CAB420: Comparing Things with Deep Neural Networks

THING A VS THING B

Comparing Things with Machine Learning

- Often in machine learning we want to know how similar two things are
- Two main uses for comparisons
  - Verification
  - Identification
- The face recognition examples from our look at PCA and LDA are good examples
  - Are these two people the same?

Verification

- Verification, seeks to answer the question “are these two things the same?”
  - Example: SmartGate at Australian Airports
    - You claim an identity via your passport
    - The system then attempts to verify that it’s you
  - Usual approach
    - Compute distance between claimed identity and observed identity
    - Apply a threshold to decide if samples are the same

Identification

- Identification, seeks to answer the question “which thing is this?”
  - Based on having a gallery of previously seen “things”, and comparing the new “thing” to all these other “things”
  - Typically returns a ranked list, i.e. the N most similar “things”
- Usual approach
  - Compute distance between new “thing” and all other previously seen “things”
  - Return a ranked list based on distance

A Simple Approach

- Telling if two things are the same can be viewed as a classification task
  - Requires us to know all classes in advance
  - Then we train a classifier to split them
  - This is what we’ve done for face rec with PCA and/or LDA and a CKNN
- However
  - What happens when a new class of things appears?
    - Open world scenario: we don’t know what all the “things” are when we train the network
    - In this case, we weed to retrain the lot
      - That gets impractical quickly

Open World Approaches

- What we want to do is
  - Computes a subspace where
    - Things of the same class are close
    - Things of different classes are far away
  - To compare two things, we
    - Project them into the subspace
    - Compute the distance between them
- This method scales and allows us to add new “classes” after training
  - Assuming the data is of the same broad type/format, etc

Replicating this with DCNNs

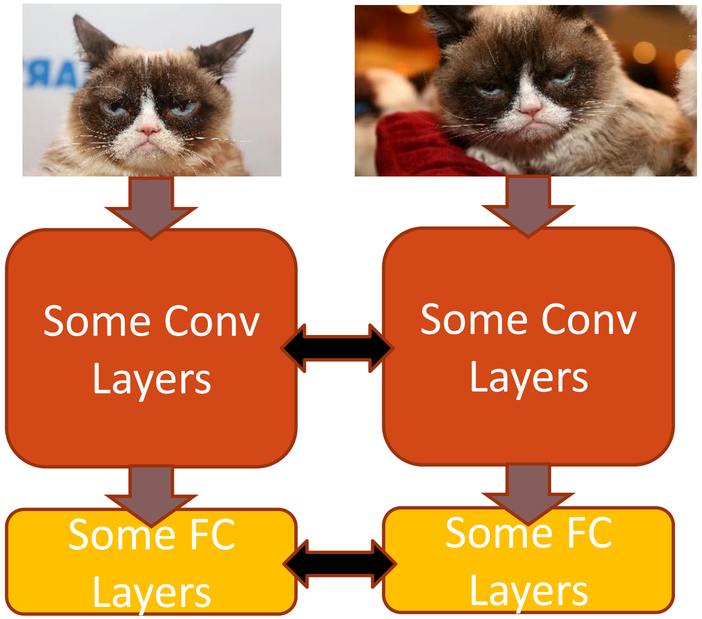
- We need to
  - Train a model that can learn a subspace where we can compare things
  - This should, given a pair of images extract a compact representation that
    - Has things of the same class close to each other
    - Has things of a different class far away from each other
    - The distance between should, ideally, reflect similarity

CAB420: Siamese Networks

COMPUTER VISION AND CATS

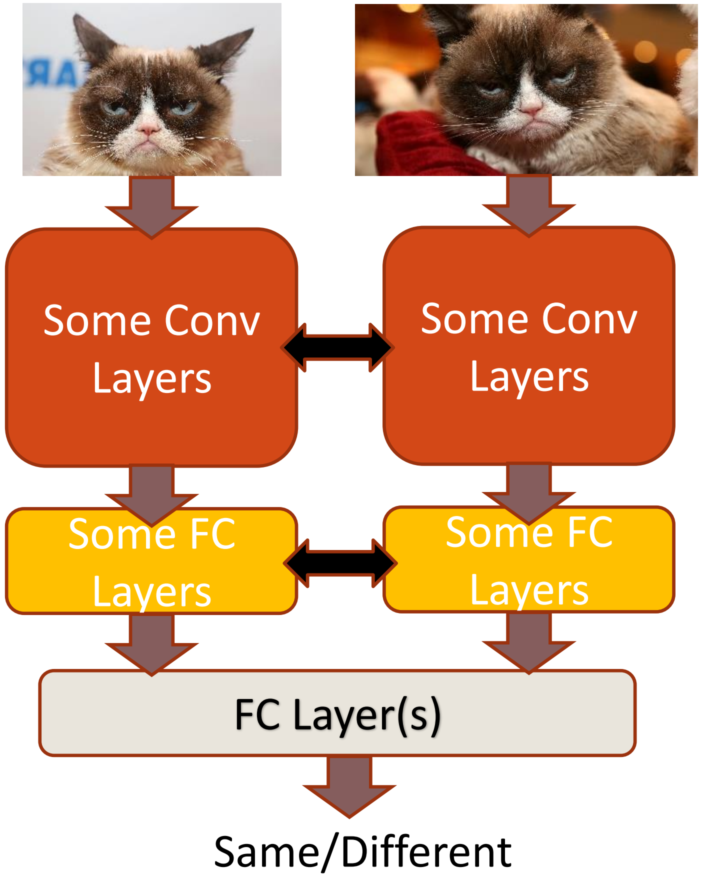
Siamese Networks

- We have two streams
  - Each is identical
    - Same structure, weights, etc.
  - For the same image, each stream will extract the same features
  - For images of the same class we'll get similar features
    - Hopefully



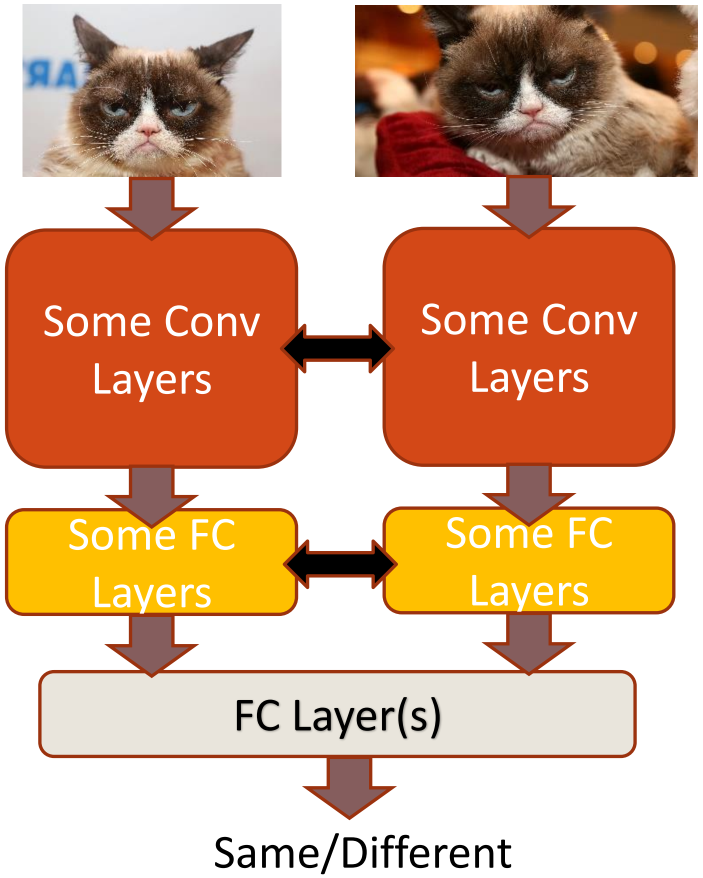
Siamese Networks

- The output of these streams is a compact representation
  - We'll call this an embedding
  - We can compare these embeddings to tell if things are the same, or different
    - May be compared by another small network



Siamese Networks

- The main stream of the network
  - Can be anything
    - VGG
    - ResNet
    - Something else
  - The embedding
    - Can be whatever size you want
    - Larger embeddings are potentially more descriptive



Comparing Things with Siamese Networks

NEURAL NETWORKS AND CATS – A MATCH MADE IN INTERNET HEAVEN

Comparing Embeddings - Naïve Solution

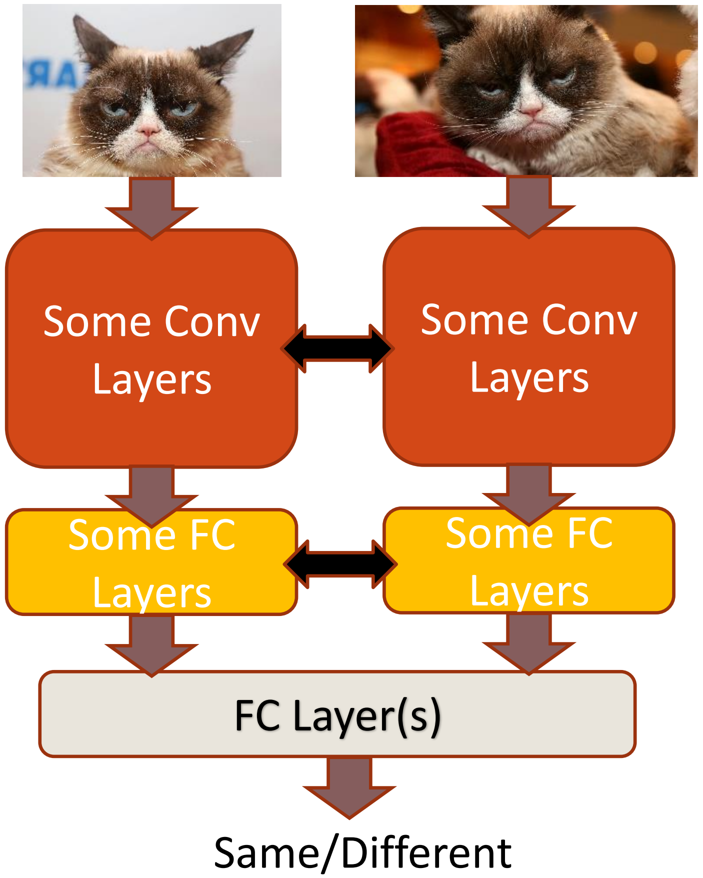
- We can view this as a binary classification task
  - A pair of images is either
    - Of the same type
    - Of a different type
  - Therefore
    - Binary Cross Entropy

