Bimonthly Progress Report II

Introduction:

Human trafficking is a pervasive global problem that affects millions of individuals, often occurring in concealed environments like hotel rooms. Organizations and law enforcement agencies that are dedicated to fighting human trafficking face significant challenges in identifying the locations of victims based on limited visual evidence. Frequently, photographs taken in hotel rooms serve as the only clues to identify these locations. However, the process of manually analyzing these images is time-consuming, labor-intensive, and prone to errors. With thousands of hotels, each with unique room configurations, accurately pinpointing locations requires an automated, scalable solution. This project aims to develop a machine learning model capable of matching images of hotel rooms to their respective hotels. By providing data-driven, efficient solutions, this project will enable faster response times and improve the success rate of victim rescue operations. With our tools and expertise, the current inefficiencies will be addressed and will set an example for how advanced computer vision can be used to tackle societal human rights issues

Progress:

Data Preparation:

The competition provided 15 GB of 1.1 million images encompassing fifty thousand hotels, with each image labeled with a unique hotel ID. Initial data acquisition faced challenges such as local resource limitations, including slow processing due to CPU constraints and limited disk space. Ultimately, the team switched to Kaggle for its ease of use and ability to directly submit to the competition to obtain MAP@5 score for performance analysis and model comparison. The images have significant variation in the interior designs of hotel rooms, lighting variation, and class imbalance where some hotels have significantly more images than others. Considering these challenges, an effective feature extraction and augmentation approach is needed to enable model generalization and improve predictive accuracy. The images from are standardized to a fixed size (384x384) via padding, resizing, and normalization, ensuring uniformity regardless of original dimensions or orientations.

To improve robustness and ensure generalization across varying lighting conditions, viewpoints, and occlusions, we incorporated Albumentations for data alteration. This included geometric transformations, brightness adjustments, and distortions directly applied to the images. Various alteration techniques were tested, including GridDropout, CLAHE (Contrast Limited Adaptive Histogram Equalization), and partial grayscale conversions. Using RGBShift at varying intensities and refined parameters for ShiftScaleRotate to apply color alteration specifics, we were able to adjust the model to improve our overall score.

Methods:

Baseline Model

To establish a baseline for comparison, we developed a random assignment of predictions, which serves to obtain a benchmark model to test against our machine learning models. This model was constructed by sorting the dataset by chain_id and hotel_id to organize the data. To assign five random hotel IDs to hotel image URL links, a Python library was used to perform this random assignment, ensuring fair distribution across hotels. The random selection model established a low baseline MAP@5 score of 0.012, helping us measure how well our machine learning models improved over a naive prediction method. This approach provides the team with a controlled way to assess whether our trained model has evolved as we expected.

ResNet-34

The ResNet architecture is primarily leveraged as our backbone model moving forward, where it uses a pre-trained library from the timm library. ResNet, pre-trained on ImageNet, provided robust general pattern recognition capabilities, which were then fine-tuned on the hotel room dataset. This is done through the ResNet architecture, where the input layer preprocesses the images to a uniform size to ensure model consistency. The initial 7x7 convolutional layer extracts low-level features such as edges and textures. Multiple residual blocks with skip connections are used to allow the model to learn deep, complex representations while mitigating the vanishing gradient problem. Global pooling then aggregates spatial features into a fixed-length vector which is then fed into a fully connected layer to predict the hotel IDs.

