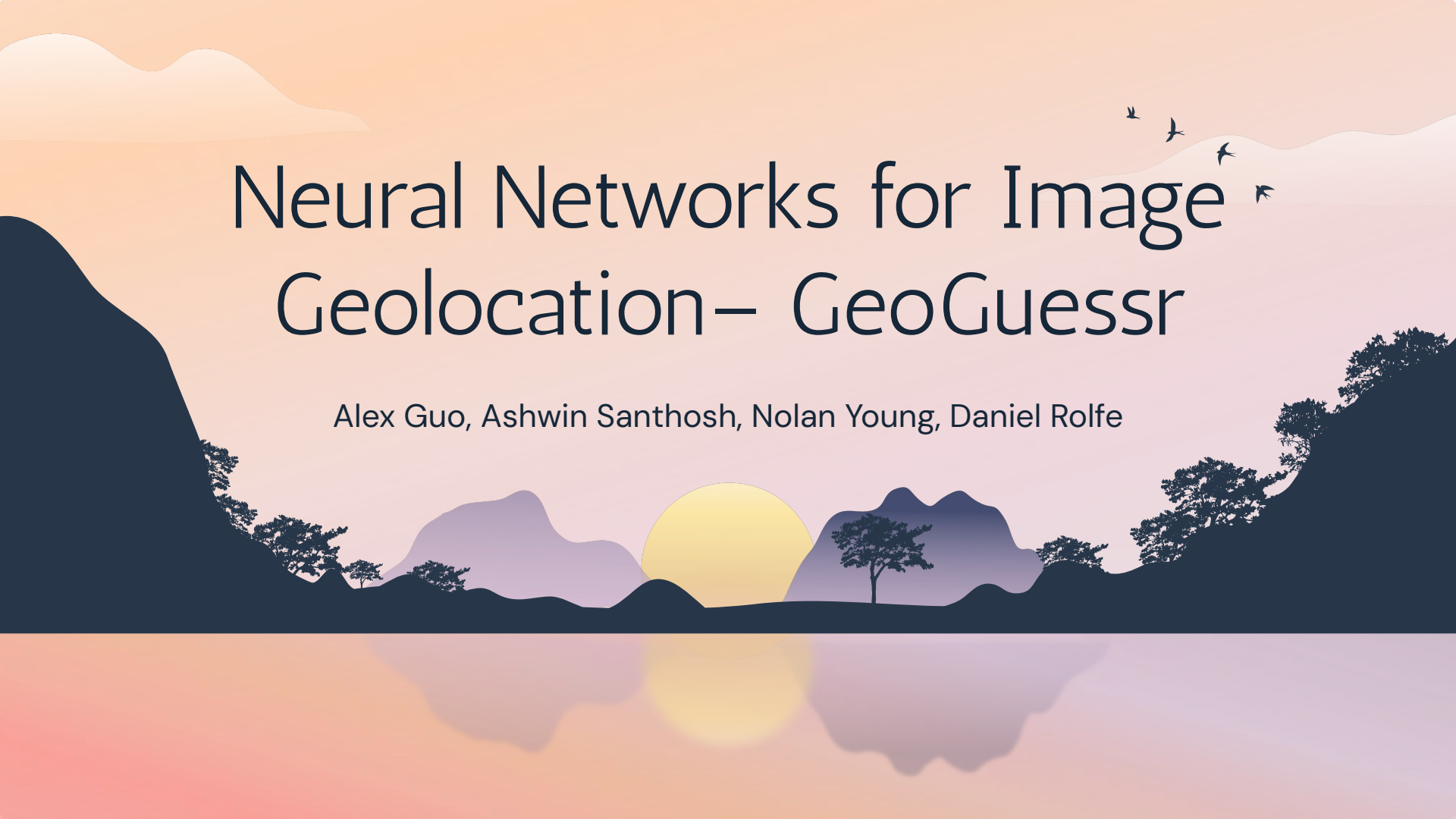
Neural Networks for Image Geolocation– GeoGuessr

Alex Guo, Ashwin Santhosh, Nolan Young, Daniel Rolfe



Problem

- Many images lack a known location, making them unusable for a wide variety of services; a GeoGuessr bot would remedy this.
- Developing an A.I. model which can predict the locations of images will allow for a wide range of utilities, including their utility with:
  - Tourist sites to provide an understanding of locations
  - Determining where images taken during/after a natural disaster were taken to better aid in the reparations process.
  - Locating personal and sentimental images without a known location
  - Being a viable competitor to a human player in GeoGuessr

Figure 1. WorldAtlas images of California (left) and Florida (right) during/after natural disasters

Data Sources

- Due to the nature of the problem with predicting any location within the US and Canada, we opted to utilize the **Google Street View API** to **randomly sample images** within each province/state/territory
- Each image is:
  - 600 x 600 pixels
  - labelled with its geographical coordinates
- Ensures model's capability to predict random localities through the entire province/state/territory and not just highly recognized locations

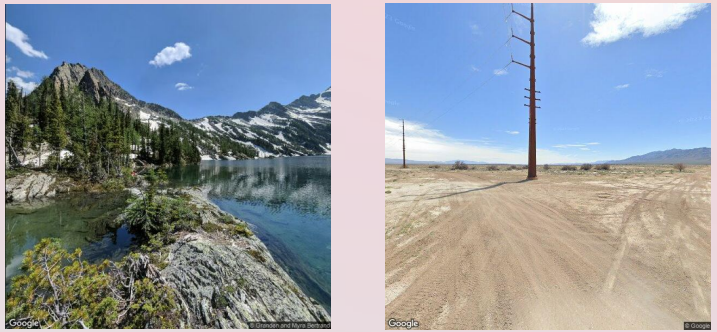


Figure 2. Image sourced from Google Street View API of region in Alberta (left) and Nevada (right)