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\bar{y}_i) + (1 - y_i)\log(1 - \bar{y}_i)$$

    - y_i = true probability, 0 or 1
    - $\bar{y}_i$ = estimated probability
    - N = batch size

Comparing Embeddings - Naïve Solution

- What this means is
  - We take our two embeddings
  - We input them into another network with a single output
    - An output of 1 indicates the same class
    - An output of 0 indicates different classes

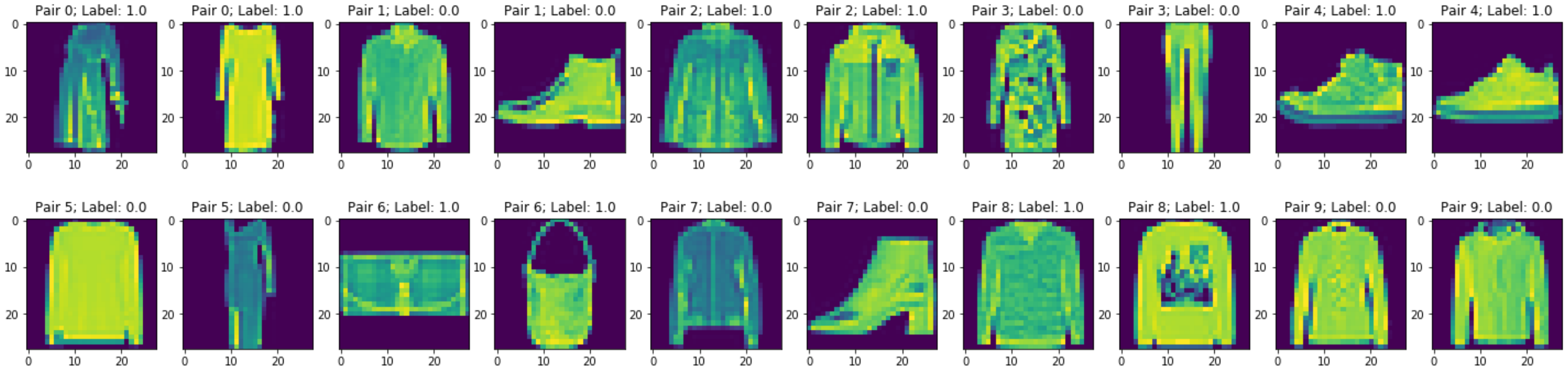


Data

- We need pairs of images
- We can generate this from a labelled dataset
  - i.e. we have class identities
- Randomly select a sample
  - For a positive pair, randomly select another sample of the same class
  - For a negative pair, randomly select another sample of a different class

An Example

- See ***CAB420_Metric_Learning_Example_1_Siamese_Networks.ipynb***
- Data
  - Fashion MNIST
  - Random pairs create, 50% positive (same class), 50% negative (different class)



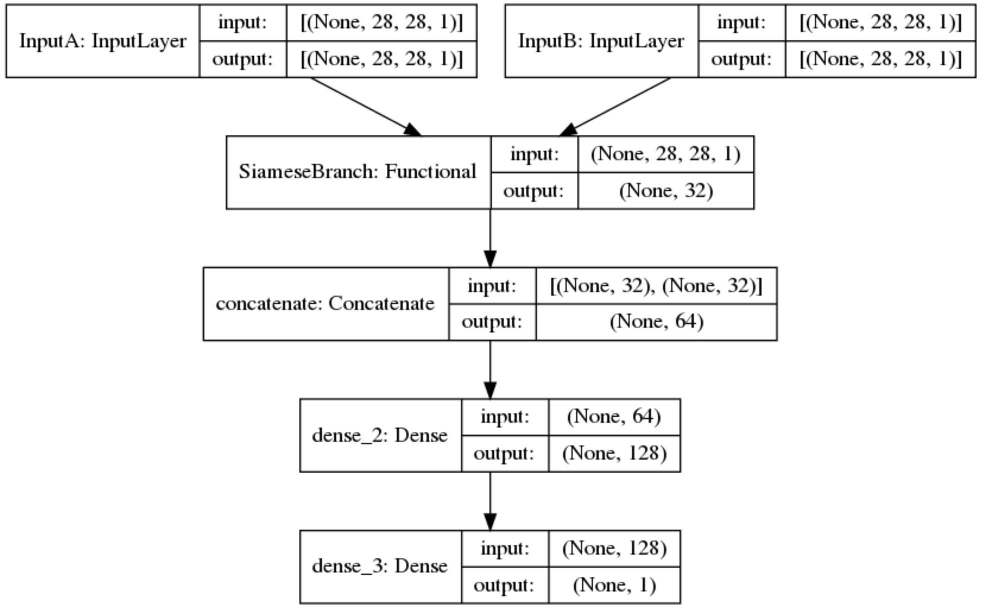
Backbone Network

- The backbone is the network that processes the two input images prior to comparing them
  - VGG Style Network
  - 3 pairs of two 2D Convolution layers
- Final dense layer of size 32
  - The embedding
- We could use a pre-trained network here and fine-tune

```
Model: "SiameseBranch"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0
conv2d_24 (Conv2D)           (None, 28, 28, 8)         80
conv2d_25 (Conv2D)           (None, 28, 28, 8)         584
batch_normalization_13 (Batc (None, 28, 28, 8)         32
activation_4 (Activation)    (None, 28, 28, 8)         0
spatial_dropout2d_12 (Spatia (None, 28, 28, 8)         0
max_pooling2d_8 (MaxPooling2 (None, 14, 14, 8)         0
conv2d_26 (Conv2D)           (None, 14, 14, 16)        1168
conv2d_27 (Conv2D)           (None, 14, 14, 16)        2320
batch_normalization_14 (Batc (None, 14, 14, 16)        64
activation_5 (Activation)    (None, 14, 14, 16)        0
spatial_dropout2d_13 (Spatia (None, 14, 14, 16)        0
max_pooling2d_9 (MaxPooling2 (None, 7, 7, 16)          0
conv2d_28 (Conv2D)           (None, 7, 7, 32)          4640
conv2d_29 (Conv2D)           (None, 7, 7, 32)          9248
batch_normalization_15 (Batc (None, 7, 7, 32)          128
activation_6 (Activation)    (None, 7, 7, 32)          0
spatial_dropout2d_14 (Spatia (None, 7, 7, 32)          0
flatten_4 (Flatten)          (None, 1568)              0
dense_11 (Dense)             (None, 256)               401664
batch_normalization_16 (Batc (None, 256)               1024
activation_7 (Activation)    (None, 256)               0
dense_12 (Dense)             (None, 32)                8224
=================================================================
Total params: 429,176
Trainable params: 428,552
Non-trainable params: 624
_______________________________________
```

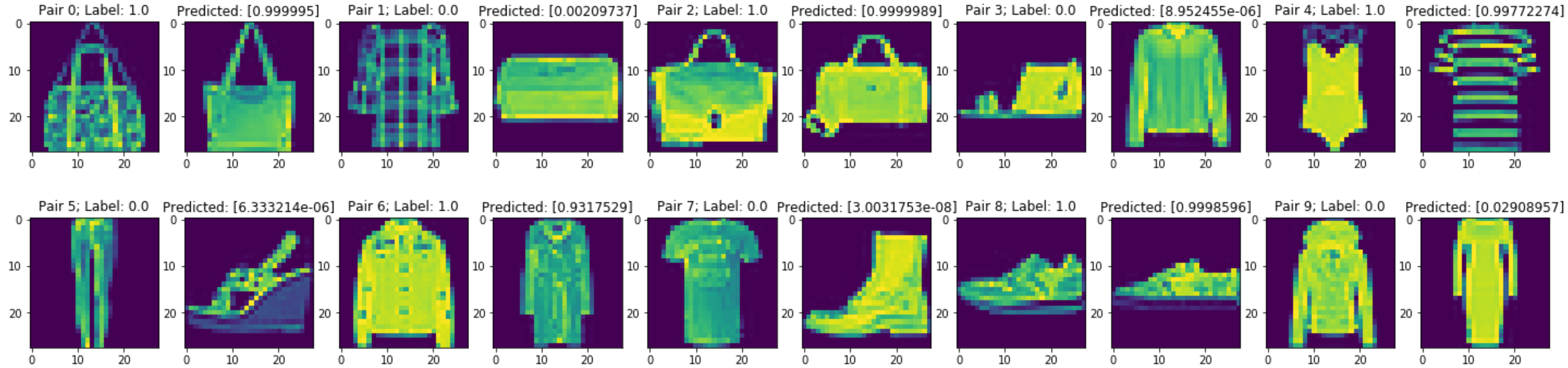
Siamese Network

- Pass two images through the same backbone
- Concatenate the output
- Pass the concatenated result through one or more addition dense layers
- Final layer of size 1:
  - True/False to indicate if images are the same or different classes



