SprintTwo

February 2, 2025

1 Semester 2, Sprint 2: Dynamic Class Balancing, Duels Mode, Better Model

Owen Kroeger

- Video Link -

- Jira Link -

- GitHub Link - Upgrade the EfficientNet architecture for a better GeoLocation model. Implement dynamic class balancing for balancing out bias in dataset item counts. Duesl mode capability was introduced to allow versus play.

Key Objectives: - Upgrade model - Dynamic Class Balancing - Implement Duels Mode

1.1 Dataset Overview (Same as Before)

We are using a dataset of ~50,000 Google Street View images organized into folders by country. The dataset was split into: - **Training set**: 70% of the images, used to train the model. - **Validation set**: 15% of the images, used to tune hyperparameters and evaluate the model during training. - **Test set**: 15% of the images, used to evaluate the final performance of the trained model.

The images were preprocessed to ensure consistent dimensions and normalization.

[ ]:
```
# Data transformations with augmentation
transform = transforms.Compose([
    transforms.Resize(IMAGE_SIZE),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.
 ↪1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Load dataset
dataset = datasets.ImageFolder(DATA_DIR, transform=transform)
class_names = dataset.classes
```

```
# Split dataset into train, val, and test
train_size = int(0.7 * len(dataset))
val_size = int(0.15 * len(dataset))
test_size = len(dataset) - train_size - val_size

print("Splitting dataset...")
train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size,␣
 ↪val_size, test_size])

# Data loaders
print("Creating data loaders...")
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE)
```

1.2 Model Architecture: EfficientNet_V2_S

EfficientNet_V2_S is a modern convolutional neural network (CNN) optimized for both accuracy and efficiency. It is an improved version of EfficientNet, developed by Google. It provides better accuracy with fewer parameters, faster training, and a smaller model size, making it more efficient.

[ ]:
```
print("Initializing model...")
model = efficientnet_v2_s(weights="IMAGENET1K_V1")
# Replace the classifier layer to match the number of classes
model.classifier[1] = nn.Linear(model.classifier[1].in_features,␣
 ↪len(class_names))
model = model.to(DEVICE)
```

1.3 Training the Model

The model is implemented into the standard deep learning process.

- **Forward Pass:** Images are passed through EfficientNetV2-S to get predictions.
- **Loss Calculation:** Measures how far predictions are from true labels.
- **Backward Pass:** Computes gradients for optimization.
- **Weight Updates:** `optimizer.step()` updates the model based on gradients.

[ ]:
```
def train_one_epoch(epoch):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for inputs, labels in tqdm(train_loader, desc=f"Training Epoch {epoch + 1}/
 ↪{EPOCHS}"):
        inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)

        # Forward pass
        outputs = model(inputs)
```

```
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Update metrics
        running_loss += loss.item() * inputs.size(0)
        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)

    return running_loss / len(train_loader.dataset), 100 * correct /␣
↪len(train_loader.dataset)
```

1.4 Dynamic Class Balancing

The model uses a dynamic class balancing strategy via a **WeightedRandomSampler**, ensuring that all classes contribute equally to training, even if they are imbalanced in their count.

- `dataset.targets` contains the class label (country) for each image.
- `np.bincount(dataset.targets)` counts how many times each class appears.
- `class_weights = 1.0 / class_counts` assigns higher weights to underrepresented classes.
- `class_weights /= class_weights.sum()` ensures the sum of all weights is 1.0, preventing sampling issues.
- Each image gets a sampling weight based on its class.
- **WeightedRandomSampler** ensures that rare classes appear more frequently than overrepresented classes.

[ ]:
```
def get_class_balanced_sampler(dataset):
    """
    Automatically balance class sampling by computing weights inversely␣
↪proportional
    to class frequencies and return a WeightedRandomSampler.
    """
    class_counts = np.bincount(dataset.targets)  # Count occurrences of each␣
↪class
    class_weights = 1.0 / class_counts           # Inverse of class frequency
    class_weights /= class_weights.sum()         # Normalize weights
    sample_weights = [class_weights[label] for label in dataset.targets]  #␣
↪Assign weight to each sample
    return WeightedRandomSampler(sample_weights, num_samples=len(dataset),␣
↪replacement=True)\
```

1.5 Duels Mode Integration

GeoGuessr is both a single and multiplayer game. **Duels** mode allows players to compete against each other, where they are given the same location and compare scores on who gets closer.

