Extreme Image Classification - Landmark Recognition

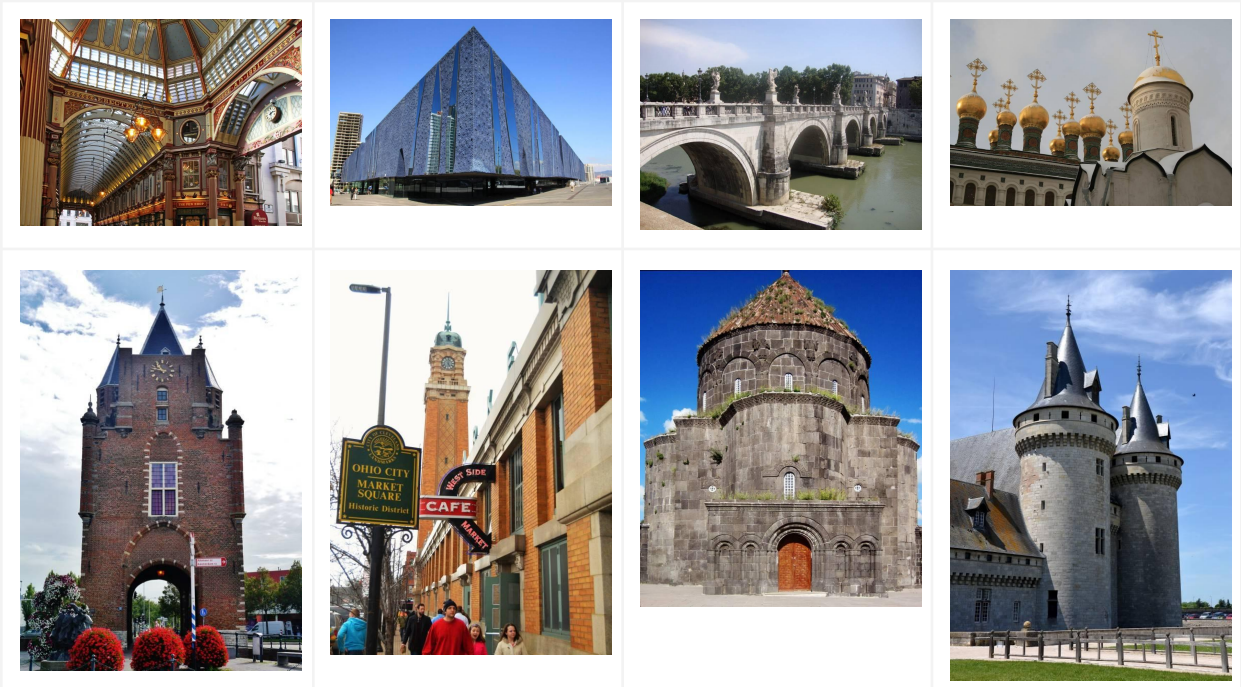
W281 Computer Vision Final Project

Vivek Bhatnagar | Varun Tanna | Charlee Stefanski

Introduction

Have you ever seen a photo of a friend or family member on vacation and wondered what landmark they are posing in front of? Have you ever wanted to find a photo of yourself in front of a specific landmark among hundreds of vacation photos? This report describes several image recognition techniques we implemented to classify images of global landmarks.

Image classification is an active area of research in computer vision, with applications in many industries, most notably in medical imaging, traffic control systems, and satellite imaging systems. We apply methods learned in class and independently researched to achieve a highly accurate classification model for the given dataset. This classifier has several utilities for individuals and businesses, such as a tool for identifying landmarks when traveling, a tool for individuals to organize photos from vacations, or a tool for recognizing landmarks in images posted on social media to provide information to users.



Executive Summary

This work aims to design and implement a model for image classification of global landmarks which is accurate and generalizable. Some unique challenges of this project are:

- Dataset is meant to reflect a crowdsourced sample of landmark images.
- Several different angles/views of a landmark are included in the dataset.
- Extremely long-tailed class distributions
- Inconsistent landmark attributes (interiors & unrelated exteriors) captured in images.

The report includes a discussion of several feature extraction methods and error analysis for using these features for classification, including:

- HOG (Histogram of Oriented Gradients) features
- Harris Corner Detection and ORB (Oriented FAST and rotated BRIEF) descriptor
- SIFT (Scale Invariant Feature Transform) features

Classifiers developed using these feature extraction techniques provided only modest test prediction accuracy. Using SIFT features for building an Image Retrieval system and a SIFT Histogram BoW Multi-Layer Perceptron model showed 55.2% and 64.2% accuracy rates, respectively. Therefore, several additional image recognition methods were implemented to find the best classifier for this problem. We used pre-trained Convolution Neural Network (CNN) architecture to achieve a model with 96.13% balanced test accuracy for 51 landmarks. Finally, we generalize the model to include hundreds of more landmarks, many of which have significantly smaller training images available. We are able to achieve a 93.20% balanced test prediction accuracy for 491 distinct landmarks. A summary of the best-performing models is in the table below.

Best Models Achieved

Model	**Balanced Test Accuracy (%)**
ResNet34 (51 Classes)	96.13
ResNext101 (491 Classes)	93.20
SIFT - BoW Model (5 classes)	64.80

Data

Our data is from a Kaggle competition ("Google Landmark Recognition 2021"), which challenges participants to solve the Landmark classification problem mentioned above. Google landmarks dataset (GLDv2) is the largest worldwide dataset to foster progress in landmark recognition and retrieval (Weyand et al., 2020). GLDv2 training set presents a realistic crowdsourced dataset with diverse types of images for each landmark.

The training dataset contains approximately 1.58 million images. The target images are named with a unique id. This id is reflected in the training dataset, which includes the image id and corresponding landmark id, which we will refer to as the "class" of the image. The dataset contains a large number of landmark classes (more than 81,000 classes). However, as we will discuss later, the number of images per class varies significantly. The following code snippets can be found in *EDA.ipynb*.

```
TRAINING_FILE = '../Data/landmark-recognition-2021/train.csv'
df_train = pd.read_csv(TRAINING_FILE)

print(df_train.columns)
print(f'Total Images: {len(df_train.landmark_id.values)}')
```

```
Index(['id', 'landmark_id'], dtype='object')
Total Images: 1580470
```

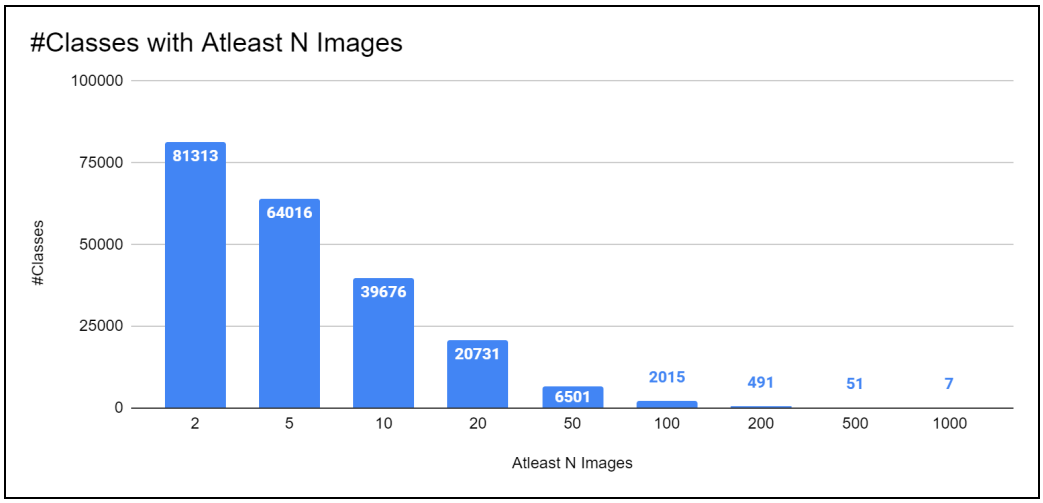
```
print(f'Total Clases: {len(classes)}')
```

```
Total Clases: 81313
```

