



# CS464: Introduction to Machine Learning

*Project Final Report*

---

**CAR VS BIKE CLASSIFICATION**

# Table of Contents

<b>Table of Contents</b>	<b>1</b>
<b>Introduction</b>	<b>2</b>
<b>Problem Description</b>	<b>3</b>
<b>Methods</b>	<b>3</b>
Dataset Split, Data Preprocessing and Analysis	3
Coding Environment	6
Trained Models Information	6
K-Nearest Neighbors (KNN)	6
Random Forest	7
Support Vector Machine (SVM)	8
Convolutional Neural Network (CNN)	9
Methodology	10
K-Nearest Neighbors (KNN)	10
Random Forest	10
Support Vector Machine	10
Convolutional Neural Networks (CNN)	10
<b>Results and Discussions</b>	<b>11</b>
K-Nearest Neighbors (KNN)	11
Effects of PCA	12
Evaluation Metrics	13
Random Forest	13
Support Vector Machine (SVM)	17
Convolutional Neural Network (CNN)	19
Evaluation Metrics	21
Comparison of Models	22
<b>Conclusion</b>	<b>24</b>
<b>Appendix</b>	<b>25</b>
Contribution of Each Team Member	25
<b>References</b>	<b>26</b>

# Introduction

In the evolving landscape of computer vision, image classification stands out as a fundamental and challenging task that can address many important problems. With the incoming advancements that utilize image classification such as surveillance systems, autonomous vehicles, and object recognition technologies, the need for accurate and efficient image classification models have started to become more critical. In this project, we focused on classifying car and bike snapshots to label them correctly using machine learning applications which can potentially contribute to surveillance systems, traffic analysis and management and car driving assistance systems.

In this classification task, we used the Car vs Bike Classification Dataset from Kaggle.com [1]. The dataset contains 2000 images of cars and 2000 images of motorbikes of different aspect ratios, sizes, and orientations, introducing variation to the dataset. To classify these images, we benchmarked and used 3 standard supervised learning algorithms: K-Nearest-Neighbors (KNN), Random Forest, Support Vector Machine (SVM), and 1 deep learning algorithm that is Convolutional Neural Networks (CNN). The dataset was composed of images of different sizes, thus we fixed all of the images' size to 64x64 and further processed the images to make them suitable for using them in our project by flattening them and converting them to the grayscale mode. Furthermore, we performed data augmentation to increase the data size and variation in the dataset as we found the initial number of images in the dataset insufficient. We applied horizontal flips, random Gaussian blur, random noise, and random crop for data augmentation. Other than that, PCA with different values is experimented on each algorithm and the results are plotted to see the effect of PCA. We splitted the dataset to use 60 percent of the dataset as the training set, the 20 percent of it as the validation set and the 20 percent of it as the test set. We used Google Colab to implement the algorithms and perform the aforementioned applications. We trained different combinations for each algorithm in the validation phase and found the best model for each model. Furthermore, we tested each algorithm's best performing variation in the test phase and reported their accuracies. We found that CNN with augmented data performs the best with 89 percent accuracy rate. Other than that, we found that KNN without augmentation performs with 70 percent accuracy rate and augmented KNN performs with 69 percent accuracy rate. For SVM, we found that Non-Augmented SVM performs with 83.13 percent accuracy rate whereas Augmented SVM performs with 83.50 percent accuracy rate. Furthermore, we reported that Random Forest without augmentation performs with accuracy rate 81.13 percent and augmented Random Forest performs with 80.25 percent accuracy rate. Finally, we reported that CNN without augmentation performs with accuracy rate 88.75 percent whereas CNN with augmentation performs with accuracy rate 89.88 percent which is the best accuracy rate as it stated before.

In the following sections, we first define the problem in “Problem Description”, then we lay out our methodology and methods in the “Methods” section. Furthermore, we report our results in the “Results” section and outline our inference in the “Discussions” section. Finally, we specify our final remarks in the “Conclusion” section.

# Problem Description

The problem at hand is a binary image classification task, focused on distinguishing between bikes and cars within a dataset that consists of 2,000 unique 64x64 images. The primary objective is to develop a machine learning model that can accurately classify an input image as either a bike or a car. This problem necessitates understanding the fundamental visual features that differentiate these two classes. Therefore, three different machine learning models and one deep learning model are used to address this problem which are K-Nearest Neighbours, Random Forest, and Support Vector Machine for machine learning models and Convolutional Neural Networks for the deep learning model to be used. This project may be used in the processes of traffic analysis and management to gather information about the numbers, movements, density of cars and bikes in traffic which could help in planning roads to reduce congestion in traffic. Besides this purpose, it may also be used in cars' driving assistance technologies in order to correctly detect motorbikes on the road which are more vulnerable than cars in traffic [2]. Another field of use is urban planning where the regional administrations may use it to obtain the density of motorbikes and cars to manage parking spaces in the cities. Other than addressing the project's contribution to incoming technological advancements in the society, the project also addresses the question how can the models be trained to comprehend the intra-variety within the classes of bikes and cars and what visual features and structural elements should be prioritized to successfully distinguish between the different types within each class. Furthermore, the project also addresses the question what data augmentation techniques should be employed to enrich the dataset and enhance the model's ability to generalize. Finally, the project also tries to find answers to the question how should the dataset be divided into training and validation subsets for effective model training and evaluation.

# Methods

## Dataset Split, Data Preprocessing and Analysis

The first challenge we faced when reading the image data into numpy arrays was that the dataset's images were colorful, and all had different sizes. When reading the images into memory, we fixed the target size to 64x64, which we found to be a good enough size for the dataset. We also ignored the RGB channels each pixel had for simplicity and to increase our model's training and predicting time and set the color mode to grayscale.

One other challenge we faced was that after reading all the images and creating the all\_images list, which is a numpy array containing the data, it turned out to be 4 dimensional: the number of samples, the width of the pixel, the height of the pixel and the alpha value determining the transparency of the pixel. However, this structure of the list was inconvenient for the 3 ML algorithms we chose to implement as we needed to represent the image with an array of 2 dimensions: number of samples and number of features. So, we overcame this issue by flattening the image array containing the pixel information and turning the last 3

dimensions into one dimension by putting pixel values side by side and having  $64 \times 64 = 4096$  features. The code for flattening the image array can be found in Figure 1.

## Data Read

```
# Function to load images from directories
def load_images_from_folder(folder, flatten):
    images = []
    labels = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = tf.keras.preprocessing.image.load_img(img_path, target_size=(64,64), color_mode='grayscale')
        img_array = tf.keras.preprocessing.image.img_to_array(img)
        # Flatten image into 1D array
        if flatten:
            images.append(img_array.flatten())
        else:
            images.append(img_array)
        labels.append(folder)
    return np.array(images), labels

# Directory paths for bikes and cars images
bikes_folder = 'Car-Bike-Dataset/Bike'
cars_folder = 'Car-Bike-Dataset/Cars'

bike_images, bike_labels = load_images_from_folder(bikes_folder, False)
car_images, car_labels = load_images_from_folder(cars_folder, False)

all_images = np.vstack((bike_images, car_images))
all_labels = np.array(bike_labels + car_labels)

bike_images_flat, bike_labels_flat = load_images_from_folder(bikes_folder, True)
car_images_flat, car_labels_flat = load_images_from_folder(cars_folder, True)
|
all_images_flat = np.vstack((bike_images_flat, car_images_flat))
all_labels_flat = np.array(bike_labels_flat + car_labels_flat)
```

Figure 1: Code Snippet for Image Flattening

To increase our model's robustness and tolerance against noises, we used data augmentation techniques that helped us both increase the size of our somewhat limited data and introduced randomness and variation to the dataset so that it could represent real world data better. The diversity we added to the dataset with this technique also helped us to prevent overfitting our models to the dataset. Note that the augmentation techniques we applied to the dataset may increase or decrease the achieved accuracy depending on the augmentation applied. In our experiments, we found out that applying random noises, random blurs, and flipping the images increased the accuracy of our models, while randomly cropping some parts of the image reduced the accuracy as apparently random cropping did not correctly represent the images in the test set we have in our experiment. The code for applying data augmentation to the dataset can be seen in Figure 2.

# Image Augmentation

```
# Function for image augmentation using imgaug
def apply_image_augmentation(images, labels):
    # Define an augmentation pipeline
    seq = iaa.Sequential([
        iaa.Fliplr(0.5), # horizontal flips
        # iaa.Crop(percent=(0, 0.1)), # random crops
        iaa.Sometimes(0.5, iaa.GaussianBlur(sigma=(0, 0.5))), # random Gaussian blur
        iaa.Sometimes(0.5, iaa.AdditiveGaussianNoise(scale=(0, 0.05*255))), # random noise
    ], random_order=True)

    augmented_images = seq(images=images)
    augmented_labels = labels

    return augmented_images, augmented_labels

# Apply image augmentation to training data
X_train_aug, y_train_aug = apply_image_augmentation(X_train, y_train)
X_train_aug = np.vstack((X_train_aug, X_train))
y_train_aug = np.hstack((y_train_aug, y_train))

X_train_aug_flat, y_train_aug_flat = apply_image_augmentation(X_train_flat, y_train)
X_train_aug_flat = np.vstack((X_train_aug_flat, X_train_flat))
y_train_aug_flat = np.hstack((y_train_aug_flat, y_train))
```

Figure 2: Code for Image Augmentation

To split the dataset into train, validation and test, we used the `train_test_split` function that came with the `scikit.model_selection` package and randomly split the dataset into two by 80% and 20%. We then split the train portion of the data furthermore to obtain a train-test-validation split of 60-20-20. While training the models, we used the `X_train` generated by this splitting procedure, and to validate the accuracy of our models with varying hyperparameters in our experiments, we used the `X_validation` generated by this procedure. To come up with final evaluation metrics of our best performing models, we concatenated the `X_train` and `X_validation` and trained our models again with the training set being `X_train + X_validation` and posted the results and evaluation metrics using the `X_test` portion of the data. The code for partitioning the dataset into train, validation and test subsets can be seen in Figure 3.

```

# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(all_images, all_labels, test_size=0.2, random_state=44)
X_train_flat, X_test_flat, y_train_flat, y_test_flat = train_test_split(all_images_flat, all_labels_flat, test_size=0.2, random_state=44)

# Perform validation split
X_val, X_train = np.split(X_train, [800])
y_val, y_train = np.split(y_train, [800])

X_val_flat, X_train_flat = np.split(X_train_flat, [800])
y_val_flat, y_train_flat = np.split(y_train_flat, [800])

# Apply fit_transform to labels to make them numeric
lb = LabelEncoder()
y_train = lb.fit_transform(y_train)
y_val = lb.fit_transform(y_val)
y_test = lb.fit_transform(y_test)

```

*Figure 3: Train - Test - Validation Partition*

## Coding Environment

We used Google Colab [4] to implement the code for training, validating, and testing the models with the Car vs Bike Classification [1] dataset. Google Colab, short for Google Colaboratory, is a hosted Jupyter Notebook service that provides access to Google's computing resources, such as GPUs, to train machine learning models. Other than Google Colab [4], we used the following Python libraries throughout our progress in the project:

- Numpy: we used numpy to utilize its functions that perform various mathematical operations on matrices and arrays. Training, validating, and testing the machine learning models require a great deal of mathematical operations that are available in this library. Hence, we utilized numpy.
- Tensorflow: Tensorflow is a rich library that can be used for various machine learning procedures. We imported Tensorflow's models regarding image data generation and 2D array models to train, validate, and test our respective models.
- Sklearn: Sklearn is a Python library that contains many efficient tools for predictive data analysis, machine learning, and statistical models. We used sklearn's "KNeighborsClassifier", "RandomForestClassifier", and "SVC" classifiers to represent our models and imported "train\_test\_split" and "accuracy\_score" modules for data split and accuracy prediction, respectively. We also used the "LabelEncoder" module to encode target labels.
- imgaug: "imgaug" is an image augmentation library that can be used for machine learning experiments. It supports many augmentation techniques. We imported the "augmenters" package from "imgaug" to perform image augmentation.

Other than these external libraries of Python, we imported the "os" module to utilize operating system dependent functionality such as the "listdir" method to get all files and directories in the specified directory.

## Trained Models Information

### K-Nearest Neighbors (KNN)

The k-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning classifier that can be utilized for regression and classification problems. It is normally used as a

classification algorithm, stemming from the intuition that similar points can be found near one another. KNN uses proximity to make classifications and predictions about the label of an individual data point. In other words, KNN algorithm works based on finding the “k” specified closest neighbors of a test data point on the set of training data points and classifies the data point based on the majority of its specified neighbors [5]. The determination of closest k neighbors is done using one of the following metrics:

- Euclidean Distance (p=2)

$$d(x, y) = \sqrt{\sum_{i=1}^n (y_i - x_i)^2}$$

It is the most commonly used distance measure. Using the formula above, Euclidean Distance measures a straight line between the two points being measured.

- Manhattan Distance (p=1)

$$d(x, y) = \left( \sum_{i=1}^m |x_i - y_i| \right)$$

Manhattan Distance measures the absolute value between two points. It is also referred to as taxicab distance or city block distance because it is visualized with a grid, looking like one navigating from one address to another in a city street.

- Minkowski Distance:

$$\text{Minkowski Distance} = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

Minkowski Distance is the generalized form of Euclidean and Manhattan distance metrics. The parameter  $p$  in the formula enables the formation of other distance metrics [5]. The KNN algorithm that is used in this project uses Minkowski Distance as its metric to evaluate its accuracy.

## Random Forest

Random Forest algorithm is a supervised learning model that randomly selects (or bags) data from the sample for training and labels the remaining as “Out of Bag”. Then, using the bootstrapped dataset as the training dataset, it randomly selects features for decision tree formations and forms the trees. The algorithm forms n different decision trees using this method. It then validates the current model based on the Out Of Bag samples performance from formed decision trees and determines an accuracy rate [6].

When the Random Forest algorithm is used to solve regression problems, mean squared error (MSE) is used to solve the respective problems. The mean squared error formula is as follows:

$$MSE = \frac{1}{N} \sum_{i=1}^N (f_i - y_i)^2$$

where  $N$  is the number of data points,  $f_i$  is the value that model returned and  $y_i$  is the actual value for data point  $i$ . Mean Squared Error formula calculates the distance between each node and the predicted actual value, and determines which branch is the better option for the forest.

On the other hand, *Gini index* is used to decide which of the branches is more likely to occur while performing Random Forest algorithm in classification problems. *Gini index* is as follows:

$$Gini = 1 - \sum_{i=1}^c (p_i)^2$$

where  $p_i$  is the relative frequency of the class that is observed and  $c$  represents the number of classes. This formula determines the Gini of each branch on a node which determines which of the branches is more likely to occur as mentioned above.

Other than Gini index, Entropy can also be used to determine how nodes branch in a decision tree. Entropy's formula is as follows:

$$Entropy = \sum_{i=1}^c -p_i * \log_2(p_i)$$

where  $p_i$  is the relative frequency of the class that is observed and  $c$  represents the number of classes. Entropy determines how the node should branch based on the probability of a certain outcome [7].

## Support Vector Machine (SVM)

Support Vector Machine is a supervised machine learning algorithm for classification and regression problems. SVM is usually used in classification objectives, and our objective also falls under classification. Support vector machine aims to find a hyperplane in an  $N$ -dimensional space where  $N$  is the number of features that specifies the data points uniquely. As there are many hyperplanes possible to separate the two classes of data points, the algorithm's aim is to find the plane with the maximum margin which is the maximum distance between data points of both classes. Maximizing the margin distance enables the algorithm to classify future data points more efficiently [8].

The advantages of support vector machines are that it is effective in high dimensional spaces, memory efficient, and versatile in a sense that different Kernel functions can be specified for the decision function.

In SVM, we want a classifier which is a linear separator with as big a margin as possible. We need to minimize  $\|w\|$  to maximize the margin and with the condition that there are no data points between  $H_1$  and  $H_2$ :

$$x_i \cdot w + b \geq +1 \text{ when } y_i = +1$$

$$x_i \cdot w + b \leq -1 \text{ when } y_i = -1$$

which can be combined into:

$$y_i(x_i \cdot w) \geq 1$$

Therefore, the main problem is to minimize  $\|w\|$  such that discrimination boundary is obeyed:

$$\min f: \frac{1}{2} \|w\|^2$$

$$g: y_i(w \cdot x_i) - b = 1 \text{ or } [y_i(w \cdot x_i) - b] - 1 = 0$$

This is a constrained optimization problem that can be solved by the Lagrangian multiplier method as it is quadratic while the surface is a paraboloid and it has a single global minimum [8].

### Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a type of deep learning model for processing grid-like data patterns, such as images. CNN, in general, consists of 3 parts: Convolution Layers, Pooling Layers, and Fully Connected Layers. Convolution and pooling layers are responsible for feature extraction, while fully connected layers map these features to a final class label. In the case of image classification, CNN starts with an input image, and this input is passed to a convolution layer. This convolution layer applies convolution operation to the input data with a kernel (a kernel is a matrix of weights that is used by convolution layers to extract features from the given input). The result of this convolution is called a feature map. After the convolution, each element of the feature map is passed through a nonlinear activation function. This nonlinear activation function is generally the rectified linear unit (ReLU). The definition of ReLU is given in Figure 4.

$$f(x) = x^+ = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases}$$

*Figure 4: The Definition of ReLu [9]*

After this convolution layer, a convolution layer or a pooling layer comes with the feature map as the input. Pooling layers are used to reduce the spatial dimensions (width and height) of the input volume. Efficient pooling layers try to retain the most valuable information from the previous layer while reducing the input dimensions. Max pooling, one of the most used pooling methods, works by choosing the maximum element from the input in non overlapping windows of size  $n \times n$  (usually  $2 \times 2$ ). After a sequence of convolution and pooling layers, the last output is generally flattened and passed to fully connected layers. Fully connected layers then map the last output to a class label. Fully connected layers consist of nodes connected to the previous layers' nodes by weights. If the previous layer has  $n$  nodes and the fully connected layer has  $m$  nodes, there will be  $n \times m$  weights. Each node represents the weighted sum of previous layers nodes. After fully connected layers, the last nodes represent the probability of each class label. One of the main advantages of the CNN method

starts after this sequence of layers. Backpropagation in CNN involves iteratively adjusting the weights of the network based on the gradients of the loss function with respect to the weights. In this way, the accuracy of CNN increases with each iteration (assuming no overfitting) [10].

## Methodology

KNN, Random Forest, SVM, and CNN algorithms are trained and tested with different hyperparameters to find the best model for each of the algorithms and their different types of variations. We have followed the following steps to train and validate the best model for the selected algorithms:

### K-Nearest Neighbors (KNN)

We trained KNN's augmented and non-augmented versions and calculated their validation accuracies with respect to increasing number of neighbors to find the best model for KNN with the best performing number of neighbors. We also employed PCA to find the most suitable number of features to choose the most optimal model for the algorithm. After the validation phase, we calculated the test accuracy of the best selected model in the test phase.

### Random Forest

We trained Random Forest's augmented and non-augmented versions and calculated their validation accuracies with respect to the number of increasing decision trees to find the best model for Random Forest algorithm with the best performing number of decision trees. We also employed PCA to Random Forest to find the most suitable number of features to choose the most optimal model for the algorithm. After the validation phase we calculated the test accuracy with the test dataset.

### Support Vector Machine

We trained SVM's augmented and non-augmented versions and calculated their validation accuracies with respect to different kernel types. We also employed PCA to the SVM to find the best number of features to train SVM with. After finding the best model for the SVM with a specific kernel type, we calculated the test accuracy of the SMV with the test dataset.

### Convolutional Neural Networks (CNN)

We train four different models of CNN that have different numbers of convolutional and pooling layers. After training these CNN models, we compare their validation accuracies and pick the best model with the highest accuracy. After finding the best model for the CNNi we calculated the test accuracy of the CNN with the test dataset.

# Results and Discussions

## K-Nearest Neighbors (KNN)

We have experimented with different k values to tune the hyperparameter on both the augmented and the non-augmented model, and aimed to find the best k value for those models. Below, it can be seen how kNN performs with different k values for both the augmented train dataset and the non-augmented train dataset. We observe that the accuracy seems to be fluctuating a lot with different k values or number of neighbors and is not monotonically increasing or decreasing. We observe for both cases of models with augmented dataset and the non-augmented dataset, their accuracy peaks at some particular k values or number of neighbors, and after that point it gradually decreases. We see that the augmented dataset model performs best at k = 9 and the accuracy level decreases with increasing k values and the non-augmented dataset model performs best at k = 5 and peaks there. Also, we can see that the dataset augmented model performs better in terms of accuracy, as the accuracy level of the peaking k value in the augmented model is higher than the accuracy level of the peaking k value in the non-augmented model. It is natural that the accuracy starts to decrease after a particular k value because by increasing the k value, we get closer to overfitting the model to the dataset and the fitted model loses its generalizability property and cannot predict correctly to the validation portion of the initial dataset. Also, we can deduce from the graph and the fluctuations happening that, after the train-test-validation data split, the training portion of the dataset may have got too imbalanced and the distribution of bike images and car images might have different densities. We can conclude that the data augmentation has been successful and it does very well represent the data in the validation set as the augmented model performed better than the non-augmented model in terms of accuracy and fluctuates much less than the non-augmented model.

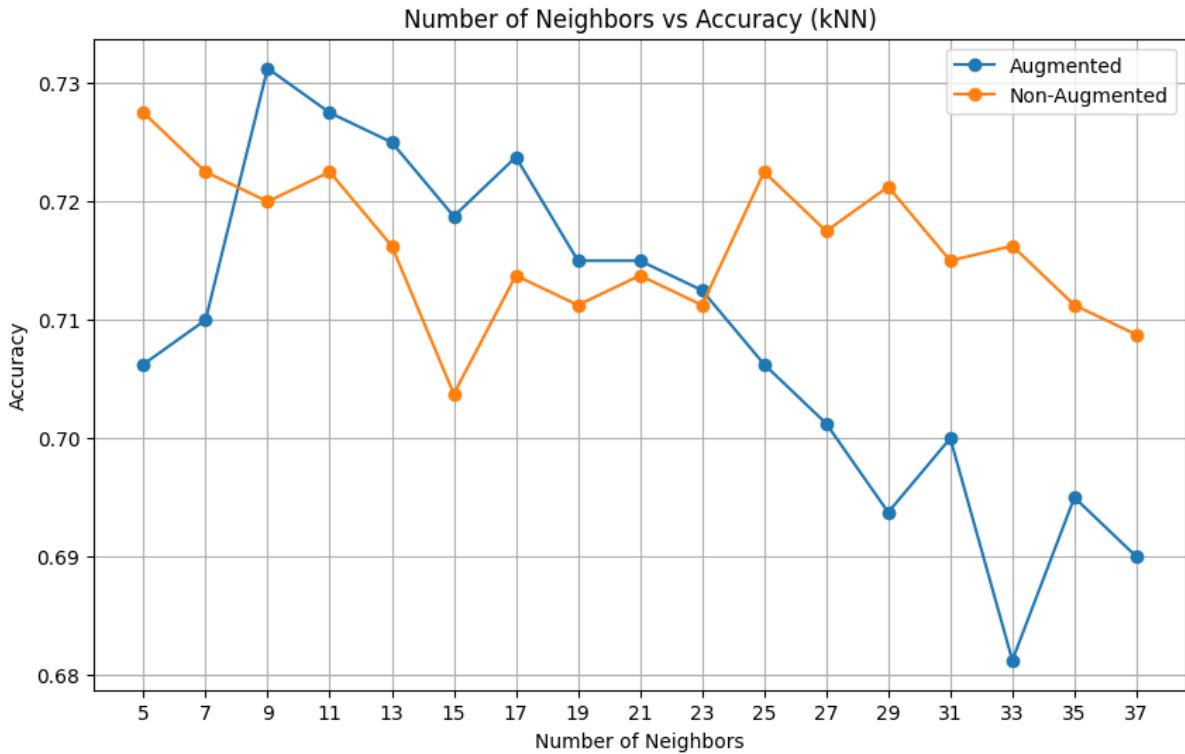


Figure 5: Number of Neighbors vs Accuracy for Augmented and Non-Augmented KNN

### Effects of PCA

To observe the effects of PCA, we have applied the PCA on our training dataset with different numbers of components with the values (1, 50, 100, 250, 500, 1000, 2400). The aim of this experiment was to observe and note for which values of n\_components, the PCA was helpful and for which values of n\_components it starts to decrease the accuracy. In the below graph, we see the n\_components at the x-axis and accuracy at the y-axis and see the plots of the model with augmented dataset and the non-augmented dataset. We see that for both the models, accuracy peaks at n\_components = 50 and starts to decrease after that point. We can infer from this data that the first 50 Principal Components seem to be carrying the most distinctive features as compared to the other numbers of Principal Components. At n\_components = 1, it is natural for the accuracy for both models to be so low because only one Principal Component cannot successfully represent the dataset. For n\_components values after 50, we can deduce that PCA starts to include some noisy features that are not really helpful for our models to discriminate using these Principal Components. Comparing the two models, it makes sense that the augmented version lies above the non-augmented version for most of the points as it performed better than the latter overall, but for some points where they are so close or the non-augmented is above the augmented, we can understand that the augmented dataset may contain some features that are included in the Principal Components that are not very well represented in the validation set.

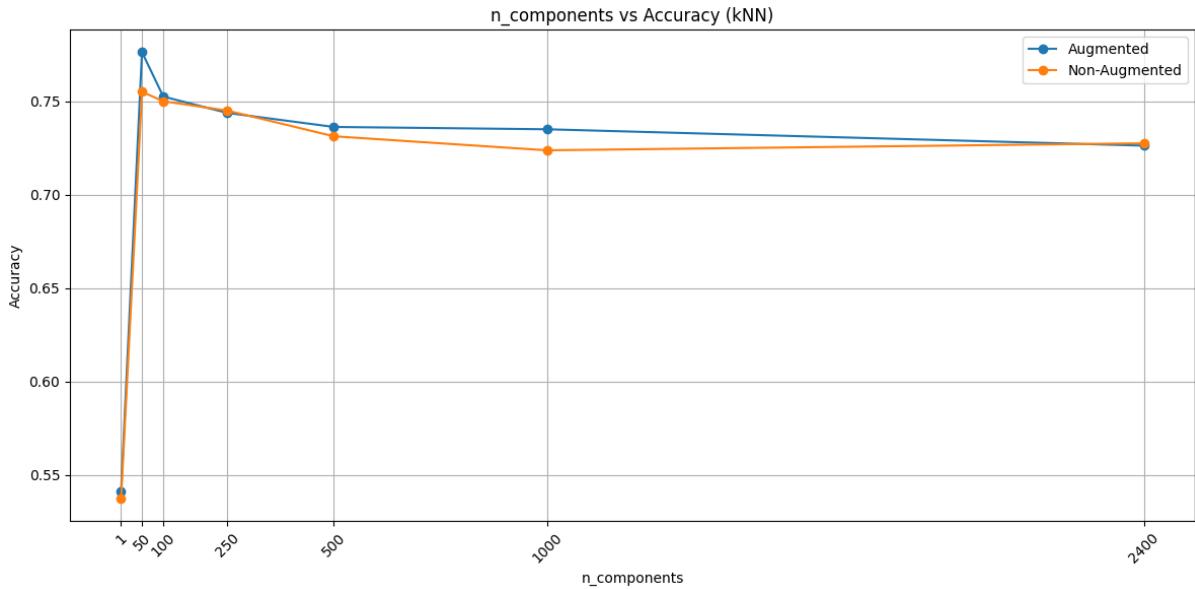


Figure 6: PCA Analysis of Augmented and Non-Augmented KNN

## Evaluation Metrics

Below, we can see the additional performance metrics and the confusion matrix we found for the kNN model for both versions of the models with their best hyperparameter k values. We observe that for both versions of the models, there is not much difference in terms of the values for their Recall, Precision and F1 score metrics. However, we can comment on the fact that the non-augmented model has higher recall and precision values whereas it performed worse in terms of accuracy i.e. it had lower accuracy value. The non-augmented model seems to be performing better at avoiding false positives and is better at capturing true positives and minimizing false negatives but struggling with overall correct classification. Please note that though the numbers we see don't indicate that, it is because of the data augmentation and its effect on the class label distribution in the validation set.

```
Metrics for Non-Augmented Model
Confusion Matrix for k=5:
[[279 134]
 [ 84 303]]

Metrics for k=5: Recall: 0.7292452653114851, Precision: 0.7309794428579534, F1 Score: 0.7272545290761685

Metrics for Augmented Model
Confusion Matrix for k=9:
[[281 132]
 [ 86 301]]

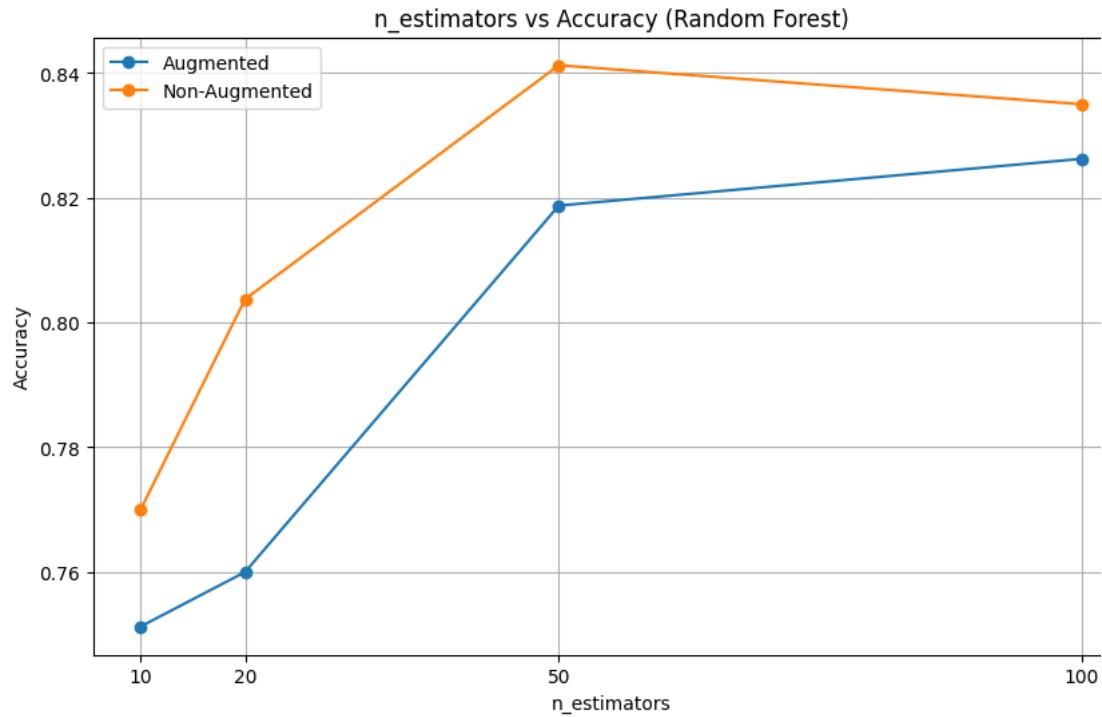
Metrics for k=9: Recall: 0.7290825934893732, Precision: 0.7304088452026606, F1 Score: 0.7273295809881175
```

Figure 7: Metrics for Augmented and Non-Augmented KNN

## Random Forest

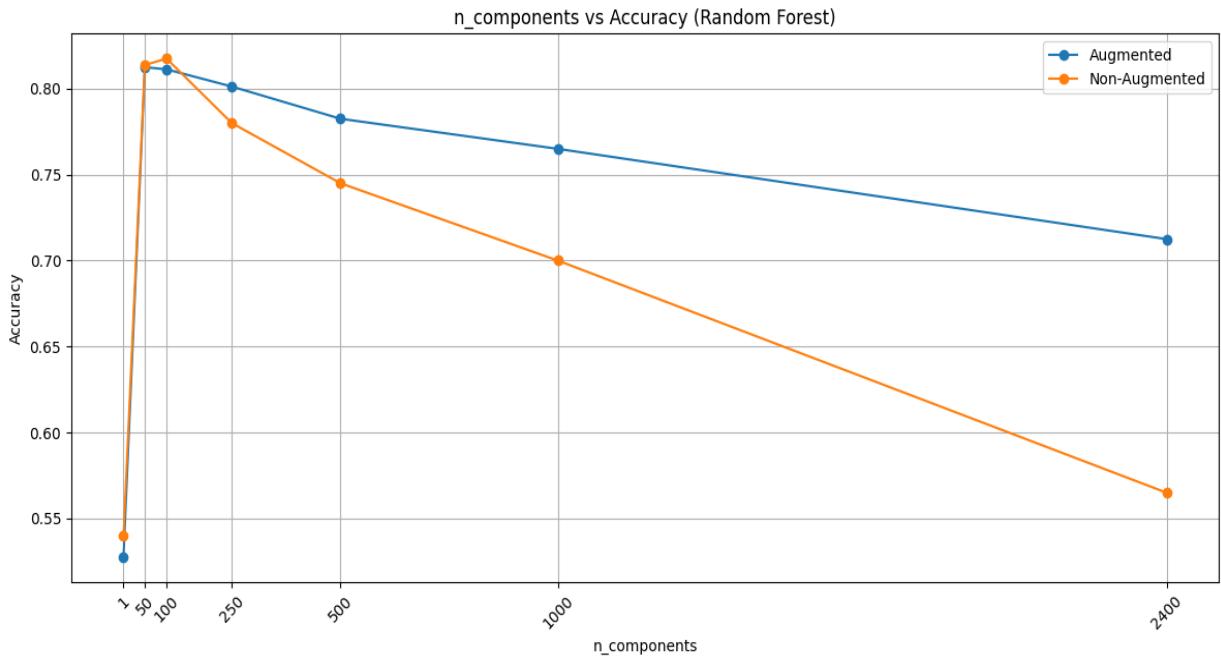
As described in previous sections in detail, the Random Forest algorithm works by creating several decision trees to classify data where the decision trees are denoted with the parameter n\_estimators in scikit-learn library. We have experimented with the Random Forest algorithm with the following range of numbers: (10, 20, 50, 100, 150). These experiments were done

with an augmented dataset and non-augmented dataset and the results of the accuracies have been plotted to interpret the results. Besides this, the best n\_estimator values for the augmented dataset and the non-augmented dataset are used to apply PCA with n different principal components. The results of this comparison have also been plotted to comment on the effect of PCA on Random Forest. Besides these plots, we have also included other performance metrics for the best augmented and the best non-augmented model.



*Figure 8: Number of Estimators vs Accuracy for Augmented and Non-Augmented Random Forest*

The above figure is the graph of accuracy vs n\_estimators which are the number of decision trees used in implementing the algorithm. For the non-augmented graph we can see that as the estimator value increases from 10 to 50, the accuracy level goes up and decreases at estimator = 100. It was expected for the accuracy to rise as the number of decision trees increased which means that many decision trees are reasonable to explain the data. It can also be seen that it decreases from an estimator , this could be related to the fact that an overfitting in decision trees causes a memorization of the data. When we look at the augmented dataset, we can see that the accuracy level goes up as the estimator values increase which is an expected result. The reason for the augmented dataset to have less accuracy then the non-augmented dataset can be closely linked to the noise increasing features used in data augmenting which were: random Gaussian blur, horizontal flips and random noise.



*Figure 9: PCA Analysis of Augmented and Non-Augmented Random Forest*

The above figure is the graph of the best models of the augmented and non-augmented datasets trained with Random Forest obtained from above and used for determining the best principal component number for applying PCA. Therefore, the values of the n\_estimators used for this section obtained from the earlier experiments are: n\_estimator = 50 for the non-augmented, n\_estimator = 100 for the augmented dataset. The experiments are done with component values in the range: ( 1, 50, 100, 250, 500, 1000, 2400). The best component value for the augmented dataset was obtained to be 50 components whereas for the non-augmented set it was obtained to be 100 components which are close enough considering the number of features in the images of the dataset. The increase of these components to a certain component value indicates that these components make up a huge portion of the variance of the data. The decreases in accuracy indicate the noisy features which are a huge portion of the features interpreted from the values in the graph.

```

For Non-Augmented Model
Results for Random Forest with n_estimators=50:
Confusion Matrix:
[[353 60]
 [ 67 320]]
Recall: 0.8407974673248619
Precision: 0.8412907268170426
F1 Score: 0.8409794165384537

For Augmented Model
Results for Random Forest with n_estimators=100:
Confusion Matrix:
[[342 71]
 [ 75 312]]
Recall: 0.8171443587289073
Precision: 0.8173826474068786
F1 Score: 0.8172429979658895

```

*Figure 10: Metrics for Augmented and Non-Augmented Random Forest*

The above figure includes the evaluation metrics other than the accuracy metric for the Random Forest algorithm for the best augmented and the best non augmented models. For both of the models, we can say that the true positives and true negatives are mostly correctly predicted from the confusion matrix. The high recall scores indicate that the actual positives were mostly correctly identified for both models but better in the non augmented model. The high precision score indicates that the positive guesses were mostly actually positives but again, higher in the non-augmented model. The F1 scores indicate the balance between precision and recall which are sufficient to say that the balance is mostly established between the recall and precision scores of the models.

## Support Vector Machine (SVM)

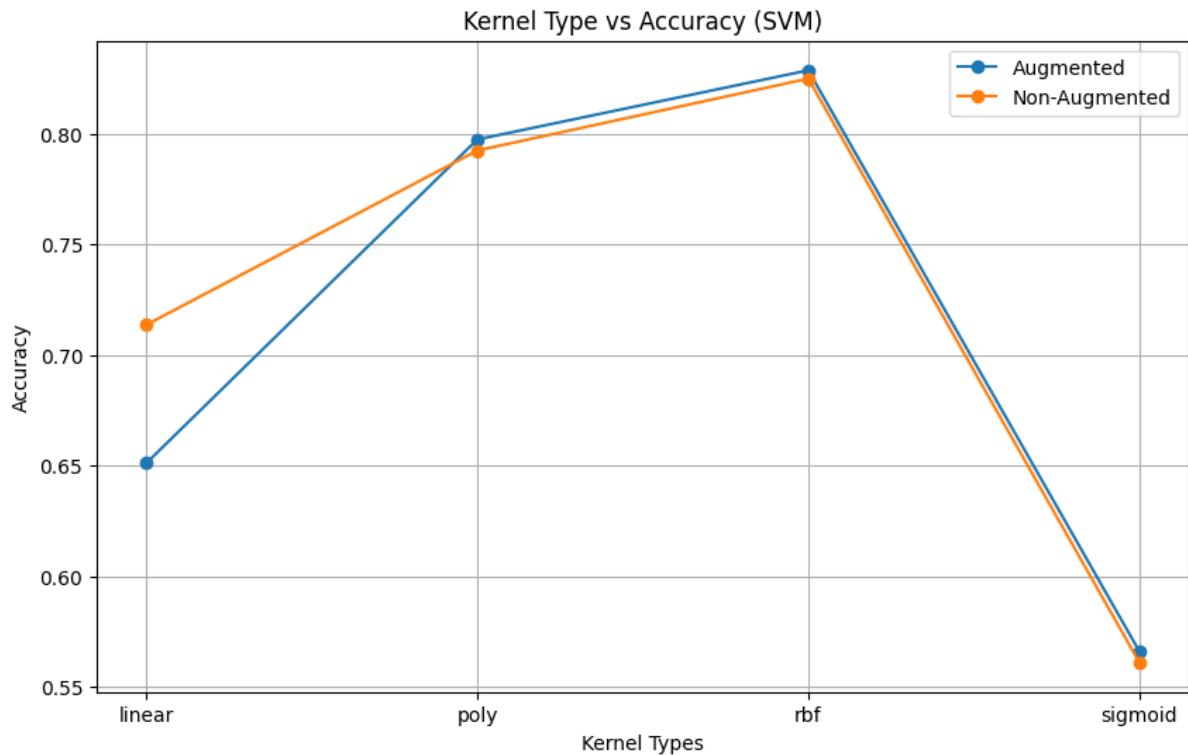
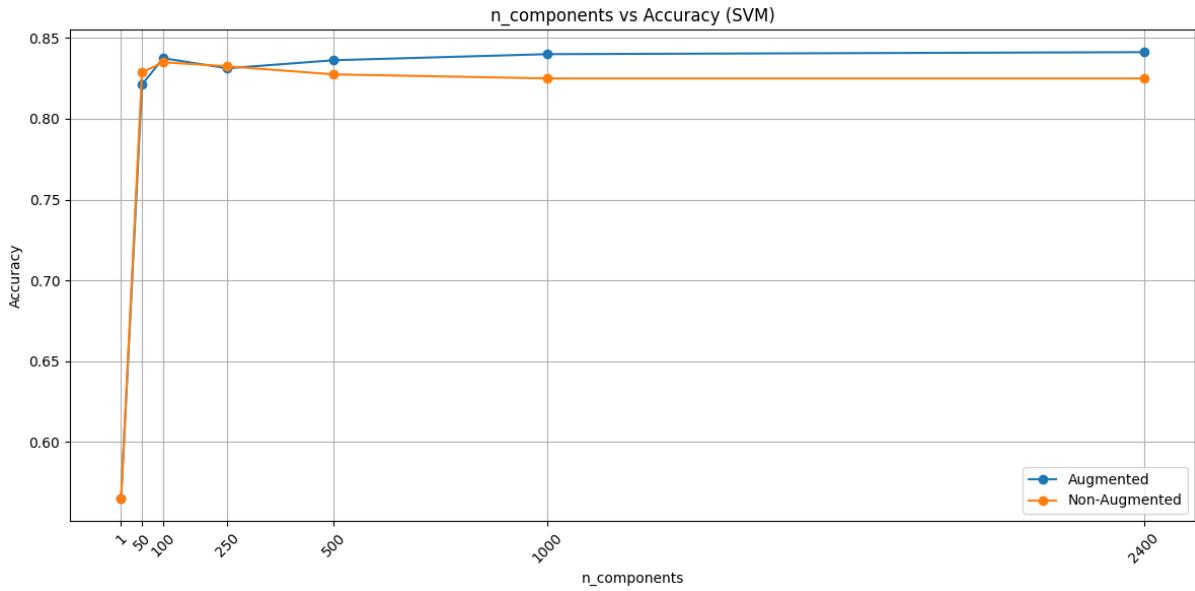


Figure 11: Kernel Type vs Accuracy for Augmented and Non-Augmented SVM

It can be seen in the figure that SVM performs differently with different Kernel types. SVM with the kernel type `rbf` performs the best among the kernel types which is the kernel type that is best for non-linear data. This indicates that our dataset is non-linear and it performs the best under conditions designed for non-linear data. SVM performs the worst with sigmoid kernel type which is the kernel type used for neural networks which indicates that our dataset is not suitable for neural network related components. Furthermore, the second worst performing kernel type is linear type which indicates that our dataset is non-linear data. `Poly` kernel type is the second best performing after `rbf` which is the second best kernel type among these kernel types used for non-linear dataset. This result is on par with our inference that our dataset is non-linear. All in all, the SVM with kernel type `rbf` performed the best due to the non-linearity of our dataset. Other than the kernel types, augmented SVM generally performed better than the non-augmented SVM in different kernel types except for the kernel type `linear`. This means that applying augmentation to the dataset increases the non-linearity of the dataset by increasing the size and variation in the dataset.



*Figure 12: PCA Analysis of Augmented and Non-Augmented SVM*

We also applied PCA to SVM to extract the optimal number of features for the optimal model for augmented and non-augmented versions of SVM. In the figure above, it can be seen that the highest accuracy achieved in 100 components both for augmented and non-augmented versions of SVM. After 100 components, both of the variations reached a point of saturation. This means that 100 components are enough to capture the large portion of variance in the dataset. After 100 components, both of the variations' accuracies decrease which means that noise is added to the features and do not really contribute to the portion of the variance captured in the dataset. Furthermore, augmented SVM performed better than non-augmented SVM overall as augmentation increases the data diversity and contributes to a richer feature space for PCA.

```

Metrics for Non-Augmented Model
Results for SVM with kernel='rbf':
Confusion Matrix:
[[336  77]
 [ 63 324]]
Recall: 0.82538431217974
Precision: 0.8250426565166031
F1 Score: 0.8249606161386311

Metrics for Augmented Model
Results for SVM with kernel='rbf':
Confusion Matrix:
[[344  69]
 [ 59 328]]
Recall: 0.8402375008602837
Precision: 0.8398972442199873
F1 Score: 0.839935974389756

```

*Figure 13: Metrics for Augmented and Non-Augmented SVM*

For both augmented and non-augmented SVM, the 'rbf' kernel type is selected and the test accuracy in the figure above is obtained with the test dataset after testing the selected best models. Non-augmented SVM obtained recall, precision and F1 score values of 0.82, 0.82, 0.82 respectively. Augmented SVM obtained recall, precision, and F1 score values of 0.84, 0.839, 0.839 respectively. The augmented SVM outperforms the non-augmented SVM across all metrics. Augmentation has contributed to a higher true positive rate, as indicated by the increased Recall and Precision. Both models have a good recall, but the augmented SVM achieves a slightly higher recall compared to the non-augmented SVM. This implies that the augmented model better captures actual positive instances. Precision for the augmented SVM is slightly higher than for the non-augmented SVM. This indicates that the augmented model is more precise in classifying instances as positive. Furthermore, the F1 score which balances precision and recall is higher for the augmented SVM compared to the non-augmented SVM. This suggests an overall improvement in the model's performance with data augmentation. Increasing the variance and size in the data with data augmentation results in better performance in capturing true positives and classifying with higher accuracies overall according to the results.

## Convolutional Neural Network (CNN)

We have experimented with four different CNN model structures to tune the layers of the CNN model. We conduct our experiments with both the augmented and the non-augmented datasets. For the base CNN model, we added a two-dimensional convolutional layer with 32 filters of size (3,3) and used ReLu as the activation function. After this convolutional layer, we reduced the size with a pooling layer that uses max-pooling of size (2,2). Finally, we

flattened all the data and obtained the model's prediction with a fully connected layer that uses the sigmoid as the activation function. For the other CNN models, we added a combination of convolutional and pooling layers with increasing filter size to our base CNN model. On the last CNN model, we added an extra fully connected layer that reduces the number of nodes to 128 with the ReLu activation function. All of the CNN models can be seen below.

```

cnn_1 = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(64, 64, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(1, activation="sigmoid")
])

cnn_2 = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(64, 64, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation="relu"),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(1, activation="sigmoid")
])

cnn_3 = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(64, 64, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation="relu"),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation="relu"),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(1, activation="sigmoid")
])

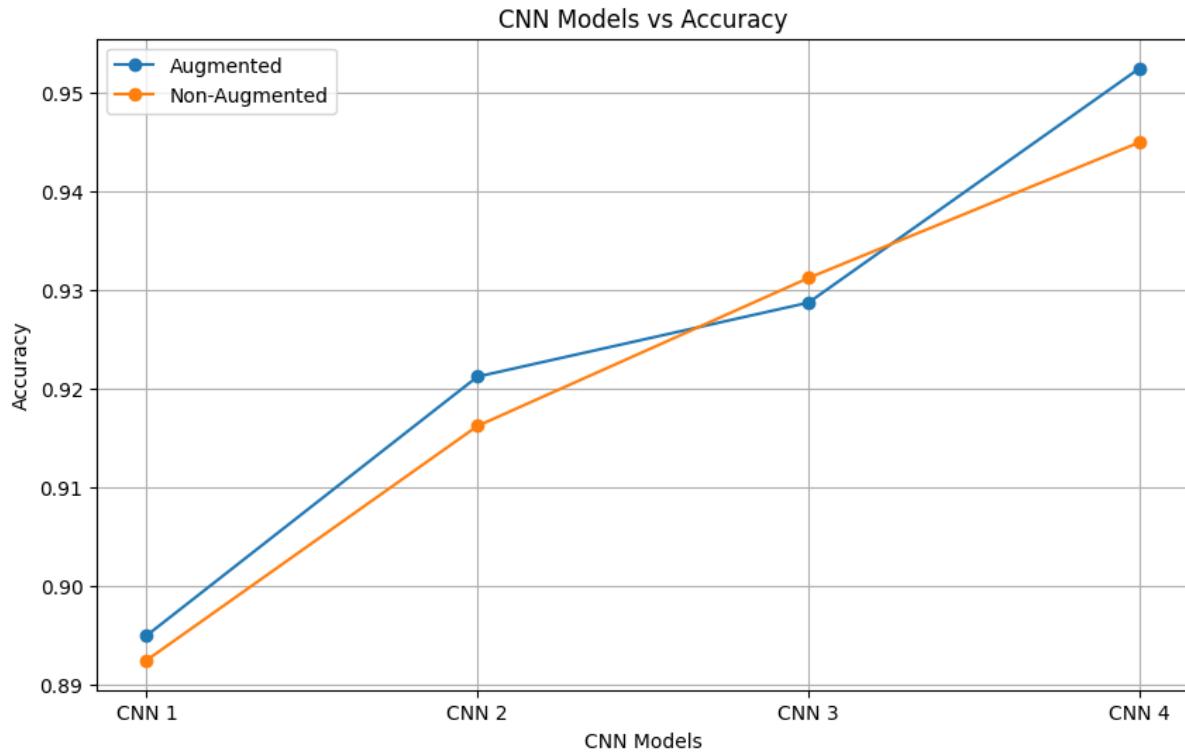
cnn_4 = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(64, 64, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation="relu"),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation="relu"),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation="relu"),
    Dense(1, activation="sigmoid")
])

```

*Figure 14: Validation Accuracy of CNN vs Layer Count*

The graph below illustrates that the validation accuracy of CNN models increases as the layer count increases, irrespective of whether the dataset is augmented or not. This was an expected behavior because lower-level features, such as edges and simple textures, can be captured by

a small number of filters. In contrast, higher-level, more complex, and abstract features may require a more significant number of filters. As a result, by increasing the number of filters with each two-dimensional convolutional layer, we capture more features, increasing validation accuracy. The only difference for CNN models 3 and 4 is the extra fully connected layer in model 4. The graph below shows that this extra layer increased validation accuracy for both augmented and non-augmented train datasets. The additional dense layer allows the model to learn higher-level abstractions of the features extracted by the convolutional layers. This is especially beneficial in image classification tasks where hierarchical feature representations are crucial.



*Figure 15: CNN Models vs Accuracy*

### Evaluation Metrics

A high recall indicates that the model captures most of the positive instances. In the below output, a recall of 0.92 for the non-augmented model means that the model correctly identified 92% of the actual positive instances. Precision measures how accurate the model is when it predicts a positive instance. A high precision of 0.95 for the non-augmented model means that it is correct 95% of the time when the model predicts a positive instance. The F1 score provides a balance between precision and recall. It is especially useful when there is an uneven class distribution. A high F1 score (e.g., 0.94 for the non-augmented model) indicates a good balance between precision and recall. These results indicate that the model trained on the augmented dataset performs better across multiple metrics. This suggests that data augmentation has positively impacted the model's ability to generalize and make accurate predictions on the validation dataset.

```

Metrics for Non-Augmented Model
25/25 [=====] - 0s 14ms/step
Confusion Matrix for model=CNN 4:
[[390 23]
 [ 29 358]]
Metrics for model=CNN 4: Recall: 0.9250645994832042, Precision: 0.9396325459317585, F1 Score: 0.9322916666666666

Metrics for Augmented Model
25/25 [=====] - 1s 25ms/step
Confusion Matrix for model=CNN 4:
[[388 25]
 [ 27 360]]
Metrics for model=CNN 4: Recall: 0.9302325581395349, Precision: 0.935064935064935, F1 Score: 0.9326424870466321

```

Figure 16: Metrics for Augmented and Non-Augmented CNN

## Comparison of Models

After tuning the models for their hyperparameters, we have come up with the best versions of the algorithms. Each model with and without augmentation performed best when:

- Augmented kNN:  $k = 9$
- Non-augmented kNN:  $k = 5$
- Augmented RF:  $n\_estimators = 50$
- Non-augmented RF:  $n\_estimators = 50$
- Augmented SVM:  $\text{kernel} = \text{rbf}$
- Non-augmented SVM:  $\text{kernel} = \text{rbf}$
- Augmented CNN: layering type 4 (see respective section)
- Non-augmented CNN: layering type = 4 (see respective section)

We retrained each model but this time merging the train and validation sets, creating an even bigger training set and finally posted the results and evaluation metrics using the test set.

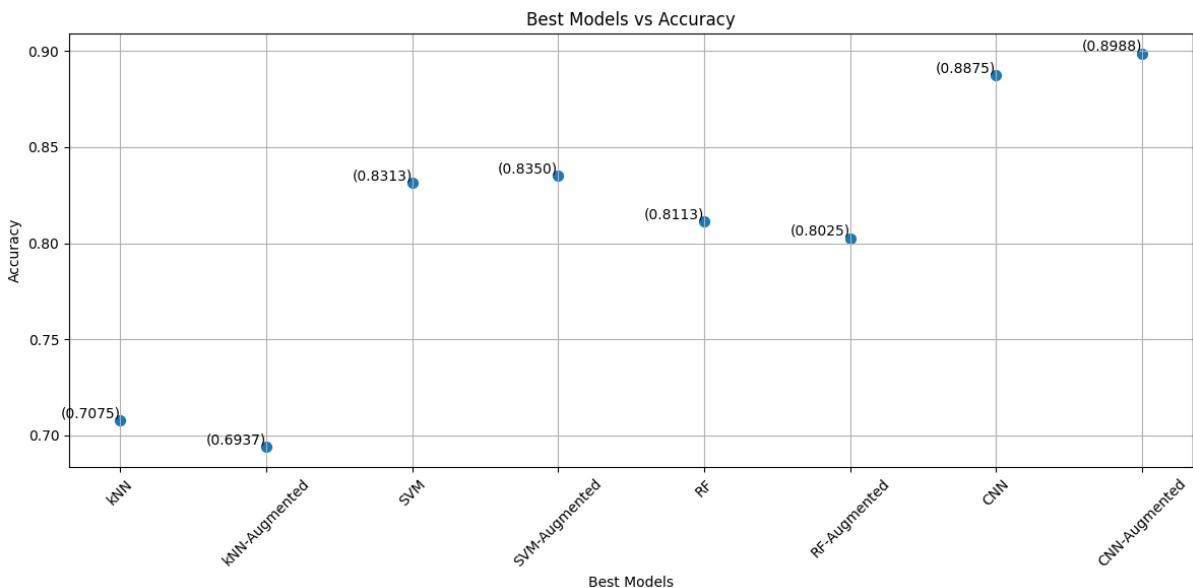


Figure 17: Best Models vs Accuracy

The metric that was used for plotting the above graph is accuracy. We have obtained the accuracies of the best performing models on different parameter optimizations as described above. The final accuracy comparison that can be obtained from the graph is: CNN applied on augmented dataset > CNN applied on non-augmented dataset > SVM applied on augmented dataset > SVM applied on non-augmented dataset > Random Forest applied on non-augmented dataset > Random Forest applied on augmented dataset > KNN applied on non-augmented dataset > KNN applied on augmented dataset. CNN came out to be the best performing model for accuracy metric with layering type = 4. In order to make a thorough comparison, other evaluation metrics, recall - precision - f1 score, that were mentioned in the previous sections should be used to make a better comparison. The evaluation metrics obtained from figures: Figure 7, Figure 10, Figure 13, Figure 16 are given in 3 significant figures for simplification.

Evaluation metrics obtained for best performing KNN model obtained from Figure 7:

For KNN with non-augmented dataset:

Recall = 0.729 , Precision = 0.731, F1 Score = 0.727

For KNN with augmented dataset:

Recall = 0.730 , Precision = 0.730, F1 Score = 0.727

Evaluation metrics obtained for best performing Random Forest model obtained from Figure 10:

For Random Forest with non-augmented dataset:

Recall = 0.841 , Precision = 0.841 , F1 Score = 0.841

For Random Forest with augmented dataset:

Recall = 0.817 , Precision = 0.817, F1 Score = 0.817

Evaluation metrics obtained for best performing SVM model obtained from Figure 13:

For SVM with non-augmented dataset:

Recall = 0.825 , Precision = 0.825 , F1 Score = 0.824

For SVM with augmented dataset:

Recall = 0.840 , Precision = 0.840, F1 Score = 0.840

Evaluation metrics obtained for best performing CNN model obtained from Figure 16:

For CNN with non-augmented dataset:

Recall = 0.925 , Precision = 0.940 , F1 Score = 0.932

For CNN with augmented dataset:

Recall = 0.930, Precision = 0.935, F1 Score = 0.933

Comparing the results for each evaluation metric separately, the recall metric essentially provides the metric of how well the model performs to correctly label the actual positives by calculating the ratio of true positives over actual positives. We can see that the comparison of the best performing models in terms of recall is close to the accuracy comparisons as the ordering is very similar besides only KNN with augmented gives a better result than KNN

with non augmented which is a relatively small difference compared to other recall differences between different models.

Comparing the results for precision, which is the metric that provides how accurate the positive guesses were by simply calculating the ratio of true positives over all positive guesses, we can obtain a similar ordering to accuracy with one major difference: non-augmented CNN gives a higher precision than augmented CNN. Therefore, the best performing model for this metric came out to be CNN applied on the augmented dataset.

Comparing the results for F1 score, which indicates a comparison point for the balance of precision and recall in model performances, we can, again, obtain a quite similar ordering to accuracy comparison. The only difference is that the KNN with augmented and non-augmented datasets yielded the same F1 score. Besides this the ordering is obtained to be the same.

The four evaluation metrics are used to determine the ultimate ordering of the models to get the best performing model. The comparison is done with the following logic: the average orders of each model based on four metrics are calculated and the final ordering is done based on the logic least order goes to top. The following best performing models comparison is obtained where CNN with augmented dataset is the final verdict for best performing model:

1. CNN with augmented dataset
2. CNN with non-augmented dataset
3. SVM with augmented dataset
4. SVM with non-augmented dataset
5. Random Forest with non-augmented dataset
6. Random Forest with augmented dataset
7. KNN with non-augmented dataset
8. KNN with augmented dataset

## Conclusion

In conclusion, the main objective of this project was to solve a binary classification problem using a car-bike dataset, by employing 4 different machine learning algorithms. We chose to use kNN, Random Forest and SVM for our regular learning models and CNN as our deep learning method. In addition, we aimed to assess the applicability and the success of data augmentation to increase the size of our limited capacity training dataset as well as analyze the outcome of PCA and how our models behave for different numbers of Principal Components. We have observed an increase in accuracy for almost all of our trained models when the dataset is augmented using the random flips, random blurs and random noises, which we can infer that the augmentation could successfully represent the test and validation sets. In addition, we have observed highest level of accuracy when PCA is applied to the dataset for 50 and 100 Principal Components, which we inferred that number of Principal

Components carried the highest level of reasonable variation and after them we have observed a decrease in accuracy which meant Principal Components started to carry noisy features as well. Then, we proceeded to tune the hyperparameters of our models using the validation set to validate the accuracies we obtained. We have chosen the best performing hyperparameters and carried the augmented and non-augmented versions of the models to the next stage of training where we merge the training and validation sets to have a broader learning possibility and plot the accuracies to find the best performing model in terms of accuracy. Lastly, we have discussed the tradeoffs between the models in the last stage, compared and contrasted them on their evaluation metrics other than accuracy such as their precision and recall values.

## Appendix

### Contribution of Each Team Member

## References

- [1] DeepNets, “Car Vs Bike Classification Dataset,” Kaggle, <https://www.kaggle.com/datasets/utkarshsaxenadn/car-vs-bike-classification-dataset>.
- [2] J. D. Christy Bieber, “Motorcycle Accident Statistics & Numbers for 2023,” Forbes, <https://www.forbes.com/advisor/legal/motorcycle-accident-statistics/> (accessed Dec. 24, 2023).
- [3] S. Gupta, “Comparing the performance of machine learning algorithms using estimated accuracy,” Science Direct, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2665917422000666>
- [4] “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,” Towards Data Science, 2018. [Online]. Available: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- [5] “colab.google,” colab.google. <https://colab.google/>
- [6] “What is the K-nearest neighbors algorithm?,” IBM, <https://www.ibm.com/topics/knn>
- [7] S. E. R, “Understand random forest algorithms with examples (updated 2023),” Analytics Vidhya, <https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/> (accessed Nov. 22, 2023).
- [8] M. Schott, “Random Forest Algorithm for Machine Learning,” Medium, <https://medium.com/capital-one-tech/random-forest-algorithm-for-machine-learning-c4b2c8cc9feb> (accessed Nov. 25, 2023).
- [9] An Idiot’s Guide to support Vector Machines (svms) - MIT, <https://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf> (accessed Nov. 25, 2023).
- [10] Wikipedia contributors, “Rectifier (neural networks),” Wikipedia, [https://en.wikipedia.org/wiki/Rectifier\\_\(neural\\_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)) (accessed Nov. 25, 2023).
- [11] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” Insights Imaging, vol. 9, no. 4, pp. 611–629, 2018 (accessed Nov. 25, 2023).