Data Sources

- Collected **325 images per province/state/territory** (325 * 63 total regions = **20475 total** images)
    - offer a substantial training, validation and test set while being manageable in terms of processing and model training time
    - Provides a diverse representation of each region's unique features
- Since all images were sourced from the same Google Streetview API, **data was all of consistent format** (official Google image formatting guidelines), preventing the need of additional data handling before processing

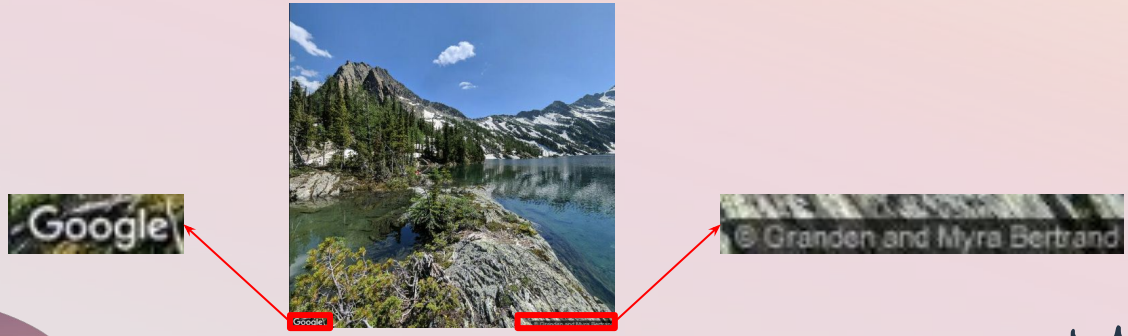
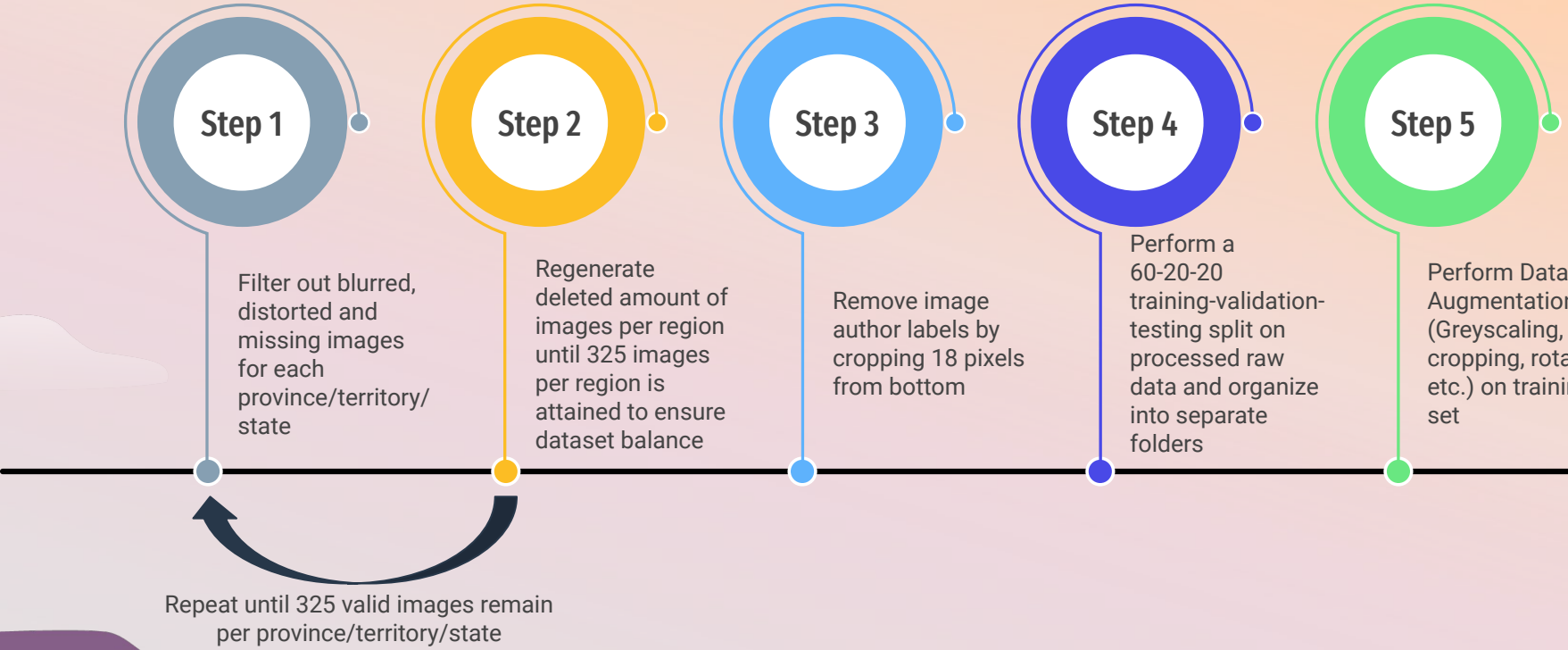


Figure 3. Demonstration of Google's standard image formatting

Data Processing



Data Processing

- Complete_Raw_Dataset
- Test_Data
- Training_Data
- Validation_Data

Dataset folder containing raw collected data and split data in distinct subfolders

Pre-Processed Raw Image

Post-Processed Image

Author labels removed to remove model's bias on image author to determine location

Data Processing (Augmentations)

