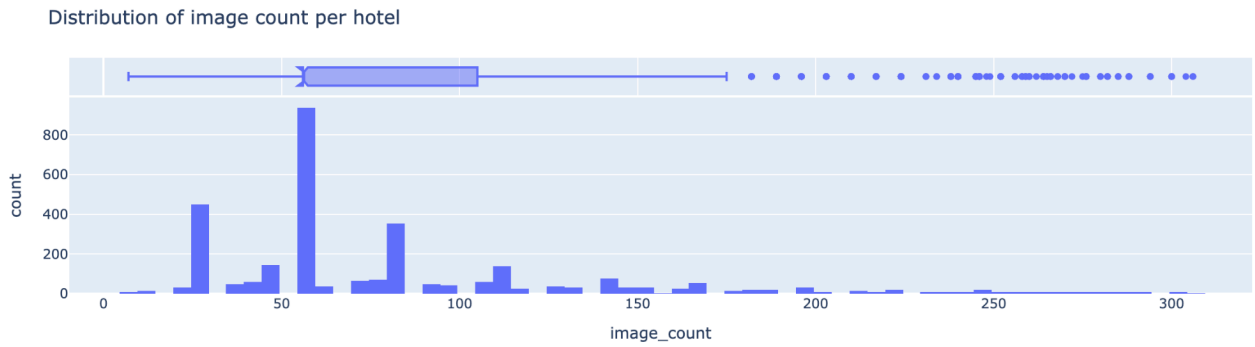
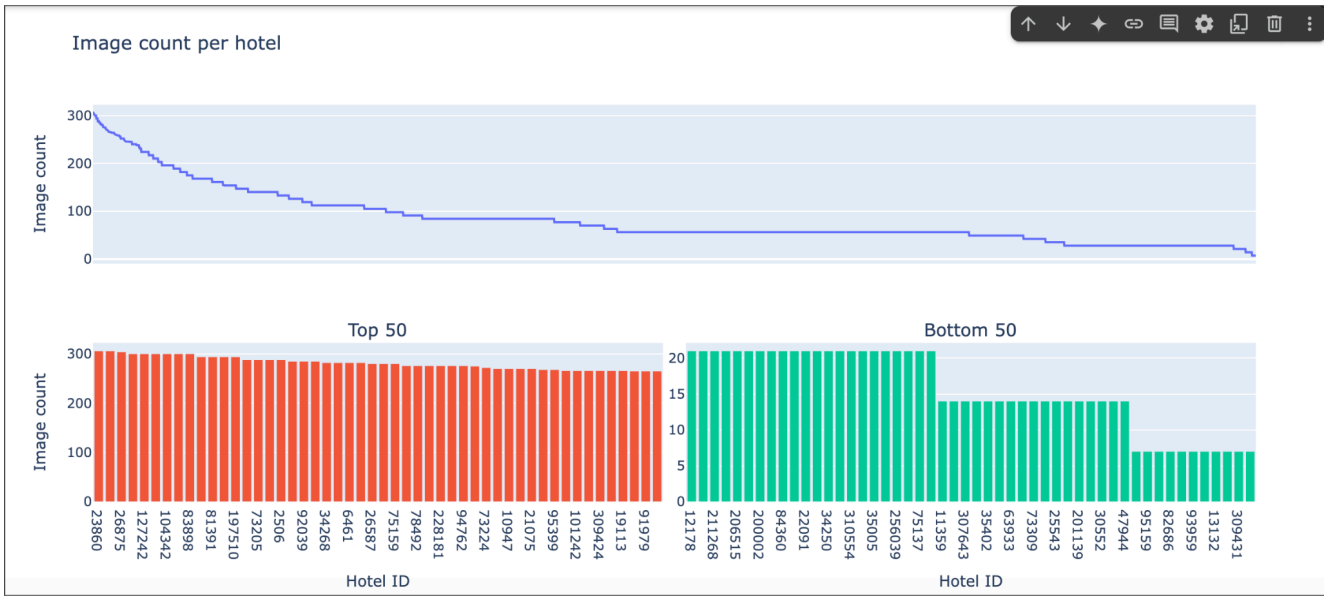
Sneha, Zinnia, Heron, Faisal, Aiah

PD 3.3 Model + Performance Report

1. Which feature engineering techniques were used? Justify your choice and describe the working of the techniques in the context of your project goals.





Feature engineering must take into consideration the limitations and strengths of the original dataset. Above are two visualizations of the distribution of images within each folder (each folder representing a different hotel). As you can see, even though the dataset is quite large (15GB) there are many hotels with very few images to work with. Some folders contained as few as five images. Because of this limitation, we needed to use image augmentation techniques which diversified the dataset in order to prevent overfitting.

