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.

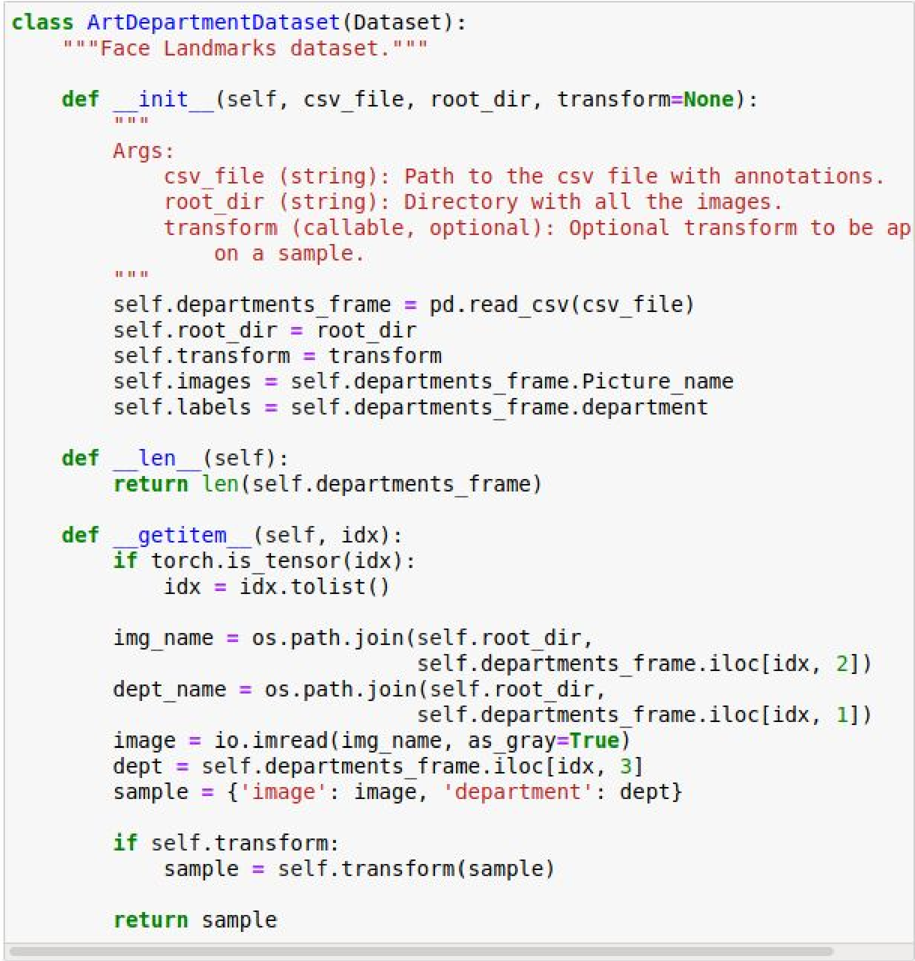
Dataset

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.

