CS 360: Machine Learning

Prof. Sara Mathieson

Fall 2020

Admin

- Office hours **today 4:30-6pm**
- Optional SVM material posted
- **Lab 8** due Friday Nov 20
  - Let me know if you would like individual deadlines
- After Thanksgiving – **two options for capstone**
  - Midterm 2 (midterm material ends Nov 20)
  - Final project (will post today)
  - Grade percentages have been updated

Outline for November 17

- Cross Entropy Loss and Handout 15
- Convolutional Neural Networks (CNNs)
- Math behind convolutions
- Handout 16

Outline for November 17

- Cross Entropy Loss and Handout 15
- Convolutional Neural Networks (CNNs)
- Math behind convolutions
- Handout 16

Notes about scores and softmax

- The output of the final fully connected layer is a vector of length *K* (number of classes)

- The raw scores are transformed into probabilities using the *softmax function*: (let s_k be the score for class *k*)

$$\hat{y}_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

- Then we apply *cross-entropy loss* to these probabilities

Notes about scores and softmax

- The output of the final fully connected layer is a vector of length *K* (number of classes)

- The raw scores are transformed into probabilities using the *softmax function*: (let s_k be the score for class *k*)

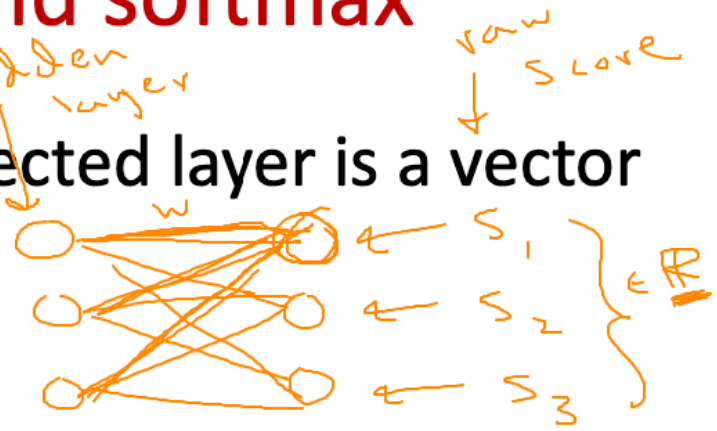
$$\hat{y}_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

Think about outside of class:
- Why do we use exp?
- Why don't we just take the max score?

- Then we apply *cross-entropy loss* to these probabilities

Notes about scores and softmax

- The output of the final fully connected layer is a vector of length *K* (number of classes)



- The raw scores are transformed into probabilities using the *softmax function*: (let s_k be the score for class *k*)

$$\hat{y}_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

activation
probabilities
$\vec{w} \cdot \vec{x}$

$$\hat{y}_k = \frac{s_k}{\sum_{j=1}^{K} s_j}$$

- Then we apply *cross-entropy loss* to these probabilities

Notes about scores and softmax

- The output of the final fully connected layer is a vector of length *K* (number of classes)

s_1 s_2 s_3

[0, 0, 0]

[5, 7, 2]

- The raw scores are transformed into probabilities using the *softmax function*: (let s_k be the score for class *k*)

$\hat{y} = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$

$$\hat{y}_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

[-75, 8, -20]

Think about outside of class:

- Why do we use exp?
- Why don't we just take the max score?

- Then we apply *cross-entropy loss* to these probabilities

Cross Entropy Loss

prob. dist.

$$H(p,q) = -\sum_{x \in X} p(x) \log q(x)$$

for us: true, pred

$$H(y,\hat{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

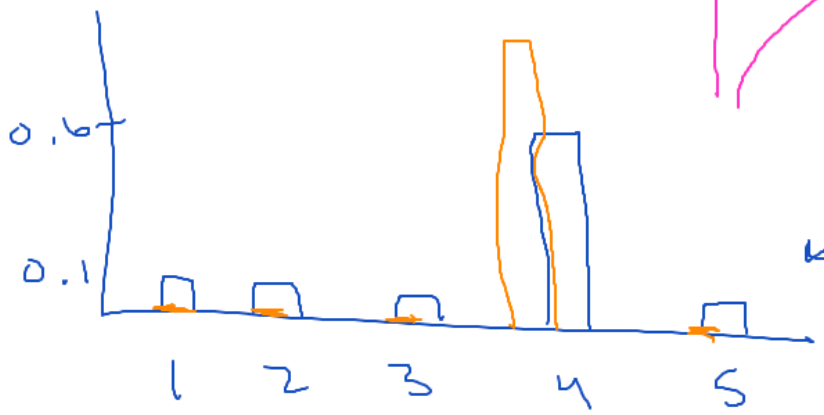
K = # classes

Example $K = 5$

true class is 4

$y = [0, 0, 0, 1, 0]$ (1, 2, 3, 4, 5)

"one-hot"



$\hat{y} = [0.1, 0.1, 0.1, 0.6, 0.1]$

0.2

$$H(y,\hat{y}) = -[0 \cdot \log(0.1) \cdot 4 + 1 \cdot \log(0.6)]$$

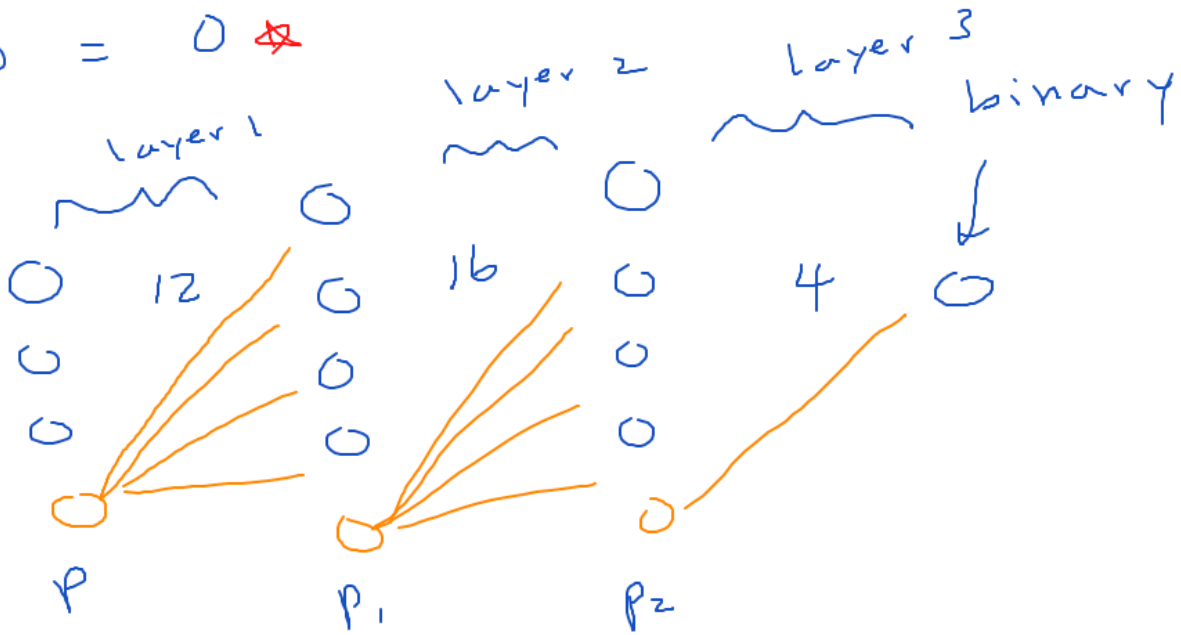
Handout 15

(1) ~~$\nabla(x) = 0.5$~~

$\tanh(x) = 0$

ReLU $f(x) = 0$ ✱

(2)



$(n, p) = (100, 3)$

↑ # examples

$12 + 16 + 4 + 4 + 4 + 1 = 41$

(3) $\left[\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\right]$ loss = 1 , $\left[\frac{3}{8}, \frac{1}{8}, \frac{1}{2}\right]$ ⇒3

Example in TensorFlow

```
class LogisticRegression(Model):

    def __init__(self):
        super(LogisticRegression, self).__init__()
        self.dense = Dense(2, activation='softmax')

    def call(self, x):
        return self.dense(x)
```

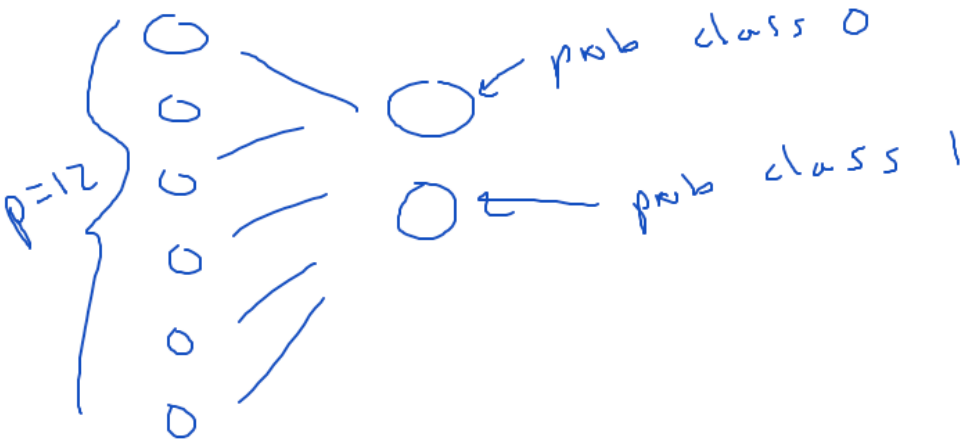
https://www.tensorflow.org/guide/effective_tf2

