Agenda:

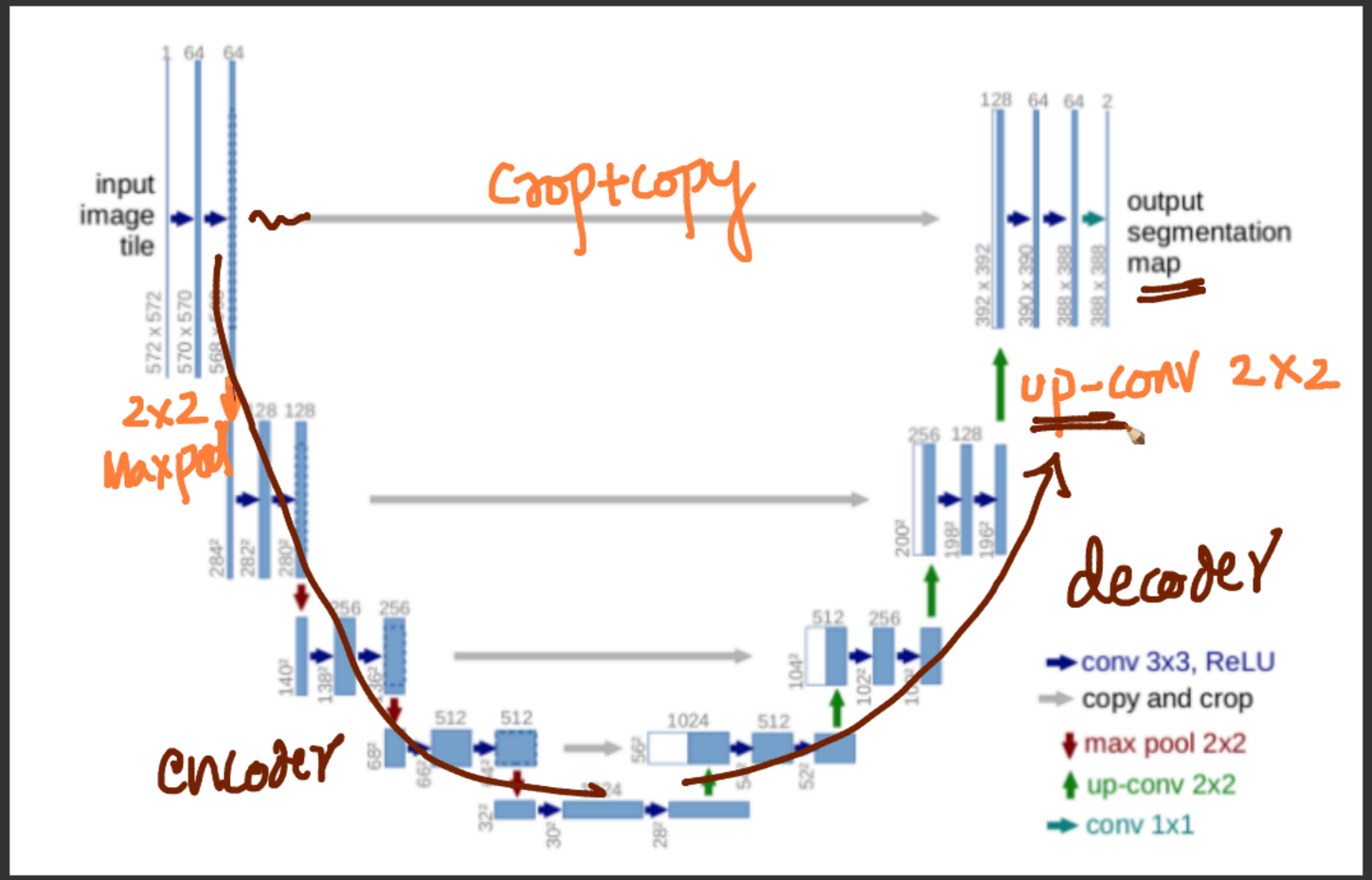
Segmentation → U-net

- Mask RCNN ✓
- DeepLap V3 (overview) + Dilated-conv

Few-Shot Learning (practical)

Signature Verification

- Siamese network ✓ + BCE / log-loss
- Contrastive Loss
- Triplet loss + variations



What can we notice ?

- Unet Model has a "U" shape architect

Though UNet performs relatively much better than FCN models, UNet is still not the SOTA for Image Segmentation tasks.

- UNet requires many layers which makes training time quite significant.
- Relatively high GPU memory for larger images

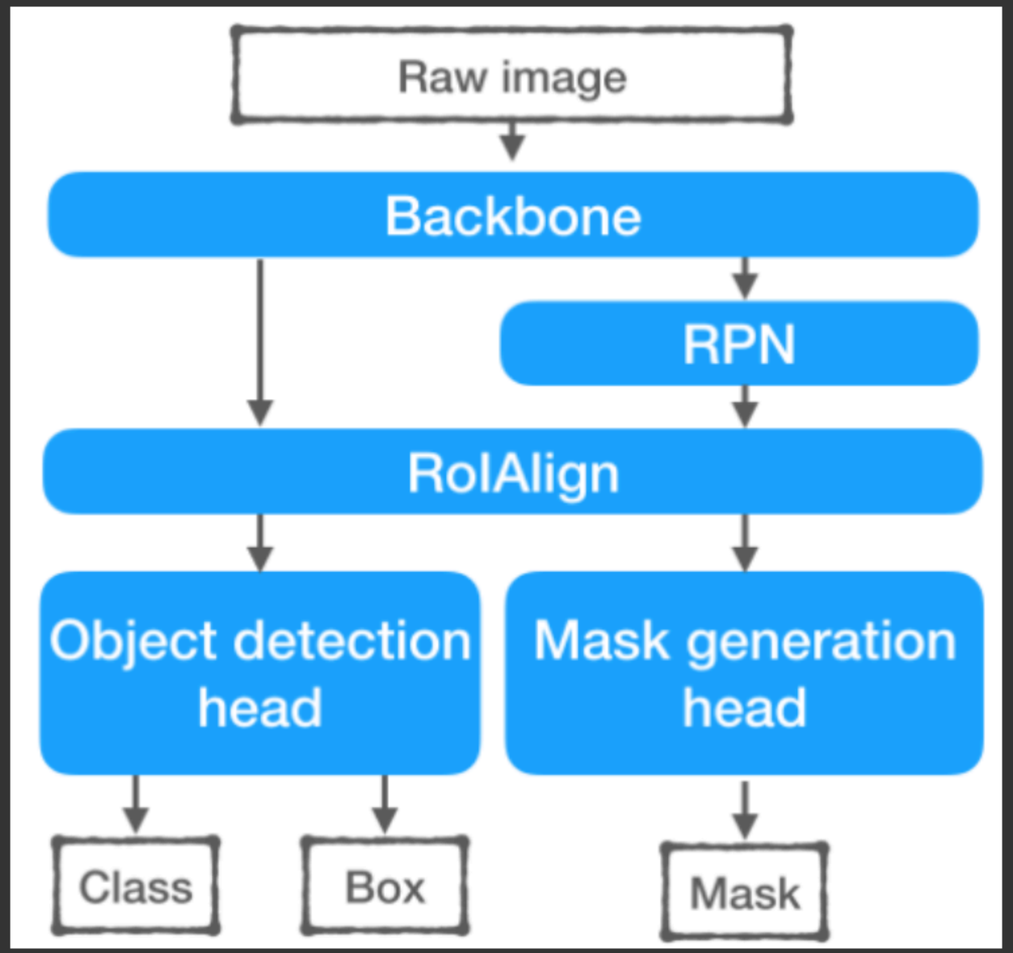
Hence a faster model with relatively less parameters is required.

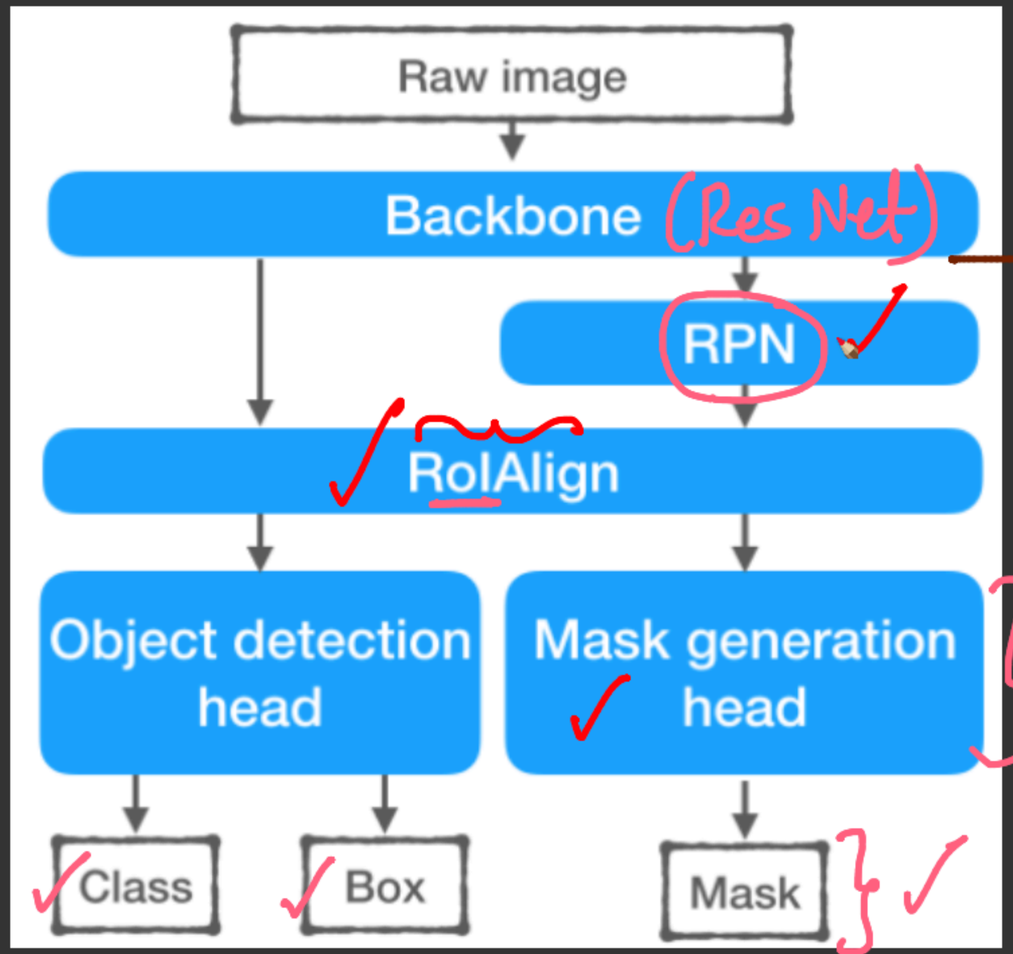
Intuition for Mask RCNN

- Recall from the Object Detection Lecture, Faster RCNN was the SOTA model that quickly produced highly accurate results

Intuition for Mask RCNN

- Recall from the Object Detection Lecture, Faster RCNN was the SOTA model that quickly produced highly accurate results





feature-map

Faster R-CNN vs Mask RCNN

- Mask R-CNN is just an extension of Faster R-CNN

What all can we notice ?

0.1	0.7	0.9	0.1

Input feature map for ROI pooling.

ROI Pooling:

0.3	0.4	0.2	0.1
0.5	0.1	0.9	0.7
0.3	0.6	0.2	0.2
0.1	0.7	0.9	0.1

2 x 2 →

0.3	0.4	0.2	0.1
0.5	0.1	0.9	0.7
0.3	0.6	0.2	0.2
0.1	0.7	0.9	0.1

Divide taken region into fixed size grid using proposals (updated coordinates).

max pool ROI

0.9	0.7
0.9	0.1

Output from ROI pool based in max pool operation.

ROI align

ROI align has the same goal as ROI pooling: take *Region of Interest* via *proposals*. But these two steps a little bit different.

1. ROI align divides each coordinate by *k*: **x / k** and do NOT take integer part.

0.1	0.7	0.9	0.1

Input feature map for ROI pooling.

ROI Pooling

0.3	0.4	0.2	0.1
0.5	0.1	0.9	0.7
0.3	0.6	0.2	0.2
0.1	0.7	0.9	0.1

2 x 2 →

0.3	0.4	0.2	0.1
0.5	0.1	0.9	0.7
0.3	0.6	0.2	0.2
0.1	0.7	0.9	0.1

Divide taken region into fixed size grid using proposals (updated coordinates).

max pool ROI

0.9	0.7
0.9	0.1

Output from ROI pool based in max pool operation.

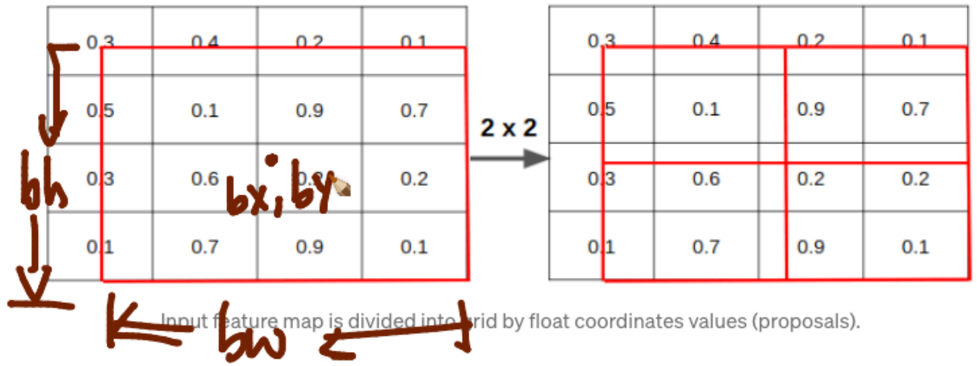
ROI align

ROI align has the same goal as ROI pooling: take *Region of Interest* via *proposals*. But these two steps a little bit different.

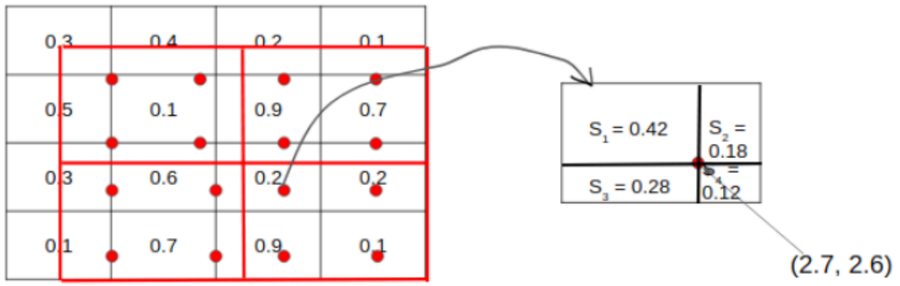
1. ROI align divides each coordinate by *k*: **x / k** and do NOT take integer part.

It means that there is no definite pixel in grid that can be taken, because new coordinates are **float** values.

2. Nevertheless, cropped part is also divided into **grid**, but for defining concrete values in these bins ROI align choose regularly **4 points** in each bin using **bilinear interpolation** (as shown in picture above). And from these 4 points maximum or average value from each bin is taken.



Input feature map is divided into grid by float coordinates values (proposals).



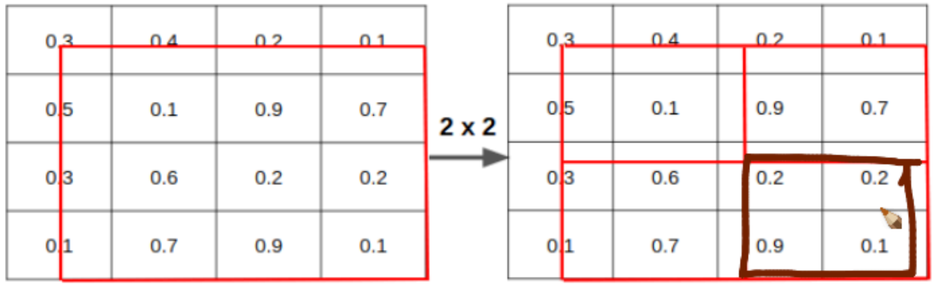
How to interpolate each point (red point within grid). Compute area of each piece in original cell that was divided by red point. Example for one of them.

InterpolatedValue

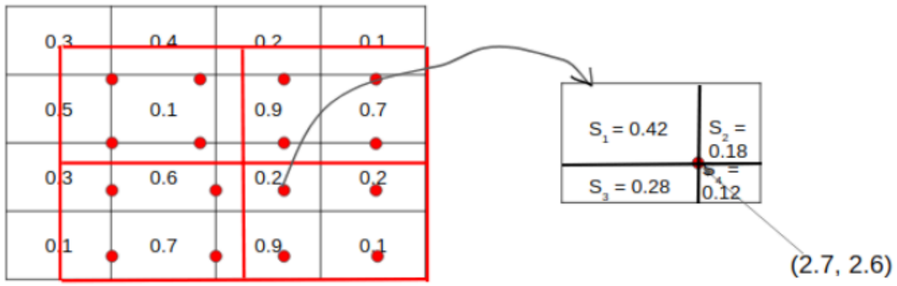
It means that there is no definite pixel in grid that can be taken, because new coordinates are **float** values.