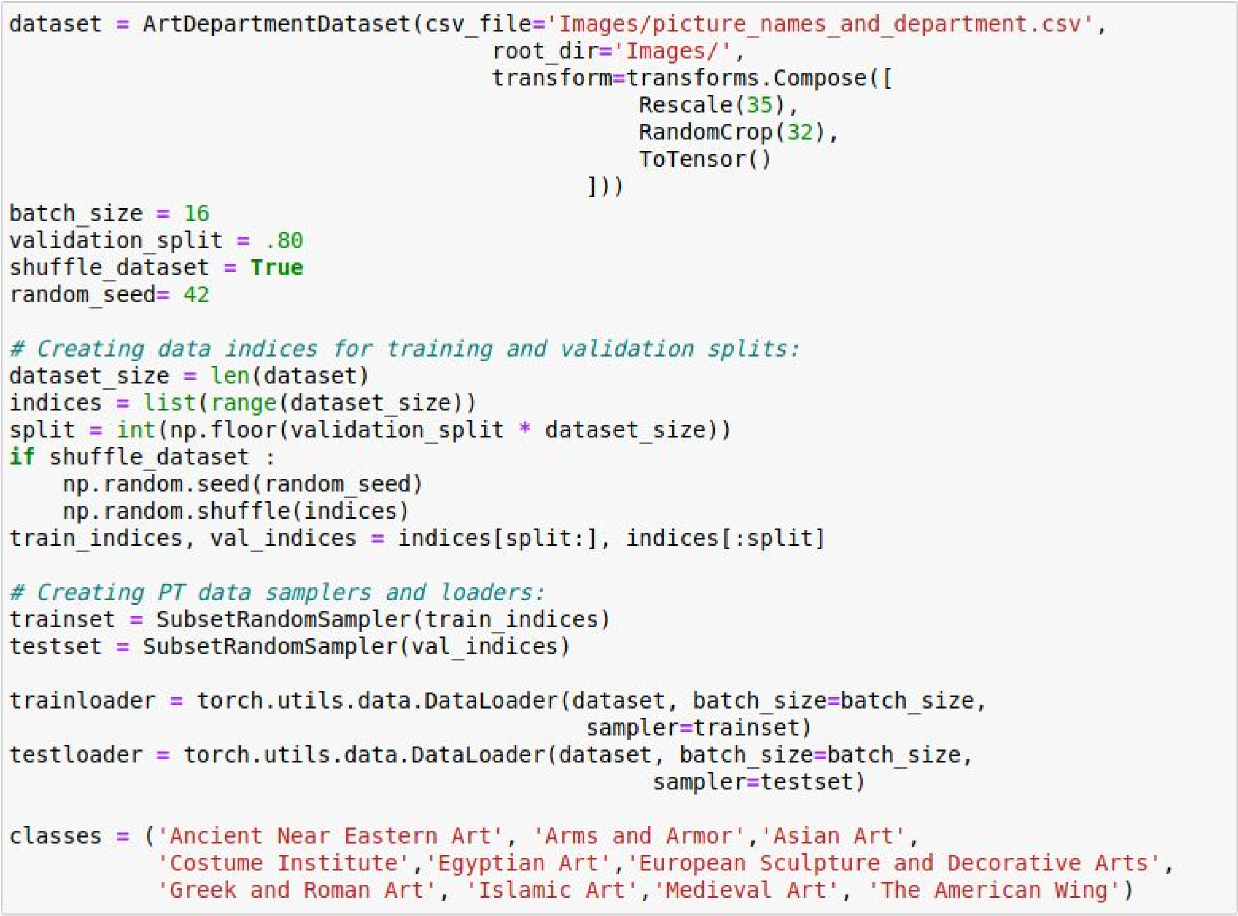
Since the ultimate goal of this project is to create an image classifier, extensive data wrangling was not necessary for feature generation. After the above download steps were executed, each image file contained the information required for classification as a string in the file title. However, significant 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).



```
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).



```
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).