- Several image augmentations were performed to ensure the model learns the features of each image and prevent overfitting. These augmentations include:
  - Image Cropping
  - Color Jitter
  - Greyscale
  - Random Perspective
  - Random Rotation

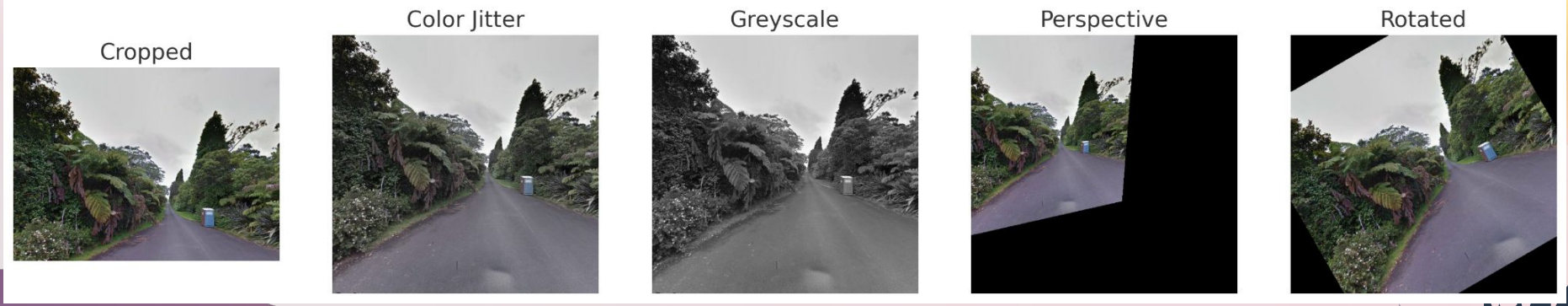
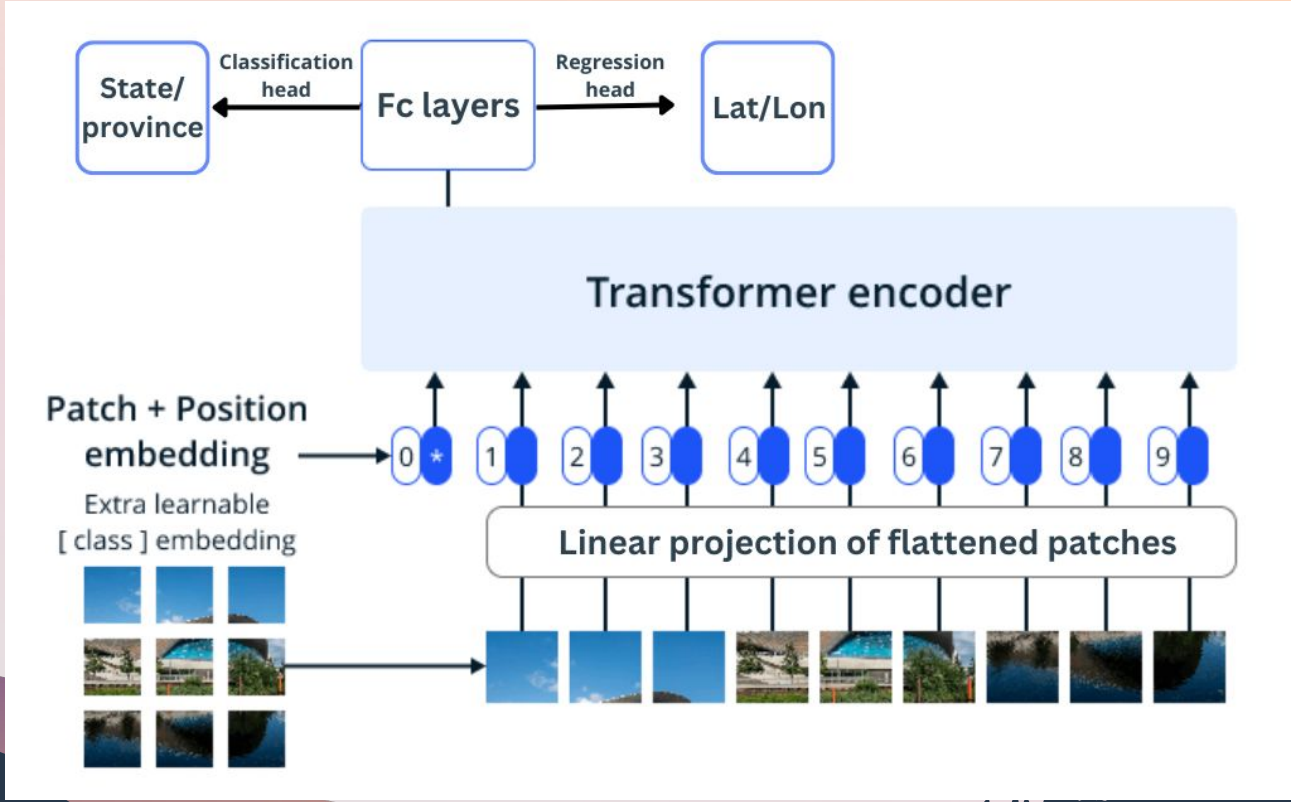


Figure 5. Output images after application of image augmentations.

