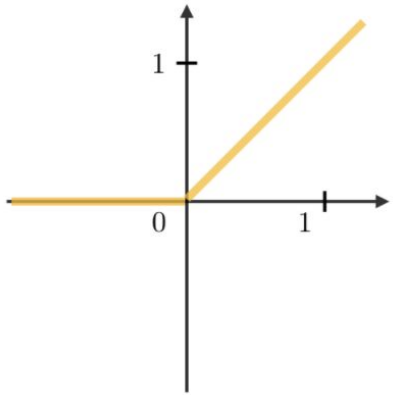
Recurrent Neural Networks

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$



https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Recurrent Neural Networks - Activation Functions

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Recurrent Neural Networks - Training

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

Loss function

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^{T} \left. \frac{\partial \mathcal{L}^{(T)}}{\partial W} \right|_{(t)}$$

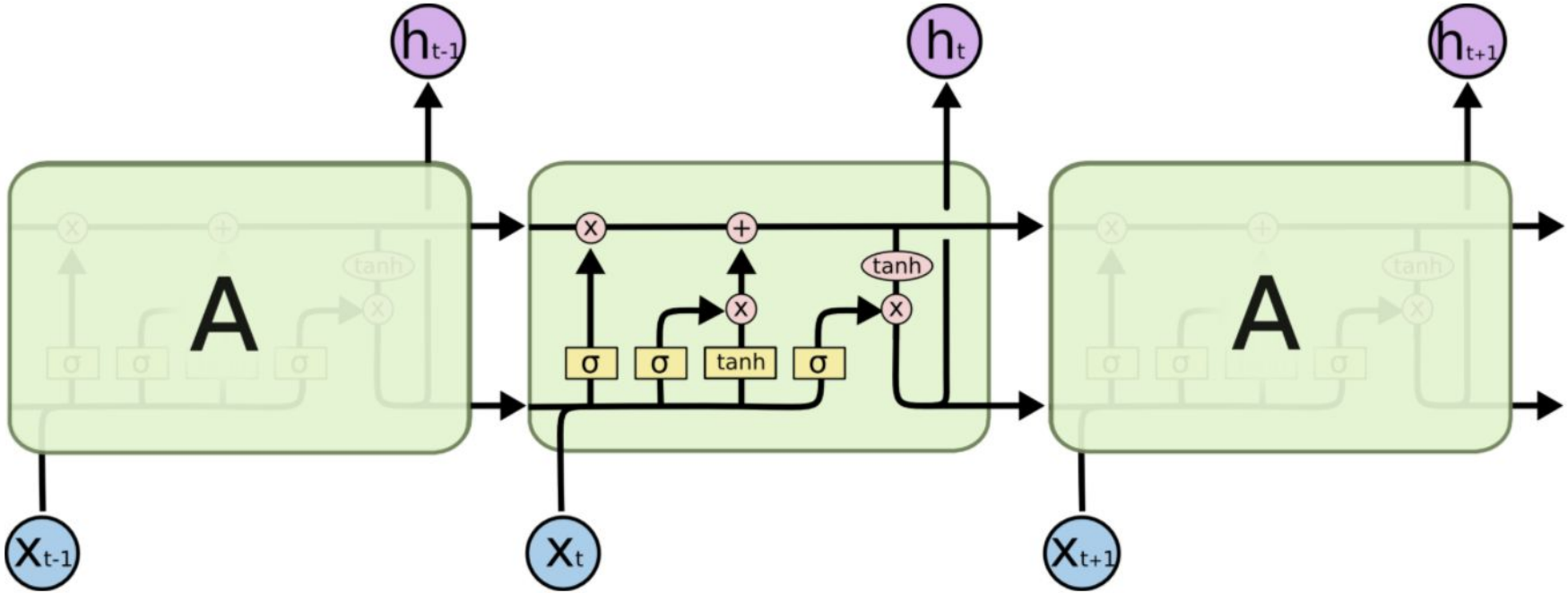
Backpropagation through time

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Recurrent Neural Networks - Limitations