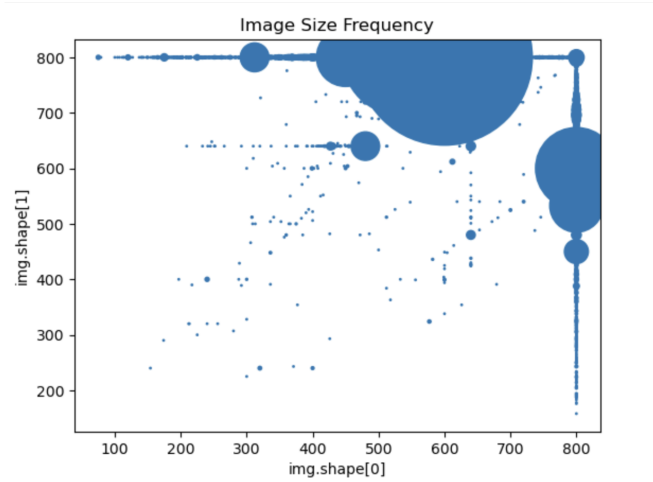
In addition to the large dataset size, we also found the dataset was relatively clean and did not require extensive preprocessing. Sample landmark images from the dataset are shown below.

While there are a large number of classes in this dataset, there is a lot of variation among the number of training images for a given class. We found extremely long-tailed class distributions - several thousand landmarks only had one or two images.

After examining the distribution of images in the dataset, we decided to use the landmarks with 500 or more images (as shown in the chart above) to develop and train our classifiers to establish a good baseline. In *Dataset.ipynb* we retrieve this subset of data; 51 classes for a total of 45,579 images). We also transform the labels of the images to a numerical value between 0 to 50 (inclusive) for cross-entropy loss computation. The lowest number of images in a class is 502, and the highest number in a class is 6,272.



There is variability in the size of the images in the dataset. The most common image size is 600x800, and all images are within 800x800 pixels. The following visualizations plot image size with the size of the bubble reflecting the frequency. The code used to generate this visualization can be found in *EDA.ipynb*.



Another challenge we found within the data was that many images were labeled as a certain landmark but did not contain what could be considered as identifying attributes of that landmark. In many cases, the images had pictures of objects or people, without the landmark in the background, but internal or unrelated external landmark surroundings. Since we could not manually parse and view all 45,000 images to see whether the images could be considered a picture of a given landmark, we continued under the assumption that the majority of 500+ images would be good enough to use. An example of a faulty image with no landmark is shown below. We see an image of people posing at a dinner table, with nothing that could reasonably be considered a visual feature of the landmark.

Testing-Training Split

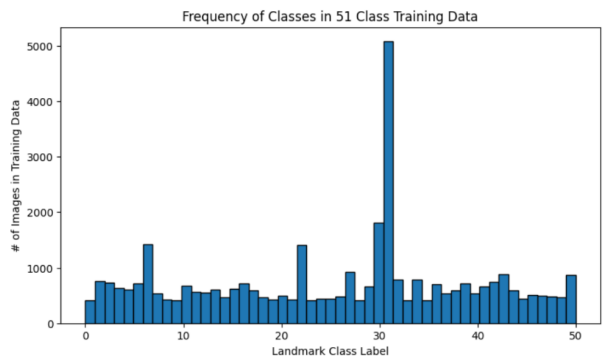
#Classes	Total Images	Training Images	Validation Images	Test Images	Models
5	500	250 (50%)	125 (25%)	125 (25%)	SIFT, ORB
51	45,579	36,918 (81%)	4,104 (9%)	4,559 (10%)	HOG, ConvNet
491	167,716	135,849 (81%)	15,096 (9%)	16,772 (10%)	ConvNet

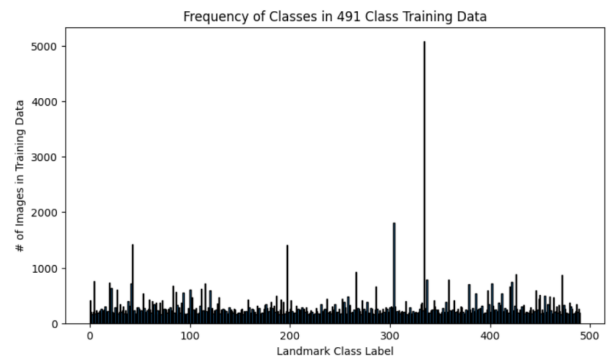
The data was separated into train, validation, and test sets. The code used to split the data can be found in *TrainingDataSplit.ipynb*. We use scikit-learn's *train_test_split* to split the data. The train/validation/test split was done in an 81/9/10 ratio, as shown in the above table.

In addition to the dataset with only 51 classes, as discussed above, to test and iterate implementations for non-learned and library features, we used a smaller subset which included 100 images from just five landmarks (Qutub Minar, Eiffel Tower, Golden Gate Bridge, Pakistan Monuments, and Niagara River). After developing our pre-trained ConvNet models, we used a larger dataset consisting of 491 classes with 200 or more images in the dataset to challenge further and evaluate our classification models' performance.

Below is a visualization of the distribution of training images for each class in the train data after splitting. For the 51-class dataset, most landmarks had a relatively even distribution of images

(between 500 and 800). However, we saw that one class had 5000+ images which is shown as the peak in the first table below. Additionally, for the 491-class dataset, we saw similar distribution where most of the classes were balanced (between 200 and 500 images). However, like before, there are a few peaks (from the 51-classes) and a few other larger landmark sets.





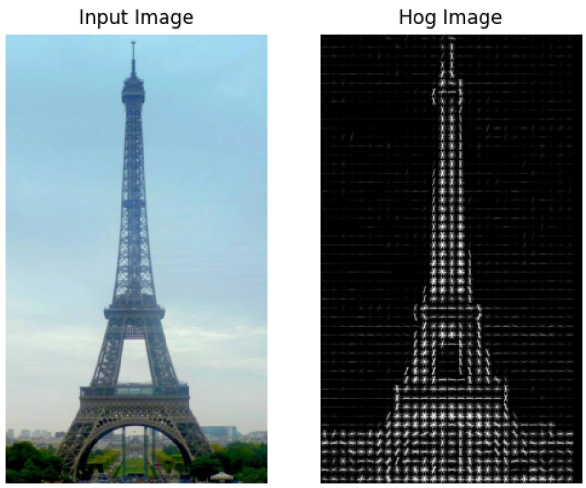
Non-Learned & Library features

We evaluated several feature extraction techniques, both library-based and non-learned. We demonstrate the implementation for each feature with example images and results from classification models using these features.

