FUNDAMENTALS OF COMPUTER VISION FINAL REPORT

Ayşe Balcı 21726985
Fatma Nur Demirbaş 21727116
Pelin Azizoğlu 21827167

ABSTRACT

Painters who are making money as a painter, or painting as a hobby want to learn their painting styles. We decided to create a project which takes a painting and could find the painter's style.

1 INTRODUCTION

Painters want to improve their talents, and to improve themselves they need to know famous painters who have painting styles like theirs. For this aim, we decided to create a project which takes a painting and could find the painter's style.
The user uploads a painting, and our project identifies which famous artist's works the uploaded painting looks like. It makes that with a dataset that has 50 different famous painters and predicts which famous painters' work people artwork looks like.

2 METHOD

Firstly, we created a function called load_datasets(). We created the torchvision.transforms module that provides various functionality to preprocess the images, we used 3 different transforms as data augmentation. We resize the image for (64*64) shape, transform them into tensors, and normalized tensor images with mean and standard deviation. We also made a horizontal flip and some rotations. We partitioned test images 0.1 of the dataset, validation images 0.2 of the dataset, and train images 0.7 of the dataset. We used torch.utils.data.random_split() to randomly split the images into train, validation and test.

```
def load_datasets(batch_size, dataset_dir):
    transform1 = transforms.Compose([
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])
    transform2 = transforms.Compose([
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])

    transform3 = transforms.Compose([
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.0, 0.0, 0.0], std=[1.0, 1.0, 1.0]),
    ])
```

We divided the data into batches using the PyTorch DataLoader class. We created three objects train_dl, val_dl, and test_dl for train, validation, and test data respectively by giving parameters for training data and batch size into the DataLoader Class.

```
val_size = int(dataset_size * VAL_SIZE)
test_size = int(dataset_size * TEST_SIZE)
train_size = dataset_size - val_size - test_size

train_ds, val_ds, test_ds = random_split(dataset, [train_size, val_size, test_size])

train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=4, pin_memory=True)
val_dl = DataLoader(val_ds, batch_size * 2, num_workers=4, pin_memory=True)
test_dl = DataLoader(test_ds, batch_size * 2, num_workers=4, pin_memory=True)

return train_dl, val_dl, test_dl
```

The fine-tuning is a method of transfer learning and obtaining image classifiers on your custom datasets from pre-trained CNNs. We used fine-tuning because if we can find a trained model that already does one task well, and that task is similar to ours in at least some remote way, then we can take advantage of everything the model has already learned and apply it to our specific task. Also, fine-tuning saves compute time, and further training might sometimes harm the model performance as well.
We used a pre-trained ResNet18 model, with all of the convolutional layers frozen and we replaced the fully connected layer, the last convolutional layer is also left unfrozen to allow training of the convolutional layer. We accomplished it by setting the required grad variable in all of the appropriate layer's parameters to False.

```
class PretrainedResidualModel(ImageClassificationBase):
    def __init__(self, num_classes, with_last_cnn=True):
        super(PretrainedResidualModel, self).__init__()

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

        # freezing the pretrained layers
        freeze_till = 7 if with_last_cnn else 8
        for idx, child in enumerate(self.model.children()):
            if idx < freeze_till:
                for params in child.parameters():
                    params.requires_grad = False

        in_features = self.model.fc.in_features
        self.model.fc = nn.Linear(in_features, num_classes)

    def forward(self, x):
        return self.model(x)
```

3 Experimental Settings

Dataset:https://www.kaggle.com/ikarus777/best-artworks-of-all-time

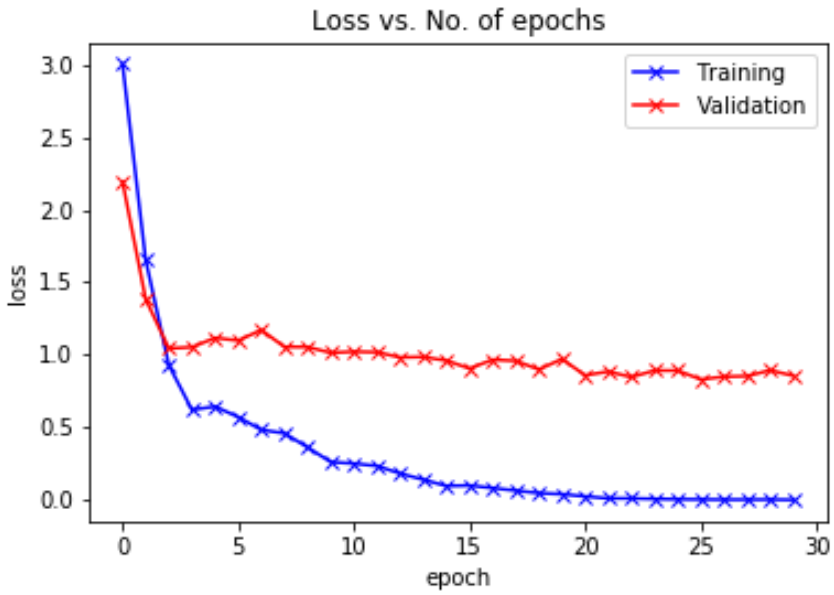
The user will upload a painting, and our project identifies which famous artist's works the uploaded painting looks like. We used the "Best artworks of All Time" dataset from Kaggle. It contains 50 different famous painters' works. We split the dataset into train, validation, and test, and start using algorithms.

We used python as a programming language and ran our codes on both PyCharm and Jupyter Notebook.

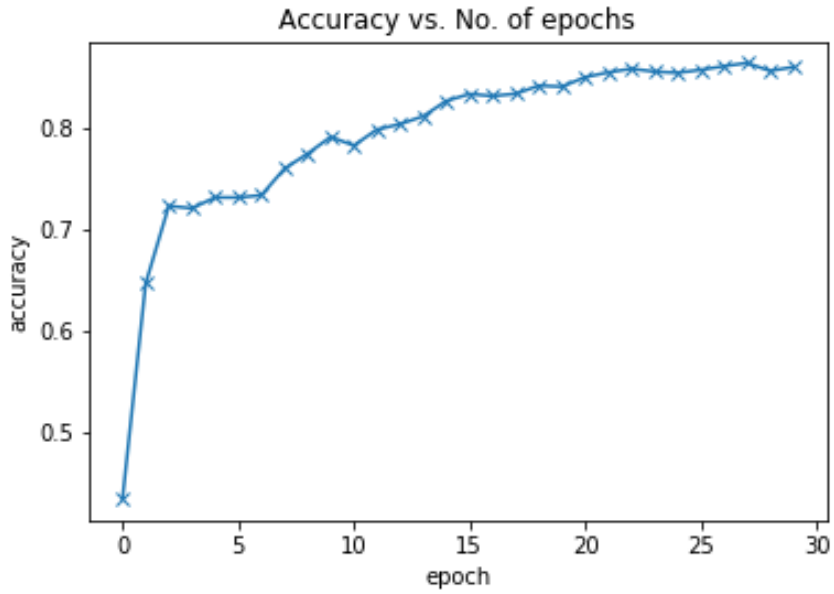
4 Experimental Results

We tried so many different models and based on the overall performance the Pretrained ResNet model performs the best out of all the experimented models. In the ResNet18 model, we choose batch size as 32, learning rate 0.001, and epoch as 30.
In the 30th epoch, validation loss keeps going to decrease but because of our computer's performance we could only train with 30 epochs.

Graph showing loss vs. number of epochs:



Graph showing accuracy vs. number of epochs:



We also save the model and load it later to predict quickly as follows:

```
torch.save(model, 'model.pth')
return results
```

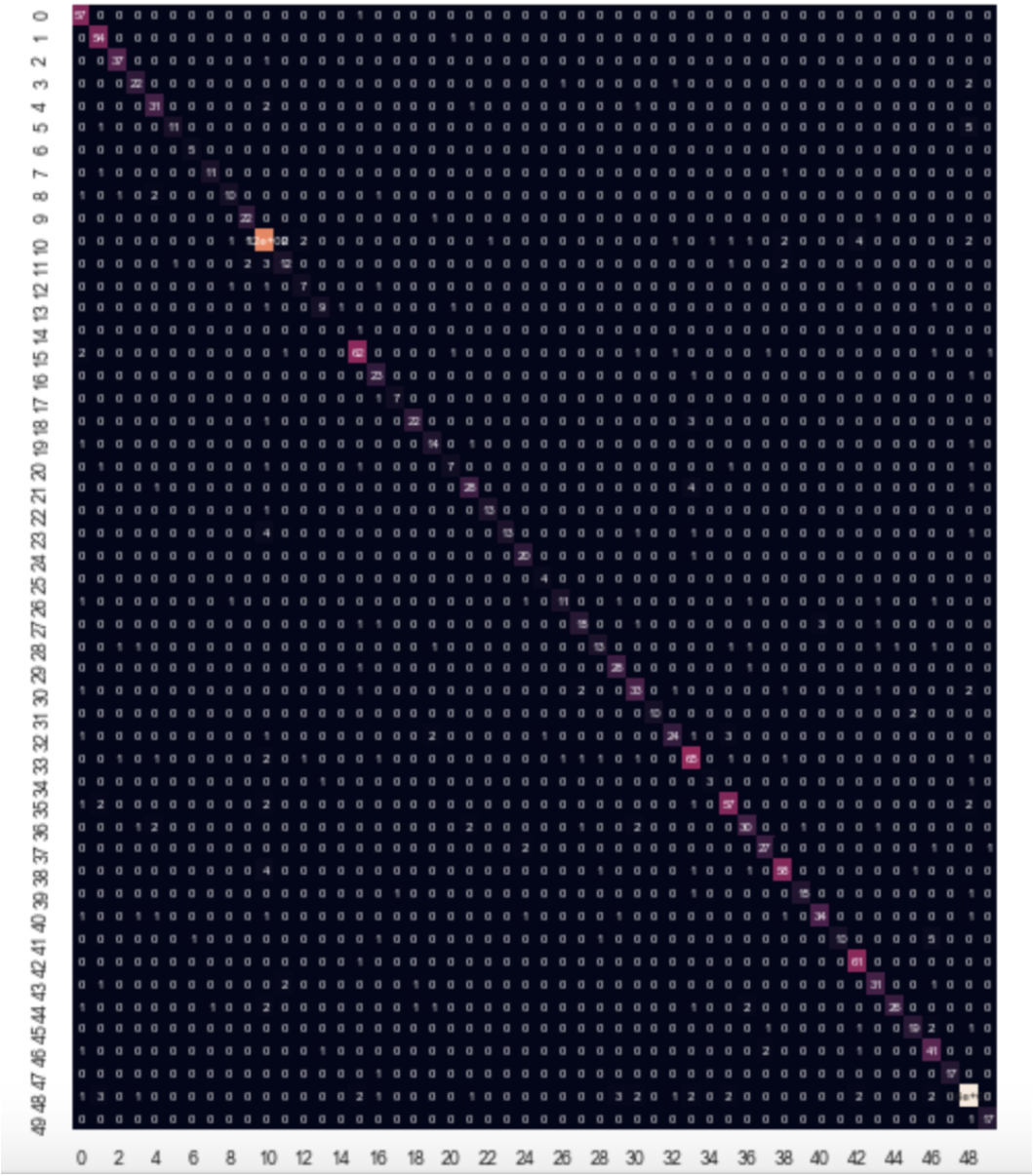
```
test_img_path = '/my_paint.jpg'

prediction(test_img_path)
```

```
model = torch.load('model.pth')
model.eval()
```

Printing test loss, test accuracy, and plotting confusion matrix for the model:

```
Test Loss: 0.8816958069801331
Test Accuracy: 0.8461804986000061
```



5 Discussions/Conclusions

We tried so many different models and based on the overall performance the Pretrained ResNet18 model performs the best out of all the experimented models.
Our project achieves success, but of course, it could improvable. Trying new, more complicated models, changing hyper-parameters with much more values, and trying new approaches could increase the accuracy and decrease the loss. Considering our computer's performance we tried our best, and we think we made this project successfully.