Обучение представлений

Введение

Хоть тема с Triplet функцией ошибок для нейронных сетей и не является новой, однако данный подход к обучению моделей не очень распространен. В данном реферате будут рассмотрены архитектуры Сиамской и Триплет нейронных сетей, что лежит в основе данного подхода, в рамках реферата будет реализована Триплет нейронная сеть, а также будет продемонстрирован результат ее работы в качестве модели эмбеддингов для задачи классификации цифр из набора MNIST

Описание аритектур нейронных сетей

Сиамская нейронная сеть

Для того, чтобы рассмотреть архитектуру Триплет нейронной сети, следует сначала ознакомиться с ее более упрощенной версией - Сиамской нейронной сети.

> **Каноническая задача для данной архитектуры**
>
> *Допустим, нам нужно сделать модель распознавания лиц для организации, в которой работает около 500 человек. Если делать такую модель с нуля на основе свёрточной нейросети (Convolutional Neural Network (CNN)), то для обучения модели и достижения хорошей точности распознавания нам понадобится много изображений каждого из этих 500 человек. Но очевидно, что такой датасет нам не собрать, поэтому не стоит делать модель на основе CNN или иного алгоритма глубокого обучения, если у нас нет достаточного количества данных. В подобных случаях можно воспользоваться сложным алгоритмом однократного обучения, наподобие сиамской сети, которая может обучаться на меньшем количестве данных.*

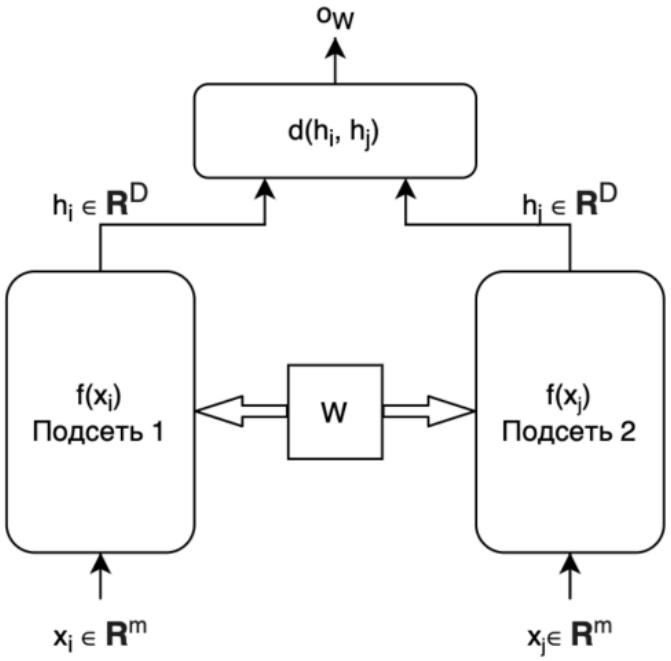
Для решения данной задачи, как уже упоминалось выше, нам подойдет сиамская нейронная сеть. Она будет учиться сравнивать пары изображений и выдавать вероятность их сходства, с помощью чего мы сможем понять является ли человек, находящийся на фотографии реальным сотрудником компании или же незнакомым нам.

Архитектура нейронной сети

Рассмотрим обобщенную архитектуру сиамской нейронной сети. В общем случае сиамская сеть состоит из двух подсетей, выходы которых подаются на вход другого модуля, который генерирует конечный выход.

Рассмотрим рисунок, представленный ниже, на котором x_i и x_j — это входы, W — общие веса/параметры, а $h_i \in R^D$ и $h_j \in R^D$ — выходы обеих подсетей.
Сиамская сеть представляет собой отображение $h_i = f(x_i)$, для которого Евклидово расстояние $d(h_i, h_j)$ максимально мало при $y_i = y_j$ и максимально велико при $yi \neq yj$.