Primary Model

geoNet: Fully Connected Neural Network



Primary Model - REDO

Convolutional Neural Network (CNN) utilizing PyTorch **AlexNet feature extractions**. This was chosen for the following reasons:

- Effective for Image Data: Spatial hierarchies, translational invariance
- Feature Extraction: Automatic learning from raw pixels
- Transfer Learning: Leverages learned visual features from large datasets (improves accuracy, reduces training time)

Architecture	• 1 Fully Connected Linear Layer: 256*6*6 input size, 244 output features • Classifier Linear Layer: 244 input features, 63 output features (classification) • Regressor Linear layer: 244 input features, 2 output features (regression of latitude and longitude)
Parameters	• Learning Rate: 1e-3 • Batch Size: 32 • Learning Rate Scheduler: Patience = 5, Factor = 0.1 • Epochs: 80

Primary Model

Layer	Type	Input Size	Output Size	Parameters
fc1	Fully Connected	768	512	393,728 (768 * 512 + 512)
bn1	BatchNorm1d	512	512	1,024 (2 * 512)
fc2	Fully Connected	512	256	131,328 (512 * 256 + 256)
bn2	BatchNorm1d	256	256	512 (2 * 256)
fc3	Fully Connected	256	128	32,896 (256 * 128 + 128)
bn3	BatchNorm1d	128	128	256 (2 * 128)
fc4	Fully Connected	128	64	8,256 (128 * 64 + 64)
bn4	BatchNorm1d	64	64	128 (2 * 64)
classifier	Fully Connected	64	63	4,095 (64 * 63 + 63)
regressor	Fully Connected	64	2	130 (64 * 2 + 2)

Primary Model

Vision Transformer (ViT): Image Feature Extraction

Model: "google/vit-base-patch16-224-in21k"

```
model_name = 'google/vit-base-patch16-224-in21k'
processor = ViTImageProcessor.from_pretrained(model_name)
full_dataset = GeoGuessrDataset(root=path, transform=processor)
```

```
import torchvision.models
vit = ViTModel.from_pretrained(model_name)
train_features, train_labels, train_coords = extract_features(train_loader, vit)
val_features, val_labels, val_coords = extract_features(val_loader, vit)
test_features, test_labels, test_coords = extract_features(test_loader, vit)
```

Baseline Model

geoCNN: Simple Convolutional Neural Network

```
class geoCNN(nn.Module):
    def __init__(self):
        super(geoCNN, self).__init__()
        self.name = "CNN"

        # Convolutional layers
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(64)

        # Pooling layer
        self.pool = nn.MaxPool2d(2, 2)

        # Fully connected layers
        self.global_avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc1 = nn.Linear(64, 50)
        self.classifier = nn.Linear(50, 63)
        self.regressor = nn.Linear(50, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.global_avg_pool(x)
        x = x.view(x.size(0), -1)  # Flattening for fully connected layers
        x = F.relu(self.fc1(x))
        province_pred = self.classifier(x)
        coords_pred = self.regressor(x)
        return province_pred, coords_pred
```

Baseline Model

Layer	Type	Input Size	Output Size	Parameters
conv1	Conv2d	(3, H, W)	(32, H, W)	896 (3 * 3 * 32 + 32)
bn1	BatchNorm2d	(32, H, W)	(32, H, W)	64 (2 * 32)
conv2	Conv2d	(32, H/2, W/2)	(64, H/2, W/2)	18,496 (32 * 3 * 3 * 64 + 64)
bn2	BatchNorm2d	(64, H/2, W/2)	(64, H/2, W/2)	128 (2 * 64)
pool	MaxPool2d	(64, H/2, W/2)	(64, H/4, W/4)	0
global_avg_pool	AdaptiveAvgPool2d	(64, H/4, W/4)	(64, 1, 1)	0
fc1	Fully Connected	64	50	3,250 (64 * 50 + 50)
classifier	Fully Connected	50	63	3,213 (50 * 63 + 63)
regressor	Fully Connected	50	2	102 (50 * 2 + 2)

Baseline Model

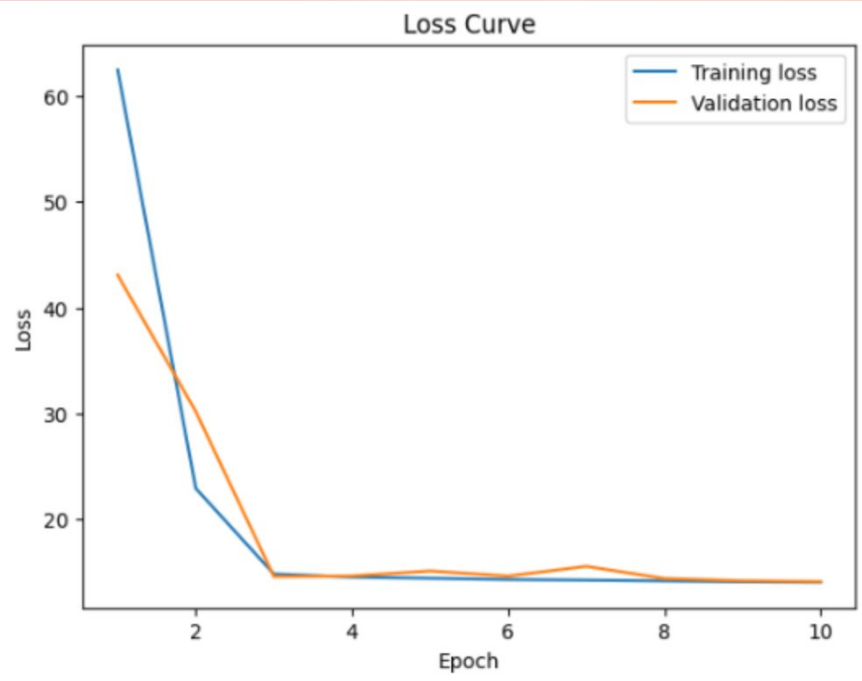
Simple Convolutional Neural Network (CNN). This was chosen for the following reasons:

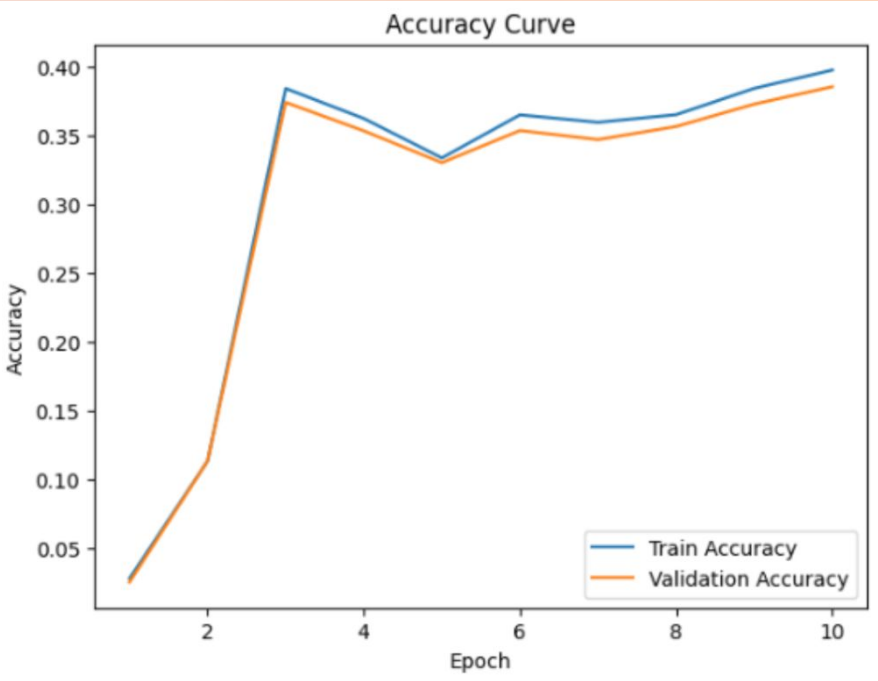
- Ease of Implementation: Simple to design and train
- Low Computational Cost: Efficient in terms of training and prediction time
- Understandability: Easier to analyze and interpret model's behaviour
- Performance Benchmark: Provides baseline for comparison with more complex primary model

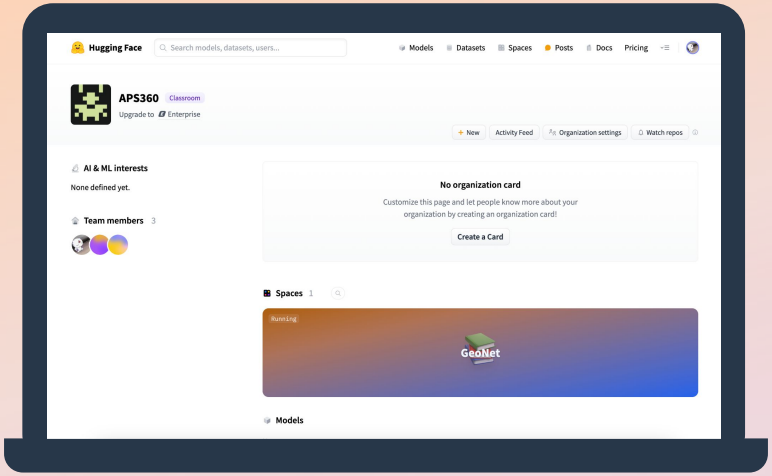
Architecture	• 3 Convolutional Layers • 3 Batch Normalizations: Normalized after each convolutional layer • Residual Block: Involves two 3x3 convolutional layers, each followed by batch normalization and ReLU • Max pooling Layer: 2x2 kernel and stride 2 • 3 Fully Connected Layers
Parameters	• Learning Rate: 1e-2 • Batch Size: 32 • Momentum: 0.9 • Epochs: 15

Baseline Model

- Peak Validation Accuracy Achieved: 38.6%






Hugging Face
Search models, datasets, users...
Models
Datasets
Spaces
Posts
Docs
Pricing
APS360
Classroom
Upgrade to Enterprise
New
Activity Feed
Organization settings
Watch repos
AI & ML interests
None defined yet.
Team members 3
No organization card
Customize this page and let people know more about your organization by creating an organization card!
Create a Card
Spaces 1
Running
GeoNet
Models