2. Nevertheless, cropped part is also divided into **grid**, but for defining concrete values in these bins ROI align choose regularly **4 points** in each bin using **bilinear interpolation** (as shown in picture above). And from these 4 points maximum or average value from each bin is taken.

ROI Align



Input feature map is divided into grid by float coordinates values (proposals).

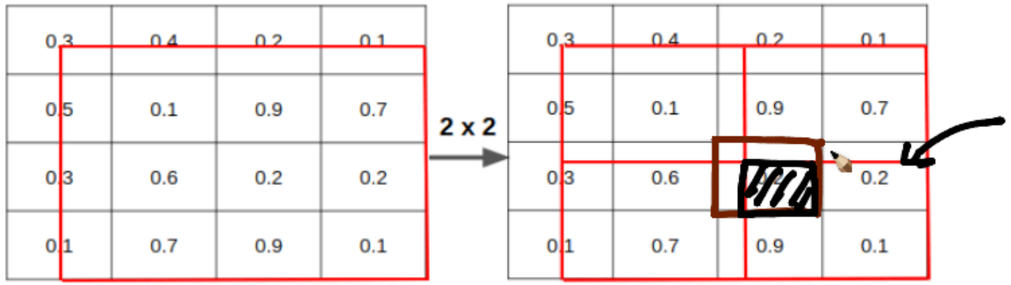


How to interpolate each point (red point within grid). Compute area of each piece in original cell that was divided by red point. Example for one of them.

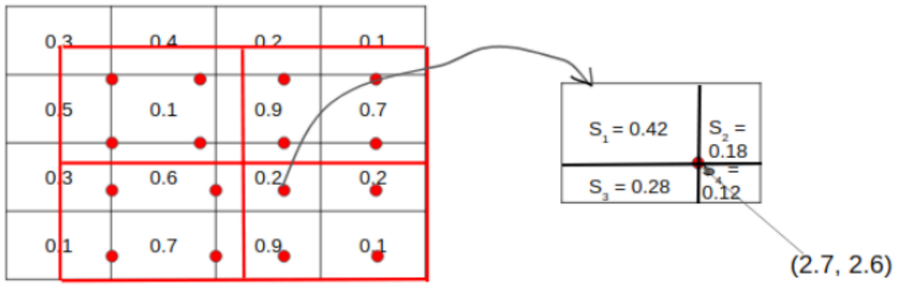
InterpolatedValue

It means that there is no definite pixel in grid that can be taken, because new coordinates are **float** values.

2. Nevertheless, cropped part is also divided into **grid**, but for defining concrete values in these bins ROI align choose regularly **4 points** in each bin using **bilinear interpolation** (as shown in picture above). And from these 4 points maximum or average value from each bin is taken.



Input feature map is divided into grid by float coordinates values (proposals).

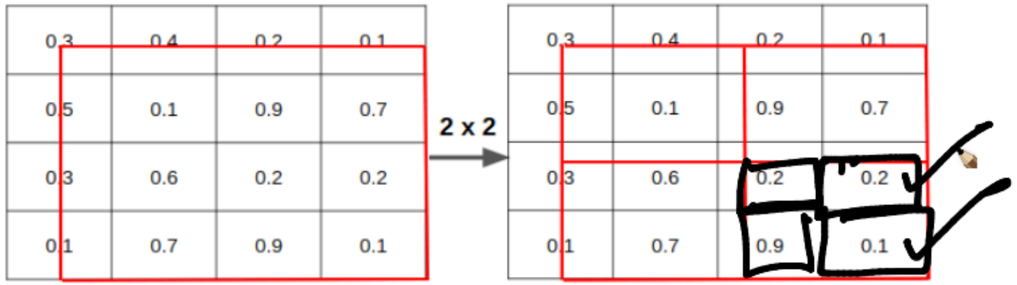


How to interpolate each point (red point within grid). Compute area of each piece in original cell that was divided by red point. Example for one of them.

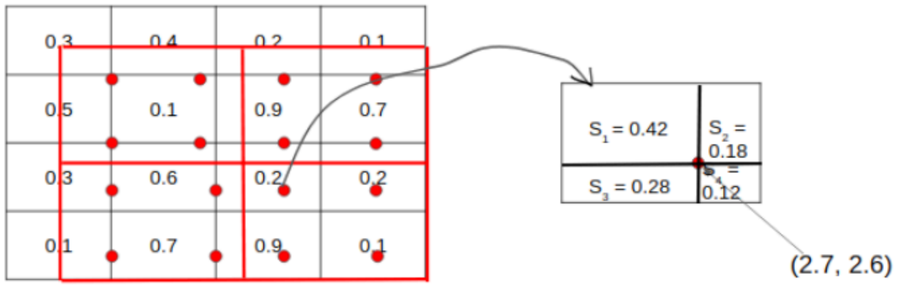
InterpolatedValue

It means that there is no definite pixel in grid that can be taken, because new coordinates are **float** values.

2. Nevertheless, cropped part is also divided into **grid**, but for defining concrete values in these bins ROI align choose regularly **4 points** in each bin using **bilinear interpolation** (as shown in picture above). And from these 4 points maximum or average value from each bin is taken.



Input feature map is divided into grid by float coordinates values (proposals).

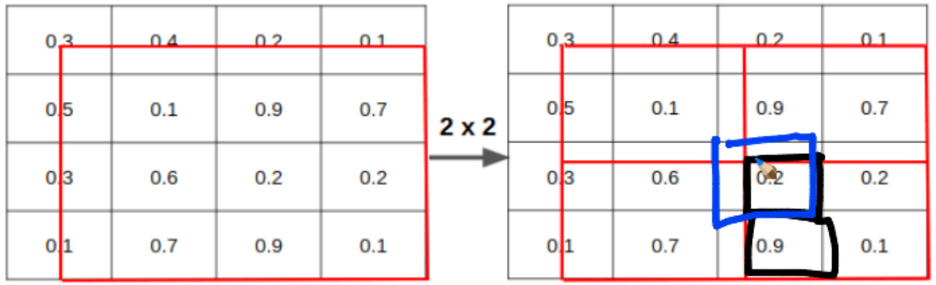


How to interpolate each point (red point within grid). Compute area of each piece in original cell that was divided by red point. Example for one of them.

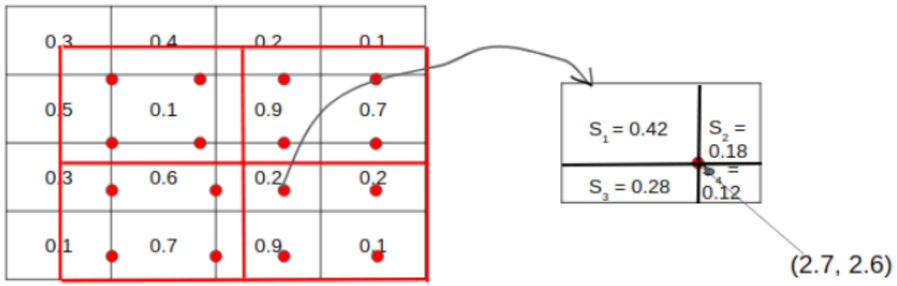
InterpolatedValue

It means that there is no definite pixel in grid that can be taken, because new coordinates are **float** values.

2. Nevertheless, cropped part is also divided into **grid**, but for defining concrete values in these bins ROI align choose regularly **4 points** in each bin using **bilinear interpolation** (as shown in picture above). And from these 4 points maximum or average value from each bin is taken.



Input feature map is divided into grid by float coordinates values (proposals).

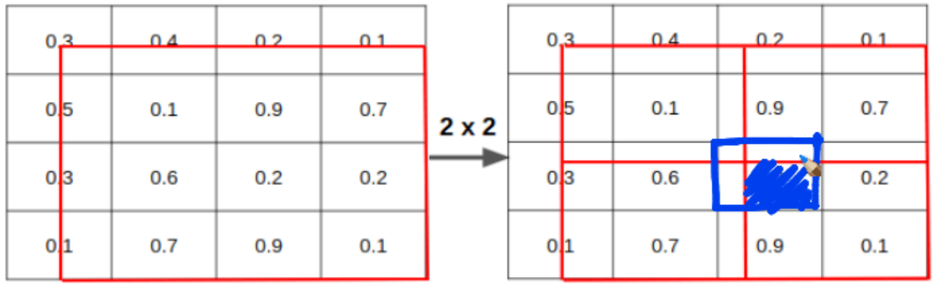


How to interpolate each point (red point within grid). Compute area of each piece in original cell that was divided by red point. Example for one of them.

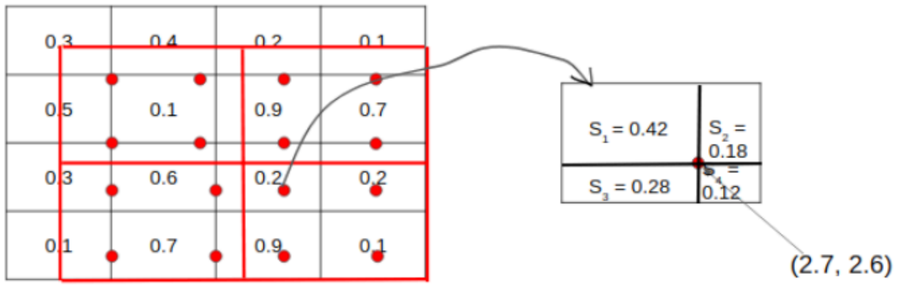
InterpolatedValue

It means that there is no definite pixel in grid that can be taken, because new coordinates are **float** values.

2. Nevertheless, cropped part is also divided into **grid**, but for defining concrete values in these bins ROI align choose regularly **4 points** in each bin using **bilinear interpolation** (as shown in picture above). And from these 4 points maximum or average value from each bin is taken.



Input feature map is divided into grid by float coordinates values (proposals).

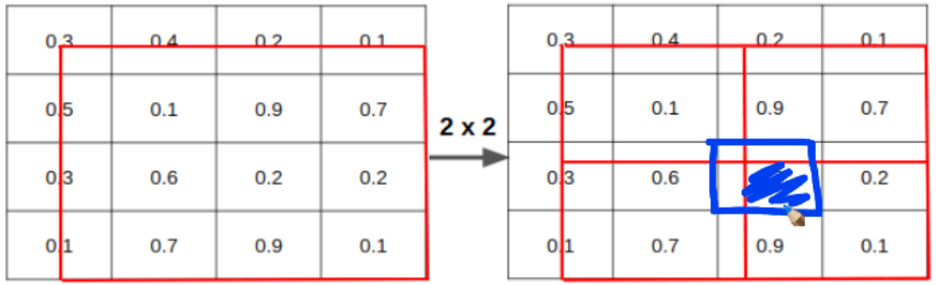


How to interpolate each point (red point within grid). Compute area of each piece in original cell that was divided by red point. Example for one of them.

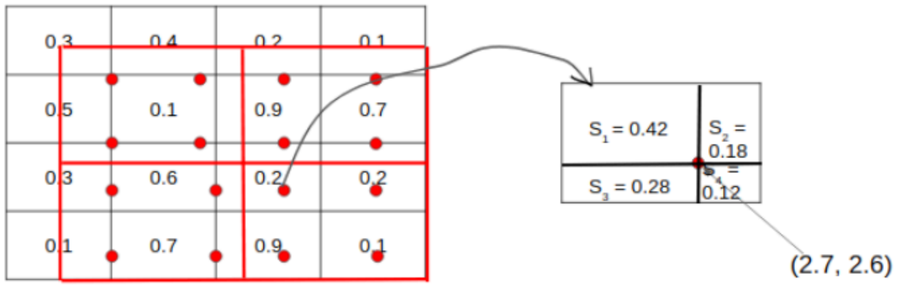
InterpolatedValue

It means that there is no definite pixel in grid that can be taken, because new coordinates are **float** values.

2. Nevertheless, cropped part is also divided into **grid**, but for defining concrete values in these bins ROI align choose regularly **4 points** in each bin using **bilinear interpolation** (as shown in picture above). And from these 4 points maximum or average value from each bin is taken.



Input feature map is divided into grid by float coordinates values (proposals).

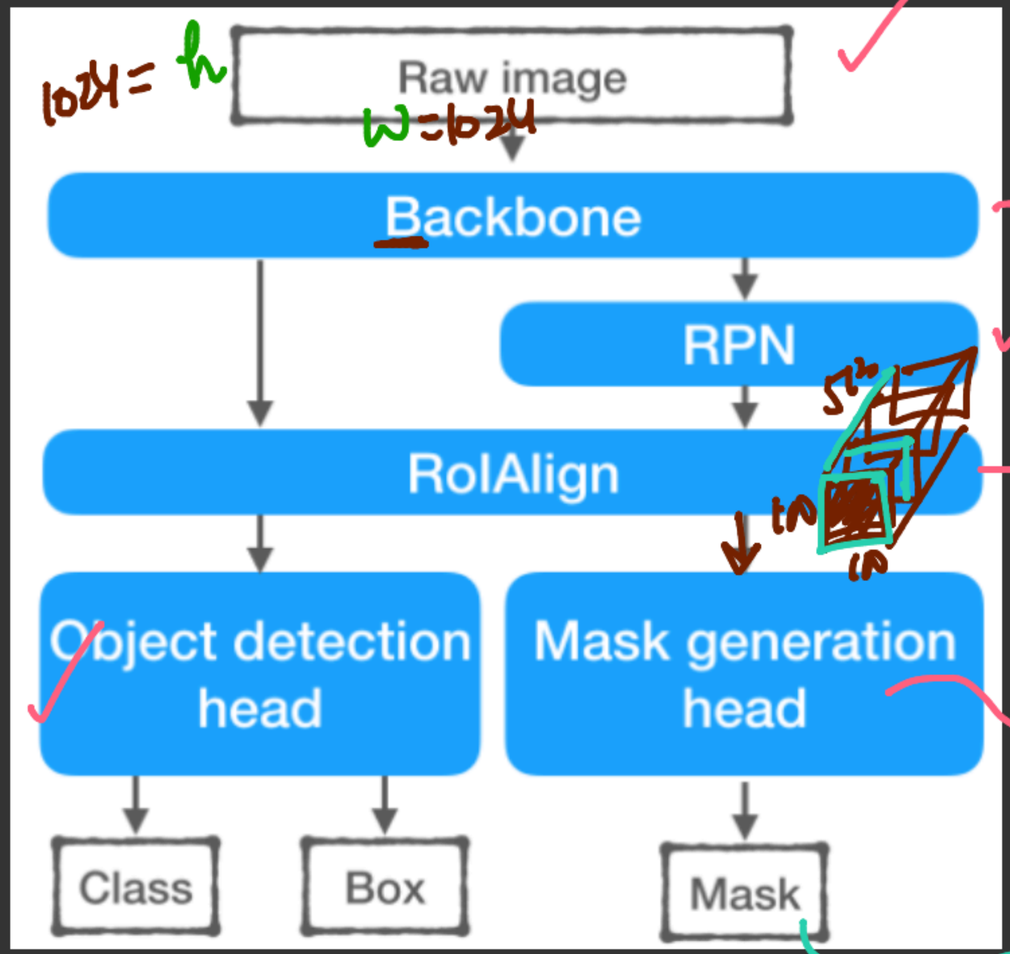


How to interpolate each point (red point within grid). Compute area of each piece in original cell that was divided by red point. Example for one of them.

InterpolatedValue

Intuition for Mask RCNN

- Recall from the Object Detection Lecture, Faster RCNN was the SOTA model that quickly produced highly accurate results



Mask-RCNN

→ ResNet

→ same as faster RCNN

→ ~ RoIPooling

Upscaling

Transposed conv

→ unet

→ FCN

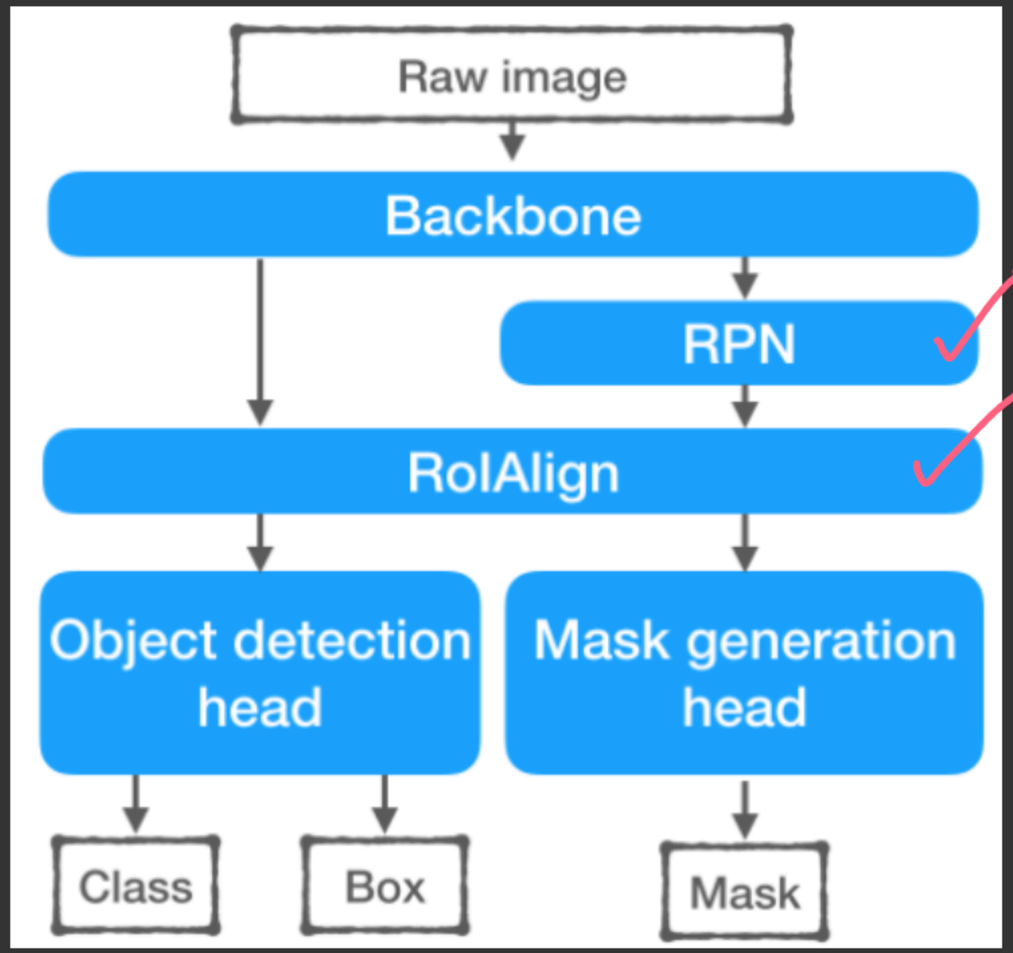
h

w

hxwx #classes

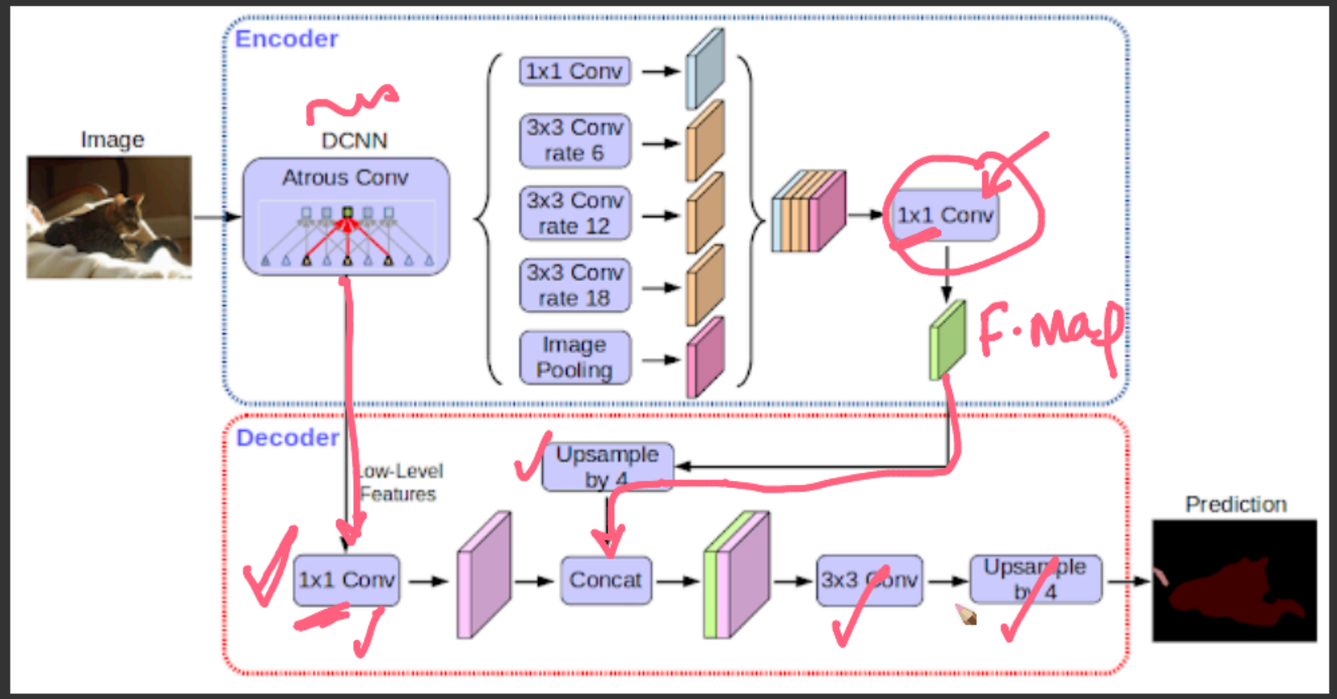
Intuition for Mask RCNN

- Recall from the Object Detection Lecture, Faster RCNN was the SOTA model that quickly produced highly accurate results

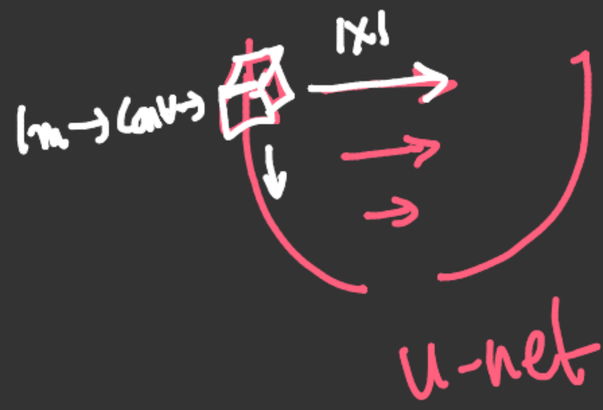


Mask RCNN

- There is another SOTA model that usually performs better due to its recent restructuring in architecture and introducing much less computationally atrous/dilated Convolution layer
- https://arxiv.org/pdf/1706.05587v3.pdf

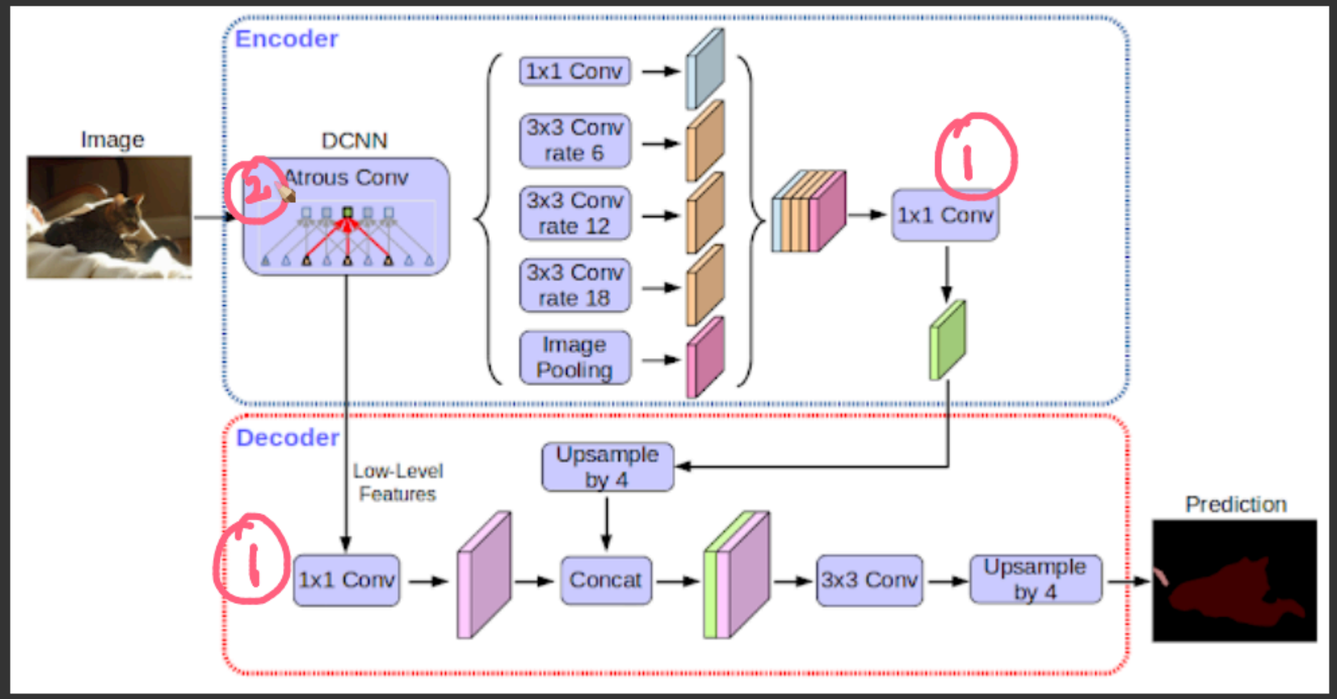


Atrous-conv /Dilated conv



Atrous Convolution

- There is another SOTA model that usually performs better due to its recent restructuring in architecture and introducing much less computationally atrous/dilated Convolution layer
- https://arxiv.org/pdf/1706.05587v3.pdf

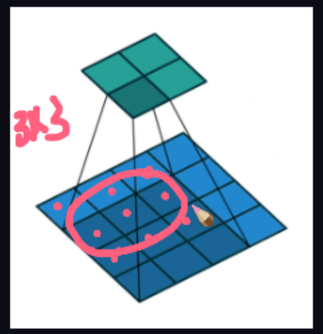
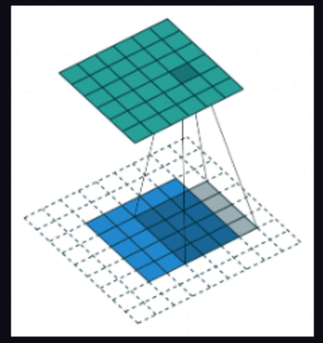
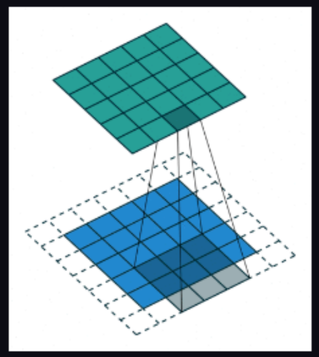
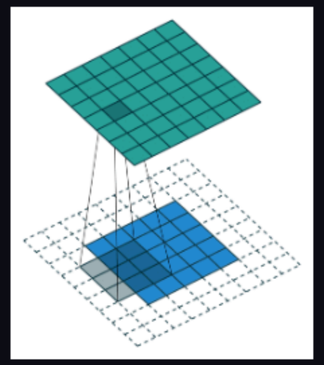
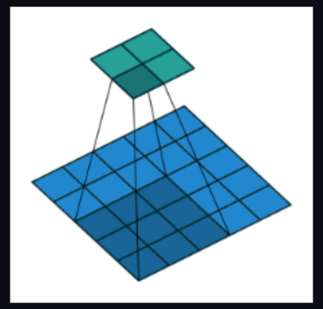
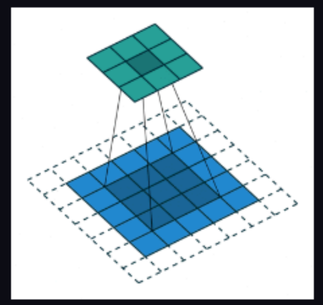
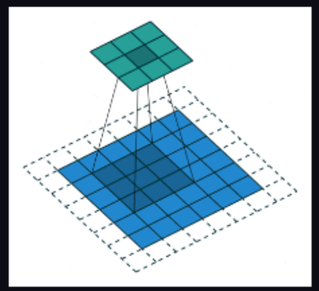


Atrous Convolution

README.md

Convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.

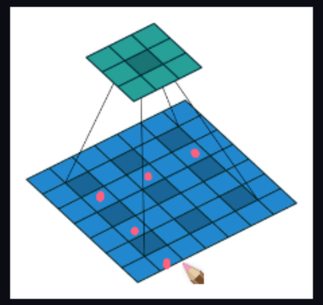
			
No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides
			
No padding, strides	Padding, strides	Padding, strides (odd)	

TeX 79.7% Python 20.3%

README.md

Dilated convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.



No padding, no stride, dilation

Generating the Makefile

From the repository's root directory:

```
$ ./bin/generate_makefile
```

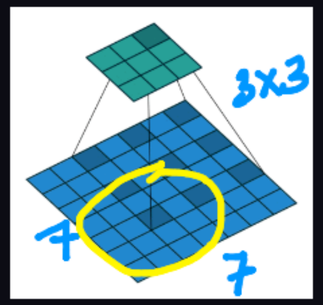
Generating the animations

From the repository's root directory:

README.md

Dilated convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.

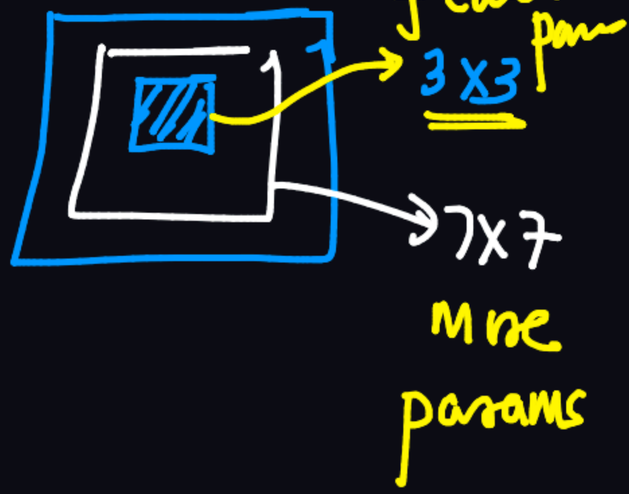


dil.conv

No padding, no stride, dilation

Pros of
(Q) Dilated conv vs regular conv?

↳ global/more expansive view 3x3 conv



Generating the Makefile

From the repository's root directory:

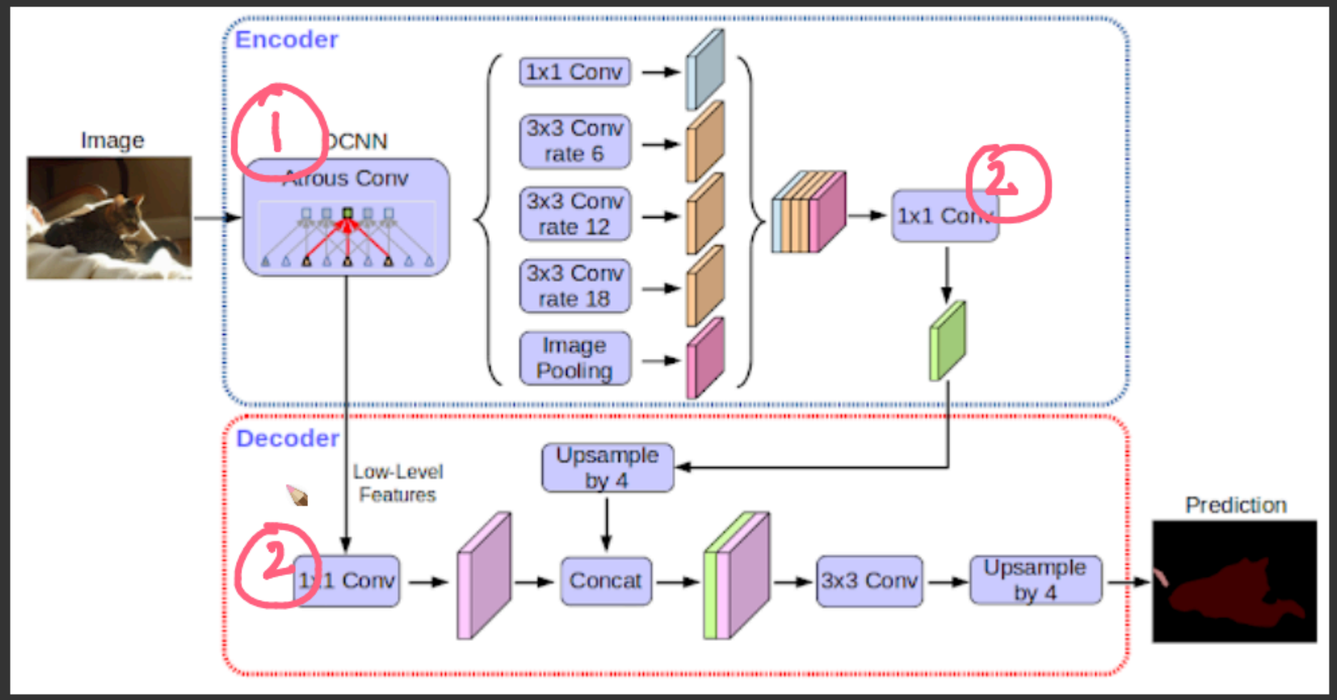
```
$ ./bin/generate_makefile
```

Generating the animations

From the repository's root directory:

Intuition of DeepLabV3 (OPTIONAL)

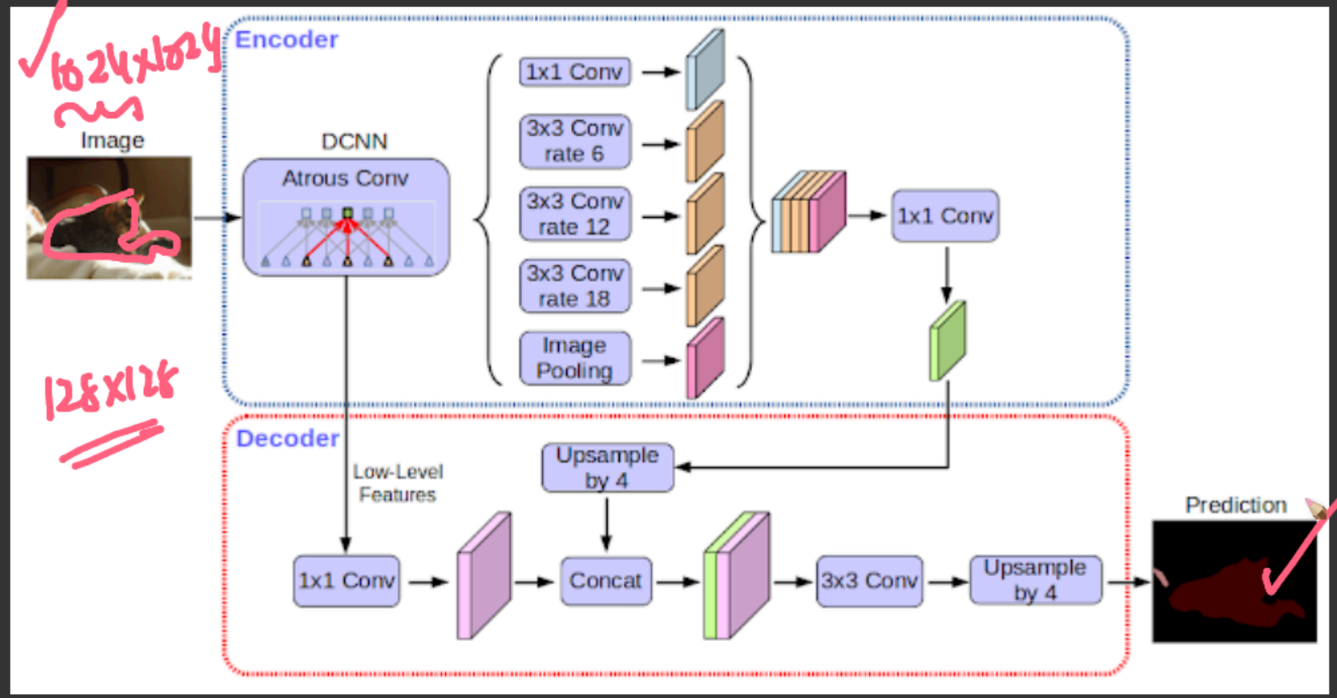
- There is another SOTA model that usually performs better due to its recent restructuring in architecture and introducing much less computationally atrous/dilated Convolution layer
- https://arxiv.org/pdf/1706.05587v3.pdf



Atrous Convolution

Intuition of DeepLabV3 (OPTIONAL)

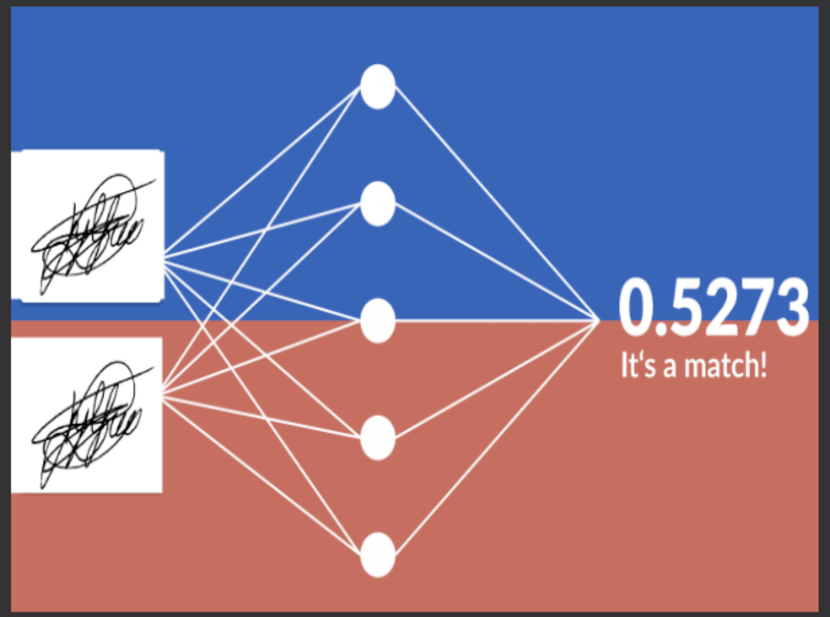
- There is another SOTA model that usually performs better due to its recent restructuring in architecture and introducing much less computationally atrous/dilated Convolution layer
- https://arxiv.org/pdf/1706.05587v3.pdf



Atrous Convolution

Problem Statement

- You are working as Data scientist in HDFC bank.
  - You have a new account holder in your bank and would like to set his signature for verification.
- You have only one sample signature from the member.
- Being a DataScientist , How can you use a neural network to perform the signature verification?



1000 - customers

3-5 signatures

samples per class

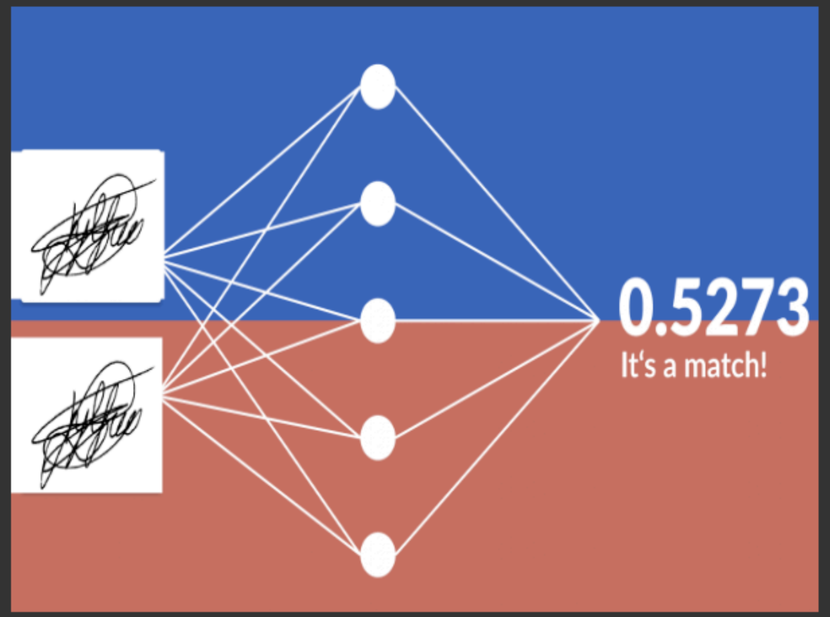
per person

TASK: Verify whether a Signature of a person is Genuine or Forged

Problem Statement

Few shot

- You are working as Data scientist in HDFC bank.
  - You have a new account holder in your bank and would like to set his signature for verification.
- You have only one sample signature from the member.
- Being a DataScientist , How can you use a neural network to perform the signature verification?



TASK: Verify whether a Signature of a person is Genuine or Forged

Train:

100 - classes

5 images per class

500 ✗ CNN

Transfer Learning (✓)

(Q) can I re-look @ the scale of data

minor - augmentation

1 2 3 4 5 6

#arrangements

Combinatorial explosion

(2) _ _ → 6 × 5 = 30 ways

(3) _ _ _ → 6 × 5 × 4 = 120 ways

100 - classes
5 images class
} → 500 points

500_{C_2} - # similar pairs

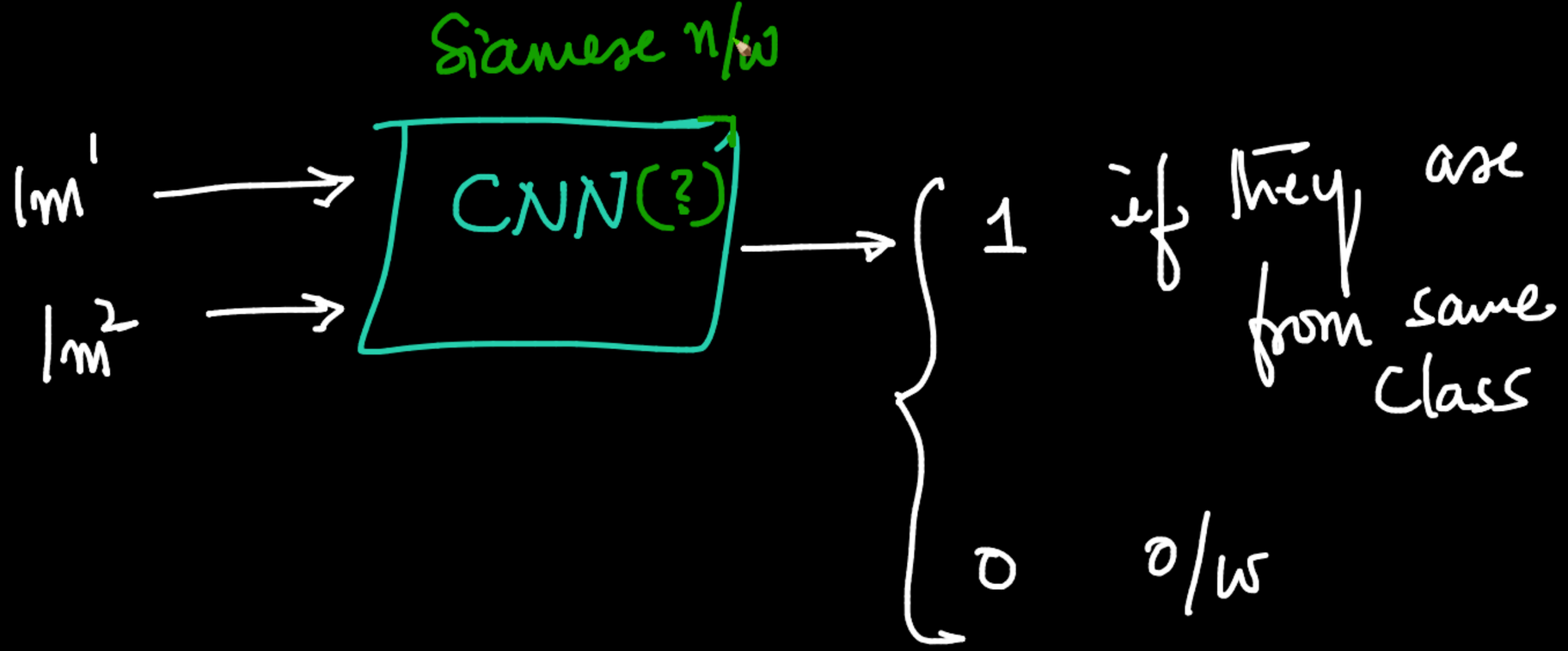
$500_{C_2} = \frac{500 \times 499}{2}$ pairs:

I_2^{10} I_3^{11} → dissimilar - classes → 0

I_2^{10} I_3^{10} → similar - class → 1

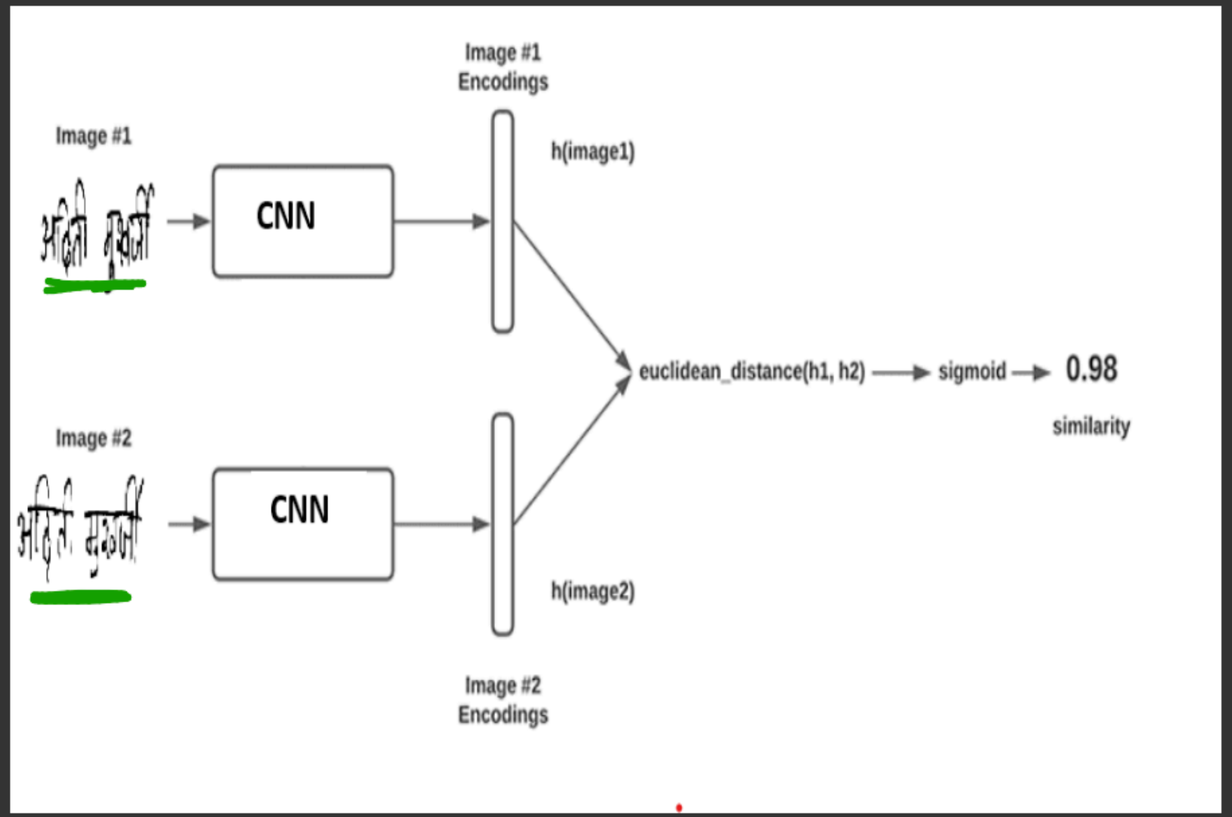
v. large # pairs

$100 \times {}^5C_2$ similar pairs



What is a Siamese Network?

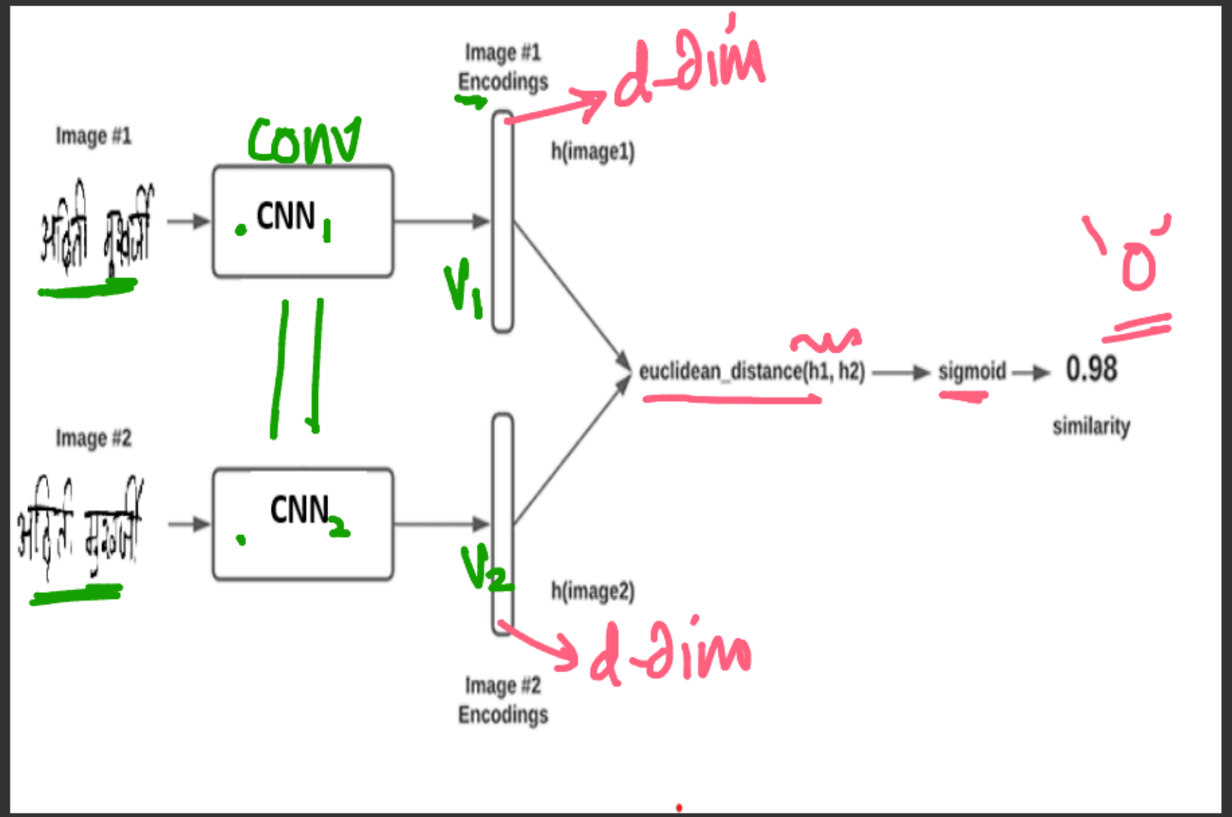
- https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf



- A **Siamese neural network** is an artificial neural network that contains two or more identical subnetwork which is also known as **twin**

What is a Siamese Network?

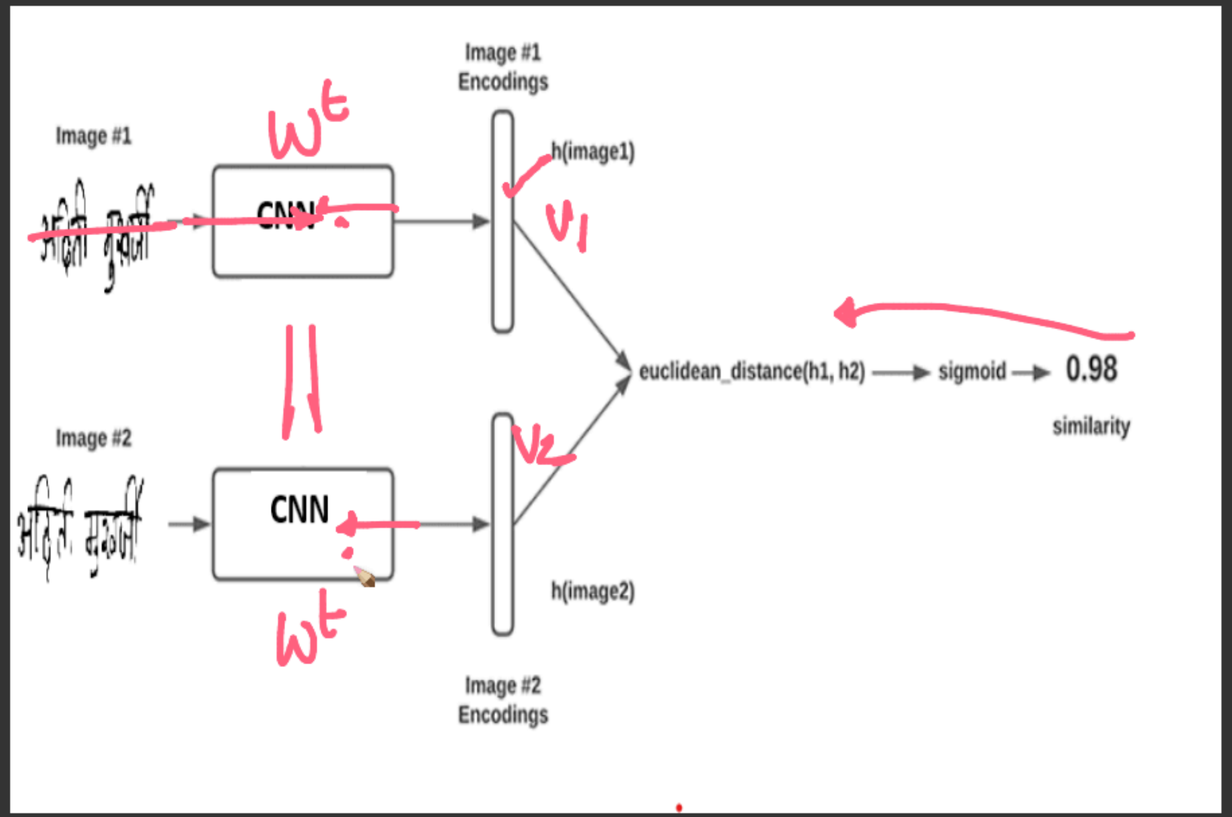
- https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf



- A **Siamese neural network** is an artificial neural network that contains two or more identical subnetwork which is also known as **twin**

What is a Siamese Network?

- https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf



- A **Siamese neural network** is an artificial neural network that contains two or more identical subnetwork which is also known as **twin**

Building the Siamese Model

```
[ ] # Building Model for feature extraction
    # base_network = build_base_network(input_dim)

    # Shape of the Input
    img_a = Input(shape=input_dim)
    img_b = Input(shape=input_dim)
    # label_inp = Input(shape=(1,))

    # Extracting embeddings of the image
    feat_vecs_a = embedding(resnet.preprocess_input(img_a))
    feat_vecs_b = embedding(resnet.preprocess_input(img_b))

    # Calculating euclidean distance between the images
    distance = Lambda(lambda tensors:K.abs(tensors[0]-tensors[1]))([feat_vecs_a, feat_vecs_b])
    prediction=Dense(1,activation='sigmoid')(distance)
    model = Model([img_a, img_b],prediction)
    model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)            [(None, 224, 224, 3  0           []
                                 )]
```

gradient problem in very deep CNNs.

```
[ ] input_dim = (224,224,3)
```

```
[ ] base_cnn = resnet.ResNet50(
        weights="imagenet", input_shape=input_dim, include_top=False
    )

    flatten = layers.Flatten()(base_cnn.output)
    dense1 = layers.Dense(512, activation="relu",kernel_regularizer=tf.keras.regularizers.l2(0.001))(flatten)
    dense1 = layers.Dropout(0.3)(dense1)
    # dense1 = layers.BatchNormalization()(dense1)
    dense2 = layers.Dense(256, activation="relu",kernel_regularizer=tf.keras.regularizers.l2(0.001))(dense1)
    dense2 = layers.Dropout(0.3)(dense2)
    # dense2 = layers.BatchNormalization()(dense2)
    output = layers.Dense(256)(dense2)

    embedding = Model(base_cnn.input, output, name="Embedding")

    trainable = False
    for layer in base_cnn.layers:
        if layer.name == "conv5_block1_out":
            trainable = True
        layer.trainable = trainable
```

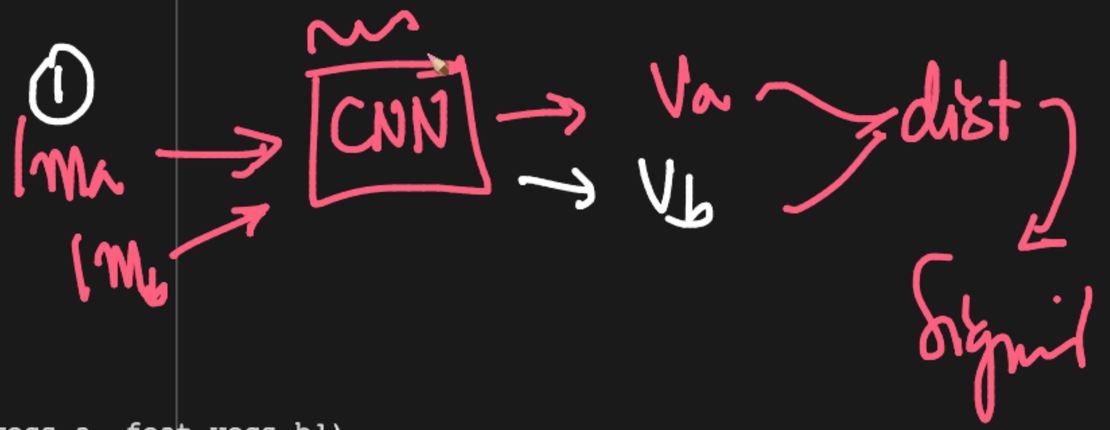
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_noto
94765736/94765736 [==============================] - 2s 0us/step
```

```
[ ]  # Building Model for feature extraction
     # base_network = build_base_network(input_dim)

     # Shape of the Input
     img_a = Input(shape=input_dim)
     img_b = Input(shape=input_dim)
     # label_inp = Input(shape=(1,))

     # Extracting embeddings of the image
     feat_vecs_a = embedding(resnet.preprocess_input(img_a))
     feat_vecs_b = embedding(resnet.preprocess_input(img_b))

     # Calculating euclidean distance between the images
     distance = Lambda(lambda tensors:K.abs(tensors[0]-tensors[1]))([feat_vecs_a, feat_vecs_b])
     prediction=Dense(1,activation='sigmoid')(distance)
     model = Model([img_a, img_b],prediction)
     model.summary()
```



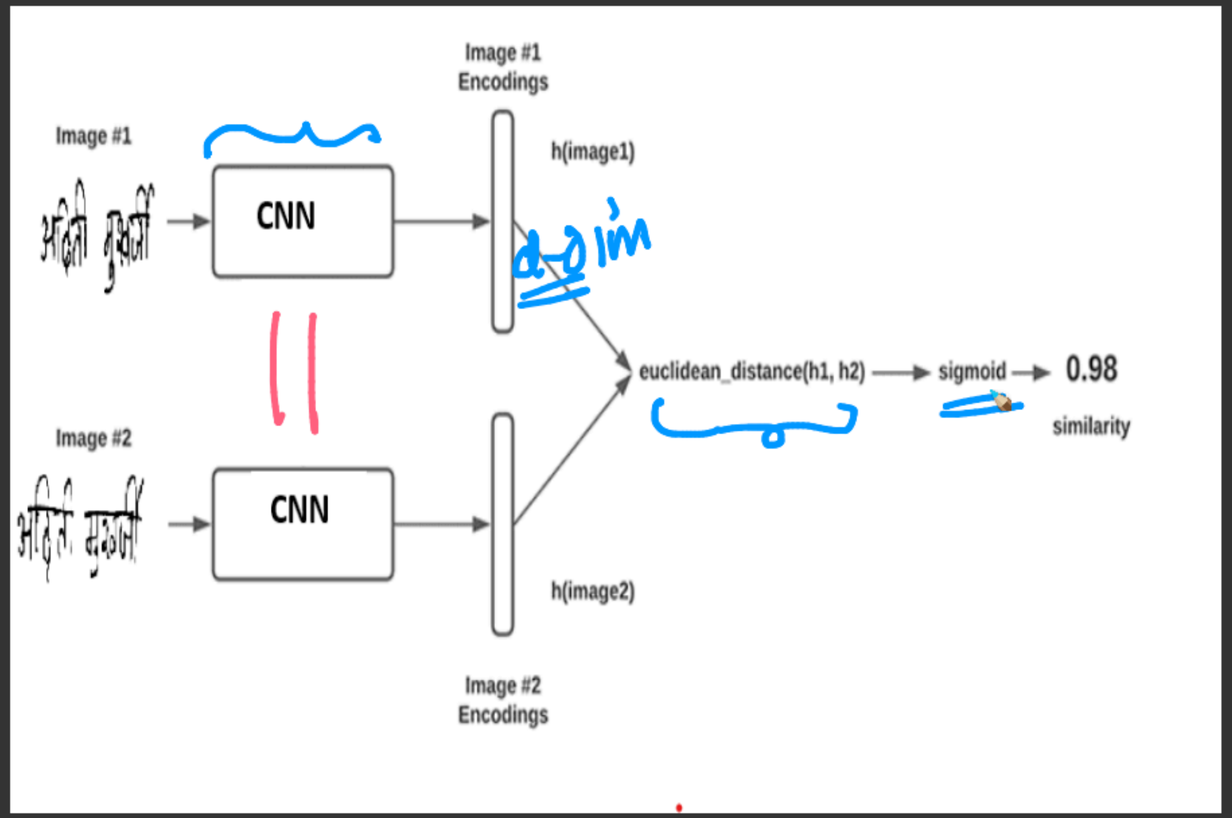

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 input_3 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 tf.__operators__.getitem (Slic  (None, 224, 224, 3)  0          ['input_2[0][0]']
 ingOpLambda)
```

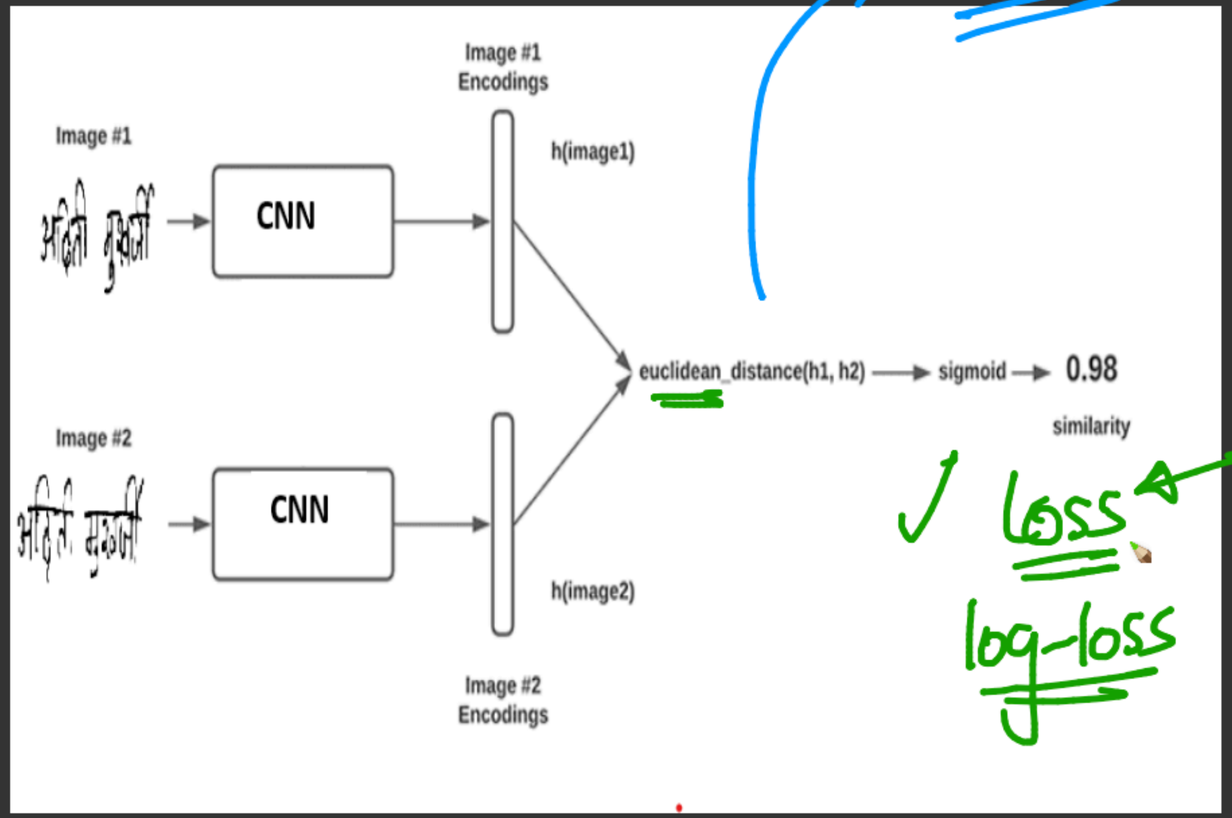
- https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf

5 images
1000 Classes
Run-time ?



- A **Siamese neural network** is an artificial neural network that contains two or more identical subnetwork which is also known as **twin neural network** or **sister network**.
- Siamese network takes two different inputs passed through two similar subnetworks with the same architecture, parameters, and

- https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf



Problem:

Run-time

$V_N \in \mathbb{R}^2$

$x_N \rightarrow$ CNN $\rightarrow$

Tr

5000 images

$V_i \in \mathbb{R}^2$

loss

log-loss

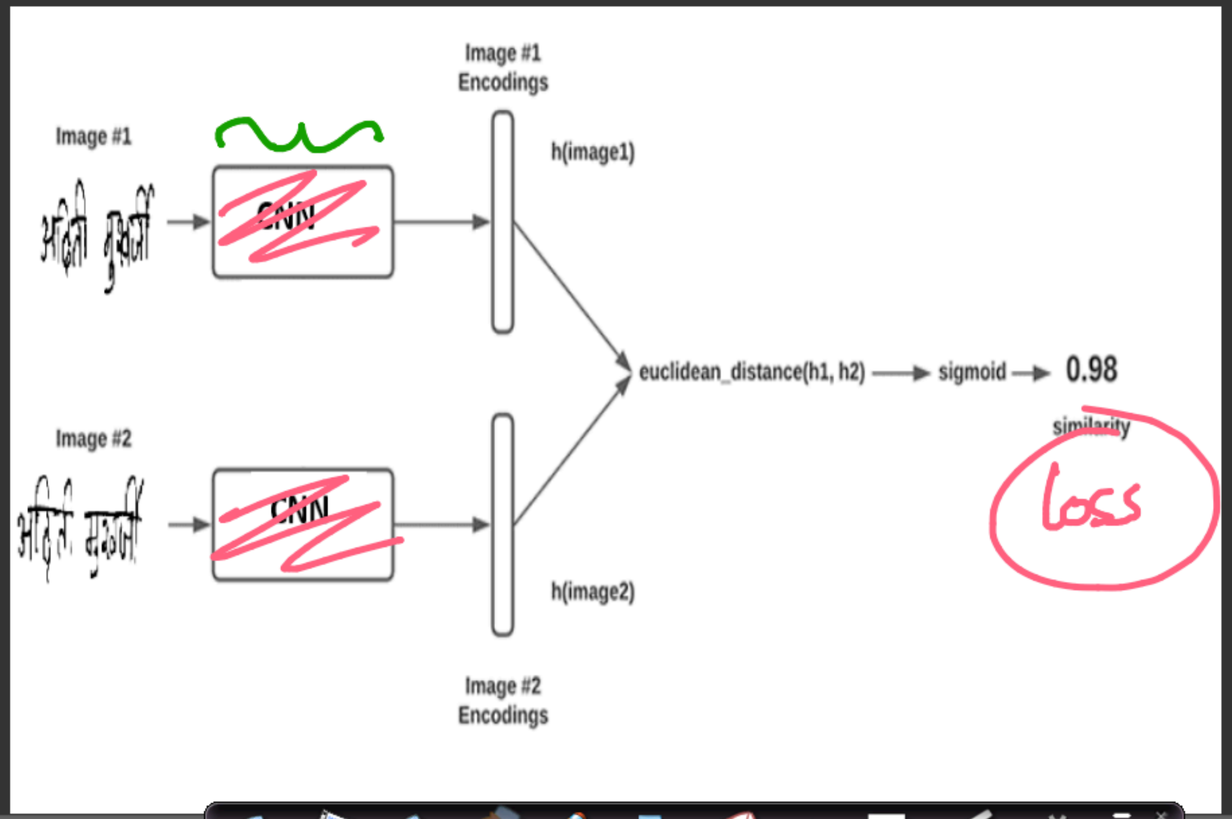
- A **Siamese neural network** is an artificial neural network that contains two or more identical subnetwork which is also known as **twin neural network** or **sister network**.
- Siamese network takes two different inputs passed through two similar subnetworks with the same architecture, parameters, and

recognition does not.

- With face recognition, you don't know which face you're looking for; with face verification, you do.

What is a Siamese Network?

- https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf



Contrastive loss
Triplet-loss
+ Sigmoid

Dataset

loss

→ pairs ✓

→ Triplets

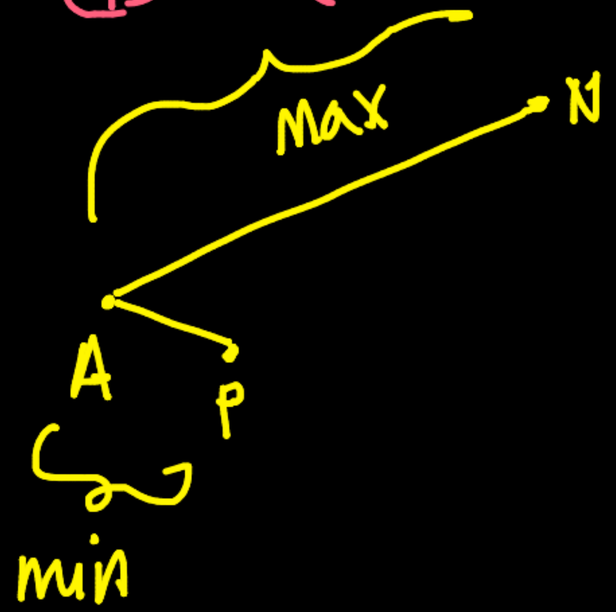
I_i^{10} — Anchor ✓

I_{i+1}^{10} — +ve (P)

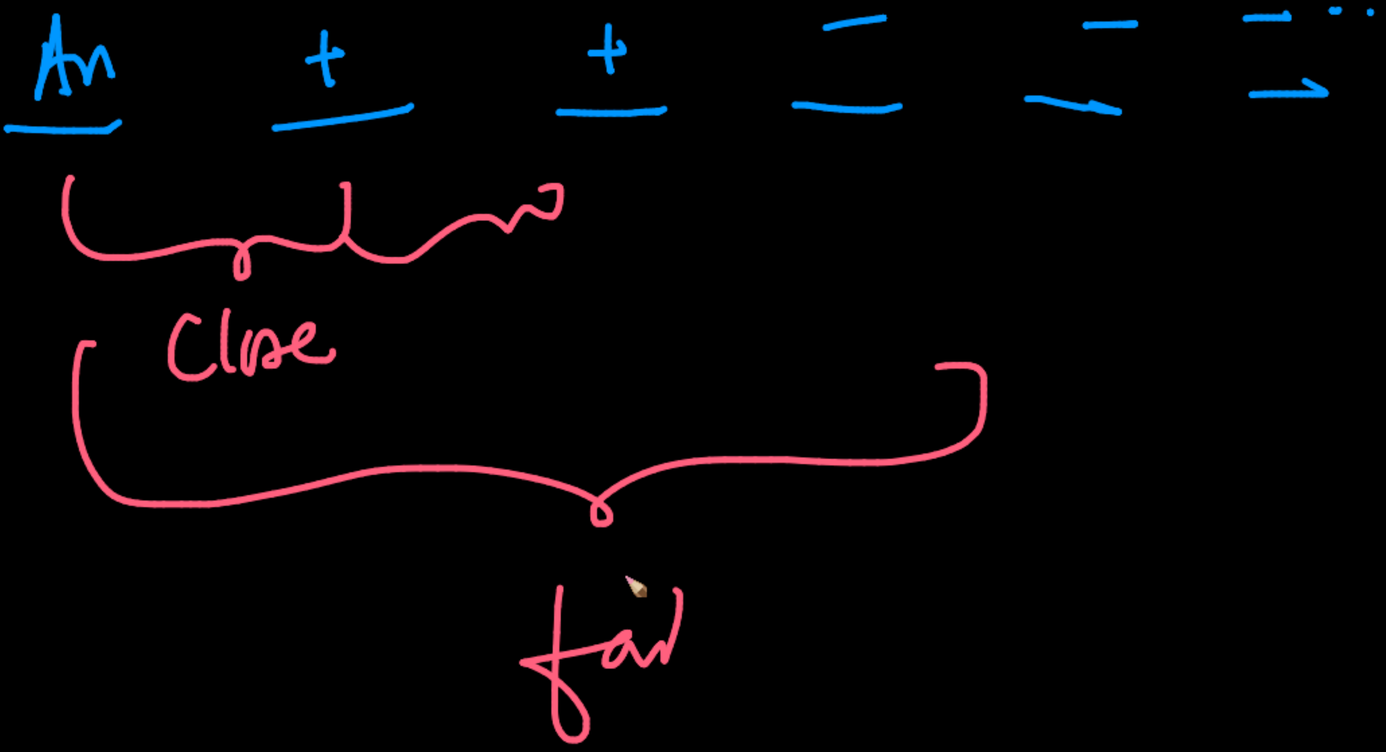
$I_i^{j \neq i}$ — -ve (N)

→ combinatorial explosion !

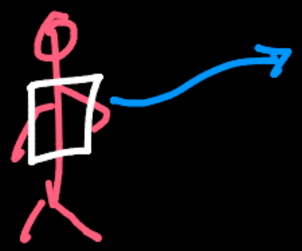
Triplet-loss (later)



→ 10 any



Q



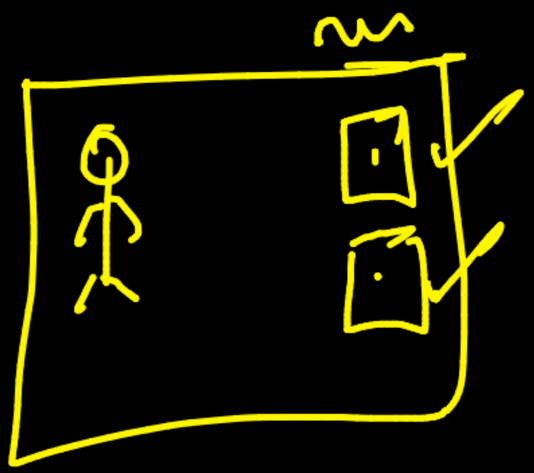
ResNet 50 → d-dim

CNN

V_q

eCommerce

shirts



$v_i \in \mathbb{R}^d$

{ item-item sim (recsys)

Approx NN

Why contrastive loss is used?

A siamese network's purpose is to distinguish between picture pairings rather than classify them. Essentially, contrastive loss measures how well the siamese network distinguishes between picture pairings. The distinction is minor yet significant.

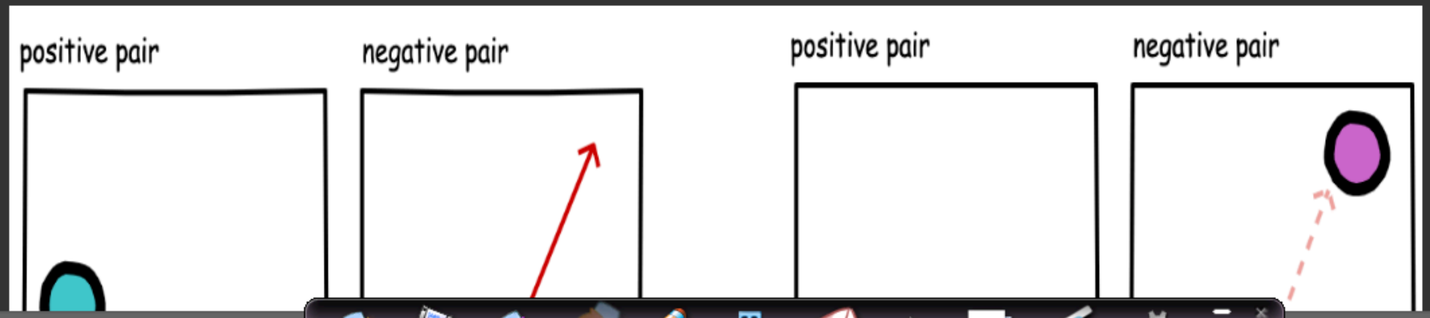
Contrastive Loss is defined as,

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

where D is the calculated distance between embeddings for each datapoint in the pair, and m is a constant value of margin.

To break this equation down:

1. The Y value is our label. It will be 1 if the image pairs are of the same class, and it will be 0 if the image pairs are of a different class.
2. The D variable is the Euclidean distance between the outputs of the sister network embeddings.
3. The max function takes the largest value of 0 and the margin, m, minus the distance.



Why contrastive loss is used?

A siamese network's purpose is to distinguish between picture pairings rather than classify them. Essentially, contrastive loss measures how well the siamese network distinguishes between picture pairings. The distinction is minor yet significant.

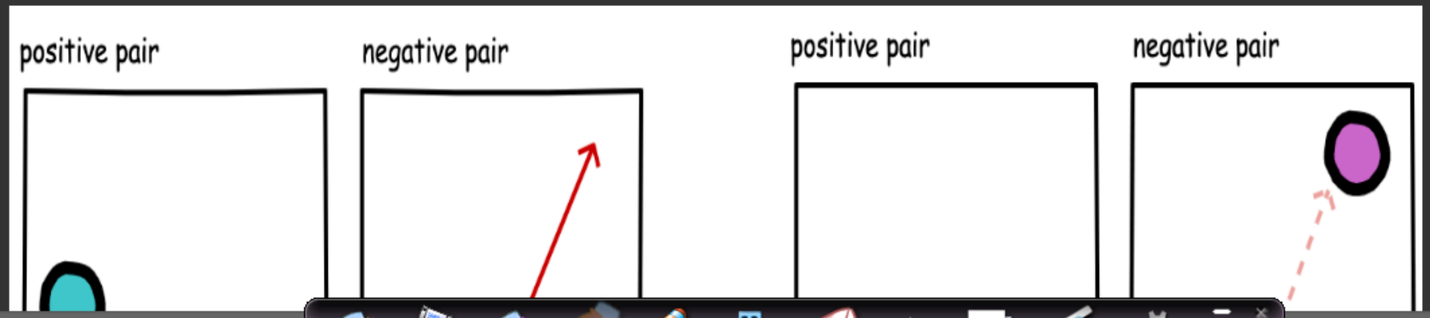
Contrastive Loss is defined as,

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

where D is the calculated distance between embeddings for each datapoint in the pair, and m is a constant value of margin.

To break this equation down:

1. The Y value is our label. It will be 1 if the image pairs are of the same class, and it will be 0 if the image pairs are of a different class.
2. The D variable is the Euclidean distance between the outputs of the sister network embeddings.
3. The max function takes the largest value of 0 and the margin, m, minus the distance.



Why contrastive loss is used?

A siamese network's purpose is to distinguish between picture pairings rather than classify them. Essentially, contrastive loss measures how well the siamese network distinguishes between picture pairings. The distinction is minor yet significant.

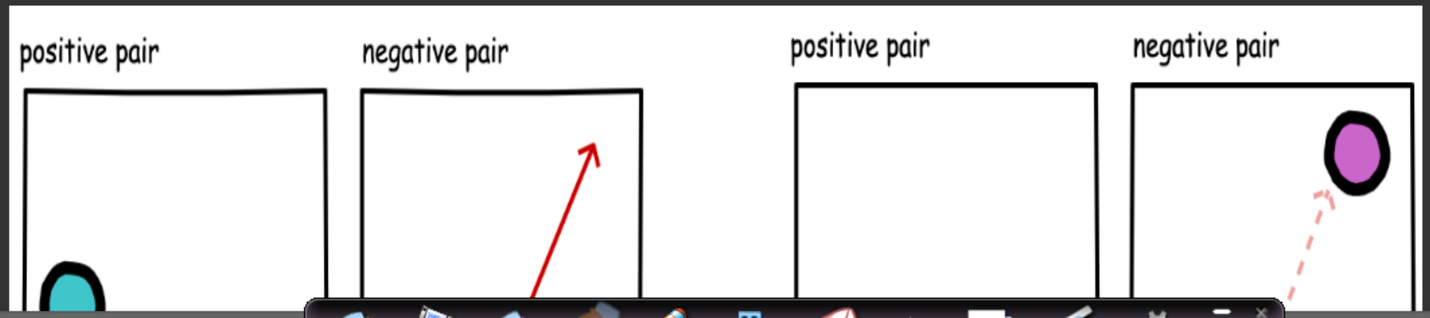
Contrastive Loss is defined as,

$$Y * D^2 + (1 - Y) * max(margin - D, 0)^2$$

where D is the calculated distance between embeddings for each datapoint in the pair, and m is a constant value of margin.

To break this equation down:

1. The Y value is our label. It will be 1 if the image pairs are of the same class, and it will be 0 if the image pairs are of a different class.
2. The D variable is the Euclidean distance between the outputs of the sister network embeddings.
3. The max function takes the largest value of 0 and the margin, m, minus the distance.



Training Accuracy: 71% Test Accuracy: 73%

- Siamese Model performs better with contrastive loss compared to binary cross entropy
- As we discussed above, Contrastive loss requires 2 inputs, **(anchor, positive)** or **(anchor, negative)**.

Why contrastive loss is not the best?

- While contrastive loss is useful, it has a limitation.
- In **Contrastive Loss**, it is already satisfied when different samples are easily distinguishable from similar ones.
- For points in a negative pair, contrastive loss will push them far apart without any knowledge of the broader embedding space.
- For example: imagine we have 10 classes, and each time we see class 1 and 2, we want to push them far apart; a result of this is that 1 might now become farther from 2 on the average, but might overlap with other classes (such as class 3, 4, 5, etc.)
- In **Triplet loss** it doesn't urge encoding anchor and positive sample in the same point within the vector space, unlike the contrastive loss.
- Also, it allows minor differences between similar samples from the same category.

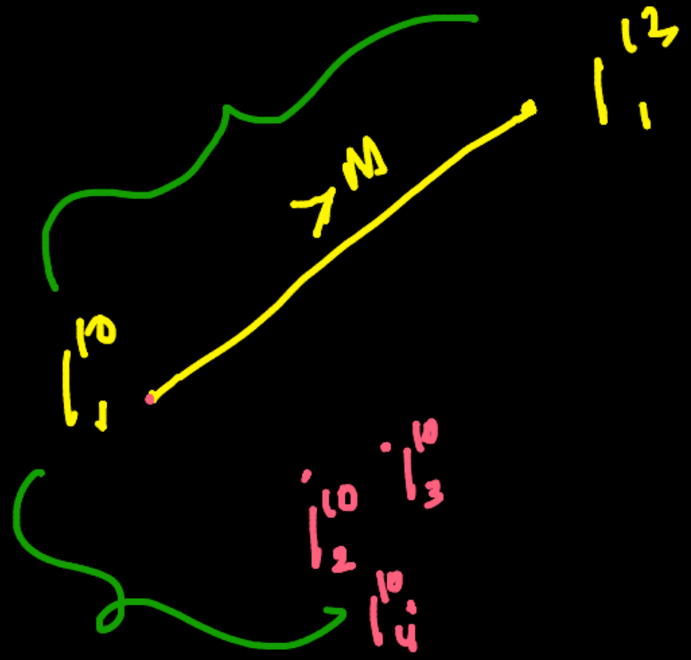
How can we use triplet loss to solve this problem?

Quiz-3

The formula for the contrastive loss, the function that is used in the siamese network for calculating image similarity, is defined as following:

$$D^2 + (1 - Y) * max(margin - D, 0)^2$$

(a) Margin is a constant that we use to enforce a maximum distance between the two images in order to consider them similar or different from



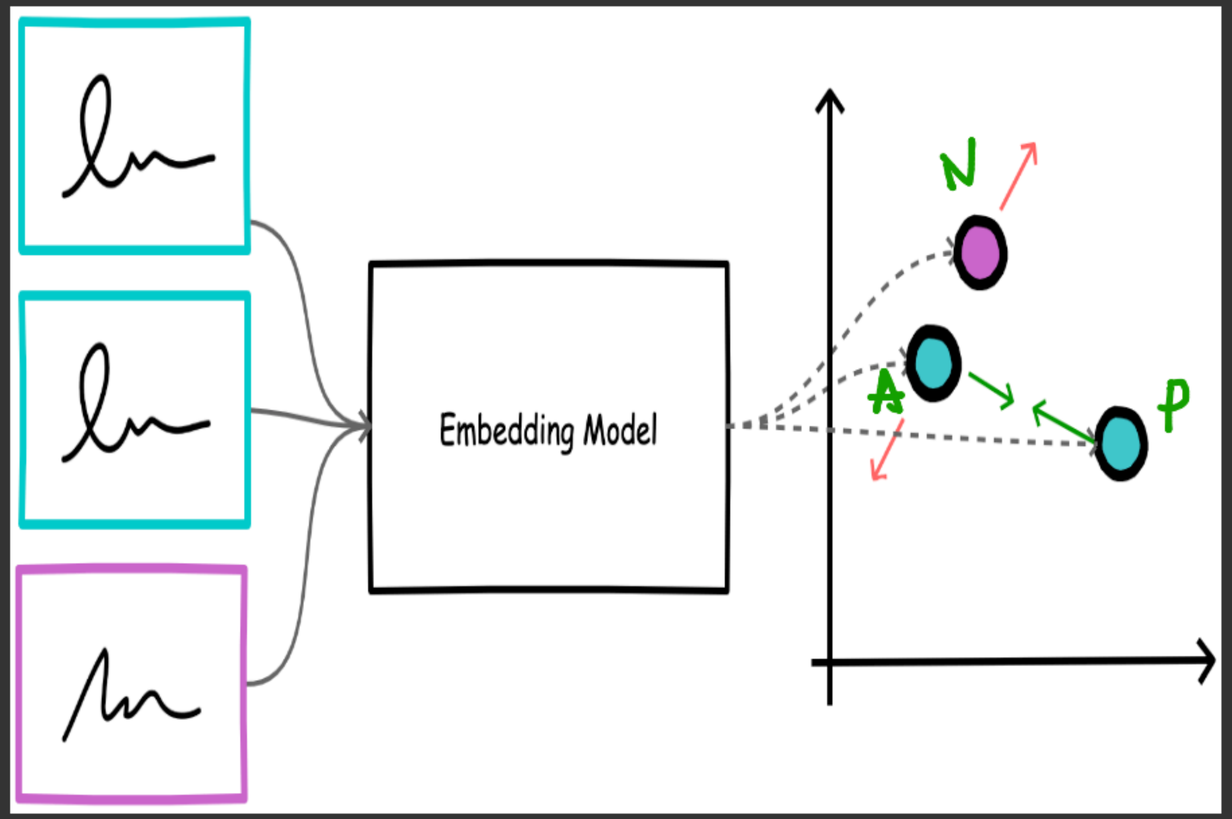
→ More dissimilar pairs

→ Similar pairs

pairs

What is triplet loss?

Contr. loss: imbalance (pairs)



- The triplet loss function is an alternative to the contrastive loss function.
  - https://arxiv.org/pdf/1503.03832.pdf

- **Anchor (A):** The main data point.
- **Positive (P):** A data point similar to Anchor.
- **Negative (N):** A different data point than Anchor.

- The distance between the baseline input and the positive input is reduced to a minimum, while the distance between the baseline input and the negative input is increased.

d(A,P) < d(A,N)

$$(f(A) - f(P))^2 < (f(A) - f(N))^2$$

- In distance function terms, we can say the following:

$$d(A, P) - d(A, N) < 0$$

- As we don't want a Siamese network to learn $f(X) = 0, X \in R$, we will add the margin, similar to a contrastive loss function:

$$d(A, P) - d(A, N) + \alpha < 0$$

- Using the following equations, we will define triplet loss as follows:

$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to b[illegible] through the network.

- **Anchor (A):** The main data point.
- **Positive (P):** A data point similar to Anchor.
- **Negative (N):** A different data point than Anchor.

- The distance between the baseline input and the positive input is reduced to a minimum, while the distance between the baseline input and the negative input is increased.

$$(f(A) - f(P))^2 < (f(A) - f(N))^2$$

- In distance function terms, we can say the following:

$$d(A, P) - d(A, N) < 0$$

- As we don't want a Siamese network to learn $f(X) = 0, X \in R$, we will add the margin, similar to a contrastive loss function:

$$d(A, P) - d(A, N) + \alpha < 0$$

- Using the following equations, we will define triplet loss as follows:

$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to b[illegible]

- **Anchor (A):** The main data point.
- **Positive (P):** A data point similar to Anchor.
- **Negative (N):** A different data point than Anchor.

- The distance between the baseline input and the positive input is reduced to a minimum, while the distance between the baseline input and the negative input is increased.

$$(f(A) - f(P))^2 < (f(A) - f(N))^2$$

- In distance function terms, we can say the following:

$$d(A, P) - d(A, N) < 0$$

- As we don't want a Siamese network to learn $f(X) = 0, X \in R$, we will add the margin, similar to a contrastive loss function:

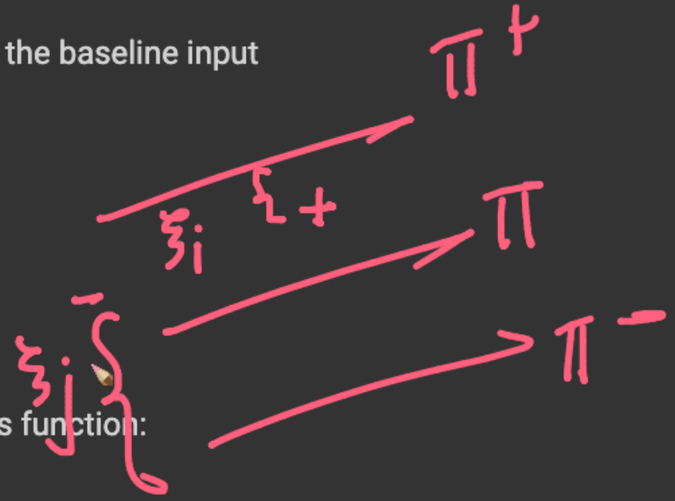
$$d(A, P) - d(A, N) + \alpha < 0$$

- Using the following equations, we will define triplet loss as follows:

$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to b[illegible]



- **Anchor (A):** The main data point.
- **Positive (P):** A data point similar to Anchor.
- **Negative (N):** A different data point than Anchor.

- The distance between the baseline input and the positive input is reduced to a minimum, while the distance between the baseline input and the negative input is increased.

$$(f(A) - f(P))^2 < (f(A) - f(N))^2$$

- In distance function terms, we can say the following:

$$d(A, P) - d(A, N) < 0$$

- As we don't want a Siamese network to learn $f(X) = 0, X \in R$, we will add the margin, similar to a contrastive loss function:

$$d(A, P) - d(A, N) + \alpha < 0$$

- Using the following equations, we will define triplet loss as follows:

$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to b[illegible]

- **Anchor (A):** The main data point.
- **Positive (P):** A data point similar to Anchor.
- **Negative (N):** A different data point than Anchor.

- The distance between the baseline input and the positive input is reduced to a minimum, while the distance between the baseline input and the negative input is increased.

$$(f(A) - f(P))^2 < (f(A) - f(N))^2$$

- In distance function terms, we can say the following:

$$d(A, P) - d(A, N) < 0$$

Max $(0, \xi_i)$

- As we don't want a Siamese network to learn $f(X) = 0, X \in R$, we will add the margin, similar to a contrastive loss function:

$$d(A, P) - d(A, N) + \alpha < 0$$

- Using the following equations, we will define triplet loss as follows:

$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

→ hinge loss

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to b[illegible]

- **Anchor (A):** The main data point.
- **Positive (P):** A data point similar to Anchor.
- **Negative (N):** A different data point than Anchor.

- The distance between the baseline input and the positive input is reduced to a minimum, while the distance between the baseline input and the negative input is increased.

$$(f(A) - f(P))^2 < (f(A) - f(N))^2$$

- In distance function terms, we can say the following:

$$d(A, P) - d(A, N) < 0$$

- As we don't want a Siamese network to learn $f(X) = 0, X \in R$, we will add the margin, similar to a contrastive loss function:

$$d(A, P) - d(A, N) + \alpha < 0$$

- Using the following equations, we will define triplet loss as follows:

$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to b[illegible]

- As we don't want a Siamese network to learn $f(X) = 0, X \in R$, we will add the margin, similar to a contrastive loss function:

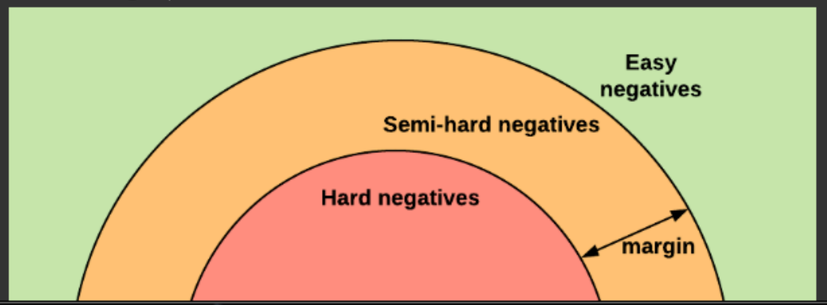
$$d(A, P) - d(A, N) + \alpha < 0$$

- Using the following equations, we will define triplet loss as follows:

$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

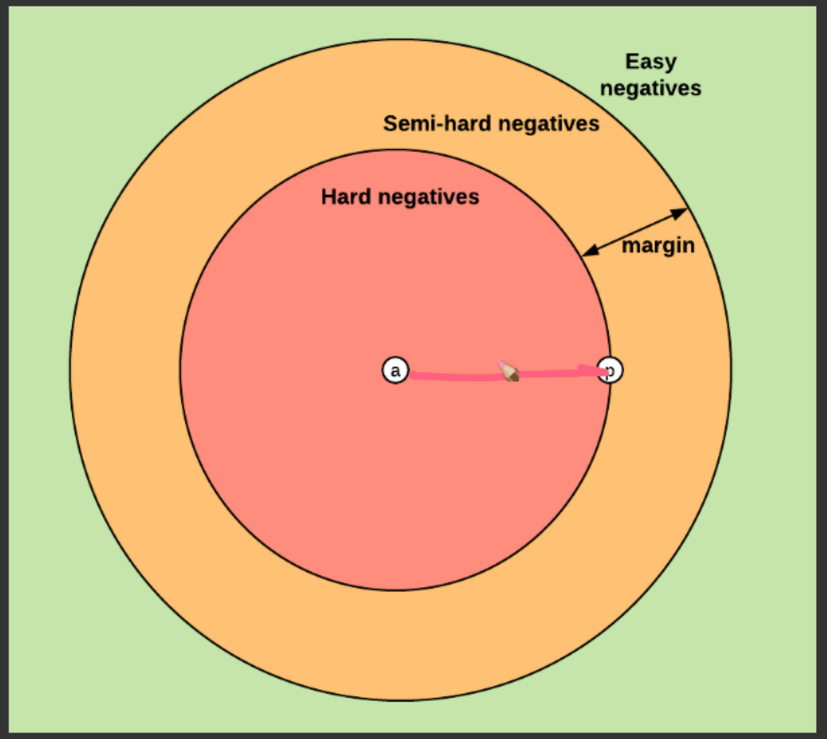
where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to backpropagation through the network.
  - **Semi-hard triplets**: result when $d(A, P) < d(A, N) < d(A, P) + \alpha$. Semi-hard triplets occur when the negative example is more distant to the anchor than the positive example, but the distance is not greater than the margin. This, therefore, results in a positive loss (i.e., the negative is far ... but not far enough.)



is 0, and the network has nothing to learn from.

- **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to backpropagation through the network.
- **Semi-hard triplets**: result when $d(A, P) < d(A, N) < d(A, P) + \alpha$. Semi-hard triplets occur when the negative example is more distant to the anchor than the positive example, but the distance is not greater than the margin. This, therefore, results in a positive loss (i.e., the negative is far ... but not far enough.)

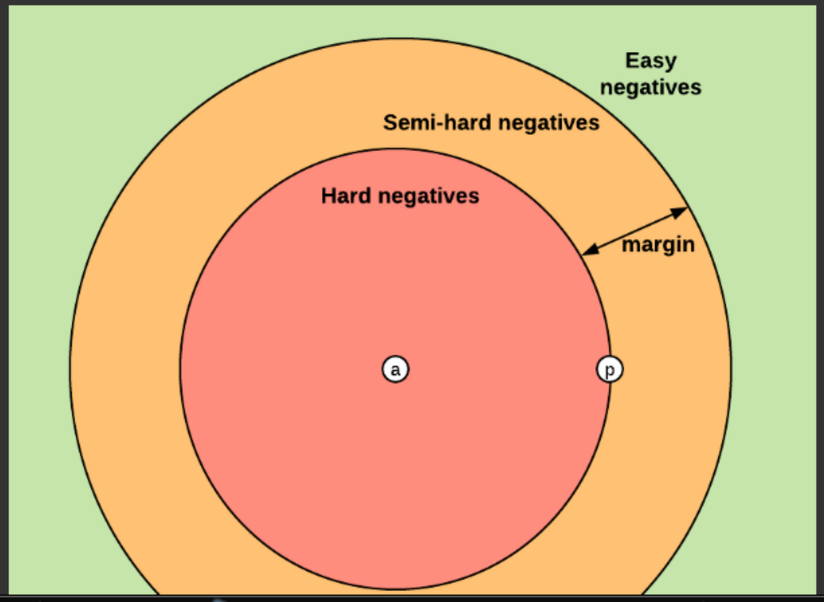


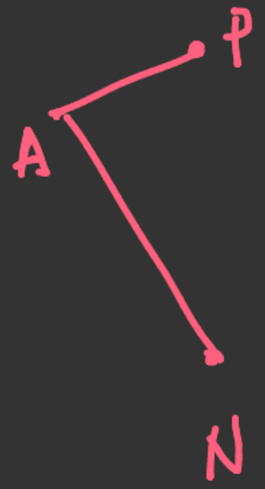
How the loss value changes during the training?

$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to backpropagation through the network.
  - **Semi-hard triplets**: result when $d(A, P) < d(A, N) < d(A, P) + \alpha$. Semi-hard triplets occur when the negative example is more distant to the anchor than the positive example, but the distance is not greater than the margin. This, therefore, results in a positive loss (i.e., the negative is far ... but not far enough.)

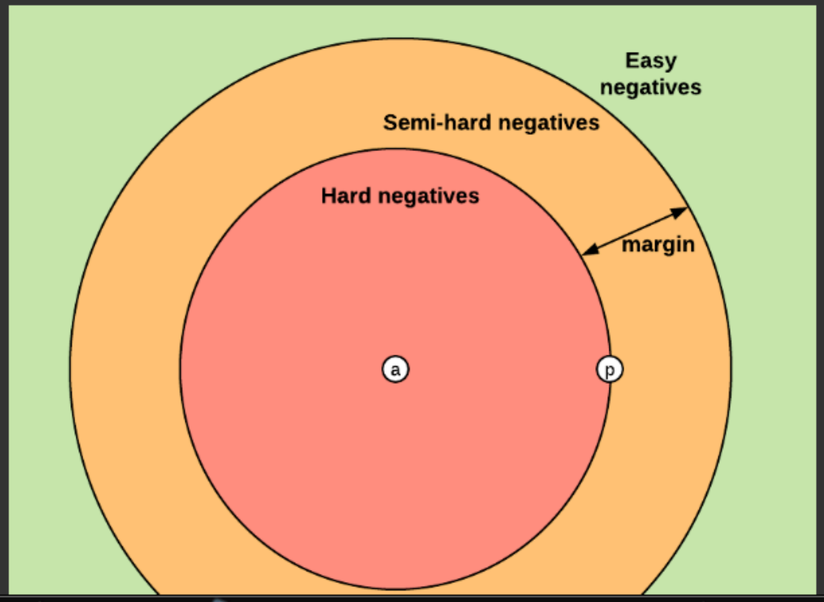




$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

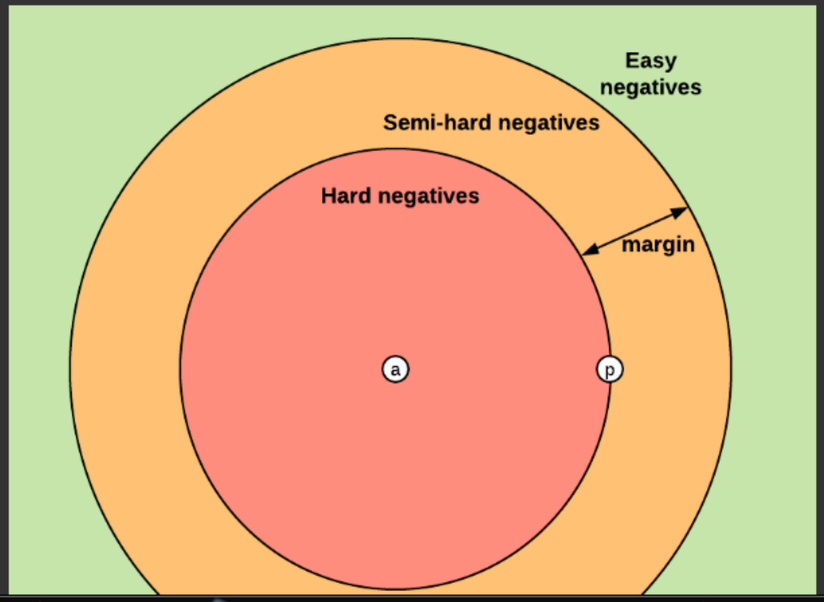
- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to backpropagation through the network.
  - **Semi-hard triplets**: result when $d(A, P) < d(A, N) < d(A, P) + \alpha$. Semi-hard triplets occur when the negative example is more distant to the anchor than the positive example, but the distance is not greater than the margin. This, therefore, results in a positive loss (i.e., the negative is far ... but not far enough.)



$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

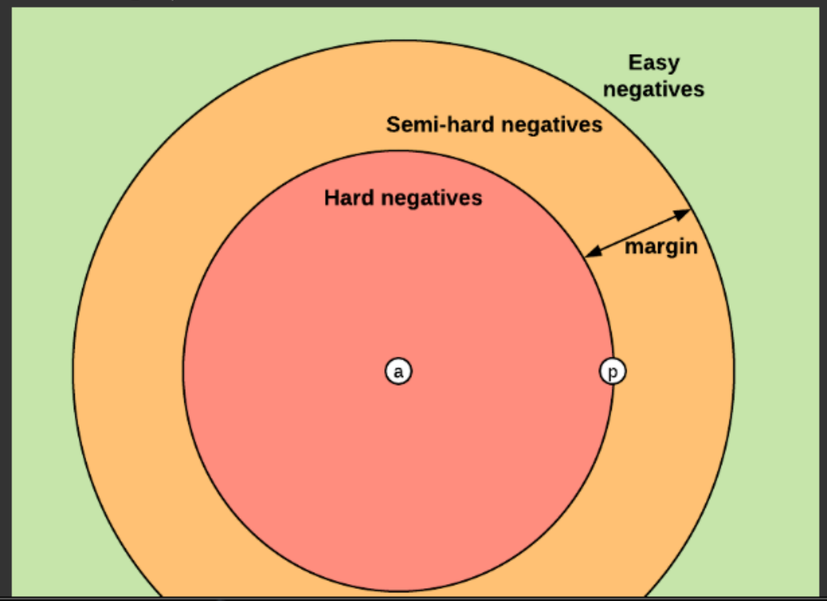
- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to backpropagation through the network.
  - **Semi-hard triplets**: result when $d(A, P) < d(A, N) < d(A, P) + \alpha$. Semi-hard triplets occur when the negative example is more distant to the anchor than the positive example, but the distance is not greater than the margin. This, therefore, results in a positive loss (i.e., the negative is far ... but not far enough.)

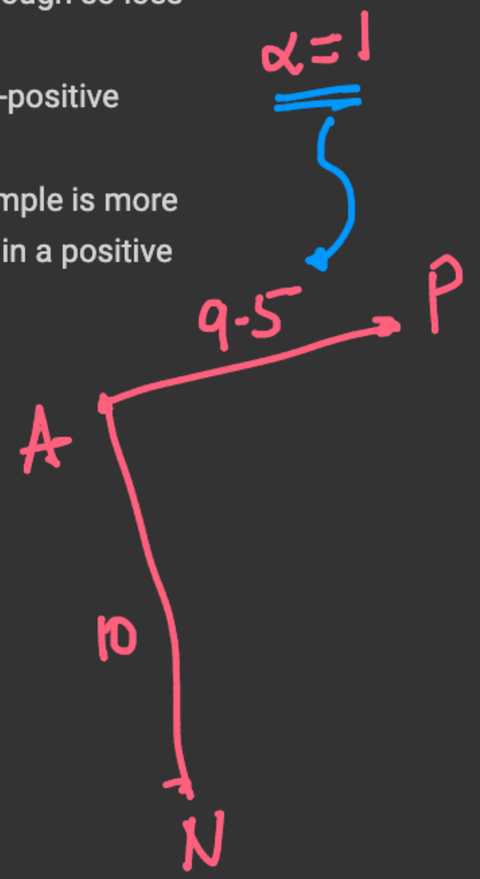


$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to backpropagation through the network.
  - **Semi-hard triplets**: result when $d(A, P) < d(A, N) < d(A, P) + \alpha$. Semi-hard triplets occur when the negative example is more distant to the anchor than the positive example, but the distance is not greater than the margin. This, therefore, results in a positive loss (i.e., the negative is far ... but not far enough.)

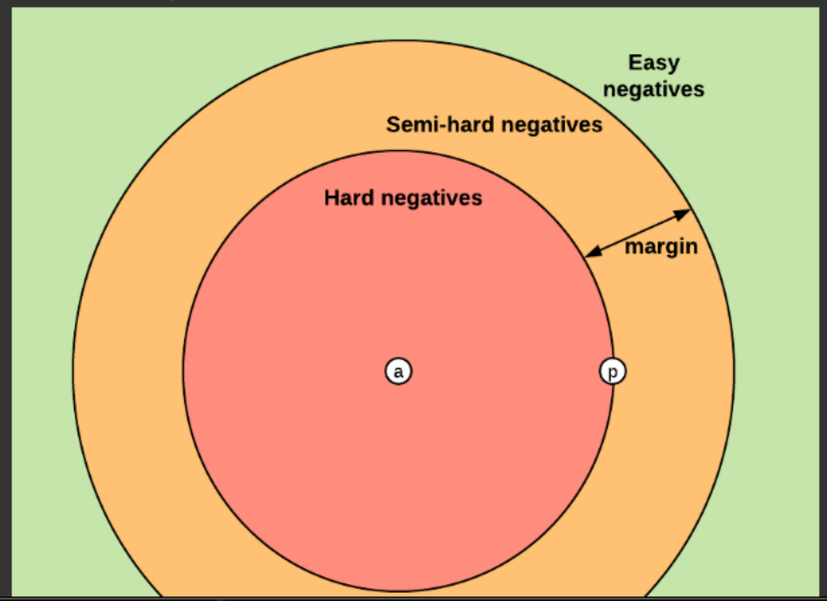




$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to backpropagation through the network.
  - **Semi-hard triplets**: result when $d(A, P) < d(A, N) < d(A, P) + \alpha$. Semi-hard triplets occur when the negative example is more distant to the anchor than the positive example, but the distance is not greater than the margin. This, therefore, results in a positive loss (i.e., the negative is far ... but not far enough.)



$$(f(A) - f(P))^2 < (f(A) - f(N))^2$$

- In distance function terms, we can say the following:

$$d(A, P) - d(A, N) < 0$$

- As we don't want a Siamese network to learn $f(X) = 0, X \in R$, we will add the margin, similar to a contrastive loss function:

$$d(A, P) - d(A, N) + \alpha < 0$$

- Using the following equations, we will define triplet loss as follows:

$$L(A, P, N) = max(0, d(A, P) - d(A, N) + \alpha)$$

where $d(A, P)$ is anchor-to-positive-distance and $d(A, N)$ is anchor-to-negative-distance and α is the margin.

- With this loss formulation, we can create three different types of triplet combinations based on how we sample:
  - **Easy triplets**: result when $d(A, N) > d(A, P) + \alpha$. Here, the sampled anchor-to-negative distance is already large enough so loss is 0, and the network has nothing to learn from.
  - **Hard triplets**: result when $d(A, N) < d(A, P)$. In this case, the anchor-to-negative distance is less than the anchor-to-positive distance, meaning high loss to backpropagation through the network.
  - **Semi-hard triplets**: result when $d(A, P) < d(A, N) < d(A, P) + \alpha$. Semi-hard triplets occur when the negative example is more distant to the anchor than the positive example, but the distance is not greater than the margin. This, therefore, results in a positive loss (i.e., the negative is far ... but not far enough.)

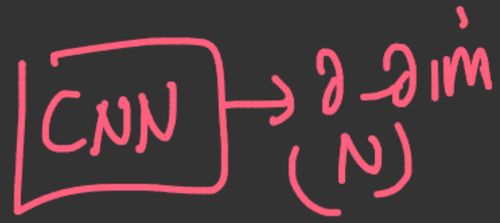
```
# Define the model
model_lossless_triplet_loss = Model(inputs=[anchor_in, pos_in, neg_in], outputs=merged_vector)
```

Now, we have changed the activation function of the last layer of the embedding to **sigmoid activation**

But, the problem with triplet loss is

- loss becomes zero when trained for more epochs

How can we solve this?



- We can create a loss function which breaks the linearity in cost function.
- In other words, make it really costly as more the error grows.

Here comes the **lossless triplet loss**

It is defined as

$$\sum_{i=1}^{n}[-ln(-\frac{(f_i^a-f_i^p)^2}{\beta})+1+\epsilon)-ln(-\frac{N-(f_i^a-f_i^n)^2}{\beta})+1+\epsilon)]$$

- Where N is the number of dimensions (Number of output of your network; Number of features for your embedding)
- β is a scaling factor.
- ϵ is the margin
- $(f_i^a - f_i^p)^2$ is the distance between anchor and positive
- $(f_i^a - f_i^n)^2$ is the distance between anchor and negative

```
[ ] # Custom Function for lossless triplet loss
    def lossless_triplet_loss(y_true, y_pred, beta=3, epsilon=1e-8):
```

```
# Define the model
model_lossless_triplet_loss = Model(inputs=[anchor_in, pos_in, neg_in], outputs=merged_vector)
```

Now, we have changed the activation function of the last layer of the embedding to **sigmoid activation**

But, the problem with triplet loss is

- loss becomes zero when trained for more epochs

How can we solve this?

- We can create a loss function which breaks the linearity in cost function.
- In other words, make it really costly as more the error grows.

Here comes the **lossless triplet loss**

It is defined as

$$\sum_{i=1}^{n}[-ln(-\frac{(f_i^a-f_i^p)^2}{\beta})+1+\epsilon)-ln(-\frac{N-(f_i^a-f_i^n)^2}{\beta})+1+\epsilon)]$$

- Where N is the number of dimensions (Number of output of your network; Number of features for your embedding)
- β is a scaling factor.
- ϵ is the margin
- $(f_i^a - f_i^p)^2$ is the distance between anchor and positive
- $(f_i^a - f_i^n)^2$ is the distance between anchor and negative

Haa Triplet loss (L)

$d(A,P) < d(A,N)+\alpha$

```
[ ] # Custom Function for lossless triplet loss
    def lossless_triplet_loss(y_true, y_pred, beta=3, epsilon=1e-8):
```

```
# Define the model
model_lossless_triplet_loss = Model(inputs=[anchor_in, pos_in, neg_in], outputs=merged_vector)
```

Now, we have changed the activation function of the last layer of the embedding to **sigmoid activation**

But, the problem with triplet loss is

- loss becomes zero when trained for more epochs

How can we solve this?

- We can create a loss function which breaks the linearity in cost function.
- In other words, make it really costly as more the error grows.

Here comes the **lossless triplet loss**

It is defined as

$$\sum_{i=1}^{n}[-ln(-\frac{(f_i^a-f_i^p)^2}{\beta})+1+\epsilon)-ln(-\frac{N-(f_i^a-f_i^n)^2}{\beta})+1+\epsilon)]$$

- Where N is the number of dimensions (Number of output of your network; Number of features for your embedding)
- β is a scaling factor.
- ϵ is the margin
- $(f_i^a - f_i^p)^2$ is the distance between anchor and positive
- $(f_i^a - f_i^n)^2$ is the distance between anchor and negative

```
[ ] # Custom Function for lossless triplet loss
    def lossless_triplet_loss(y_true, y_pred, beta=3, epsilon=1e-8):
```

Recap:

Few shot

→ Siamese n/w

- → BCE → pairs
- → Contr. loss → pairs
- → Triplet loss → Hard(α)
  - ↳ hard + Semi hard
  - → soft (ε)

→ Dataset generation through Combinatorial explosion

Practical

CUDA ← Nvidia

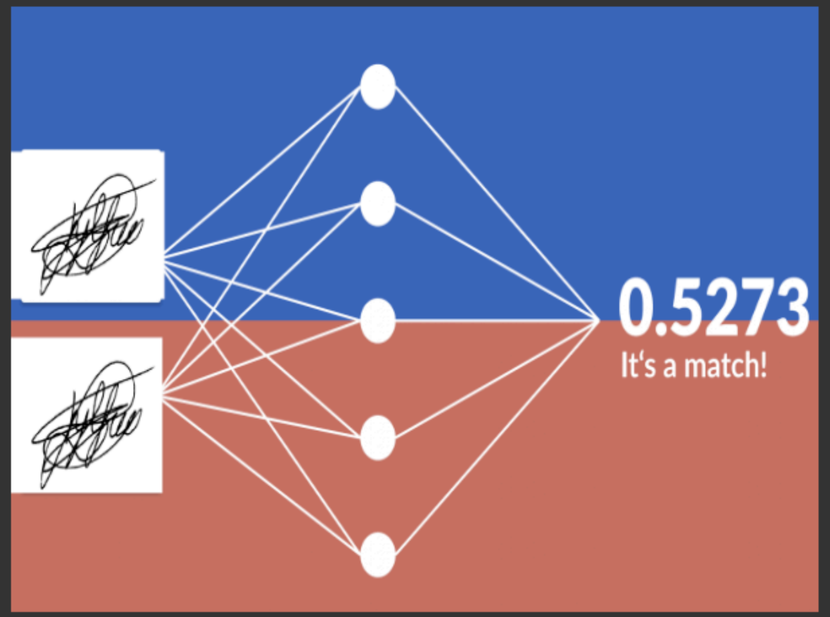
Tensorflow / PyTorch / Keras ...

CUDA (C++)

Nvidia GPUs

Problem Statement

- You are working as Data scientist in HDFC bank.
    - You have a new account holder in your bank and would like to set his signature for verification.
- You have only one sample signature from the member.
- Being a DataScientist , How can you use a neural network to perform the signature verification?



TASK: Verify whether a Signature of a person is Genuine or Forged