Сеть возвращает оценку o_W того, насколько различны x_i и x_j



На практике архитектура сиамской нейронной сети не будет так сильно отличаться от обычной. Решающим фактором в данном случае будет являться функция потерь, благодаря которой модель сможет учиться находить схожие объекты.

Мы рассмотрим классический для данной архитектуры **Contrastive Loss**, математически его можно выразить как:

$$L(x_i, x_j, z_{ij}) = (1 - z_{ij})\|h_i - h_j\|_2^2 + z_{ij}\max(0, \tau - \|h_i - h_j\|_2^2)$$

Где:

- $L(x_i, x_j, z_{ij})$ - это потеря между x_i и x_j с указанием метки z_{ij}, которое принимает значение 1 если пара (x_i, x_j) - положительный пример, 0 - если отрицательный
- τ - это пороговое значение, в контрастивной функции потерь используется для определения расстояния, на котором модель считает, что две точки данных являются различными.
- $\|h_i - h_j\|_2^2$ - это квадрат евклидова расстояния между эмбеддингами двух объектов h_i и h_j. Оно используется для оценки схожести между парами объектов: **позитивные пары** (одного класса) должны быть ближе друг к другу, в то время как **негативные пары** (разных классов) должны находиться на большем расстоянии.

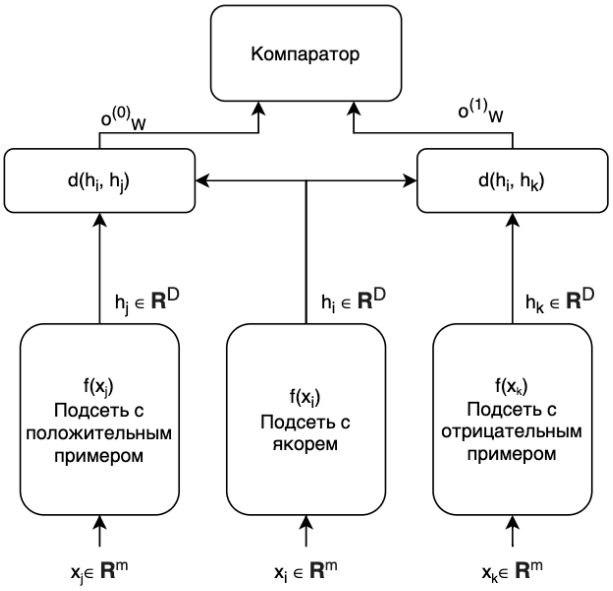
На этом особенности архитектуры заканчиваются. Сама сеть может иметь подсети любого вида, да и область задач не ограничивается только лишь изображениями. Теперь можем перейти к следующей архитектуре.

Триплет нейронная сеть

Триплет нейронная сеть, хоть и используется для решения схожих типов задач, пошла немного дальше, она позволяет представлять информацию в виде эмбеддингов, схожесть которых можно измерить например косинусным сходстовом или евклидовым расстоянием.

Архитектура нейронной сети

Triplet имеет фиксированное количество подсетей. Их в данном случае три, архитектура подсети может быть в свою очередь любой(в разумном контексте), в каноническом варианте подсети являются сверточными сетями.



Компоненты Triplet Loss

1. Якорь (a): Это эталонный входной образец.
2. Положительный (p): Этот образец имеет ту же метку, что и якорь, и должен быть близок к нему в пространстве встраивания.
3. Отрицательный (n): Этот образец имеет другую метку по сравнению с якорем и должен находиться дальше от него в пространстве встраивания.

