AI Capstone HW1 report

111550151 徐嘉亨

March 16, 2025

Abstract

This report evaluates the performance of two supervised learning models and one unsupervised learning model in distinguishing AI-generated animal images from real ones. The experiments explore various factors, including parameter selection, the impact of data quantity, data augmentation, and dimensionality reduction.

1 Introduction

AI-generated vs. real image classification aims to distinguish between images created by AI (such as GANs or deep learning models) and those captured by real-world cameras. This problem has gained importance due to the rise of realistic AI-generated content, such as deepfakes, synthetic images, and manipulated media. The motivation behind this classification task is to develop reliable detection methods to combat misinformation, safeguard digital media integrity, and identify AI-generated content in applications like security, media, and social networks.

2 Dataset

Dataset Link

For real animal images, I used a web crawler to collect 300 images from `Unsplash`. For AI-generated animal images, I set up the Stable Diffusion web UI on my local machine following this `instruction`. I then used ChatGPT to generate 300 prompts for synthetic images and a Python script to generate 300 corresponding images (each size is 512x512). For convenience, I randomly sampled 225 images from both real and fake categories for training, while the remaining 150 images were used for testing, ensuring a balanced dataset. These are reference images 1, 2.

3 Experimental Setup

3.1 Supervised Learning

For supervised learning, I use Support Vector Machine and Logistic Regression as my methods. I first perform parameter selection to determine the best model, which serves as my baseline. Then, I conduct experiments to compare with the baseline, focusing on the impact of data quantity, data augmentation, and dimensionality reduction. For each

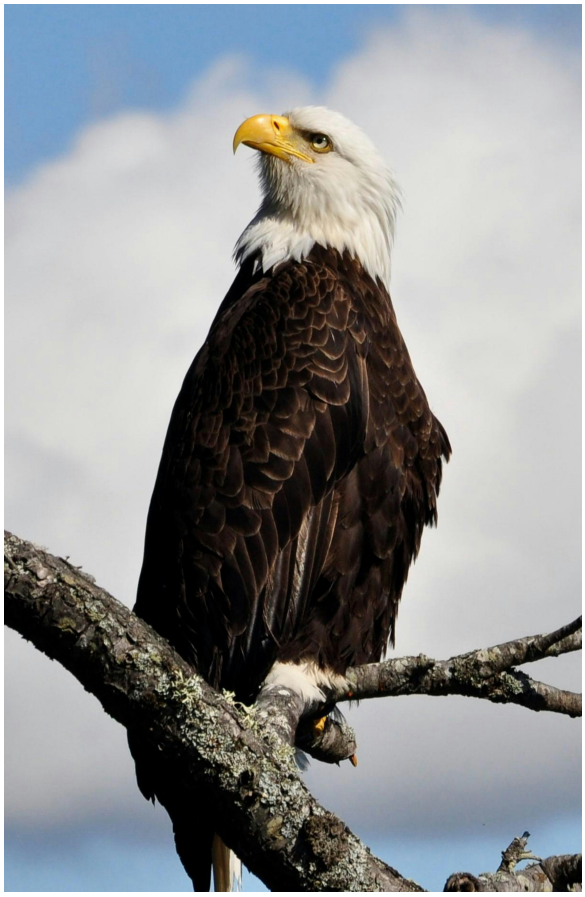

Figure 1: Real Image

Figure 2: AI-Generated Image

experiment, I evaluate performance using accuracy, confusion matrix, precision, recall, F1-score, AUROC, and cross-validation accuracy.

3.2 Unsupervised Learning

For unsupervised learning, I use the K-means algorithm. Similarly, I first perform parameter selection to determine the best model as my baseline. Then, I conduct the same experiments as in supervised learning. For each experiment, I evaluate performance using external metrics, including Adjusted Rand Index (ARI), Normalized Mutual Information (NMI), and Clustering accuracy (By Hungarian Algorithm).

4 Experiments and Results

4.1 SVM

4.1.1 Baseline Model

I first use Histogram of Oriented Gradients (HOG) to extract image features. For parameter selection, I consider image resize dimensions, cells per block (HOG), and the SVM kernel. According to Table 1, I observe that cells per block (3,3) outperform (2,2), so I exclude experiments with (2,2) and a resize dimension of (512,512).

As shown in the results, using larger cells per block and a larger resize size improves performance. Additionally, setting the kernel function to linear achieves the best accuracy, AUROC, and cross-validation accuracy. However, when using the sigmoid kernel, performance drops significantly if the image is resized to (512,512). Based on these findings, I choose cells per block (3,3), resize (512,512), and kernel = linear as my baseline model, and these figures 4, 5, 6 present the confusion matrix, ROC curve, and classification report of the baseline model.

Cells per Block	Resize	Kernel	Accuracy (%)	AUROC	CV Accuracy
(2,2)	(128,128)	linear	67.33	0.75	0.6756 ± 0.0458
(2,2)	(128,128)	poly	68.00	0.76	0.6644 ± 0.0463
(2,2)	(128,128)	sigmoid	52.00	0.51	0.5044 ± 0.0401
(3,3)	(128,128)	linear	72.67	0.78	0.6533 ± 0.0424
(3,3)	(128,128)	poly	70.67	0.80	0.6889 ± 0.0433
(3,3)	(128,128)	sigmoid	52.00	0.58	0.5289 ± 0.0586
(3,3)	(512,512)	linear	**77.33**	**0.86**	**0.7600 ± 0.0565**
(3,3)	(512,512)	poly	70.00	0.84	0.7422 ± 0.0761
(3,3)	(512,512)	sigmoid	31.33	0.74	0.3000 ± 0.0243

Table 1: Experimental Results for HOG+SVM Configurations, Boldface denotes best performance

4.1.2 Impact of Data Quantity

In this part, I examine whether the data quantity significantly affects the performance in this task. I test the baseline model using 100, 200, 300, and 450 training images (complete dataset) for comparison. It is important to note that the real and AI images are still balanced, meaning that for each training set, the number of real and AI images is equal (e.g., 100 training images = 50 real images + 50 AI images), and the testing set consists of 150 images.

As shown in Table 2 and Figure 7, we observe that as the data quantity increases, accuracy, AUROC, and cross-validation accuracy improve. However, with 300 training images, the model achieves the highest accuracy, while AUROC and cross-validation accuracy do not show as significant improvement.

Figures 8, 9, 10 present the confusion matrix, ROC curve, and classification report of the model which is trained on 300 images.

Data quantity	Accuracy (%)	AUROC	CV Accuracy
100	72.67	0.84	0.7300 ± 0.0980
200	76.67	0.85	0.7400 ± 0.0561
300	**78.67**	0.83	0.7467 ± 0.0452
450	77.33	**0.86**	**0.7600 ± 0.0565**

Table 2: Experimental Results for Data Quantity, Boldface denotes best performance

4.1.3 Data Augmentation

In this section, I compare the performance of the model with and without data augmentation. I consider four data augmentation methods: left-right flipping, rotation by 10°, brightness adjustment using **cv2.convertScaleAbs** with $\alpha = 1.2, \beta = 10$, and adding Gaussian noise with $\mu = 0, \sigma = 15$. Each method expands the training set to 900 images, consisting of the original 450 images and 450 augmented images.

As shown in Table 3 and Figure 11, the baseline model achieves the highest accuracy on the test set. Although data augmentation with left-right flipping results in the highest AUROC, the difference compared to the baseline is minimal. Regarding cross-validation accuracy, left-right flipping, rotation, and brightness adjustment outperform the baseline. Notably, brightness adjustment achieves significantly higher cross-validation accuracy than other methods; however, its test set accuracy and AUROC are relatively poor. On the other hand, adding Gaussian noise leads to the worst performance.

Figures 12 to 23 present the confusion matrices, ROC curves, and classification reports for models trained on the augmented datasets.

Augmentation Method	**Accuracy (%)**	**AUROC**	**CV Accuracy**
Baseline	**77.33**	0.86	0.7600 ± 0.0565
Flip	76.00	**0.87**	0.8256 ± 0.0210
Rotation	74.67	0.84	0.8356 ± 0.0147
Brightness	76.00	0.79	**0.9278 ± 0.0145**
Gaussian noise	71.33	0.79	0.7111 ± 0.0355

Table 3: Experimental Results for Data Augmentation, Boldface denotes best performance

4.1.4 Dimensionality Reduction

In this section, I evaluate whether reducing the dimensionality of features significantly affects performance in this task. I apply Principal Component Analysis (PCA) for dimensionality reduction. The original feature dimension after HOG extraction is 311,364. First, I determine the number of components needed to retain at least 95% of the variance after applying PCA, which is 399 (Figure 3). Based on this, I conduct experiments by reducing the number of components to 450 (since the number of components must be between 0 and min(samples, features) = 450), 400, 200, 100, 50, and 10, and compare the results with the baseline (without dimensionality reduction).

```
Number of components to retain 95% variance: 399
```

Figure 3: Number of components

As shown in Table 4 and Figure 24, even when reducing the feature dimensions from 311,364 to 450, the model maintains the same performance, indicating that many features are redundant for this task. Furthermore, as the feature dimension decreases from 400 to 10, performance improves.

Data Dimensions	Accuracy (%)	AUROC	CV Accuracy
311,364 (Baseline)	77.33	**0.86**	0.7600 ± 0.0565
450	77.33	**0.86**	0.7600 ± 0.0565
400	72.67	0.82	0.6644 ± 0.0661
200	76.67	0.80	0.6711 ± 0.0389
100	78.00	0.85	0.7244 ± 0.0606
50	**80.00**	**0.86**	0.7711 ± 0.0480
10	78.00	**0.86**	**0.8356 ± 0.0653**

Table 4: Experimental Results for Dimensionality Reduction, Boldface denotes best performance

4.2 Logistic Regression

4.2.1 Baseline Model

Same setting as SVM, but logistic regression does not have a kernel function. Instead, I examine the effect of different penalty terms. Specifically, I compare the L1 norm and L2 norm penalties.

As shown in Table 5, increasing the cell size per block and using a larger resize dimension improves performance when applying the L1 norm penalty. However, using the L2 norm penalty results in a performance decline. Based on these findings, I select cells per block (3,3), resize (512,512), and the L1 norm penalty as my baseline model. Although it does not achieve the best results across all metrics (accuracy, AUROC, and cross-validation accuracy), it consistently outperforms other configurations overall.

Figures 43, 44, and 45 present the confusion matrix, ROC curve, and classification report for the baseline model.

Cells per Block	Resize	Penalty	Accuracy (%)	AUROC	CV Accuracy
(2,2)	(128,128)	L1	66.00	0.72	0.6356 ± 0.0435
(2,2)	(128,128)	L2	**70.00**	0.75	0.6600 ± 0.0389
(3,3)	(128,128)	L1	68.67	0.76	0.6600 ± 0.0431
(3,3)	(128,128)	L2	67.33	**0.77**	0.6556 ± 0.0404
(3,3)	(512,512)	L1	**70.00**	0.76	0.6822 ± 0.0395
(3,3)	(512,512)	L2	66.00	0.75	**0.6956 ± 0.0818**

Table 5: Experimental Results for HOG+Logistic Regression Configurations, Boldface denotes best performance

4.2.2 Impact of Data Quantity

Using the same setting as SVM, we observe from Table 6 and Figure 46 that as the data quantity increases, accuracy, AUROC, and cross-validation accuracy generally improve. However, with 300 training images, the model experiences a slight drop in performance.

Data quantity	Accuracy (%)	AUROC	CV Accuracy
100	62.67	0.69	0.6500 ± 0.0632
200	66.67	0.74	**0.6850 ± 0.0339**
300	65.33	0.73	0.6767 ± 0.0735
450	**70.00**	**0.76**	0.6822 ± 0.0395

Table 6: Experimental Results for Data Quantity, Boldface denotes best performance

4.2.3 Data Augmentation

Using the same setting as SVM, we observe from Table 7 and Figure 47 that the model achieves performance similar to that of SVM. Regarding cross-validation accuracy, left-right flipping, rotation, and brightness adjustment outperform the baseline. On the other hand, adding Gaussian noise results in the worst overall performance. Figures 48 to 60 present the confusion matrices, ROC curves, and classification reports for models trained on the augmented datasets

Augmentation Method	Accuracy (%)	AUROC	CV Accuracy
Baseline	70.00	0.76	0.6822 ± 0.0395
Flip	68.00	**0.78**	0.7222 ± 0.0274
Rotation	**73.33**	0.75	0.7411 ± 0.0354
Brightness	67.33	0.71	**0.9056 ± 0.0230**
Gaussian noise	64.00	0.66	0.6278 ± 0.0149

Table 7: Experimental Results for Data Augmentation, Boldface denotes best performance

4.2.4 Dimensionality Reduction

Using the same setting as SVM, we observe from Table 8 and Figure 60 that, similar to the findings in SVM, performance improves as the feature dimension decreases from 400 to 10.

Data Dimensions	Accuracy (%)	AUROC	CV Accuracy
311,364 (Baseline)	70.00	0.76	0.6822 ± 0.0395
450	**77.33**	**0.86**	0.7622 ± 0.0413
400	75.33	0.83	0.7133 ± 0.0394
200	76.67	0.83	0.6511 ± 0.0654
100	76.67	**0.86**	0.7089 ± 0.0585
50	**77.33**	0.85	0.7489 ± 0.0464
10	**77.33**	**0.86**	**0.8222 ± 0.0652**

Table 8: Experimental Results for Dimensionality Reduction, Boldface denotes best performance

4.3 K-Means

4.3.1 Baseline Model

For K-Means, I focus on comparing cells per block and resize size, evaluating the results using accuracy, ARI, and NMI. Additionally, I use t-SNE to visualize the clustering results. As shown in Table 9, a larger resize size improves clustering performance. However, smaller cells per block outperform larger ones. Based on these findings, I select cells per block (2,2) and resize (512,512) as my baseline model. Figures 79 to 82 show the clustering results.