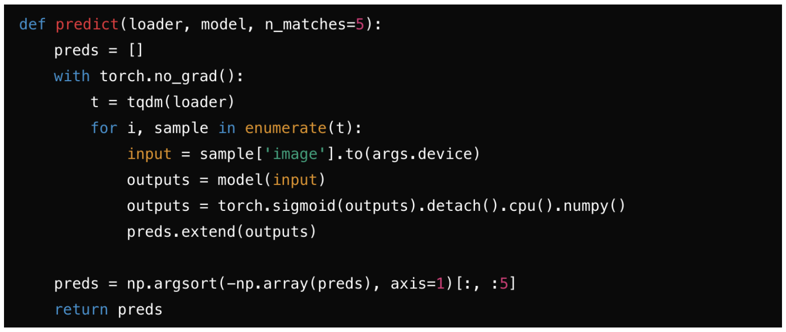
ResNet-34 directly counters the problem of vanishing gradients through residual learning, equipping it well for application to deep learning. To enhance classification accuracy, we used Test-Time Augmentation (TTA), a technique that improves accuracy by making multiple inferences for each image with slight variations and averaging the predictions. Our TTA included horizontal flipping, three forward passes for each image, and prediction averaging, which yielded an 0.015 accuracy increase without additional model retraining.

Important parameters were selected to trade off performance and efficiency. The batch size was 64 for stable gradient updates and computational convenience. The model dynamically adjusted to the number of unique hotel IDs in the data. Sigmoid activation was employed for probability output, and CUDA acceleration where available.

Three TTA transformations were selected to trade off between inference speed and accuracy gains.

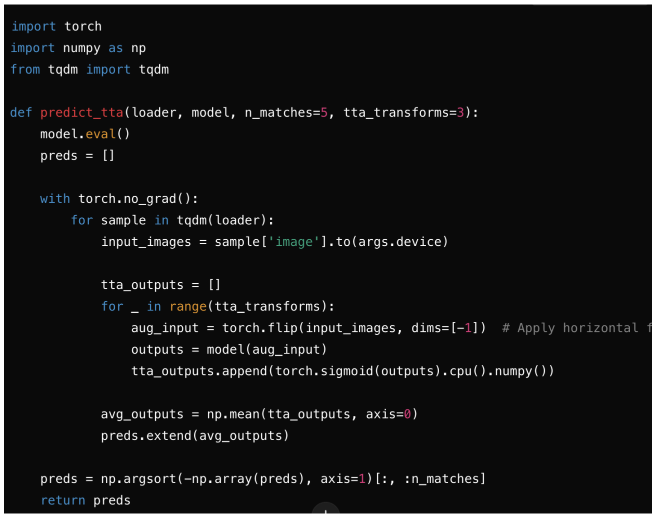
We enhanced model stability through the use of ResNet-34 in combination with TTA, which produced more stable predictions across various image conditions. The way TTA works is that it applies transformations to test images to improve model generalization. We want to add this as it averages predictions from multiple augmented versions of the same image to reduce uncertainty. Specifically, it returns the top 5 most confident predictions per image and then generates 3 augmented versions of each image to improve robustness. Thus, multiple augmented versions of a test image are generated and passed through the model. Where there are predictions for each augmented version. Lastly, the final prediction is obtained by averaging or voting among these predictions to increase stability and reduce noise.

Replaced code:



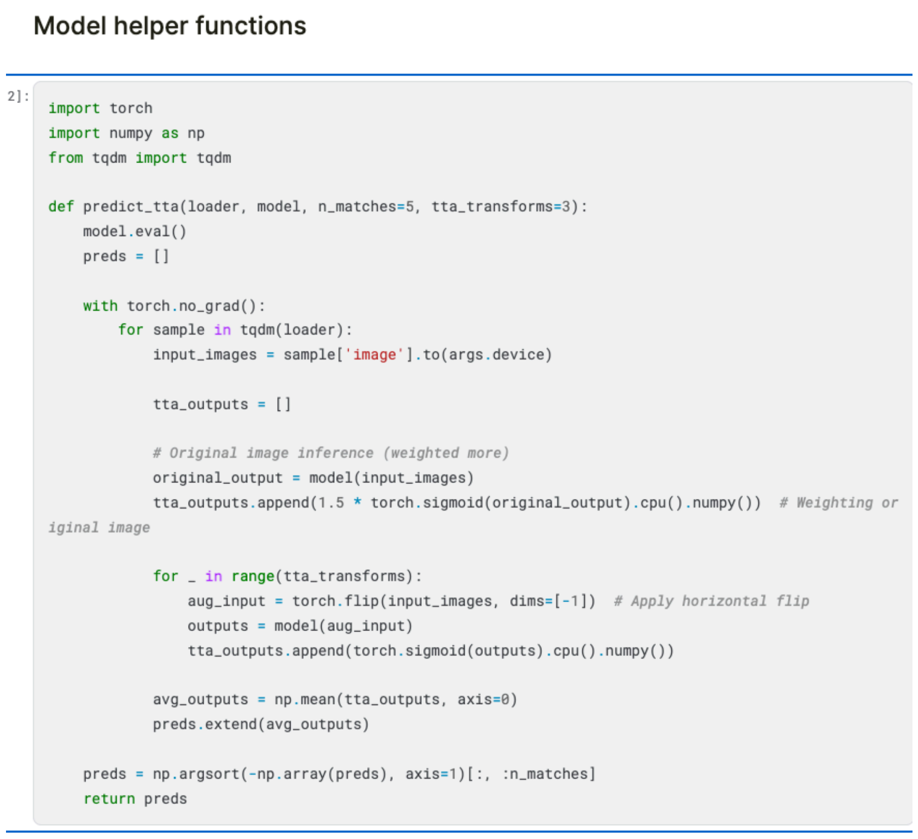
```
def predict(loader, model, n_matches=5):
    preds = []
    with torch.no_grad():
        t = tqdm(loader)
        for i, sample in enumerate(t):
            input = sample['image'].to(args.device)
            outputs = model(input)
            outputs = torch.sigmoid(outputs).detach().cpu().numpy()
            preds.extend(outputs)

    preds = np.argsort(-np.array(preds), axis=1)[:, :5]
    return preds
```

Bimonthly Project Report I code:



```
import torch
import numpy as np
from tqdm import tqdm

def predict_tta(loader, model, n_matches=5, tta_transforms=3):
    model.eval()
    preds = []

    with torch.no_grad():
        for sample in tqdm(loader):
            input_images = sample['image'].to(args.device)

            tta_outputs = []
            for _ in range(tta_transforms):
                aug_input = torch.flip(input_images, dims=[-1])  # Apply horizontal f
                outputs = model(aug_input)
                tta_outputs.append(torch.sigmoid(outputs).cpu().numpy())

            avg_outputs = np.mean(tta_outputs, axis=0)
            preds.extend(avg_outputs)

    preds = np.argsort(-np.array(preds), axis=1)[:, :n_matches]
    return preds
```

Updated Bimonthly Project Report II code:



Model helper functions

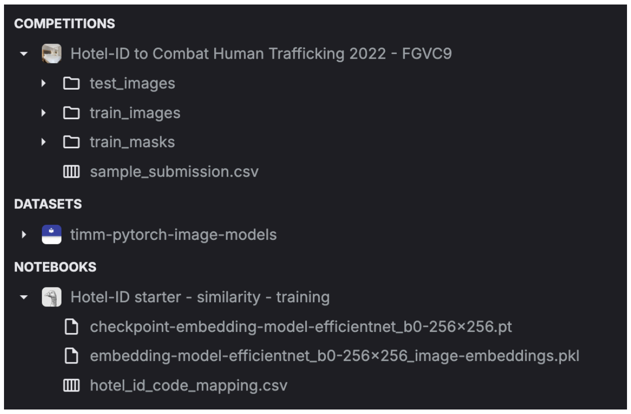
```
2]:
import torch
import numpy as np
from tqdm import tqdm

def predict_tta(loader, model, n_matches=5, tta_transforms=3):
    model.eval()
    preds = []

    with torch.no_grad():
        for sample in tqdm(loader):
            input_images = sample['image'].to(args.device)

            tta_outputs = []

            # Original image inference (weighted more)
            original_output = model(input_images)
            tta_outputs.append(1.5 * torch.sigmoid(original_output).cpu().numpy())  # Weighting or
iginal image

            for _ in range(tta_transforms):
                aug_input = torch.flip(input_images, dims=[-1])  # Apply horizontal flip
                outputs = model(aug_input)
                tta_outputs.append(torch.sigmoid(outputs).cpu().numpy())

            avg_outputs = np.mean(tta_outputs, axis=0)
            preds.extend(avg_outputs)

    preds = np.argsort(-np.array(preds), axis=1)[:, :n_matches]
    return preds
```

