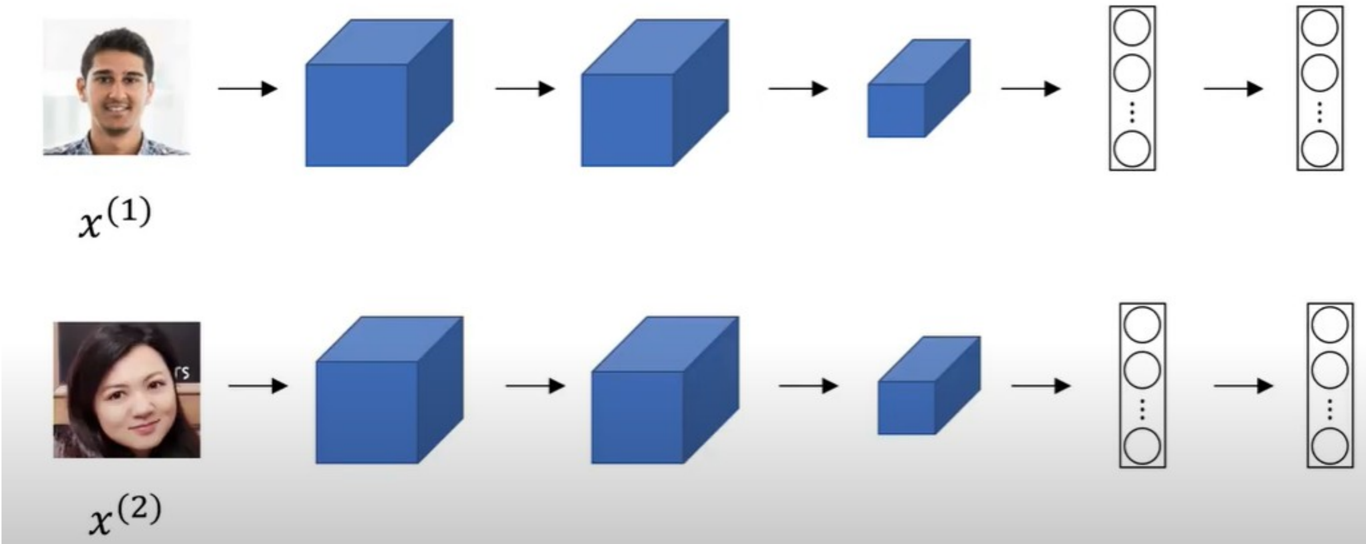
Computer Vision and Deep Learning

Lecture 8

Siamese networks

Two or more inputs are encoded and the output features are compared

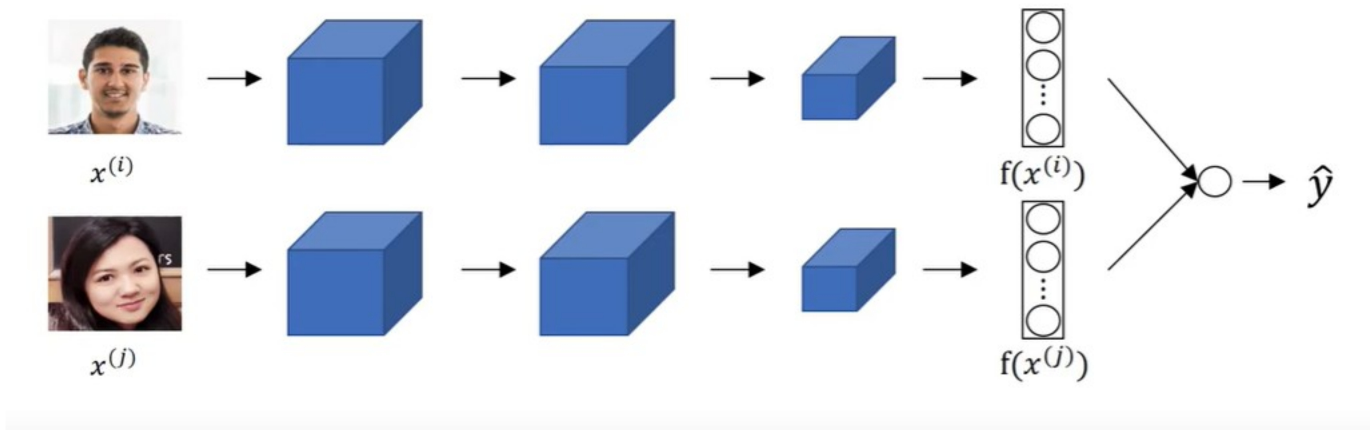


If $x^{(i)}, x^{(j)}$ are the same person, $\left\|f(x^{(i)}) - f(x^{(j)})\right\|^2$ is small.

If $x^{(i)}, x^{(j)}$ are different persons, $\left\|f(x^{(i)}) - f(x^{(j)})\right\|^2$ is large.

Example credit: Andrew Ng, deeplearning.ai, Convolutional Neural Networks

Face verification as a binary classification problem



σ ¿

Example credit: Andrew Ng, deeplearning.ai, Convolutional Neural Networks

```python
def make_pairs(images, labels):
    # initialize two empty lists to hold the (image, image) pairs and
    # labels to indicate if a pair is positive or negative
    pairImages = []
    pairLabels = []

    # calculate the total number of classes present in the dataset
    # and then build a list of indexes for each class label that
    # provides the indexes for all examples with a given label
    numClasses = len(np.unique(labels))
    idx = [np.where(labels == i)[0] for i in range(0, numClasses)]

    # loop over all images
    for idxA in range(len(images)):
        # grab the current image and label belonging to the current
        # iteration
        currentImage = images[idxA]
        label = labels[idxA]

        # randomly pick an image that belongs to the *same* class
        # label
        idxB = np.random.choice(idx[label])
        posImage = images[idxB]

        # prepare a positive pair and update the images and labels
        # lists, respectively
        pairImages.append([currentImage, posImage])
        pairLabels.append([1])

        # grab the indices for each of the class labels *not* equal to
        # the current label and randomly pick an image corresponding
        # to a label *not* equal to the current label
        negIdx = np.where(labels != label)[0]
        negImage = images[np.random.choice(negIdx)]

        # prepare a negative pair of images and update our lists
        pairImages.append([currentImage, negImage])
        pairLabels.append([0])

    # return a 2-tuple of our image pairs and labels
    return (np.array(pairImages), np.array(pairLabels))
```

https://www.pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/

```
def build_siamese_model(inputShape, embeddingDim=48):
    # specify the inputs for the feature extractor network
    inputs = Input(inputShape)
    # define the first set of CONV => RELU => POOL => DROPOUT layers
    x = Conv2D(64, (2, 2), padding="same", activation="relu")(inputs)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.3)(x)
    # second set of CONV => RELU => POOL => DROPOUT layers
    x = Conv2D(64, (2, 2), padding="same", activation="relu")(x)
    x = MaxPooling2D(pool_size=2)(x)
    x = Dropout(0.3)(x)
    # prepare the final outputs
    pooledOutput = GlobalAveragePooling2D()(x)
    outputs = Dense(embeddingDim)(pooledOutput)
    # build the model
    model = Model(inputs, outputs)
    # return the model to the calling function
    return model
```

