Artwork Classification with Convolutional Neural Network

Problem

Large art collections can contain hundreds of thousands of pieces, with more pieces being accumulated all the time. Cataloging incoming art pieces requires a museum employee to make a subjective call The classification of these pieces into correct departments and collections requires art historians and museum curators to assess and categorize each work, representing a huge labor cost to institutions that typically are run on small budgets. By automating the higher levels of classification (e.g what department a given piece should be sent to for further analysis), art historians can spend less time on the initial sorting of incoming pieces. Additionally, this model could be used in digitizing the back-catalog of an institution, as only images of each artwork will be needed to generate a classification.

The client for this model would be any institution that curates a large catalog of art work. These institutions could include: museums, auction houses, and even personal collectors. The model trained for this project will focus on classifying artworks into the departments used at the Metropolitan Museum of Art (MET), but it will be generalizable enough to re-train to meet the needs of other institutions. Additionally, the training catalog is so broad this model will likely give more granular classifications than is necessary for smaller institutions. These classifications can be grouped when fewer departments are involved, likely making the classifier more accurate as the more specific departments of the MET are grouped into larger classes at smaller museums and auction houses.

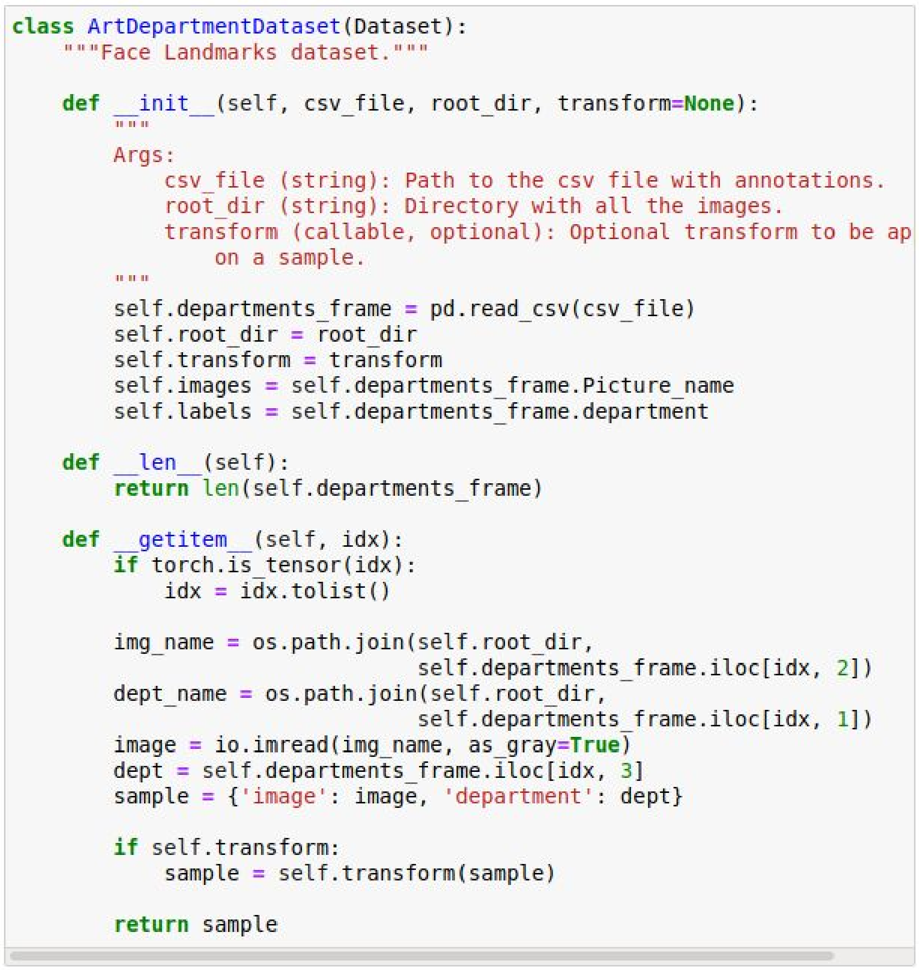
Dataset

To create a dataset for this project, the REST API was used to download metadata and an image for works of art in the MET database. The MET allows public access to all of its more than 450,000 works of art, most of which include images of each piece as well. The data collection for this project took place in two stages: generating a dataframe containing the information for each unique piece of art, and downloading the image data of each work.

