GITHUB: https://github.com/nazar1ous/SOP_Metric_Learning.git
Models architectures:

Baseline feature extractor - Resnet18

```
self.resnet18 = models.resnet18(pretrained=True)

self.feature_extractor = torch.nn.Sequential(
    self.resnet18.conv1,
    self.resnet18.bn1,
    self.resnet18.relu,
    self.resnet18.maxpool,

    self.resnet18.layer1,
    self.resnet18.layer2,
    self.resnet18.layer3,
    self.resnet18.layer4,

    self.resnet18.avgpool
)
```

All of the classifier architectures (

Fine-tuned with vanilla Cross-Entropy and classification approach)

, will have a slight changes:

```
self.model = torch.nn.Sequential(
    self.feature_extractor,
    torch.nn.Linear(self.feature_extractor.last_layer_dim, num_classes)
)
```

Data Augmentation:

Horizontal flip, RandomBrightnessContrast. Also, normalization is used from ImageNet

```
self.norm = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize(mean=[0.485, 0.456, 0.406],
                          std=[0.229, 0.224, 0.225])
     ])

self.transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
])
```

Training process:

Train/valid split ratio is 0.8/0.2.

Optimizers, and scheduler:

```
def configure_optimizers(self):
    self.optimizer = torch.optim.Adam(self.parameters(), lr=1e-3)
    self.scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(self.optimizer, patience=self.patience)
    return {"optimizer": self.optimizer,
            "lr_scheduler": self.scheduler,
            "monitor": "valid_accuracy"}
```

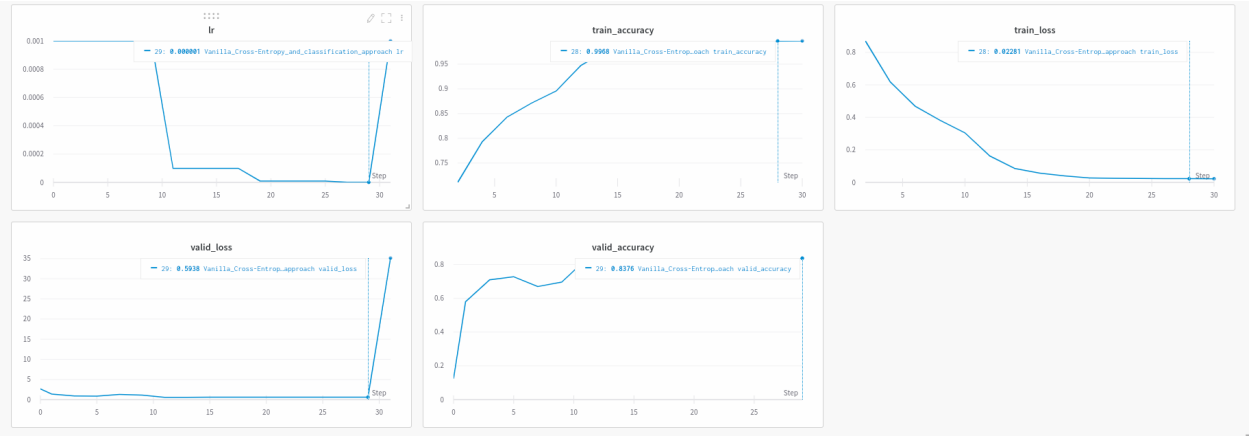
Of course, one can use miners, reducers for losses (but, this is far beyond the scope of the homework)

Notes, before proceeding:
Also, as retrieval task metrics can be calculated, by using index (for example, using annoy table), It takes too much time to do it for each validation step! (model changes as training step goes, and it makes recalculate the index each step).
So, I did it once, in the final step!

However, for classifiers, I plot the accuracy metric in wandb.
Also, I train the classifiers on 12 super classes (instead of many classes, because of cuda memory constraint).

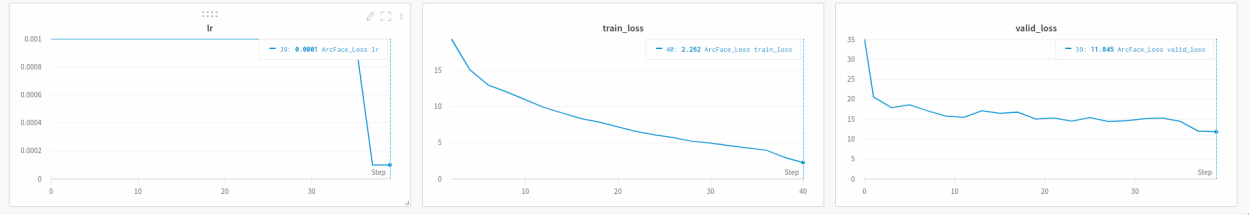
WANDB Training results:

Fine-tuned with vanilla Cross-Entropy and classification approach



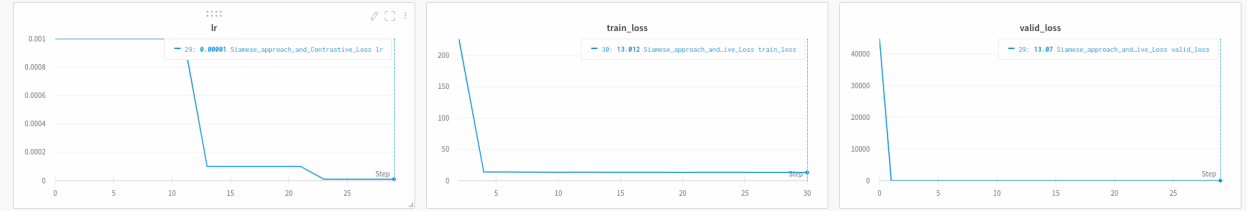
As one can see, we stick to overfitting (train_accuracy vs val_accuracy), however 85% for such a light backbone is still ok for me. It is possible to add more batch normalization, or MaxPool, however I had no extra time. The same (relatively small) overfitting one can see on each next training.