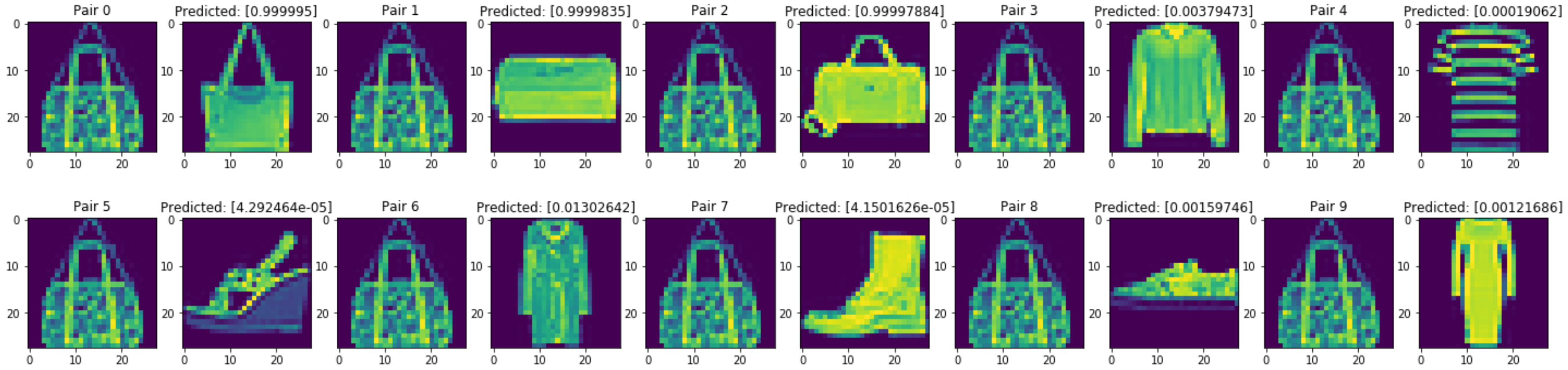
Some Predictions

- All show correct results
  - Values close to 1 for postive (same) pairs
  - Values close to 0 for negative (different) pairs



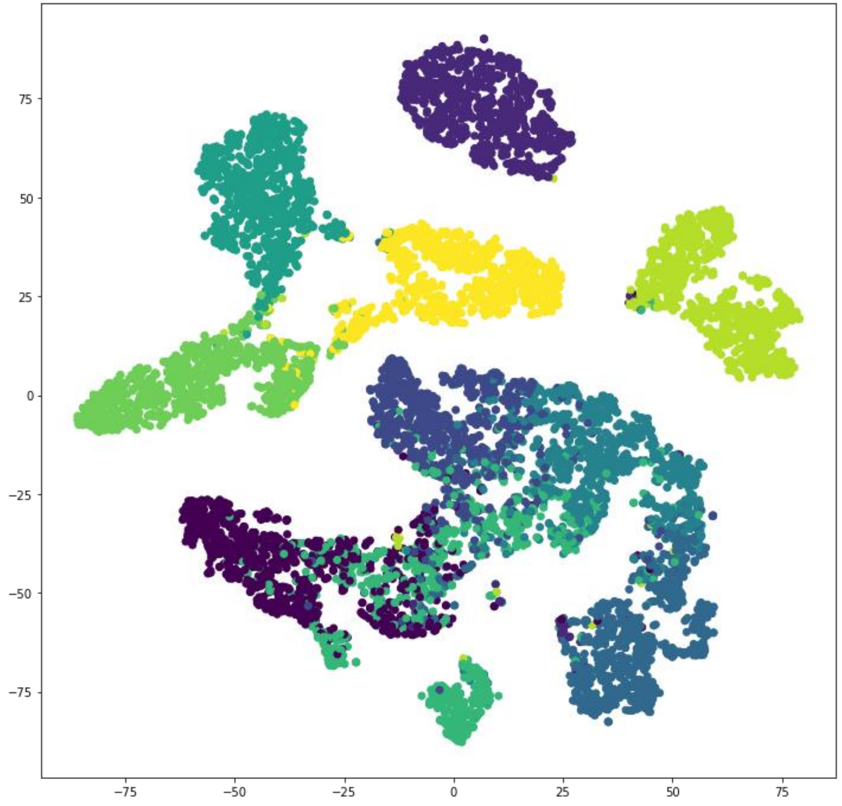
Some Predictions

- Same query image in all cases
  - Correct labels are predicted in all cases
  - Little, if any, meaning in the actual scores beyond same/different



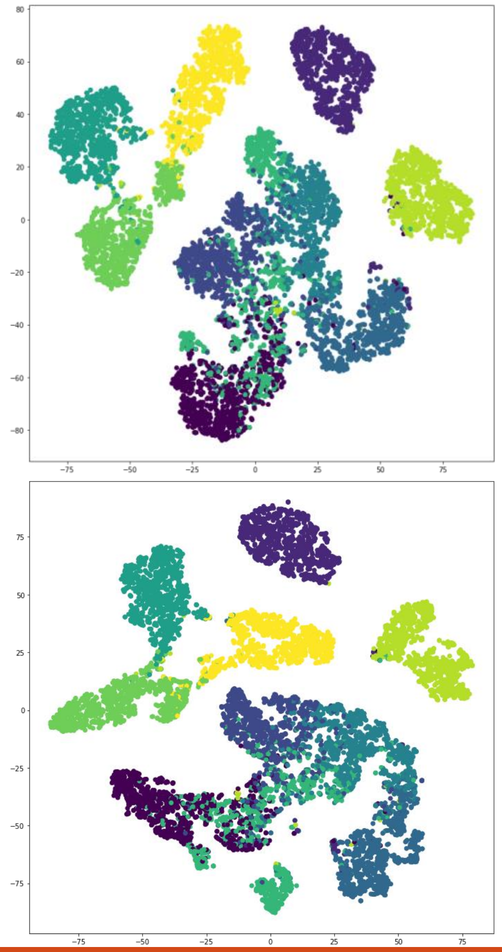
What did the network learn?

- Visualise embeddings using t-SNE
- Pass entire test set through the backbone network
  - Extract embeddings
- Plot embeddings in 2D using t-SNE
  - Some classes are well separated
  - Others, not so much



Comparing with a regular classification network

- Take our same backbone, train for classification using categorical cross entropy
  - Extact embeddings (size 32) and plot with t-SNE
- Top: DCNN + Categorical Cross Entropy
- Bottom: Siamese
- Both networks perform simiarly
  - Slightly cleaner class boundaries with the Siamese network
  - To be expected, both networks have the same structure and a very similar loss



CAB420: A Better Loss (Contrastive Loss)

POOR PERFORMANCE MAKES ME GRUMPY

What’s Wrong with Binary Cross Entropy?

- With our first approach we can determine if two items are the same or not by applying a threshold
- But…
  - The loss (BCE) does not force similar images to have similar features
  - We rely on another network to do this
  - This limits our ability to “rank” things

Contrastive Loss

- What we'd like is to try to replicate LDA
  - Have examples of the same class close to each other
  - Have examples of different classes far from each other
- Contrastive Loss

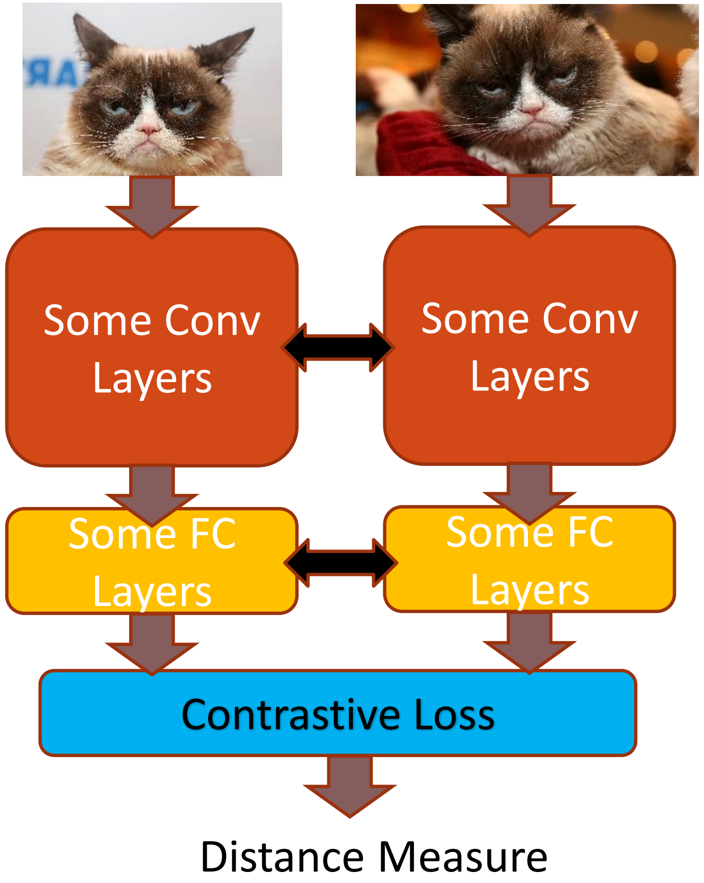
$$L = \frac{1}{N}\sum_{i=1}^{N} y_i d_i + (1 - y_i)\max(margin\ - d_i, 0)$$

$$d_i = |\bar{y}_1 - \bar{y}_2|_2$$

  - note that we can use a different distance metric here, such as cosine or L1
  - y_i is the pair label; 1 indicates the pair is the same class; 0 indicates they are different
  - Contrastive loss aims to separate positive and negative pairs of embeddings by a margin