1. **Creating Local Database of Art Data:** API access to the MET database does not allow for a single request for all of their information on artwork. Instead requests must be made per work of art using a unique identifier called an object ID. However, a request can be made for all object IDs in the MET database, so a series of all identifiers was generated by making a single API request for this object information. This series was then iterated through to make requests for all information associated with these objects. This information came in the json format, which was appended to a list. Once the list had iterated through all possible object IDs a dataframe was created from the list and saved as a CSV file.
2. **Downloading images of each artwork:** Among the features of the previously generated dataframe was a URL to a jpeg image file of an artwork. In order to train a classifier, the images for each work needed to be downloaded locally. To do this the urllib.request python package was used. A series was created containing all of the links to image files, which was then iterated through to download the image file to my

computer. The images were named with their object ID and their associated department so that they could be correctly identified as they were imported into the classifier. This step represented a bottleneck in my data acquisition pipeline in both time and memory. Downloading each image took a few seconds and even the smallest image files were around 50 MB. These constraints made it not feasible to download the full 450,000+ image dataset, so the scraper was run until around 100,000 images were reached. 100,000 images was the chosen stopping point because I hypothesized that there would be an adequate number of training images for most departments represented in a quarter of the total database.

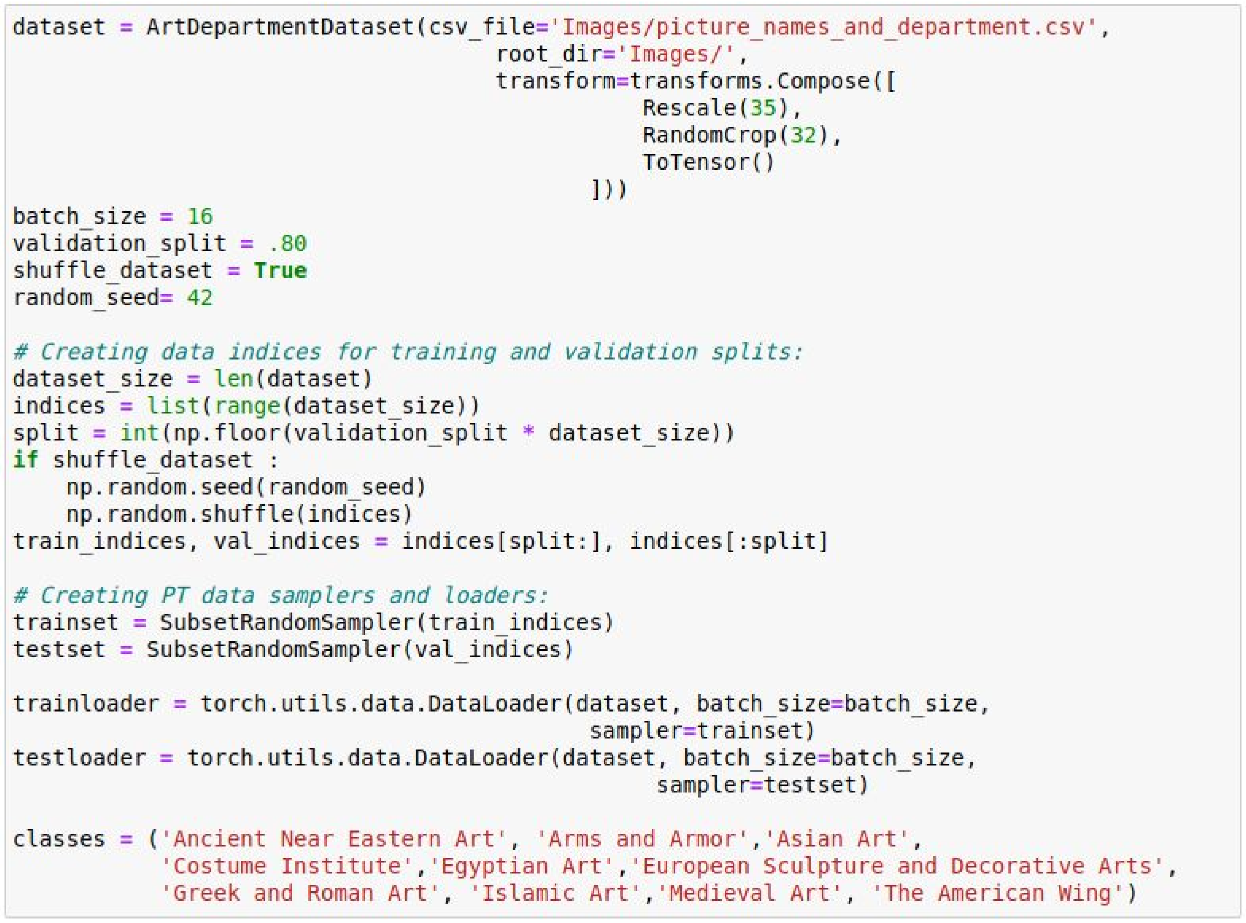
Since the ultimate goal of this project is to create an image classifier, extensive data wrangling was not necessary for feature generation. After the above download steps were executed, each image file contained the information required for classification as a string in the file title. However, significant work took place to generate a python class for importing and transforming the image data into a CNN. A custom class was created that takes a CSV file containing the image file name and associated department from a given folder (Figure 1).



```python
class ArtDepartmentDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be ap
                on a sample.
        """
        self.departments_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform
        self.images = self.departments_frame.Picture_name
        self.labels = self.departments_frame.department

    def __len__(self):
        return len(self.departments_frame)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = os.path.join(self.root_dir,
                                self.departments_frame.iloc[idx, 2])
        dept_name = os.path.join(self.root_dir,
                                self.departments_frame.iloc[idx, 1])
        image = io.imread(img_name, as_gray=True)
        dept = self.departments_frame.iloc[idx, 3]
        sample = {'image': image, 'department': dept}

        if self.transform:
            sample = self.transform(sample)

        return sample
```