1. HOG Feature - Neural Network

A Histogram of Oriented Gradients (HOG) (McConnell, 1984) features counts occurrences of gradient orientation in localized portions of an image. We use scikit-image functions to calculate HOG features on randomized crop training images. A visualization of HOG features for one image in the training data is below (Singh, 2019).

```
image = Image.open(self.image_paths[item])
augmented_image = self.augmentations(image)
hogfv, hog_image = hog(augmented_image, orientations=9, pixels_per_cell = (16,16),
               cells_per_block=(2,2), visualize=True, multichannel=True)

image_tensor = torch.tensor(hogfv.astype(np.float32))
```

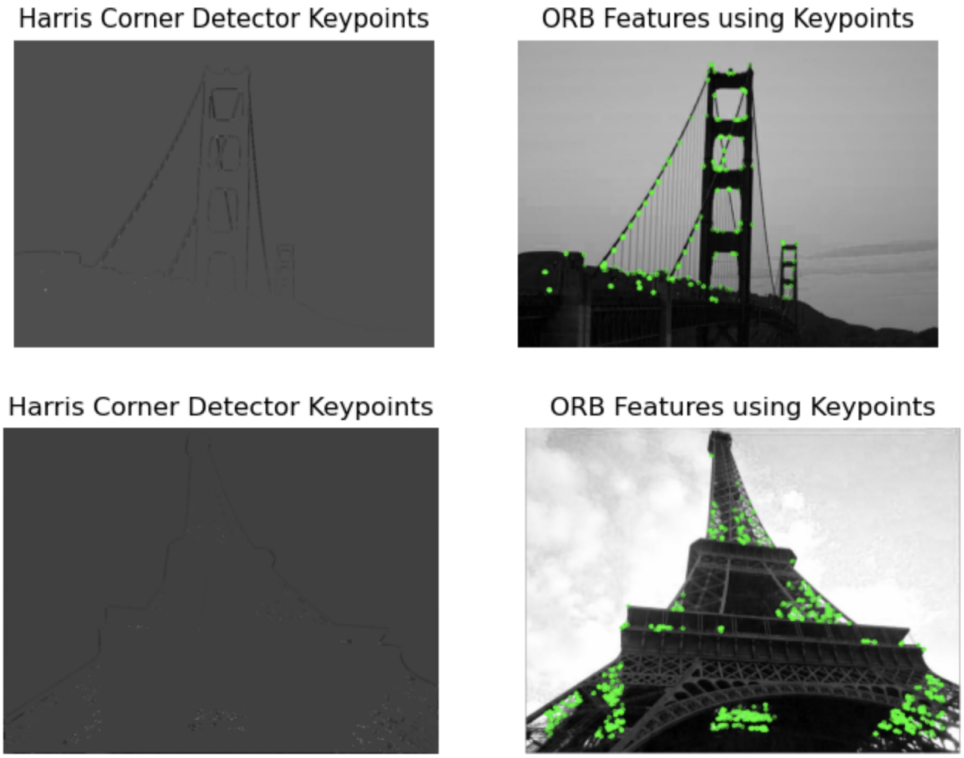


Extracted HOG features (20736) are fed into a neural network with a single hidden layer for multi-class (51) landmark recognition (cross-entropy loss). The HOG-NN model resulted in a 20.92% training and 19.01% test accuracy on our 51-class test set. The implementation can be found in *HOG.ipynb*. The snippet below shows the model implementation:

```
class GLRModel(tez.Model):
    def __init__(self, num_classes):
        super().__init__()
        self.hidden = nn.Linear(20736, 128)
        self.out = nn.Linear(128, num_classes)

    def forward(self, image):
        hidden = self.hidden(image)
        outputs = self.out(hidden)

        return outputs
```

2. Image Retrieval - Harris Corner Detector and ORB Descriptor Matching

We have created a reference image index (50 per class) and computed key points using Harris Corner Detector (Harris & Stephens, 1988). The local features of the corner keypoints are extracted using ORB (Oriented FAST and rotated BRIEF) descriptor. Examples of the Harris Corner Detector Keypoints and the resulting ORB descriptor is shown below.



We use a Brute-Force Matcher for a test image to find the k best matches with each reference image with default parameters (k=2). The label of the reference image with the highest good matching points is the model prediction. This image retrieval technique for recognition achieves a maximum accuracy of 28.8% on the 5-class test set. The implementation of this classifier can be found in *TF-Model-ORB.ipynb*. The most important code snippets are below.

```python
def HarrisKeypointDetector(in_image, n=2, w=3, k=0.04, verbose=True):

    harrisImage = cv2.cornerHarris(in_image, n, w, k)
    thresh = 0.05 * harrisImage.max()
    harrisMaxImage = harrisImage > thresh

    height, width = in_image.shape[:2]
    features = []
    for y in range(height):
        for x in range(width):

            if not harrisMaxImage[y, x]:
                continue

            f = cv2.KeyPoint()
            f.pt = (x, y)
            f.response = harrisImage[y,x]
            features.append(f)

    return features
```

```python
def ORBFeatureDescriptor(grayImage, keypoints):
    orb = cv2.ORB_create()
    kp, des = orb.compute((grayImage * 255).astype(np.uint8), keypoints)
    return des
```

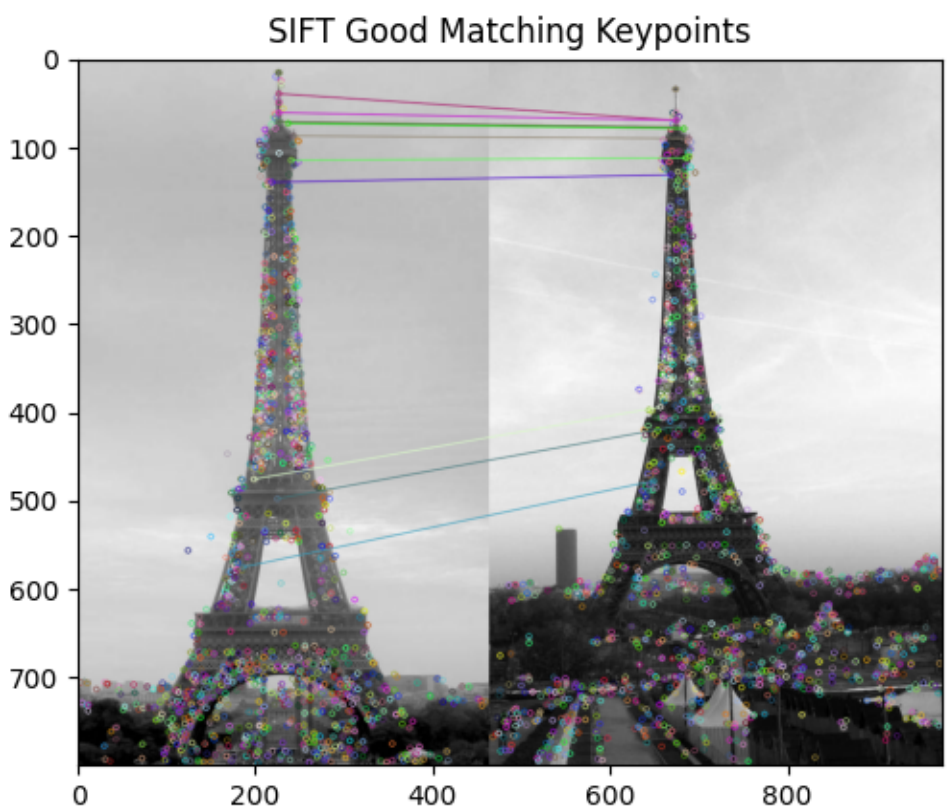
```python
def feature_matching(desc1, desc2):

    # BFMatcher with default params
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(desc1, desc2, k=2)

    # store all the good matches as per Lowe's ratio test.
    good = []
    for m,n in matches:
        if m.distance < 0.7*n.distance:
            good.append(m)

    return len(good)
```