A variety of feature engineering techniques were used in our models. 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 ColorJitter 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. RGBShift was most likely redundant with ColorJitter and had no effect, and we dropped many other augmentations such as CutMix for similar reasons.

The Albumentations library does not apply these changes directly to the dataset, but rather it randomly updates the training data each time the model updates its weights. Each augmentation is given a variable "p" for the percent chance that any given image will have that augmentation applied during each iteration. This ensures that the model is being trained on a slightly different dataset each time it updates.

We're using weighted loss functions to make the model more sensitive to underrepresented hotels by penalizing incorrect predictions more heavily in those cases. We also apply test-time augmentation (TTA) to the ResNet model, which involves generating multiple transformed versions of a test image and averaging the predictions. This helps the model make more stable and confident predictions, even when the image conditions are challenging. Techniques like data augmentation and TTA are essential to simulate these real world challenges during both training and inference.

TTA generates several transformed versions of each test image and averages their predictions. This process helps stabilize the model's output, reducing the impact of any single transformation that might otherwise lead to inconsistent predictions. By generating more confident and averaged predictions, TTA directly contributes to a higher MAP@5 score, enhancing the model's reliability and effectiveness in real-world challenging conditions.

2. Which modeling algorithms were used? Justify your choice and describe the working of the algorithms in the context of your project goals.

We used two modeling architectures, ResNet and EfficientNet. 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.

The ResNet architecture is primarily leveraged as our backbone model moving forward, where it uses a pre-trained timm model from the PyTorch library. ResNet, pre-trained on ImageNet, provides robust general pattern recognition capabilities, which were then fine-tuned on the hotel room dataset.

ResNet-34 directly counters the problem of vanishing gradients through residual learning, making it well equipped for deep learning applications. 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. We attempted alternative techniques such as L2 regularization, but found that TTA combined with dropout regularization was the most effective.

In parallel with the development of the ResNet model, we have also been working on fine-tuning the EfficientNet model for our dataset. 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. The model and its embeddings are handled in two separate notebooks. The embedding notebook outputs a pickle file containing the model weights, which is then used as the input for the prediction notebook.

EfficientNet achieves state-of-the-art accuracy with fewer parameters, making it ideal for our image-heavy task. In our custom architecture, we replaced the final classification layer with an embedding head followed by a dropout (p=0.3) and then a classification layer. Before computing cosine similarities for our final inference, we L2 normalize the 512-dim embeddings, ensuring the similarity scores reflect directional agreement rather than magnitude. We also extended our Albumentations pipeline to include stronger geometric and color variations so the model sees a wider array of simulated lighting and perspective conditions.

For the EfficientNet model, the training and implementation 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.

With the data cleaned and standardized, the main feature extraction techniques were all CNN-based models. Specifically, the ResNet-34 and EfficientNet architectures are the backbones of our project models. With these models pre-trained on the ImageNet dataset, the networks were fine-tuned in detecting any patterns and visual cues that are unique to each hotel's interior.

3. Which metrics are used to evaluate the performance of chosen machine learning model(s)? How would you justify the choice of these metrics?

The primary evaluation metric used in our project is Mean Average Precision at 5 (MAP@5). This metric evaluates the ability of the model to rank the correct hotel ID within its top five predictions for each input image. MAP@5 assigns more credit when the correct label appears earlier in the prediction list, allowing us to assess both the accuracy and the ranking confidence of the model. This makes MAP@5 especially suitable for our use case, where multiple hotels can have similar-looking rooms, and correctly narrowing the prediction down to a few candidates is still highly useful for investigators.

Unlike basic accuracy metrics that only reward the top-1 prediction, MAP@5 accounts for partial correctness by measuring how well the correct label is positioned within the top 5 outputs. This is more aligned with real-world scenarios, especially in the context of hotel room image classification, where identifying the exact hotel from visually similar rooms is a complex task.

To further support robust model evaluation, we used 5-fold stratified cross-validation. Each fold was split to maintain a balanced distribution of hotel IDs, helping ensure that our results were not biased by any particular train-test split. This allowed us to test the stability of our models across different subsets of the data and measure how consistent our performance improvements were across multiple runs.

We also evaluated how test-time augmentation (TTA) affected the model's performance using MAP@5. TTA involves generating multiple transformed versions of a test image and averaging the model's predictions across those versions. This method helped improve prediction stability and confidence, particularly for images captured in poor lighting or from unusual angles. Using MAP@5 to evaluate these predictions gave us a meaningful view of how augmentation strategies improved the model's ranking reliability.

Overall, MAP@5 serves as a comprehensive and meaningful metric for this task. It reflects not only whether the model is accurate, but also whether it ranks the correct hotel ID high

enough to be useful in real-world human trafficking investigations, where narrowing possibilities quickly is a critical need.