Efficient Net-b0

In parallel with the development of the ResNet model, we have also been working on fine-tuning the EfficientNet model for our dataset. To achieve a balance between accuracy and efficiency, we carefully selected key parameters for training. The training process was conducted over 5 epochs with a batch size of 64, which was sufficient to handle the complexity of the model's architecture.The learning rate was adjusted to 1e-3 to allow for steady convergence without causing instability. Given that EfficientNet is a deep architecture requiring extensive training iterations, we incorporated a OneCycleLR scheduler to dynamically adjust the learning rate throughout the training process. This approach helped mitigate overfitting while promoting efficient learning. Additionally, an embedding size of 512 was used.

The model and its embeddings are handled in two separate notebooks. The first notebook is responsible for creating the model and generating the image embeddings, while the second notebook leverages these embeddings to make predictions on new images. Since the competition rules prohibit the submission of a notebook requiring internet access, and internet access is necessary for various packages (specifically the Hugging Face library pulled from the TIMM package), workaround was devised: the embedding notebook outputs a pickle file containing the model weights, which is then used as the input for the prediction notebook.

Input into the testing notebook:



```
COMPETITIONS
▾ Hotel-ID to Combat Human Trafficking 2022 - FGVC9
    ▸ test_images
    ▸ train_images
    ▸ train_masks
      sample_submission.csv
DATASETS
▸ timm-pytorch-image-models
NOTEBOOKS
▾ Hotel-ID starter - similarity - training
      checkpoint-embedding-model-efficientnet_b0-256×256.pt
      embedding-model-efficientnet_b0-256×256_image-embeddings.pkl
      hotel_id_code_mapping.csv
```

This structure was recreated and tested, and while the embedding notebook has an 8-hour runtime, once run and inputted into the prediction notebook, the runtime is 15 minutes. While attempts have been made to apply the TTA used in the ResNet model, TTA relies on a percentage prediction, while the Effecient-Net prediction notebook outputs the hotel IDs based on embeddings. Using the mode instead of the average was attempted, but it did not improve accuracy, so the next step is to implement this change in the embedding notebook to see if the issue was the prediction notebook itself. More time will also be spent on evaluating changing the arguments to improve accuracy.

	Run time	**MAP@5**	**Notes**
Random Assignment	5-10 minutes	0.012	Baseline using random hotel ID predictions.
EfficientNet-b0	8+ hours	0.204	Demonstrated improvement over ResNet; however, embedding generation is slower.
ResNet-34 Baseline	15 min	0.156	Initial performance without TTA enhancements.
ResNet-34 w/ TTA	30 minutes	0.182	Improved score after applying Test-Time Augmentation (TTA).

It is important to note that we do not claim one model is definitively superior to another. Instead, each model contributes unique insights, and their performance is converging towards improving our overall MAP@5 score. This multi-model approach allows us to validate improvements across different methods and ultimately refine our system for more robust hotel ID classification.

Performance:

MAP@5

The MAP@5 score is the main measurable metric that assesses the model's capacity to accurately rank the correct hotel label in its top five predictions for every image. By yielding a definite numerical score, the MAP@5 score makes it simple to compare various models and enables improvement to be tracked over time.

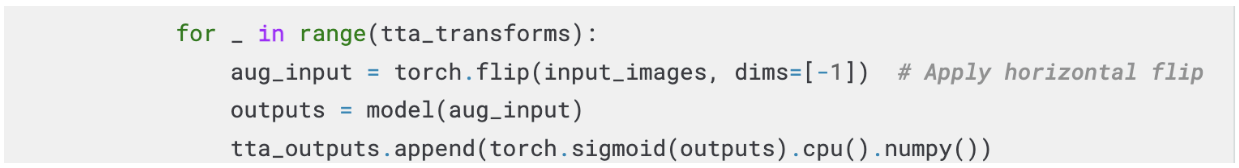
The MAP@5 measure will be evaluated by comparing the model's predictions on a test or validation set of images where we know the ground-truth labels. The MAP@5 score overall will be the average precision over the set of all images. Further, the model's performance will also be compared against a baseline, e.g., a naive or an already-existing model, to see if there is an improvement. The competition leaderboard establishes benchmarks, enabling direct comparison of the MAP@5 score against others and thereby guaranteeing an intensive and systematic assessment process.

Essentially, the model generates a ranked list of the top 5 predicted hotel IDs for each image using mean average precision at 5 (MAP@5). If the correct hotel ID appears in the top 5 rankings, the prediction receives partial credit based on the ranking.

Resnet

As mentioned in the last progress report, the TTA improved the Resnet model this week, the MAP@5 increased from 0.166 to 0.181 with the application of a horizontal flip shown below. This feature remains our only current augmentation as **i**t’s a key structural feature without distorting spatial relationships, compensates for slight differences in camera perspective, introduces minimal risk of artifacts compared to other augmentations, and works well with symmetrical hotel layouts and feature consistency. We will discuss the effects of other augmentations soon.

Current augmentations to TTA model:



```
        for _ in range(tta_transforms):
            aug_input = torch.flip(input_images, dims=[-1])  # Apply horizontal flip
            outputs = model(aug_input)
            tta_outputs.append(torch.sigmoid(outputs).cpu().numpy())
```

Since then, we have further tested other methods on the TTA to improve the accuracy, where additional augmentations hurt the performance, while adding weight to the original image prediction improved prediction stability.

Starting with the augmentations that we tested but did not include, we concluded that over-processing images introduced distortions rather than useful variation, leading to feature misinterpretation. Certain augmentations (e.g., brightness shifts, blur, rotations) altered key details, making it harder for the model to recognize consistent patterns across images. Specifically, Color Jitter (±10% brightness/contrast): Simulates lighting differences, but too much can distort image details. Small Rotations (±5°) & Translations (±5%): Mimics slight camera movements, but larger values misalign features. Gaussian Blur (sigma 0.1–1.0): Reduces reliance on sharp edges, but excessive blur erases fine details. Therefore, instead of improving generalization, these transformations sometimes forced the model to adapt to unrealistic variations, reducing MAP@5 performance.

Additional Augmentations Code:

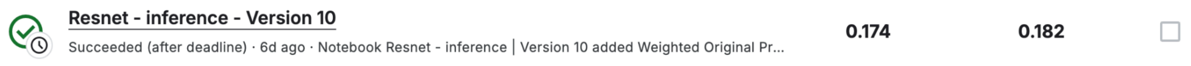
```
            # Define transformations for TTA
            transform_list = [
                lambda x: torch.flip(x, dims=[-1]),  # Horizontal Flip
                lambda x: T.ColorJitter(brightness=0.1, contrast=0.1)(x),  # Mild brightness/c
ontrast
                lambda x: T.RandomAffine(degrees=5, translate=(0.05, 0.05))(x),  # Small rotat
ions/translations
                lambda x: T.GaussianBlur(kernel_size=3, sigma=(0.1, 1.0))(x)  # Mild blur
            ]
```

However, by increasing the original image weight reduced reliance on noisy augmentations is reduced, maintaining focus on meaningful patterns. Additionally, it ensured the model prioritized high-confidence features from the original image while still leveraging slight augmentations. Lastly, it prevented overfitting to augmented variations by stabilizing predictions and reducing inconsistencies across TTA-generated images.

