Siamese and Triplet networks

applications and implementation

Adam Bielski

adam.bielski@tooploox.com

@BielskiAdam

TOOPLOOX

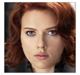
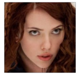
Plan

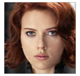
- Metric learning intuition
- Siamese networks
- Triplet networks
- PyTorch implementation
- Smarter training
- PyTorch again

Metric learning

```
def dist(img1, img2):
    return np.sqrt(np.sum(np.square(img1-img2)))
```

Is this the same person?

dist( , ) = 8372.90

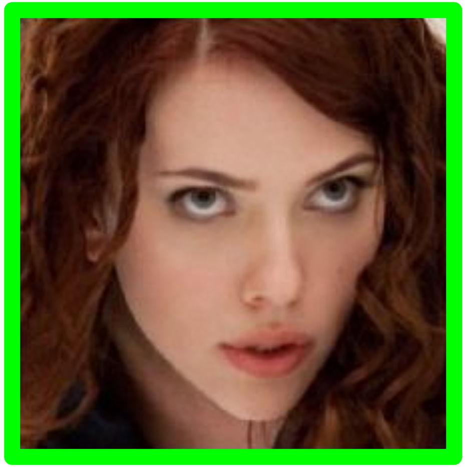
dist( , ) = 8222.07

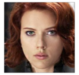
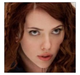
How can we do it better?

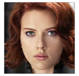
Metric learning

```
def dist(img1, img2):
    return np.sqrt(np.sum(np.square(img1-img2)))
```

Is this the same person?

f() f() f()

dist(f(), f()) = 0.05

dist(f(), f()) = 1.2

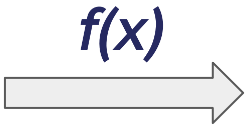
What is *f()*?

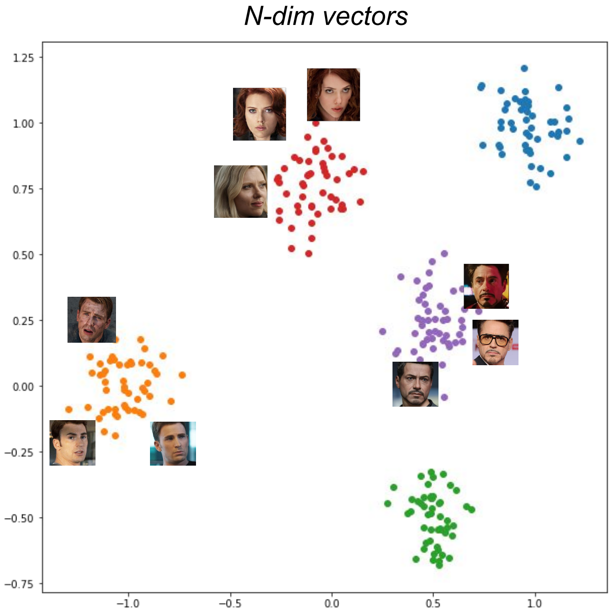
Let's extract features

...or learn it from data!

Metric embedding learning

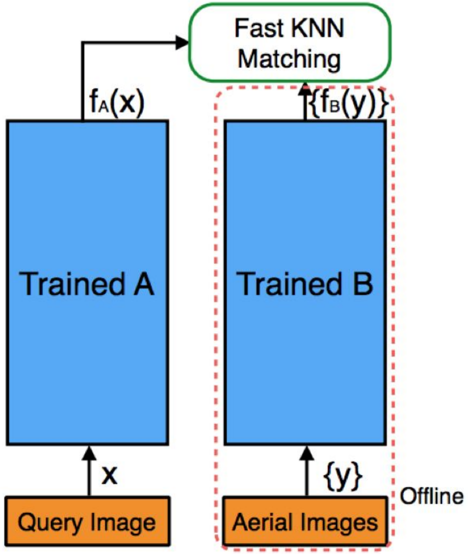
Map **semantically similar points** onto **metrically close points**





Metric learning applications

- Biometrics (face/speaker/fingerprint recognition)
- Matching
  - Match text to image
  - Descriptors matching
- Retrieval
  - Image search
  - Documents retrieval
- k-shot learning



(b) Testing

Lin et al. CVPR'15

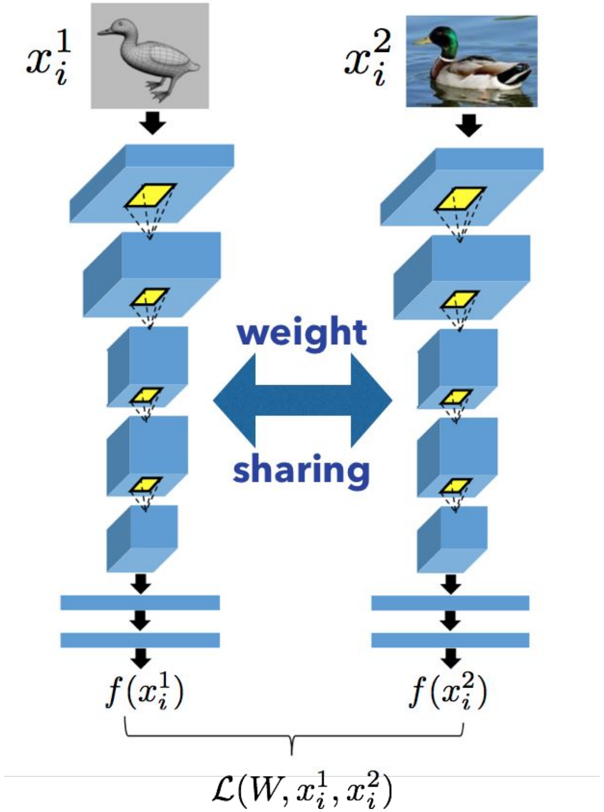
Siamese network

- Two branches with shared weights
- Two inputs
- **Contrastive Loss** defined for pairs

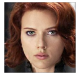
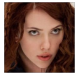
$$\min_{W_1, W_2} \frac{1}{2}(1-y)D^2 + \frac{1}{2} y \max\{0, m - D\}^2$$

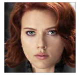
$$D = |f(W_1, x_i^1) - g(W_2, x_i^2)|_2 \qquad y = \begin{cases} 0 & \text{if } (x_i^1, x_i^2) \text{ are similar} \\ 1 & \text{otherwise} \end{cases}$$

m : margin of desirable distance for a dissimilar par



Siamese network

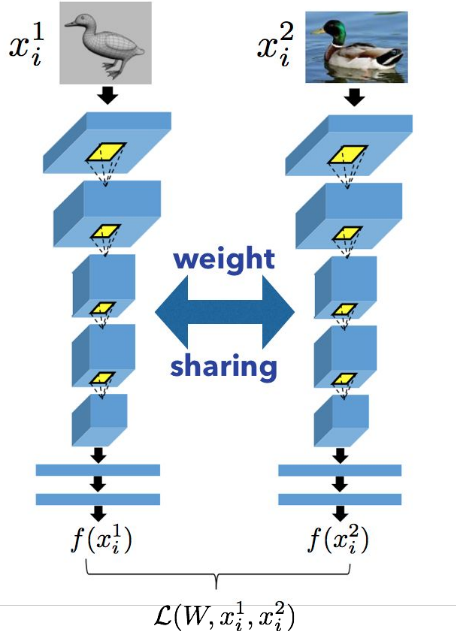
Make dist(f(), f()) as close as possible

Make dist(f(), f()) separated by >= m

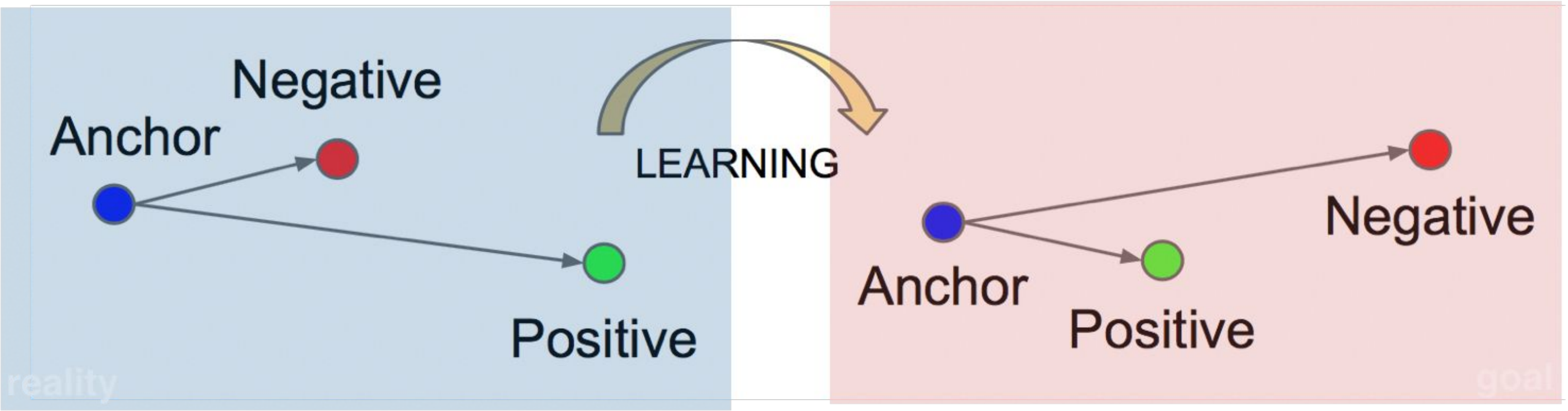
$$\min_{W_1, W_2} \frac{1}{2}\boxed{(1-y)D^2} + \frac{1}{2}\boxed{y \max\{0, m - D\}^2}$$

$$D = |f(W_1, x_i^1) - g(W_2, x_i^2)|_2 \qquad y = \begin{cases} 0 & \text{if } (x_i^1, x_i^2) \text{ are similar} \\ 1 & \text{otherwise} \end{cases}$$

m : margin of desirable distance for a dissimilar par



Triplet networks



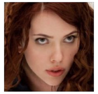
Schroff et al., CVPR'15

Triplet networks

- Three branches with shared weights
- Three inputs - anchor, positive, negative
- **Triplet Loss** defined for… triplets

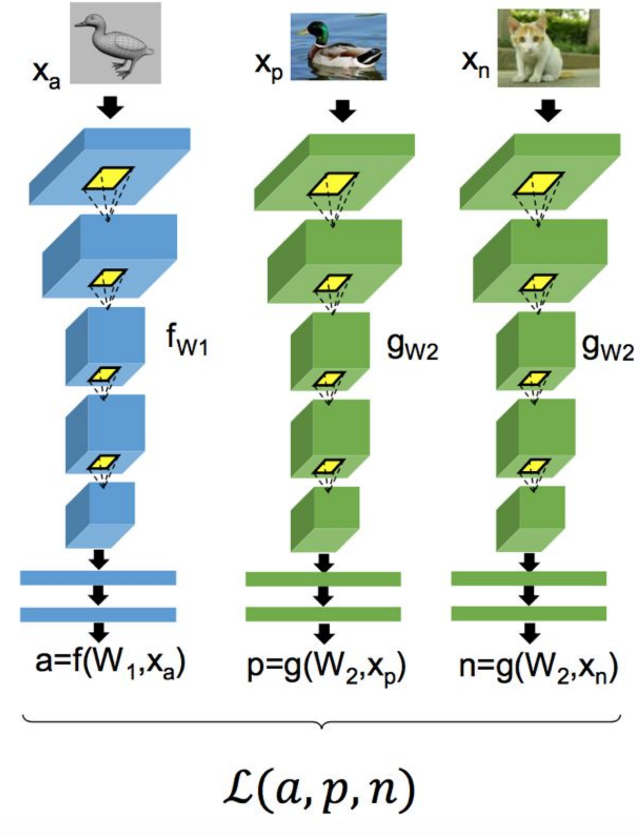
$$\mathcal{L}(a,p,n) = \frac{1}{2}\max(0, m + D^2(a,p) - D^2(a,n))$$

```
      Make dist(f(
```

```
), f(
```

```
))
Smaller than dist(f(
```

```
), f(
```

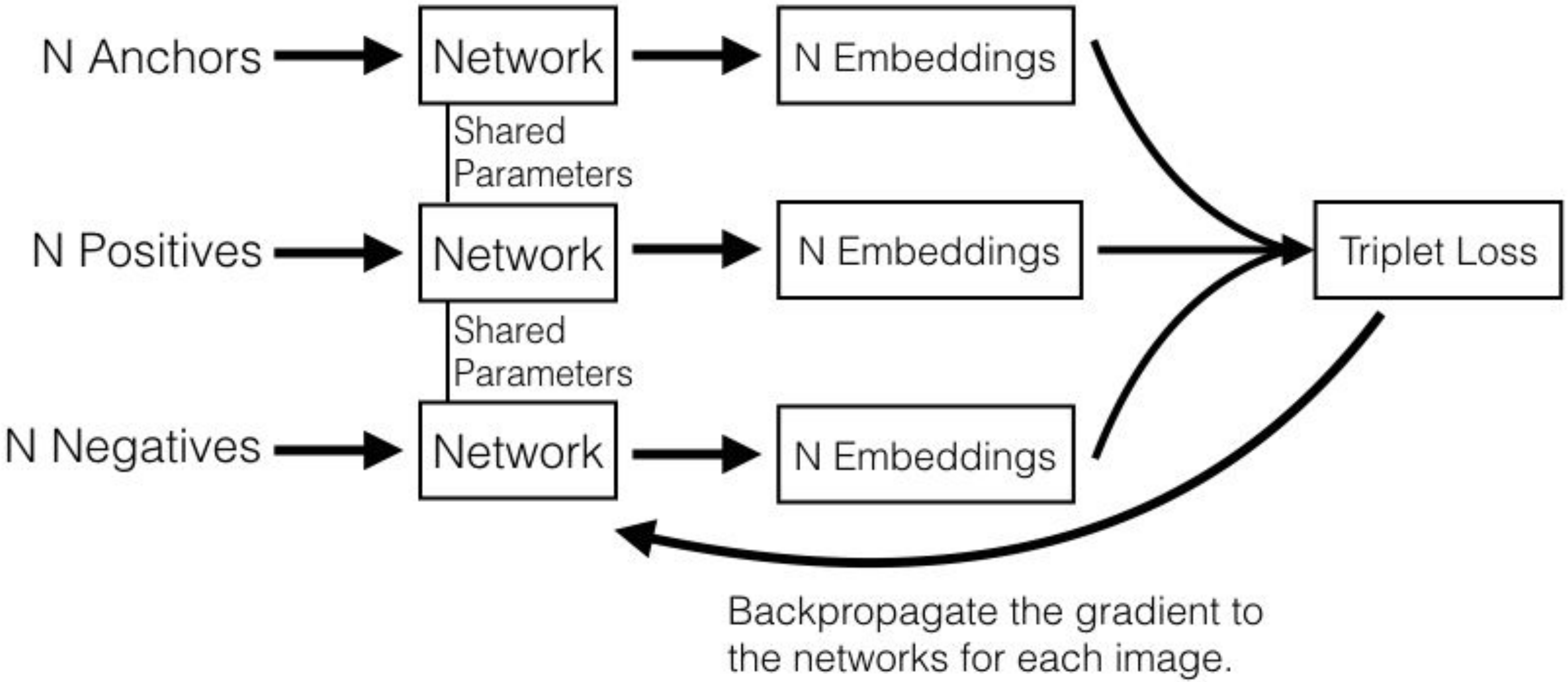
```
))
    by margin >=  m
```



Classical implementation

1. **Sample** pairs/triplets
2. **Extract** embeddings
3. Compute **loss** on pairs/triplets
4. Backpropagate

Classical implementation



http://bamos.github.io/2016/01/19/openface-0.2.0/

The Incredible PYTORCH

https://github.com/adambielski/siamese-triplet/

Embedding network

```
class EmbeddingNet(nn.Module):
    def __init__(self):
        super(EmbeddingNet, self).__init__()
        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))

        self.fc = nn.Sequential(nn.Linear(64 * 4 * 4, 256),
                                nn.PReLU(),
                                nn.Linear(256, 256),
                                nn.PReLU(),
                                nn.Linear(256, 2)
                                )

    def forward(self, x):
        output = self.convnet(x)
        output = output.view(output.size()[0], -1)
        output = self.fc(output)
        return output
```

Siamese network

Wrapper for Embedding network

```
class SiameseNet(nn.Module):
    def __init__(self, embedding_net):
        super(SiameseNet, self).__init__()
        self.embedding_net = embedding_net

    def forward(self, x1, x2):
        output1 = self.embedding_net(x1)
        output2 = self.embedding_net(x2)
        return output1, output2

    def get_embedding(self, x):
        return self.embedding_net(x)
```

MNIST-like dataset sampling pairs

```
class SiameseMNIST(Dataset):
    """
    Train: For each sample creates randomly a positive or a negative pair
    Test: Creates fixed pairs for testing
    """

    def __init__(self, mnist_dataset):
        self.mnist_dataset = mnist_dataset
        (...)

    def __getitem__(self, index):
        if self.train:
            target = np.random.randint(0, 2) # positive or negative pair
            img1, label1 = self.train_data[index], self.train_labels[index]
            img2 = self.sample(target, label1)
        return (img1, img2), target

    def __len__(self):
        return len(self.mnist_dataset)
```

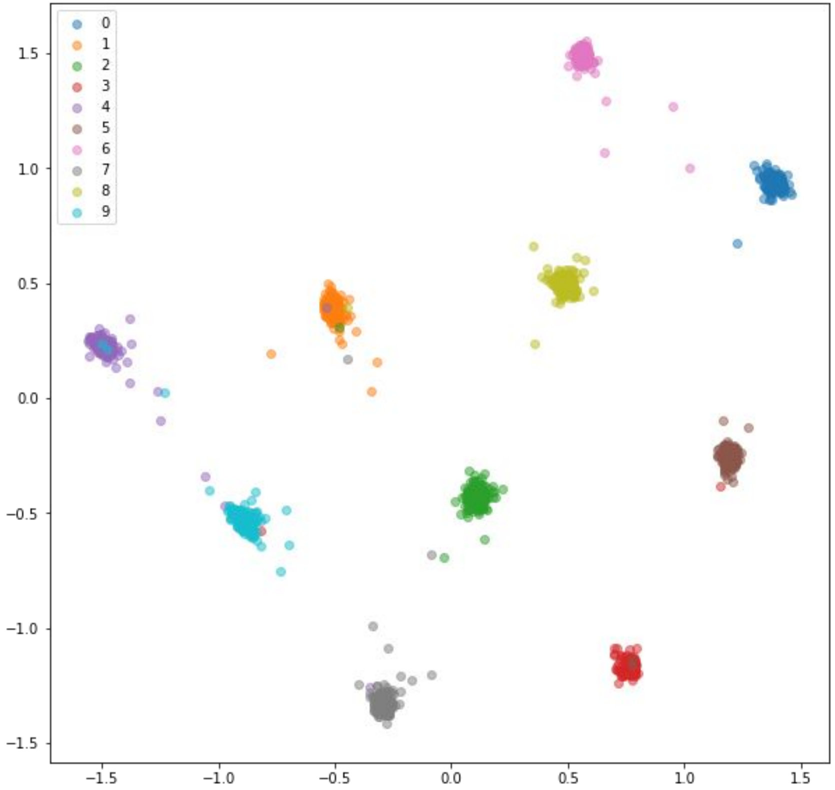
Contrastive loss

```
class ContrastiveLoss(nn.Module):
    """
    Contrastive loss
    Takes embeddings of two samples and a target label == 1 if samples are from the same class and label == 0 otherwise
    """
    def __init__(self, margin):
        super(ContrastiveLoss, self).__init__()
        self.margin = margin

    def forward(self, output1, output2, target, size_average=True):
        distances = (output2 - output1).pow(2).sum(1)  # squared distances
        losses = 0.5 * (target.float() * distances +
                        (1 + -1 * target).float() * F.relu(self.margin - distances.sqrt()).pow(2))
        return losses.mean() if size_average else losses.sum()
```

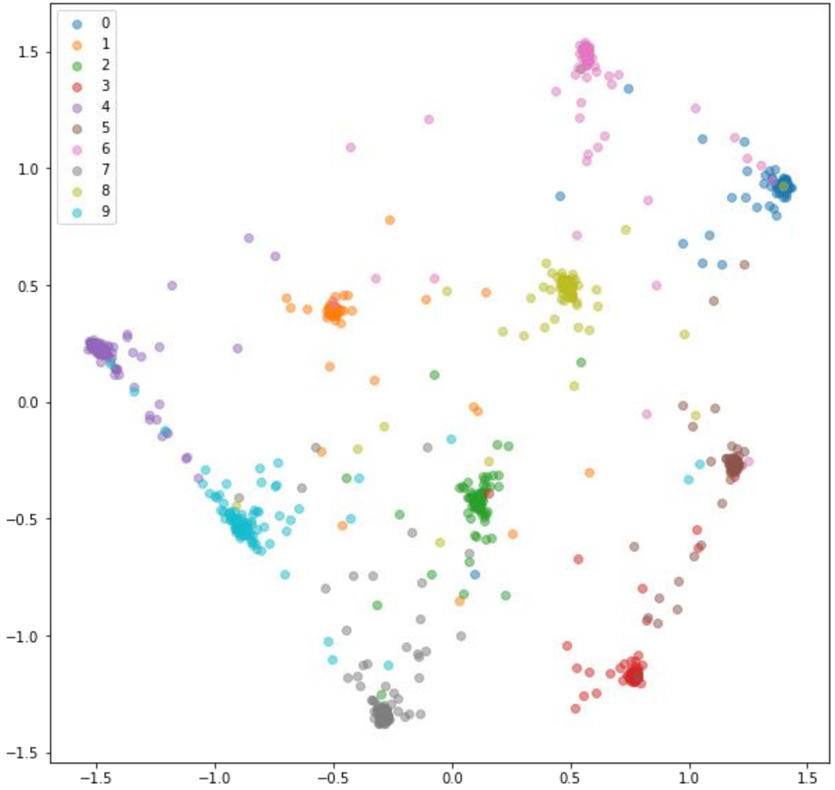
Putting it together

```
from torchvision.datasets import MNIST
train_dataset, test_dataset = MNIST('../data/MNIST', train=True, download=True, (...)), (..._

siamese_train_dataset = SiameseMNIST(train_dataset) # Returns pairs of images and target same/different
siamese_test_dataset = SiameseMNIST(test_dataset)
batch_size = 128
siamese_train_loader = torch.utils.data.DataLoader(siamese_train_dataset, batch_size=batch_size, shuffle=True)
siamese_test_loader = torch.utils.data.DataLoader(siamese_test_dataset, batch_size=batch_size, shuffle=False)

embedding_net = EmbeddingNet()
model = SiameseNet(embedding_net)

margin = 1.
loss_fn = ContrastiveLoss(margin)
lr = 1e-3
optimizer = optim.Adam(model.parameters(), lr=lr)
scheduler = lr_scheduler.StepLR(optimizer, 8, gamma=0.1, last_epoch=-1)
n_epochs = 20
fit(siamese_train_loader, siamese_test_loader, model, loss_fn, optimizer, scheduler, n_epochs)
```

Siamese - visualization



Train



Validation

TOOPLOOX

Triplet network

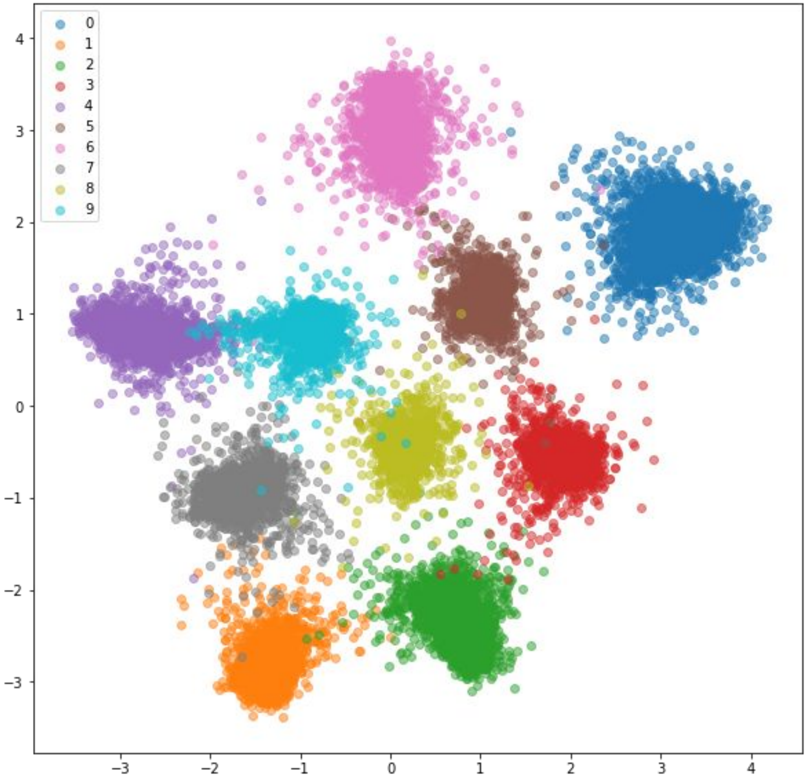
Wrapper for Embedding network

```
class TripletNet(nn.Module):
    def __init__(self, embedding_net):
        super(TripletNet, self).__init__()
        self.embedding_net = embedding_net

    def forward(self, x1, x2, x3):
        output1 = self.embedding_net(x1)
        output2 = self.embedding_net(x2)
        output3 = self.embedding_net(x3)
        return output1, output2, output3

    def get_embedding(self, x):
        return self.embedding_net(x)
```

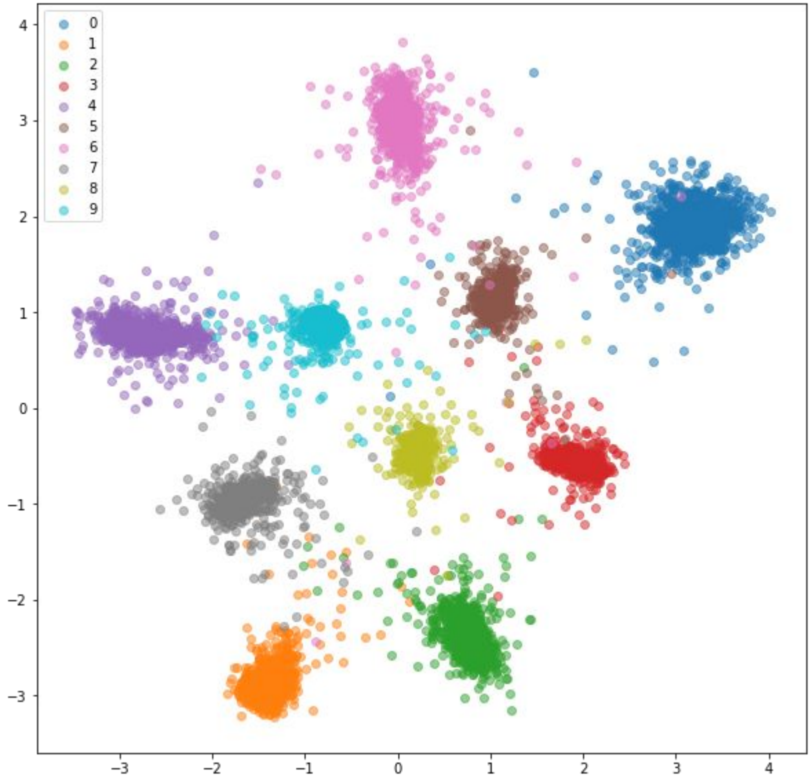
MNIST-like dataset sampling triplets

```
class TripletMNIST(Dataset):
    """
    Train: For each sample (anchor) randomly chooses a positive and negative samples
    Test: Creates fixed triplets for testing
    """

    def __init__(self, mnist_dataset):
        self.mnist_dataset = mnist_dataset
        (...)

    def __getitem__(self, index):
        if self.train:
            img_anchor, label_anchor = self.train_data[index], self.train_labels[index]
            img_positive, img_negative = self.sample(label_anchor)
        return (img_anchor, img_positive, img_negative), []

    def __len__(self):
        return len(self.mnist_dataset)
```

Triplet network - visualization



Train



Validation

TOOPLOOX

Problems with training

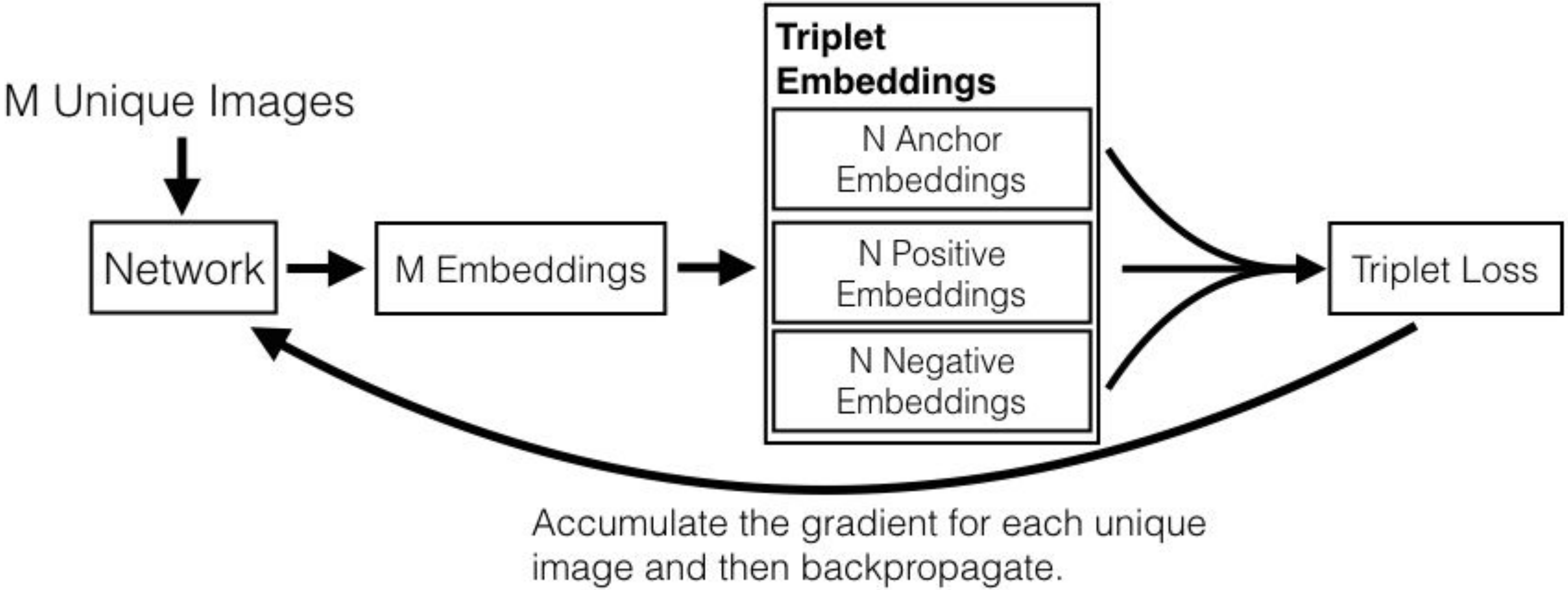
- Number of possible **pairs/triplets** grows **quadratically/cubically**
- *f(x)* quickly learns to map trivial pairs/triplets; random selection is **uninformative**
- It's **inefficient** - e.g. with batch size *3*B*, we use only *B* triplets, but there are

Online negative mining!

First **extract batch of embeddings**, then **select informative tuples**

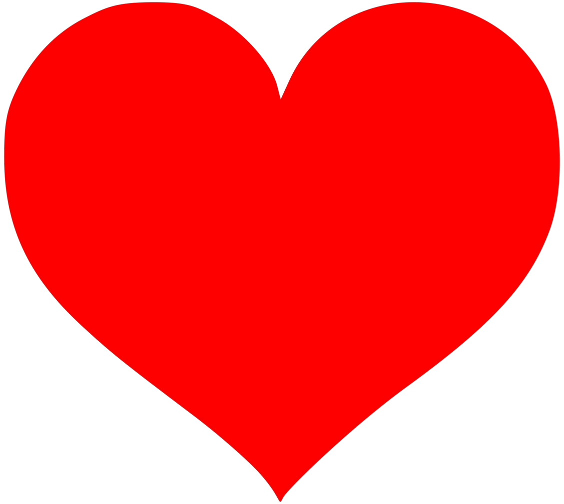
1. Sample a **minibatch** (e.g. *N* classes, *M* samples from each class)
2. **Extract** embeddings
3. Find **hard** pairs/triplets **within** a minibatch
4. Compute **loss** for them
5. Backpropagate

Online negative mining!



TOOPLOOX

http://bamos.github.io/2016/01/19/openface-0.2.0/



PYTORCH

https://github.com/adambielski/siamese-triplet/

Online Contrastive Loss

```
class OnlineContrastiveLoss(nn.Module):

    def __init__(self, margin, pair_selector):
        super(OnlineContrastiveLoss, self).__init__()
        self.margin = margin
        self.pair_selector = pair_selector

    def forward(self, embeddings, target):
        positive_pairs, negative_pairs = self.pair_selector.get_pairs(embeddings, target)
        if embeddings.is_cuda:
            positive_pairs = positive_pairs.cuda()
            negative_pairs = negative_pairs.cuda()
        positive_loss = (embeddings[positive_pairs[:, 0]] - embeddings[positive_pairs[:, 1]]).pow(2).sum(1)
        negative_loss = F.relu(
            self.margin - (embeddings[negative_pairs[:, 0]] - embeddings[negative_pairs[:, 1]]).pow(2).sum(
                1).sqrt()).pow(2)
        loss = torch.cat([positive_loss, negative_loss], dim=0)
        return loss.mean()
```

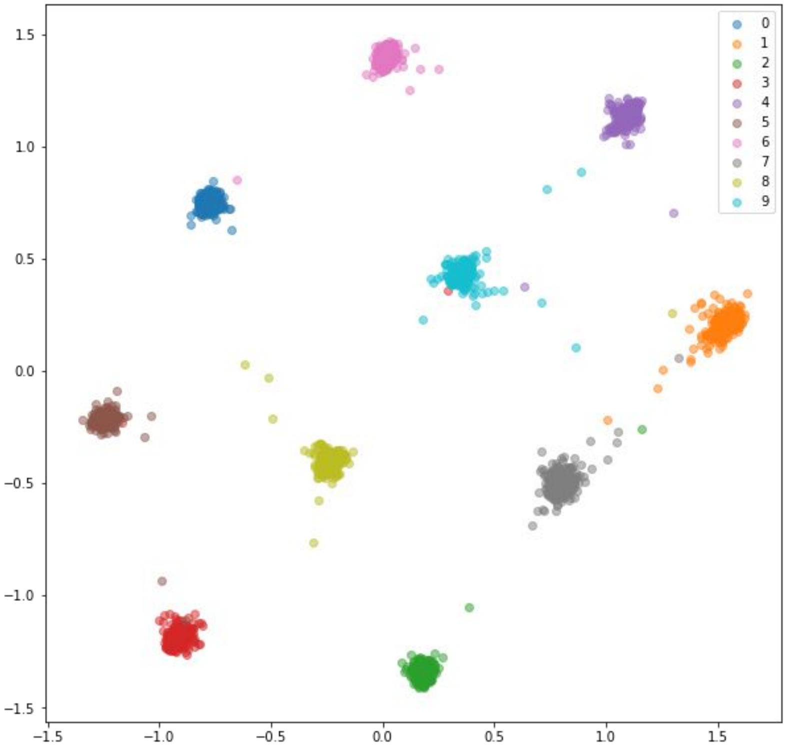
Pair selectors

```
class PairSelector:
    def __init__(self):
        pass

    def get_pairs(self, embeddings, labels):
        raise NotImplementedError

Implementations: AllPositivePairSelector, HardNegativePairSelector

class HardNegativePairSelector(PairSelector):
    def __init__(self):
        super(HardNegativePairSelector, self).__init__()

    def get_pairs(self, embeddings, labels):
        distance_matrix = pdist(embeddings)
        (...) // select pairs based on distances in minibatch
```

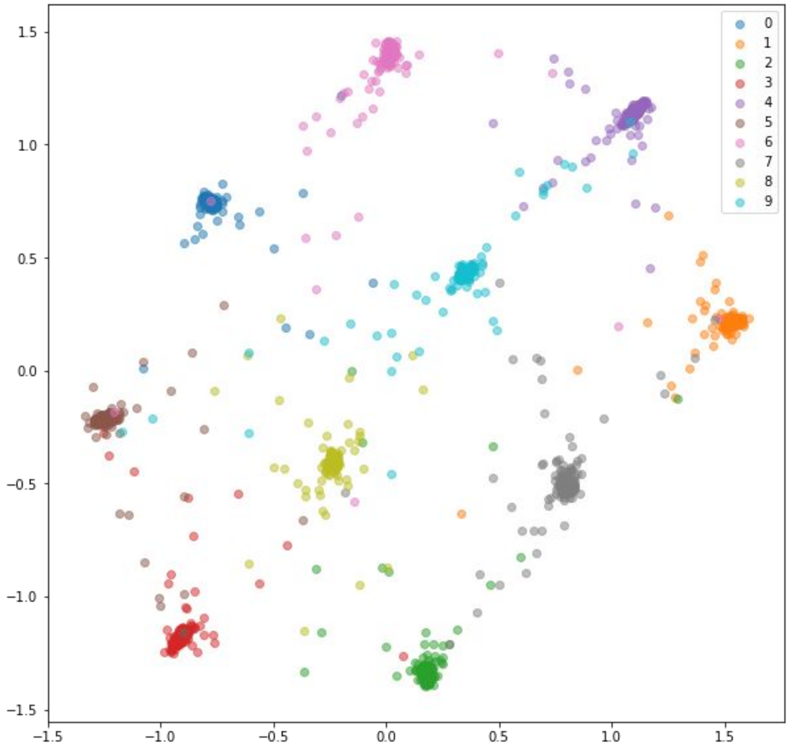
Training

```
from datasets import BalancedBatchSampler
train_batch_sampler = BalancedBatchSampler(train_dataset, n_classes=10, n_samples=25)
test_batch_sampler = BalancedBatchSampler(test_dataset, n_classes=10, n_samples=25)

online_train_loader = torch.utils.data.DataLoader(train_dataset, batch_sampler=train_batch_sampler)
online_test_loader = torch.utils.data.DataLoader(test_dataset, batch_sampler=test_batch_sampler)

margin = 1.
embedding_net = EmbeddingNet()
model = embedding_net

loss_fn = OnlineContrastiveLoss(margin, HardNegativePairSelector())
lr = 1e-3
optimizer = optim.Adam(model.parameters(), lr=lr)
scheduler = lr_scheduler.StepLR(optimizer, 8, gamma=0.1, last_epoch=-1)
n_epochs = 20
```

Siamese online mining - visualization



Train



Validation

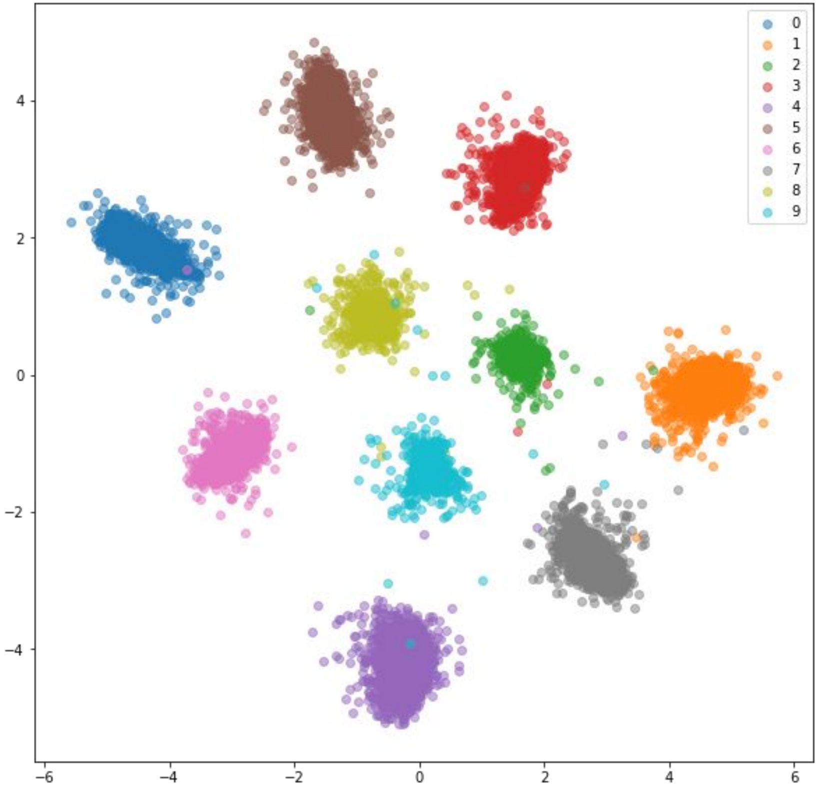
Online Triplet Loss

```
class OnlineTripletLoss(nn.Module):

    def __init__(self, margin, triplet_selector):
        super(OnlineTripletLoss, self).__init__()
        self.margin = margin
        self.triplet_selector = triplet_selector

    def forward(self, embeddings, target):
        triplets = self.triplet_selector.get_triplets(embeddings, target)

        ap_distances = (embeddings[triplets[:, 0]] - embeddings[triplets[:, 1]]).pow(2).sum(1)
        an_distances = (embeddings[triplets[:, 0]] - embeddings[triplets[:, 2]]).pow(2).sum(1)
        losses = F.relu(ap_distances - an_distances + self.margin)
        return loss.mean(), len(triplets)
```

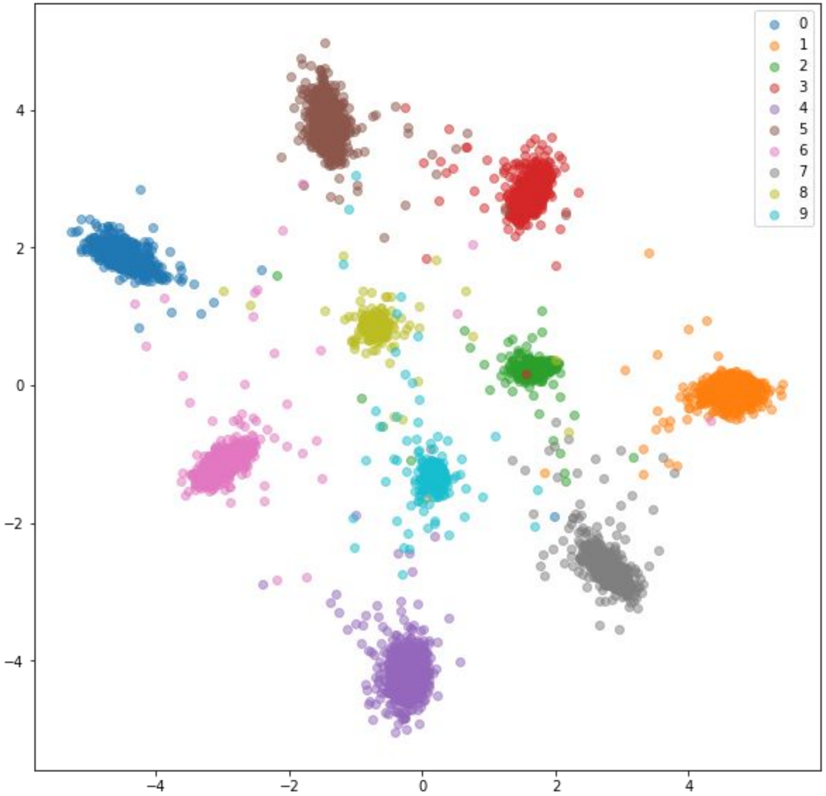
Triplet selectors

```
class TripletSelector:
    def __init__(self):
        pass

    def get_triplets(self, embeddings, labels):
        raise NotImplementedError

Implementations: AllTripletSelector, HardestNegativeTripletSelector, RandomNegativeTripletSelector,
SemihardNegativeTripletSelector

class RandomNegativeTripletSelector(PairSelector):
    def __init__(self):
        super(HardNegativePairSelector, self).__init__()

    def get_pairs(self, embeddings, labels):
        distance_matrix = pdist(embeddings)
        (...) // for every possible positive pair, a random hard negative is chosen
```

Triplet network online mining - visualization

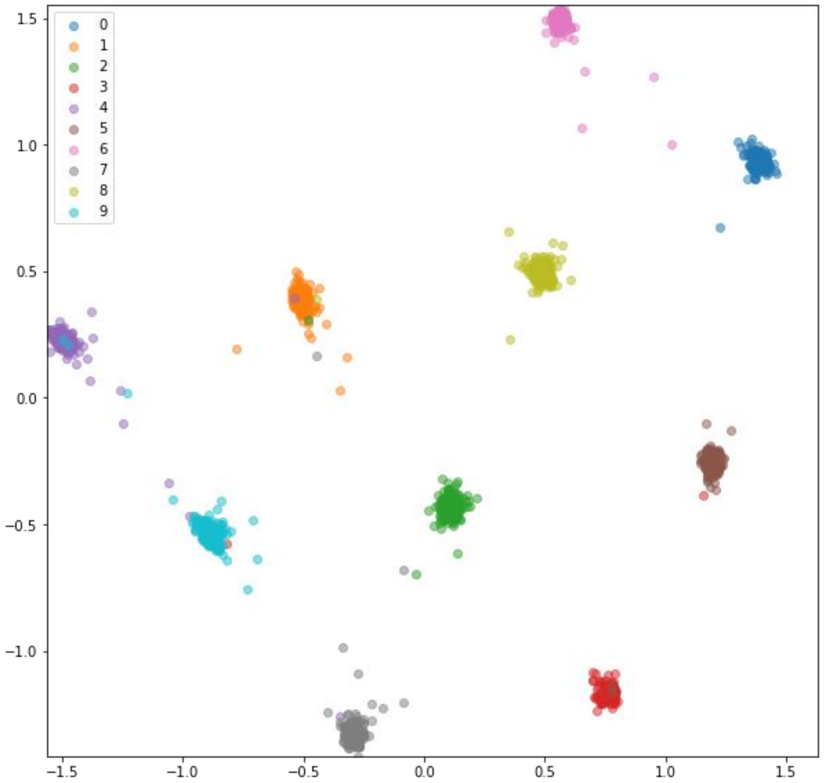


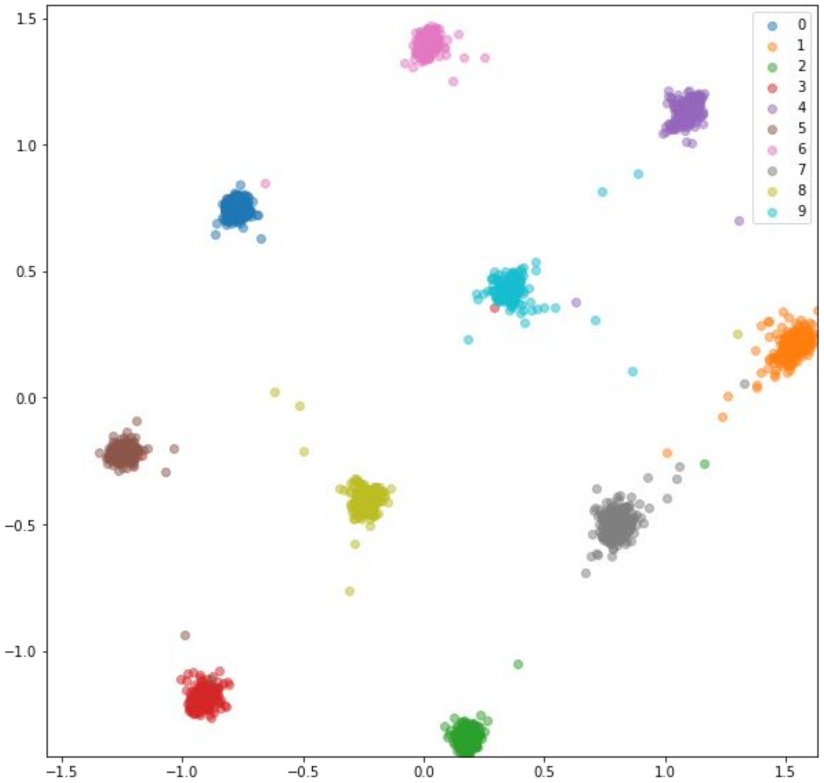
Train



Validation

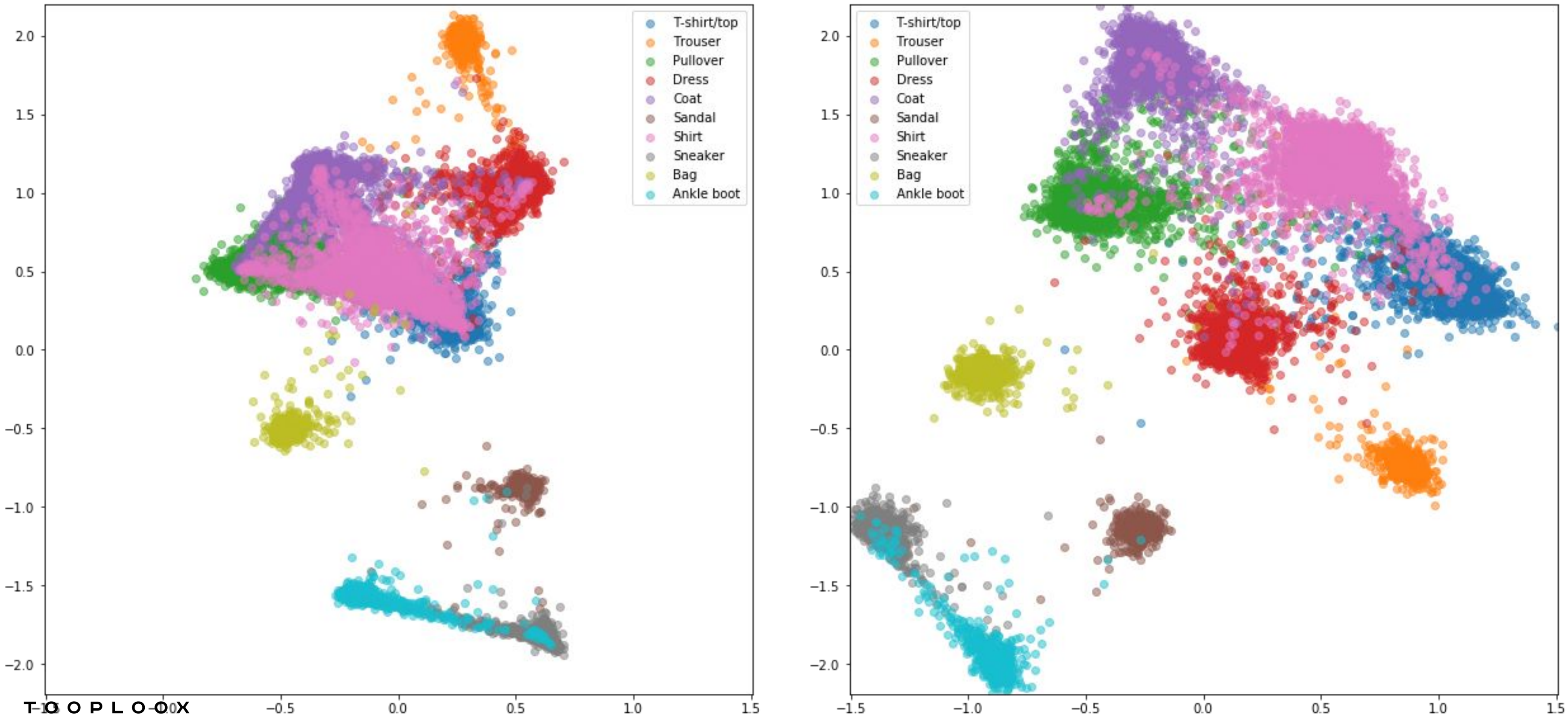
MNIST Siamese - classical vs online mining





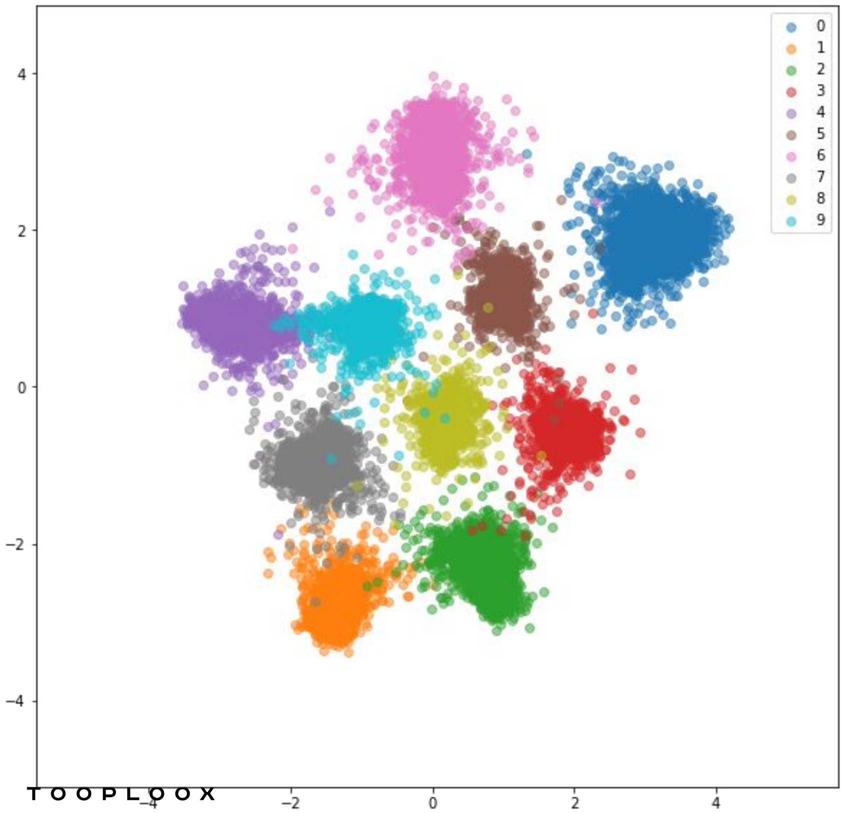
TOOPLOOX

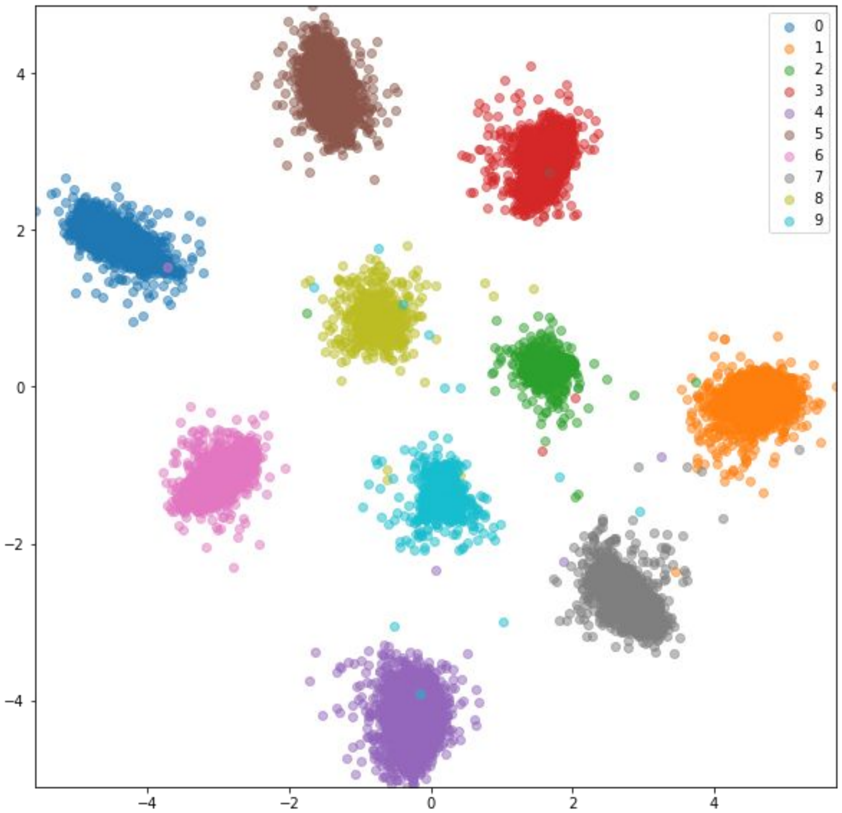
FashionMNIST Siamese - classical vs online mining



TOOPLOOX

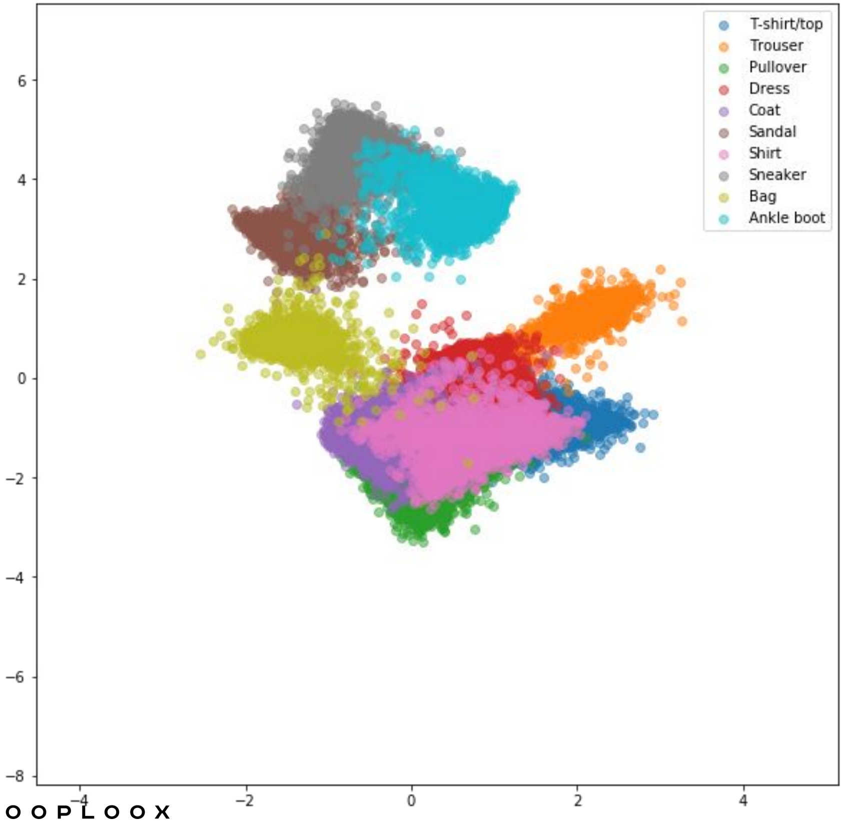
MNIST Triplet - classical vs online mining

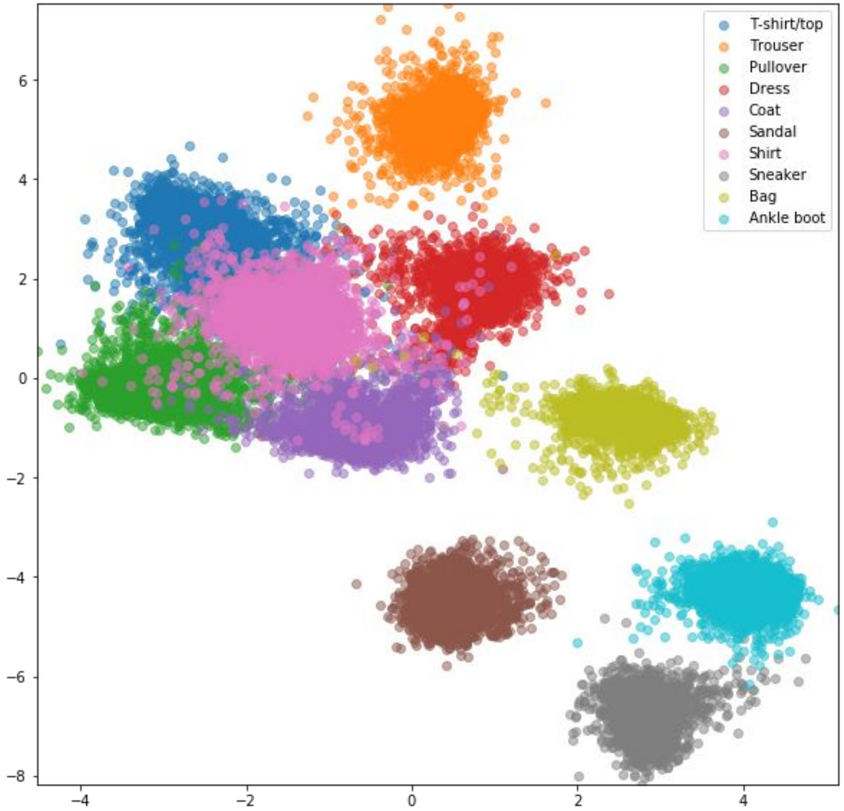




TOOPLOOX

FashionMNIST Triplet - classical vs online mining





TOOPLOOX

Thank you!

Check out our job offers

tooplox.com/jobs

Adam Bielski

adam.bielski@tooploox.com

@BielskiAdam

TOOPLOOX