Demo

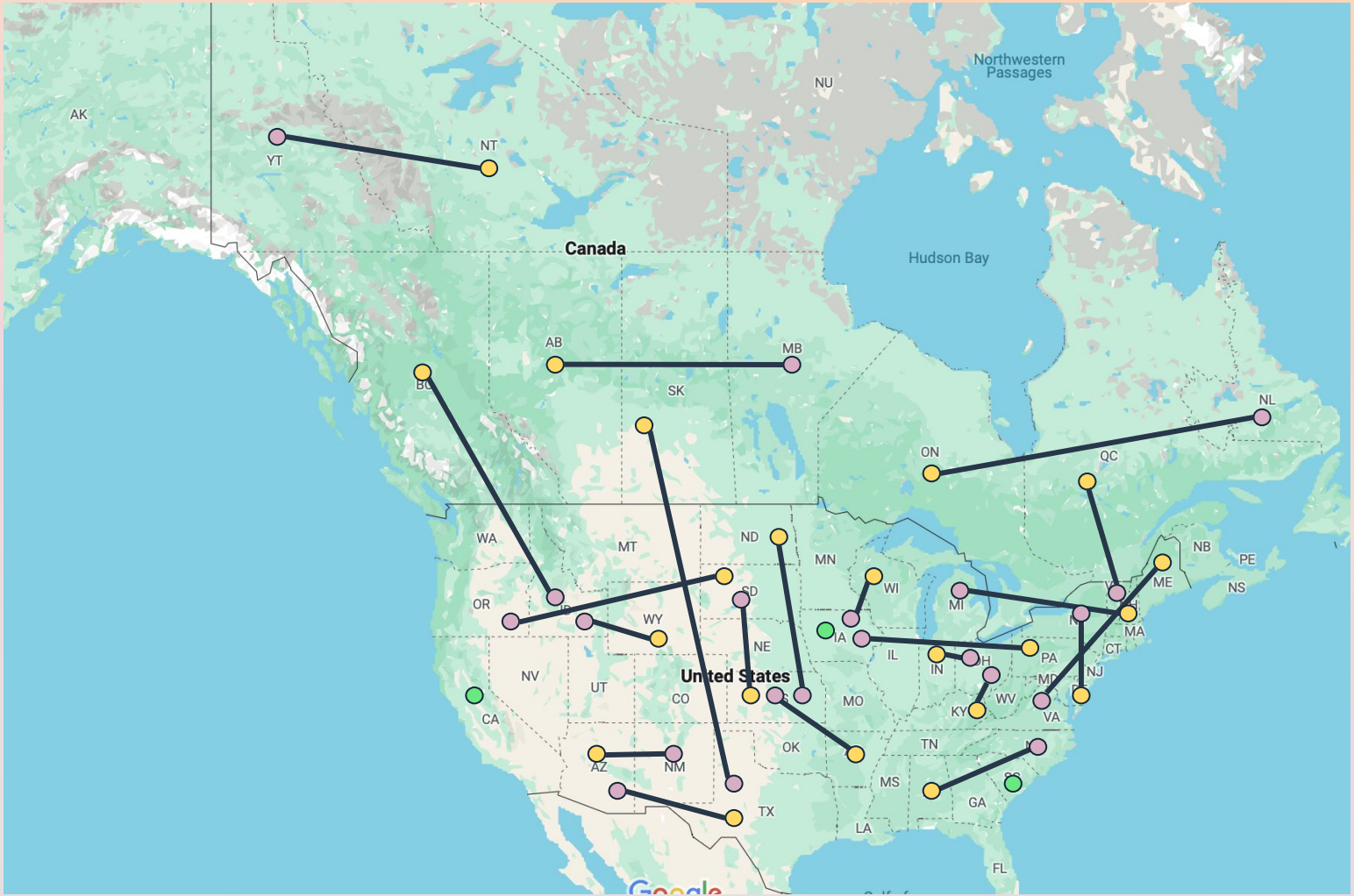
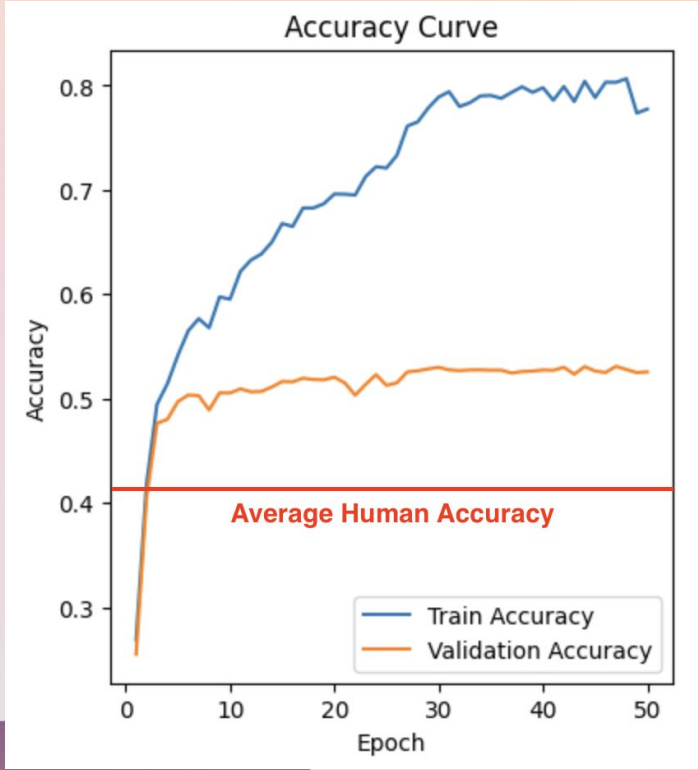


Image Location

Model Prediction (Incorrect)

Model Prediction (Correct)

Quantitative Results - Primary Model



$$Accuracy = e^{-x/1160}$$

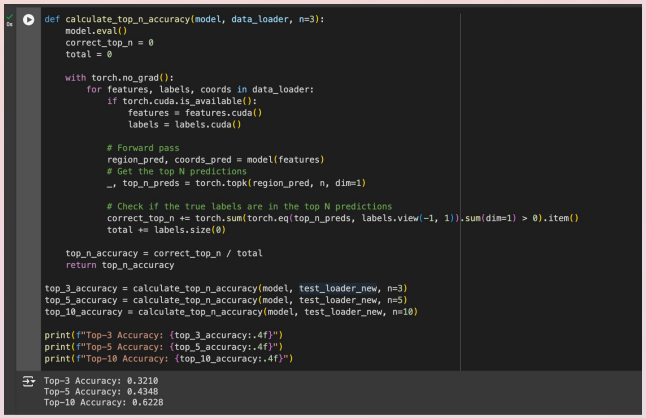
$$x = 6371 * arccos(sin(actual\ latitude) * sin(predicted\ latitude) + cos(actual\ latitude) * cos(predicted\ latitude) * cos(predicted\ longitude - actual\ longitude))$$

