Project report

Painters by Number

Noor Abo El Foul

Goal:

The goal of this project is to predict if a pair of images are artworks made by the same artist or not.

Data:

For this project I used a dataset from Kaggle's Painter by the Numbers challenge.

I used different artists in the train and test data because I wanted to check how the model generalizes to unseen artists.

I used the csv files from Kaggle's and created csv files for train and test that have different artists, we submitted them.

For training and validation data I used 17,997 different images, that belongs to 1,208 different artists, I divided it such that 772 different artists in the training set and 190 different artists for the validation set (I get rid of the artists that have less than 2 images).

For test data, I used 2936 images that belongs to 91 unique artists (all the artists that have at least 2 paints).

In train, validation and test I make sure that the artists are completely different.

From the data we took 10 images for each artist, (if an artist has less than 10 images then I took them all) after that, I generated 30,000 triple (anchor, positive, negative) randomly for the training, I generated the 30,000 triples each epoch in order to prevent Overfitting and train the model on bigger data, I generated 6,000 (20% of 30,000) triple randomly for the validation and test **each**.

(**Note**: The images I used were very big so I applied center crop with size (224, 224) on each image.)

Pre-Processing:

I downloaded the data and created csv files(creating_csv.py) as I explained before, because of the size of the data I couldn't upload it to Drive so I worked on pycharm(pick_and_crop.py) in order to center crop the images with size 224x224 and save them to zip file.

Network Architecture:

The architecture of the Siamese Network that we used consists of a pre-trained ResNet-18 model with the last two layers (average pooling and fully connected layer) removed,

followed by a dropout layer with a dropout rate of 0.25, a fully connected layer with 256 output features, another dropout layer with a dropout rate of 0.30, and finally a fully connected layer with 64 output features.

1. Input: Images passed through a ResNet-18.

2. ResNet-18 features: Extracted using all but the last two layers of the ResNet-18 model.

3. Dropout: Regularization with a dropout rate of 0.25 applied to the extracted features.

4. Fully Connected Layer 1: 25088 input features (flattened from the ResNet output) mapped to 256 features.

5. Dropout: Regularization with a dropout rate of 0.30 applied to the output of the first fully connected layer.

6. Fully Connected Layer 2: 256 input features mapped to 64 output features.

Loss Function: **Triplet Loss**, the goal of triplet loss is to minimize the distance between the anchor and the positive images while raising the gap between the anchor and the negative images.

Optimizer: optim.Adam(siamese_net.parameters(), lr=0.00001,weight_decay=0.001).

Margin (for Triplet loss): 5.

Batch size = 200.

Learning rate = 0.00001.

Weight_decay = 0.001.

```
class SiameseNetwork(nn.Module):
def __init__(self):
      super(SiameseNetwork, self).__init__()
      resnet = models.resnet18(pretrained=True)
      self.resnet_features =
      nn.Sequential(*list(resnet.children())[:-2])
      self.dropout1 = nn.Dropout(0.25)
      self.fc1 = nn.Linear(25088, 256)
      self.dropout2 = nn.Dropout(0.30)
      self.fc2 = nn.Linear(256, 64)
def forward_once(self, x):
      x = self.resnet_features(x)
      x = x.view(x.size(0), -1)
      x = self.dropout1(x)
      x = F.relu(self.fc1(x))
      x = self.dropout2(x)
      x = self.fc2(x)
      return x
def forward(self, input1):
      output1 = self.forward_once(input1)
return output1
```

```
def TripletLoss(anchor ,positive , negative,margin ):
      pos = torch.norm(anchor - positive, p=2, dim=1)
      neg = torch.norm(anchor - negative, p=2, dim=1)
      loss_values = torch.relu(pos - neg + margin)
      return torch.mean(loss_values), pos, neg
```

*Transformations that I applied on the training data made the accuracy for the training lower in order to prevent Overfitting and do a better generalization to unseen data.

transform_n we applied on training data.

val_transform we applied on validation and test data.

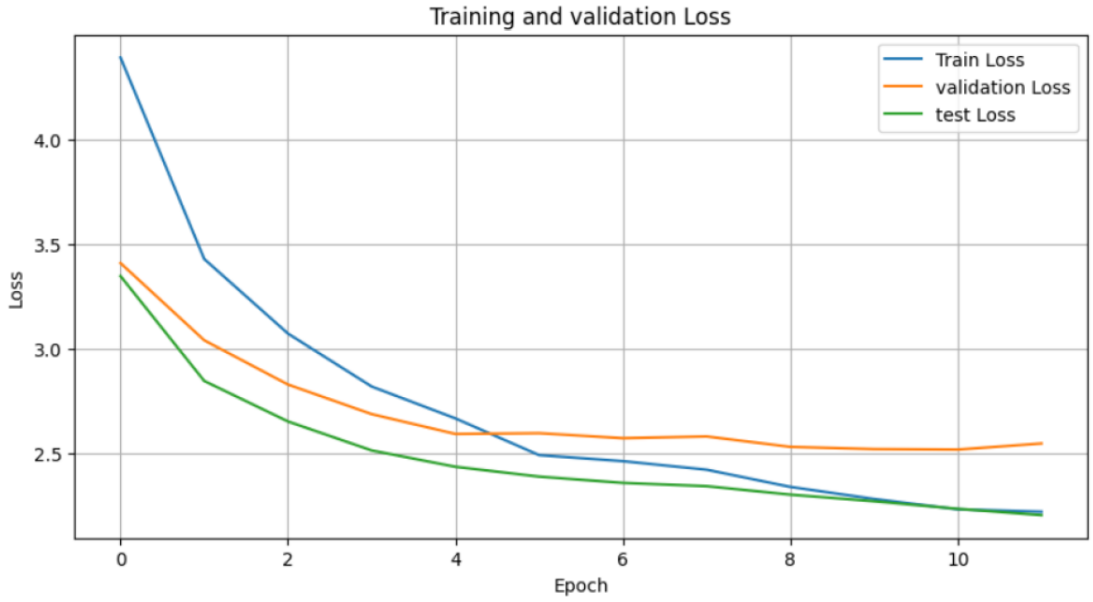
```
transform_n = transforms.Compose([
transforms.RandomRotation(degrees=15),
transforms.RandomVerticalFlip(),
```

```
transforms.RandomHorizontalFlip(),
transforms.RandomAffine(degrees=15, scale=(0.8, 1.2),
shear=10),
transforms.RandomPerspective(distortion_scale=0.6,
p=0.2),
transforms.ToTensor(),
transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225]),
])
val_transform = transforms.Compose([
transforms.ToTensor(),
transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225]),
])
```

