Convolutional Neural Networks
&
Improving Deep Neural Networks

Project Proposal

Topic | Convolutional Neural Networks & Improving Deep Neural Networks

Description | Build a CNN and apply it to detection and recognition tasks, use neural style transfer to generate art, and apply algorithms to image and video data

Train test sets, analyze variance for DL applications, use standard techniques and optimization algorithms, and build neural networks in TensorFlow

Expected Duration | 8 weeks

Team Member | 정재윤, 조성근, 최정우

Table of Contents

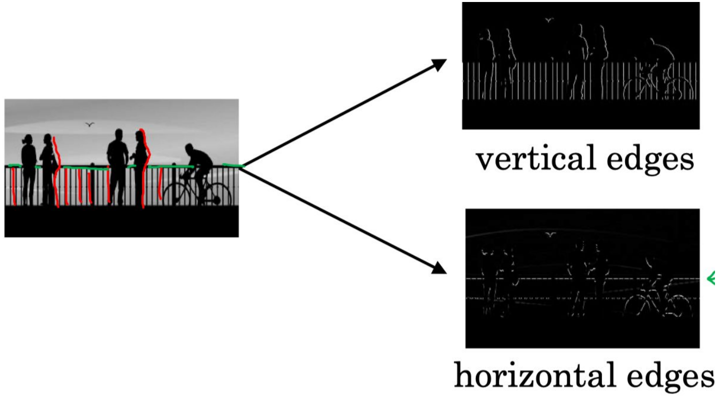
Computer Vision

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs.

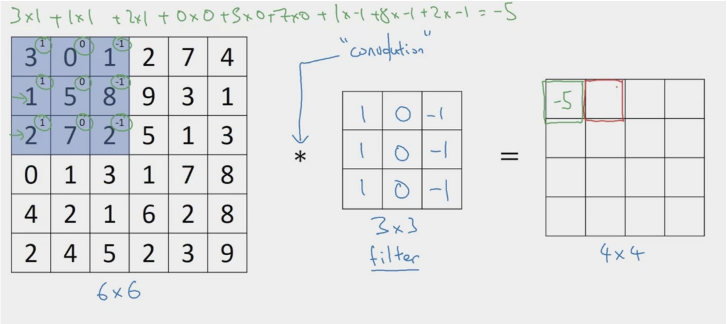
Image detection:

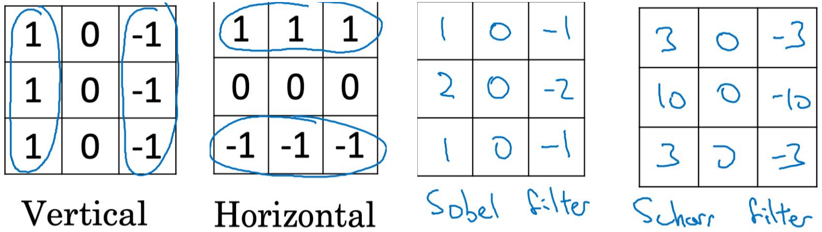
Cat? (0/1)

Edge Detection:

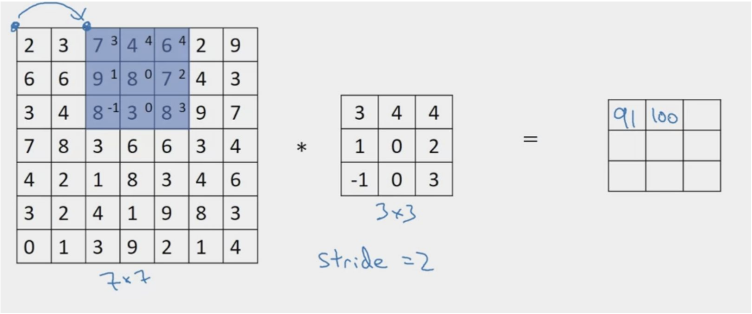


Edge Detection

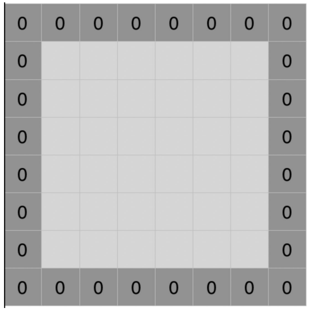




Stride:

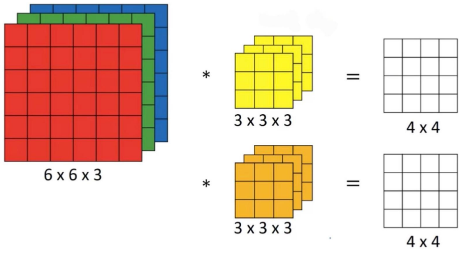


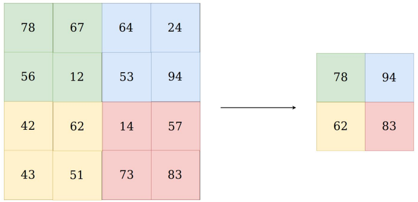
Padding:

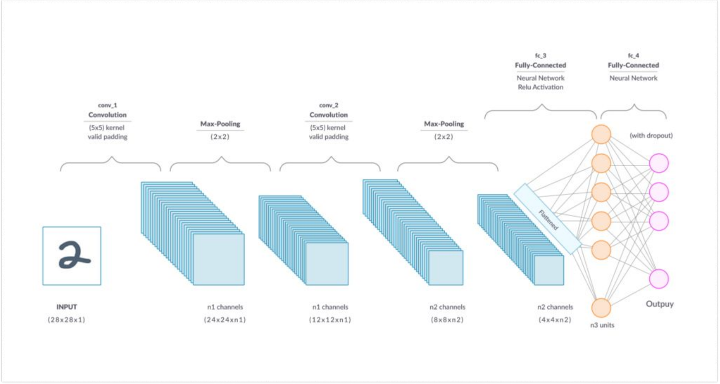


Types of Layers in CNN

- There are three layers in CNN: convolution, pooling, and fully-connected
- **Convolution**: gets an input of a matrix applies filters to produce the output
- **Pooling**: either takes the max or average of a section and reduces the parameters (Max Pooling is more common)
- **Fully Connected**: input is the 3 dimensional matrix flattened and fed into the fully connected layers







ResNet: Residual Network

Problems with Classic Networks

- Vanishing gradient

Residual Block

- Shortcut / Skip Connection

Deep Layers

- Improved performance

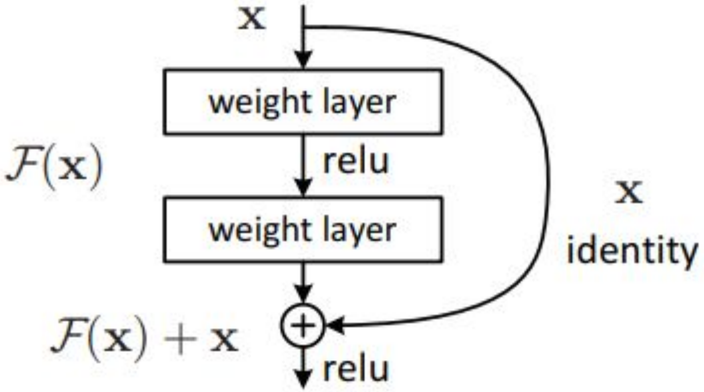
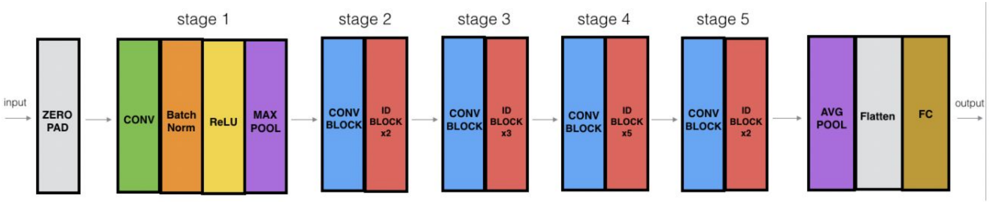


Figure 2. Residual learning: a building block.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Demo:

```python
# UNQ_C3
# GRADED FUNCTION: ResNet50

def ResNet50(input_shape = (64, 64, 3), classes = 6):
    """
    Stage-wise implementation of the architecture of the popular ResNet50:
    CONV2D -> BATCHNORM -> RELU -> MAXPOOL -> CONVBLOCK -> IDBLOCK*2 -> CONVBLOCK -> IDBLOCK*3
    -> CONVBLOCK -> IDBLOCK*5 -> CONVBLOCK -> IDBLOCK*2 -> AVGPOOL -> FLATTEN -> DENSE

    Arguments:
    input_shape -- shape of the images of the dataset
    classes -- integer, number of classes

    Returns:
    model -- a Model() instance in Keras
    """

    # Define the input as a tensor with shape input_shape
    X_input = Input(input_shape)


    # Zero-Padding
    X = ZeroPadding2D((3, 3))(X_input)

    # Stage 1
    X = Conv2D(64, (7, 7), strides = (2, 2), kernel_initializer = glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3)(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    # Stage 2
    X = convolutional_block(X, f = 3, filters = [64, 64, 256], s = 1)
    X = identity_block(X, 3, [64, 64, 256])
    X = identity_block(X, 3, [64, 64, 256])

    ### START CODE HERE

    ## Stage 3 (≈4 lines)
    X = convolutional_block(X, f = 3, filters = [128,128,512], s = 2)
    X = identity_block(X, 3,  [128,128,512])
    X = identity block(X, 3,  [128,128,512])
    X = identity_block(X, 3,  [128,128,512])

    ## Stage 4 (≈6 lines)
    X = convolutional_block(X, f = 3, filters = [256, 256, 1024], s = 2)
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])
    X = identity_block(X, 3, [256, 256, 1024])

    ## Stage 5 (≈3 lines)
    X = convolutional_block(X, f = 3, filters = [512, 512, 2048], s = 2)
    X = identity_block(X, 3, [512, 512, 2048])
    X = identity_block(X, 3, [512, 512, 2048])

    ## AVGPOOL (≈1 line). Use "X = AveragePooling2D(...)(X)"
    X = AveragePooling2D((2, 2))(X)

    ### END CODE HERE

    # output layer
    X = Flatten()(X)
    X = Dense(classes, activation='softmax', kernel_initializer = glorot_uniform(seed=0))(X)


    # Create model
    model = Model(inputs = X_input, outputs = X)

    return model
```



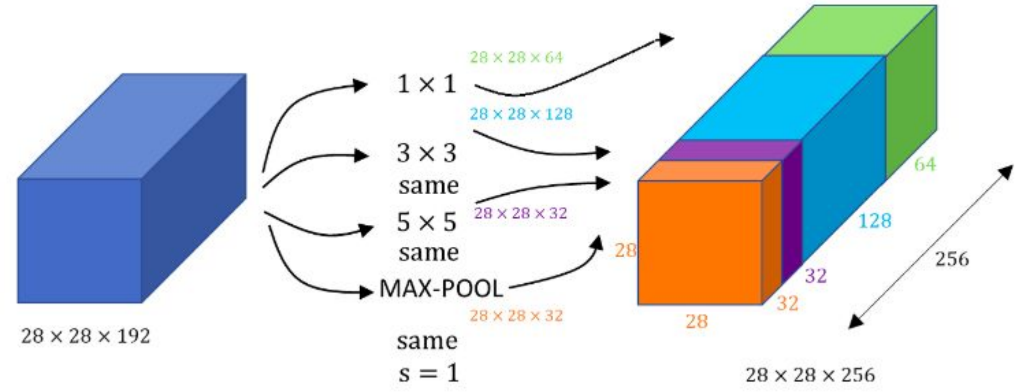
- Zero-padding pads the input with pad of (3,3)
- 2D convolution has 64 filters of shape Shape (7,7) and used stride of (2,2)
- Max Pooling of (3,3)
- Process of Convolution block and ID blocks
- 2D average pooling uses window of shape(2,2)
- Fully connected (Dense) layer reduces its input to the number of classes using a softmax activation.

Inception Network

"Bottleneck layer"

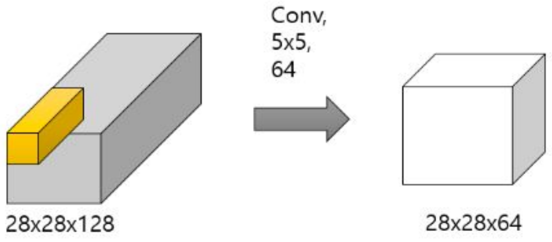
Problems with existing models
- Too many parameters

1x1 Convolution Network
- Maintain input size
- Reduce computational cost

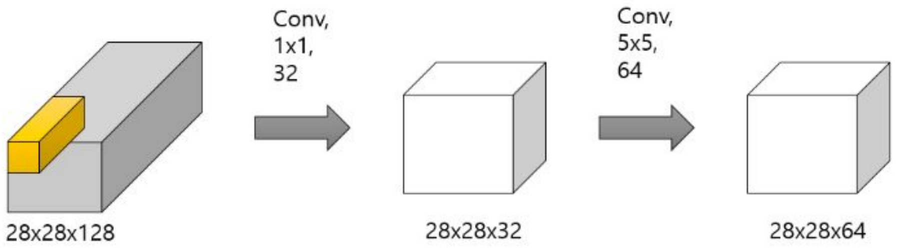


Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

Inception Network



#params = 28x28x64x5x5x128 = 160M



#params = 28x28x32x128x1x1 = 4.8M

#params = 28x28x64x5x5x32 = 40M

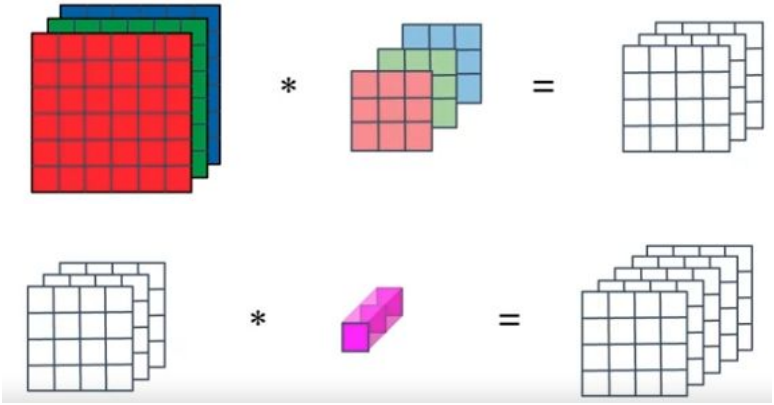
#total = 44.8M

MobileNet

- Computational cost can be too high for devices with smaller GPU or for mobile devices

- MobileNet uses depthwise and pointwise convolutions to reduce the number of computations

- Mobile v2 uses the bottleneck method to enlarge the representation which allows the neural network to learn richer functions



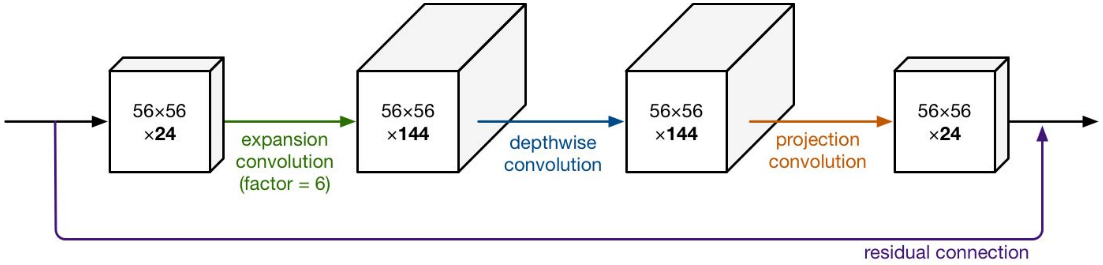


Table of Contents

Basic Concepts of Object Localization

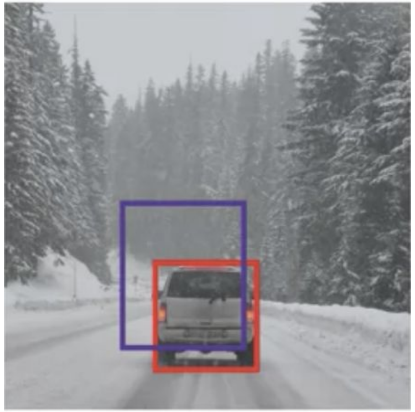
- To classify an image and localize it, the image is put through a ConvNet and has outputs of y with labels: p_c, b_x, b_y, b_h, b_w, and class label
- The image is split into boxes to detect objects in each box (called bounding boxes)
- Each pixel or box is given these labels to determine what type of object exists, and where it is located

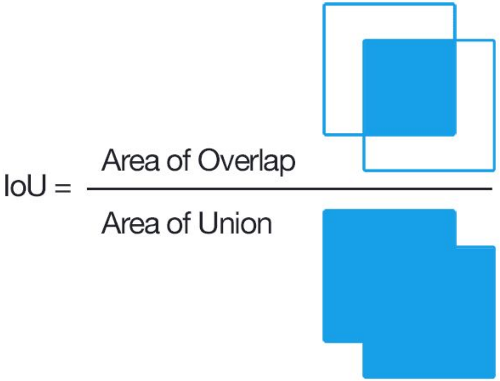
Intersection Over Union

- In two bounding boxes, IoU determines the size of intersection, and divides by size of the union

- Generally is determined “correct” if IoU is greater or equal to 0.5

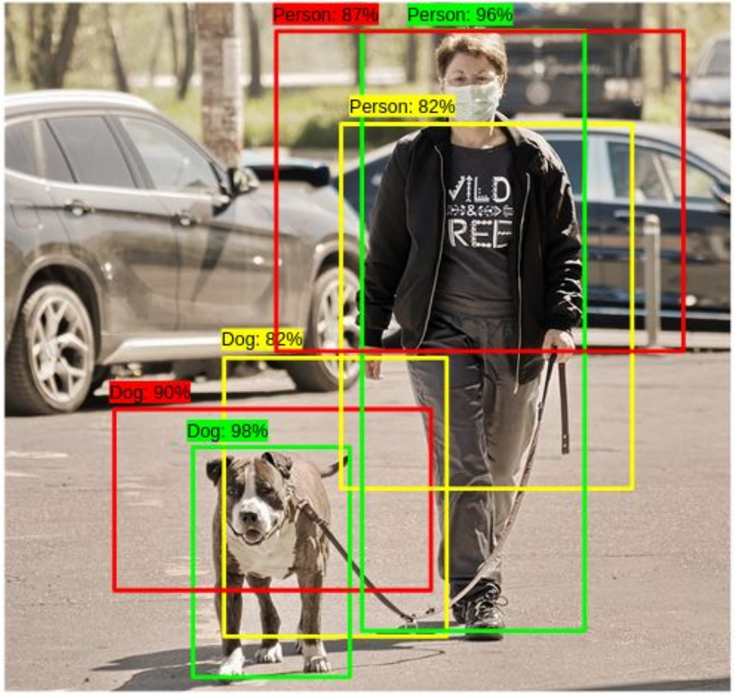
- It is also used to check how similar two boxes are to each other



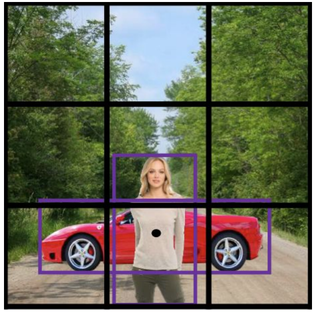


Non-max Suppression

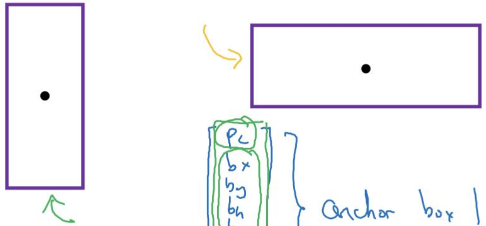
- Determines which bounding box is the most "accurate"
- Calculates the IoU for each bounding box, and then chooses the one with the highest value
- General Algorithm is:
  - Discard all boxes that has an IoU less than 0.6
  - Choose the box with the highest IoU
  - Discard remaining boxes that have IoU greater than 0.5



Anchor Boxes



Anchor box 1: Anchor box 2:



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

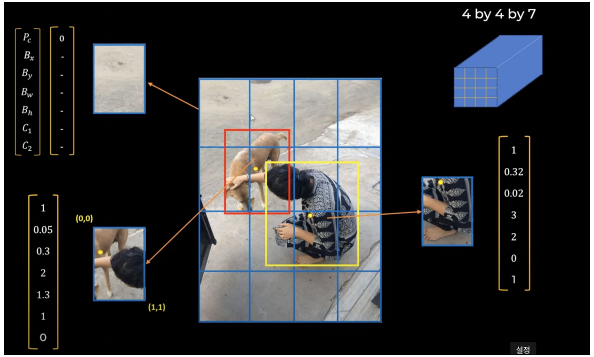
$$y = \begin{bmatrix} \left.\begin{matrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{matrix}\right\} \text{anchor box 1} \\ \left.\begin{matrix} p_c \\ b_x \\ \vdots \\ c_3 \end{matrix}\right\} \text{anchor box 2} \end{bmatrix}$$

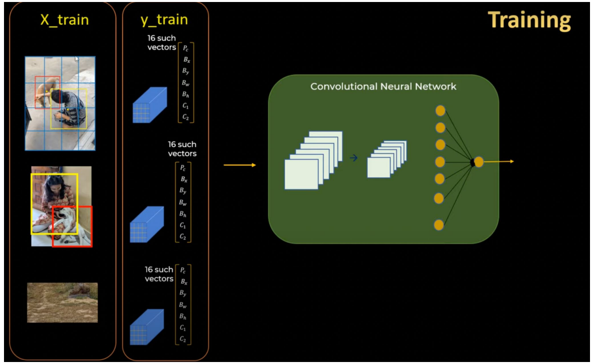
With two anchor boxes:

- Each object in training image is assigned to grid cell with object's midpoint & anchor box for the grid cell with highest IoU (highest overlap/non-overlap)

Either choose by hand for how many anchor boxes or use K-means clustering to figure out unique shapes

YOLO





Non-max Suppression:

Pros:

- Fast-paced object detection

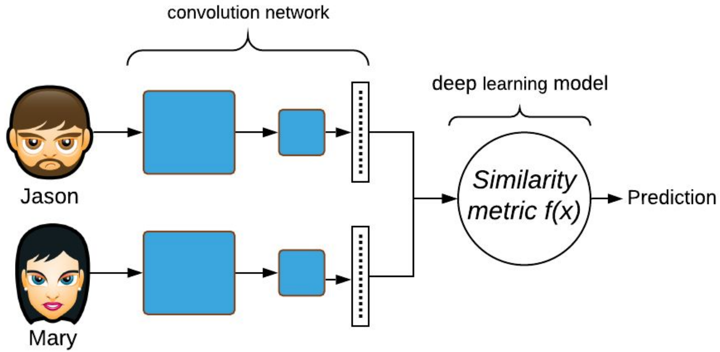
Cons:

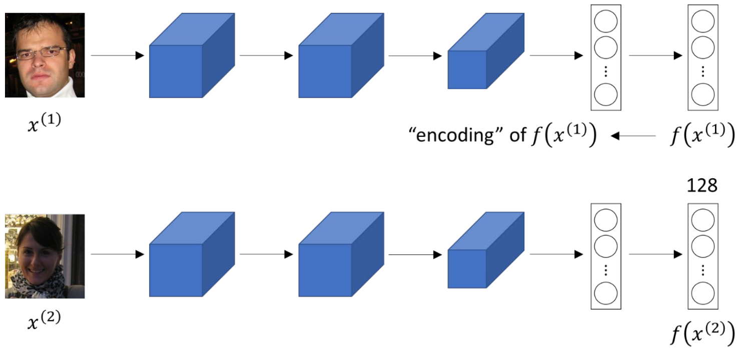
- Struggles to detect small objects
- Hard to segregate objects in groups

One Shot Learning & Siamese Network

- Similarity Function
  - Use the function d(img1, img2) to determine the degree of difference
  - Compare d to τ to check if the input image matches the image in the database

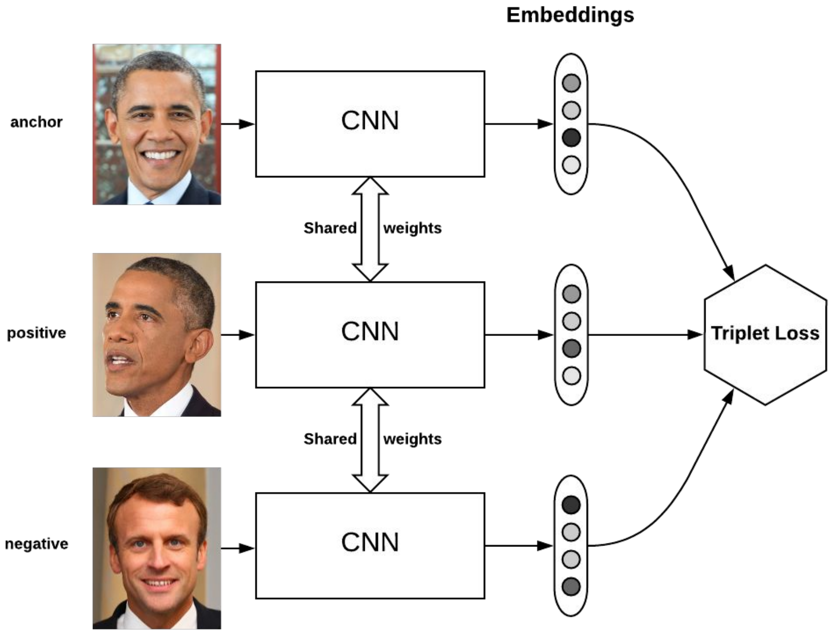
- Siamese Network
  - Use the vector computed by the CNN and set it as an encoding of the image
  - The Siamese network compared these "encodings" of two images to determine if they are the same person or not (uses the distance function)





Triplet Loss

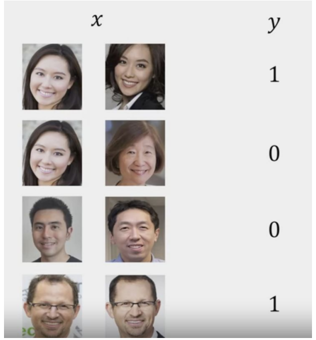
- 3 images per data: Anchor, Positive, Negative
- Maximize the difference between:
  - Anchor & Positive, and
  - Anchor & Negative
- Choose “hard-to-train” Negative
  - $d(\mathrm{A}, \mathrm{P}) + \alpha \leq d(\mathrm{A}, \mathrm{N})$
  - $d(\mathrm{A}, \mathrm{P}) \approx d(\mathrm{A}, \mathrm{N})$

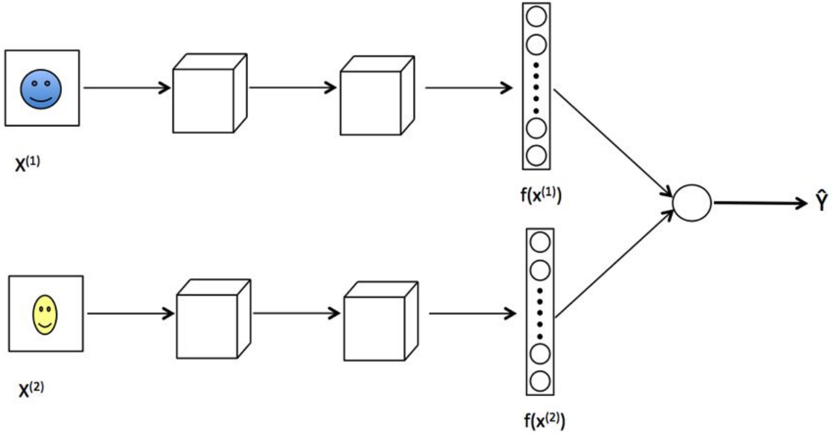


$$Loss = \sum_{i=1}^{N} \left[\|f_i^a - f_i^p\|_2^2 - \|f_i^a - f_i^n\|_2^2 + \alpha \right]_+$$

Binary Classification

- Compare 128D vector of two images
- Compute y_hat to predict the input face
  - Output as either 0(False) or 1(True)
- Can be used for Face Verification

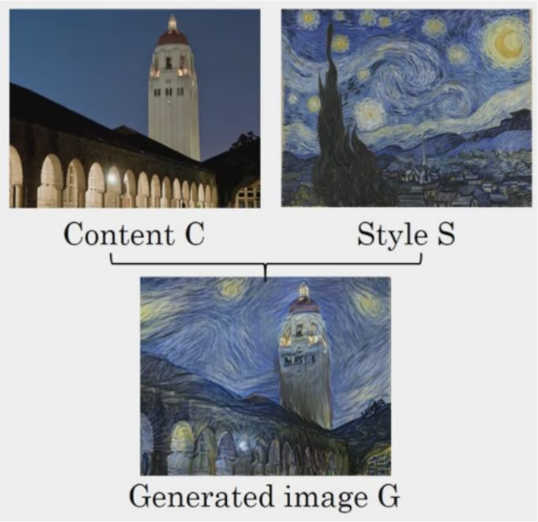




$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \left| f(x^{(i)})_k - f(x^{(j)})_k \right| + b\right)$$

Neural Style Transfer

Cost Function:

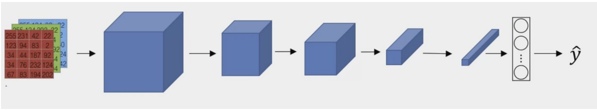


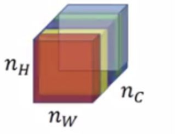
$$\overbrace{J_{total}(G)}^{\text{total cost function}} = \alpha \times \underbrace{J_{content}(C,G)}_{\text{similarity of content image to generated image}} + \beta \times \overbrace{J_{style}(S,G)}^{\text{similarity of style image to generated image}}$$

Content Cost Function:

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F^{l}_{ij} - P^{l}_{ij}\right)^2$$

Style Cost Function:





$$J^{[l]}_{style}(S,G) = \frac{1}{(2n^{[l]}_H n^{[l]}_W n^{[l]}_C)^2} \sum_k \sum_{k'} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'}\right)^2$$

```python
def triplet_loss(y_true, y_pred, alpha = 0.2):

    anchor, positive, negative = y_pred[0], y_pred[1], y_pred[2]

    # Step 1: Compute the (encoding) distance between the anchor and the positive
    pos_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,positive)),axis=-1)
    # Step 2: Compute the (encoding) distance between the anchor and the negative
    neg_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,negative)),axis=-1)
    # Step 3: subtract the two previous distances and add alpha.
    basic_loss = tf.add(tf.subtract(pos_dist,neg_dist),alpha)
    # Step 4: Take the maximum of basic_loss and 0.0. Sum over the training examples.
    loss = tf.reduce_sum(tf.maximum(basic_loss,0.0))

    return loss
```

```python
FRmodel.compile(optimizer = 'adam', loss = triplet_loss, metrics = ['accuracy'])
load_weights_from_FaceNet(FRmodel)
```

```python
database = {}
database["danielle"] = img_to_encoding("images/danielle.png", FRmodel)
database["younes"] = img_to_encoding("images/younes.jpg", FRmodel)
database["tian"] = img_to_encoding("images/tian.jpg", FRmodel)
database["andrew"] = img_to_encoding("images/andrew.jpg", FRmodel)
database["kian"] = img_to_encoding("images/kian.jpg", FRmodel)
database["dan"] = img_to_encoding("images/dan.jpg", FRmodel)
database["sebastiano"] = img_to_encoding("images/sebastiano.jpg", FRmodel)
database["bertrand"] = img_to_encoding("images/bertrand.jpg", FRmodel)
database["kevin"] = img_to_encoding("images/kevin.jpg", FRmodel)
database["felix"] = img_to_encoding("images/felix.jpg", FRmodel)
database["benoit"] = img_to_encoding("images/benoit.jpg", FRmodel)
database["arnaud"] = img_to_encoding("images/arnaud.jpg", FRmodel)
```

```python
def verify(image_path, identity, database, model):

    # Step 1: Compute the encoding for the image. Use img_to_encoding() see example above. (≈ 1 line)
    encoding = img_to_encoding(image_path, model)
    # Step 2: Compute distance with identity's image (≈ 1 line)
    # the L² norm of the difference between two vectors is equivalent to the Euclidean distance between the two points
    dist = np.linalg.norm(encoding - database[identity])
    # Step 3: Open the door if dist < 0.7, else don't open (≈ 3 lines)
    if dist<0.7:
        print("It's " + str(identity) + ", welcome home!")
        door_open = True
    else:
        print("It's not " + str(identity) + ", please go away")
        door_open = False

    return dist, door_open
```

```python
verify("images/camera_0.jpg", "younes", database, FRmodel)
```

```
It's younes, welcome home!
(0.65939283, True)
```

```python
verify("images/camera_2.jpg", "kian", database, FRmodel)
```

```
It's not kian, please go away
(0.86224014, False)
```