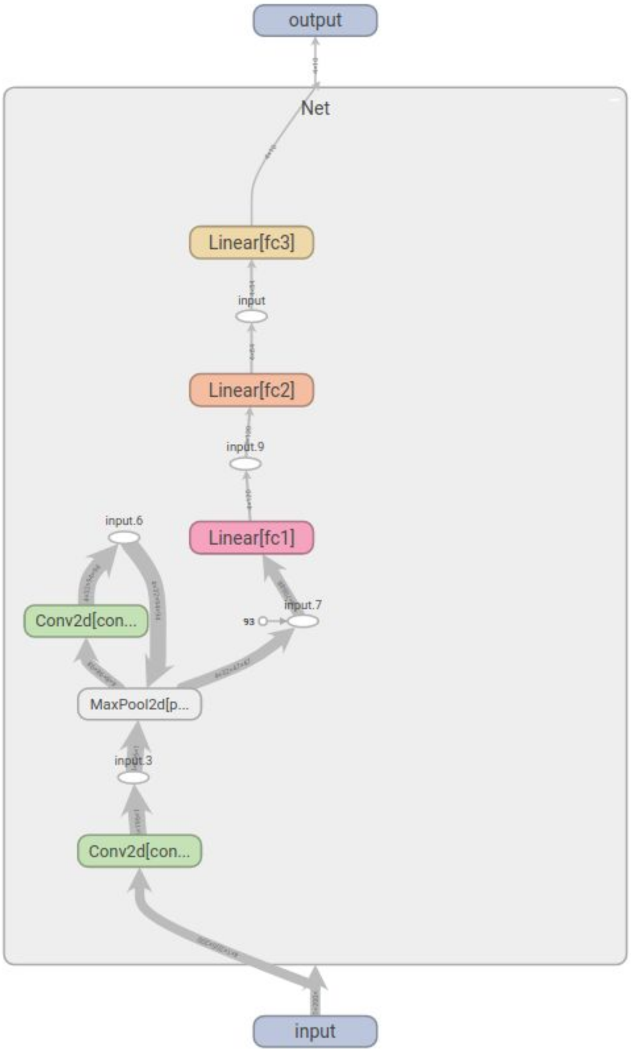


Figure 3: Schematic of CNN

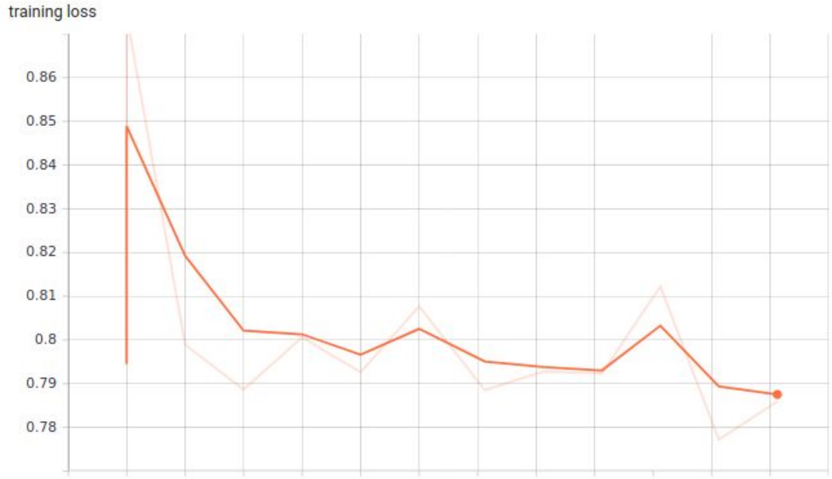
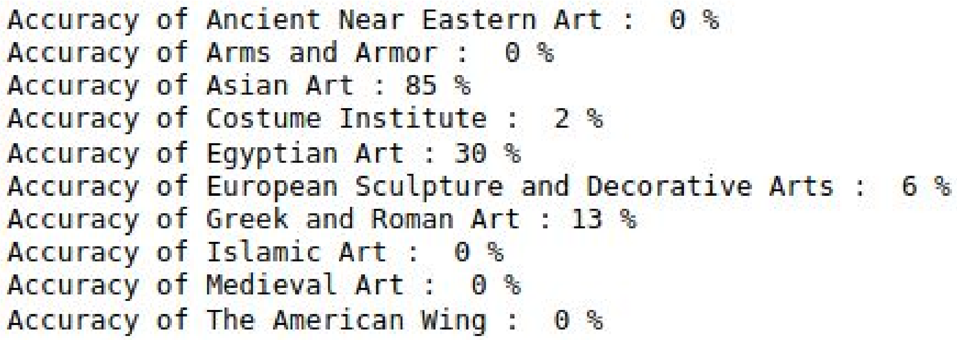


Figure 4: Training Loss By Step

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

Tuning the Convolutional Neural Network

After the engineering of the data loaders and the basic architecture of the CNN was complete, it was necessary to select the best hyperparameters for image classification. A significant barrier in hyperparameter tuning this model was the long training and testing times. This project was conceptualized with PyTorch so that model training could be performed on a GPU, but a CUDA compatible GPU was not accessible at this stage.

To save training time, the best loss function and learning rate were inferred from the paper *Fine-tuning Convolutional Neural Networks for fine art classification* (Cetinic et al., Expert Systems with Applications). Parameters such as: image batch size, rescaling, normalization, the number of convolutional layers in the CNN, and the number of nodes in the fully connected layer were tested for two training epochs.

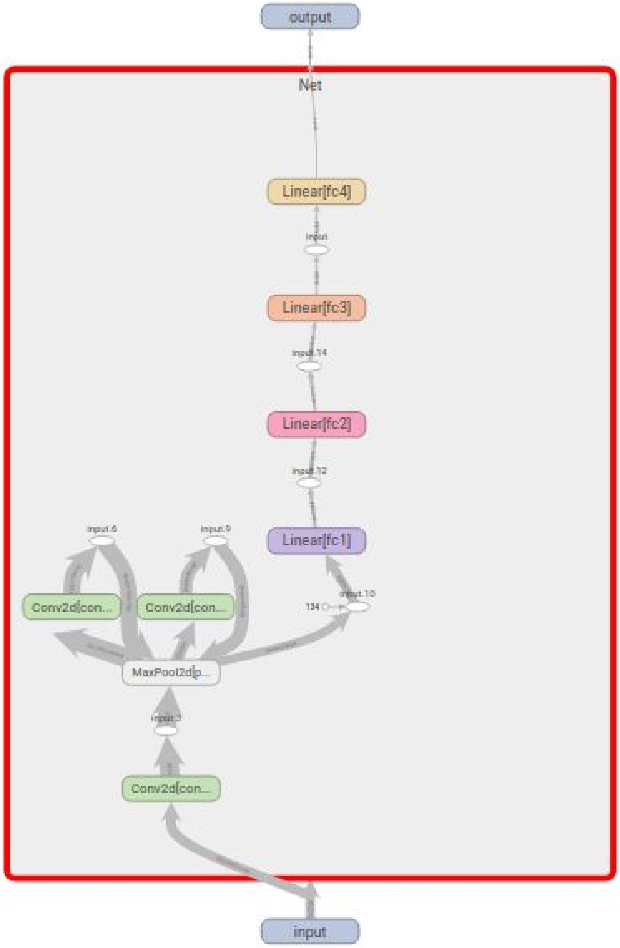
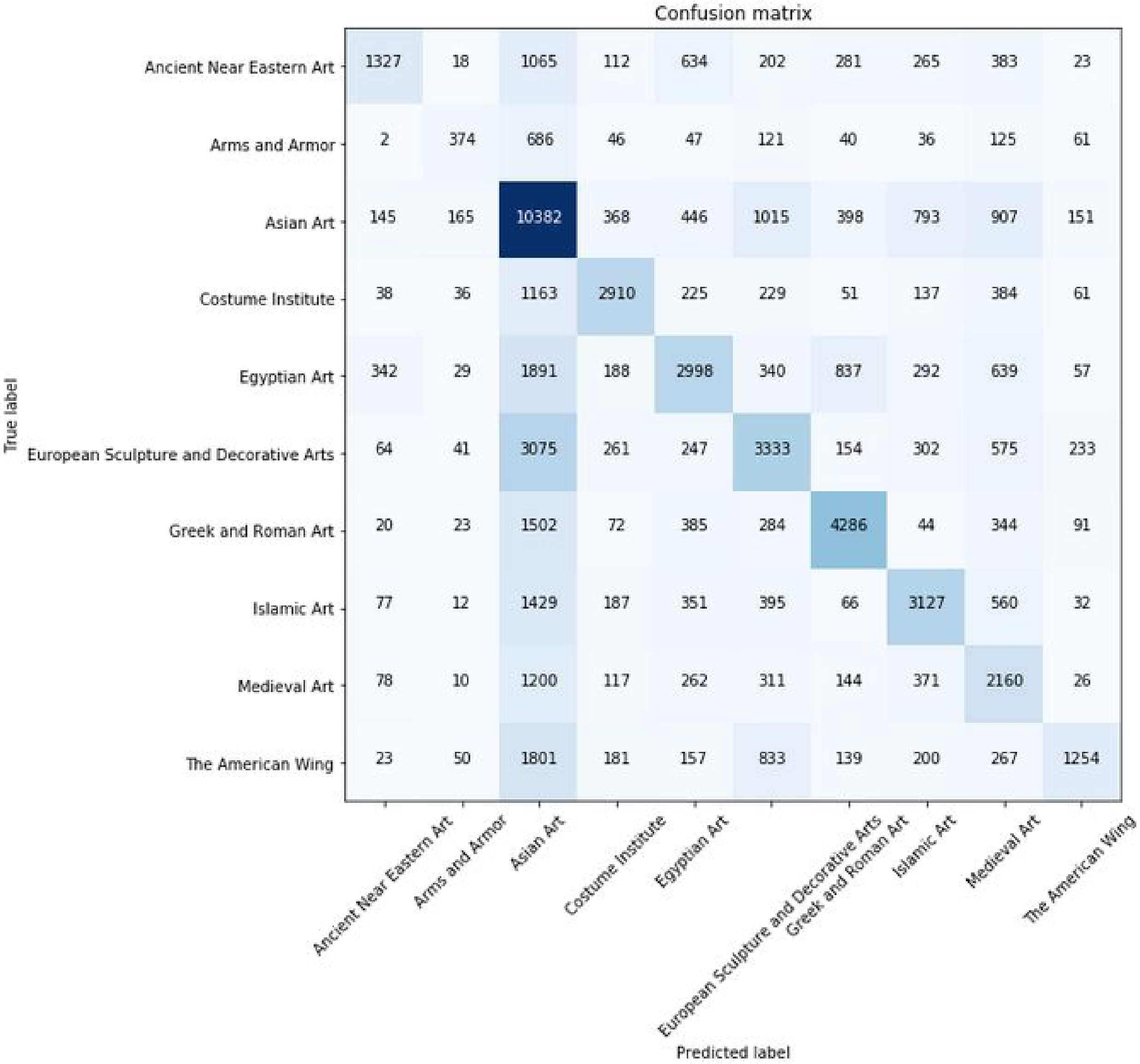


Figure 6: Final CNN Architecture

The parameters with the lowest running loss were selected for more training epochs. This selection process yielded a model that contained 3 convolutional layers, 2 max pooling layers, and 4 fully connected layers (Figure 6).

For this project, the potential cost of false positives are low as this model would likely be deployed with human curation at the end. Because of this low potential cost, overall accuracy was the metric focused on when evaluating the overall performance of the model. Training the CNN for 20 epochs led to an average test accuracy of around 55%, with some categories performing very well and others pulling down the average accuracy (Figure 7).



True label \ Predicted label	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
Ancient Near Eastern Art	1327	18	1065	112	634	202	281	265	383	23
Arms and Armor	2	374	686	46	47	121	40	36	125	61
Asian Art	145	165	10382	368	446	1015	398	793	907	151
Costume Institute	38	36	1163	2910	225	229	51	137	384	61
Egyptian Art	342	29	1891	188	2998	340	837	292	639	57
European Sculpture and Decorative Arts	64	41	3075	261	247	3333	154	302	575	233
Greek and Roman Art	20	23	1502	72	385	284	4286	44	344	91
Islamic Art	77	12	1429	187	351	395	66	3127	560	32
Medieval Art	78	10	1200	117	262	311	144	371	2160	26
The American Wing	23	50	1801	181	157	833	139	200	267	1254

Figure 7: Confusion Matrix from Best Classifier

Implementation of Transfer Learning

Training a CNN on a low-powered CPU is extremely time intensive, in an attempt to decrease training time a pretrained model was applied to this classifier (transfer learning). Previous art classifiers used the pre-trained ResNet model from *Deep Residual Learning for Image Recognition* (He et al., arXiv.org) to some success. This model was trained on 1.2 million images of 1,000 classes, the specific one used for this project was ResNet18 which contains 5 layers (Figure 8). The model was imported with pretrained weights and used to classify the image data.

The performance of this model was surprisingly poor, achieving an overall accuracy of 9% with a training loss that remained steady over 5 training epochs. Interestingly, the model predicted greek and roman art more often than any other class (Figure 9). This is possibly due to the image training set for ResNet being focused around object detection rather than landscapes and abstract shapes.