1.5.1 Wait for Transition

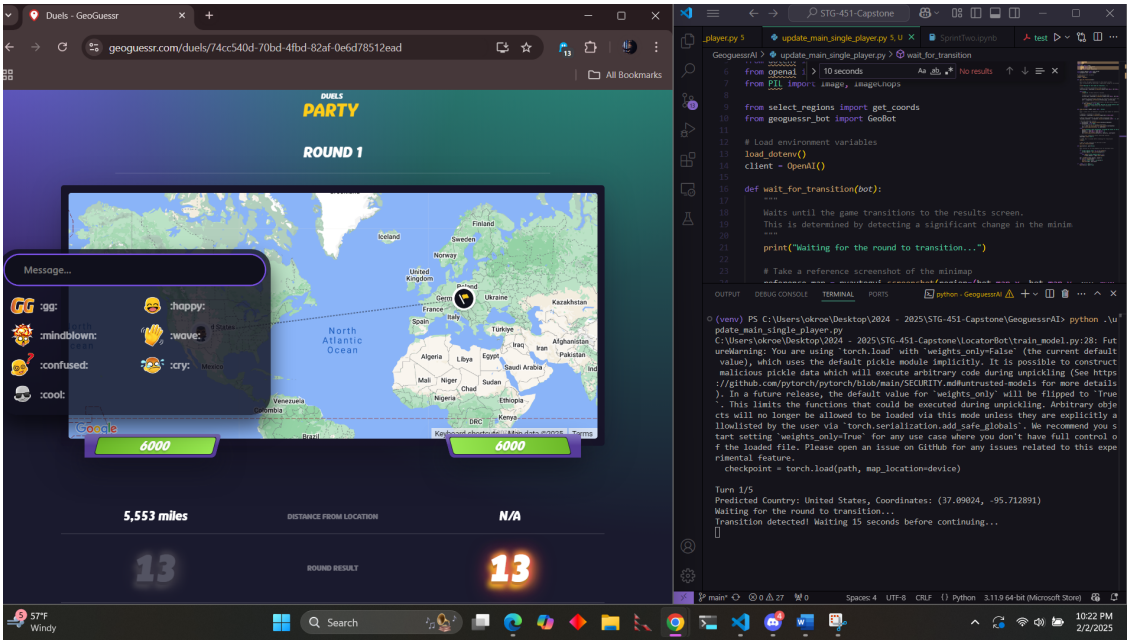
Instead of the user being able to control when to go to the next round, a **Duels** mode round ends when both players have made a guess. Then a comparison, between round screen shows the round result, and a countdown starts before moving on to the next round.

The bot used to be able to click next when it made a guess, but must now wait until the round is over.

A `wait_for_transition` function was implemented that tracks the minimap in the bottom left. Once this map has disappeared / changed, it knows that the round has ended, and it waits the 15 seconds until the next round starts.

[ ]:
```python
def wait_for_transition(bot):
    """
    Waits until the game transitions to the results screen.
    This is determined by detecting a significant change in the minimap area.
    """
    print("Waiting for the round to transition...")

    # Take a reference screenshot of the minimap
    reference_map = pyautogui.screenshot(region=(bot.map_x, bot.map_y, bot.
↪map_w, bot.map_h))

    while True:
        sleep(0.5)  # Check every 0.5 seconds

        # Capture a new screenshot of the minimap area
        current_map = pyautogui.screenshot(region=(bot.map_x, bot.map_y, bot.
↪map_w, bot.map_h))

        # Compute the difference between the reference and current image
        diff = ImageChops.difference(reference_map, current_map)

        # If there's a noticeable difference, the map likely changed␣
↪(transition detected)
        if diff.getbbox() is not None:
            print("Transition detected! Waiting 15 seconds before continuing...
↪")
            sleep(15)  # Additional wait after transition
            break
```


```
Turn 2/5
Predicted Country: Germany, Coordinates: (51.165691, 10.451526)
Waiting for the round to transition...
```



1.6 Challenges

- Updated model has still not finished training, although showing much higher accuracies so far.
- Duels mode bot has a slight coordinates calibration issue. All coordinate guesses are shifted to the left a few degrees.

1.7 Next Steps

- Train model on a Lab computer to finish training quicker and save the model locally.
- Update Duels mode to run for infinite rounds (until game ends).
- Fix calibration issue with coordinates.
- Region guessing?