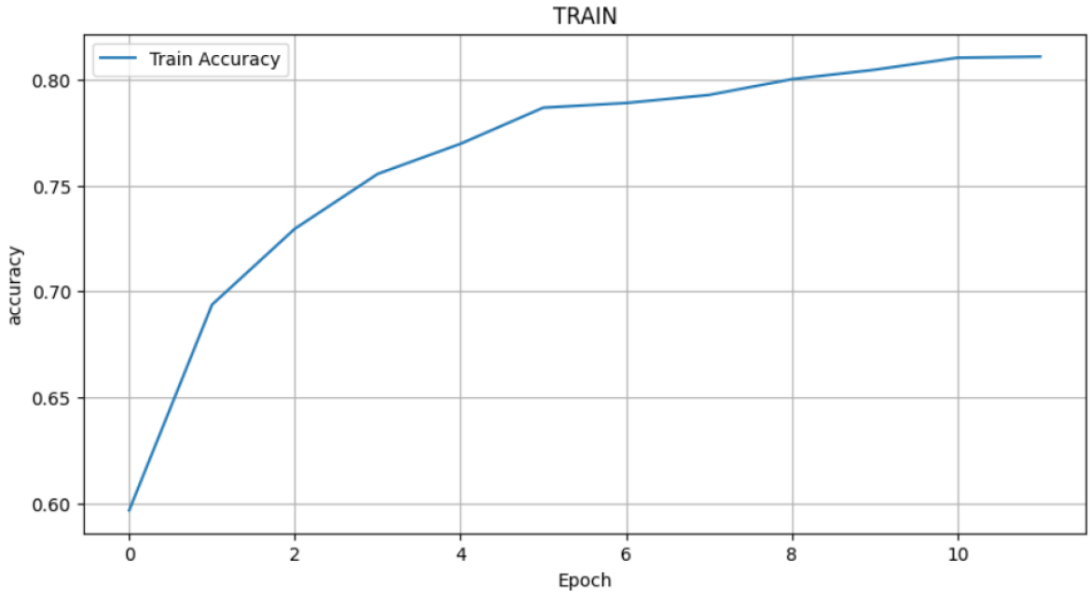
Plots:

After epoch number 12 the model starts to overfitt the data and the accuracy of the validation and test started decreasing.

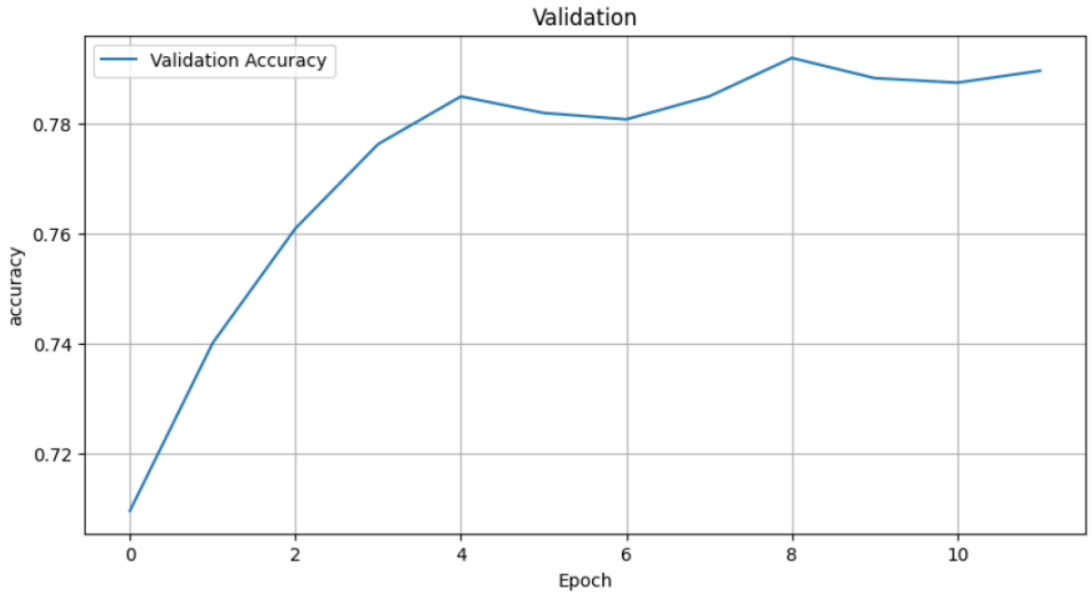
Training, Validation and Test loss:



Train Accuracy:



Validation Accuracy:



Test Accuracy: 81.7833%.