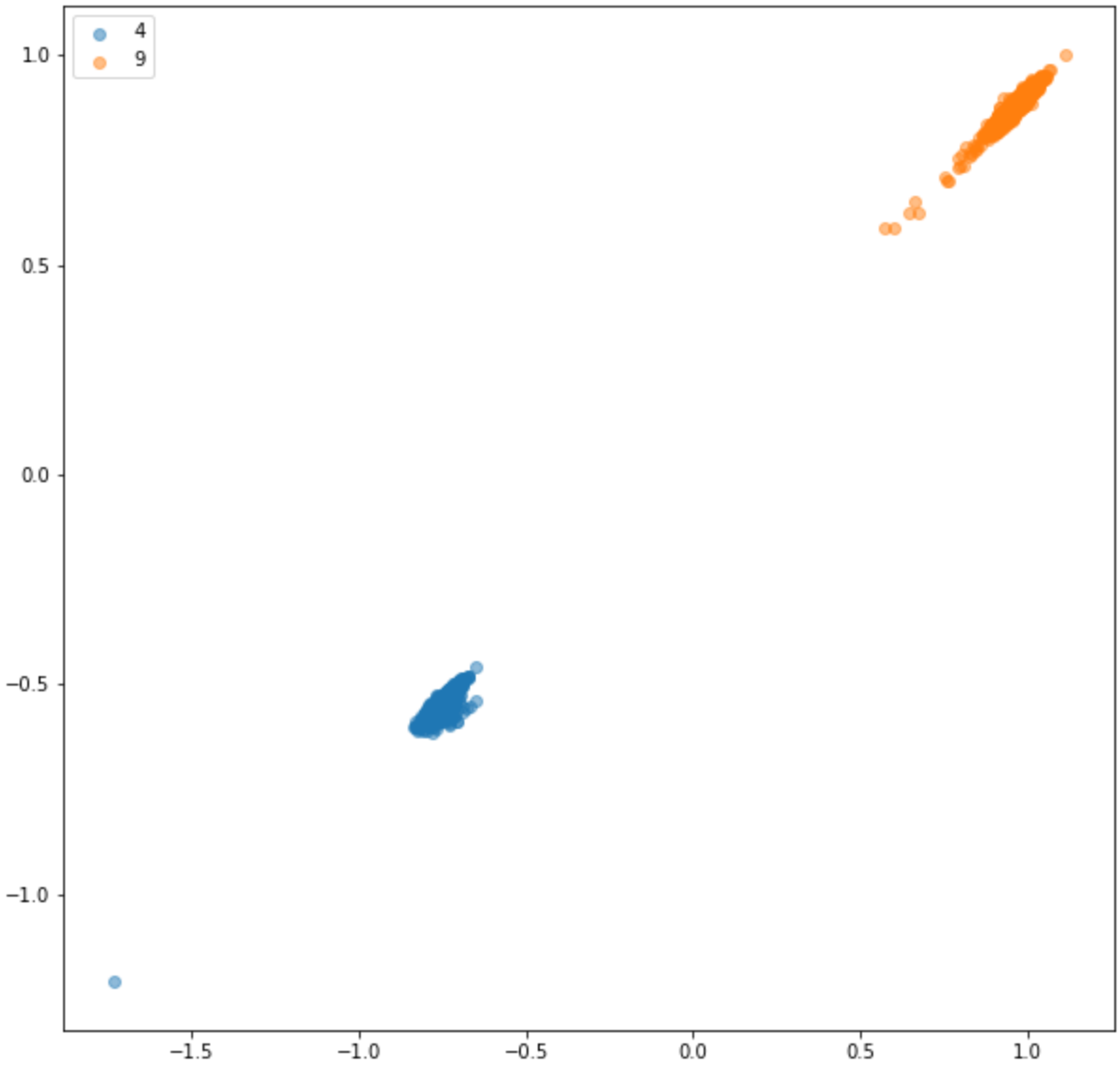
Reflection Essay - Lab 12 Inlab - Clustering Images Using Metric learning

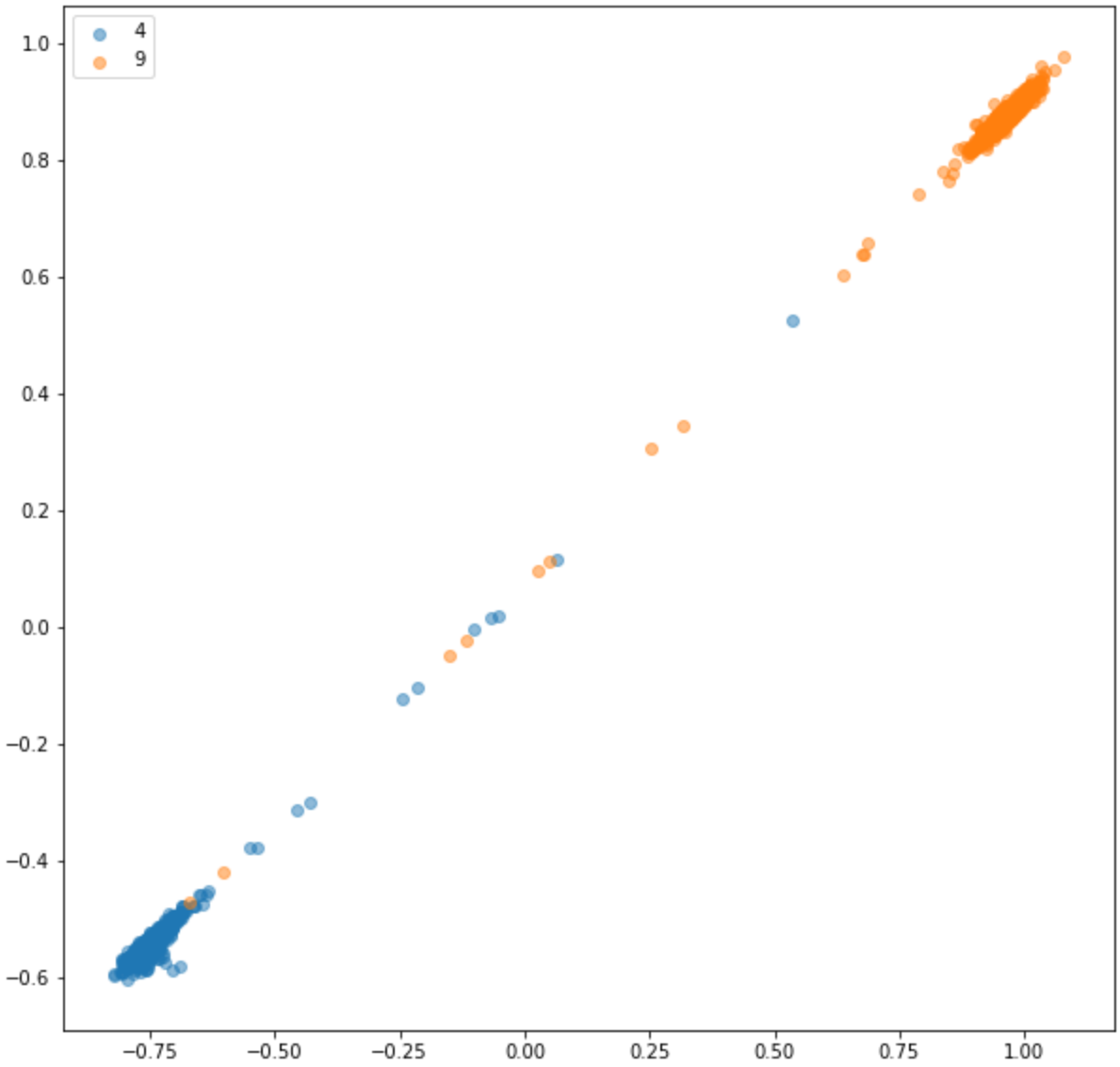
The lab was based on **feature extraction using Metric Learning**.
Features were extracted using a Siamese Network which consisted of two embedding generation networks.
If the embedding (compressed information about an input image) generated by the 2 networks in the siamese pair is f(x_o) and f(x_1), the network trains on the **contrastive loss function**

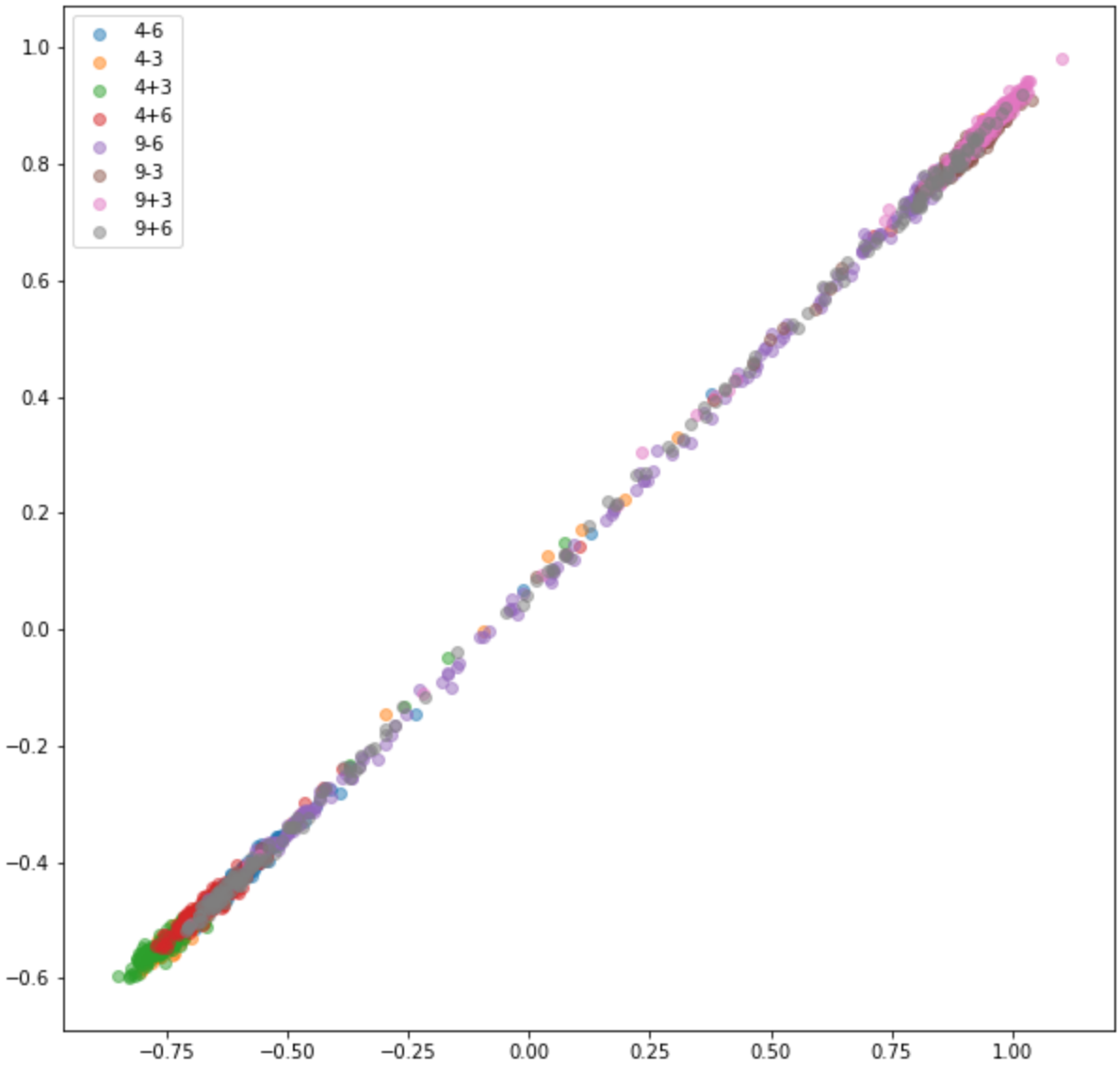
$$L_{contrastive}(x_0, x_1, y) = \frac{1}{2} y \| f(x_0) - f(x_1) \|_2^2 + \frac{1}{2}(1 - y)\{\max(0, m - \| f(x_0) - f(x_1) \|_2)\}^2$$

We filled in all the missing details as required by the # FILL HERE and questions and get the plots below for the embedding results for the **train, test and augmented test respectively** at the end.

In all the plots the the x and y axis are both features/embeddings associated with various images which are all points on the scatter plots.


4
9
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0


4
9
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


4-6
4-3
4+3
4+6
9-6
9-3
9+3
9+6
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Answers to the questions:

1. The mean listed is the mean of the entire MNIST data (after normalizing pixel values in the range [0..1]. Update the mean and standard deviation to reflect that we are loading only the characters '4' and '9'. (Note you can leave this alone without getting stuck for this assignment).
A : In the MNIST_SIAM class the self.data object was used to find the mean and variance of the dataset containing only 4 and 9

```
self.mean = self.data.mean()/255
self.std = self.data.std()/255
```

2. Is the 233rd item in the train data a '4' or a '9'? (Note we are counting from 0)
A: Simple print statement was added in MNIST_BASIC to get the 233rd item of train data. It was **found out to be '4'**.

```
print(self.class_labels[232])
```

3. What are the indices of the images for the 466th test pair-fed to the network? (Note we are counting from 0)
A: Simple print statement was added in MNIST_SIAM to get the 466th pair in the test data. The indices of the pair were **(930,1789,1)**

```
print(self.pairs[465][0])
print(self.pairs[465][1])
print(self.pairs[465][2])
```

4. With respect to the pairs of images fed to the network, what is your guess of the maximum value of a scalar that the network sees as the input? We are looking for a symbolic answer?
A: All the images are fed to the network post normalization. Thus each pixel value is mapped to a number between (0,1) by first dividing by 255 followed by mean subtraction and division by the standard deviation. So the max value that is possible at the input is

$$(1 - \mu)/\sigma$$

5. How many '4's are present in the training dataset?
A: **train : 3000 test : 900** total: 3900

6. What is the size of the output embedding? Why was this number chosen?
A: output put embedding size = 2x1
This size was chosen as it gives us large compression while keeping separation between classes 4 and 9 significant.
Further, we can only visualize a size 2 feature/embedding space in a 2D plot.

7. Write your conclusions on the 3 different plots given. Specifically, will you trust this embedding to differentiate the two characters in both the test datasets?
Which particular type of character creates maximum confusion?

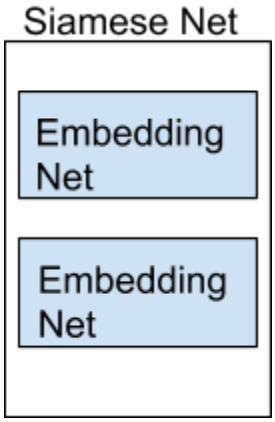
A: We observe that the test and the train data (first and second images respectively) are well separated by our trained network, however under the augmented dataset which consists of images of the numbers 4 and 9 horizontally displaced by an amount lying in the set
{-6, -3, +3, +6},
We see that the performance is not good, in particular, the letters 9 displaced by large amounts (+6/-6) perform quite poorly with numerous outliers lying outside the 2 main clusters in the lower left and the upper right corners.

8. For your outlab, we want to you to (later on) update the notebook so that this confusion is reduced. Provide some idea on how this should be done in this
Inlab.
A: A NN perform well on the data similar to the data that it has seen before while training. So if we want to increase the accuracy on the augmented data set then we will have to train the model on the augmented data set as well.

Reflection Essay - Lab 12 Outlab - Clustering Images Using Metric learning

The lab was based on **feature extraction using Metric Learning**. Metric learning is a learning paradigm through which we attempt to learn embeddings (encoded versions) of images (or other data) to establish similarity or dissimilarity between them.

An approach to extract relevant features from metric learning and project our data into a lower-dimensional embedding space is to train a Siamese Network.



A siamese network consists of a pair of identical embedding generation networks. These networks share the same set of weights.
Each embedding generation network typically consists of a sequence of Convolution (CNN) layers followed by a sequentially fully connected network terminating in a small (in our case 2) number of nodes. The number of nodes in the final layer tells us the dimensionality of the underlying embedding space.
In our case, we have used the below architecture for an individual embedding network -

CNN layers

```
self.convnet = nn.Sequential(nn.Conv2d(1, 16, 5), nn.PReLU(),
                             nn.MaxPool2d(2, stride=2),
                             nn.Conv2d(16, 64, 3), nn.PReLU(),
                             nn.MaxPool2d(2, stride=2))
```

Fully connected linear layers

```
self.fc = nn.Sequential(nn.Linear(64 * 5 * 5, 256),
                        nn.PReLU(),
                        nn.Linear(256, 64),
                        nn.PReLU(),
                        nn.Linear(64, 2)
                        )
```

Compared to the network provided to us as part of the starter code notebook for this lab exercise, we have slightly reduced the number of weights in the CNN layers of the network.

The version provided to us in starter code-

```
self.convnet = nn.Sequential(nn.Conv2d(1, 32, 5), nn.PReLU(),
                             nn.MaxPool2d(2, stride=2),
                             nn.Conv2d(32, 64, 5), nn.PReLU(),
                             nn.MaxPool2d(2, stride=2))
```

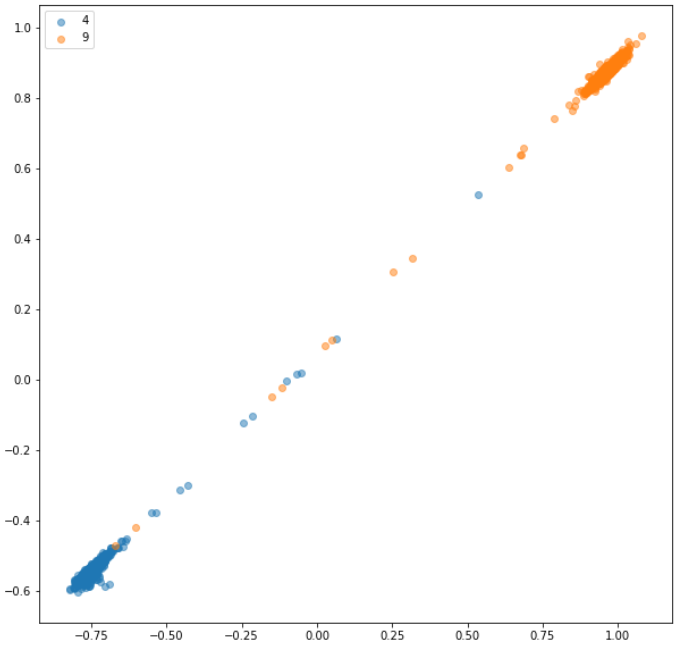
The reason for doing this is discussed in **question 1**.

The loss function used to train a Siamese Network is crucial to achieving the goal of distinguishing between similar and dissimilar MNIST characters.
If the embeddings (compressed information about an input image) corresponding to the first and second image are f(x_o) and f(x_1), the network trains on the below **contrastive loss function**

$$L_{contrastive}(x_0, x_1, y) = \frac{1}{2} y \|f(x_0) - f(x_1)\|_2^2 + \frac{1}{2}(1-y)\{\max(0, m - \|f(x_0) - f(x_1)\|_2)\}^2$$

When the label y = 1, we assume that the input pair is semantically similar and we incentivize f(x_o) being close to f(x_1).
Whereas when y = 0, the input pair is dissimilar and we wish to make the distance between the embeddings of the inputs as at least - m (The margin hyperparameter, typically set to 1.0)

If trained correctly, the network is able to separate MNIST characters from different classes into distinct clusters in the embedding space.
An example of this is shown below, here we distinguish between MNIST 4 and 9.

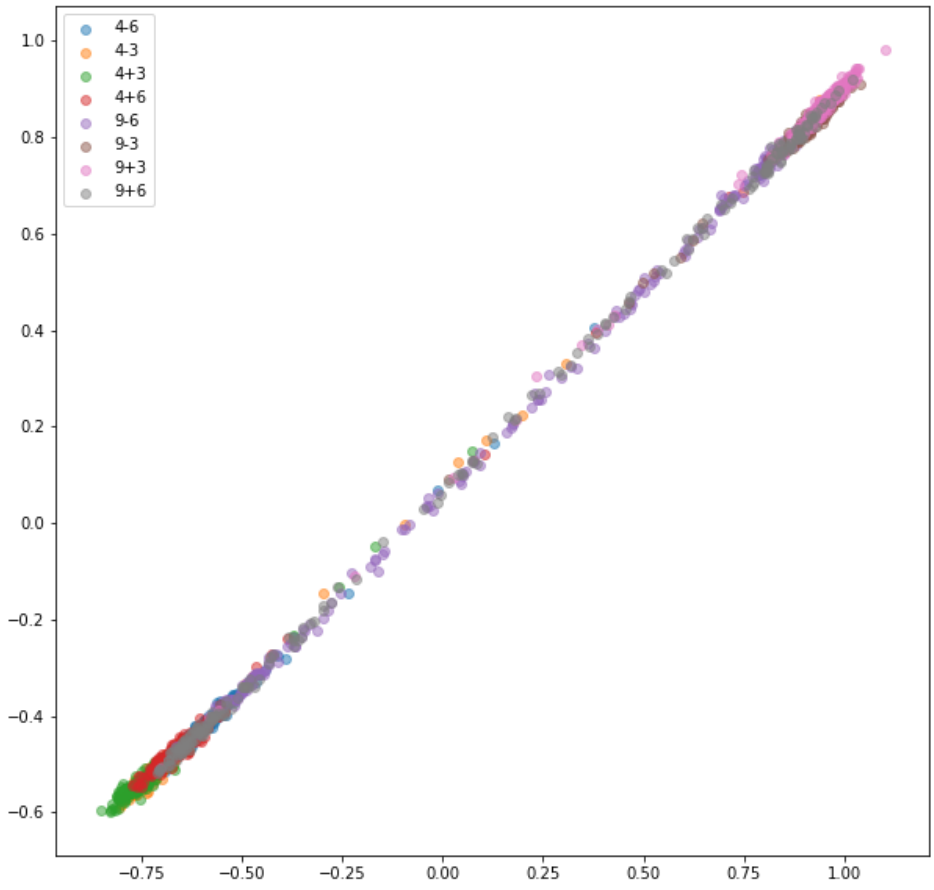


Goal of outlab exercise-

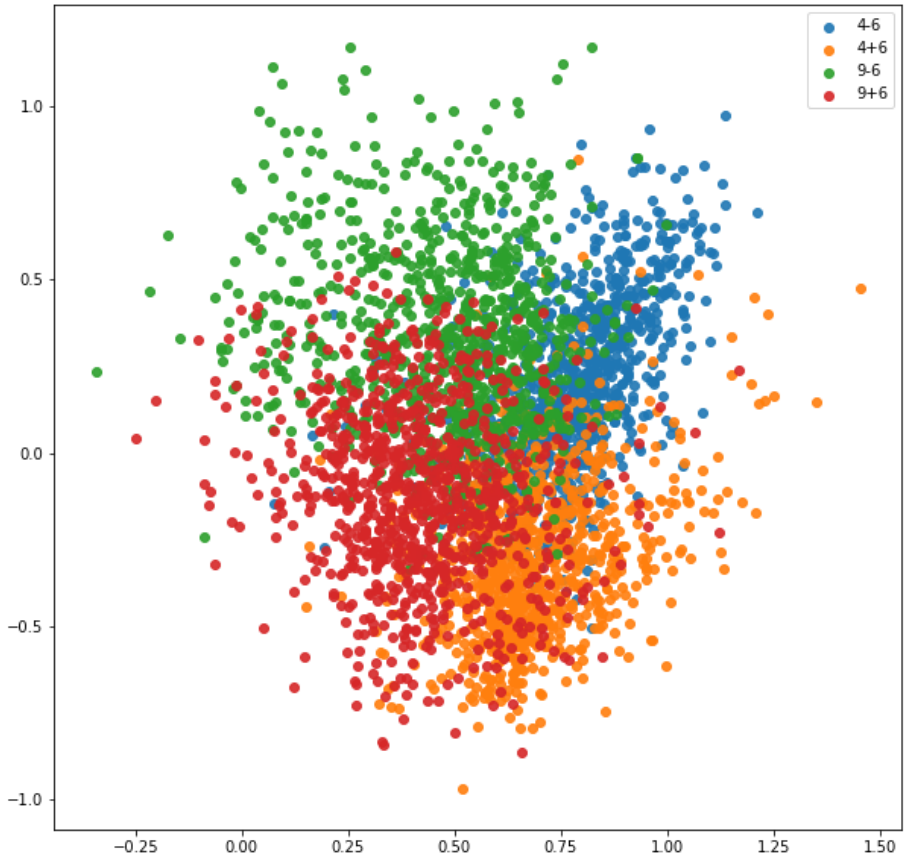
As part of the inlab exercise, we successfully distinguished between 4s and 9s by clustering them into dense, well-separated clusters. However, we were also interested in the performance of our classifier on an augmented dataset consisting of horizontally translated versions of MNIST characters.

On this augmented test set, though the network is able to associate augmented characters with their correct classes in most cases, we are unable to make “intra-class” distinctions between the characters. That is we are unsure as to what the origin class (Whether + or - 6 augmentation) of a certain datapoint was.

The aim of the outlab exercise thus is to move from a representation like this -

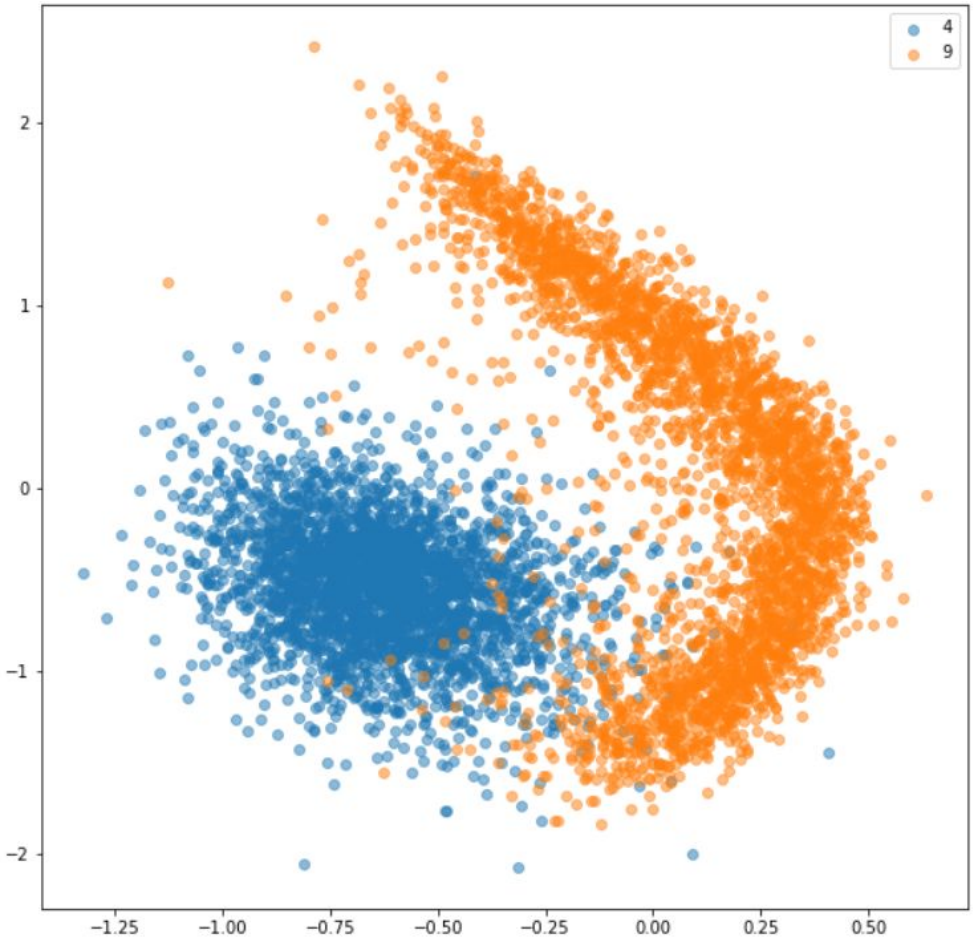


To a representation like this-

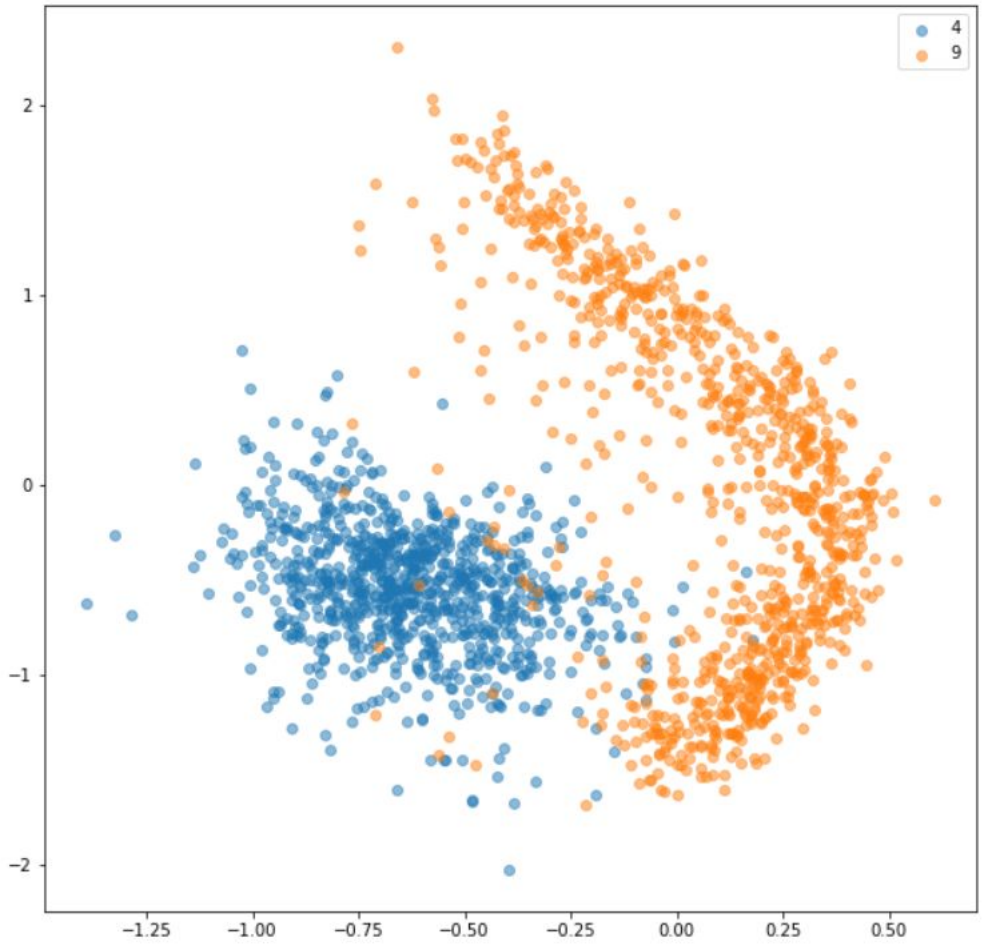


To achieve this goal, a possible method is to train only with the top 5 nearest neighbors of a certain MNIST character (4 or 9) within its respective class as the **positive pairs**.
The above suggestion leaves open to interpretation the way of choosing the **negative pairs.** To explore the effect of different choices, we tried 2 training regimes differing in the way they choose negative pairs.
1. Negative pairs shown to the network were (4, any random 9 from dataset) and (9, any random 4 from dataset) (This is identical to the inlab). The results for this looked as follows -

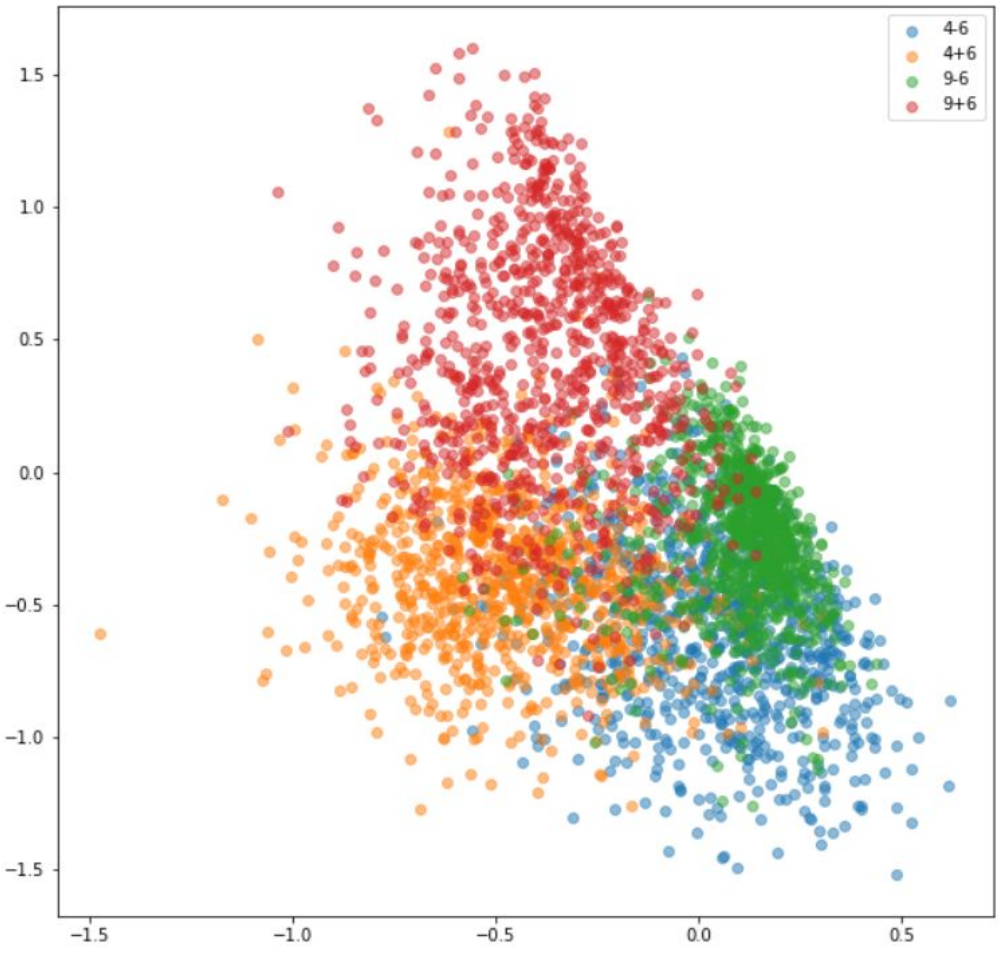
Output on the training set



Output on the test set

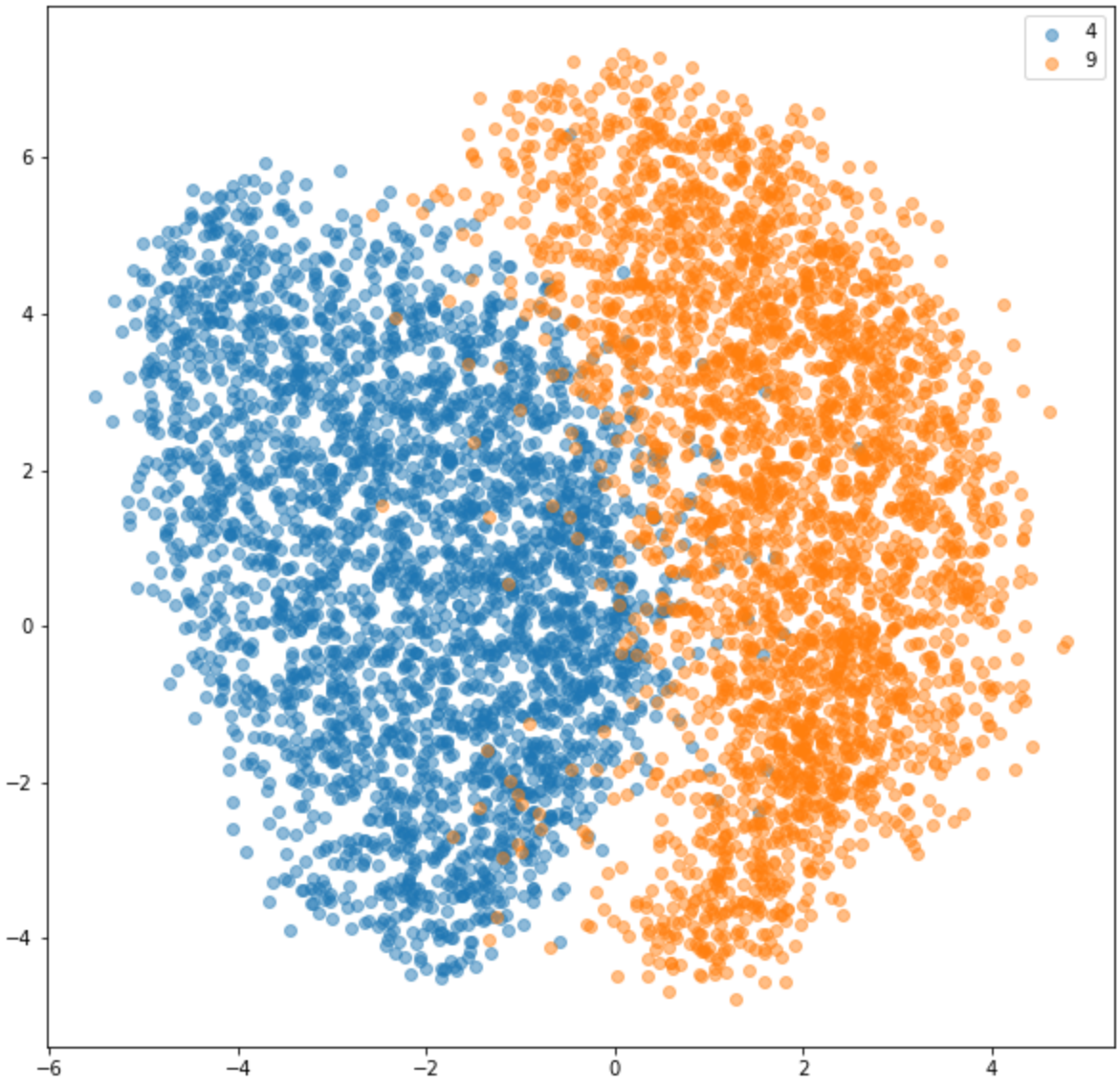


Output on the augmented set

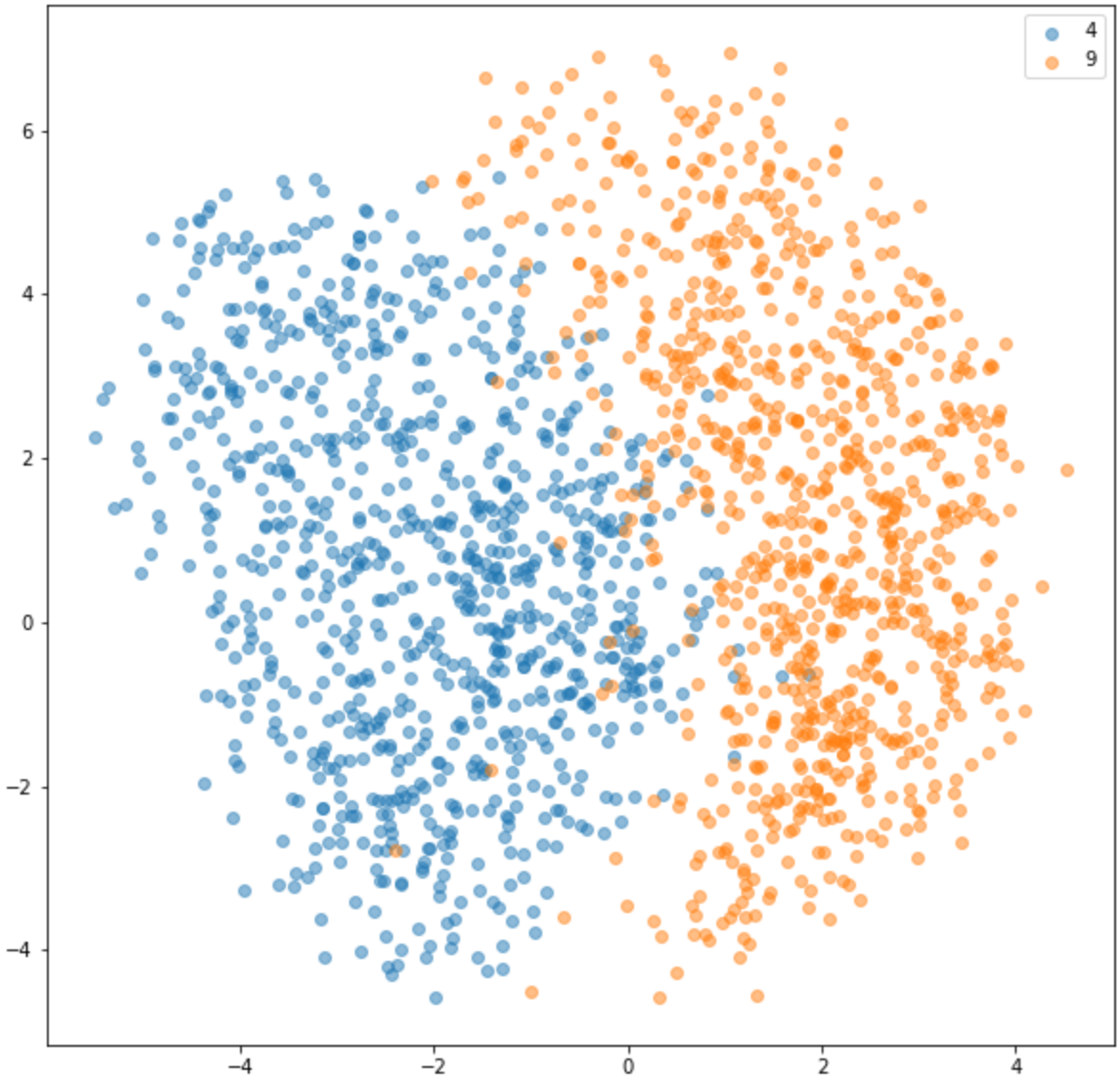


2. Negative pairs shown to the network were (4, any random 9 from dataset), (4, any random non NN 4 from dataset), (9, any random 4 from dataset), (9, any random non NN 9 from dataset). Note that this means some (4, 4) and some (9,9) pairs would be taken as negative by the network. The results for this looked as follows -

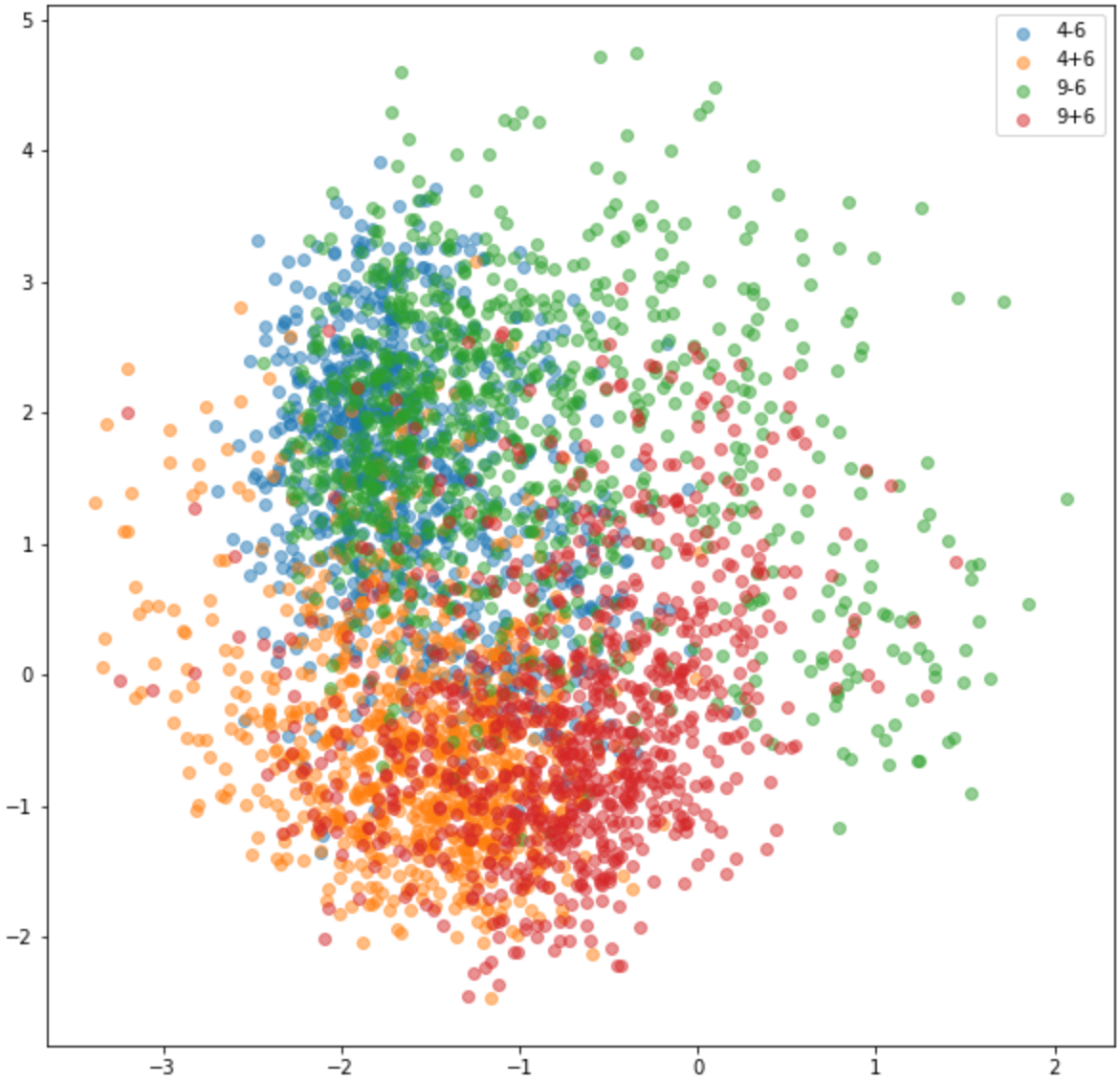
Results on training data-set



Results on test data-set



Results on augmented dataset



Both regimes seem to have their pros and cons.
Under regime 1 - we present negative pairs as only the ones differentiating between MNIST 4 and 9 s. This causes the network to learn a more dense clustering. However, this reduces the spread of the data for all three cases which might be seen as a **negative from the point of view of intra-class differences.**
Under regime 2 - the negative pairs also consist of (4, 4) and (9, 9) pairs (most of the time the pairs cannot be a NN, but can with a very small probability). Since the network learns to even put two 4 characters (or 9) in different classes (unless they are NN), the overall spread we get is higher, however this virtue backfires somewhat when trying to visualize the augmented dataset since the high spread **emphasizes intra-class** differences but diminishes **inter-class differences.**

Question 1: Why does the new nearest neighbor training regime achieve this type of clustering?

In the inlab exercise, while training our network, we taught it to learn similarities between any pair of 4s or any pair of 9s. This leads to the network recognising some high level features common to all characters of a particular type.
The network gains a high level understanding of what a 4 looks like and is able to embed this knowledge into a 2 dimensional space. This leads to a dense clustering within the family of 4s and 9s - i.e great inter-class separation.
When a pair of 4s (or 9s) are close to each other some sort of distance metric (cosine similarity in our case) they are likely to be similar in many respects like size, details of curvature and shape etc. This means that the network will not be unable to get a good understanding of what a **generic 4** should look like and it would end up giving us a larger spread.
In a way we are **sabotaging the feature clustering performance** of our network to get a greater spread. This is also the reason that we **decrease the representative capacity of the CNN** layers of our SIAMESE network. Fewer weights make it harder for the network to recognise features which further leads to a greater spread in the embedding space.
This achieves the aim of **improving intra-class separation at the cost of inter-class separation**. Since differences within a class would be invisible under a dense clustering. But under a poorer quality spread out clustering such differences would be visible.

Question 2: Compared to what has been provided, what would be an even better solution? Sketch it. How would you change the training regime if you want this even better result?

As mentioned above, using the nearest neighbor for training the intraclass difference is enhanced at the cost of the interclass differences which leads to us having distribution for the augmented data as provided.
A better solution would have been if we would have got four separate clusters corresponding to each of the augmented data types (4-6,4+6,9-6,9+6). That is we wish to maintain both a good **inter-class** and a good **intraclass** separation.
To better achieve this goal, we consider two modified loss functions

1. **Based on Triplet loss**: Under triplet loss, we calculate the loss arising from a collection of three images - an Anchor image x_a, an image of the same class as the anchor image x_p (positive) and an image from the other class x_n (negative).
   The triplet loss function makes sure that x_a is closer to x_p as compared to x_n by a certain margin - m. Overall this regime leads to slightly better inter-class separation on unseen data as we had seen in the earlier Metric Learning lab.
   In particular, in our setting, x_a would be the current datapoint being accessed, x_p would be a random NN of x_a and x_n would be a random element of the other class 9.

2. **Based on correcting for the wrong negative pairs:** In this, we design a new loss function such that we get different loss values for different types of pairs that we have i.e. **a true positive** pair, (4 or 9 with its NN), **false negative** pair (4/9 with another 4/9 but not its NN) and the **negative pair** (4/9, random 9/4).
   The loss function then will be defined in such a way that it penalizes the classification of positive as negative the most and the penalty decreases for the positive as false negative classification. This type of loss function will lead to pushing away the positive and negative pairs more relative to the positive and false negative pairs thus giving us better inter-class and intraclass separation.