Example in TensorFlow

```
class LogisticRegression(Model):

    def __init__(self):
        super(LogisticRegression, self).__init__()
        self.dense = Dense(2, activation='softmax')

    def call(self, x):
        return self.dense(x)
```



https://www.tensorflow.org/guide/effective_tf2

Example in TensorFlow

```
X = np.random.rand(64, 12)
y = random.choices([0,1],k=64)

train_dset = tf.data.Dataset.from_tensor_slices((X, y)).batch(1)
```

```
logreg_model = LogisticRegression()
output = logreg_model.call(X)
print(output)

for v in logreg_model.trainable_variables:
    print("Variable: ", v.name)
    print("Shape: ", v.shape)
```

https://www.tensorflow.org/guide/effective_tf2

Example in TensorFlow

```
X = np.random.rand(64, 12)
y = random.choices([0,1],k=64)

train_dset = tf.data.Dataset.from_tensor_slices((X, y)).batch(1)
```

```
logreg_model = LogisticRegression()
output = logreg_model.call(X)
print(output)

for v in logreg_model.trainable_variable
    print("Variable: ", v.name)
    print("Shape: ", v.shape)
```

```
[[0.537787   0.46221307]
 [0.5326083  0.46739173]
 [0.63962054 0.36037943]
 [0.38047576 0.6195243 ]
 [0.6580146  0.34198546]
 [0.27457774 0.72542226]
 [0.63098085 0.36901915]
 [0.38368452 0.6163154 ]
 [0.34184858 0.65815145]
 [0.68468815 0.31531185]
 [0.42326403 0.5767359 ]
 [0.626032   0.37396798]
 [0.57787275 0.42212722]
 [0.47416764 0.5258323 ]
```

```
Variable:  dense/kernel:0
Shape:  (12, 2)
Variable:  dense/bias:0
Shape:  (2,)
```

https://www.tensorflow.org/guide/effective_tf2

Example in TensorFlow

```
X = np.random.rand(64, 12)
y = random.choices([0,1],k=64)

train_dset = tf.data.Dataset.from_tensor_slices((X, y)).batch(1)
```

```
logreg_model = LogisticRegression()
output = logreg_model.call(X)
print(output)

for v in logreg_model.trainable_variable
    print("Variable: ", v.name)
    print("Shape: ", v.shape)
```

```
[[0.537787   0.46221307]
 [0.5326083  0.46739173]
 [0.63962054 0.36037943]
 [0.38047576 0.6195243 ]
 [0.6580146  0.34198546]
 [0.27457774 0.72542226]
 [0.63098085 0.36901915]
 [0.38368452 0.6163154 ]
 [0.34184858 0.65815145]
 [0.68468815 0.31531185]
 [0.42326403 0.5767359 ]
 [0.626032   0.37396798]
 [0.57787275 0.42212722]
 [0.47416764 0.5258323 ]
```

```
Variable:  dense/kernel:0
Shape:  (12, 2)
Variable:  dense/bias:0
Shape:  (2,)
```

https://www.tensorflow.org/guide/effective_tf2

Example in TensorFlow

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam()

for e in range(100):
    for images, labels in train_dset:
        with tf.GradientTape() as tape:
            predictions = logreg_model(images)
            loss = loss_object(labels, predictions)
        gradients = tape.gradient(loss, logreg_model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, logreg_model.trainable_variables))

        print(labels.numpy(), predictions.numpy())
```

https://www.tensorflow.org/guide/effective_tf2

Example in TensorFlow

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam()

for e in range(100):
    for images, labels in train_dset:
        with tf.GradientTape() as tape:
            predictions = logreg_model(images)
            loss = loss_object(labels, predictions)
        gradients = tape.gradient(loss, logreg_model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, logreg_model.trainable_variables))

        print(labels.numpy(), predictions.numpy())
```

```
[0] [[0.537787   0.46221304]]
[0] [[0.53572917 0.46427083]]
[1] [[0.6453543  0.35464567]]
[0] [[0.38647363 0.6135264 ]]
[0] [[0.6648405  0.33515948]]
[1] [[0.2842549 0.7157451]]
[1] [[0.63803864 0.36196133]]
[0] [[0.39332294 0.606677  ]]
[0] [[0.35306314 0.6469369 ]]
[0] [[0.696235 0.303765]]
[1] [[0.43879274 0.56120723]]
```

https://www.tensorflow.org/guide/effective_tf2

Example in TensorFlow

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy()
optimizer = tf.keras.optimizers.Adam()

for e in range(100):
    for images, labels in train_dset:
        with tf.GradientTape() as tape:
            predictions = logreg_model(images)
            loss = loss_object(labels, predictions)
        gradients = tape.gradient(loss, logreg_model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, logreg_model.trainable_variables))

        print(labels.numpy(), predictions.numpy())
```

Friday

100 epoch

"SGD"