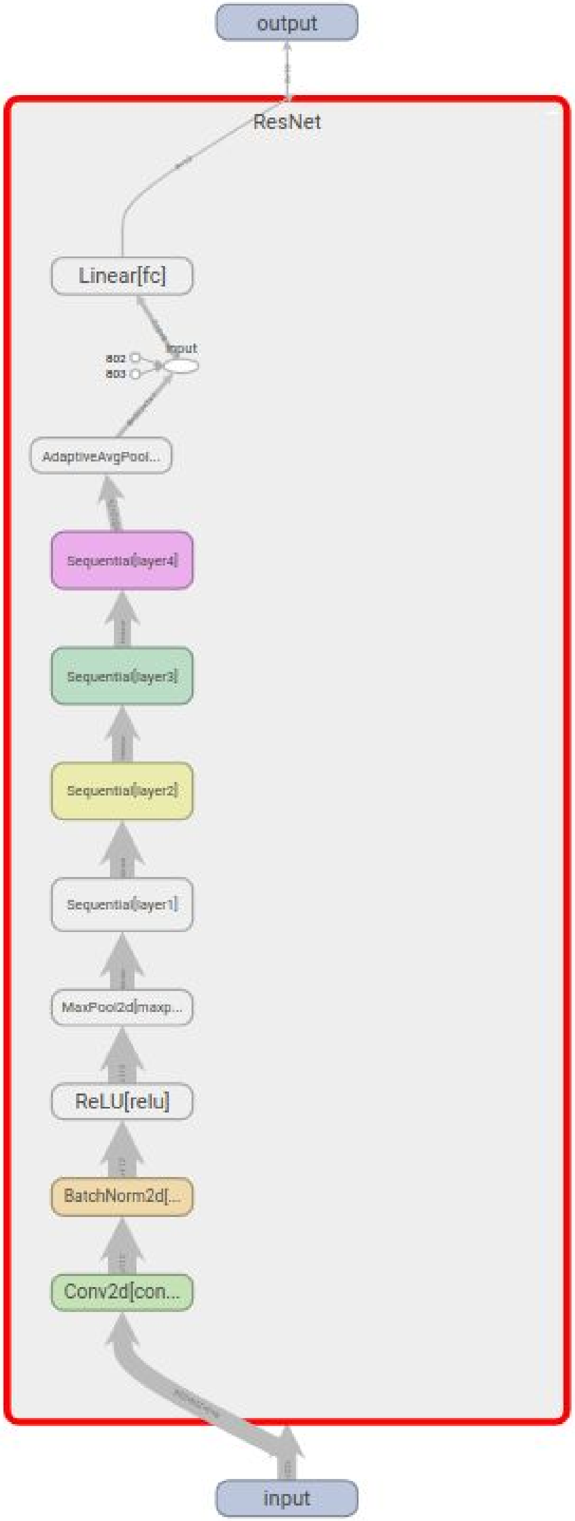
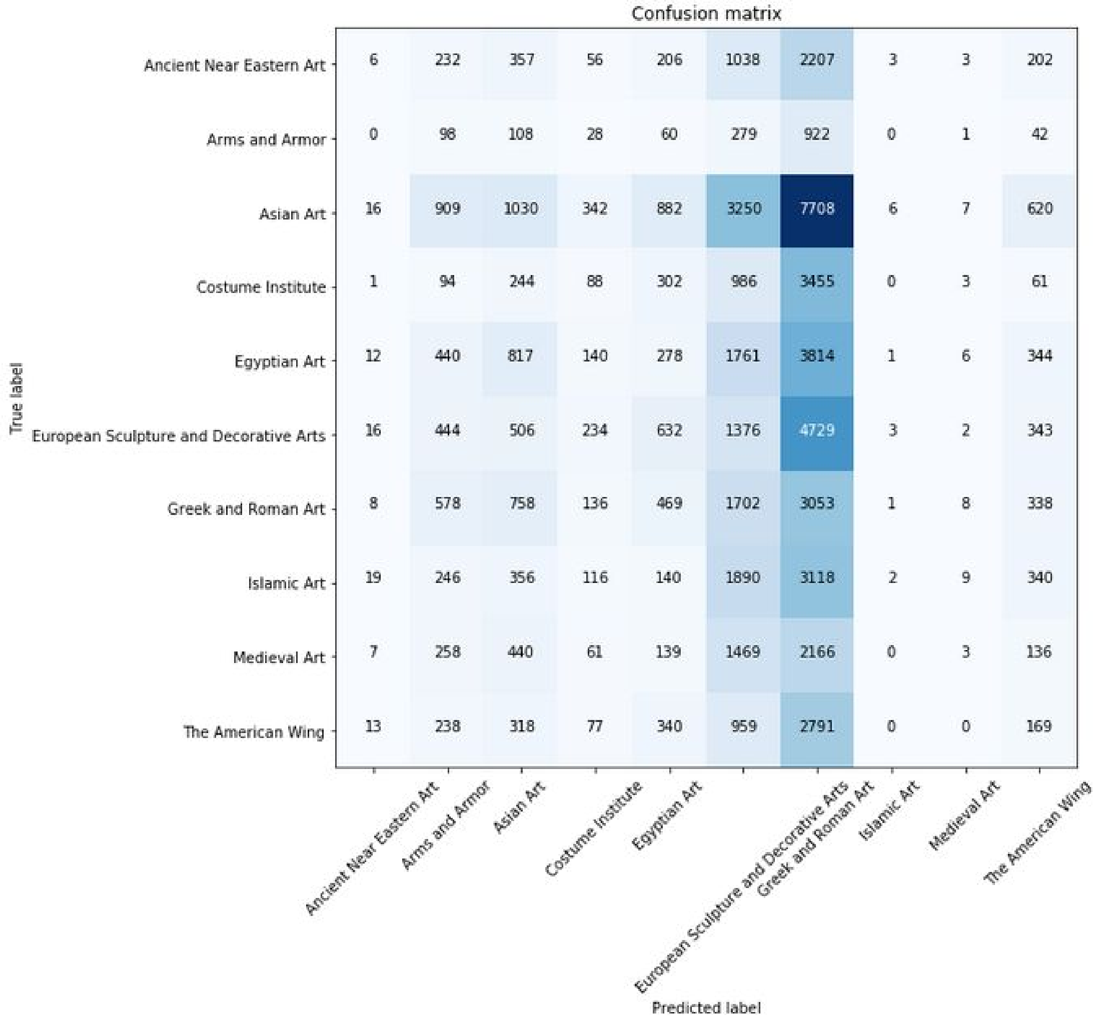