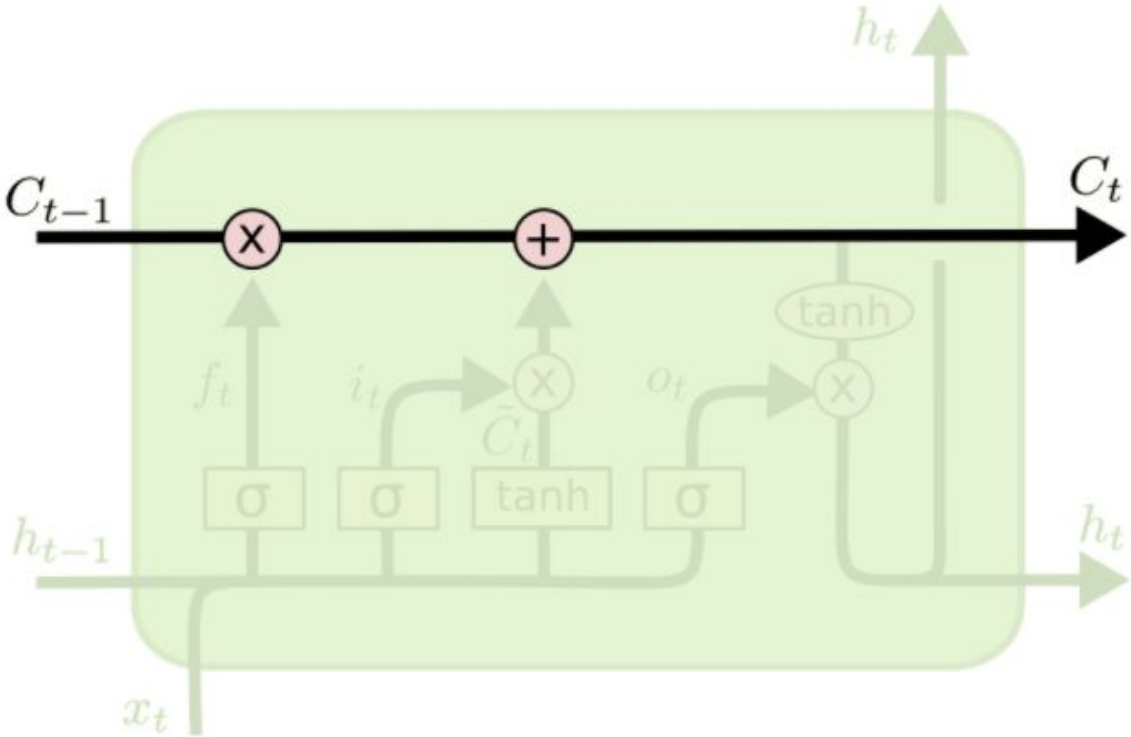
- Learning long-term dependencies
  - “I grew up in France... I speak fluent *French*.”
- Vanishing/exploding gradients

Recurrent Neural Networks - LSTMs



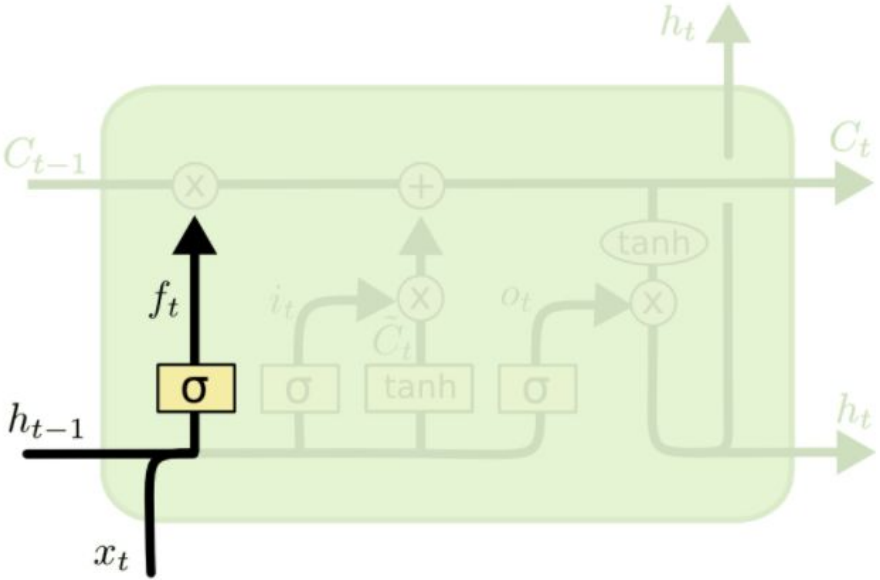
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks - LSTMs



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks - LSTMs

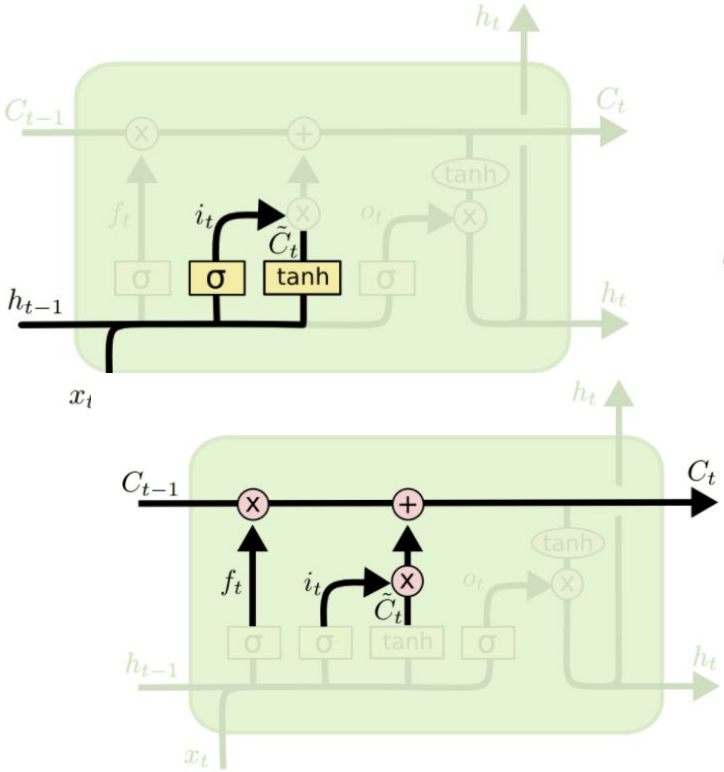


Forget gate

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks - LSTMs



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

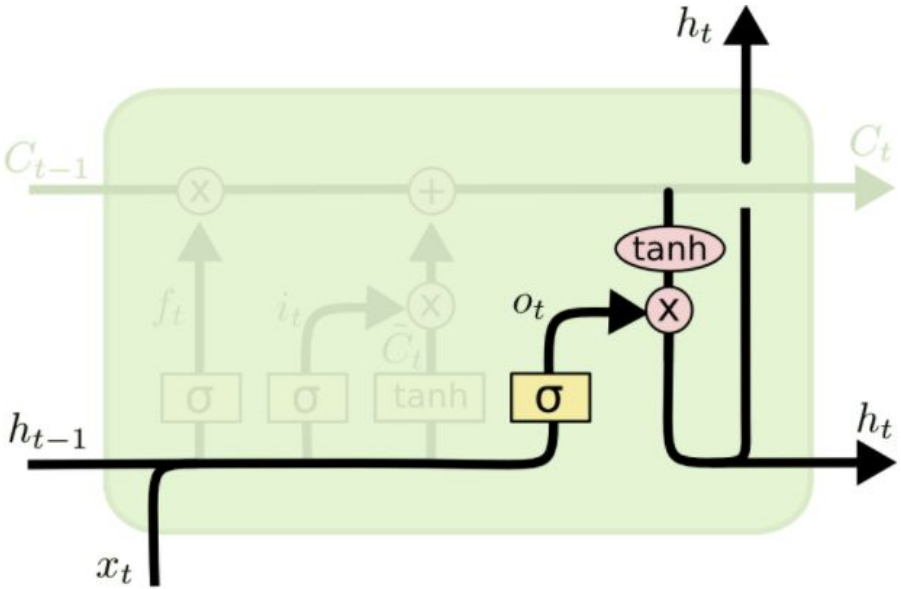
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

input gate

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks - LSTMs



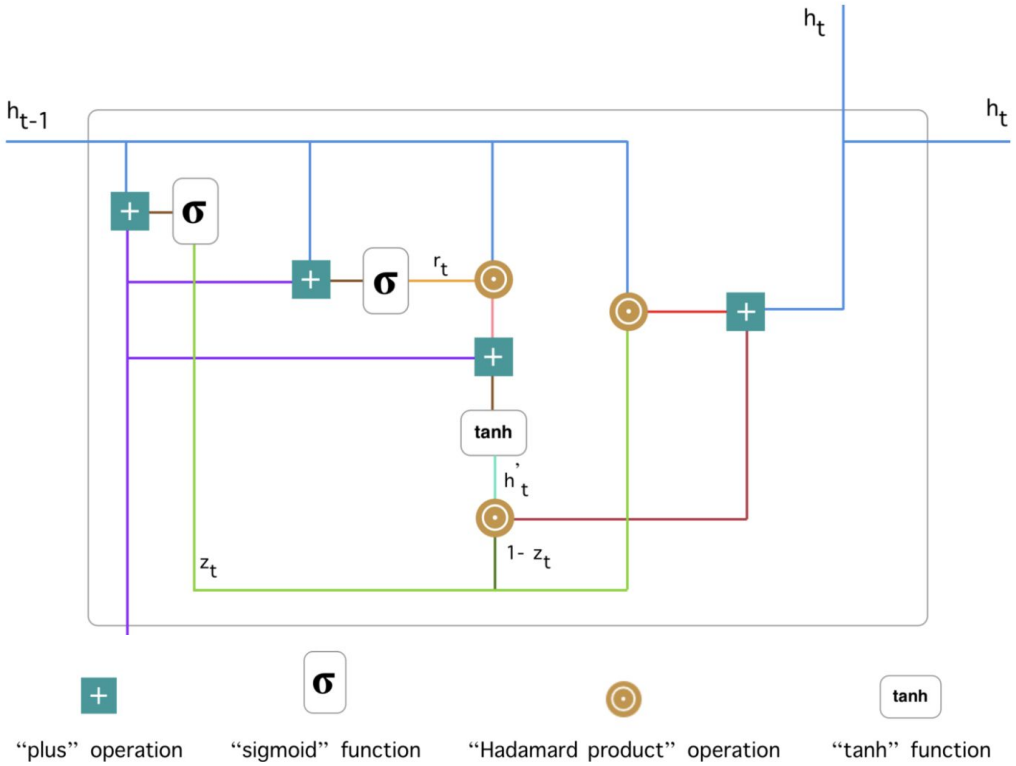
Output gate

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

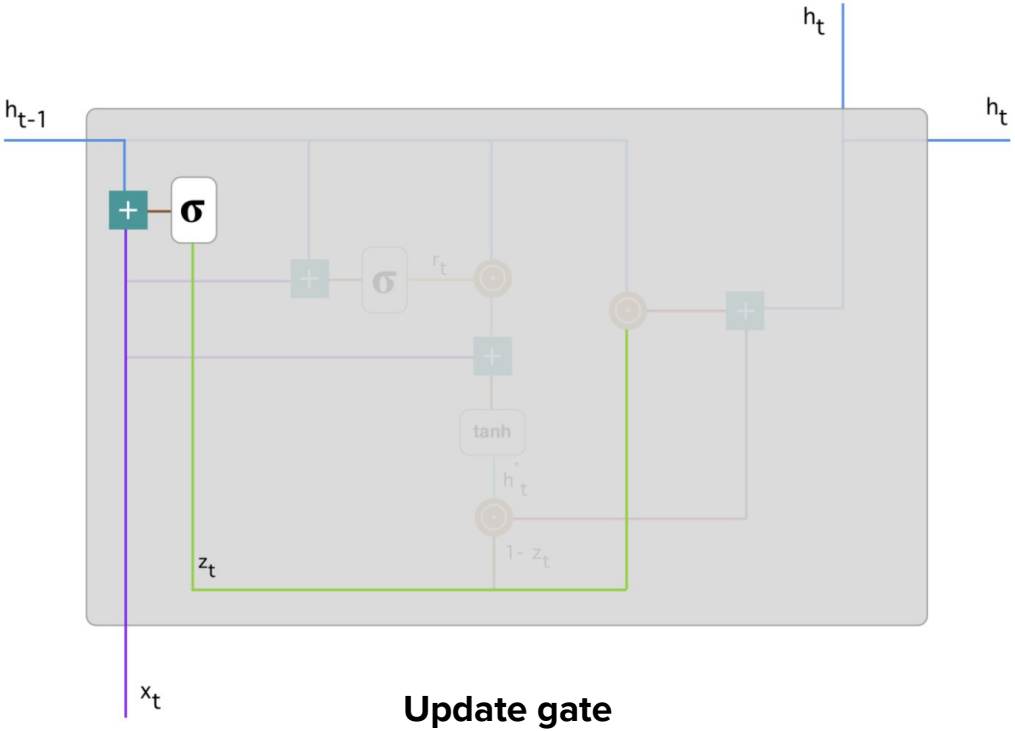
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks - GRUs



https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

Recurrent Neural Networks - GRUs

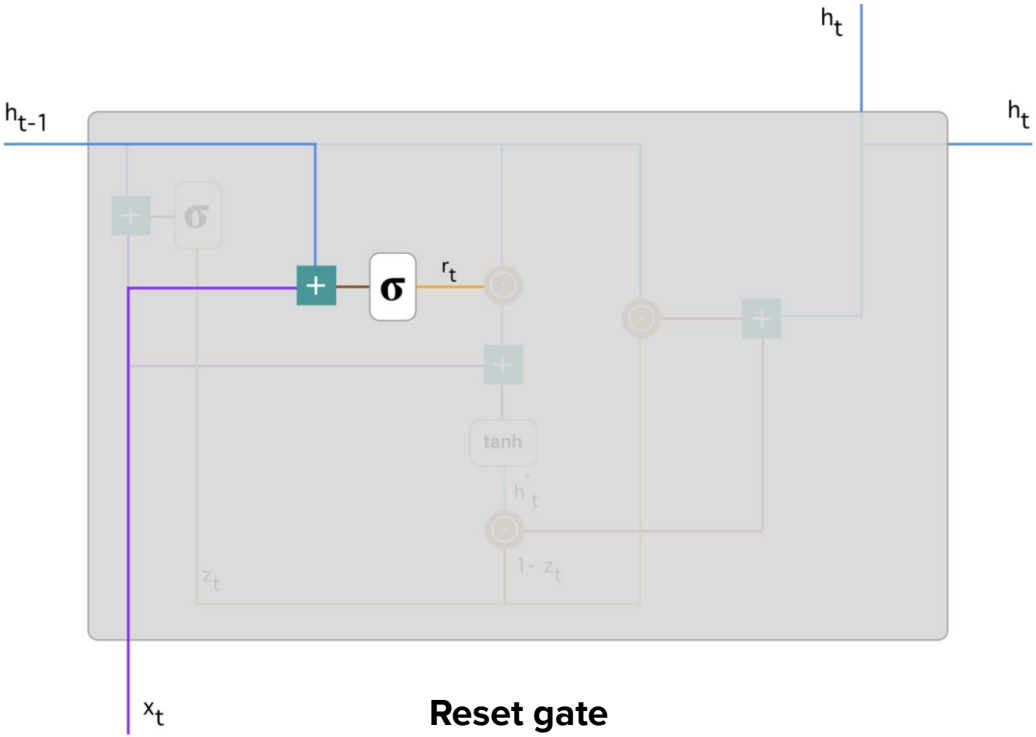


Update gate

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

Recurrent Neural Networks - GRUs

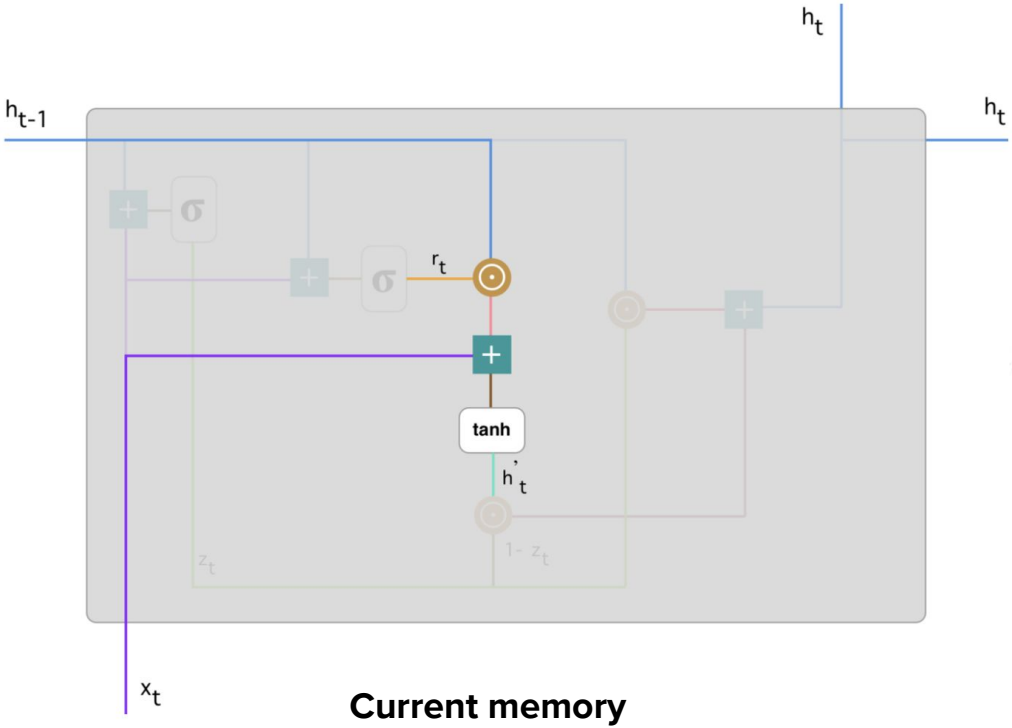


Reset gate

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

Recurrent Neural Networks - GRUs

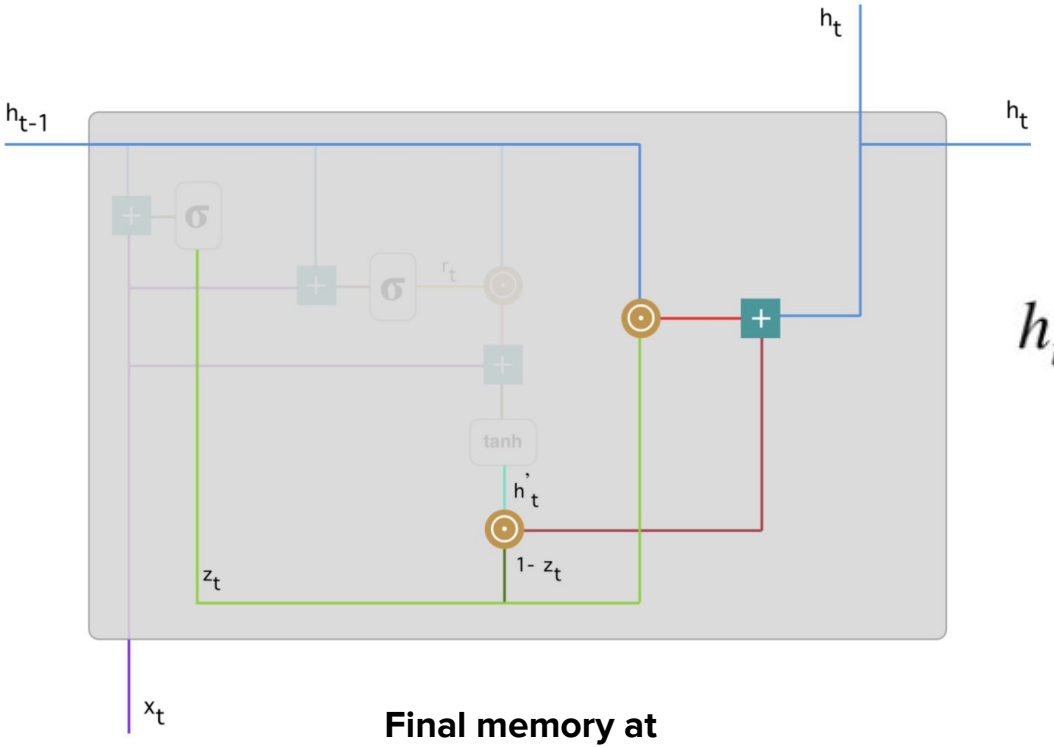


Current memory content

$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

Recurrent Neural Networks - GRUs



Final memory at current step

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$

https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

Questions?