Weighted Original Prediction code:

```
            # Original image inference (weighted more)
            original_output = model(input_images)
            tta_outputs.append(1.5 * torch.sigmoid(original_output).cpu().numpy())  # Weighti
ng original image
```

Thus, our most recent model now yield an accuracy of 0.182.



Resnet - inference - Version 10
Succeeded (after deadline) · 6d ago · Notebook Resnet - inference | Version 10 added Weighted Original Pr...
0.174 0.182

EfficientNet

Originally, issues were getting the EfficientNet model to run due to the speed of the program, the packages needed, and the multiple notebooks. These problems were solved by running the program in Kaggle itself, as Kaggle can run notebooks in the background. The previous issue of Kaggle notebooks ot having updated packages largely solved itself, as after trying Kaggle again after a month, it decided to load properly.

The largest hurdle was understanding the dual nature of the EfficientNet model notebook and the Internet issue. While there are imports on Kaggle that contain timm—the main package that requires Internet access—none of them have the specific huggingface package required, so the double notebook setup is required.

With the nature of these notebooks understood, they were run with a MAP@5 score of 0.204, which is the highest score that has been received so far. The improvements found in the ResNet model will be implemented in the EfficientNet model so that the large embedding notebook does not need to be run for every test. Hopefully, with this setup, the ResNet team can find successful methods, and the EfficientNet team can add those methods as they are found to see if they also improve the EfficientNet model. This way, the best of both models can be combined.

Plan:

In the future, our attention will shift from individual ResNet and EfficientNet models to a hybrid system that utilizes both parts of both notebooks for enhanced feature extraction and similarity matching. EfficientNet will be used as the main backbone for extracting high-level spatial features, while the techniques developed around the ResNet model will be added on top of it. These sets of features will be concatenated into a common embedding space, allowing the model to better distinguish similar hotel rooms.

For the ResNet Model, rather than running only the augmented images through the model, we want to add the augmented images to the dataset with the original images and then rerun the model.

Integration into the existing inference pipeline will ensure Albumentations compatibility, which is already improving image robustness and diversity. Cosine similarity for image matching will continue to be used and measure improvements in MAP@5 scores and inference speeds.

To optimize performance, we will try out different embedding sizes (1024, 2048, 4096) and implement memory management techniques to handle large-scale embeddings. Benchmarking will confirm real-world performance. Currently, the initial framework is set with EfficientNet, Albumentations is fully incorporated. Hybrid model architecture is in progress where EfficientNet would process global features first, and then the techniques from the ResNet model will be used to enhance predictive capability. We are also optimizing embedding generation and similarity matching functions to fuse outputs from both networks.

In the immediate space, however, given the current state of performance, the next immediate steps will focus on optimizing training speed and accuracy. We plan to experiment with reducing model complexity, implementing mixed-precision training to speed up computations, and fine-tuning batch sizes and learning rates to balance efficiency and model convergence. Additionally, we will investigate whether the current augmentations need to be modified or streamlined to avoid unnecessary computation overhead.

A project dashboard is also currently being developed. It will have the ability to take uploaded images, as long as they are of a certain type, and use them as an input into the model. Unfortunately, the dataset does not have hotel names, only IDs, so this particular software won't be directly applicable. However, with slight modifications to the dataset, the same model could be used and the output would be a hotel name that could be pursued by law enforcement agencies.

Remaining Questions:

- What is the best user interface development software or program for our project?
- Will the final product allow users to choose between the two models or use the most accurate model?

Expected Results:

By the end of the two-week deadline, we hope to have answered our previous questions, and have developed a working UI for our model. There will be a functioning model at the best of its capabilities paired with a functional UI, leading to a usable final product that can predict hotels based off unlabeled input images.