Figure 8: ResNet18 Architecture



	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
Ancient Near Eastern Art	6	232	357	56	206	1038	2207	3	3	202
Arms and Armor	0	98	108	28	60	279	922	0	1	42
Asian Art	16	909	1030	342	882	3250	7708	6	7	620
Costume Institute	1	94	244	88	302	986	3455	0	3	61
Egyptian Art	12	440	817	140	278	1761	3814	1	6	344
European Sculpture and Decorative Arts	16	444	506	234	632	1376	4729	3	2	343
Greek and Roman Art	8	578	758	136	469	1702	3053	1	8	338
Islamic Art	19	246	356	116	140	1890	3118	2	9	340
Medieval Art	7	258	440	61	139	1469	2166	0	3	136
The American Wing	13	238	318	77	340	959	2791	0	0	169

Figure 9: ResNet Classifier Confusion Matrix

Conclusions

Overall, this project has been successful in creating a convolutional neural network for classifying artworks into different departments. Despite the maximum overall average accuracy of 50% achieved with these data, this classifier would still be useful in a real world context. The classifier demonstrates at least some feature recognition of each class and correctly classifies the majority of the departments regularly. As is, this tool could be implemented along with human oversight. Cases where the model has a high prediction confidence of just one category can be automatically sorted, and cases where confidence is lower can be passed to an expert to make the final curation decision.

Running experiments to find out what the best hyperparameters are for multiple CNN configurations would be the immediate next step. Due to computational limitations it was not feasible to test the whole range of CNN hyperparameters on this image set, so assumptions had to be made about what would be the best loss function and optimization settings based on previous literature. However these assumptions were made for a different art dataset trying to predict different classifications, making it unlikely that these are truly the best settings for this classifier.

This dataset and classifier architecture could also be expanded on to make other classifications such as: which artist made the piece, what artistic movement a piece belongs to, or even the country that the work comes from. These would be interesting classification problems, and could also help in the identification of artwork with unknown origins.