Triplet Loss можно математически выразить следующим образом:

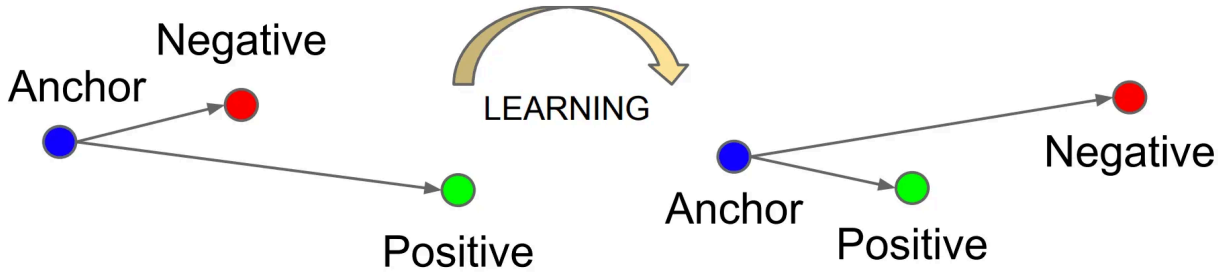
$$L(a, p, n) = \max(d(a, p) - d(a, n) + \tau, 0)$$

Где:

- $L(a, p, n)$ - это потеря триплета для якоря a, положительного объекта p и отрицательного объекта n.
- $d(x_i, x_j)$ - это метрика расстояния (обычно используется евклидово расстояние) между двумя образцами (x_i, x_j)
- τ - это пороговое значение, который обеспечивает желаемое разделение между расстояниями пар якорь-положительный и якорь-отрицательный.

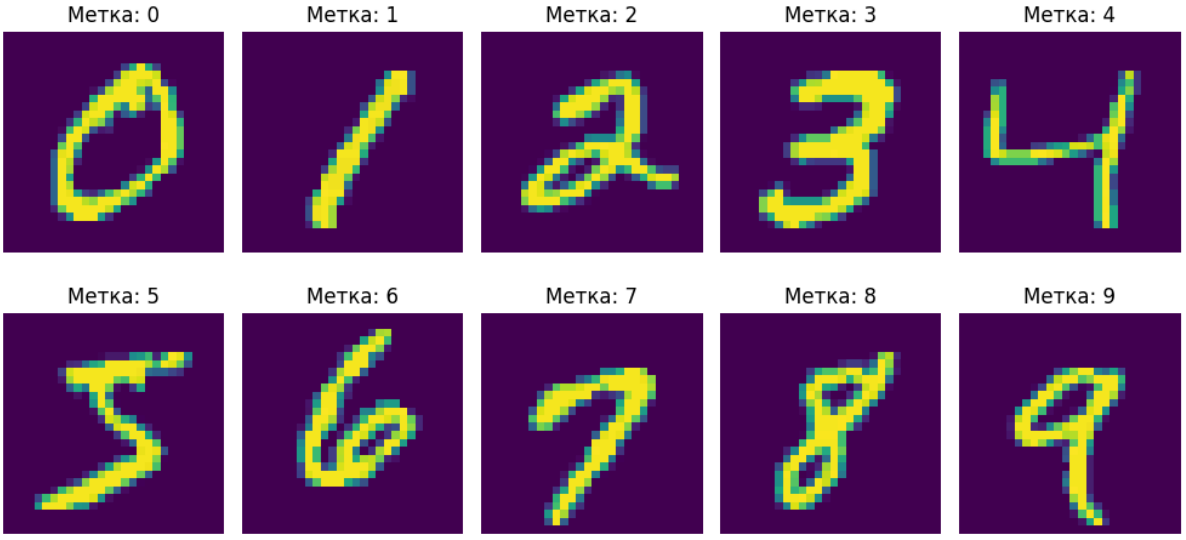
Его цель заключается в минимизации расстояния между парой **якорь-положительный** при одновременном максимизации расстояния между парой **якорь-отрицательный**. В частности, она стремится поддерживать запас τ так, чтобы выполнялось следующее условие:

$$d(a, p) + \tau < d(a, n)$$



Реализация

Реализовывать мы будем Триплет нейронную сеть на Pytorch, в качестве набора данных мы возьмем MNIST10.