3. Image Retrieval - SIFT Features Matching

We have created a reference image index (50 per class) and computed key points and descriptors using SIFT (Scale Invariant Feature Transform) (Lowe, 1999). We use a Flann-Based Matcher for a test image to find the k best matches with each reference image. The class with the highest aggregate good matching points is the model prediction. This image retrieval technique for recognition achieves an accuracy of 55.2% on the 5-class test set. The code used for this exercise is in *TF-Model-SIFT.ipynb*.

```python
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
SIFT = cv2.xfeatures2d.SIFT_create()
_, des_sift = SIFT.detectAndCompute(image, None)
```

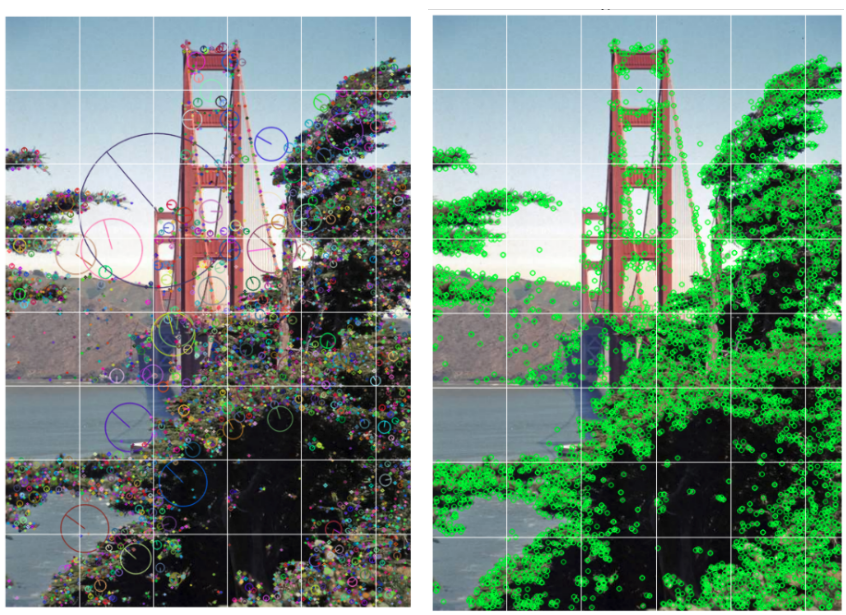
```python
def good_matching(matches, factor):
    good_points = []
    for m, n in matches:
        if m.distance < (factor*n.distance):
            good_points.append(m)

    return len(good_points)


def good_matching(des_a, des_b):
    flann = cv2.FlannBasedMatcher(index_params, search_params)
    matches = flann.knnMatch(des_a, des_b, k=2)

    return matches
```



4. BoW model - SIFT Features

In this approach, we used SIFT features, which detect interest points in an image. We again use our reduced dataset with five landmark classes: Qutub Minar, Eiffel Tower, Golden Gate Bridge, Pakistan Monument, and Niagara Falls. We first calculated our feature vector, length 128, using the SIFT Create function. This gave us different key points and local patches of interest, as shown in the two images of the Golden Gate Bridge shown below. From here, we will take this vector of descriptors and cluster them using KMeans. We have used our 5 classes * 10 to get 50 center means. After this, we will calculate a frequency histogram for the SIFT descriptors based on the prediction from KMeans. This results in a histogram with 50 bins corresponding to each of the cluster centers, thus creating a bag of visual words for each image. An example image of the histogram for a given image is shown below. We then pass this training feature vector into our Multi-Layer Perceptron Neural Network to finally get a testing accuracy of 64.8%. The code discussed here can be found in *TF-Model_SIFT-BoW.ipynb*.

Golden Gate Bridge Points of Interest



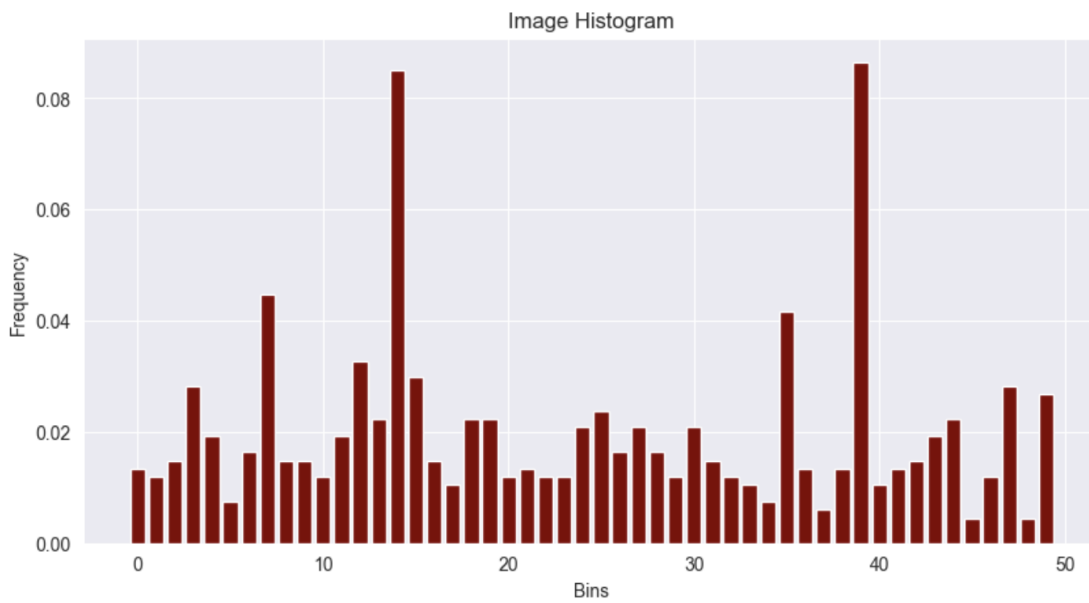
Code Snippets for KMeans Clustering (1), Histogram Generation (2) and Model Fitting (3)

```
# Create 5 * 10 center means ( 5 = num classes we have)
k = np.size(landmarks) * 10

# set the batch size and fit to the descriptors.
batch_size = np.size(os.listdir(img_path)) * 3
kmeans = MiniBatchKMeans(n_clusters=k, batch_size=batch_size, verbose=1).fit(dico)
```

```
kmeans.verbose = False

# This will store the bag of visual words for each image we have
histo_list = []
pbar = ProgressBar()
for leaf in pbar(train.id):
    img = cv2.imread(img_path + str(leaf) + ".jpg")
    kp, des = sift.detectAndCompute(img, None)

    # initialize empty histogram  with k bins. (k is the num classes * 10)
    histo = np.zeros(k)
    nkp = np.size(kp)

    # For each desicriptor we will add 1/128  to the frequency for a bin in the histograms
    for d in des:
        idx = kmeans.predict([d])
        histo[idx] += 1/nkp

    histo_list.append(histo)
```

```
mlp = MLPClassifier(verbose=True, max_iter=600000)
mlp.fit(X, Y)
```