Contrastive Loss Network

- What this means is
  - We take our two embeddings
  - We compute the loss directly on them
    - No subnetwork needed



Contrastive Loss Margin

- What should it be?
  - Depends on the range of the magnitude of the embedding
    - Bigger embedding, potentially bigger margin
  - Could change it dynamically
    - As the network learns, make it bigger
  - Or, normalise the embedding
    - Embeddings now have a constant magnitude, regardless of size
    - Can then set the margin to 1 and regardless of embedding size, this will work
    - We’ll do this

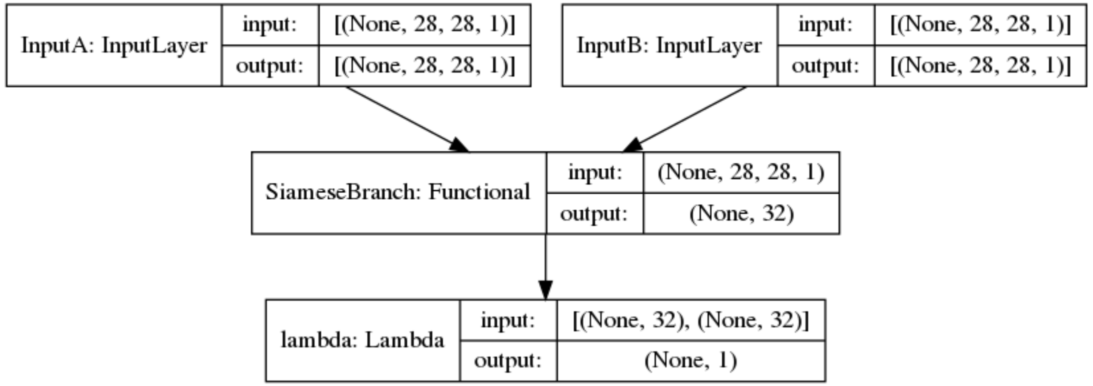
An Example

- See ***CAB420_Metric_Learning_Example_2_Contrastive_Loss.ipynb***

- Data
  - Same as Example 1

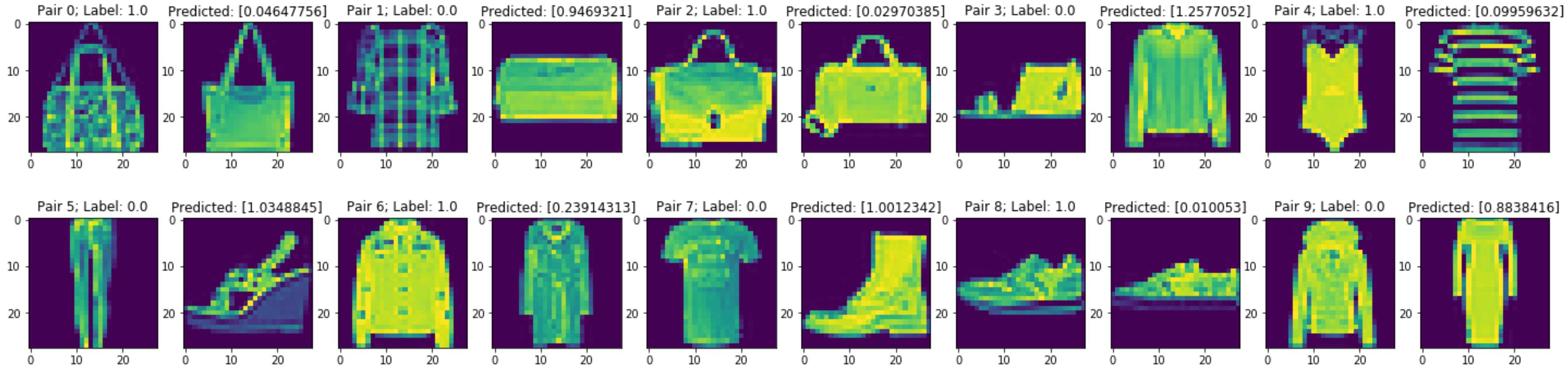
The Network

- Very similar to Example 1
  - Same backbone
  - No feature concatenation and dense layer(s)
  - Loss is computed directly on the two embeddings from the backbone
    - Simpler network on the whole



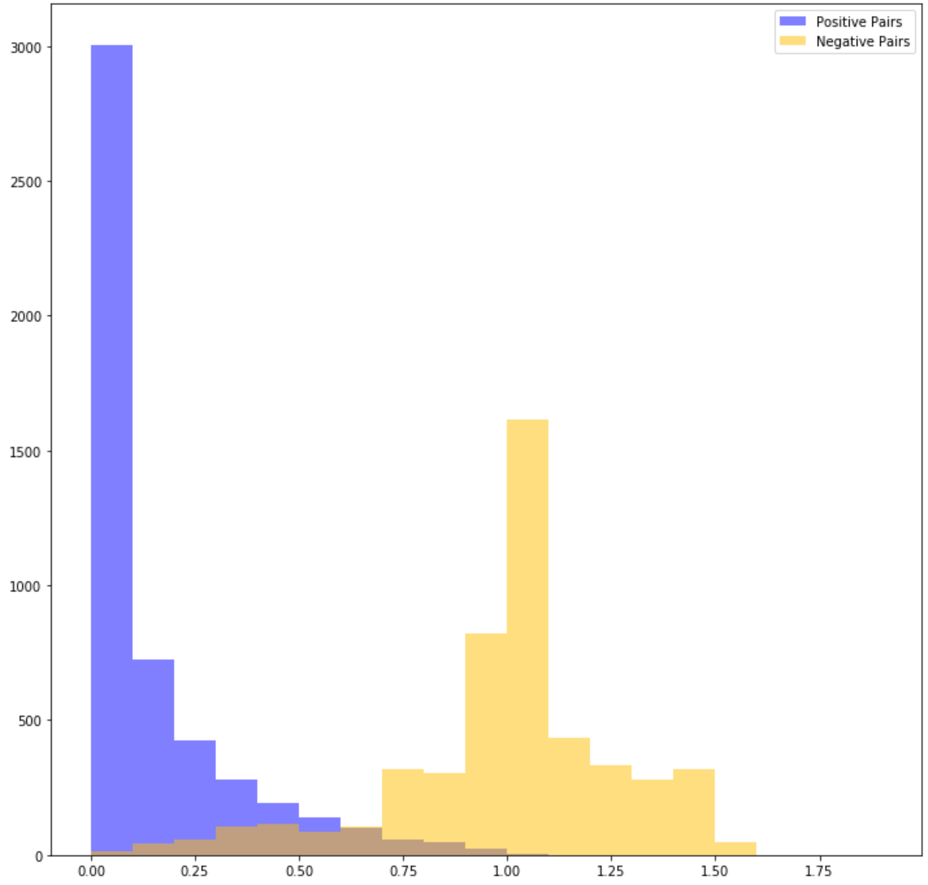
Some Predictions

- Items that are the same (label 1) should have a small distance
  - Similar items have distances less than 0.5
  - Dissimilar items have distances greater than 0.5



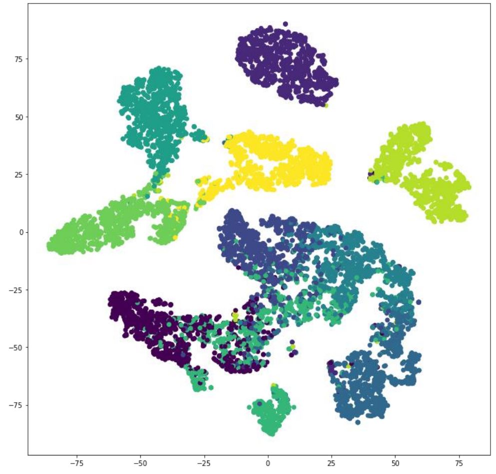
Distribution of Distances

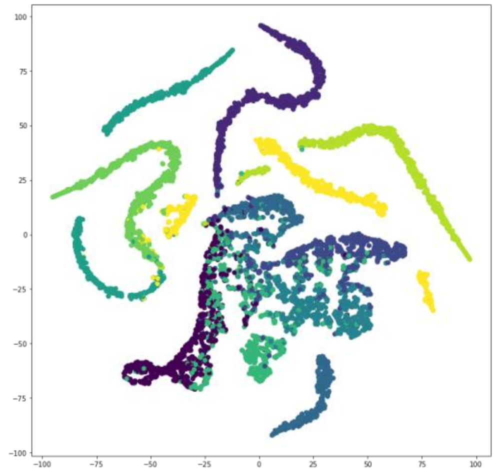
- This plot shows
  - A histogram of distances for matched pairs
  - A histogram of distances for mismatched pairs
- We have trained the network to have items of the same class closer together in feature space
  - This has, mostly, been achieved
  - Some overlap in distributions
    - Would lead to errors when making decisions



Embedding Space

- t-SNE plots of the learned embedding
  - Top: Binary Cross Entropy
  - Bottom: Contrastive loss
- Contrastive loss leads to much clearer class boundaries
  - Still some confusion
  - Some classes are broken into multiple clusters
    - Possibly capturing different semantic details
    - Different types of shoes perhaps?