Quantitative Results - Primary Model

Top-N classification accuracy:
measures how often the true class label is among the top N predicted classes

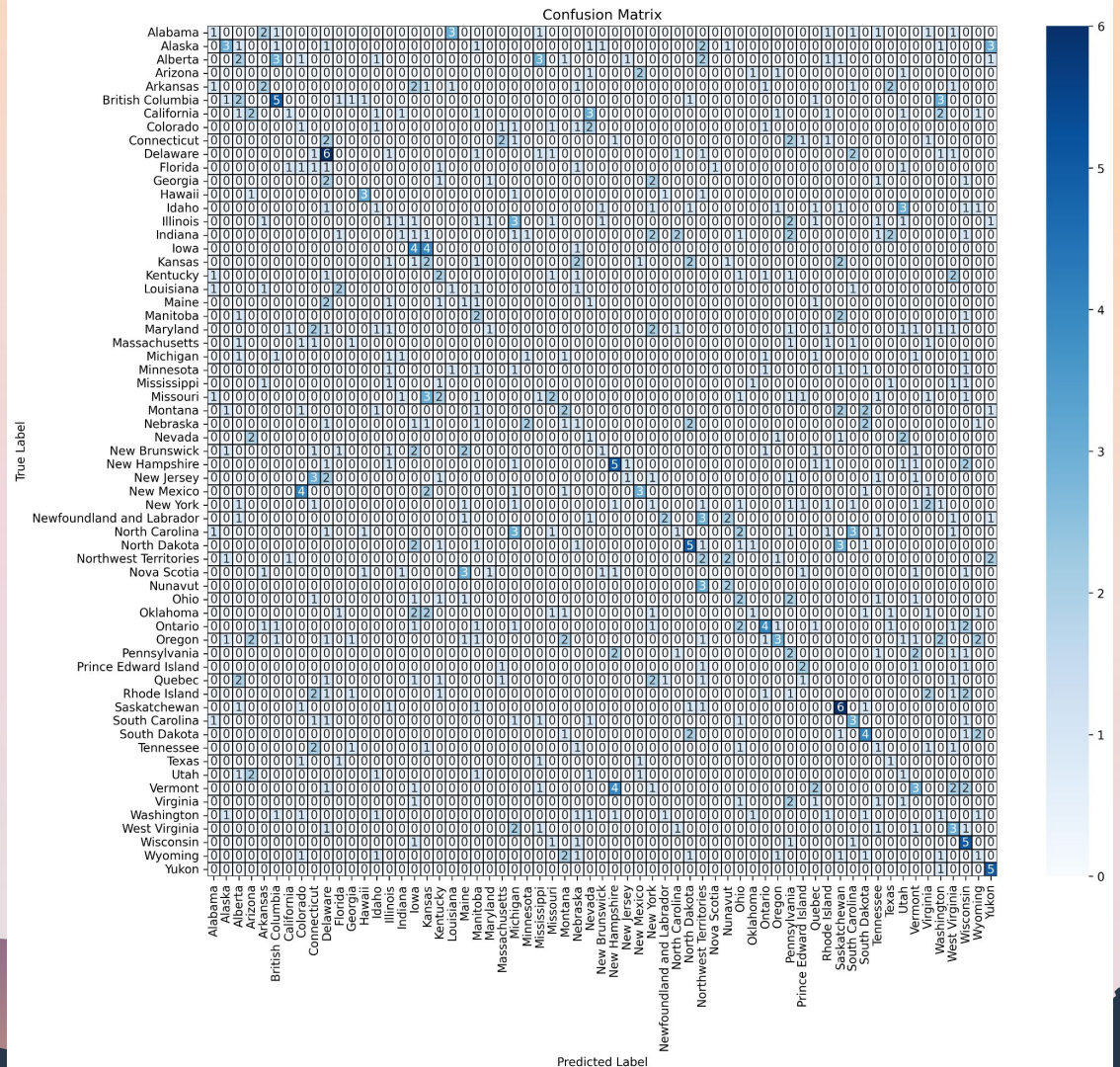
Quantitative Results - Primary Model

Top-3	Top-5	Top-10
32.10%	43.48%	62.28%



```python
def calculate_top_n_accuracy(model, data_loader, n=3):
    model.eval()
    correct_top_n = 0
    total = 0

    with torch.no_grad():
        for features, labels, coords in data_loader:
            if torch.cuda.is_available():
                features = features.cuda()
                labels = labels.cuda()

            # Forward pass
            region_pred, coords_pred = model(features)
            # Get the top N predictions
            _, top_n_preds = torch.topk(region_pred, n, dim=1)

            # Check if the true labels are in the top N predictions
            correct_top_n += torch.sum(torch.eq(top_n_preds, labels.view(-1, 1)).sum(dim=1) > 0).item()
            total += labels.size(0)

    top_n_accuracy = correct_top_n / total
    return top_n_accuracy

top_3_accuracy = calculate_top_n_accuracy(model, test_loader_new, n=3)
top_5_accuracy = calculate_top_n_accuracy(model, test_loader_new, n=5)
top_10_accuracy = calculate_top_n_accuracy(model, test_loader_new, n=10)

print(f"Top-3 Accuracy: {top_3_accuracy:.4f}")
print(f"Top-5 Accuracy: {top_5_accuracy:.4f}")
print(f"Top-10 Accuracy: {top_10_accuracy:.4f}")
```

```
Top-3 Accuracy: 0.3210
Top-5 Accuracy: 0.4348
Top-10 Accuracy: 0.6228
```