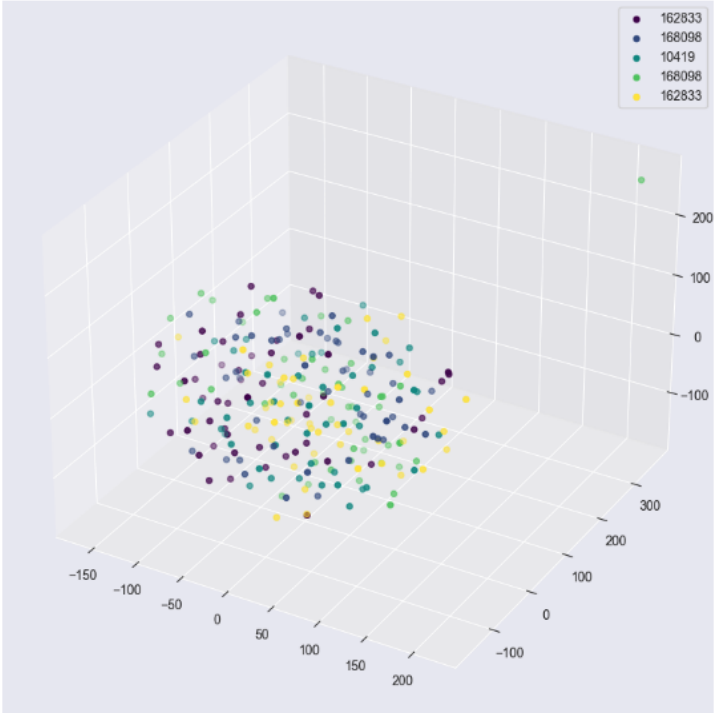
Bag of Visual Words Histogram for a Sample Image



In addition to the histogram, we tried to look at t-distributed stochastic neighbor embedding (t-SNE) to see if we could visualize this high-dimensional feature vector in a lower-dimensional space. We noticed that the 2D representation didn't do much justice to the histogram features we passed into our model. However, we did see a bit of clustering on the 3-D view as the color groups started to come together a little bit, but there isn't a clear pattern we can extract from this.

2-Dimensional t-SNE visualization

3-Dimensional t-SNE visualization



Results and Error Analysis: Non-Learned and Library Features

Feature Type	Testing Accuracy(%)
HOG Feature (51 classes)	19.01
Harris Corner Detector & ORB Descriptor Matching	28.8
SIFT Features Matching	55.2
SIFT - BoW Model	64.8

As we have gone through these various feature techniques to see which one produces the best results, we have found that using SIFT generally yields better results than the HOG features and the Harris Corner Detector (ORB descriptor matching). Between the two approaches of using SIFT features, we showed that first clustering these descriptors to get a histogram of the visual words is a better approach as the model will be looking at the peaks of the histograms passed into the model. One thing that we have not done is passing in a 2-layer SIFT vector into the model, so pursuing this option may yield better results than what we currently have.

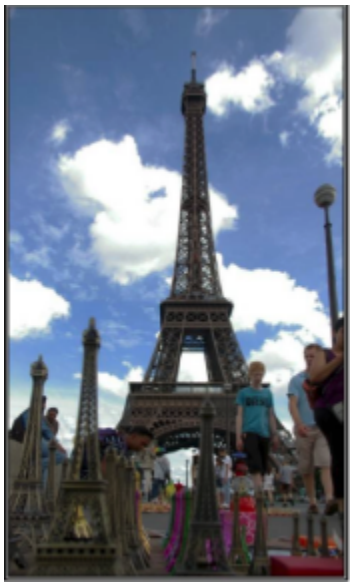
The max accuracy achieved here was around 65%, which meant that we still had enough misclassifications to find some trends that would explain some of these. From the confusion matrix below, we saw that there wasn't a particular class that was misclassified more than the other, which can be slightly attributed to there not being a class imbalance and also indicates that our model can be used as a decent predictor here.

Confusion Matrix for BoW - SIFT Model

True / Pred	**Niagara Falls**	**Qutub Minar**	**Eiffel Tower**	**Pakistan Monument**	**Golden Gate Bridge**
Niagara Falls	15	2	3	3	2
Qutub Minar	0	17	5	1	2
Eiffel Tower	1	3	16	2	3
Pakistan Monument	0	2	1	17	5
Golden Gate Bridge	2	1	4	3	15

However, we did see some instances where our model predicted that the Qutub Minar was the Eiffel tower on five different occasions. When looking at these instances, we found that the angle at which the picture was taken and how much of the monument was captured could explain some of this error. In the left picture below, we can only see the top portion of the Qutub Minar, revealing only the uppermost part of the tower. Since we are not given the complete depiction of the correct landmark, the classification cannot pick up all the relevant features, thus allowing the possibility for misclassification, as seen here. Similarly, we see a small segment of the Golden Gate Bridge in the right image below, which is also misclassified as the Eiffel tower, likely due to the lower part of the bridge looking like a tower.

Continuing to look at the misclassified images, we can see that additional noise/features near the point of interest can disrupt the classification. For example, in the below Eiffel Tower image, we can see other Eiffel Tower toys may create additional points of interest when getting the SIFT features. We can improve such results by trying to blur the background of an image.

Qutub Minar (left) and Golden Gate Bridge (right) Misclassified as the Eiffel Tower		**Eiffel Tower Misclassified as the Pakistan Monument**
		

As mentioned in the data portion of this paper, we are also seeing some images that aren’t pertinent to the overall dataset. As mentioned in the data section, we also have images that aren't pertinent to the overall dataset. These can be images from the landmark's point of view, as shown below to the left, with the picture taken from the Eiffel Tower. Or these can be inside the landmark, as shown in the image of the stone on the right-hand side. Either way, these images

will impact the training of the dataset and will not be classified correctly, showing us that if we went ahead and filtered these images, our model would perform better.

Images Not Pertinent to Classification

Convolutional Neural Network

A breakthrough in building models for image recognition/retrieval came with the discovery that Convolutional neural networks (CNN or ConvNet) could be used to progressively extract higher- and higher-level representations of the image content. Instead of preprocessing the data to derive features like textures and shapes, a CNN takes the image's raw pixel data as input, learns how to extract these features, and infers what object they include. *Baseline-{ConvNet-Architecture}.ipynb* notebooks contains code for all models (*Torchvision 0.11.0*) described in this section.