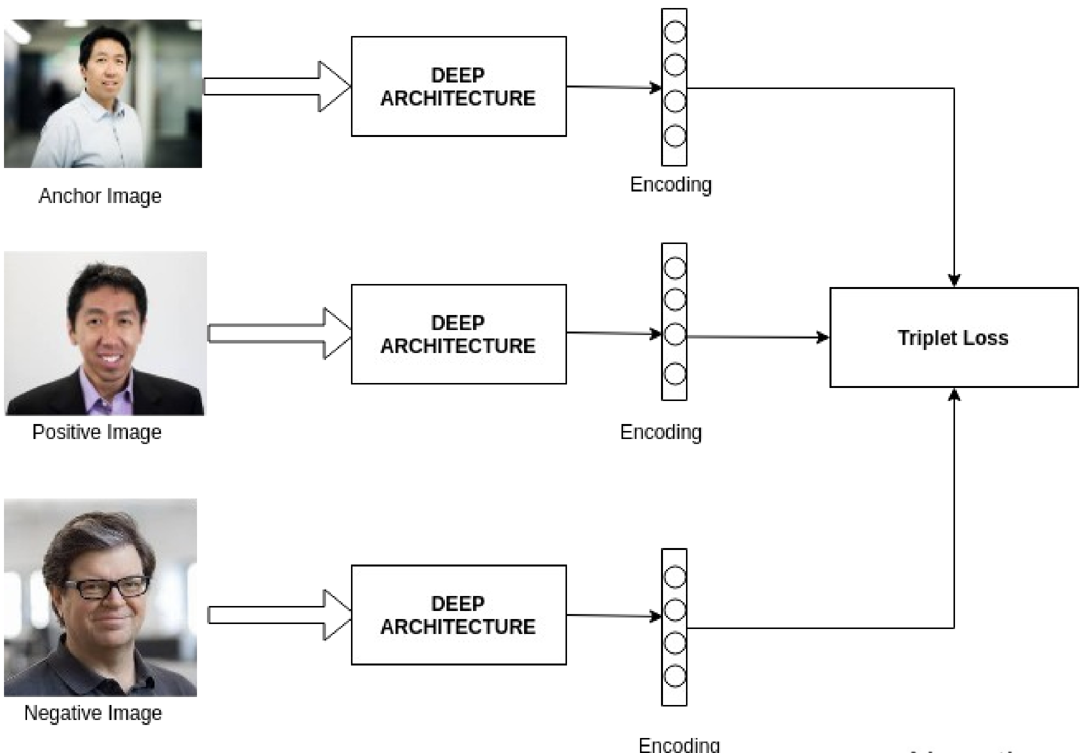
https://www.pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/

```
imgA = Input(shape=config.IMG_SHAPE)
imgB = Input(shape=config.IMG_SHAPE)
featureExtractor = build_siamese_model(config.IMG_SHAPE)
featsA = featureExtractor(imgA)
featsB = featureExtractor(imgB)
distance = Lambda(utils.euclidean_distance)([featsA, featsB])
outputs = Dense(1, activation="sigmoid")(distance)
model = Model(inputs=[imgA, imgB], outputs=outputs)

model.compile(loss="binary_crossentropy", optimizer="adam",
metrics=["accuracy"])
# train the model
print("[INFO] training model...")
history = model.fit(
[pairTrain[:, 0], pairTrain[:, 1]], labelTrain[:],
validation_data=([pairTest[:, 0], pairTest[:, 1]], labelTest[:]),
batch_size=config.BATCH_SIZE,
epochs=config.EPOCHS)
```

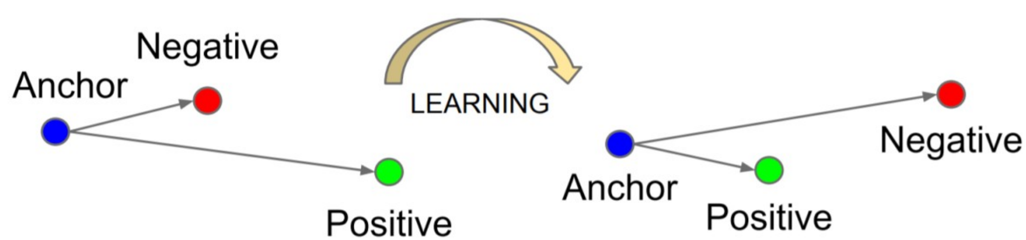
https://www.pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/

Triplet loss



TRIPLETS:

- **Anchor**
- **Positive**
- **Negative**



Triplet loss

$$||f(x_i^a) - f(x_i^p)||_2^2 + \alpha \leq ||f(x_i^a) - f(x_i^n)||_2^2$$

$$\mathcal{L}(A, P, N) = \max\left(\|\mathrm{f}(A) - \mathrm{f}(P)\|^2 - \|\mathrm{f}(A) - \mathrm{f}(N)\|^2 + \alpha, 0\right)$$

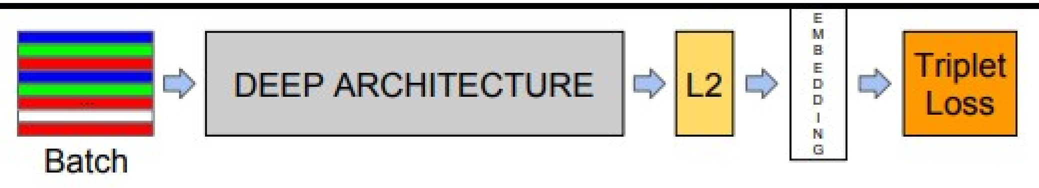


Figure 2. **Model structure.** Our network consists of a batch input layer and a deep CNN followed by L_2 normalization, which results in the face embedding. This is followed by the triplet loss during training.

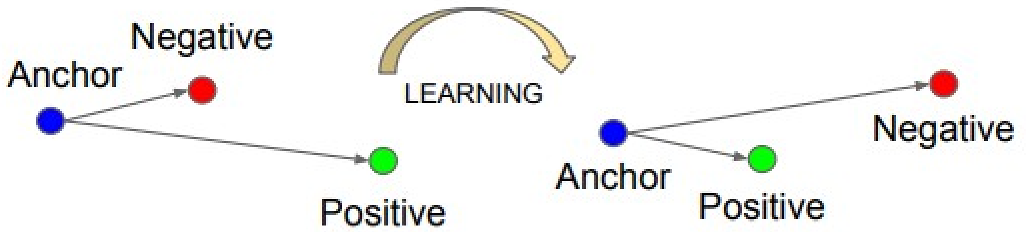
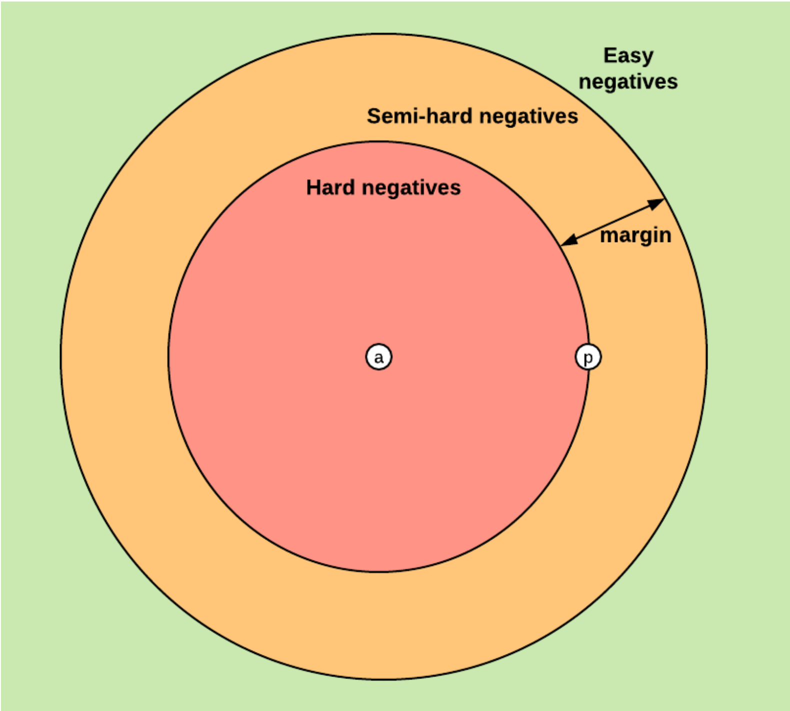