4. Provide evaluations of your machine learning model(s) using the defined performance metrics.

We evaluated our ResNet-34 model using the Mean Average Precision at 5 (MAP@5) metric, which is the official evaluation measure for the Hotel-ID competition and is well-aligned with our project goals. This metric reflects how well the model ranks the correct hotel ID within its top 5 predictions for each image — a crucial factor in practical investigative use, where narrowing the search to a few hotels significantly aids law enforcement.

The baseline ResNet model achieved a MAP@5 of 0.166. We incrementally improved this performance to 0.185 by incorporating a weighted inference strategy, horizontal flipping during test-time augmentation (TTA), and confidence-weighted averaging of predictions. These enhancements were directly implemented and tested in the predict_tta function within our codebase.

Weighted inference was introduced to give additional importance to the model's prediction on the original, unmodified image. Specifically, we applied a weight of 1.5× to the original image's prediction logits before averaging them with those from the augmented images. This helped reduce the influence of potentially noisy or overly distorted augmentations and ensured that high-confidence features were prioritized. In practice, this provided a more stable and reliable prediction output, especially when the original image was clear and well-lit.

Horizontal flipping was the primary augmentation applied during both training and inference. In test-time augmentation, flipping was chosen for its ability to simulate slight camera angle variations without introducing artifacts that could mislead the model. Hotel rooms often exhibit symmetrical layouts (e.g., bed-center placement, window positions), which made horizontal flips a structurally meaningful transformation. Other augmentations such as heavy distortion, blur, or large rotations were tested but removed due to decreased performance or instability.

Confidence-weighted averaging was used in the TTA pipeline to combine predictions from both the original and augmented images. This approach involves applying a sigmoid function to normalize the output logits before averaging, which ensures that predictions with stronger confidence have more influence on the final result. This mechanism filters

out uncertain predictions and emphasizes consistent visual patterns across multiple views of the same image.

The TTA process used in our inference pipeline applies three randomly selected augmentations per image from a defined list including horizontal flip, brightness/contrast adjustments, affine transformations (small rotation and translation), and Gaussian blur. These were applied via PyTorch and Albumentations within the ResNet prediction function, ensuring that test images were subjected to the same types of variability the model might face in real-world scenarios, such as different lighting conditions, slight angles, or minor occlusions.

After applying these techniques, our best-performing ResNet configuration reached a MAP@5 of 0.185, which is a good increase from the baseline. While this may seem modest, it is significant in a highly granular multi-class classification task with over 50,000 hotel classes. The improvement also reflects increased model robustness across diverse room images, even when training data per hotel is limited.

Additionally, the inference pipeline remained computationally efficient. Based on logs from our Kaggle notebook, the full submission process — including prediction generation and mapping of hotel ID codes — took ~20 minutes, highlighting the practicality of the approach for real-time instances.