Image Augmentation

Image augmentation artificially creates training images through different transformations such as random crop, rotation, vertical and horizontal flips, and others. Here are augmentations used for training and validation/test images:

Training Images	torchvision.transforms.RandomResizedCrop(size=(400, 400))
Validation/Test Images	torchvision.transforms.CenterCrop(size=(400, 400)) torchvision.transforms.Resize(size=(400, 400))

Dataset Class

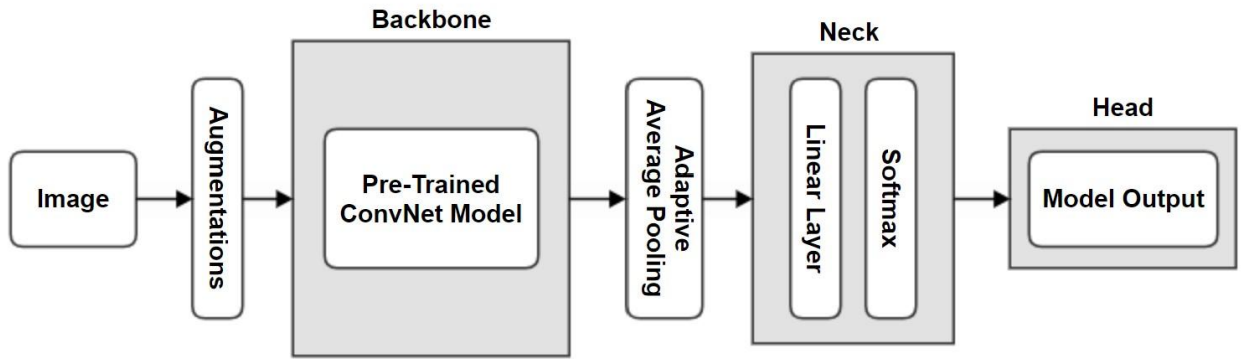
We have built a PyTorch dataset class and implemented the required methods (init, len, getitem). Each dataset item will have an image tensor and corresponding landmark target. The dataset object is then fed into a DataLoader (batch size 32, shuffle=True) for training the model.

```
class GLRDataset:
    def __init__(self, image_paths, targets, augmentations=None):
        self.image_paths = image_paths
        self.targets = targets
        self.augmentations = augmentations

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

    def __getitem__(self, item):
        targets = self.targets[item]
        image = Image.open(self.image_paths[item])
        augmented_image = self.augmentations(image)
        image_tensor = torch.tensor(np.transpose(np.array(augmented_image), (2, 0, 1)).astype(np.float32))

        return {
            'image': image_tensor,
            'targets': torch.tensor(targets),
        }
```

```
df_train = pd.read_csv(TRAINING_DATA_PATH)
train_image_paths = [
    os.path.join(IMAGE_FOLDER_PATH, x + '.jpg') for x in df_train.id.values
]
train_targets = df_train.label.values
train_aug = torchvision.transforms.Compose(
    [
        torchvision.transforms.RandomResizedCrop(size=(IMAGE_SIZE[0], IMAGE_SIZE[1]))
    ]
)
train_dataset = GLRDataset(
    image_paths = train_image_paths,
    targets = train_targets,
    augmentations = train_aug
)
train_data_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=TRAIN_BATCH_SIZE,
    shuffle=True
)
print(f'Total Batches in Train: {len(train_data_loader)}')
```

Model Architecture

Formulating this as a multi-class image recognition problem, we have employed multiple pre-trained ConvNet architectures - ResNet18 (He et al., 2015), InceptionNet (Szegedy et al., 2016), GoogleNet (Szegedy et al., 2014), ResNet34, ResNext101 (Xie et al., 2016), and EfficientNet (Tan & Le, 2019). We have substituted the final linear layer in these architectures with a custom, fully connected layer depending on the number of #classes (51 or 491) in the dataset. We have frozen the first few layers of these models for specific experiments and fine-tuned the remaining model parameters on our task.




```python
class GLRModel(tez.Model):
    def __init__(self, num_classes):
        super().__init__()
        self.convnet = torchvision.models.resnet34(pretrained=True)
        self.convnet.fc = nn.Linear(512, num_classes)

        layer_index = 0
        for child in self.convnet.children():
            if layer_index <= 5:
                for param in child.parameters():
                    param.requires_grad = False
            layer_index += 1

    def forward(self, image):
        outputs = self.convnet(image)
        return outputs
```

Training, Loss Function, & Hyperparameters

We have fine-tuned the model parameters using the PyTorch training framework. Cross entropy loss on validation data for each epoch is used to avoid overfitting. Model weights are stored if validation loss improves compared to the previous best loss. The linear scheduler updates the learning rate after each step for model convergence. The hyperparameters of the best-performing model are described below:

Loss Function	Cross Entropy Loss
Optimizer	AdamW
Scheduler	Linear with 500 warm-up steps
Regularization	Weight Decay (except bias parameters)
Learning Rate	6e-5
Epochs	20
Batch Size	32
Image Size	(400, 400)

```python
def loss_function(outputs, targets):
    return nn.CrossEntropyLoss()(outputs, targets)
```

```python
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=500,
    num_training_steps=num_train_steps
)

best_loss = float('inf')
print(f'Model Training Started')

for epoch in range(EPOCHS):

    total_train_loss = 0.0
    total_train_correct = 0
    for batch_index, data in tqdm(enumerate(train_data_loader), total=len(train_data_loader)):

        image = data['image']
        targets = data['targets']

        image = image.to(device)
        targets = targets.to(device)

        optimizer.zero_grad()
        outputs = model(image=image)

        loss = loss_function(outputs, targets)

        with torch.no_grad():
            total_train_loss += loss.item()
            total_train_correct += count_correct(outputs, targets)

        loss.backward()
        optimizer.step()
        scheduler.step()
```

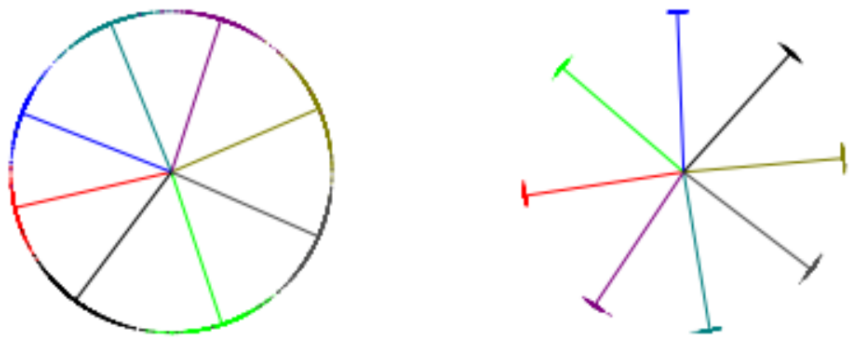
Arcface Loss

In multi-class classification networks, we empirically use softMax and categorical cross-entropy loss at the end of the network. SoftMax transforms numbers into probabilities, and the class with the highest probability is the model prediction. The categorical cross-entropy loss calculates the difference between two distributions of probabilities and is minimized during back-propagation. The drawback of SoftMax is that it does not produce a safety margin, which means that the borders are blurry.

Arcface loss (Deng et al., 2021) has empirically improved model accuracy for multi-class classification problems dealing with thousands of classes. Arcface modifies the softmax function and introduces scale and margin in the computation. Arcface function:

$$L_1 = -\frac{1}{N}\sum_{i=1}^{N}\frac{e^{s\cdot cos(\theta_{y_i}+m)}}{e^{s\cdot cos(\theta_{y_i}+m)} + \sum_{j=1,j\neq y_i}^{n} e^{s\cdot cos\theta_j}}$$

where y_i is the true logit. The angular margin between W_{yi} and x_i is m. This helps penalize the embedding vectors that go far and helps bring the embedding features of a particular class closer. Scale parameter s improve the range that the loss function covers. This leads to smaller intraclass geodesic distance compared to interclass geodesic distance as shown below:



(a) Softmax (b) ArcFace

Evaluation Metric

Our dataset has an apparent class imbalance problem; because of this, we used Accuracy and Balanced Accuracy as our Evaluation metrics.

Results (#51 Classes)

We trained multiple backbone models with different hyperparameters. The table below shows the best-performing model for each backbone architecture. **We achieved 96.13% accuracy on the test set by fine-tuning (frozen layers) the ResNet34 model.**

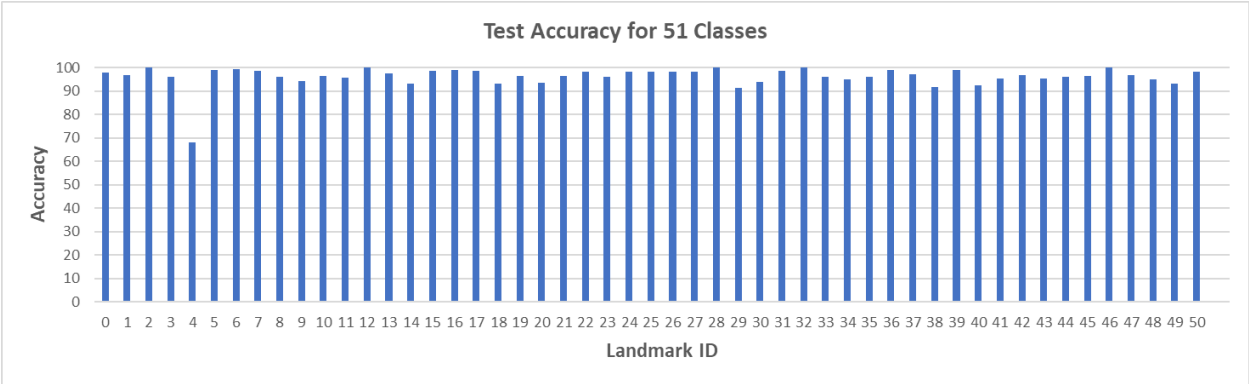
Version	Backbone Model	Training Accuracy	Validation Accuracy	Test Accuracy	Balanced Test Accuracy
v2	**ResNet18**	97.7	95.42	**95.39**	**95.1**
v3	**InceptionNetV3**	97.74	96.54	**96.05**	**95.56**
v4	**GoogleNet**	96.69	94.88	**94.23**	**93.75**
v5	**ResNet34**	97.77	96.2	**96.07**	**95.81**
v5.1	**ResNet34**	97.95	96.22	**96.51**	**96.13**
v6	**ResNext101**	97.47	95.81	**95.83**	**95.29**
v7	**EfficientNetB4**	97.69	96.86	**96.07**	**95.62**

Results (#491 Classes)

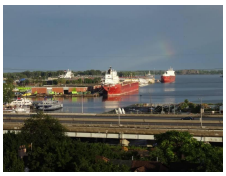
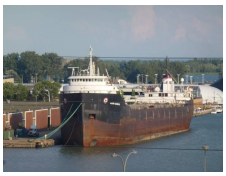
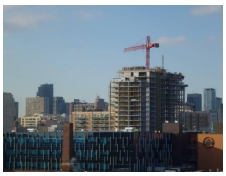
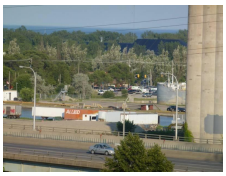
The above results with 51 classes motivated us to include more categories in our dataset. **Fine-tuning the complete ResNext101 architecture (no frozen layers) achieved the highest balanced accuracy of 93.20% on our test set.**

Version	Backbone Model	Training Accuracy	Validation Accuracy	Test Accuracy	Balanced Test Accuracy
v100	**InceptionNetV3**	92.2	92.5	**91.9**	**91.33**
v101	**ResNet34**	94.65	91.86	**91.44**	**91.02**
v102	**ResNext101**	96.22	93.79	**93.51**	**93.20**
v102.1	**ResNext101**	94.4	92.16	**91.31**	**90.94**
v103	**EfficientNetB4**	93.94	91.3	**90.91**	**90.42**
v202	**ResNet34 + Arcface**	93.19	87.53	**87.06**	**86.39**

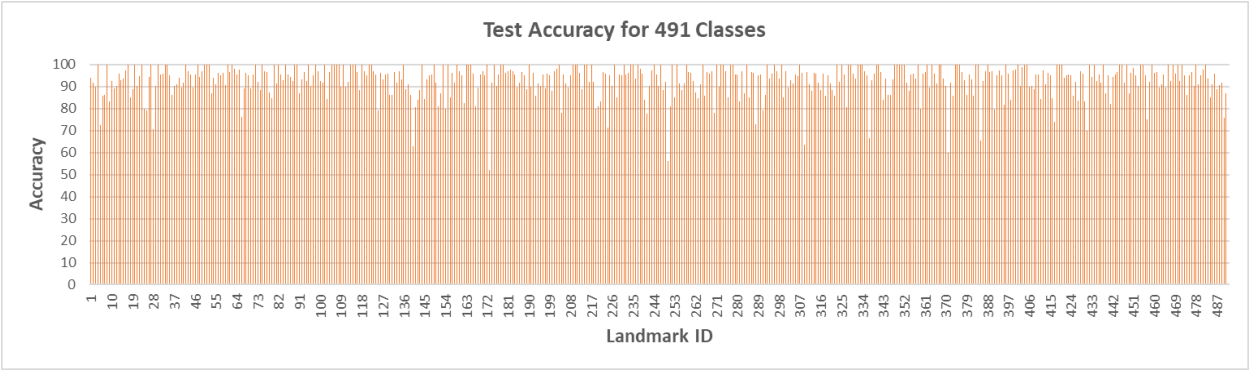
Error Analysis



- 50/51 landmark categories have more than 90% accuracy.
- 5/51 landmark categories has 100% accuracy.
- Only a single class (ID: 4) has low accuracy of 68% on test set. Here are few sample of images of this class:

Correct Prediction				
Incorrect Prediction				

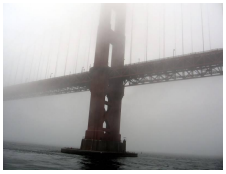
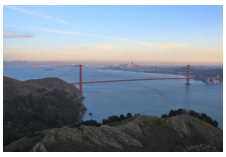
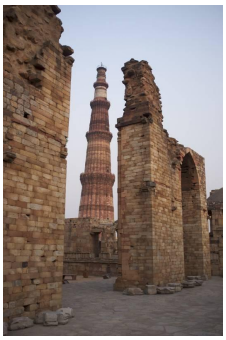
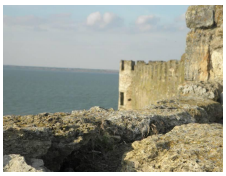
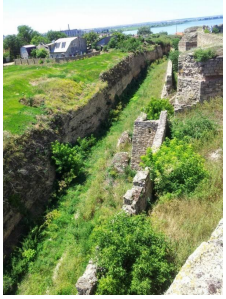
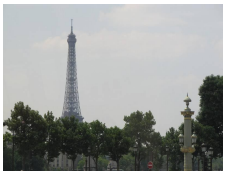
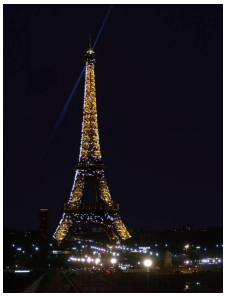
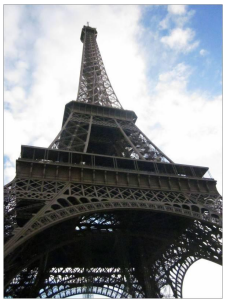
From the above images, it appears that landmark id - 4 is related to a seaport, but there are other classes related to sea or dock category (landmark id - 30). The model is confused for this category and often predicts landmark id - 30 (25% of landmark id - 4 images).



- 370/491 landmark categories have more than 90% accuracy.
- 126/491 landmark categories has 100% accuracy.
- 23/491 landmark categories have less than 80% accuracy, the lowest accuracy is 52.17%.

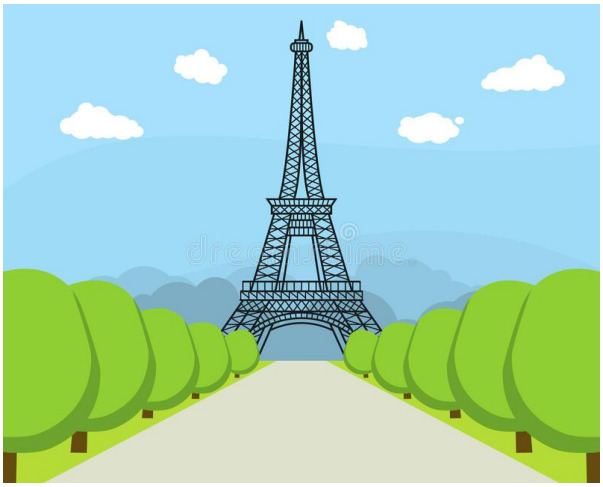
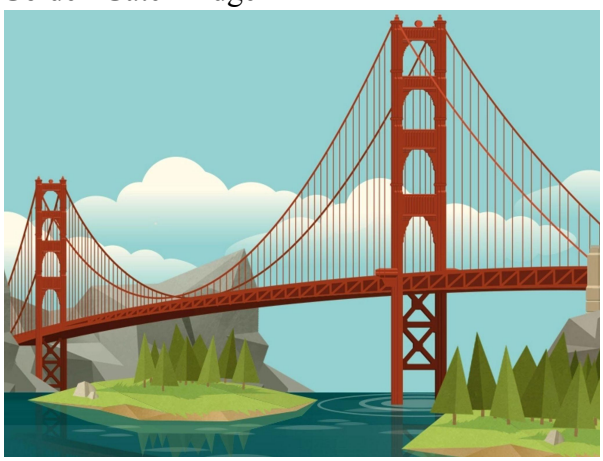
Model training with arcface loss is sensitive to the *margin* and *scale* parameter, which depends on *#classes* in the dataset. We trained multiple models, but none of the experiments improved the model performance. In future work, we will continue exploring the arcface loss by including more classes in the dataset to see if that makes a difference.

The model identifies the structures of these landmarks well, regardless of distance, light, or angles. It usually fails for images related to landmark interiors or unrelated landmark surroundings. Here are samples landmark categories with correct and incorrect predictions:

Landmark	**Correct Prediction**			**Incorrect Prediction**
Golden Gate Bridge				
Qutub Minar				
Bilhorod-Dnister fortress				
Eiffel Tower				None

Generalizability

Original data is split into training, validation, and test set. We have ensured that there is no data leak and final results are reported on test (blind) sets. We avoid overfitting by using Weight Decay (L2 regularization) for model parameters. Additionally, after each epoch, we compute the validation set loss, and only if the model shows improvement on the validation set - are we saving the model. Even increasing the number of classes from 51 to 491, the model accuracy didn't decrease significantly. Further, to prove the model's generalizability, we have downloaded cartoon/painting versions of these landmarks and ran through our best-performing models. The model correctly identifies following sample images:

Eiffel Tower 	Niagara Falls
Qutub Minar	Golden Gate Bridge 

Conclusion

Modeling and recognizing landmarks at a world scale is a valuable yet challenging task. Google Landmarks dataset (GLD) contains images annotated with labels representing human-made and natural landmarks. We started multi-class image recognition with traditional non-learned features, and SIFT histogram BoW model achieved the best accuracy of ~65% on our 5-class dataset. We used a subset of the original GLD data and created two sets of high-frequency images belonging to 51 and 491 landmark categories. We have explored multiple pre-trained ConvNet architectures, and ResNet34 achieved the highest balanced accuracy of 96.13% on 51-class dataset, and RexNext101 achieved the highest balanced accuracy of 93.20% on 491-class dataset. We showed our models' limitations and generalizability, which can be further extended to train on full ~81K landmark categories in GLD.

References

Deng, J., Guo, J., Yang, J., Xue, N., Cotsia, I., & Zafeiriou, S. (2021). ArcFace: Additive Angular Margin Loss for Deep Face Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 10.1109/tpami.2021.3087709

Google Landmark Recognition 2021. (2021). Kaggle. Retrieved December 9, 2022, from https://www.kaggle.com/competitions/landmark-recognition-2021/data

Harris, C., & Stephens, M. (1988). A COMBINED CORNER AND EDGE DETECTOR. *Plessey Research Roke Manor*. http://www.bmva.org/bmvc/1988/avc-88-023.pdf

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. 10.48550/ARXIV.1512.03385

Lowe, D. (1999). Object Recognition from Local Scale-Invariant Features. *Proc. of the International Conference on Computer Vision, Corfu*. https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf

McConnell, R. (1984). Retrieved December 9, 2022, from https://patents.google.com/patent/US4567610

Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: an efficient alternative to SIFT or SURF. *IEEE International Conference on Computer Vision (ICCV)*. http://www.vision.cs.chubu.ac.jp/CV-R/pdf/Rublee_iccv2011.pdf

Singh, A. (2019, September 4). *Feature Descriptor | Hog Descriptor Tutorial*. Analytics Vidhya. Retrieved December 9, 2022, from https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. 10.48550/ARXIV.1602.07261

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going Deeper with Convolutions. 10.48550/ARXIV.1409.4842

Tan, M., & Le, Q. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. 10.48550/ARXIV.1905.11946

torchvision.models — Torchvision 0.11.0 documentation. (n.d.). PyTorch. Retrieved December 9, 2022, from https://pytorch.org/vision/0.11/models.html

Weyand, T., Araujo, A., Cao, B., & Sim, J. (2020). Google Landmarks Dataset v2 -- A Large-Scale Benchmark for Instance-Level Recognition and Retrieval. *arXiv*, *2004*(01804), 1-18. https://arxiv.org/abs/2004.01804

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2016). Aggregated Residual Transformations for Deep Neural Networks. 10.48550/ARXIV.1611.05431

Bag-of-words model with SIFT descriptors. (n.d.). Kaggle. Retrieved December 9, 2022, from https://www.kaggle.com/code/pierre54/bag-of-words-model-with-sift-descriptors/notebook

Lecture 17: Bag-of-Features (Bag-of-Words). (2012, November 27). YouTube. Retrieved December 9, 2022, from https://www.youtube.com/watch?v=iGZpJZhqEME