SprintOne

January 19, 2025

1 Semester 2, Sprint 1: Upgrading the model with EfficientNet

Owen Kroeger

- Video Link -

- Jira Link -

- GitHub Link - Using the EfficientNet architecture and a dataset of labeled geolocation images, we train, validate, and test an upgraded model.

Key Objectives: - Upgrade model - Integrate into GeoguessrBot

1.1 Dataset Overview

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

EfficientNet-B0, a compact and efficient architecture pre-trained on ImageNet, was initialized. The final classification layer (classifier[1]) was replaced with a custom layer for dataset-specific classification.

[ ]:
```
print("Initializing EfficientNet_B0...")
model_effnet = efficientnet_b0(weights="IMAGENET1K_V1")
model_effnet.classifier[1] = nn.Linear(model_effnet.classifier[1].in_features,
 ↪len(class_names))
model_effnet = model_effnet.to(DEVICE)

optimizer_effnet = torch.optim.Adam(model_effnet.parameters(), lr=LEARNING_RATE)
```

1.3 Training the Model

We train the EfficientNet model on the training dataset for a specified number of epochs. The training process involves: 1. **Forward Pass**: Feed images into the model to calculate predictions. 2. **Loss Calculation**: Compute the difference between predictions and actual labels using CrossEntropyLoss. 3. **Backward Pass**: Update model parameters to minimize the loss using the Adam optimizer.

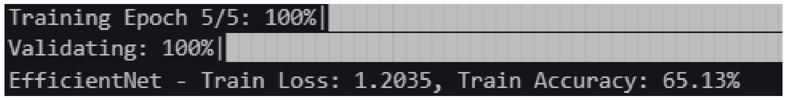
We validate the model after each epoch to monitor its performance.

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

        # Update progress bar
        pbar.set_postfix(Loss=running_loss / total, Accuracy=100 * correct /
 ↪total)

    epoch_loss = running_loss / len(train_loader.dataset)
    epoch_acc = 100 * correct / len(train_loader.dataset)
    return epoch_loss, epoch_acc

# Validation function
def validate(model):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in tqdm(val_loader, desc="Validating"):
            inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)

            # Forward pass
            outputs = model(inputs)
            loss = criterion(outputs, labels)

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

    val_loss = running_loss / len(val_loader.dataset)
    val_acc = 100 * correct / len(val_loader.dataset)
    return val_loss, val_acc
```

1.4 Evaluating the Model

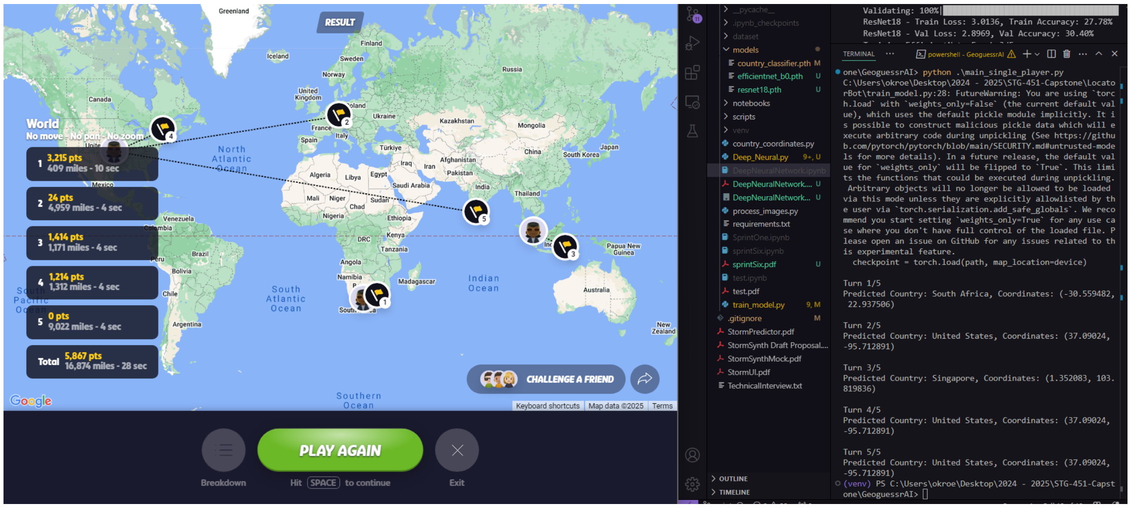
Once the model is trained, we evaluate it on the test dataset to measure its accuracy. During training, the EfficientNet model was up to 60% accuracy.



1.5 Integration with GeoBot

The EfficientNet model, 'efficientnet_b0.pth' was integrated into the GeoguessrBot. The fully connected layer (fc) was replaced with a Sequential block consiting of: 1. A linear layer with 512 hidden units. 2. A ReLU activation. 3. Dropout for regularization. 4. A final linear layer mapping to the number of classes.

[ ]:
```
# Training code
model.fc = nn.Linear(model.fc.in_features, num_classes)

# Old vs New code

model.fc = nn.Sequential(
    nn.Linear(model.fc.in_features, 512),
    nn.ReLU(),
    nn.Dropout(p=0.5),
    nn.Linear(512, num_classes)
)

# Loading code
model.fc = nn.Linear(model.fc.in_features, len(class_names))

model.fc = nn.Sequential(
    nn.Linear(model.fc.in_features, 512),
    nn.ReLU(),
    nn.Dropout(p=0.5),
    nn.Linear(512, len(class_names))
)
```



1.6 Challenges

- **Class Imbalance**: The USA is still much overrepresented in the dataset, leading to a large bias and overfitting. The EfficientNet model boasts 60% accuracy, but only because it guesses the USA a lot, resulting in a "fake" accuracy.

1.7 Next Steps

1. Implement techniques to handle class imbalance, such as oversampling or weighted loss.
2. Increase the resolution of the images and train on a better computer for more accurate results.
3. Expand the GeoguessrBot to play Duels mode.