Fine-tuned using ArcFace Loss



This network is not fully converged, and can be future trained more.

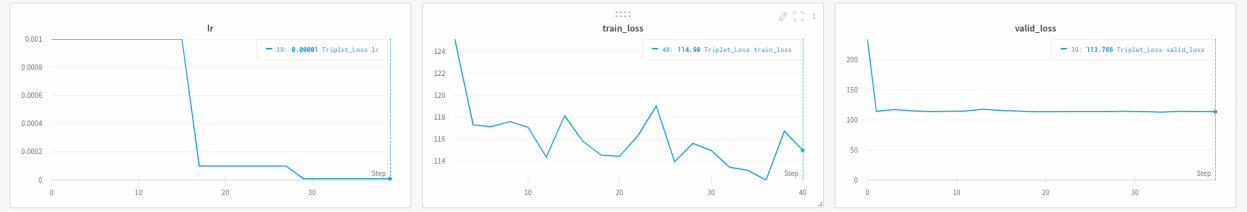
Fine-tuned using the Siamese approach and Contrastive Loss



Because of the light backbone, small number of classes (12), and two models initializing: EMB1 = model(anchor), EMB2 = model(neg/pos).

Backward pass: Loss(EMB1, EMB2).backward()

[optional] Fine-tuned using Triplet Loss



The same is here :)

Retrieval Results (on the test): All visualizations are for the first class to be able easily compare results. In case of need, it is possible to show visualizations for each class. The metrics are computed for the whole test dataset.

Plain pre-trained ImageNet backbone

```
{'precision': 0.041666666666666664, 'recall': 0.08333333333333333, 'f1-score': 0.05555555555555555, 'accuracy': 0.05263157894736842}
Classes MAP@5 = 9.67231e-07
Super Classes MAP@5 = 0.008342
```

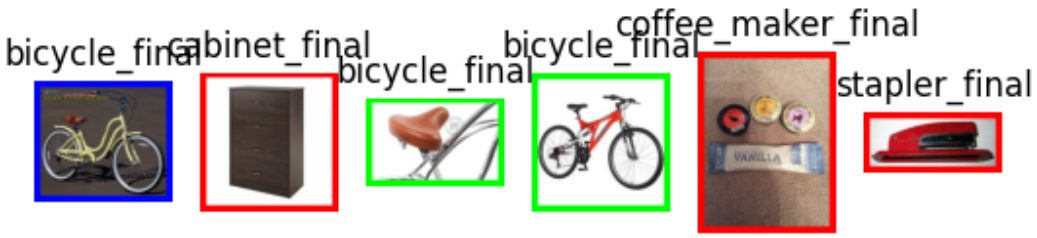
Fine-tuned with vanilla Cross-Entropy and classification approach

```
{'precision': 0.09090909090909091, 'recall': 0.09090909090909091, 'f1-score': 0.09090909090909091, 'accuracy': 0.07142857142857142}
Classes MAP@5 = 8.6982e-06
Super Classes MAP@5 = 0.009642
```


bicycle_final
bicycle_final
lamp_final
sofa_final
sofa_final
sofa_final

Fine-tuned using ArcFace Loss

```
{'precision': 0.2997002997002997, 'recall': 0.2244155844155844, 'f1-score': 0.22840909090909092, 'accuracy': 0.14285714285714285}
Classes MAP@5 = 0.000139901
Super Classes MAP@5 = 0.0192323
```



Fine-tuned using the Siamese approach and Contrastive Loss

```
{'precision': 0.3974025974025974, 'recall': 0.3603896103896104, 'f1-score': 0.36511593498340766, 'accuracy': 0.4107142857142857}
Classes MAP@5 = 0.0001177738
Super Classes MAP@5 = 0.2492323
```

[optional] Fine-tuned using Triplet Loss:

```
{'precision': 0.5, 'recall': 0.5454545454545454, 'f1-score': 0.5151515151515151, 'accuracy': 0.6}
Classes MAP@5 = 0.0002587345
Super Classes MAP@5 = 0.582145
```

Conclusion:

1. Pain pretrained resnet18 did not show good results, because the ImageNet dataset drastically differs from SOP data, and it was trained on ImageNet. So, the features are not so representative for SOP data.
2. The fine-tuned version of this network shows pretty good results on the classification for super classes. Also, it is more prone to overfitting than future approaches, because of trying to find the representative vector for data, not by explicitly classifying it.
3. As it was expected the next networks show superior results on our retrieving task, at least, because they were constructed for it (ArcFace, Siamese Approach and Contrastive Loss, Triplete Loss).
4. The feature extractor, which was trained on Triplete Loss, showed the best results with respect to MAP@5, and accuracy score.