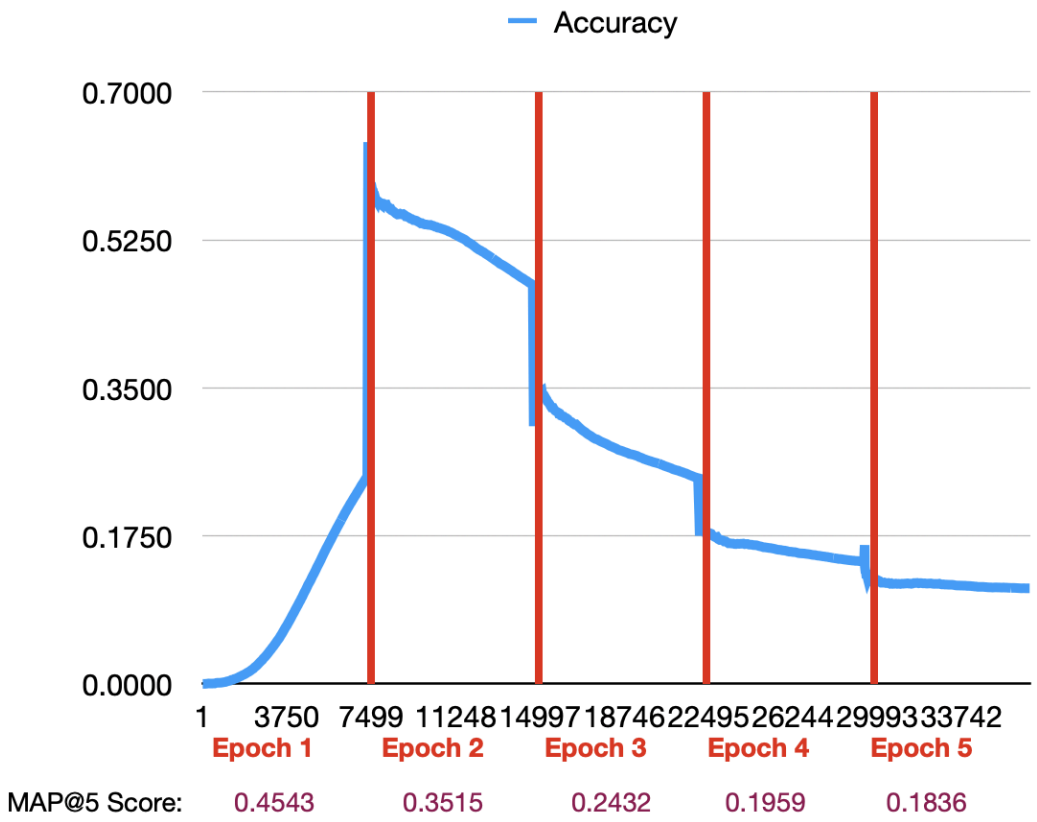
In summary, the ResNet-34 model showed meaningful gains in predictive accuracy after targeted inference adjustments. The combination of TTA, weighted inference, and careful augmentation selection allowed us to maintain inference efficiency while improving the model's ability to consistently rank the correct hotel within the top 5 predictions — a critical capability for supporting human trafficking investigations in the field.

```python
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 = []
            confidence_scores = []

            # Original image inference (weighted 1.5x)
            original_output = model(input_images)
            original_prob = torch.sigmoid(original_output).cpu().numpy()
            tta_outputs.append(1.5 * original_prob)
            confidence_scores.append(1.5)

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

                confidence = np.mean(prob_outputs)
                confidence_scores.append(confidence)

                tta_outputs.append(prob_outputs)

            # Confidence-weighted averaging
            weighted_sum = np.sum([w * p for w, p in zip(confidence_scores, tta_outputs)], axis=0)
            total_weight = np.sum(confidence_scores)
            avg_outputs = weighted_sum / total_weight

            preds.extend(avg_outputs)

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

However, there were many implementations that yielded no improvements. Specifically, incorporation of additional techniques such as Top-2 Confident Flips, Temperature Scaling, Monte Carlo (MC) Dropout, and a Classification Head with Learnable Temperature Scaling did not lead to measurable improvements in model performance. Upon evaluation, these methods appeared to introduce either redundancy or adjustments misaligned with the primary performance objective, which is top-5 ranking accuracy measured by MAP@5. The Top-2 Confident Flips strategy provided limited benefit, as the augmented images were highly similar to the already-included horizontal flip, thereby adding redundancy rather than meaningful diversity. Temperature Scaling and the learnable temperature head are designed to improve confidence calibration; however, since MAP@5 prioritizes

ranking rather than calibrated probabilities, these modifications did not yield performance gains. Similarly, MC Dropout, intended to capture predictive uncertainty, introduced additional stochastic variation without contributing to more robust feature representations. Given that the current Test-Time Augmentation (TTA) pipeline already incorporates confidence-weighted averaging to account for prediction stability and robustness, the added techniques did not offer complementary value. This suggests that further improvements are more likely to come from enhancing feature diversity and representation quality rather than from confidence calibration or uncertainty modeling alone.

For the EfficientNet, enhanced augmentation like larger shift/scale/rotate limits and intensified ColorJitter, a more conservative OneCycleLR schedule (div_factor=20), stronger regularization (dropout = 0.3, weight_decay = 1e-4), and L2-normalization of embeddings helped us achieve a MAP@5 = 0.196 on our validation split. When submitted to Kaggle, this configuration scored MAP@5 = 0.204 on the public leaderboard and 0.196 on the private leaderboard—demonstrating that when using EfficientNet, our validation reliably reflects real-world performance



The above image is based on data scraped from the EfficientNet model's logs. A Python script was created to pull numeric data from the "accuracy"scores in the logs and create a CSV file. This was used to generate a line graph, which was then overlaid with lines separating each epoch of the run.

As you can see, the accuracy of the first epoch was over 60% and the MAP@5 score was 0.45, over twice the score at the end of Epoch 5. Typically when using multiple epochs we would expect the accuracy to increase over time. However, the testing dataset is very small (only one image that we have access to and several hidden images through the Kaggle competition). And the training dataset has as few as 5 images for some of the hotels. Because of this, running the model through multiple epochs may be overfitting the data.

When the model was rerun and tested with 2 epochs instead of 5 however, it was found that the model had a private MAP@5 of 0.076 and a public score of 0.084, which was barely better than our baseline model. This shows that while the accuracy was decreasing, something was happening that still made the model more capable of classification. When 6 epochs were tried, the private MAP@5 score was 0.115, and the public score was 0.127. We also tried running a PCA, which was too computationally expensive, and was unable to run, along with a mode-based version of the TTA, which did not improve the score. More improvements will be tried, such as a MDS (multidimensional scaling) in place of the TTA, or incorporating the TTA architecture in different parts of the code to improve performance.