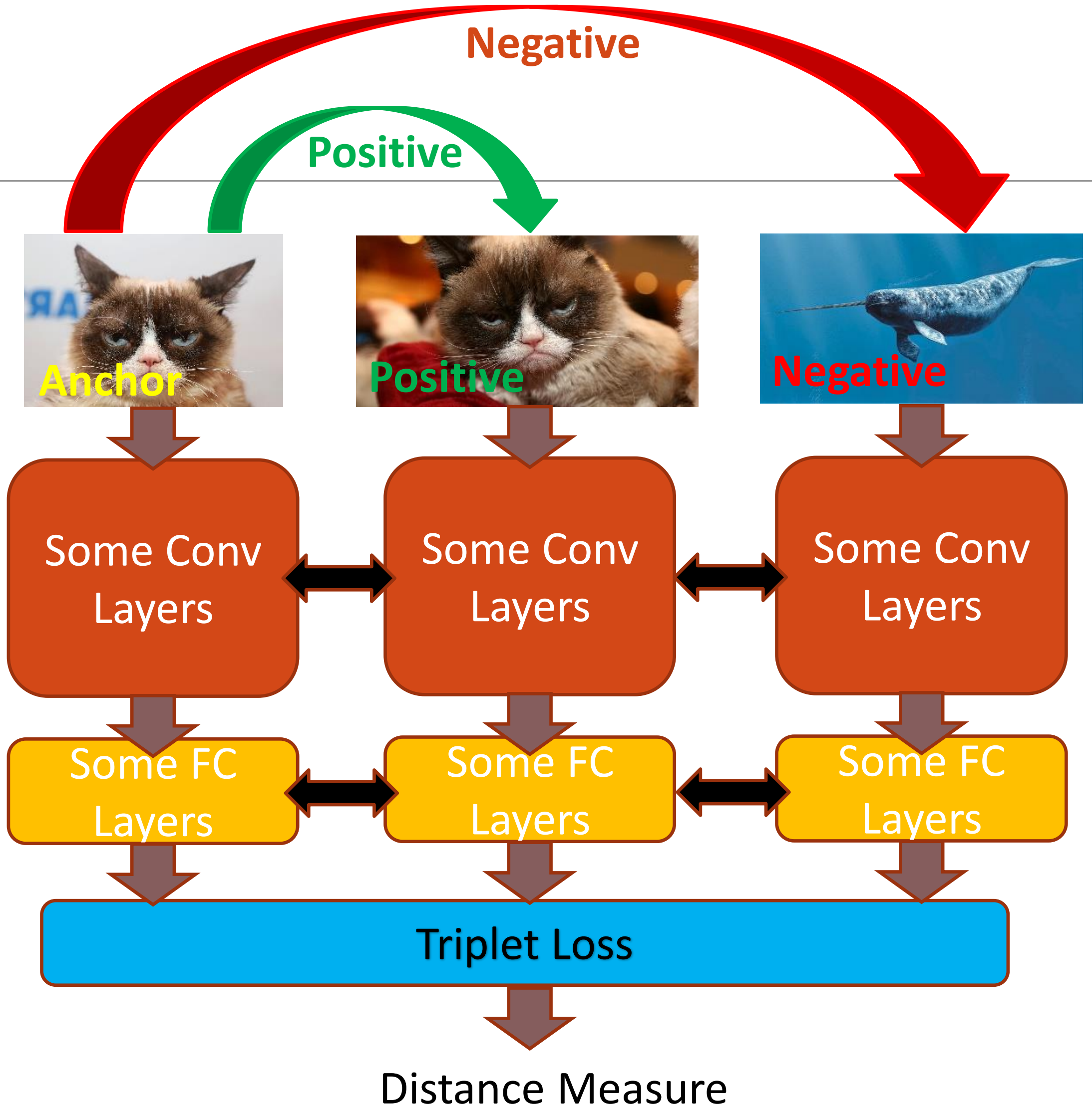


CAB420: Because Two’s Not Enough

TRIPLET NETWORKS

Triplets

- Three images
  - Anchor
  - Positive
  - Negative
- Two pairs
  - Positive Pair
  - Negative Pair
- Pull the positive pair embeddings close
- Push the negative pair embeddings far away



Triplet Loss

$$L = \max(d(a,p) - d(a,n) + margin, 0)$$

- $d(a,p)$, distance between anchor and positive
- $d(a,n)$, distance between anchor and negative
- As usual we can use whatever distance function we like

- The margin
  - Serves much the same role as it does in the contrastive loss
  - Again, if we normalise vectors we can fix this at 1

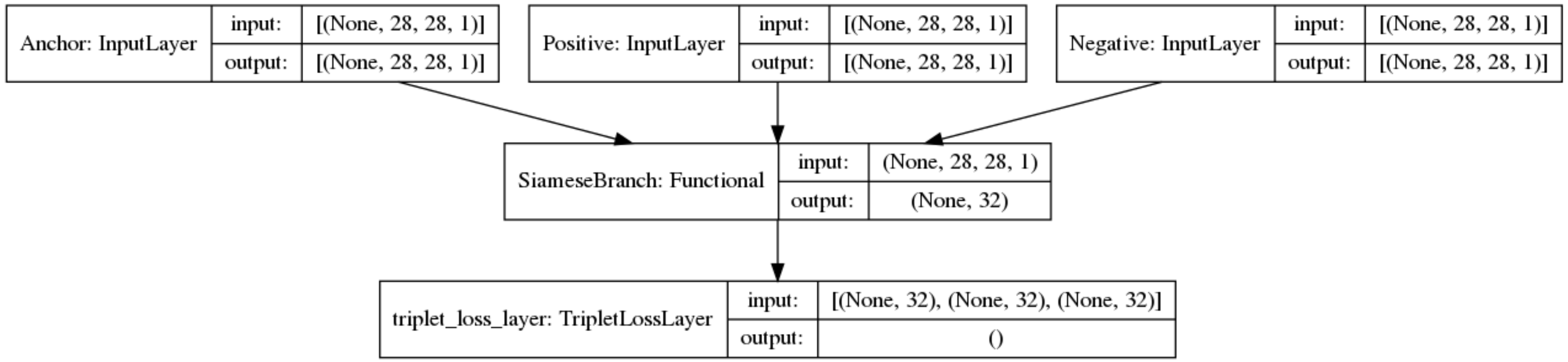
An Example

- See ***CAB420_Metric_Learning_Example_3_Triplet_Loss.ipynb***

- Data
  - Same as first two example, except we now have triplets
  - Each sample has
    - A random **anchor** image
    - A random **positive** image, the same class as the anchor
    - A random **negative** image, of a different class to the anchor

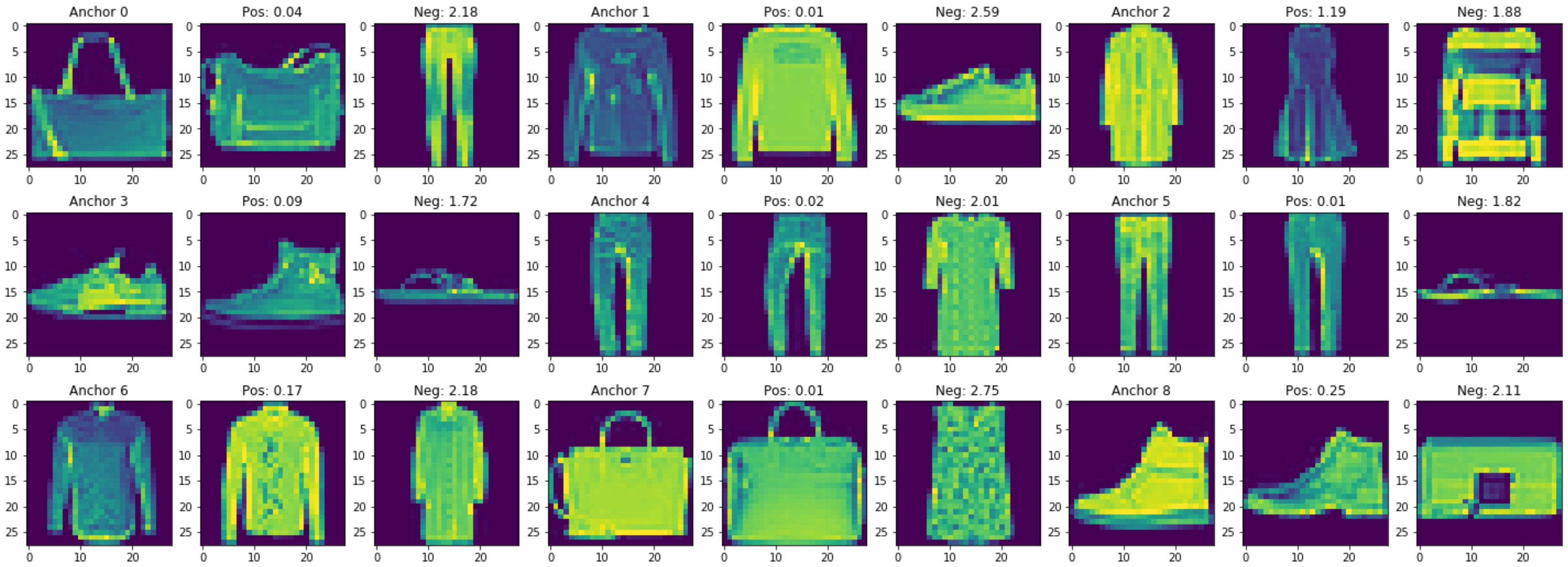
The Network

- Same backbone as Examples 1 and 2
- Three inputs
- Triplet loss
  - As per contrastive loss, no additional layers beyond the backbone
  - Loss computed directly on the embeddings



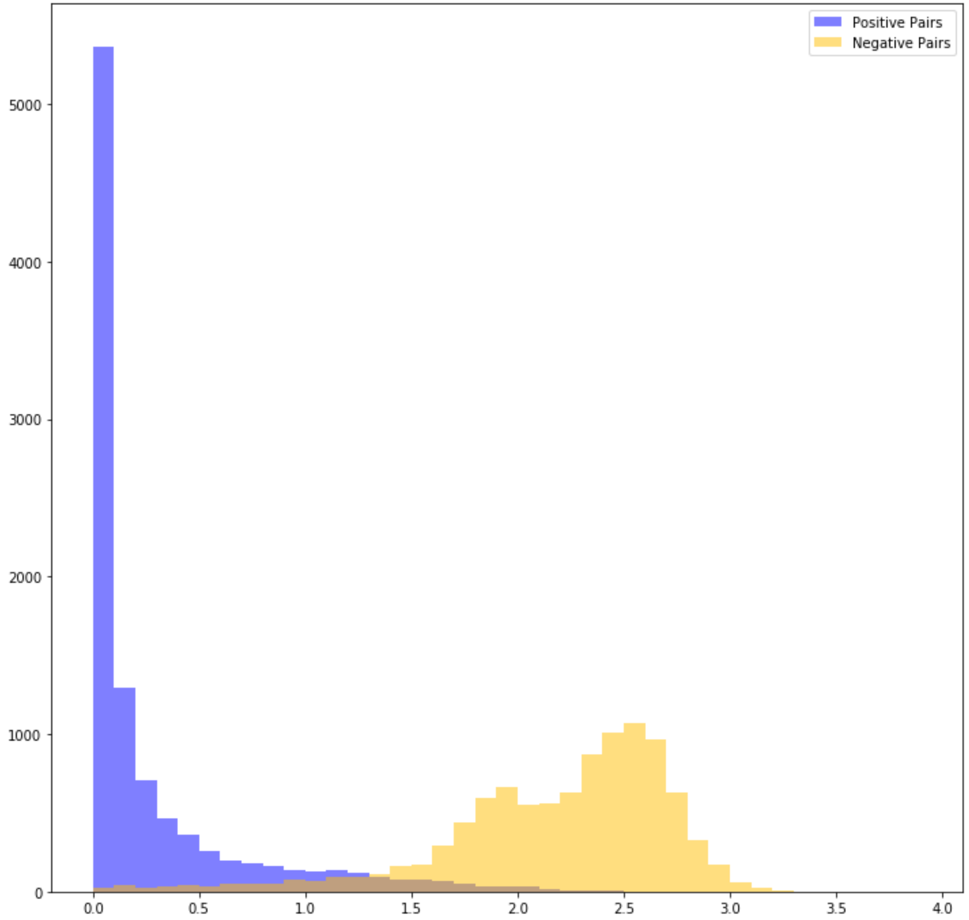
Some Predictions

- Visualising results for triplets
  - Negative images much further from the anchor than positive images



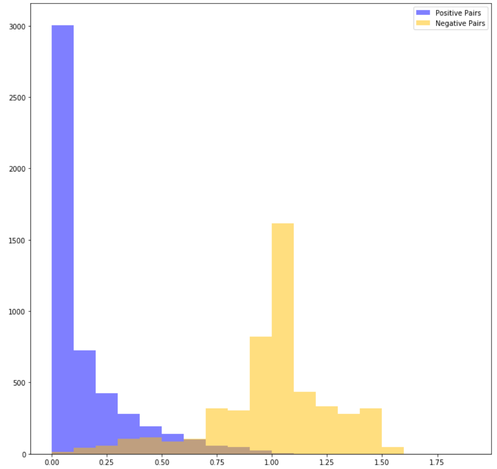
Distribution of Distances

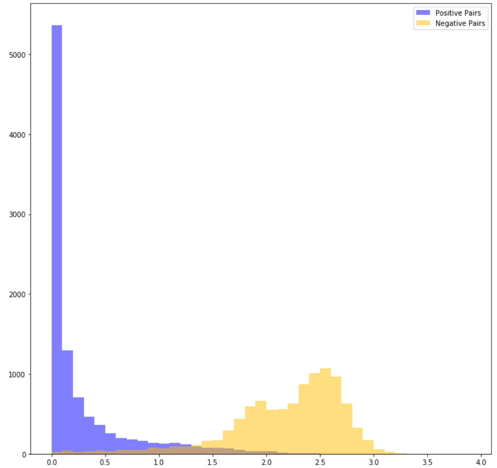
- This plot shows
  - A histogram of distances for matched pairs
  - A histogram of distances for mismatched pairs
- Positive and Negative distributions well seaprated
  - Though with some overlap
- Negative distribution has a greater spread



Comparing Distributions

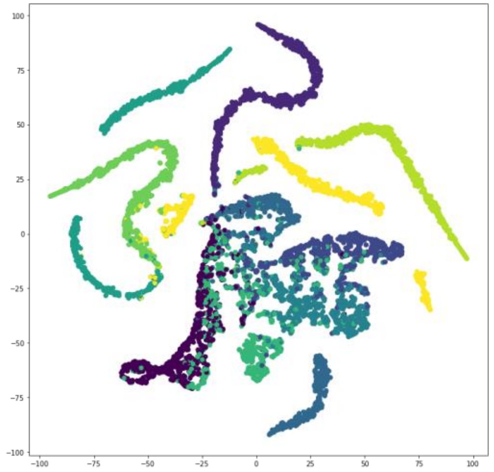
- Top: Constrastive Loss
- Bottom: Triplet Loss
- Triplet loss leads to clearer separation
  - Mean of negative pair distances further from the mean of positive pair distances
  - Negative pair spread larger in both cases

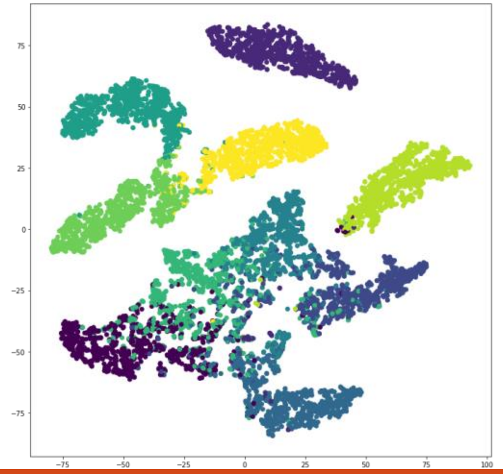




Embedding Space

- t-SNE plots of the learned embedding
  - Top: Contrastive loss
  - Bottom: Triplet loss
- Similar embedding spaces
  - Slighlty better class separation for triplet loss





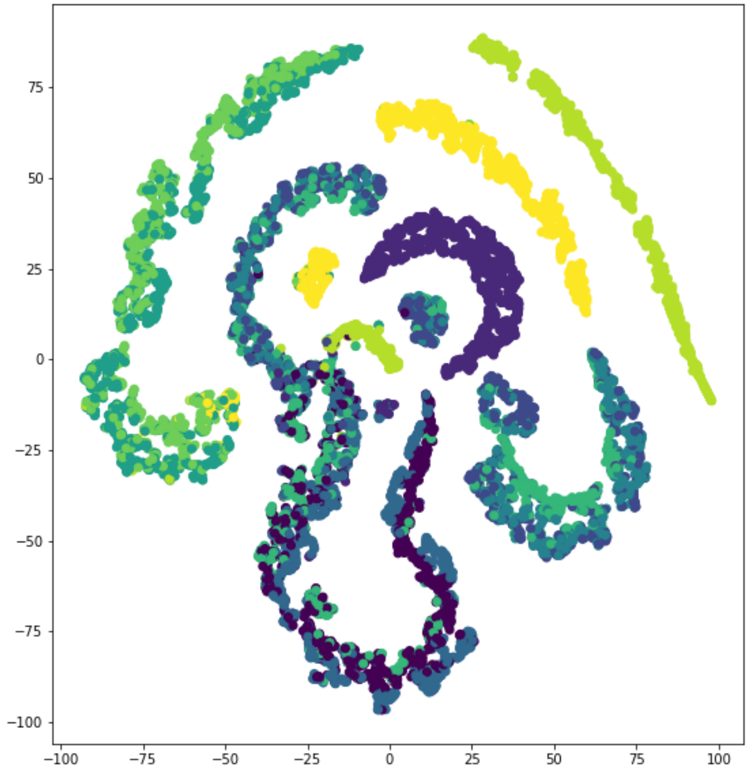
CAB420: Embedding Size

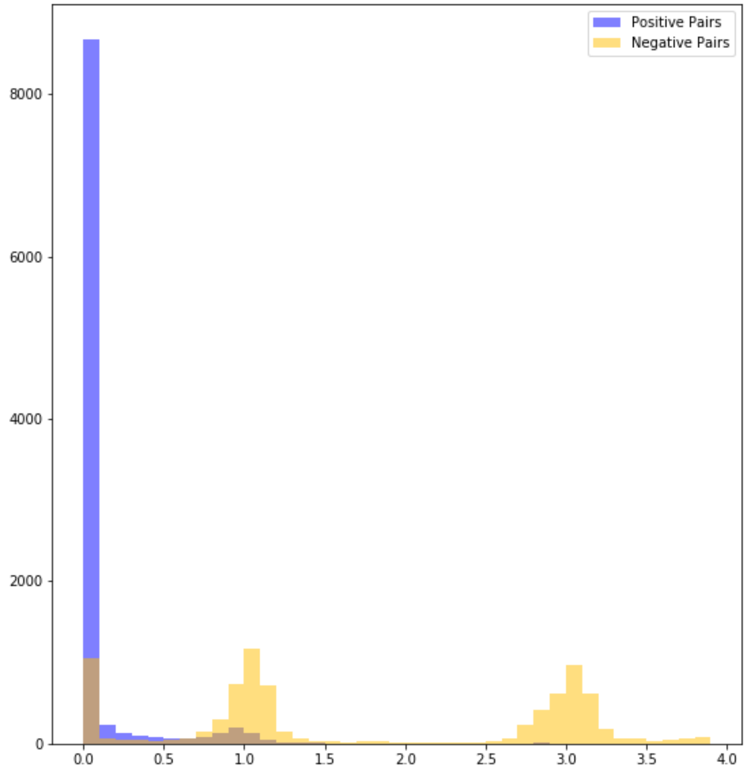
AND WHY 32 MAY NOT BE THE BEST CHOICE

Embedding Size

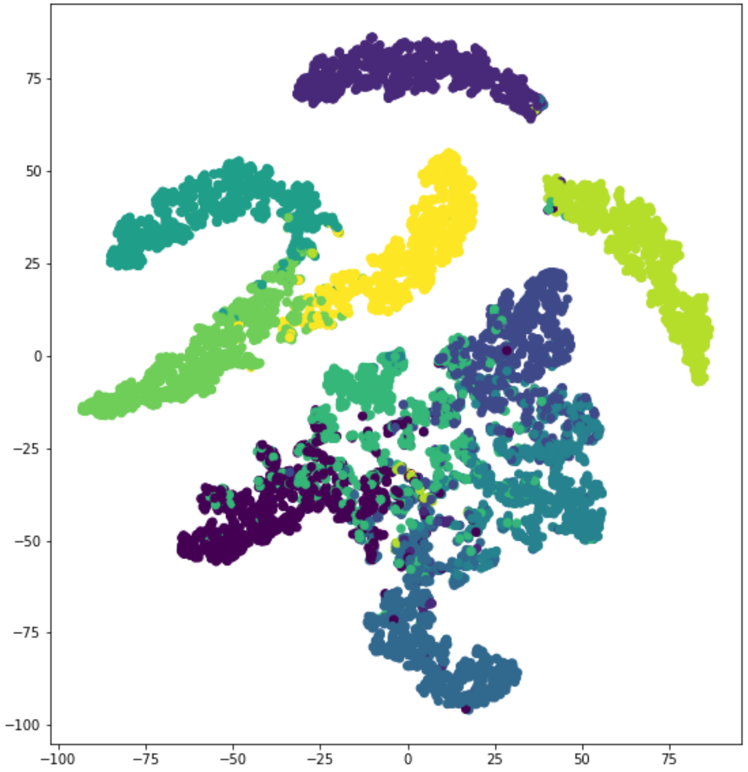
- In all examples we have used an embedding size of 32
- What happens when this changes? We'll try
  - 2
  - 4
  - 8
  - 32
  - 128
- All using triplet loss, and the same backbone with Fashion MNIST
  - See ***CAB420_Metric_Learning_Additional_Example_Embedding_Size.ipynb*** for code

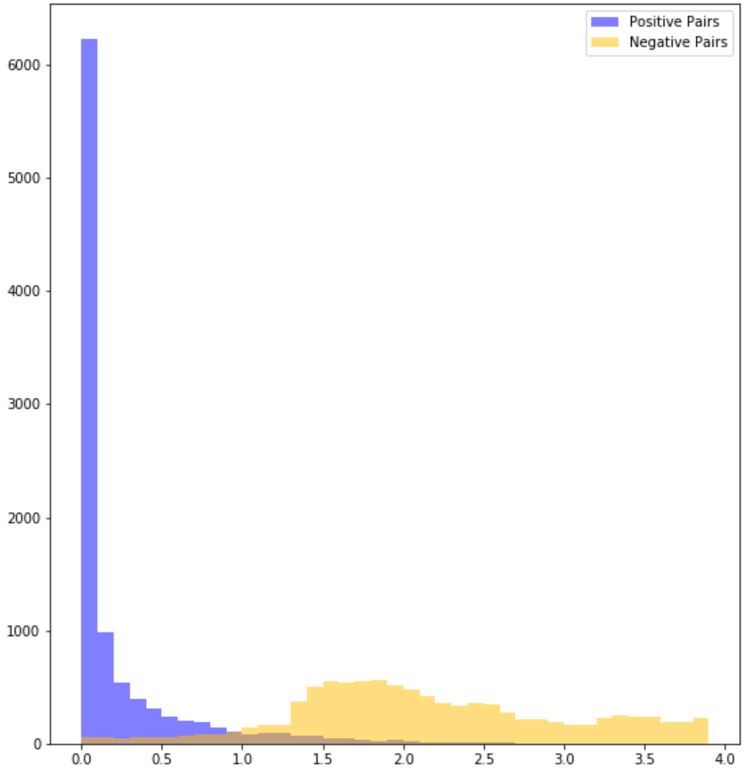
Embedding Size = 2



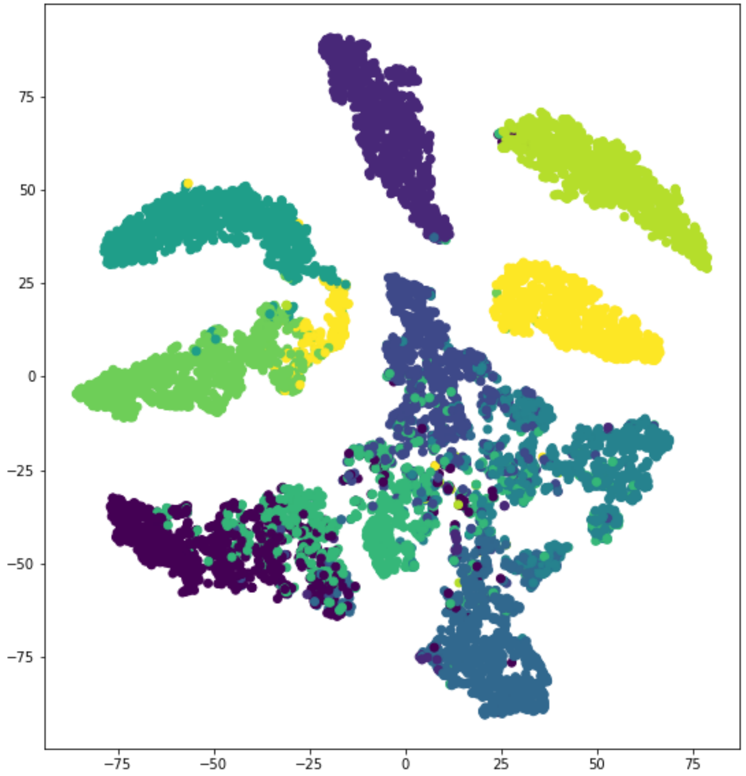


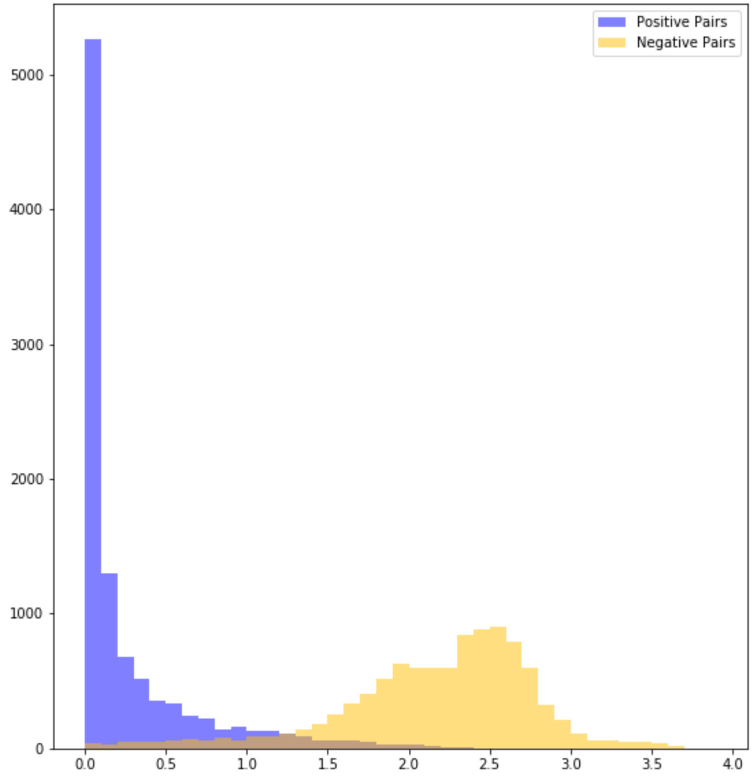
Embedding Size = 4



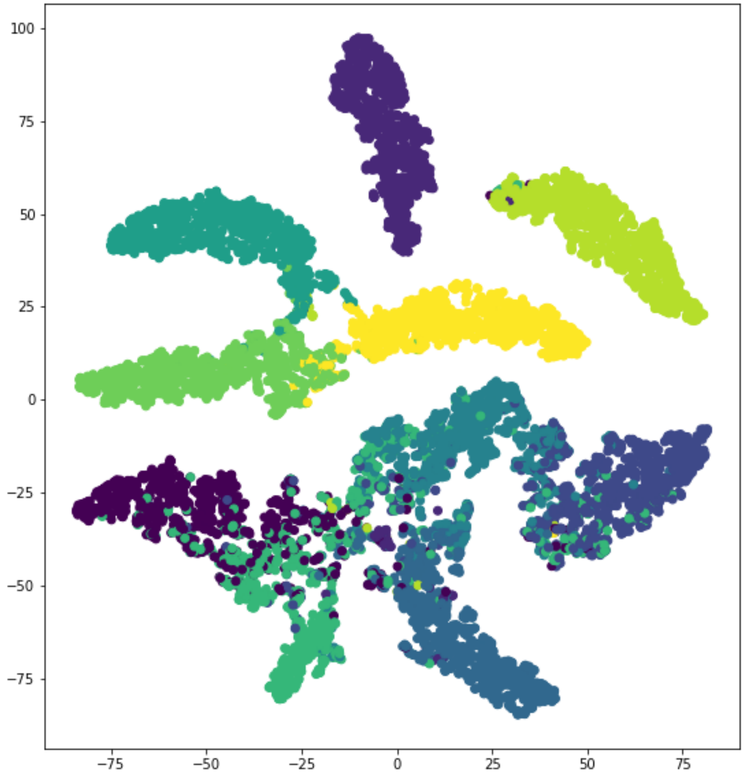


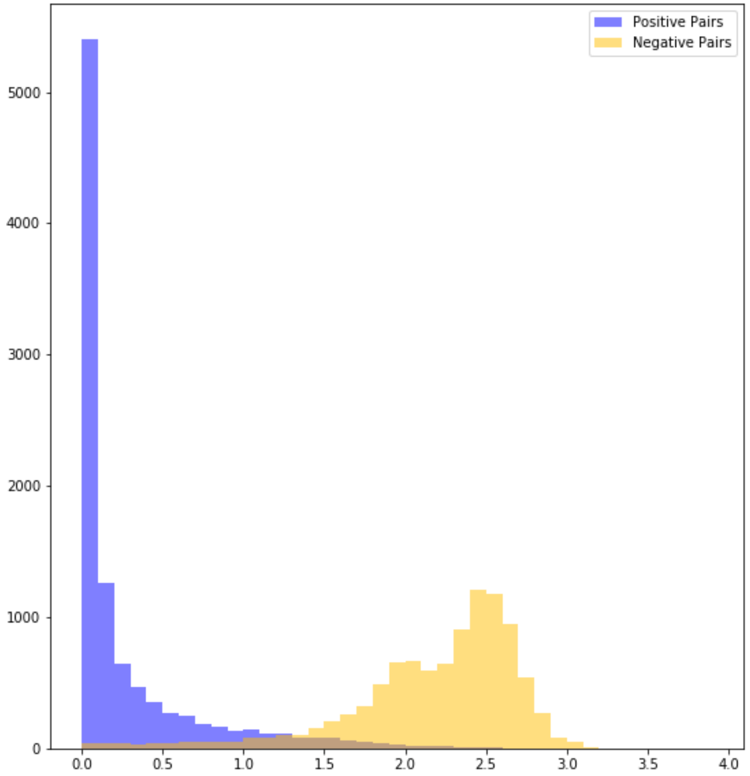
Embedding Size = 8



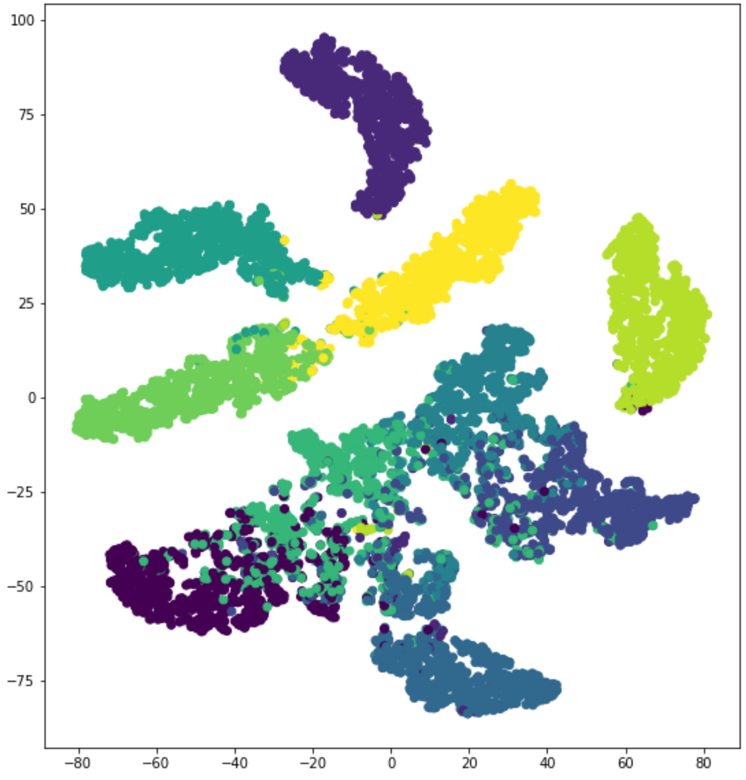


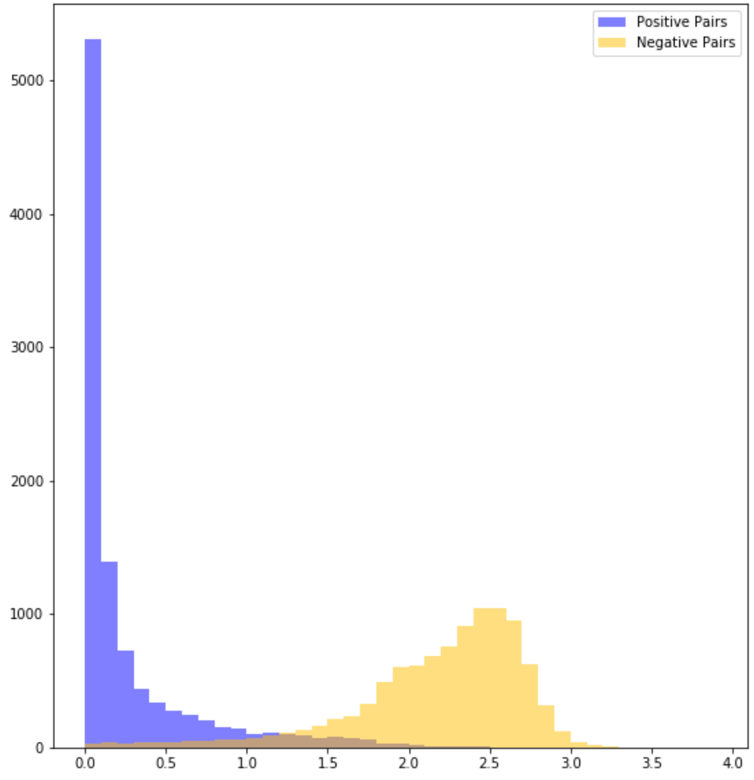
Embedding Size = 32





Embedding Size = 128





Embedding Size

- Smaller embeddings perform worse
  - 2 and 4 are noticeably worse than others
- Continued improvement as embedding size increases
  - Improvement slows as embedding gets larger
- Improvement gains for larger embeddings limited by
  - Backbone, we have the same backbone for all networks, and this can only capture so much detail
  - Data, we have 10 classes. Do we need 128 dimensions to separate these?
- Embedding size should be considered for each problem

Siamese Network Applications

WHAT DO I ACTUALLY DO WITH THIS?

Applications of Siamese Networks

- Biometric systems
  - Solve the problem of "are these two people the same"
  - Compare sets of embeddings to determine if a person is the same
- Content Based Retrieval
  - Find content like a provided sample, i.e. finding similar images
  - Compare the embedding for a given sample to a database of other content, returning the most relevant
    - Can order results by distance to return the most relevant results first

Why Siamese Networks?

- Extend better to unseen classes
- Consider a biometric system. We seek to enrol a new person:
  - For a Siamese network, we
    - Compute an embedding for their enrollment data
    - Add this to the existing database
  - For a classifiation DNN, where the network is trained to classify the identify, we
    - Modify the network to contain an additional output class
    - Re-train (possibly just fine-tune) the network

Summary Time

CATS, NARWHALS, AND LOSS FUNCTIONS

Comparing things with DCNNs

- We can replicate the idea of LDA with a DCNN
  - Supervised data required
  - No ability to map from learned feature back to the input space
- Use a DCNN as a feature extractor
  - Requires us to formulate a loss function to encourage
    - Samples from the same class to be close together
    - Samples from different classes to be far apart
- A naïve approach of binary cross entropy can do a fair job of verification, but
  - Does not provide great embeddings;
  - Won’t provide a good ranking of similarity.

Comparing things with DCNNs

- Contrastive loss considers pairs of images
  - Tries to force similar pairs to be closer together than negative pairs by a margin
- Triplet loss uses three images and forms two pairs
  - Tries to make the positive pair closer together than the negative pair by a margin
- For both, we can use normalisation to simplify setting the margin

Comparing things with DCNNs

- What we ultimately compare is a learned embedding
  - Compact representation of the input feature
- We can control the size of the embedding
  - Hyper-parameter chosen at design time
  - Bigger embedding means a richer description
    - May take longer to train
    - May have severe impacts on run-time for some tasks
  - More complex problems (larger number of classes, greater variation) will require a larger embedding