4.3.2 Impact of Data Quantity

Since clustering doesn't require a train-test split, I examine the impact of data quantity using 100, 200, 300, and 600 (whole dataset) images. I compare the results using accuracy, ARI, and NMI. As shown in Table 10 and Figure 83, performance generally improves as the data quantity increases. However, with 300 training images, the model reaches its peak performance. Figures 84 to 86 display the clustering results.

4.3.3 Data Augmentation

Using the same settings as for SVM, the total number of images is increased to 1200, and accuracy, ARI, and NMI are compared. As shown in Table 11 and Figure 87, data augmentation does not improve clustering performance and even results in a decline. Figures 88 to 91 present the clustering results.

4.3.4 Dimensionality Reduction

In the same setting as SVM, with the use of (2,2) cells, the total feature dimension is 142,884. The performance was compared using accuracy, ARI, and NMI. As shown in Table 12 and Figure 92, there was a significant performance drop as the feature dimension decreased from 450 to 200. However, the performance returned to baseline levels as the feature dimension further decreased from 100 to 10. Figures 93 to 98 show the clustering results.

5 Discussion

- **Experiment Results and Observations:**
  - The experiments showed that larger cells per block (3,3) and a larger image size (512,512) improved performance in supervised learning, which aligns with the intuition that more detailed feature extraction (larger blocks) and higher resolution images enable the model to capture more discriminative features. However, in the context of unsupervised learning, smaller cells per block (2,2) achieved higher performance, which was unexpected. This may be because smaller cells capture finer, more localized features in the image, which could be more relevant for clustering compared to the broader features captured by larger cells.

– The linear kernel was found to give the best performance in terms of accuracy, AUROC, and cross-validation accuracy, confirming that for this type of task, the simpler linear decision boundary was sufficient. The sigmoid kernel, on the other hand, significantly decreased performance, particularly with higher-resolution images (512,512). This indicates that the problem is likely linearly separable in feature space, and the sigmoid kernel struggled to capture that.

– The logistic regression baseline model selection results align with expectations to some extent. Since the L1 norm encourages sparsity in feature selection, it may help filter out irrelevant or redundant features extracted by HOG, leading to improved performance. However, the performance drop with the L2 norm penalty is somewhat unexpected. L2 regularization typically helps prevent overfitting by penalizing large coefficients, but in this case, it seems to hinder performance. One possible explanation is that L2 forces small but nonzero weights on all features, making it less effective when many features are redundant. Since HOG produces high-dimensional feature vectors, L2 might distribute importance across too many less relevant features, reducing classification effectiveness.

– As the number of training images increased, the overall performance improved, which aligns with the expectation that more data generally enhances model performance. However, the model's accuracy did not improve significantly beyond a certain point and even started to drop, indicating a saturation point where adding more data no longer led to substantial gains in performance.

– In supervised learning, left-right flipping resulted in the highest AUROC, as expected, because it introduces variation that helps the model generalize better. However, brightness adjustment did not improve test accuracy, though it significantly boosted cross-validation accuracy, suggesting overfitting to the training data. Overall, data augmentation seems to improve cross-validation accuracy, except for when adding Gaussian noise. For unsupervised learning, clustering algorithms group similar data points based on feature space similarity. Augmenting images through transformations like rotation or brightness adjustment can disrupt the consistency of feature representations, making it harder for the clustering algorithm to identify meaningful patterns. Gaussian noise introduces randomness that does not reflect the underlying structure. While data augmentation is beneficial for improving classification performance, it may distort the inherent structures for clustering tasks, leading to poorer clustering results, as observed in the comparison of accuracy, ARI, and NMI.

– In supervised learning, reducing the number of features from 311,364 to around 10 improved performance, as expected. PCA effectively eliminated redundant features, making the model more efficient and robust without losing critical information. In unsupervised learning, the key takeaway is that clustering performance is highly sensitive to the number of features. The performance drop when reducing dimensions from 142,884 to 200 is likely due to the loss of essential features. However, the recovery observed with dimensions around 100-200 suggests that dimensionality reduction helped remove noise and redundancy,

allowing the clustering algorithm to focus on the most relevant features. Reducing dimensions too much (e.g., down to 10) again led to poor performance due to the loss of critical information.

- **Important Factors Affect Performance:**
  - Image resolution and feature extraction
  - Kernel selection for SVM
  - Penalty selection for logistic regression
  - Parameters selection for models
  - Dataset Balance
  - Data quantity
  - Data augmentation
  - Dimensionality reduction
- **Future Experiments:**
  - Hyperparameter optimization: I would try a broader range of hyperparameters. For the SVM, such as varying the regularization parameter C. For the logistic regression, such as varying the solver algorithm.
  - Different augmentation strategies: I would experiment with more complex data augmentation techniques, such as zooming, cropping, or adding motion blur, to increase variability in the training set and test the model's robustness.
  - Ensemble methods: I would explore combining the SVM with other classifiers, such as Random Forests or deep learning-based models like Convolutional Neural Networks (CNNs), to see if ensemble methods improve overall performance.
  - Advanced dimensionality reduction techniques: While PCA performed well, other techniques like t-SNE or UMAP could be tested for comparison in terms of how well they reduce dimensionality while retaining important information for classification.
- **Findings and Questions:**
  - Higher resolution images (512,512) combined with more detailed feature extraction (3,3 cells) result in better performance for image classification tasks, emphasizing the importance of high-quality feature representation. However, for clustering, more localized features (2,2 cells) yield better results.
  - Larger datasets and feature dimensionality reduction generally improve model performance, but more data does not always translate to a linear increase in performance, especially after a certain threshold.

- The choice of augmentation method can significantly impact model generalization, but the effectiveness of each method depends on how well it aligns with the characteristics of the dataset.

- How would performance change if I used a deep learning model, such as a CNN, for feature extraction and classification? Would it outperform the traditional ML approach?

- Is there an optimal balance between data augmentation and model complexity that minimizes overfitting while maximizing generalization?

- How well would the model generalize to a different, unseen dataset with similar characteristics? Would performance be consistent across different datasets, or would further tuning be needed?

References

[1] https://unsplash.com/s/photos/animal?license=free

[2] https://github.com/AUTOMATIC1111/stable-diffusion-webui

[3] https://medium.com/@MudSnail/the-importance-of-logistic-regression-in-image-classification-1966d07e7a0c

[4] https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_hog.html

[5] https://blog.csdn.net/sinat_34474705/article/details/80219617

[6] https://blog.csdn.net/liulina603/article/details/8291093

[7] https://scikit-learn.org/stable/modules/svm.html

[8] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

[9] https://scikit-learn.org/stable/modules/generated/sklearn.utils.shuffle.html

[10] https://scikit-learn.org/stable/api/sklearn.metrics.html

[11] https://docs.opencv.org/4.x/index.html

[12] https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation

[13] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[14] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[15] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

Appendix A. Figures, Charts, Tables

A.1 SVM

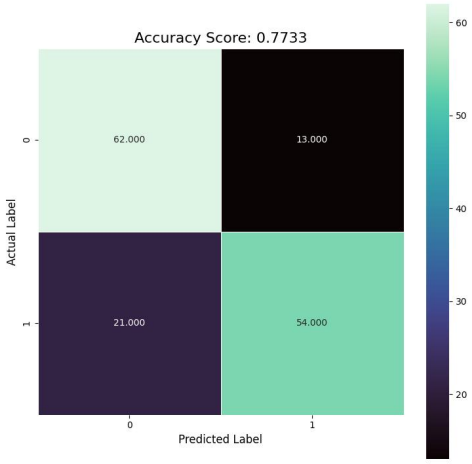

Figure 4: Confusion Matrix of Baseline Model

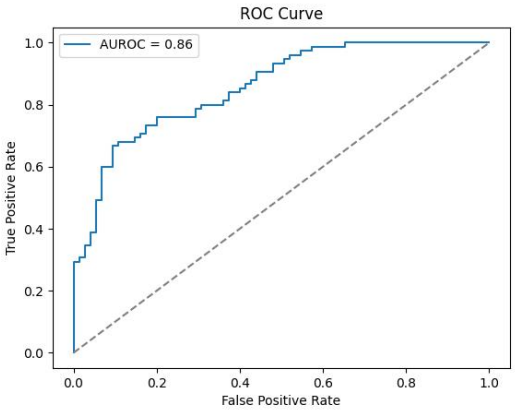

Figure 5: ROC Curve of Baseline Model

Classification Report:				
	precision	recall	f1-score	support
0	0.75	0.83	0.78	75
1	0.81	0.72	0.76	75
accuracy			0.77	150
macro avg	0.78	0.77	0.77	150
weighted avg	0.78	0.77	0.77	150

Figure 6: Classification report of Baseline Model

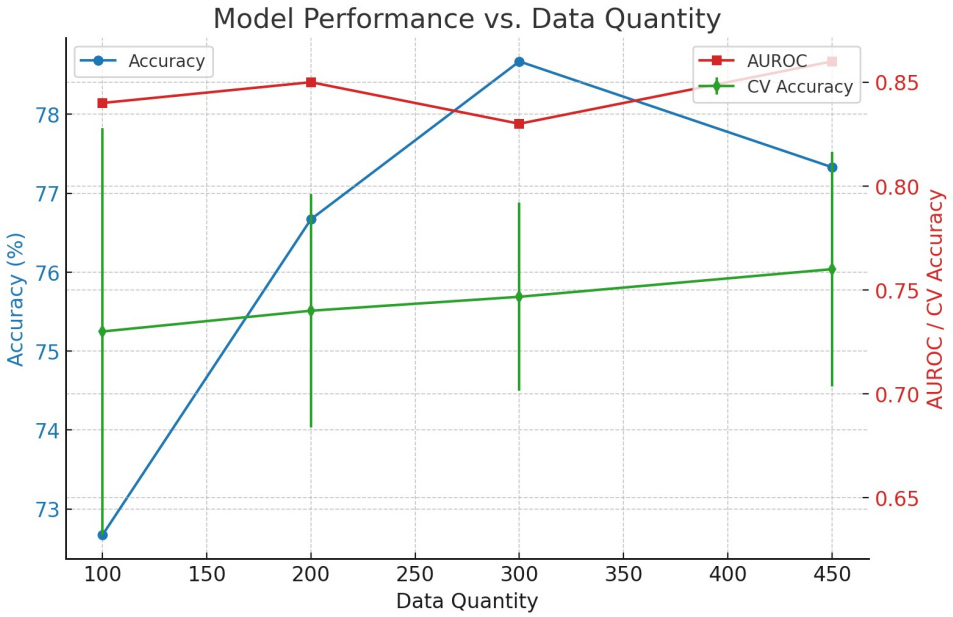

Figure 7: Performance over Data Quantity

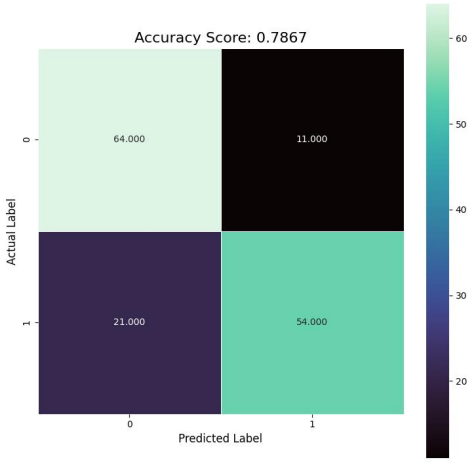

Figure 8: Confusion Matrix of Data 300

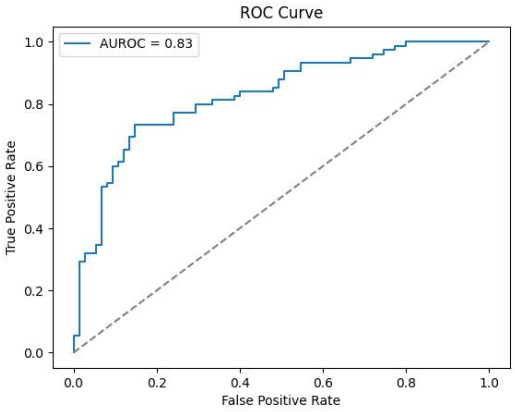

Figure 9: ROC Curve of Data 300

Classification Report:

	precision	recall	f1-score	support
0	0.75	0.85	0.80	75
1	0.83	0.72	0.77	75
accuracy			0.79	150
macro avg	0.79	0.79	0.79	150
weighted avg	0.79	0.79	0.79	150

Figure 10: Classification report of Data 300

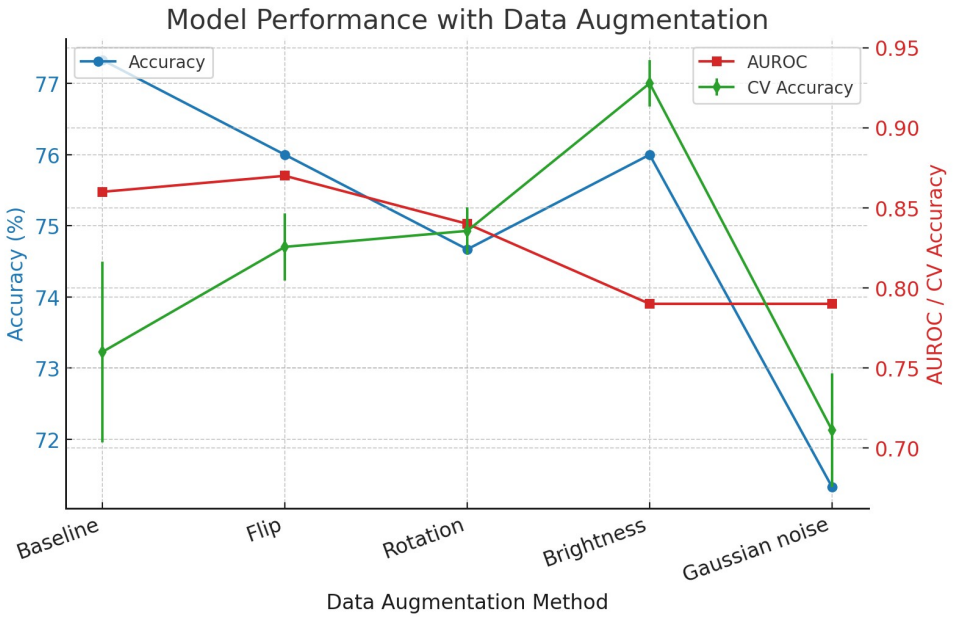

Figure 11: Performance over Data Augmentation

Figure 12: Confusion Matrix of Left-Right Flip

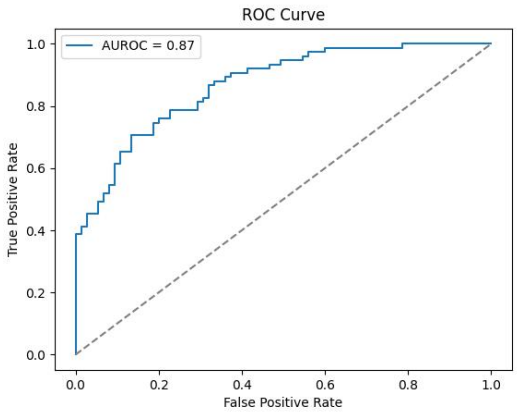

Figure 13: ROC Curve of Left-Right Flip

Classification Report:				
	precision	recall	f1-score	support
0	0.73	0.81	0.77	75
1	0.79	0.71	0.75	75
accuracy			0.76	150
macro avg	0.76	0.76	0.76	150
weighted avg	0.76	0.76	0.76	150

Figure 14: Classification report of Left-Right Flip

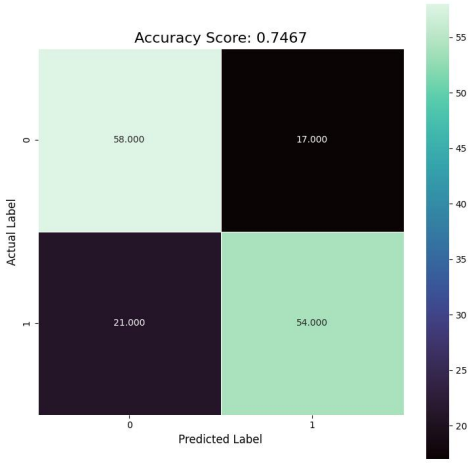

Figure 15: Confusion Matrix of Rotation

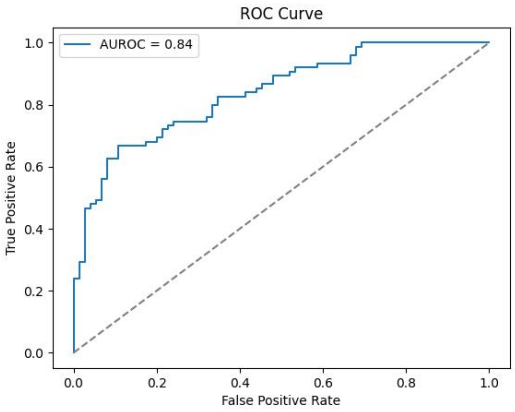

Figure 16: ROC Curve of Rotation

Classification Report:				
	precision	recall	f1-score	support
0	0.73	0.77	0.75	75
1	0.76	0.72	0.74	75
accuracy			0.75	150
macro avg	0.75	0.75	0.75	150
weighted avg	0.75	0.75	0.75	150

Figure 17: Classification report of Rotation

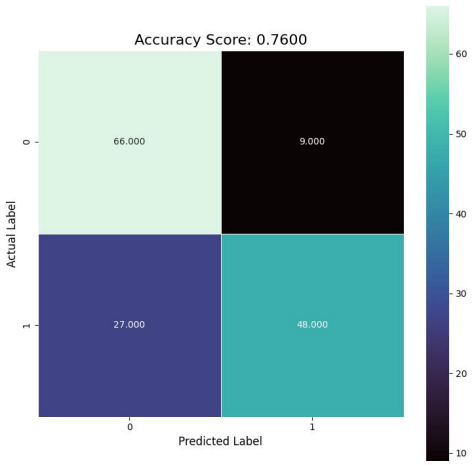

Figure 18: Confusion Matrix of Brightness

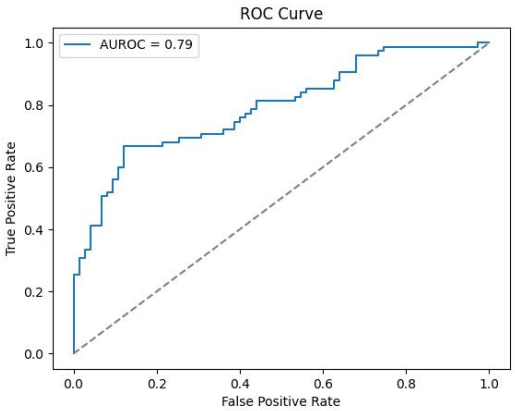

Figure 19: ROC Curve of Brightness

Classification Report:

	precision	recall	f1-score	support
0	0.71	0.88	0.79	75
1	0.84	0.64	0.73	75
accuracy			0.76	150
macro avg	0.78	0.76	0.76	150
weighted avg	0.78	0.76	0.76	150

Figure 20: Classification report of Brightness

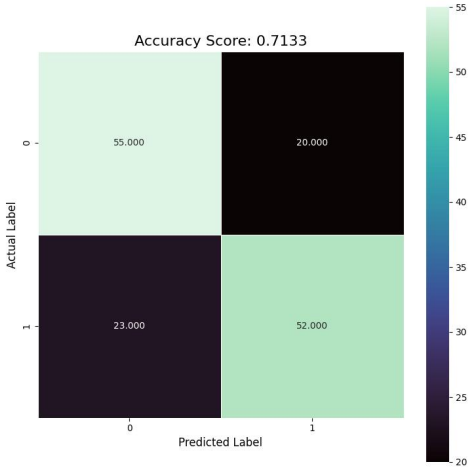

Figure 21: Confusion Matrix of Gaussian Noise

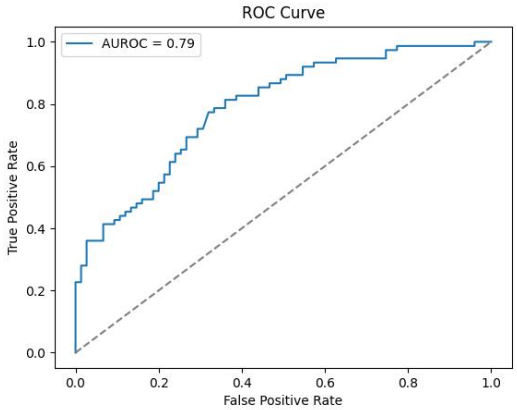

Figure 22: ROC Curve of Gaussian Noise

Classification Report:

	precision	recall	f1-score	support
0	0.71	0.73	0.72	75
1	0.72	0.69	0.71	75
accuracy			0.71	150
macro avg	0.71	0.71	0.71	150
weighted avg	0.71	0.71	0.71	150

Figure 23: Classification report of Gaussian Noise

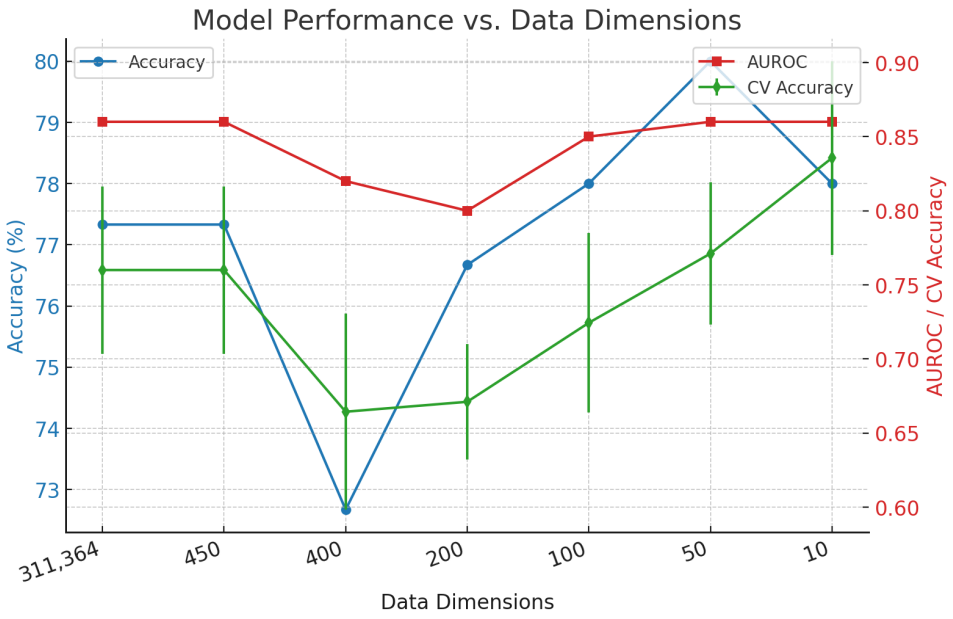

Figure 24: Performance over Dimensionality Reduction

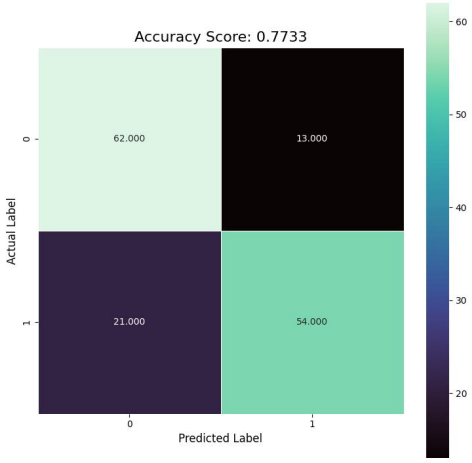

Figure 25: Confusion Matrix of Dimensions 450

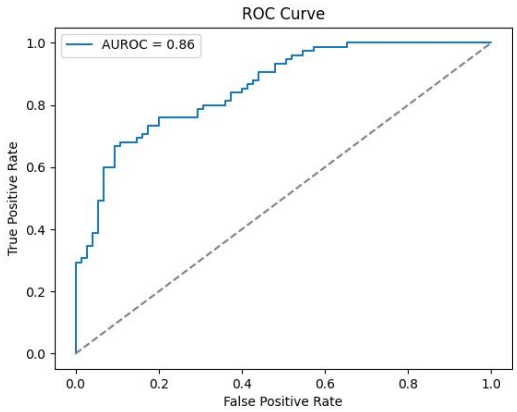

Figure 26: ROC Curve of Dimensions 450

Classification Report:				
	precision	recall	f1-score	support
0	0.75	0.83	0.78	75
1	0.81	0.72	0.76	75
accuracy			0.77	150
macro avg	0.78	0.77	0.77	150
weighted avg	0.78	0.77	0.77	150

Figure 27: Classification report of Dimensions 450

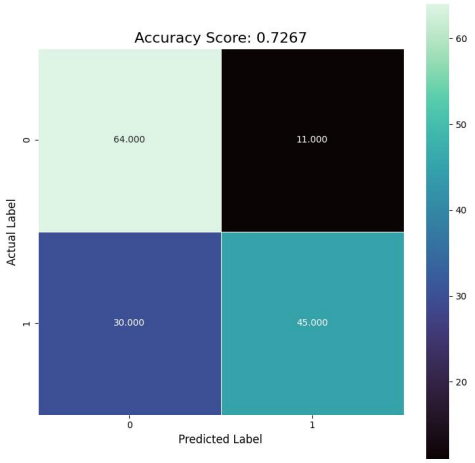

Figure 28: Confusion Matrix of Dimensions 400

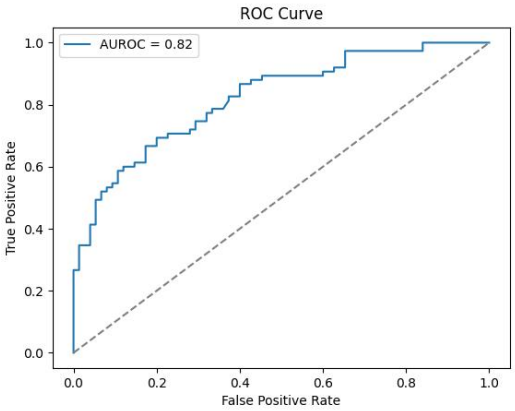

Figure 29: ROC Curve of Dimensions 400

Classification Report:				
	precision	recall	f1-score	support
0	0.68	0.85	0.76	75
1	0.80	0.60	0.69	75
accuracy			0.73	150
macro avg	0.74	0.73	0.72	150
weighted avg	0.74	0.73	0.72	150

Figure 30: Classification report of Dimensions 400

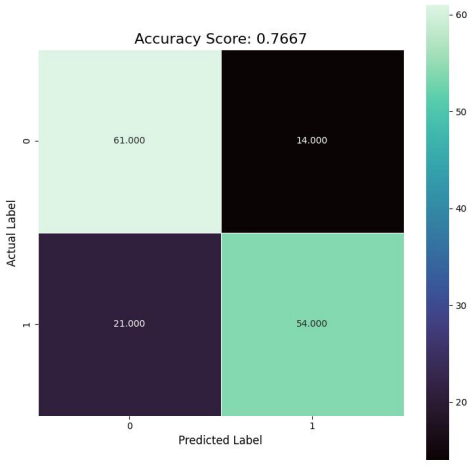

Figure 31: Confusion Matrix of Dimensions 200

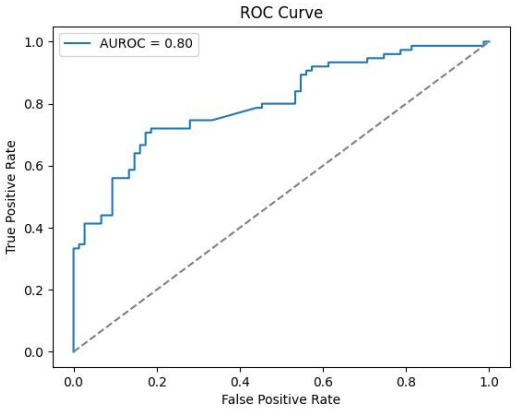

Figure 32: ROC Curve of Dimensions 200

Classification Report:

	precision	recall	f1-score	support
0	0.74	0.81	0.78	75
1	0.79	0.72	0.76	75
accuracy			0.77	150
macro avg	0.77	0.77	0.77	150
weighted avg	0.77	0.77	0.77	150

Figure 33: Classification report of Dimensions 200

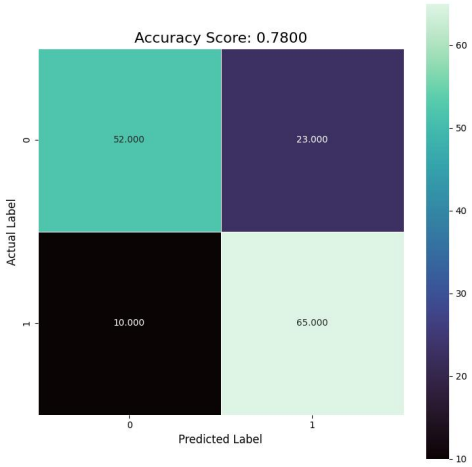

Figure 34: Confusion Matrix of Dimensions 100

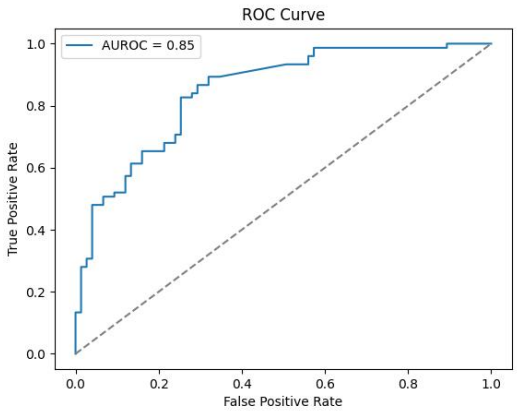

Figure 35: ROC Curve of Dimensions 100

Classification Report:

	precision	recall	f1-score	support
0	0.84	0.69	0.76	75
1	0.74	0.87	0.80	75
accuracy			0.78	150
macro avg	0.79	0.78	0.78	150
weighted avg	0.79	0.78	0.78	150

Figure 36: Classification report of Dimensions 100

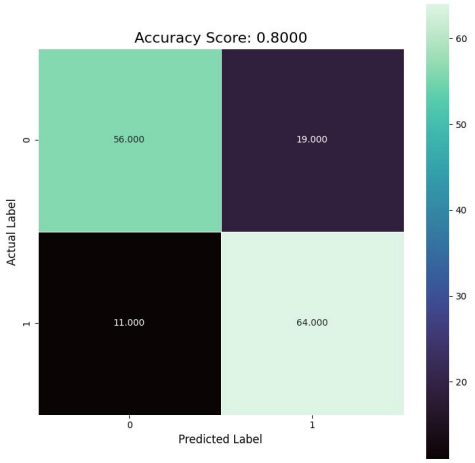

Figure 37: Confusion Matrix of Dimensions 50

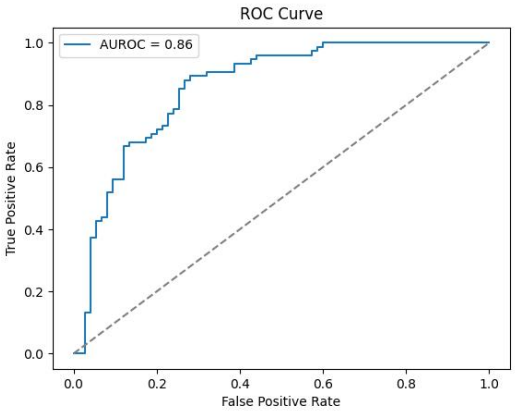

Figure 38: ROC Curve of Dimensions 50

Classification Report:				
	precision	recall	f1-score	support
0	0.84	0.75	0.79	75
1	0.77	0.85	0.81	75
accuracy			0.80	150
macro avg	0.80	0.80	0.80	150
weighted avg	0.80	0.80	0.80	150

Figure 39: Classification report of Dimensions 50

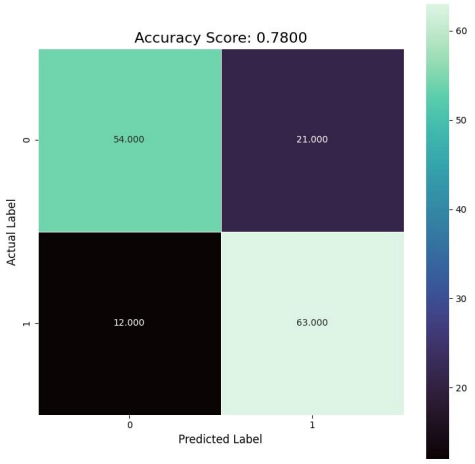

Figure 40: Confusion Matrix of Dimensions 10

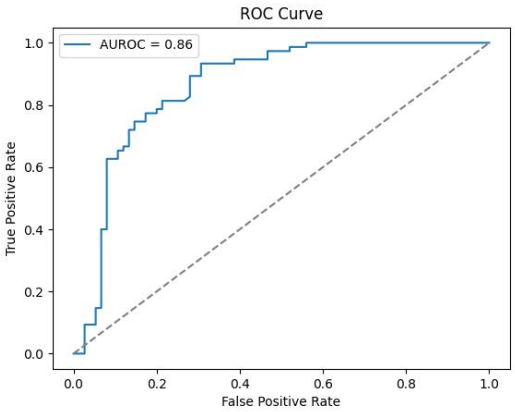

Figure 41: ROC Curve of Dimensions 10

```
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.72      0.77        75
           1       0.75      0.84      0.79        75

    accuracy                           0.78       150
   macro avg       0.78      0.78      0.78       150
weighted avg       0.78      0.78      0.78       150
```

Figure 42: Classification report of Dimensions 10

A.2 Logistic Regression

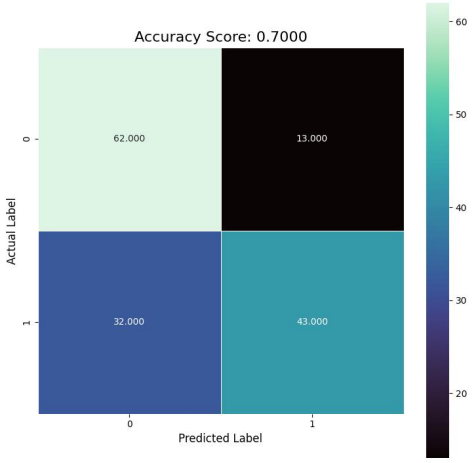

Figure 43: Confusion Matrix of Baseline Model

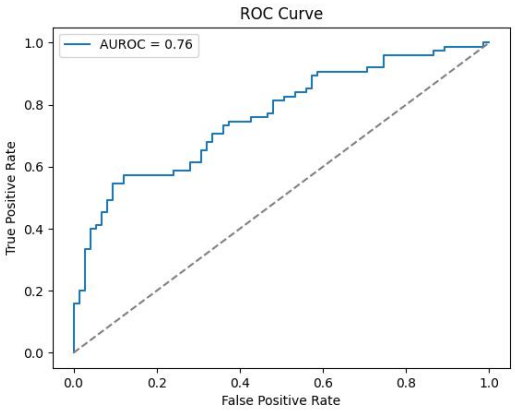

Figure 44: ROC Curve of Baseline Model

Classification Report:				
	precision	recall	f1-score	support
0	0.66	0.83	0.73	75
1	0.77	0.57	0.66	75
accuracy			0.70	150
macro avg	0.71	0.70	0.70	150
weighted avg	0.71	0.70	0.70	150

Figure 45: Classification report of Baseline Model

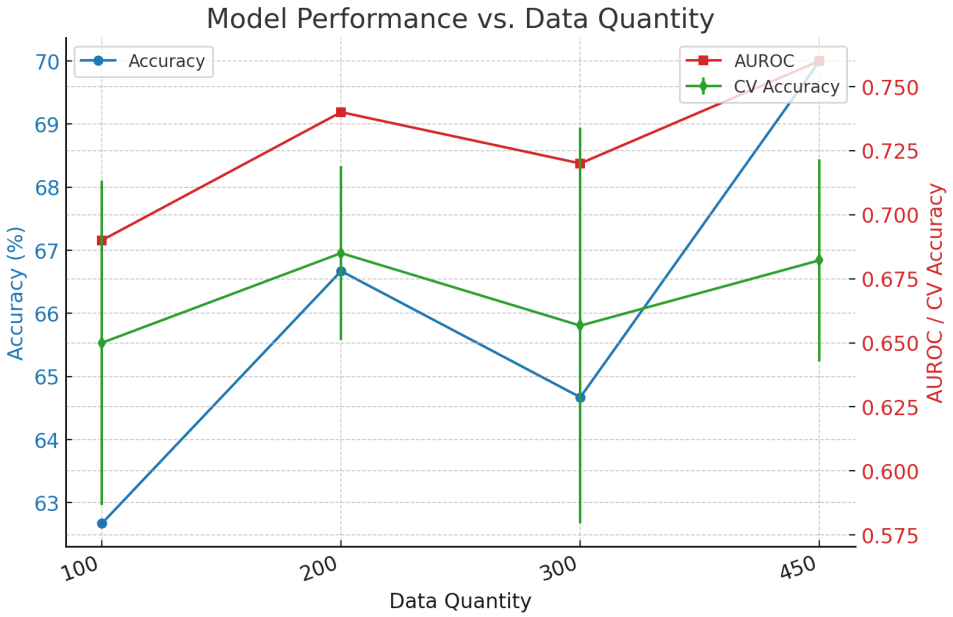

Figure 46: Performance over Data Quantity

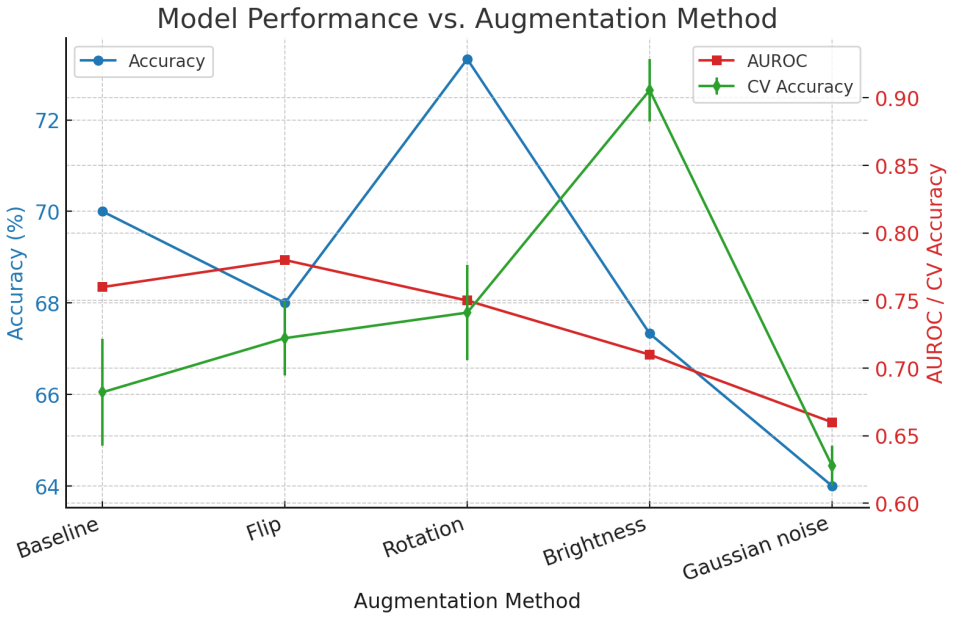

Figure 47: Performance over Data Augmentation

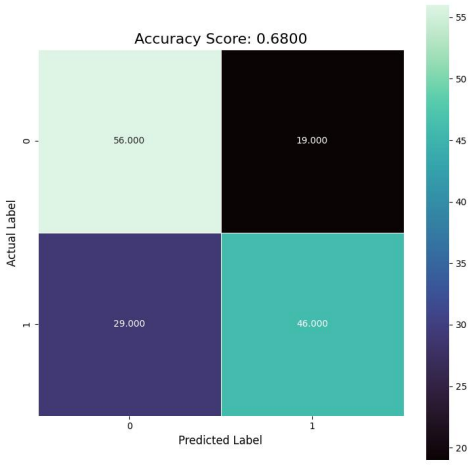

Figure 48: Confusion Matrix of Left-Right Flip

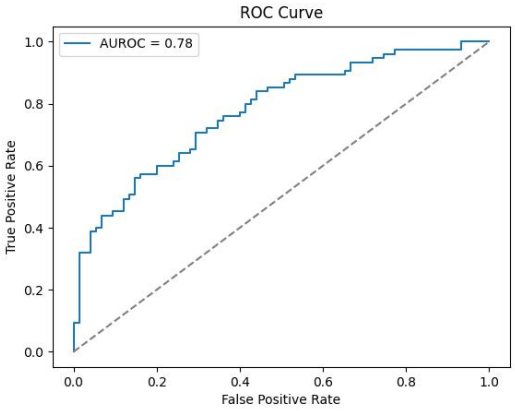

Figure 49: ROC Curve of Left-Right Flip

Classification Report:	precision	recall	f1-score	support
0	0.66	0.75	0.70	75
1	0.71	0.61	0.66	75
accuracy			0.68	150
macro avg	0.68	0.68	0.68	150
weighted avg	0.68	0.68	0.68	150

Figure 50: Classification report of Left-Right Flip

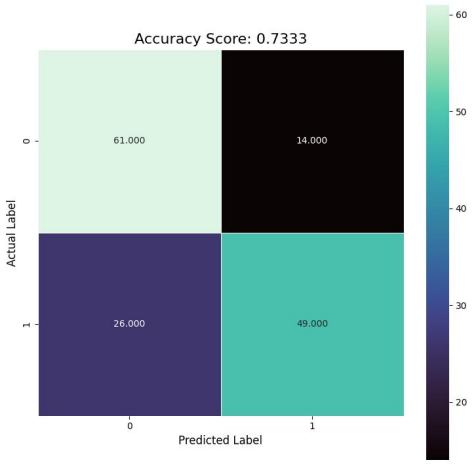

Figure 51: Confusion Matrix of Rotation

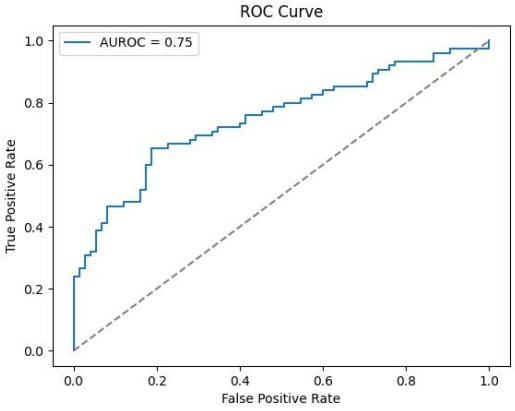

Figure 52: ROC Curve of Rotation

Classification Report:

	precision	recall	f1-score	support
0	0.70	0.81	0.75	75
1	0.78	0.65	0.71	75
accuracy			0.73	150
macro avg	0.74	0.73	0.73	150
weighted avg	0.74	0.73	0.73	150

Figure 53: Classification report of Rotation

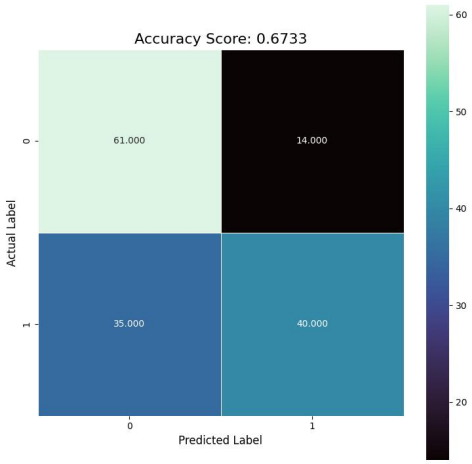

Figure 54: Confusion Matrix of Brightness

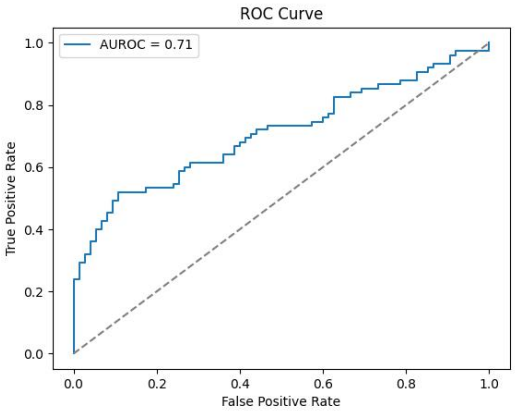

Figure 55: ROC Curve of Brightness

Classification Report:

	precision	recall	f1-score	support
0	0.64	0.81	0.71	75
1	0.74	0.53	0.62	75
accuracy			0.67	150
macro avg	0.69	0.67	0.67	150
weighted avg	0.69	0.67	0.67	150

Figure 56: Classification report of Brightness

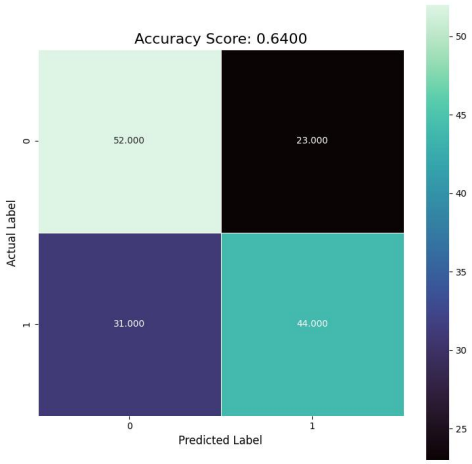

Figure 57: Confusion Matrix of Gaussian Noise

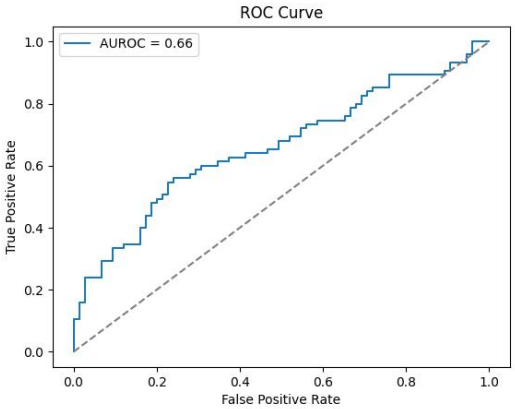

Figure 58: ROC Curve of Gaussian Noise

Classification Report:	precision	recall	f1-score	support
0	0.63	0.69	0.66	75
1	0.66	0.59	0.62	75
accuracy			0.64	150
macro avg	0.64	0.64	0.64	150
weighted avg	0.64	0.64	0.64	150

Figure 59: Classification report of Gaussian Noise

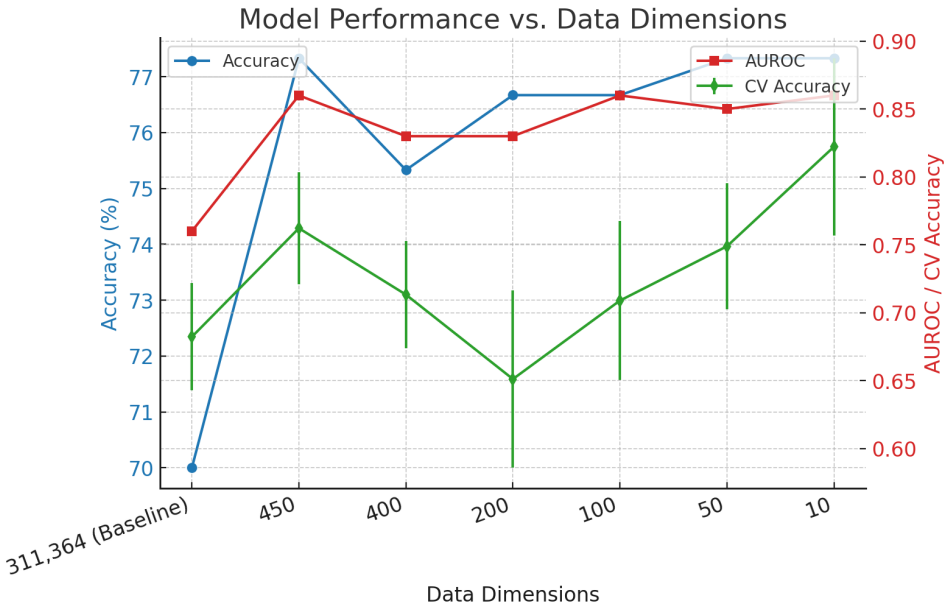

Figure 60: Performance over Dimensionality Reduction

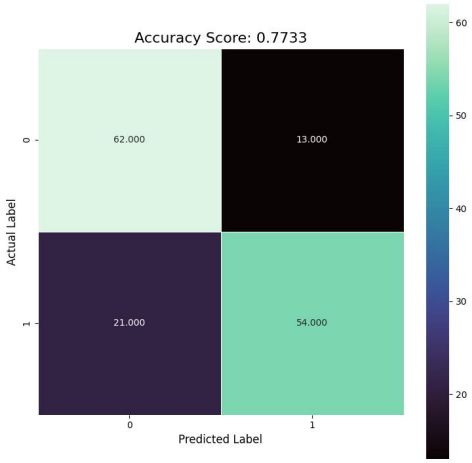

Figure 61: Confusion Matrix of Dimensions 450

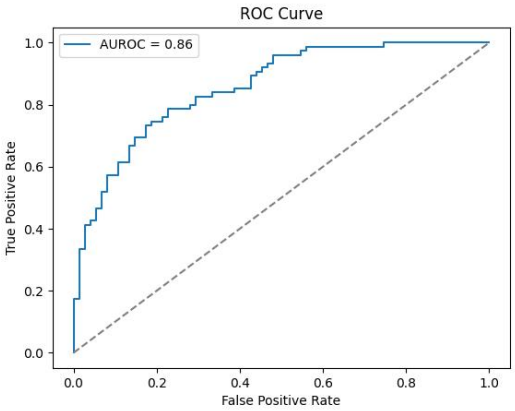

Figure 62: ROC Curve of Dimensions 450

Classification Report:				
	precision	recall	f1-score	support
0	0.75	0.83	0.78	75
1	0.81	0.72	0.76	75
accuracy			0.77	150
macro avg	0.78	0.77	0.77	150
weighted avg	0.78	0.77	0.77	150

Figure 63: Classification report of Dimensions 450

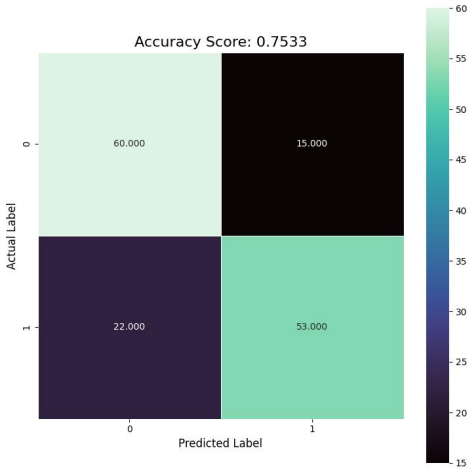

Figure 64: Confusion Matrix of Dimensions 400

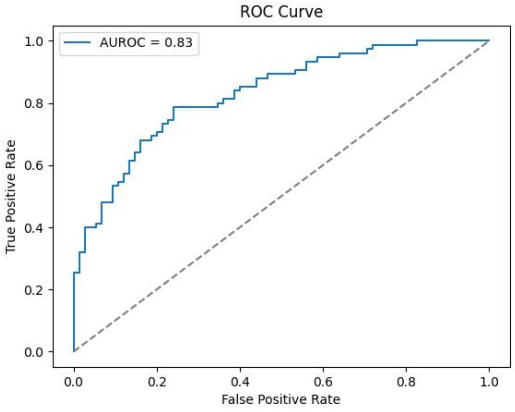

Figure 65: ROC Curve of Dimensions 400

Classification Report:

	precision	recall	f1-score	support
0	0.73	0.80	0.76	75
1	0.78	0.71	0.74	75
accuracy			0.75	150
macro avg	0.76	0.75	0.75	150
weighted avg	0.76	0.75	0.75	150

Figure 66: Classification report of Dimensions 400

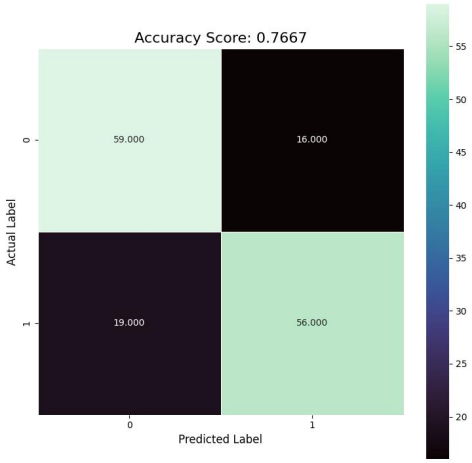

Figure 67: Confusion Matrix of Dimensions 200

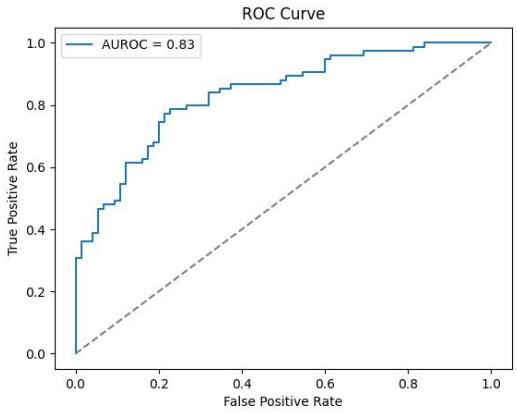

Figure 68: ROC Curve of Dimensions 200

Classification Report:

	precision	recall	f1-score	support
0	0.76	0.79	0.77	75
1	0.78	0.75	0.76	75
accuracy			0.77	150
macro avg	0.77	0.77	0.77	150
weighted avg	0.77	0.77	0.77	150

Figure 69: Classification report of Dimensions 200

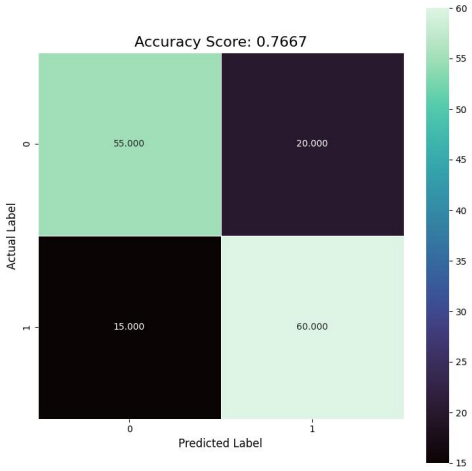

Figure 70: Confusion Matrix of Dimensions 100

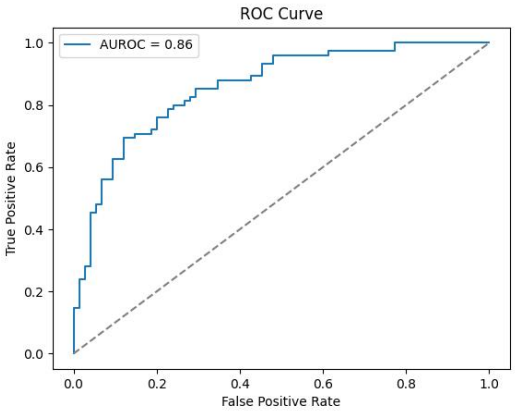

Figure 71: ROC Curve of Dimensions 100

Classification Report:				
	precision	recall	f1-score	support
0	0.79	0.73	0.76	75
1	0.75	0.80	0.77	75
accuracy			0.77	150
macro avg	0.77	0.77	0.77	150
weighted avg	0.77	0.77	0.77	150

Figure 72: Classification report of Dimensions 100

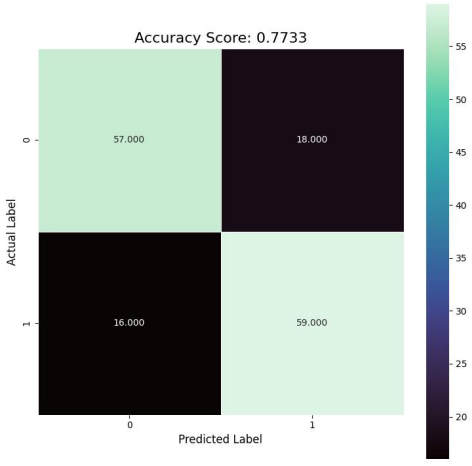

Figure 73: Confusion Matrix of Dimensions 50

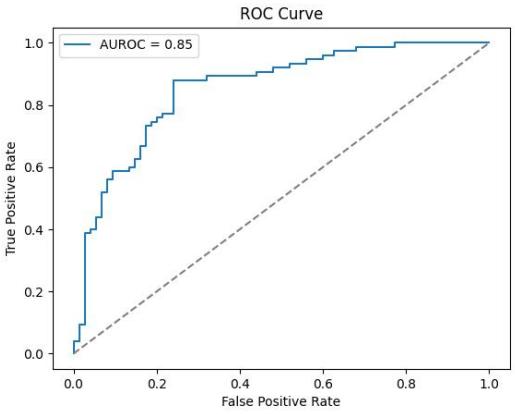

Figure 74: ROC Curve of Dimensions 50

Classification Report:				
	precision	recall	f1-score	support
0	0.78	0.76	0.77	75
1	0.77	0.79	0.78	75
accuracy			0.77	150
macro avg	0.77	0.77	0.77	150
weighted avg	0.77	0.77	0.77	150

Figure 75: Classification report of Dimensions 50

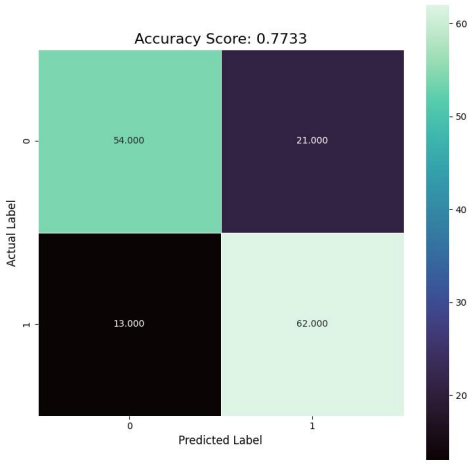

Figure 76: Confusion Matrix of Dimensions 10

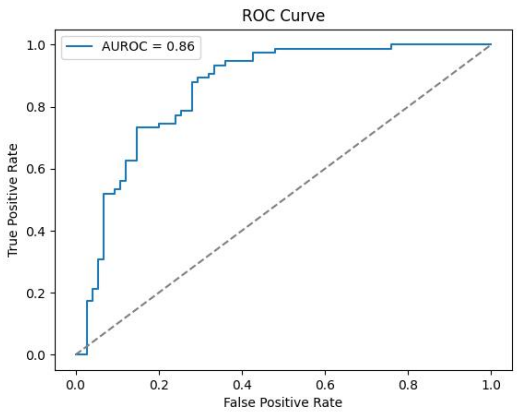

Figure 77: ROC Curve of Dimensions 10

Classification Report:				
	precision	recall	f1-score	support
0	0.81	0.72	0.76	75
1	0.75	0.83	0.78	75
accuracy			0.77	150
macro avg	0.78	0.77	0.77	150
weighted avg	0.78	0.77	0.77	150

Figure 78: Classification report of Dimensions 10

A.3 K-Means

Cells per Block	Resize	Accuracy (%)	ARI	NMI
(2,2)	(128,128)	60.33	0.0416	0.0489
(3,3)	(128,128)	59.83	0.0375	0.0489
(2,2)	(512,512)	**64.00**	**0.0770**	**0.0674**
(3,3)	(512,512)	63.00	0.0661	0.0525

Table 9: Experimental Results for HOG+K-Means Configurations, Boldface denotes best performance

Data quantity	Accuracy (%)	ARI	NMI
100	55.00	0.0035	0.0210
200	62.00	0.0534	0.0481
300	**65.67**	**0.0952**	**0.0733**
600	64.00	0.0770	0.0674

Table 10: Experimental Results for Data Quantity, Boldface denotes best performance

Augmentation Method	Accuracy (%)	ARI	NMI
Baseline	**64.00**	**0.0770**	**0.0674**
Flip	63.00	0.0669	0.0611
Rotation	50.75	-0.0006	0.0002
Brightness	62.42	0.0610	0.0567
Gaussian noise	50.08	0.0000	0.0017

Table 11: Experimental Results for Data Augmentation, Boldface denotes best performance

Data Dimensions	Accuracy (%)	ARI	NMI
142,884 (Baseline)	**64.00**	**0.0770**	**0.0674**
450	50.17	0.0000	0.0033
400	50.17	0.0000	0.0033
200	50.17	0.0000	0.0033
100	63.00	0.0662	0.0601
50	63.33	0.0697	0.0624
10	63.00	0.0662	0.0583

Table 12: Experimental Results for Dimensionality Reduction, Boldface denotes best performance

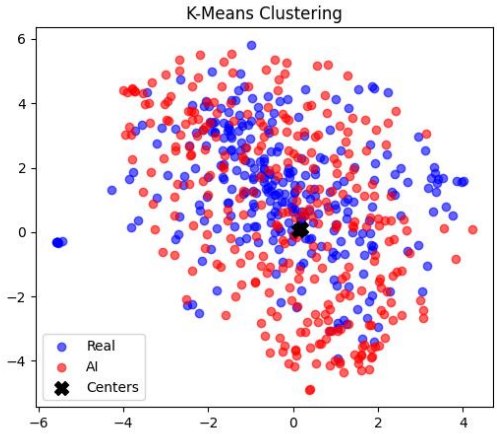

Figure 79: Clustering Result of (2, 2), (128, 128)

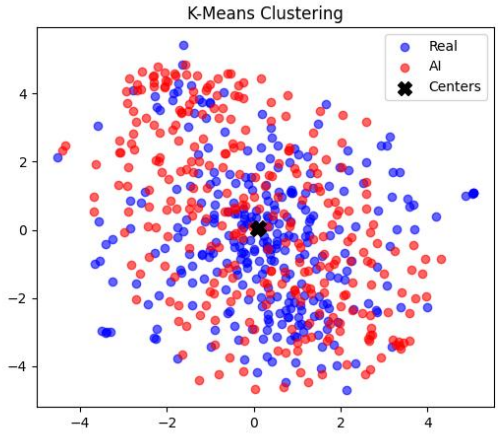

Figure 80: Clustering Result of (3, 3), (128, 128)

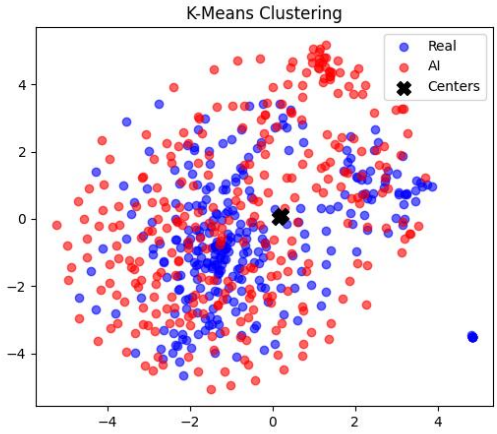

Figure 81: Clustering Result of (2, 2), (512, 512)

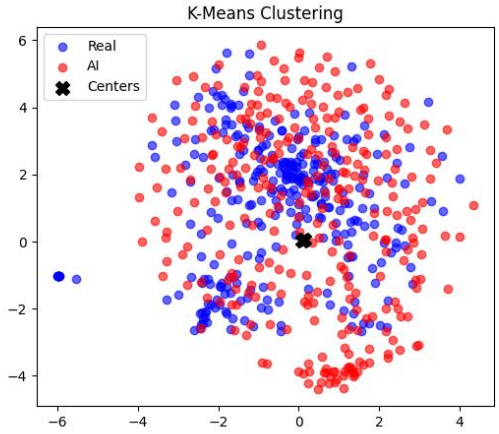

Figure 82: Clustering Result of (3, 3), (512, 512)

Figure 83: Performance over Data Quantity

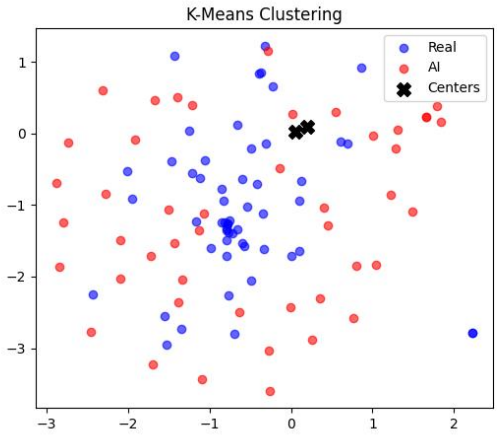

Figure 84: Clustering Result of 100 data

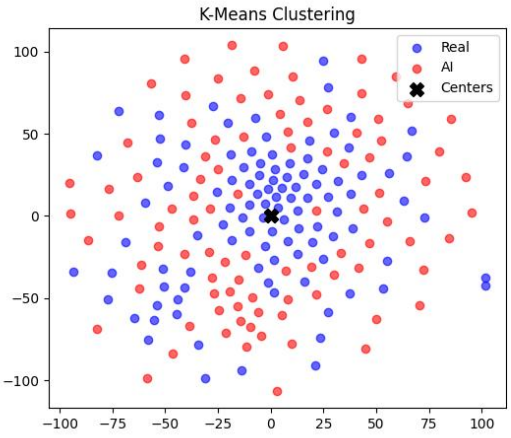

Figure 85: Clustering Result of 200 data

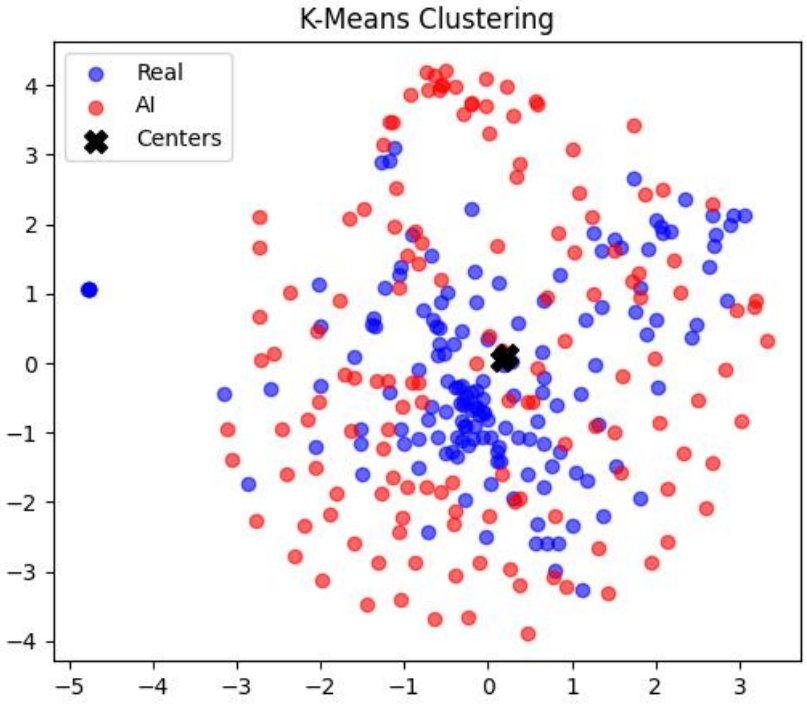

Figure 86: Clustering Result of 300 data

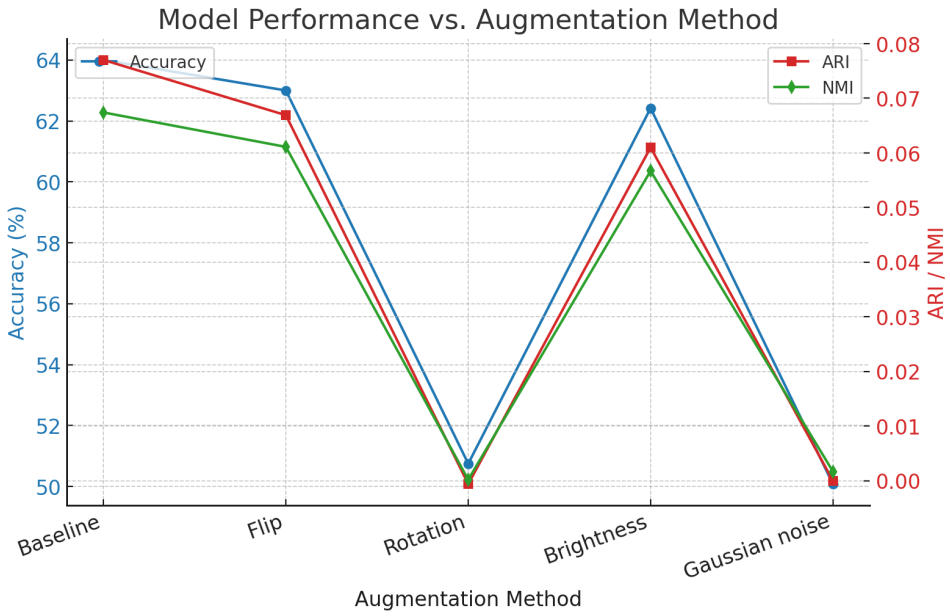

Figure 87: Performance over Data Augmentation

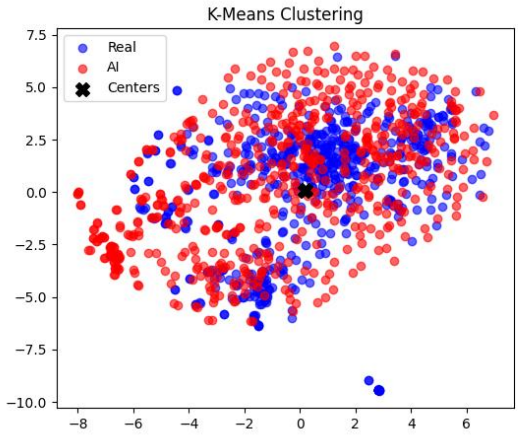

Figure 88: Clustering Result of Flip

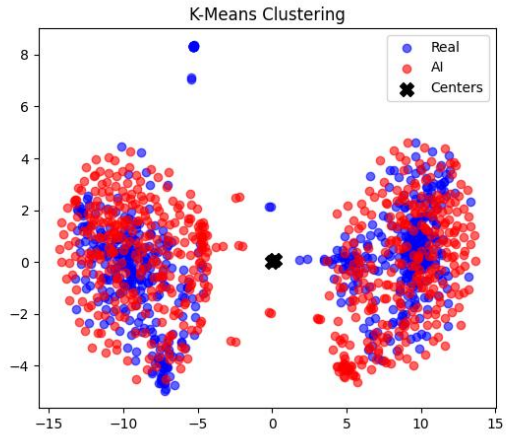

Figure 89: Clustering Result of Rotation

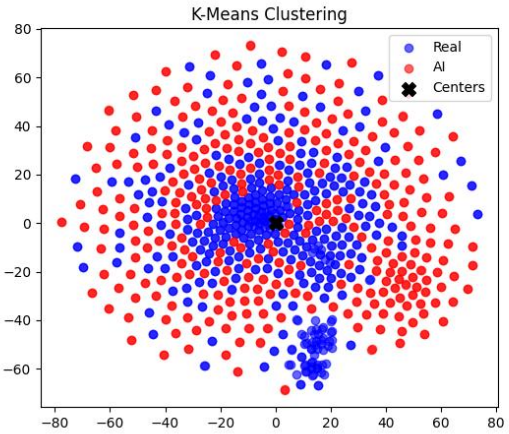

Figure 90: Clustering Result of Brightness

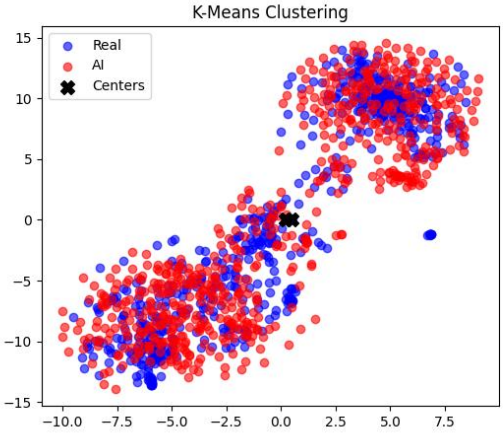

Figure 91: Clustering Result of Gaussian Noise

Figure 92: Performance over Dimensionality Reduction

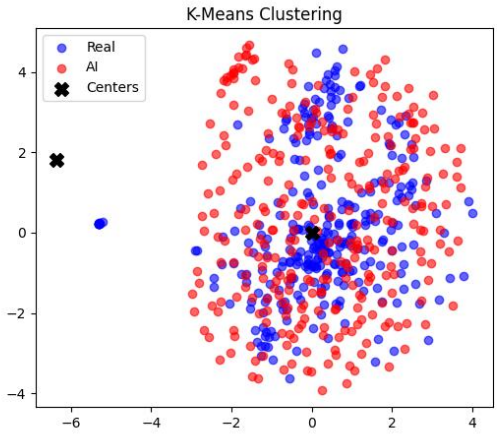

Figure 93: Clustering Result of Dimensions 450

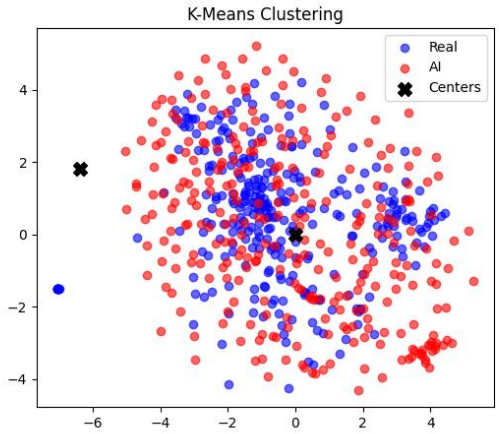

Figure 94: Clustering Result of Dimensions 400

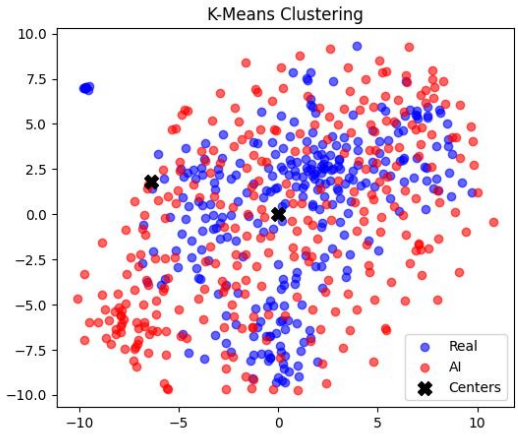

Figure 95: Clustering Result of Dimensions 200

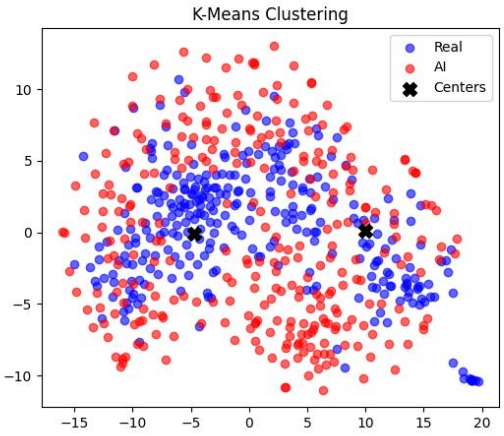

Figure 96: Clustering Result of Dimensions 100

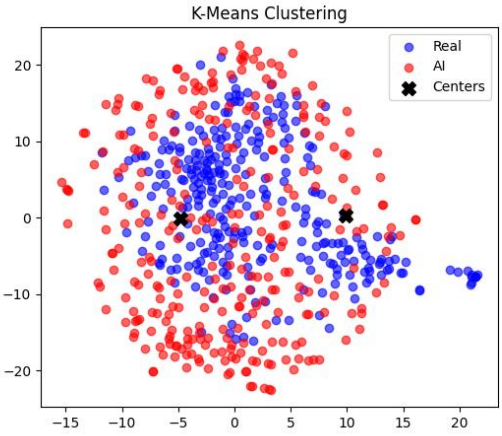

Figure 97: Clustering Result of Dimensions 50

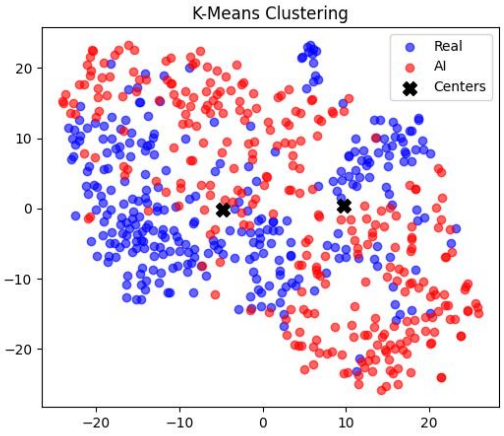

Figure 98: Clustering Result of Dimensions 10

Appendix B. Program Code

B.1 Web Crawler

Listing 1: Web crawler Code

```
import os
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
import requests

# Set the folder for saving images
SAVE_DIR = "raw_data/real_animal_images"
os.makedirs(SAVE_DIR, exist_ok=True)

# Start Chrome browser
service = Service(ChromeDriverManager().install())
options = webdriver.ChromeOptions()
options.add_argument("--headless") # Run without opening a browser window
options.add_argument("--disable-gpu")
options.add_argument("--window-size=1920x1080")
driver = webdriver.Chrome(service=service, options=options)

# Target URL
URL = "https://unsplash.com/s/photos/animal?license=free"
driver.get(URL)

# Simulate scrolling to ensure more images are loaded
click = False
for _ in range(50):
    if not click:
        try:
            # Try to find and click the "Load More" button
            load_more_button = driver.find_element(By.XPATH,
                "//button[contains(text(),'Load more')]")
            load_more_button.click() # Click the "Load More" button
            time.sleep(2) # Wait for images to load
            click = True
            print("Click success!")
        except Exception as e:
            print("No more 'Load More' button or failed to click:", e)
    driver.execute_script("window.scrollBy(0, 1000);")
    time.sleep(2)

# Grab all image URLs
images = driver.find_elements(By.TAG_NAME, "img")
image_urls = [img.get_attribute("src") for img in images if
    img.get_attribute("src")]
```

```
44
45 # Filter out base64 and advertisement images
46 filtered_urls = [url for url in image_urls if url and not
       url.startswith("data:image") and "https://images.unsplash.com/photo" in
       url]
47 print(len(filtered_urls))
48
49 # Close the browser
50 driver.quit()
51
52 # Download images
53 for i, img_url in enumerate(filtered_urls[:300]): # Limit to downloading 300
       images
54     try:
55         img_data = requests.get(img_url).content
56         with open(os.path.join(SAVE_DIR, f"{i+1}.jpg"), "wb") as f:
57             f.write(img_data)
58         print(f"Downloaded: animal_{i+1}.jpg")
59     except Exception as e:
60         print(f"Failed to download image {i+1}: {e}")
61
62 print("All images have been downloaded!")
```

B.2 Stable Diffusion web UI

Listing 2: Generate AI images Code

```
1 import requests
2 import os
3 import base64
4
5 # Stable Diffusion API endpoint
6 url = "http://127.0.0.1:7860/sdapi/v1/txt2img"
7
8 SAVE_DIR = "raw_data/ai_animal_images"
9 os.makedirs(SAVE_DIR, exist_ok=True)
10
11 # Read prompts
12 with open('prompts.txt', 'r') as f:
13     prompts = f.readlines()
14
15 with open('negative prompt.txt', 'r') as ff:
16     negative_prompts = ff.readline()
17
18 # Generate images
19 for i, prompt in enumerate(prompts):
20     payload = {
21         "prompt": prompt.strip(),
22         "negative_prompt": negative_prompts.strip(),
23         "steps": 50,
```

```
        "cfg_scale": 7,
        "width": 512,
        "height": 512,
        "sampler_index": "DPM++ 2M SDE",
        "scheduler": "Align Your Steps",
        "seed": -1
    }
    response = requests.post(url, json=payload)
    if response.status_code == 200:
        img_data = response.json().get("images")[0] # The returned image is
            base64 encoded
        # Decode base64 and save as a file
        img_bytes = base64.b64decode(img_data)
        img_path = os.path.join(SAVE_DIR, f"{i+1}.jpg")
        with open(img_path, "wb") as f:
            f.write(img_bytes)
        print(f"Generated: {img_path}")
    else:
        print(f"Failed to generate image {i+1}")
        print(response.status_code)

print("AI-generated animal images are complete!")
```

B.3 Cleaning Data

Listing 3: Clean data Code

```
import os
import random
from PIL import Image

# Original data folder
real_dir = "raw_data/real_animal_images"
ai_dir = "raw_data/ai_animal_images"

# Target folders
train_real_dir = "clean_data/train/real"
train_ai_dir = "clean_data/train/ai"
test_real_dir = "clean_data/test/real"
test_ai_dir = "clean_data/test/ai"

# Ensure target folders exist
for folder in [train_real_dir, train_ai_dir, test_real_dir, test_ai_dir]:
    os.makedirs(folder, exist_ok=True)

# Get all image paths
real_images = [os.path.join(real_dir, f) for f in os.listdir(real_dir)]
ai_images = [os.path.join(ai_dir, f) for f in os.listdir(ai_dir)]

# Shuffle the images randomly
```

```
random.shuffle(real_images)
random.shuffle(ai_images)

# Split into training/testing sets
train_real = real_images[:225]
test_real = real_images[225:300]
train_ai = ai_images[:225]
test_ai = ai_images[225:300]

def process_and_save(images, output_dir):
    for img_path in images:
        img = Image.open(img_path).convert("RGB")
        img_name = os.path.basename(img_path)
        img.save(os.path.join(output_dir, img_name))

# Move images to corresponding folders
process_and_save(train_real, train_real_dir)
process_and_save(test_real, test_real_dir)
process_and_save(train_ai, train_ai_dir)
process_and_save(test_ai, test_ai_dir)

print(f"Training set: Real: {len(train_real)}, AI: {len(train_ai)}")
print(f"Testing set: Real: {len(test_real)}, AI: {len(test_ai)}")
```

B.4 SVM

Listing 4: SVM Code

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from skimage.feature import hog
from sklearn.svm import SVC
from sklearn.utils import shuffle
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix,
    classification_report, roc_auc_score, roc_curve
from sklearn.decomposition import PCA

# Flip image horizontally
def flip_image(image):
    return cv2.flip(image, 1)

# Rotate image by small angle
def rotate_image(image, angle=10):
    h, w = image.shape[:2]
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
```

```
    return cv2.warpAffine(image, M, (w, h))

# Adjust brightness and contrast
def adjust_brightness_contrast(image, alpha=1.2, beta=10):
    return cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

# Add Gaussian noise
def add_gaussian_noise(image, mean=0, std=15):
    noise = np.random.normal(mean, std, image.shape).astype(np.uint8)
    return cv2.add(image, noise)

# Set HOG parameters
hog_params = dict(
    pixels_per_cell=(8, 8),
    cells_per_block=(3, 3),
    orientations=9,
    block_norm='L2-Hys'
)

def extract_hog_features(image_path, augment=False):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE) # Read in grayscale
    image = cv2.resize(image, (512, 512)) # Resize image
    if augment:
        image = add_gaussian_noise(image)
    features = hog(image, **hog_params)
    return features

# Read data
real_train = 'clean_data/train/real'
real_test = 'clean_data/test/real'
ai_train = 'clean_data/train/ai'
ai_test = 'clean_data/test/ai'
X_train, X_test, Y_train, Y_test = [], [], [], []

for img_file in os.listdir(real_train):
    X_train.append(extract_hog_features(os.path.join(real_train, img_file)))
    #X_train.append(extract_hog_features(os.path.join(real_train, img_file),
        augment=True))
    Y_train.append(0) # Label as real (0)
    #Y_train.append(0)
for img_file in os.listdir(real_test):
    X_test.append(extract_hog_features(os.path.join(real_test, img_file)))
    Y_test.append(0)
for img_file in os.listdir(ai_train):
    X_train.append(extract_hog_features(os.path.join(ai_train, img_file)))
    #X_train.append(extract_hog_features(os.path.join(ai_train, img_file),
        augment=True))
    Y_train.append(1) # Label as AI (1)
    #Y_train.append(1)
for img_file in os.listdir(ai_test):
    X_test.append(extract_hog_features(os.path.join(ai_test, img_file)))
```

```
    Y_test.append(1)

X_train = np.array(X_train)
X_test = np.array(X_test)
Y_train = np.array(Y_train)
Y_test = np.array(Y_test)

# Set random seed for reproducibility
random_seed = 42

# Shuffle X_train and Y_train
X_train, Y_train = shuffle(X_train, Y_train, random_state=random_seed)
print(len(X_train), len(Y_train), len(X_test), len(Y_test), X_train.shape,
    X_test.shape)

'''pca = PCA() # Calculate how many components can retain 95% variance after
    dimensionality reduction
pca.fit(X_train)
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
n_components_95 = np.argmax(cumulative_variance >= 0.95) + 1
print(f"Number of components to retain 95% variance: {n_components_95}")
exit(0)'''

pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
print(X_train_pca.shape, X_test_pca.shape)

# Train SVM
svm = SVC(kernel='linear', probability=True)
svm.fit(X_train_pca, Y_train)

# Prediction & Evaluation
y_pred = svm.predict(X_test_pca)
accuracy = accuracy_score(Y_test, y_pred)
conf_matrix = confusion_matrix(Y_test, y_pred)
report = classification_report(Y_test, y_pred)
roc_auc = roc_auc_score(Y_test, svm.predict_proba(X_test_pca)[:, 1])

# Plot Confusion Matrix
plt.figure(figsize=(9, 9))
sns.heatmap(conf_matrix, annot=True, fmt=".3f", linewidths=0.5, square=True,
    cmap="mako")
plt.ylabel('Actual Label', size=12)
plt.xlabel('Predicted Label', size=12)
plt.title(f'Accuracy Score: {accuracy:.4f}', size=16)
plt.savefig('SVM_graph/confusion_matrix_SVM13.jpg')
plt.show()

# Cross-validation
cv_scores = cross_val_score(svm, X_train_pca, Y_train, cv=5)
```

```
119
120 # Plot ROC Curve
121 fpr, tpr, _ = roc_curve(Y_test, svm.predict_proba(X_test_pca)[:, 1])
122 plt.plot(fpr, tpr, label=f'AUROC = {roc_auc:.2f}')
123 plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
124 plt.xlabel('False Positive Rate')
125 plt.ylabel('True Positive Rate')
126 plt.title('ROC Curve')
127 plt.legend()
128 plt.savefig('SVM_graph/ROC_SVM13.jpg')
129 plt.show()
130
131 # Output results
132 print(f'Accuracy: {accuracy * 100:.2f}%')
133 print(f'AUROC = {roc_auc:.2f}')
134 print(f'Confusion Matrix:\n{conf_matrix}')
135 print(f'Classification Report:\n{report}')
136 print(f'Cross-Validation Accuracy: {cv_scores.mean():.4f} ±
    {cv_scores.std():.4f}')
```

B.5 Logistic Regression

Listing 5: Logistic Regression Code

```
1  import os
2  import cv2
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import seaborn as sns
6  from skimage.feature import hog
7  from sklearn.linear_model import LogisticRegression
8  from sklearn.utils import shuffle
9  from sklearn.model_selection import cross_val_score
10 from sklearn.metrics import accuracy_score, confusion_matrix,
       classification_report, roc_auc_score, roc_curve
11 from sklearn.decomposition import PCA
12
13 # Flip image horizontally
14 def flip_image(image):
15     return cv2.flip(image, 1)
16
17 # Rotate image by a small angle
18 def rotate_image(image, angle=10):
19     h, w = image.shape[:2]
20     center = (w // 2, h // 2)
21     M = cv2.getRotationMatrix2D(center, angle, 1.0)
22     return cv2.warpAffine(image, M, (w, h))
23
24 # Adjust brightness and contrast
25 def adjust_brightness_contrast(image, alpha=1.2, beta=10):
```

```
    return cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

# Add Gaussian noise
def add_gaussian_noise(image, mean=0, std=15):
    noise = np.random.normal(mean, std, image.shape).astype(np.uint8)
    return cv2.add(image, noise)

# Set HOG parameters
hog_params = dict(
    pixels_per_cell=(8, 8),
    cells_per_block=(3, 3),
    orientations=9,
    block_norm='L2-Hys'
)

def extract_hog_features(image_path, augment=False):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE) # Read in grayscale
    image = cv2.resize(image, (512, 512)) # Resize image
    if augment:
        image = add_gaussian_noise(image)
    features = hog(image, **hog_params)
    return features

# Load data
real_train = 'clean_data/train/real'
real_test = 'clean_data/test/real'
ai_train = 'clean_data/train/ai'
ai_test = 'clean_data/test/ai'
X_train, X_test, Y_train, Y_test = [], [], [], []
for img_file in os.listdir(real_train):
    X_train.append(extract_hog_features(os.path.join(real_train, img_file)))
    #X_train.append(extract_hog_features(os.path.join(real_train, img_file),
        augment=True))
    Y_train.append(0) # Label as real (0)
    #Y_train.append(0)
for img_file in os.listdir(real_test):
    X_test.append(extract_hog_features(os.path.join(real_test, img_file)))
    Y_test.append(0)
for img_file in os.listdir(ai_train):
    X_train.append(extract_hog_features(os.path.join(ai_train, img_file)))
    #X_train.append(extract_hog_features(os.path.join(ai_train, img_file),
        augment=True))
    Y_train.append(1) # Label as AI (1)
    #Y_train.append(1)
for img_file in os.listdir(ai_test):
    X_test.append(extract_hog_features(os.path.join(ai_test, img_file)))
    Y_test.append(1)

X_train = np.array(X_train)
X_test = np.array(X_test)
Y_train = np.array(Y_train)
```

```
Y_test = np.array(Y_test)

# Set random seed for reproducibility
random_seed = 42

# Shuffle X_train and Y_train
X_train, Y_train = shuffle(X_train, Y_train, random_state=random_seed)
print(len(X_train), len(Y_train), len(X_test), len(Y_test), X_train.shape,
    X_test.shape)

pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
print(X_train_pca.shape, X_test_pca.shape)

# Train Logistic Regression
clf = LogisticRegression(penalty='l1', solver='liblinear')
clf.fit(X_train_pca, Y_train)

# Prediction & Evaluation
y_pred = clf.predict(X_test_pca)
accuracy = accuracy_score(Y_test, y_pred)
conf_matrix = confusion_matrix(Y_test, y_pred)
report = classification_report(Y_test, y_pred)
roc_auc = roc_auc_score(Y_test, clf.predict_proba(X_test_pca)[:, 1])

# Plot Confusion Matrix
plt.figure(figsize=(9,9))
sns.heatmap(conf_matrix, annot=True, fmt=".3f", linewidths=0.5, square=True,
    cmap="mako")
plt.ylabel('Actual Label', size=12)
plt.xlabel('Predicted Label', size=12)
plt.title(f'Accuracy Score: {accuracy:.4f}', size=16)
plt.savefig('LR_graph/confusion_matrix_LR11.jpg')
plt.show()

# Cross-validation
cv_scores = cross_val_score(clf, X_train_pca, Y_train, cv=5)

# Plot ROC Curve
fpr, tpr, _ = roc_curve(Y_test, clf.predict_proba(X_test_pca)[:, 1])
plt.plot(fpr, tpr, label=f'AUROC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.savefig('LR_graph/ROC_LR11.jpg')
plt.show()

# Output results
```

```
print(f'Accuracy: {accuracy * 100:.2f}%')
print(f'AUROC = {roc_auc:.2f}')
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{report}')
print(f'Cross-Validation Accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}')
```

B.6 K-Means

Listing 6: K-Means Code

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from skimage.feature import hog
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from scipy.optimize import linear_sum_assignment

# Left-right flip
def flip_image(image):
    return cv2.flip(image, 1)

# Small angle rotation
def rotate_image(image, angle=10):
    h, w = image.shape[:2]
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    return cv2.warpAffine(image, M, (w, h))

# Adjust brightness and contrast
def adjust_brightness_contrast(image, alpha=1.2, beta=10):
    return cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

# Add noise
def add_gaussian_noise(image, mean=0, std=15):
    noise = np.random.normal(mean, std, image.shape).astype(np.uint8)
    return cv2.add(image, noise)

# Set HOG parameters
hog_params = dict(
    pixels_per_cell=(8, 8),
    cells_per_block=(2, 2),
    orientations=9,
    block_norm='L2-Hys'
```

```
)

def extract_hog_features(image_path, augment=False):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE) # Read in grayscale
    image = cv2.resize(image, (512, 512)) # Resize
    if augment:
        image = add_gaussian_noise(image)
    features = hog(image, **hog_params)
    return features

# Load data
real_train = 'clean_data/train/real'
real_test = 'clean_data/test/real'
ai_train = 'clean_data/train/ai'
ai_test = 'clean_data/test/ai'
X_full, y_true = [], []
for img_file in os.listdir(real_train):
    X_full.append(extract_hog_features(os.path.join(real_train, img_file)))
    #X_full.append(extract_hog_features(os.path.join(real_train, img_file),
        augment=True))
    y_true.append(0) # Label as real (0)
    #y_true.append(0)
for img_file in os.listdir(real_test):
    X_full.append(extract_hog_features(os.path.join(real_test, img_file)))
    #X_full.append(extract_hog_features(os.path.join(real_test, img_file),
        augment=True))
    y_true.append(0)
    #y_true.append(0)
for img_file in os.listdir(ai_train):
    X_full.append(extract_hog_features(os.path.join(ai_train, img_file)))
    #X_full.append(extract_hog_features(os.path.join(ai_train, img_file),
        augment=True))
    y_true.append(1) # Label as AI (1)
    #y_true.append(1)
for img_file in os.listdir(ai_test):
    X_full.append(extract_hog_features(os.path.join(ai_test, img_file)))
    #X_full.append(extract_hog_features(os.path.join(ai_test, img_file),
        augment=True))
    y_true.append(1)
    #y_true.append(1)

X_full = np.array(X_full)
y_true = np.array(y_true)

# Set random seed for reproducibility
random_seed = 42

# Shuffle X_train and Y_train
X_full, y_true = shuffle(X_full, y_true, random_state=random_seed)
print(len(X_full), len(y_true), X_full.shape, y_true.shape)

```

```
pca = PCA(n_components=10)
X_full = pca.fit_transform(X_full)
print(X_full.shape)

# Clustering
kmeans = KMeans(n_clusters=2, random_state=random_seed)
y_pred = kmeans.fit_predict(X_full)

# Evaluation
ari = adjusted_rand_score(y_true, y_pred)
nmi = normalized_mutual_info_score(y_true, y_pred)

# Compute Clustering Accuracy (Hungarian Algorithm)
contingency_matrix = np.zeros((2, 2))
for i in range(len(y_true)):
    contingency_matrix[y_true[i], y_pred[i]] += 1

row_ind, col_ind = linear_sum_assignment(-contingency_matrix)
accuracy = contingency_matrix[row_ind, col_ind].sum() / len(y_true)

# Remap clusters to match true labels
mapped_clusters = np.zeros_like(y_pred)
for i, j in zip(row_ind, col_ind):
    mapped_clusters[y_pred == j] = i

# Reduce dimensions for visualization
X_embedded = TSNE(n_components=2, random_state=random_seed,
    perplexity=30).fit_transform(X_full)

colors = ['blue', 'red']
label_names = ["Real", "AI"]

# Plot Clusters
plt.figure(figsize=(6, 5))
for i, label in enumerate(np.unique(y_true)): # Iterate over unique labels
    (Only 0 and 1)
    plt.scatter(
        X_embedded[y_true == label, 0],
        X_embedded[y_true == label, 1],
        c=colors[i],
        label=label_names[i],
        alpha=0.6
    )
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
    c='black', marker='X', s=100, label="Centers")
plt.title('K-Means Clustering')
plt.legend()
plt.savefig('K_graph/result17.jpg')
plt.show()

# Output results
```

```
135 print(f"ARI: {ari:.4f}")
136 print(f"NMI: {nmi:.4f}")
137 print(f"Clustering accuracy: {accuracy * 100:.2f}")
```