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

Triplet selection

- Offline triplet mining
- Online triplet mining
  - Batch all
  - Batch hard



https://omoindrot.github.io/triplet-loss

Online triplet mining

- batch of faces as input of size B=PK, composed of P different persons with K images each
  - **batch all**: select all the valid triplets, and average the loss on the hard and semi-hard triplets
    - this produces a total of PK(K−1)(PK−K) triplets (PK anchors, K−1 possible positives per anchor, PK−K possible negatives)
  - **batch hard**: for each anchor, select the hardest positive (biggest distance d(a,p)) and the hardest negative among the batch
    - this produces PK triplets

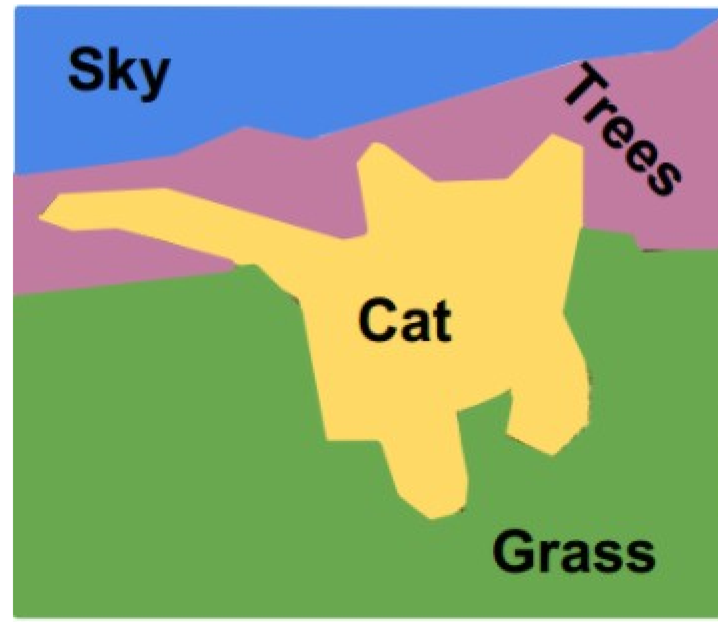
Training neural nets

- http://karpathy.github.io/2019/04/25/recipe/
- https://www.youtube.com/watch?v=NUmbgp1h64E
- https://www.youtube.com/watch?v=SjQyLhQIXSM&list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc&index=2
- https://www.youtube.com/watch?v=C1N_PDHuJ6Q&list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc&index=3

Computer vision tasks

Computer vision tasks

Classification
What object is in this image?
CAT



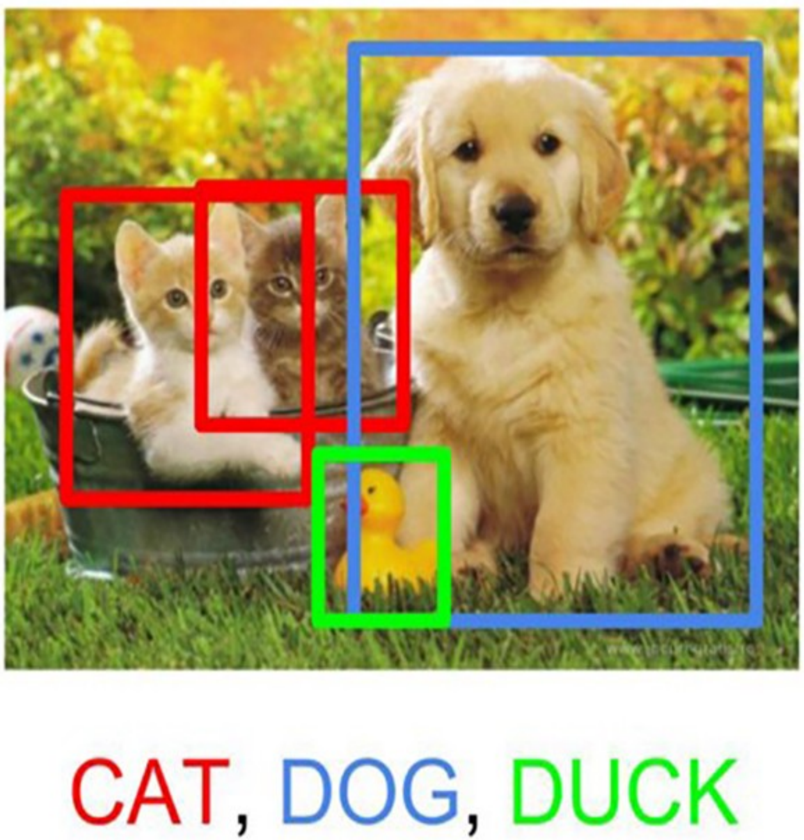
Semantic segmentation
What label has each pixel?
Pixel level, we are not interested in objects

https://tariq-hasan.github.io/concepts/computer-vision-semantic-segmentation/

Computer vision tasks

Object localization
What object is in this image and
where is it located?
CAT

Computer vision tasks



Object detection: multiple objects (the label and position - bounding box - of each object)

Instance segmentation: multiple objects (the label and position of each object)

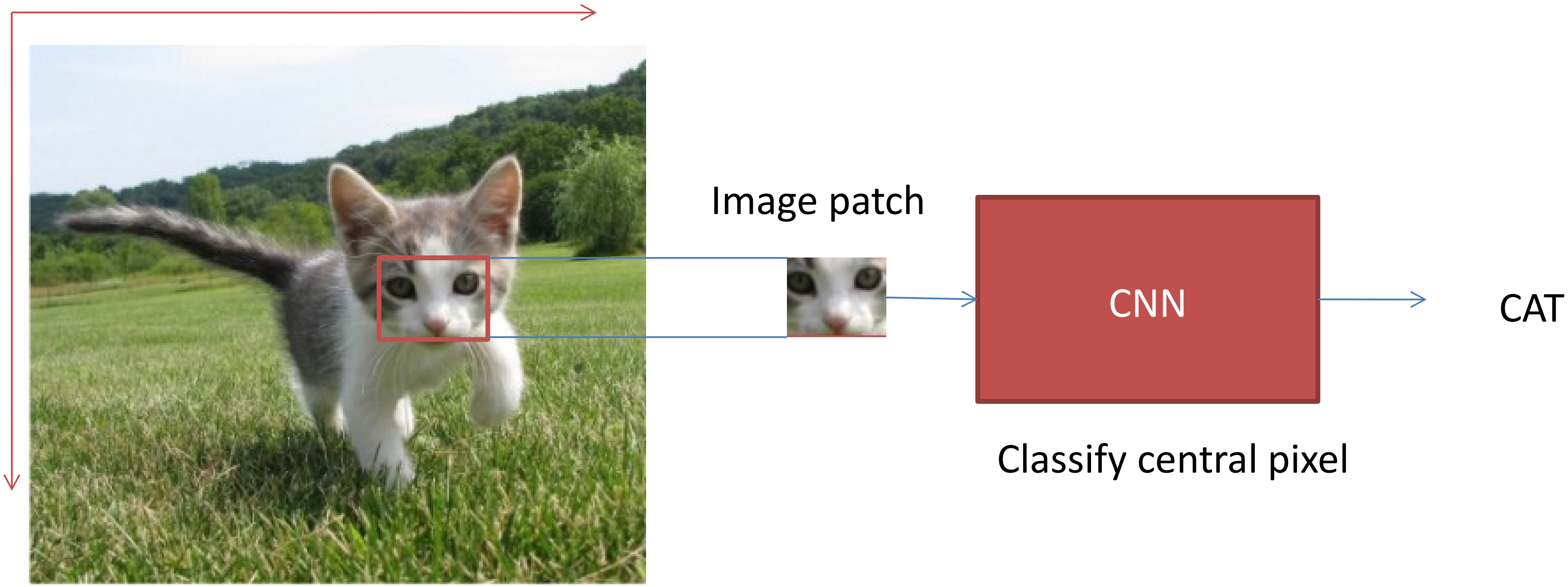
Image source: https://static.artfido.com/2018/05/cover-12.jpg

Semantic segmentation

- Understating the image at **pixel level:** each pixel is labelled individually with a class
- Groups both semantics and location
  - Global information: WHAT?
  - Local information: WHERE?

Semantic segmentation

Naïve approach – sliding window

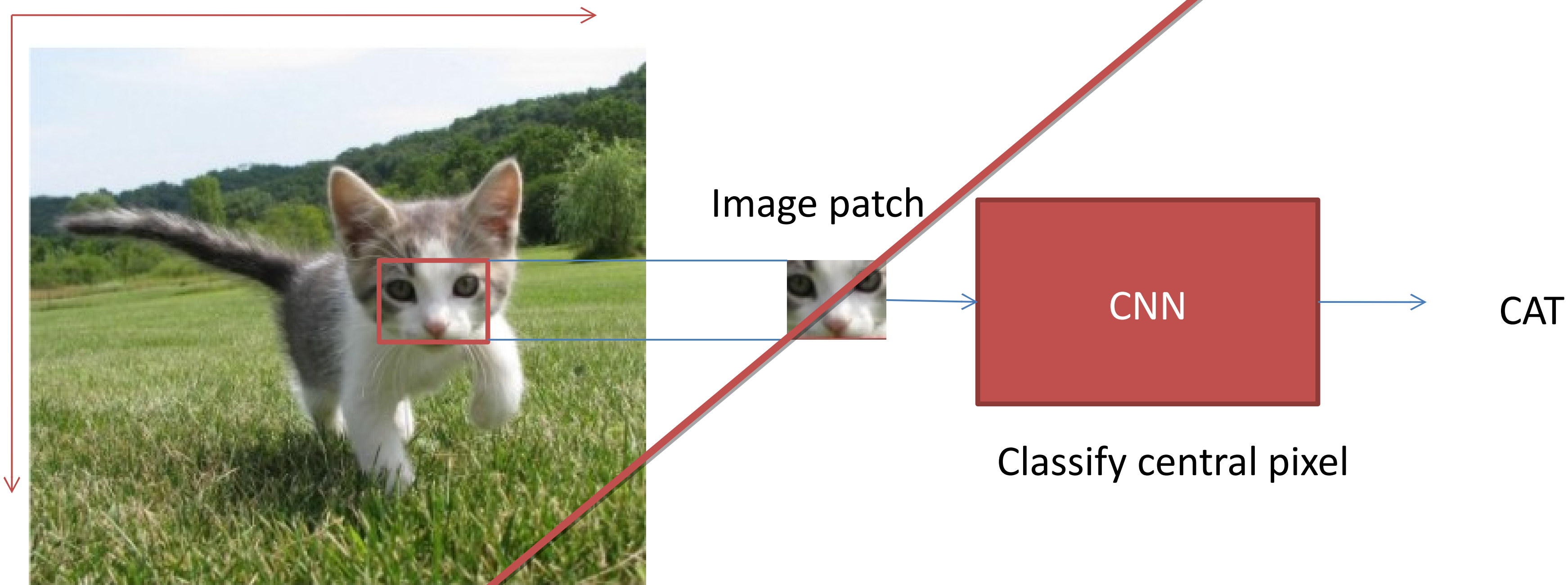


Slide window over the input image and classify each image patch using a CNN

https://papers.nips.cc/paper/2012/file/459a4ddcb586f24efd9395aa7662bc7c-Paper.pdf

Semantic segmentation

Naïve approach – sliding window



Slide window over the input image and classify each image patch using a CNN

Highly inefficient!!

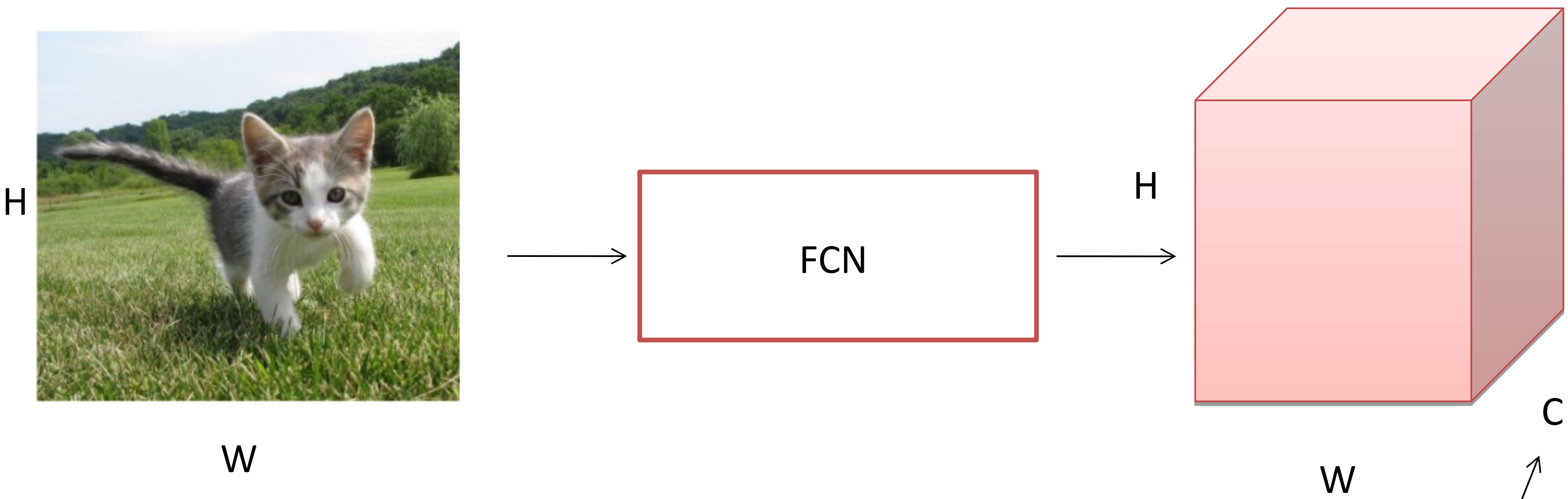
Let's design a CNN to classify all pixels at once!

- How would we encode the output (we need to output a class label for each pixel)?
- What would be the input and the output size of this network?

Fully convolutional neural networks

Fully convolutional neural networks

The output should have the same spatial size as the input



Output depth:
Number of classes

Input

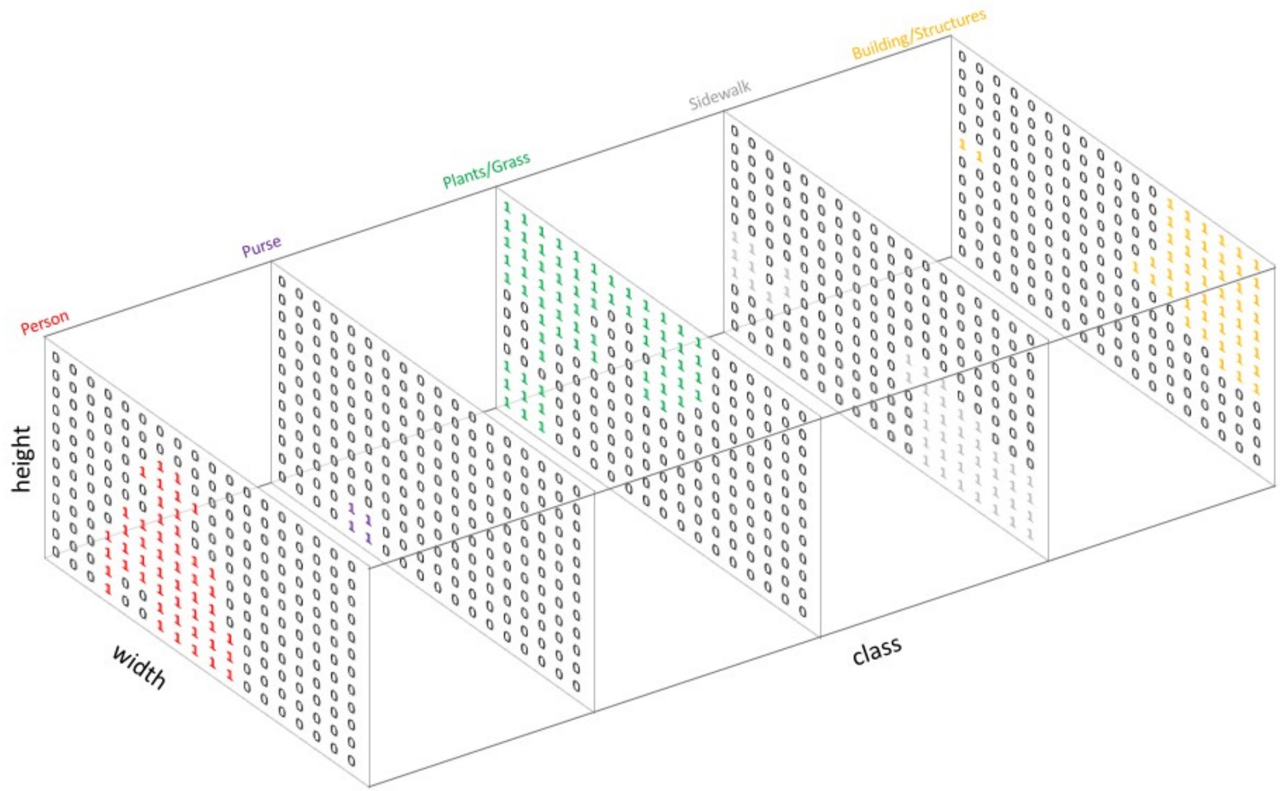
segmented

1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4

Semantic Labels

https://www.jeremyjordan.me/semantic-segmentation/#loss



Person
Purse
Plants/Grass
Sidewalk
Building/Structures
height
width
class

Output layer

Apply cross entropy pixel-wise on the last layer of the network

2	1
1	0

Ground truth label image

Class 0

0	0
0	1

Class 1

0	1
1	0

Class 2

1	0
0	0

3 classes (0, 1, 2)
One hot encodings
2 – [0, 0, 1]
1 – [0, 1, 0]
0 – [1, 0, 0]

Fully convolutional neural networks (FCN)

- How to preserve the spatial size of the output map (input size must be equal to the output size)?

Fully convolutional neural networks

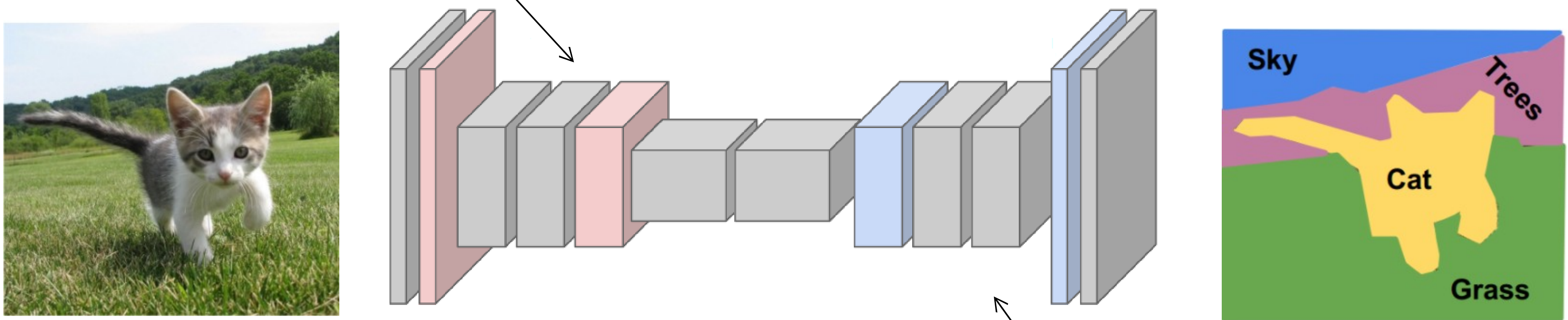
- How to preserve the spatial size of the output map (input size must be equal to the output size)?
  - Don't use any layers that reduce the spatial size

Fully convolutional neural networks

- How to preserve the spatial size of the output map (input size must be equal to the output size)?
  - ~~Don't use any layers that reduce the spatial size -> inefficient~~
  - Use two paths in the network: a down-sampling and an up-sampling path

Fully convolutional neural networks

Down-sampling: strided
convolutions, pooling layers



Up-sampling: NN, bilinear
interpolation, max unpooling,
transposed convolution

Up-sampling techniques

Up-sampling

Nearest neighbour

4	5
10	20

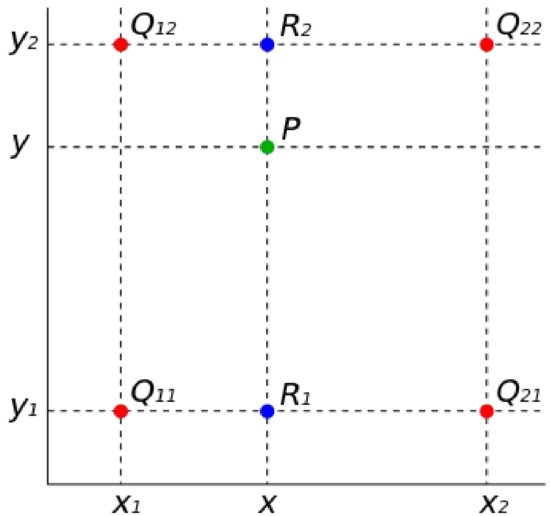
2x2

4	4	5	5
4	4	5	5
10	10	20	20
10	10	20	20

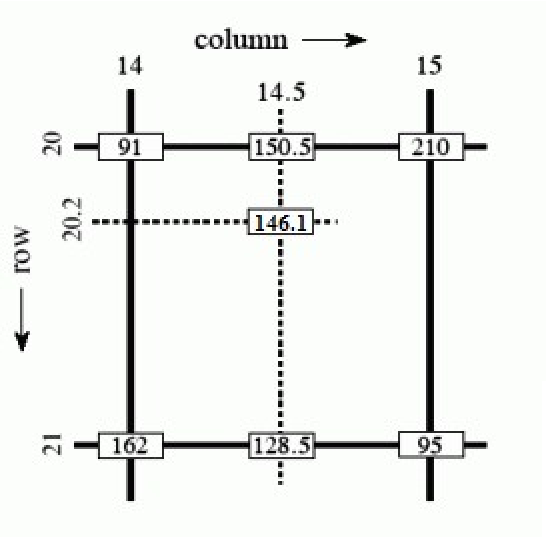
4x4

Up-sampling

Bilinear interpolation



$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}),$$
$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}).$$



$$I_{20,14.5} = \frac{15 - 14.5}{15 - 14} \cdot 91 + \frac{14.5 - 14}{15 - 14} \cdot 210 = 150.5,$$
$$I_{21,14.5} = \frac{15 - 14.5}{15 - 14} \cdot 162 + \frac{14.5 - 14}{15 - 14} \cdot 95 = 128.5,$$

$$I_{20.2,14.5} = \frac{21 - 20.2}{21 - 20} \cdot 150.5 + \frac{20.2 - 20}{21 - 20} \cdot 128.5 = 146.1.$$

Up-sampling

Bilinear interpolation

10	20
30	40

2x2

10	12.5	17.5	**20**
15	17.5	22.5	25
25	27.5	32.5	35
30	32.5	37.5	**40**

4x4

Up-sampling in keras

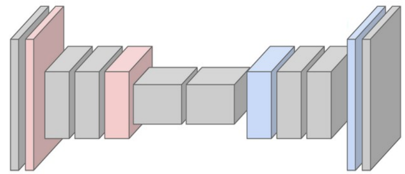
`UpSampling2D` class

```
tf.keras.layers.UpSampling2D(
    size=(2, 2), data_format=None, interpolation="nearest", **kwargs
)
```

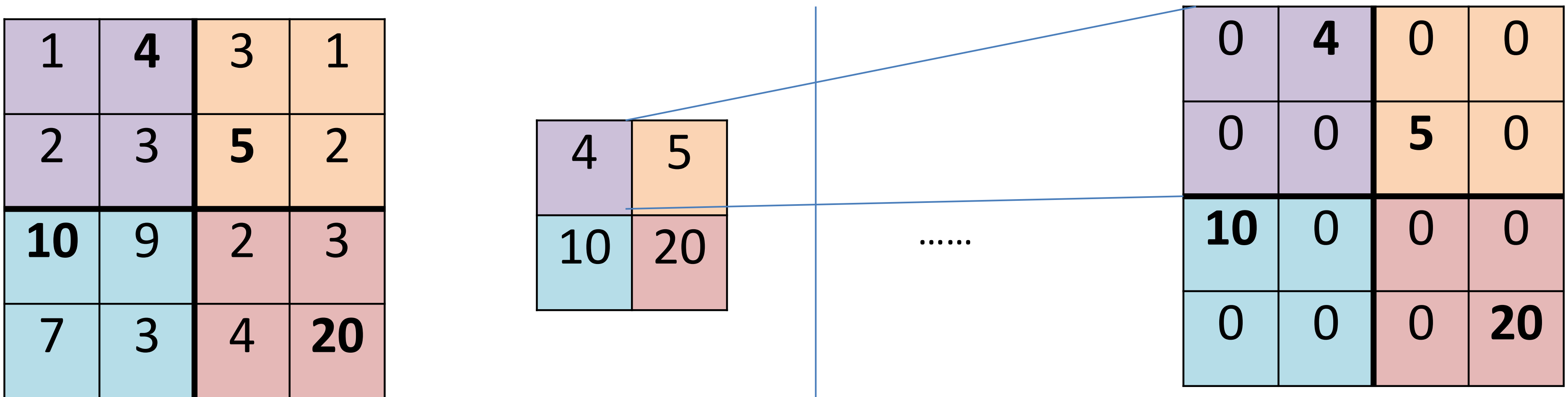
Upsampling layer for 2D inputs.

Interpolation:
- nearest
- bilinear

Max un-pooling



Symmetric structure of the FCN



Pooling: remember the position of the maximum element within the receptive field of the layer

Max Un-Pooling: use the position of the max

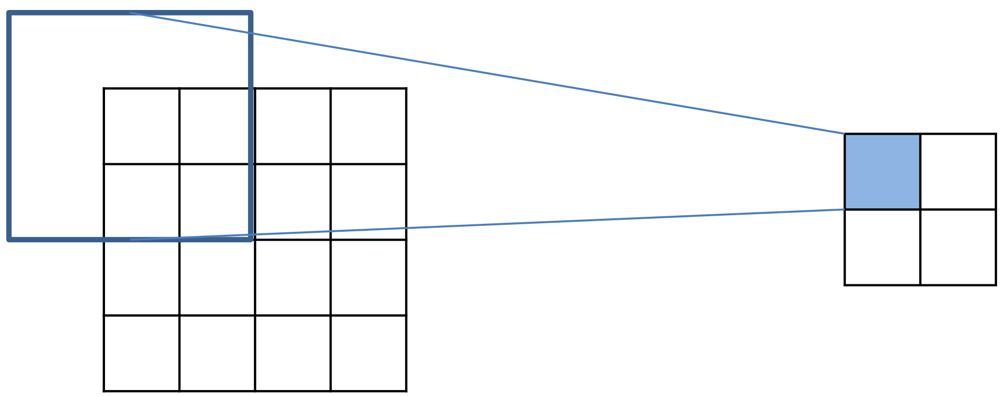
DOWNSAMPLING PATH

UP-SAMPLING PATH

Transposed convolutions

Learnable up-sampling

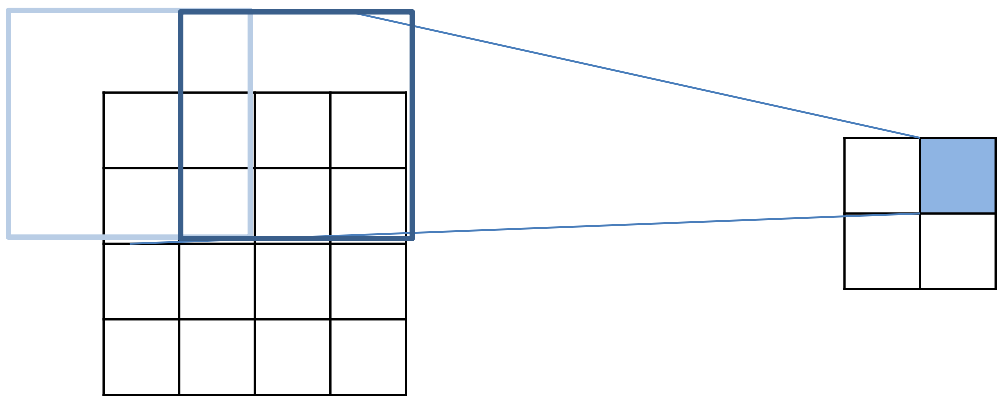
Remember *strided* convolutions



3x3, stride 2, pad 1

Transposed convolutions
Learnable up-sampling

Remember *strided* convolutions

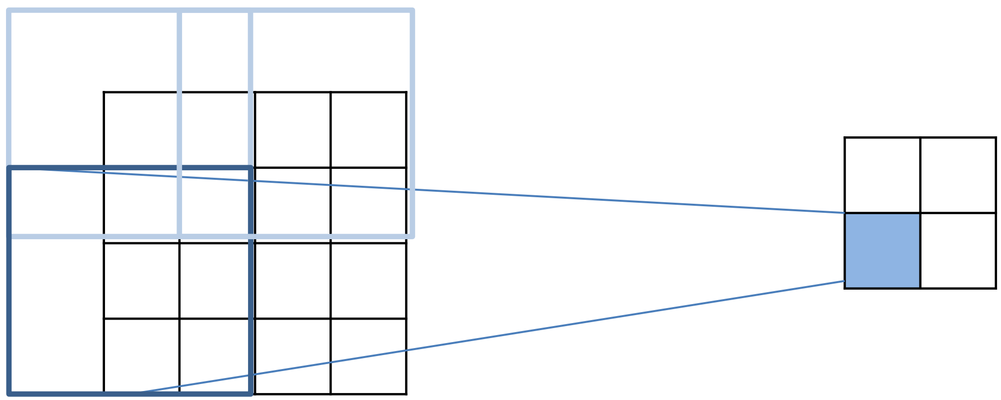


3x3, stride 2, pad 1

Transposed convolutions

Learnable up-sampling

Remember *strided* convolutions

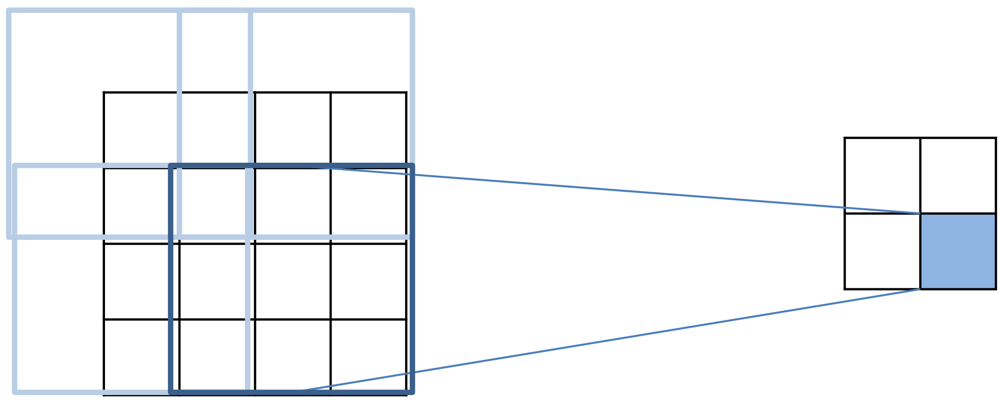


3x3, stride 2, pad 1

Transposed convolutions

Learnable up-sampling

Remember *strided* convolutions

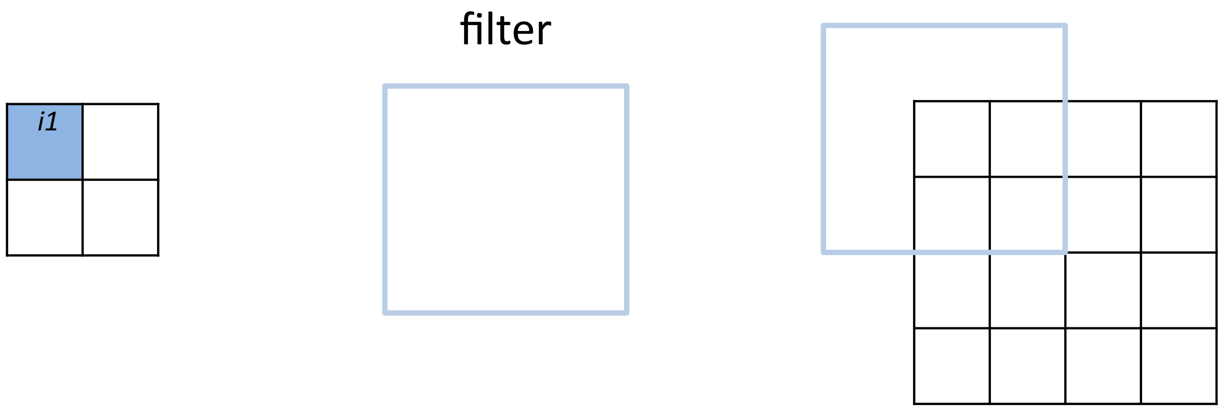


3x3, stride 2, pad 1

Transposed convolutions

Learnable up-sampling

NOW: transposed convolutions



The input feature *i1* is multiplied with the filter values

i1 is like a weight for the filter value.

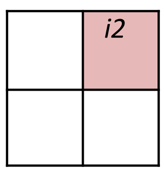
Copy the “weighted” filter in the corresponding output

3x3, stride 2, pad 1

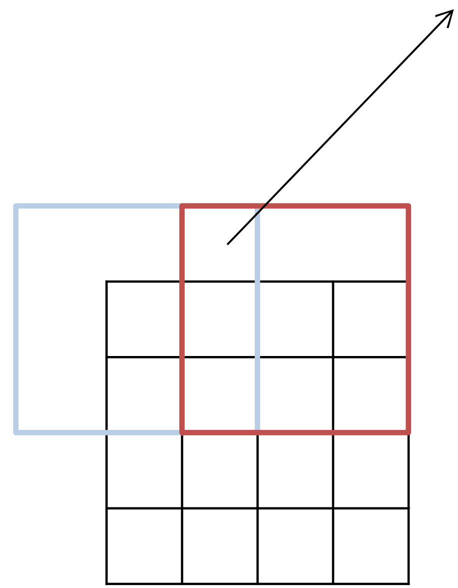
Transposed convolutions

Learnable up-sampling

NOW: transposed convolutions



filter



Where there is overlap, sum up the values

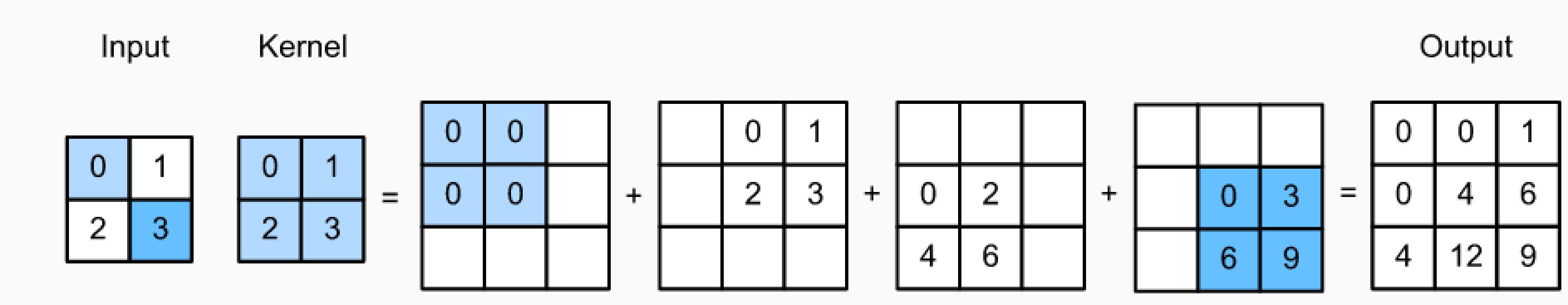
The input feature *i2* is multiplied with the filter values
i2 is like a weight for the filter value.
Copy the “weighted” filter in the corresponding output

3x3, stride 2, pad 1

Transposed convolutions

Learnable up-sampling

The weights of the filter are learned
At each position multiply the filter values with the corresponding position of the input layer to get the result of the output
Add overlapping positions



Example: 1 stride and 0 padding

Transposed convolutions

Learnable up-sampling

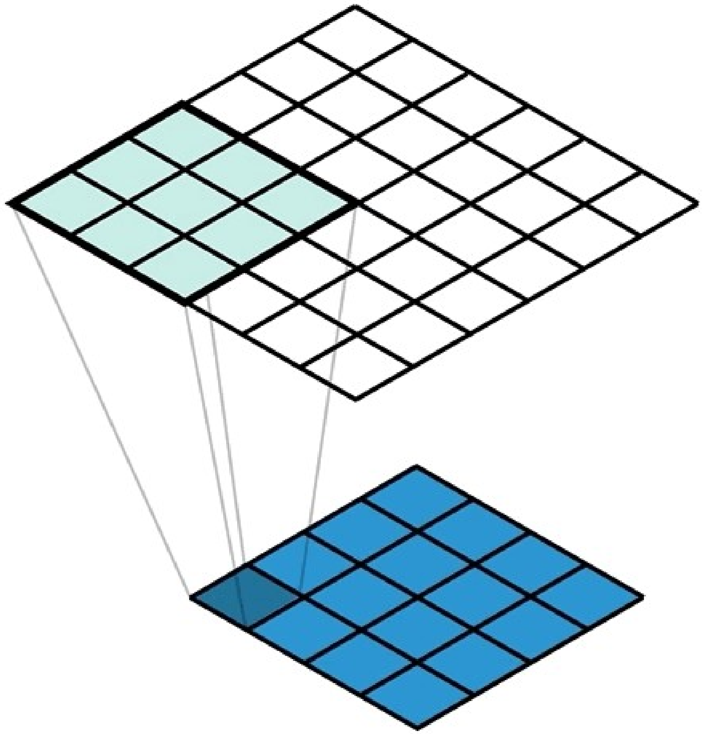


Image source: https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

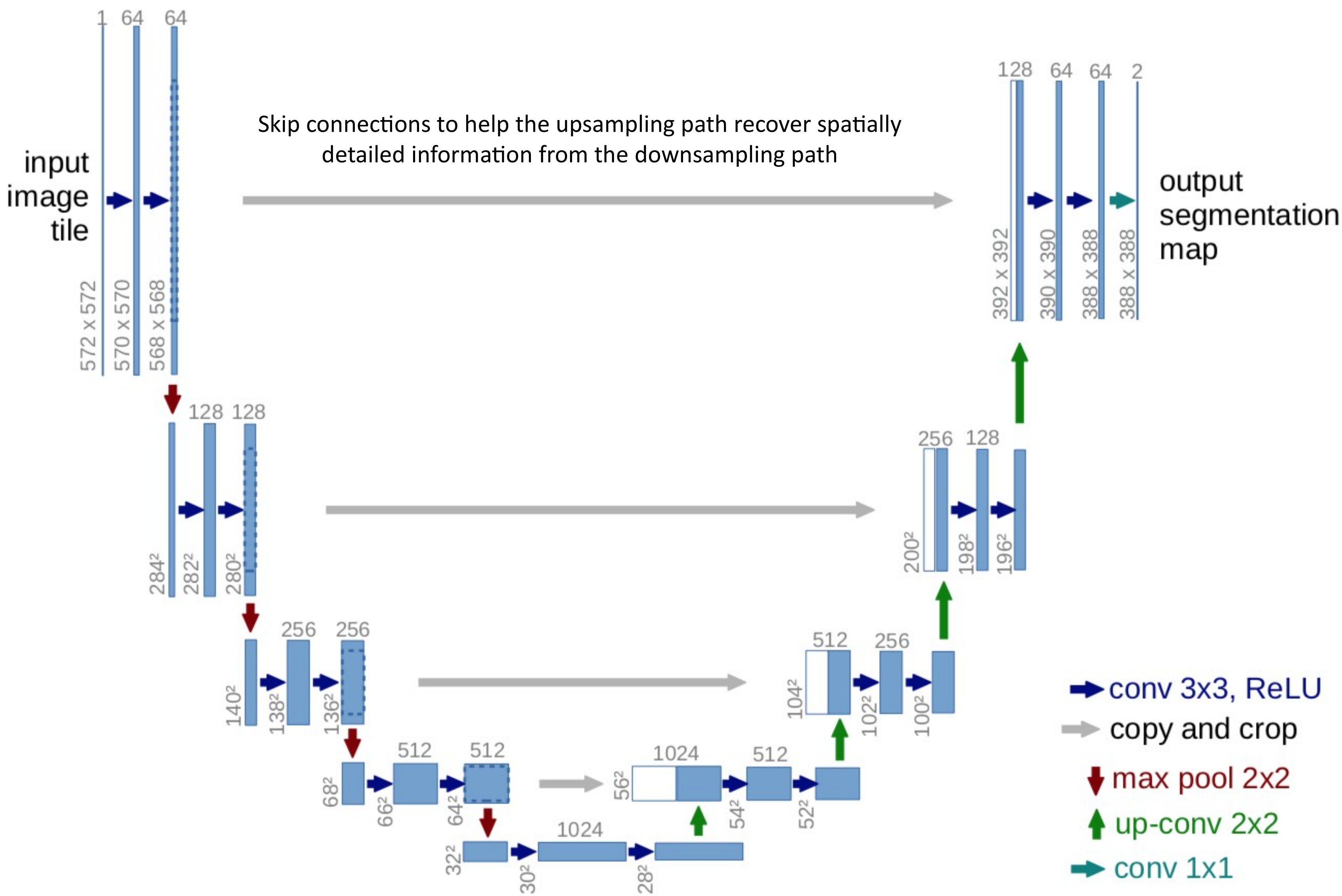
Transposed convolutions in keras

Conv2DTranspose class

```
tf.keras.layers.Conv2DTranspose(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    output_padding=None,
    data_format=None,
    dilation_rate=(1, 1),
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

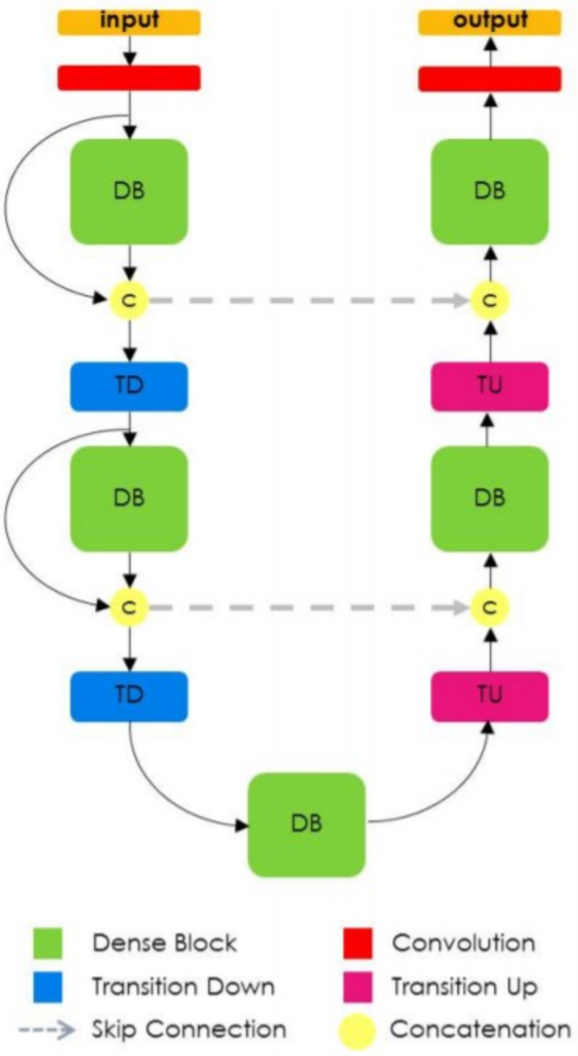
Transposed convolution layer (sometimes called Deconvolution).

U-Net, 2015


Skip connections to help the upsampling path recover spatially detailed information from the downsampling path
input image tile
output segmentation map
1
64
64
572 x 572
570 x 570
568 x 568
128
128
256
256
512
512
1024
1024
512
512
256
256
128
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

100 layers tiramisu, 2017

- Similar to U-Net
- uses Dense Block for convolutions and transposed convolutions



The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation, 2016