Figure 1: Custom Class Definition

This file is then iterated through to import grayscale image data along with its associated department. As this iteration takes place transformations such as rescaling, cropping, and converting image data to a tensor occurs. Ultimately this enables this class to be passed to a DataLoader (pre-defined by PyTorch), which performs the loading batch-wise as a CNN is being trained (Figure 2).



```python
dataset = ArtDepartmentDataset(csv_file='Images/picture_names_and_department.csv',
                                    root_dir='Images/',
                                    transform=transforms.Compose([
                                               Rescale(35),
                                               RandomCrop(32),
                                               ToTensor()
                                           ]))
batch_size = 16
validation_split = .80
shuffle_dataset = True
random_seed= 42

# Creating data indices for training and validation splits:
dataset_size = len(dataset)
indices = list(range(dataset_size))
split = int(np.floor(validation_split * dataset_size))
if shuffle_dataset :
    np.random.seed(random_seed)
    np.random.shuffle(indices)
train_indices, val_indices = indices[split:], indices[:split]

# Creating PT data samplers and loaders:
trainset = SubsetRandomSampler(train_indices)
testset = SubsetRandomSampler(val_indices)

trainloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
                                           sampler=trainset)
testloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
                                              sampler=testset)

classes = ('Ancient Near Eastern Art', 'Arms and Armor','Asian Art',
           'Costume Institute','Egyptian Art','European Sculpture and Decorative Arts',
           'Greek and Roman Art', 'Islamic Art','Medieval Art', 'The American Wing')
```

Figure 2: Sample of DataLoading Pipeline

Initial Findings

To establish a baseline for classifier performance, a simple CNN was first created using pytorch that contained: two convolutional layers, two max pooling layers, and three linear layers (Figure 3). The loss criterion used was CrossEntropy, and stochastic gradient descent was the optimization algorithm applied.

The initial input data was split 80-20 between training and testing data, then each image was downscaled to 250 pixels in width and randomly cropped a 225 pixel square from these images. The purpose of these steps was to include most of the image data while also making the input data uniform for the CNN. Images were then loaded in randomly selected batches of 4 to train the network.

The model was trained using two different epochs of the training data to reduce the possibility of noise induced through the random batching of images. The performance of the model was monitored in real time using TensorBoard through training loss, and making test predictions every 1000 steps (Figure 4).

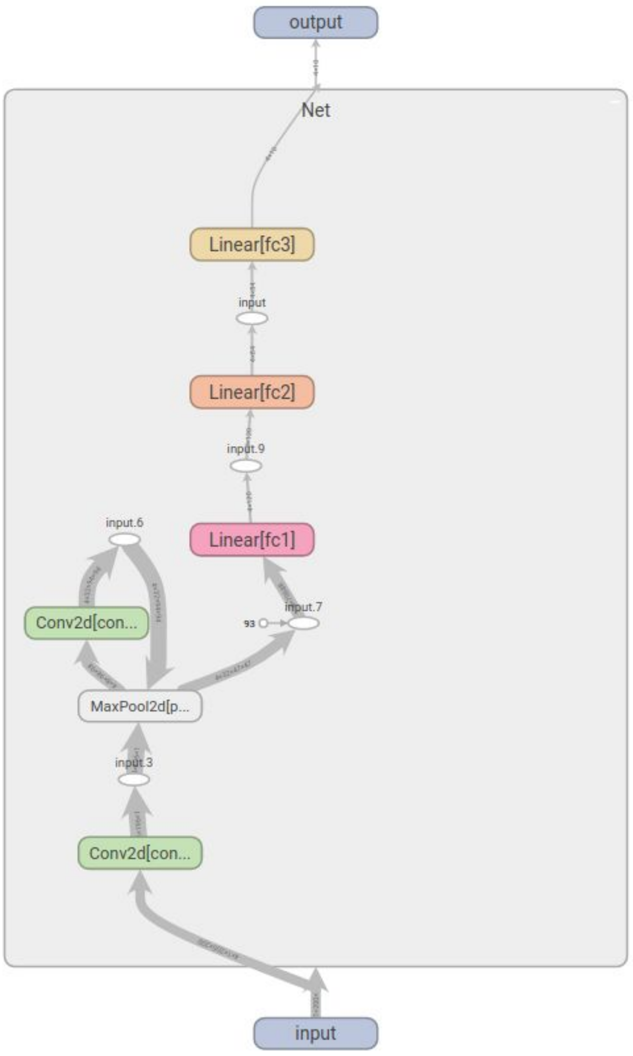


Figure 3: Schematic of CNN

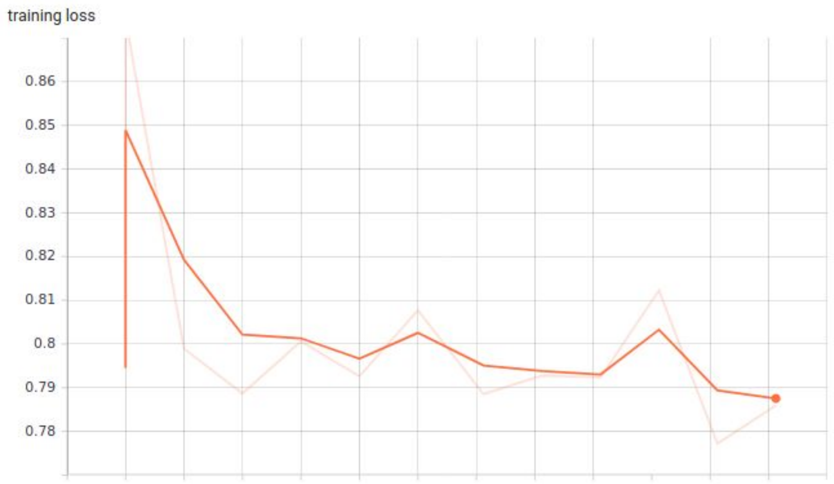
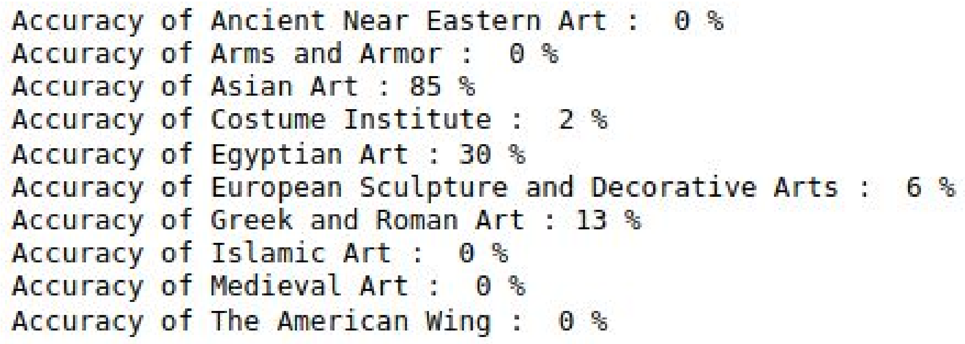


Figure 4: Training Loss By Step

The training loss gradually decreased as training proceeded, but the overall performance of the classifier was poor. Of the ten possible categories only half had a prediction accuracy of greater than 0% (Figure 5).



```
Accuracy of Ancient Near Eastern Art :  0 %
Accuracy of Arms and Armor :  0 %
Accuracy of Asian Art : 85 %
Accuracy of Costume Institute :  2 %
Accuracy of Egyptian Art : 30 %
Accuracy of European Sculpture and Decorative Arts :  6 %
Accuracy of Greek and Roman Art : 13 %
Accuracy of Islamic Art :  0 %
Accuracy of Medieval Art :  0 %
Accuracy of The American Wing :  0 %
```

Figure 5: Testing Accuracy By Category

Next Steps

Moving forward with this project, the CNN will be refined through extensive hyperparameter tuning.