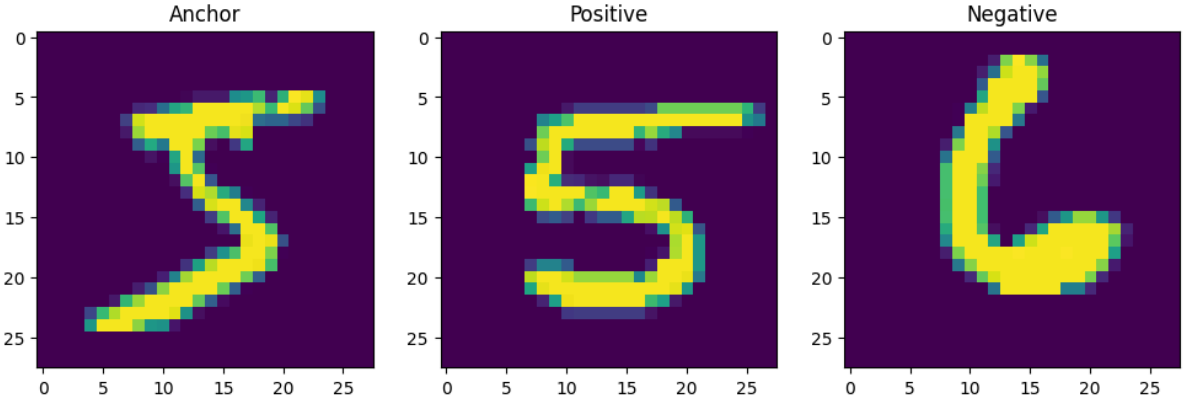


мне цветовая палитра viridis очень уж нравится, так что буду ее использовать

Создание триплетов

К созданию триплетов существует множество подходов, как создание полностью случайно, так и с объединением в один триплет сложных примеров, для сложных примеров в триплет помещаются якорь, положительный пример того же класса, но с большой дистанцией от якоря, отрицательный пример другого класса, но с маленькой дистанцией до якоря, однако мы не будем рассматривать такое сложное формирование набора данных и составим триплеты с помощью случайных подвыборок.

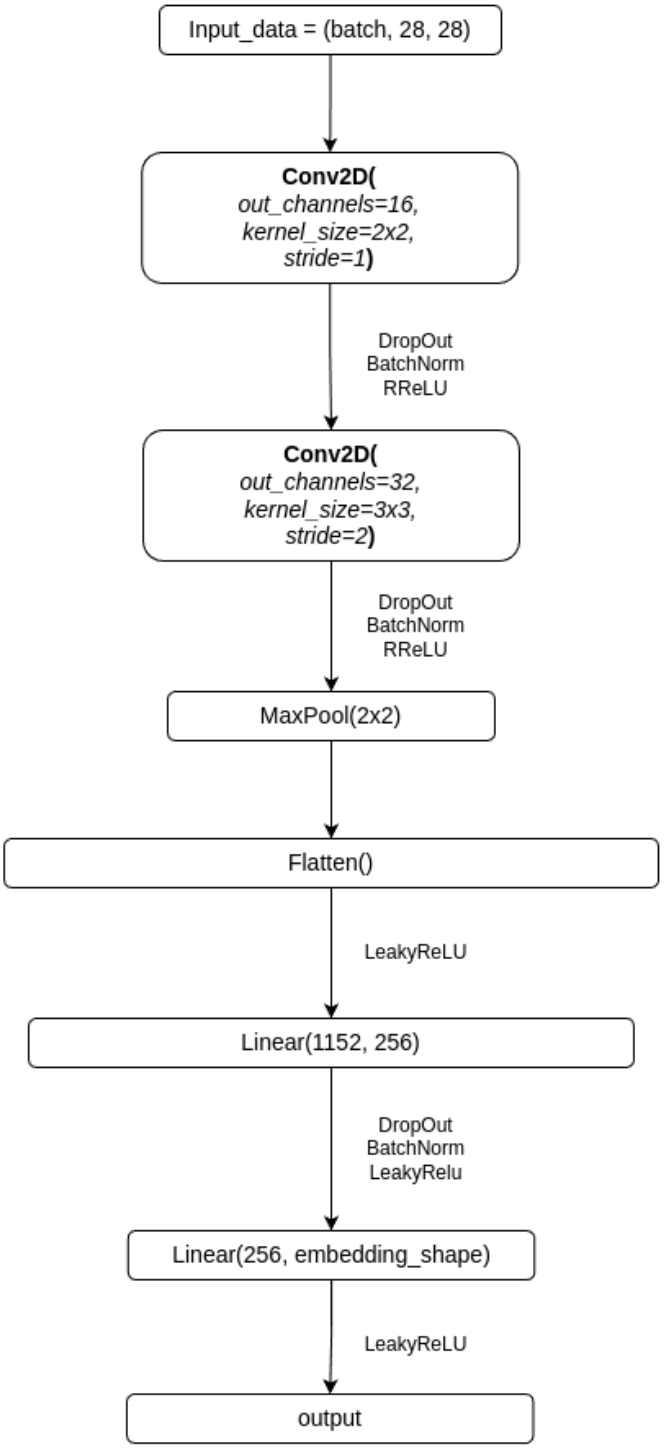
Реализация данного пункта находится в классе TripletDataset. На выходе мы получаем датасет, состоящий из якоря, позитивного и негативного примеров, а также метку класса якоря. Визуализируем один набор триплетов:



На основе датасетов были собраны DataLoader-ы train и test, batch выбран равным 32,

Определим архитектуру модели

Я буду использовать сверточную нейронную сеть CNN как основу для моделей подсети триплета, после чего буду разворачивать свертки к вектору и приводить к эмбеддингу выбранного размера.
Ниже представлен граф с архитектурой модели:

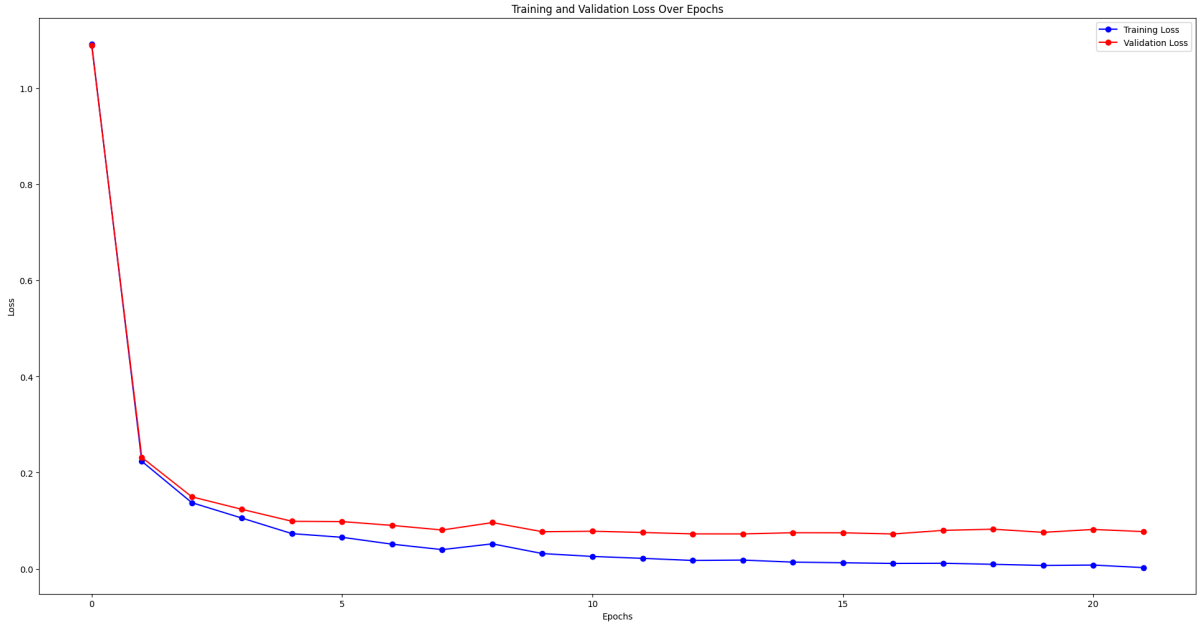


Обучение модели

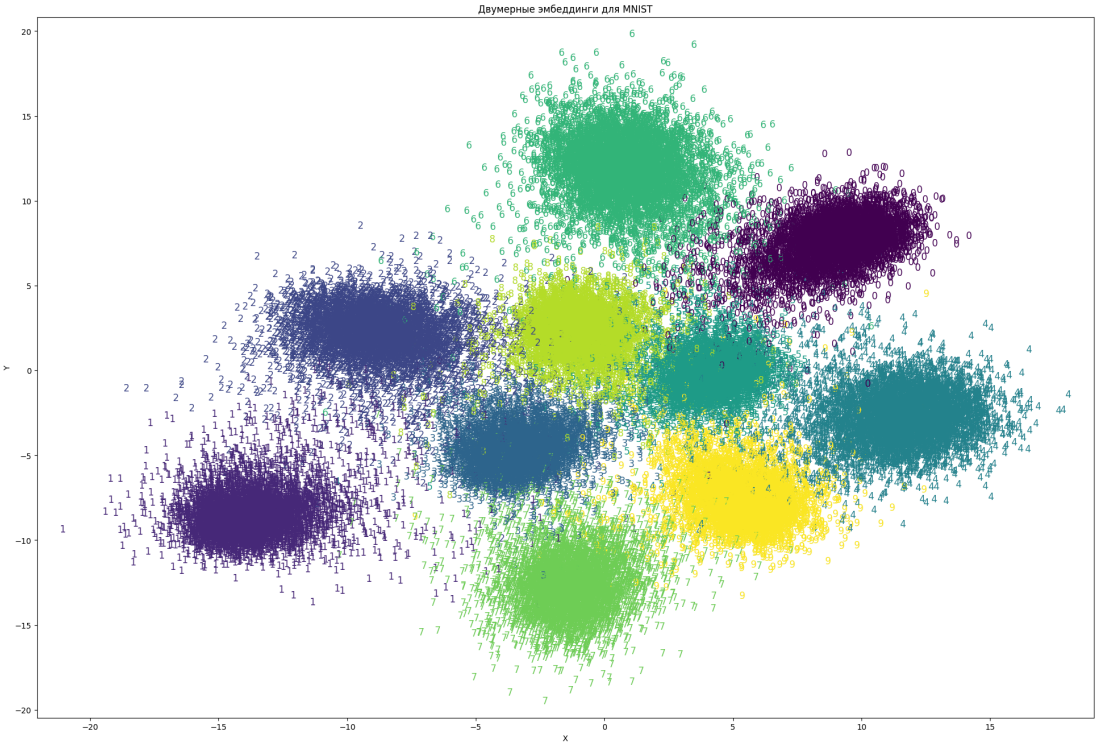
Двумерные эмбеддинги

Для начала обучим модель на формирование двумерных эмбеддингов для более наглядной визуализации значений.
Параметры обучения:

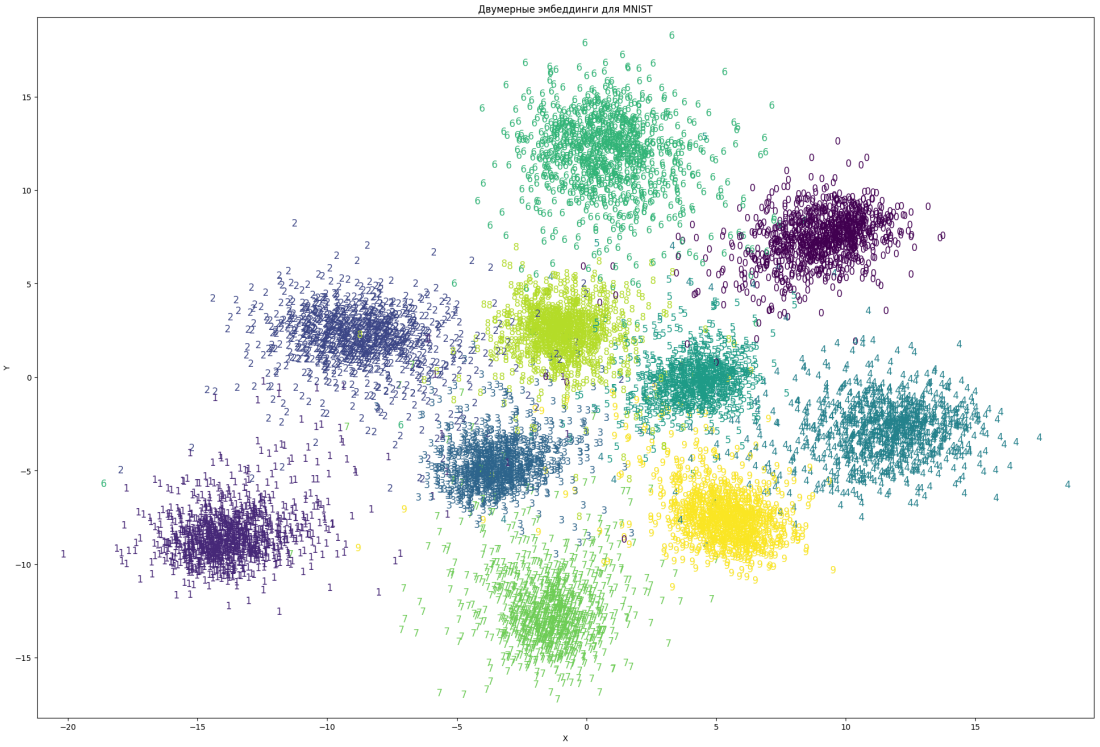
- LossFunction - TripletMarginLoss(margin=1.5)
- Optimizer - Adam
- LearningRate - 1e-4
- patience - 5 для EarlyStopping
- patience - 3 для LrScheduler
- LrScheduler - ReduceLROnPlateau(factor=0.5, patience=3)
- Epochs - 30
  EarlyStopping сработал на 21 эпохе, выведем визуализацию двумерных эмбеддингов для MNIST



Train Loader



Test Loader



Как можно увидеть, модель достаточно хорошо собрала цифры в кластеры. Возможно при формировании более сложных триплетов или настройке гиперпараметров, можно будет достичь лучшего результата