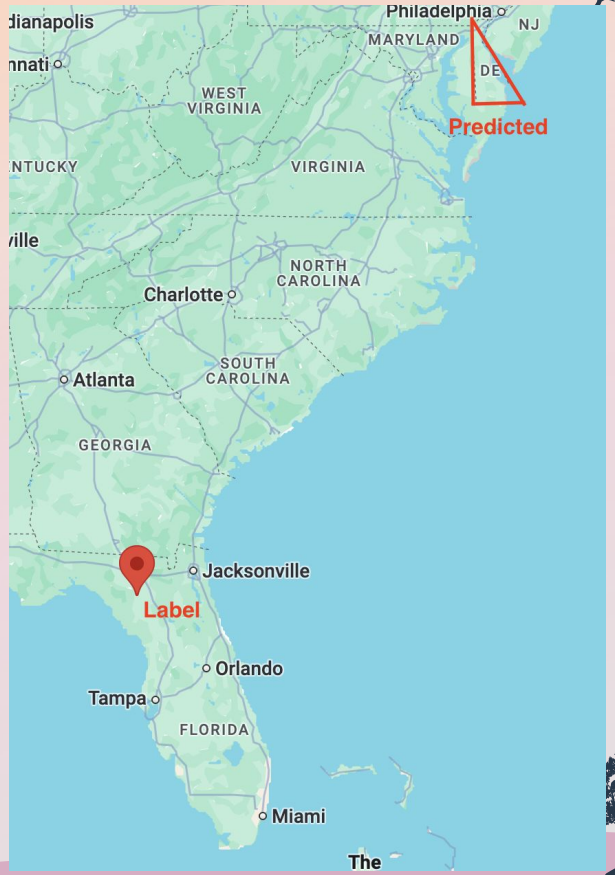

Confusion Matrix
True Label
Predicted Label
Alabama
Alaska
Alberta
Arizona
Arkansas
British Columbia
California
Colorado
Connecticut
Delaware
Florida
Georgia
Hawaii
Idaho
Illinois
Indiana
Iowa
Kansas
Kentucky
Louisiana
Maine
Manitoba
Maryland
Massachusetts
Michigan
Minnesota
Mississippi
Missouri
Montana
Nebraska
Nevada
New Brunswick
New Hampshire
New Jersey
New Mexico
New York
Newfoundland and Labrador
North Carolina
North Dakota
Northwest Territories
Nova Scotia
Nunavut
Ohio
Oklahoma
Ontario
Oregon
Pennsylvania
Prince Edward Island
Quebec
Rhode Island
Saskatchewan
South Carolina
South Dakota
Tennessee
Texas
Utah
Vermont
Virginia
Washington
West Virginia
Wisconsin
Wyoming
Yukon
0
1
2
3
4
5
6

Quantitative Results - Primary Model

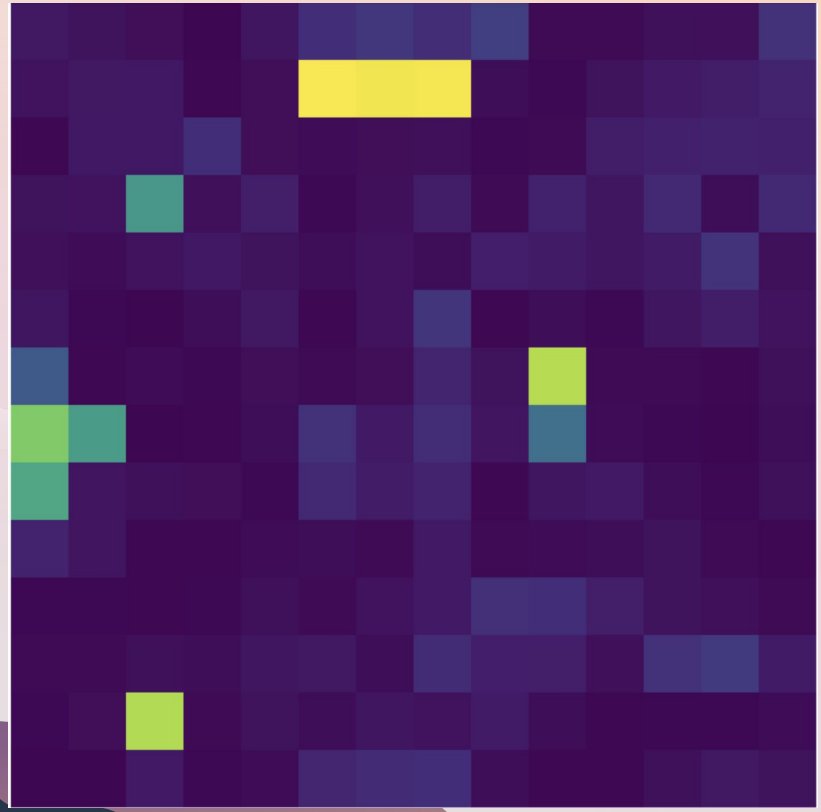
Class	Name	Precision	Recall
7	Colorado	0.0769	0.1111
52	South Dakota	0.2667	0.3636
50	Saskatchewan	0.2857	0.4615
60	Wisconsin	0.1852	0.5000
62	Yukon	0.3571	0.8333

Qualitative Results - Primary Model

Sample Input



Qualitative Results - Primary Model



Qualitative Results - Sample 1

Qualitative Results - Sample 2

Qualitative Results - Sample 3

Takeaways

- Importance of Data Preparation
- Balancing the Dataset
- Geolocation: Challenging, with ongoing research
- Generalization and Next Steps