```
[0] [[0.537787   0.46221304]]
[0] [[0.53572917 0.46427083]]
[1] [[0.6453543  0.35464567]]
[0] [[0.38647363 0.6135264 ]]
[0] [[0.6648405  0.33515948]]
[1] [[0.2842549  0.7157451 ]]
[1] [[0.63803864 0.36196133]]
[0] [[0.39332294 0.606677  ]]
[0] [[0.35306314 0.6469369 ]]
[0] [[0.696235 0.303765]]
[1] [[0.43879274 0.56120723]]
```

y $\hat{y}$

sparse

https://www.tensorflow.org/guide/effective_tf2

Outline for November 17

- Cross Entropy Loss and Handout 15
- Convolutional Neural Networks (CNNs)
- Math behind convolutions
- Handout 16

Motivation for moving away from FC architectures

- For a 32x32x3 image (very small!) we have p=3072 features in the input layer

- For a 200x200x3 image, we would have p=120,000! *doesn't scale*

Motivation for moving away from FC architectures

- For a 32x32x3 image (very small!) we have p=3072 features in the input layer

- For a 200x200x3 image, we would have p=120,000! *doesn't scale*

- FC networks do not explicitly account for the structure of an image and the correlations/ relationships between nearby pixels

Idea: 3D volumes of neurons

- Do not “flatten” image, keep it as a volume with *width*, *height*, and *depth*

Idea: 3D volumes of neurons

- Do not “flatten” image, keep it as a volume with *width*, *height*, and *depth*
- For **CIFAR-10**, we would have:
  - Width=32, Height=32, Depth=3

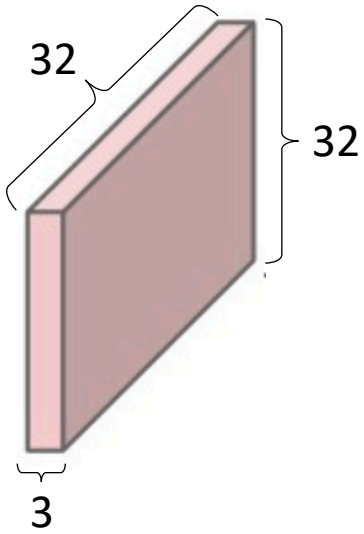


Image: modified from Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Idea: 3D volumes of neurons

- Do not “flatten” image, keep it as a volume with *width*, *height*, and *depth*
- For **CIFAR-10**, we would have:
  - Width=32, Height=32, Depth=3
- Each layer is also a 3 dimensional volume

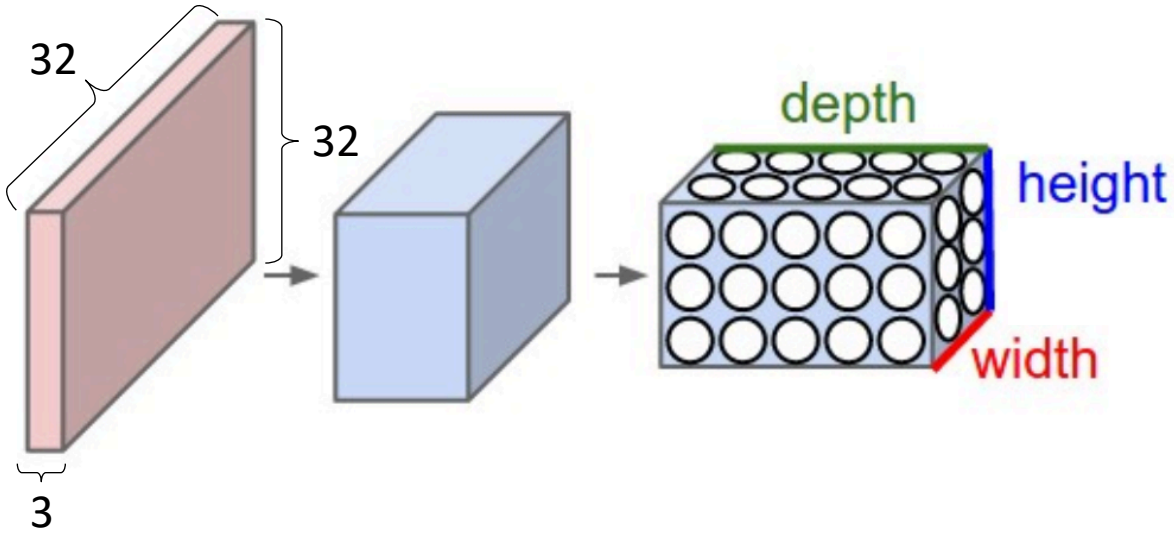


Image: modified from Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Idea: 3D volumes of neurons

- Do not “flatten” image, keep it as a volume with *width*, *height*, and *depth*
- For **CIFAR-10**, we would have:
  - Width=32, Height=32, Depth=3
- Each layer is also a 3 dimensional volume
- The output layer is 1x1xC, where C is the number of classes (10 for CIFAR-10)

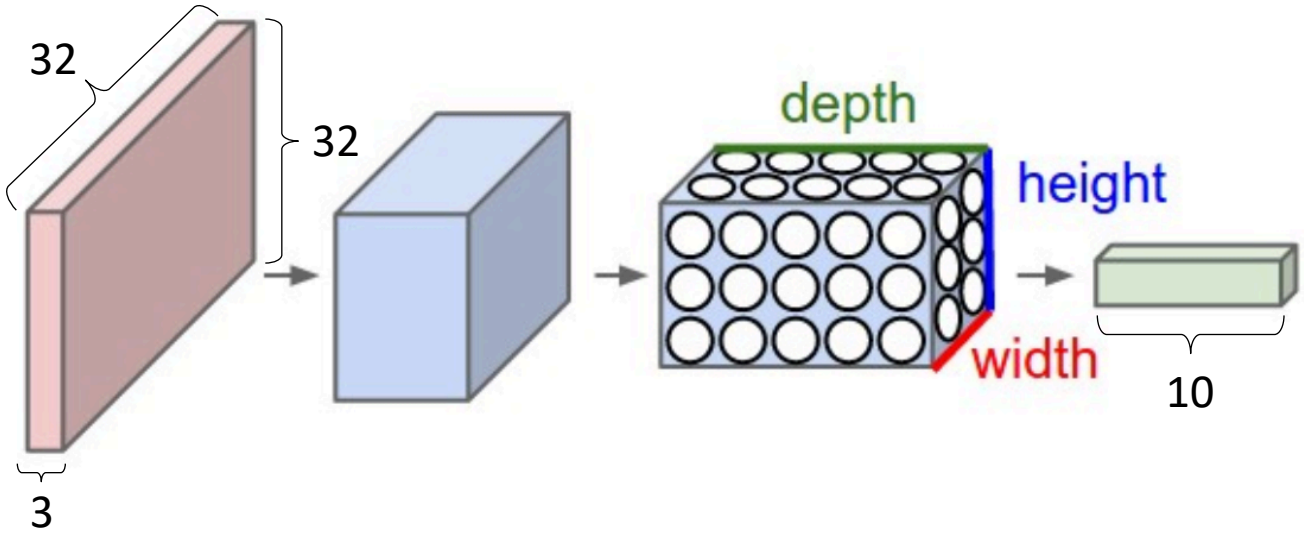


Image: modified from Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Layers of a Convolutional Neural Network (CNN)

- **INPUT**: raw pixels of a color image, i.e. 32x32x3

Layers of a Convolutional Neural Network (CNN)

- **INPUT**: raw pixels of a color image, i.e. 32x32x3

- **CONV**: compute information about a local region of the image using a filter. Example: 12 filters would product a volume of 32x32x12

More info: http://cs231n.github.io/convolutional-networks/

Layers of a Convolutional Neural Network (CNN)

- **INPUT**: raw pixels of a color image, i.e. 32x32x3

- **CONV**: compute information about a local region of the image using a filter. Example: 12 filters would product a volume of 32x32x12

- **RELU**: apply max(0,x), same volume 32x32x12

More info: http://cs231n.github.io/convolutional-networks/

Layers of a Convolutional Neural Network (CNN)

- **INPUT**: raw pixels of a color image, i.e. 32x32x3

- **CONV**: compute information about a local region of the image using a filter. Example: 12 filters would product a volume of 32x32x12

- **RELU**: apply max(0,x), same volume 32x32x12

- **POOL**: downsample, i.e. with result 16x16x12

More info: http://cs231n.github.io/convolutional-networks/

Layers of a Convolutional Neural Network (CNN)

- **INPUT**: raw pixels of a color image, i.e. 32x32x3
- **CONV**: compute information about a local region of the image using a filter. Example: 12 filters would product a volume of 32x32x12
- **RELU**: apply max(0,x), same volume 32x32x12
- **POOL**: downsample, i.e. with result 16x16x12
- **FC** (fully-connected): produce probabilities for each class, i.e. volume 1x1x10

More info: http://cs231n.github.io/convolutional-networks/

Example CNN architecture

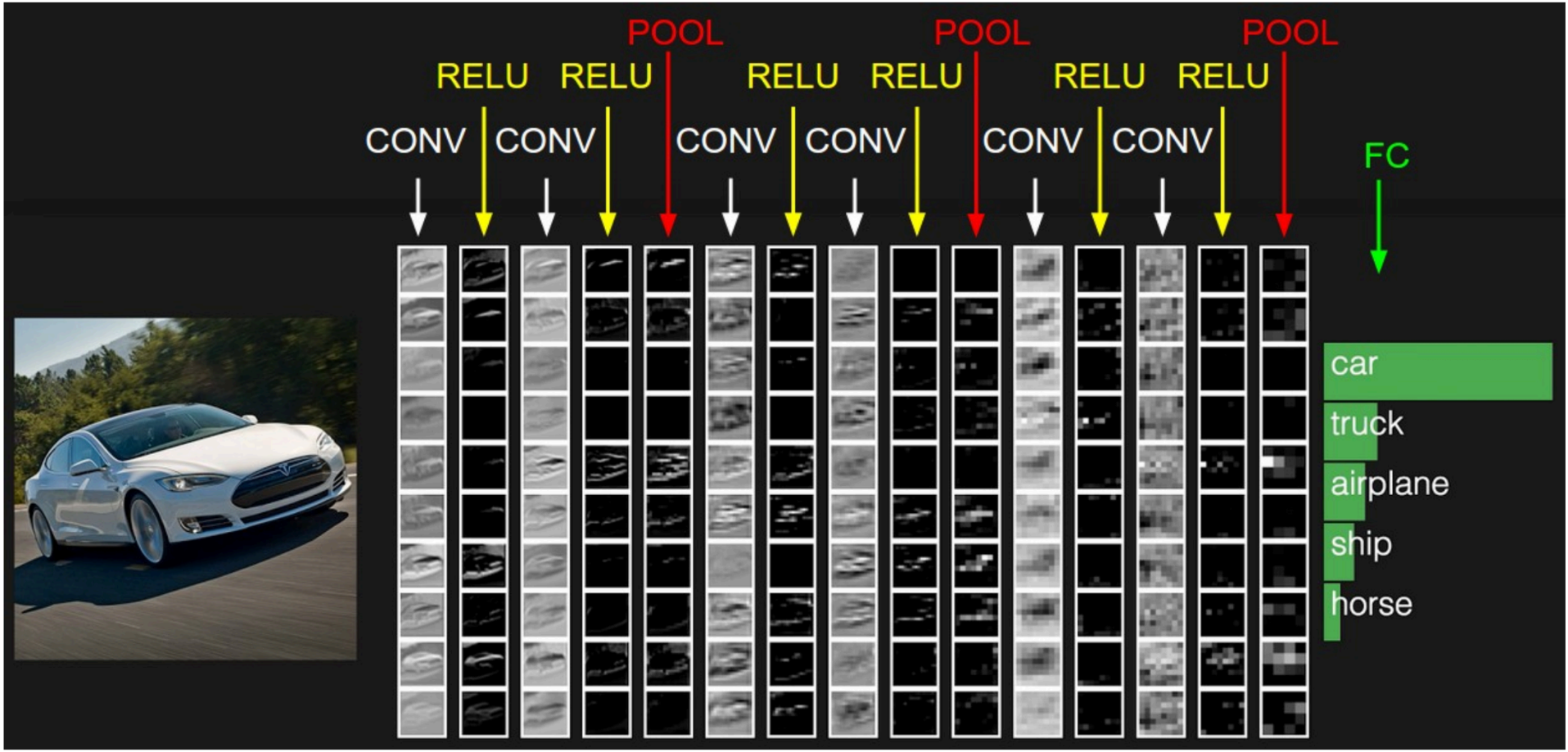
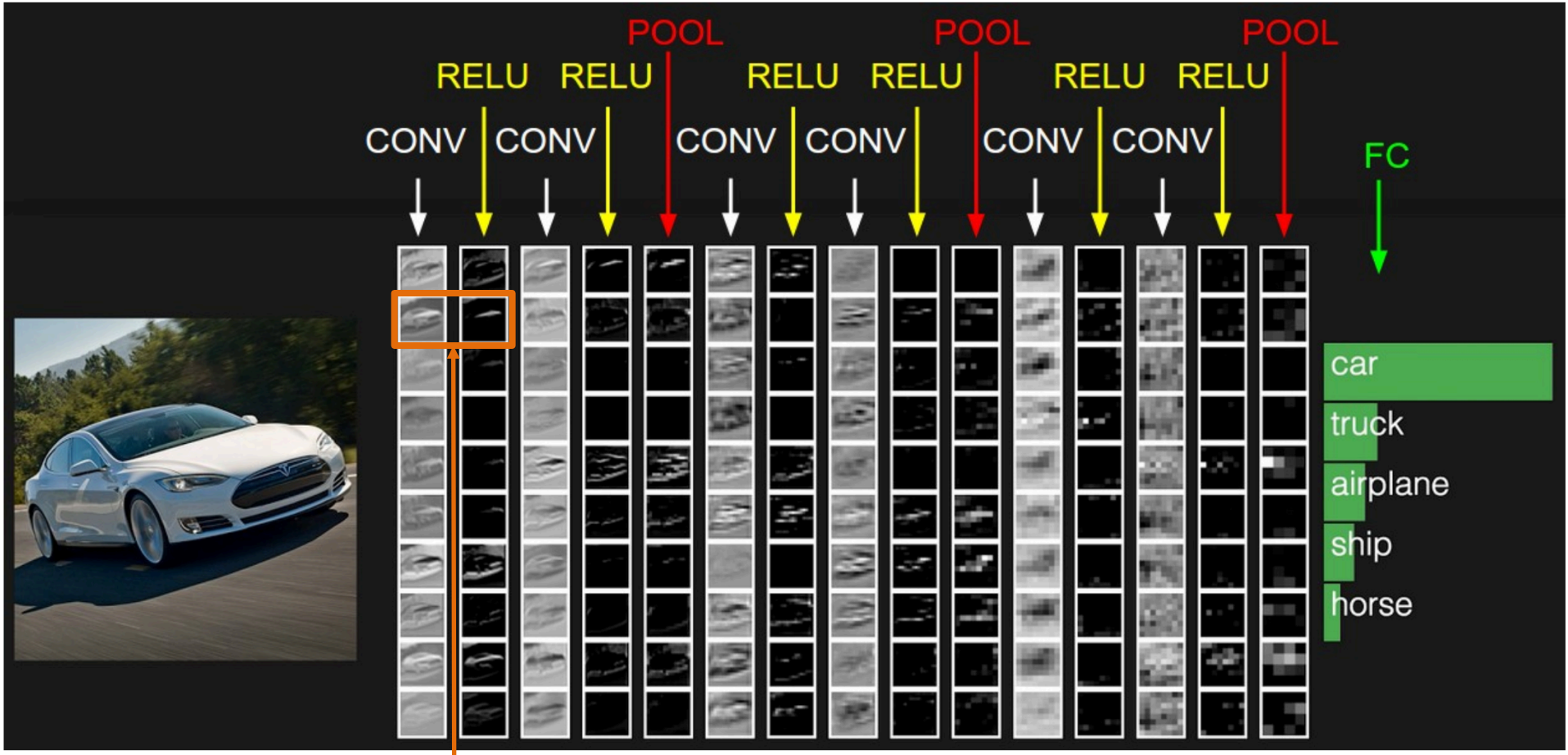


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

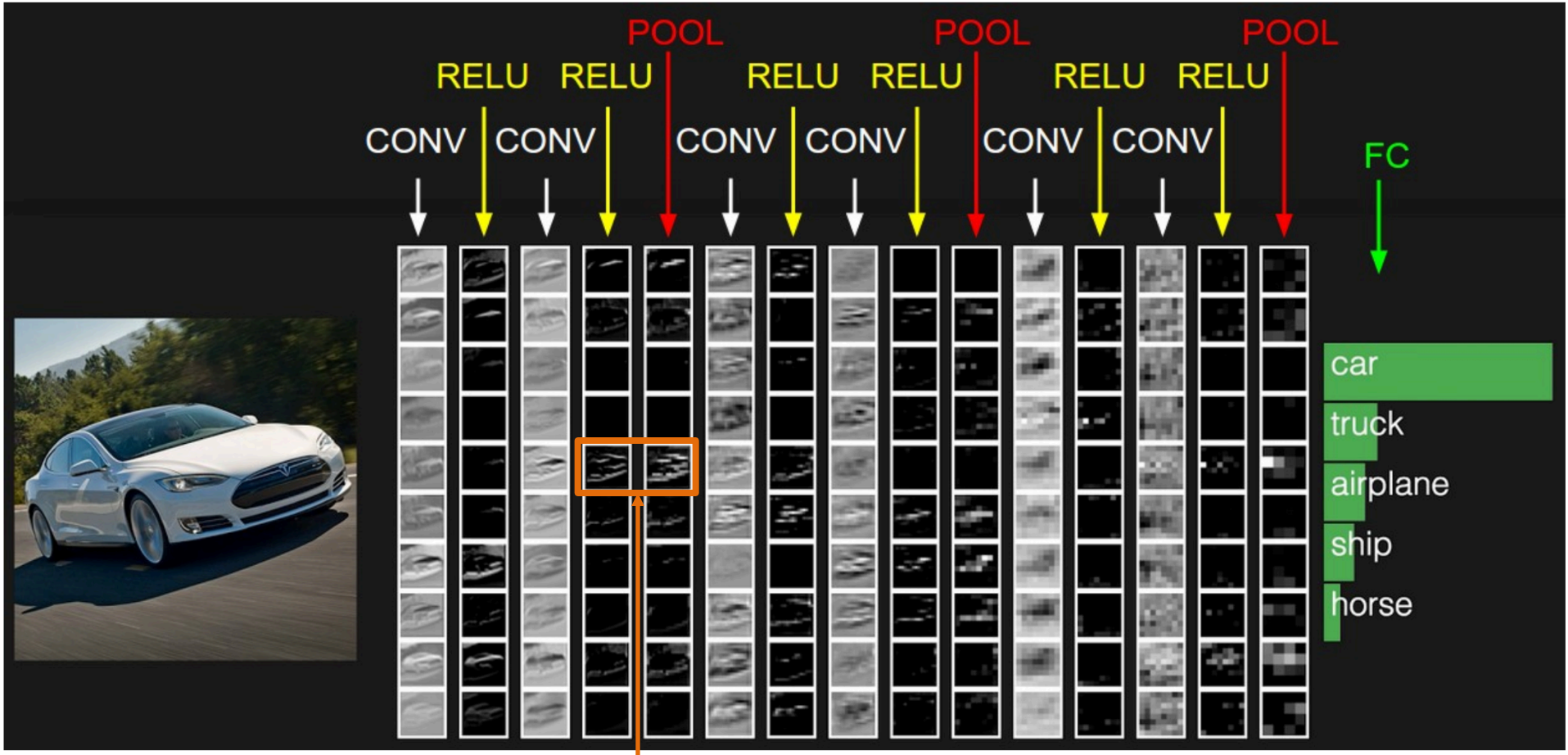
Example CNN architecture



ReLU zeros out less relevant information, highlighting an interesting feature (i.e. hood of car here)

Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Example CNN architecture



POOL reduces the size of the volume
but keeps relevant features

Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Visualization of an entire network

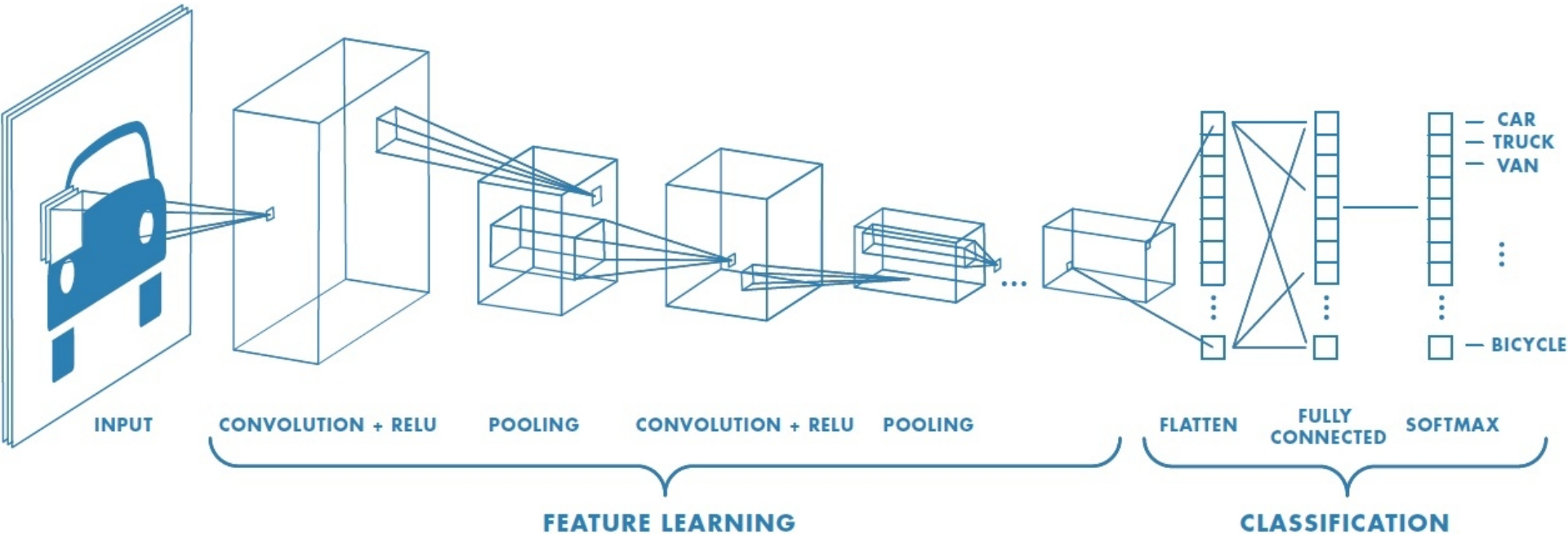


Image from MathWorks: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

Idea: local “receptive field”

- A convolutional filter (matrix) can pick up on local features in the original image through an element-wise dot-product

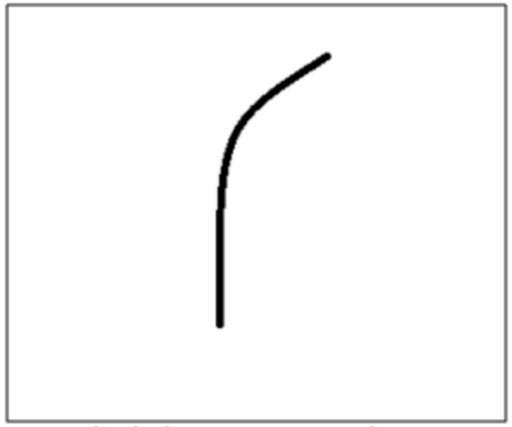
- Note an important *asymmetry*: we will look at a small “patch” of the image relative to its width and height, but we will look all the way through the depth!

Intuition: as learning progresses, filters become specialized for certain types of features

Example:
“Curve” filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

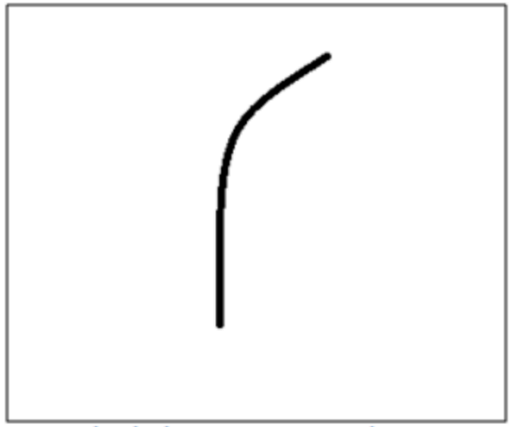
Images by: Adit Deshpande

Intuition: as learning progresses, filters become specialized for certain types of features

Example:
"Curve" filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

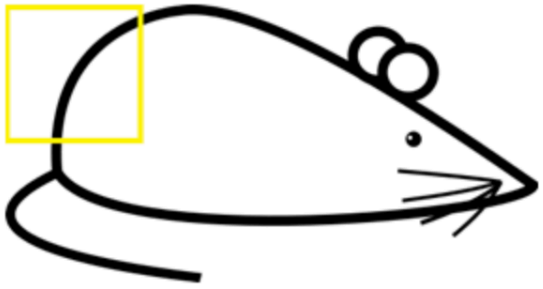
Pixel representation of filter



Visualization of a curve detector filter

Say we apply this filter to an image

Original image



Visualization of the filter on the image

Images by: Adit Deshpande

Output of convolutions will “light up” if filter “matches” receptive field, but not otherwise

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Output to next layer:
6600

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Images by: Adit Deshpande

Output of convolutions will “light up” if filter “matches” receptive field, but not otherwise

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Output to next layer:
6600

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Output to next layer:
0

Multiplication and Summation = 0

Images by: Adit Deshpande