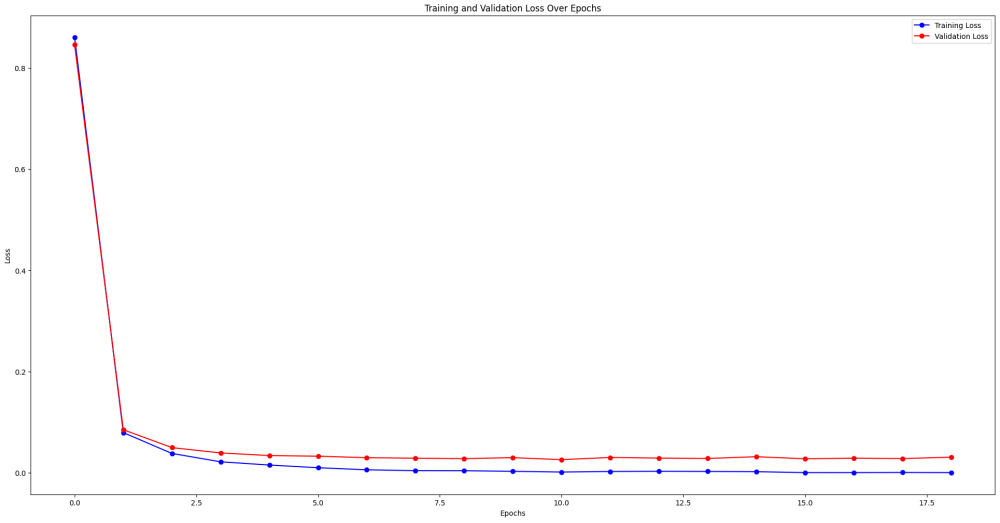
Многомерные эмбеддинги

Теперь обучим модель создавать эмбеддинги размером 64, параметры обучения:

Параметры обучения:

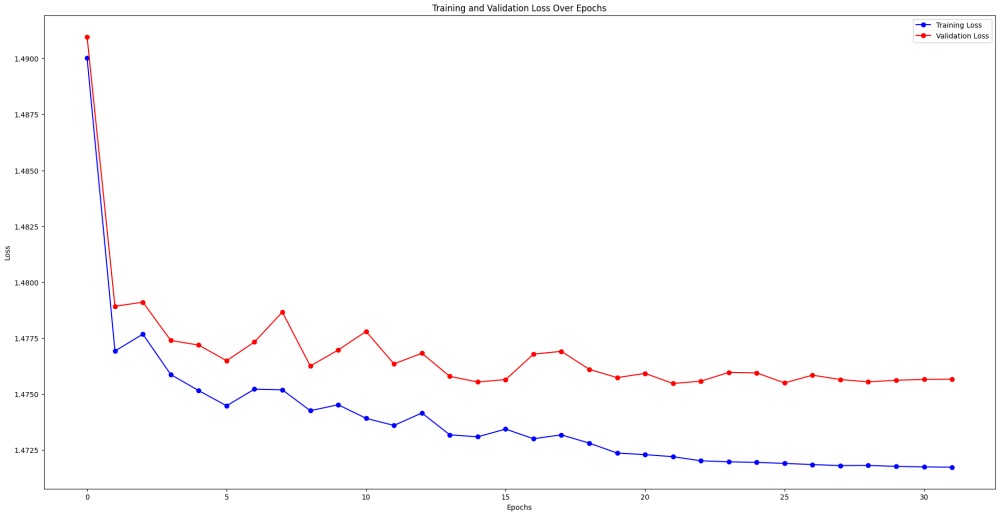
- LossFunction - TripletMarginLoss(margin=1.5)
- Optimizer - Adam
- LearningRate - 1e-4
- patience - 8 для EarlyStopping
- patience - 3 для LrScheduler
- LrScheduler - ReduceLROnPlateau(factor=0.5)
- Epochs - 50

EarlyStopping сработал на 18 эпохе



Теперь с помощью обученной модели сформируем набор данных, который будет содержать эмбеддинги и метки. После чего создадим простой линейный многоклассовый классификатор. Его архитектуру приводить не будем, так как это не является предметом данного реферата.

График значений Loss функции для многоклассовой классификации:



Результаты тестирования

Classififcation Report для тестового набора:

Class	Precision	Recall	F1-Score	Support
0	0.98	0.99	0.99	980
1	0.99	1.00	0.99	1135
2	0.98	0.99	0.99	1032
3	0.98	0.99	0.99	1010
4	0.98	0.99	0.99	982
5	0.99	0.99	0.99	892
6	0.99	0.98	0.99	958
7	0.98	0.98	0.98	1028
8	0.98	0.98	0.98	974
9	0.99	0.97	0.98	1009
Accuracy			**0.99**	**10000**
Macro Avg	**0.99**	**0.99**	**0.99**	**10000**
Weighted Avg	**0.99**	**0.99**	**0.99**	**10000**

Confussion Matrix: