1. Introduction

This project centres on developing a machine learning model to solve a binary image classification problem: distinguishing between images of cats and dogs. The challenge, though commonly straightforward for humans and the animals themselves, presents unique difficulties for an automated system due to the significant variation in poses and appearances of the animals within the images.

The ultimate goal of this project is to create an efficient and accurate classifier that can reliably determine whether new, unseen images contain a cat or a dog, thus pushing the boundaries of what machine learning algorithms can achieve in terms of image recognition and classification.

2. Pre-Processing data

Below are the steps that are taken to perform the pre processing the provided 1000 images of cats and dogs for training and 100 images of them for testing purpose.

1. Loading and Filtering Images

The code iterates over files within a specified directory, selectively processing files that match common image formats and are within a set limit to keep the dataset manageable. This selective approach ensures that only relevant images are loaded for further processing. Here we store the train and test data separately as the trained data have the file name which as the label to identify if the image is of 'dog' or 'cat'.

2. Resizing Images

When images are loaded, each is resized to a standardized dimension of 350x350 pixels. This uniformity in image size is crucial as it ensures that all input data is consistent, which helps in maintaining the model's learning effectiveness. Variations in the dimensions of images could potentially disrupt the model's ability to process and learn from the data accurately, thereby affecting the overall performance of the model.

3. Normalization

The pixel values of each image are scaled down to a range between 0 and 1 through normalization. This step is essential for machine learning models because it helps reduce skewness in the distribution of pixel values. By normalizing the data, the model benefits from smoother gradients, which enhances the efficiency of gradient descent during training. This leads to better convergence, allowing the model to learn more effectively and improve its performance on image-related tasks.

4. Categorization

During image processing, each image is categorized based on specific keywords found in its filenames, a method critical for supervised learning models. In supervised learning, it's essential that every input (image) is paired with an output label (category), which directs the learning process. The graph below illustrates the distribution of images categorized into 'dogs' and 'cats' based on their filenames within the dataset. This categorization helps in accurately training the model by providing clear, distinct labels, which are necessary for the model to learn and differentiate between the categories effectively.

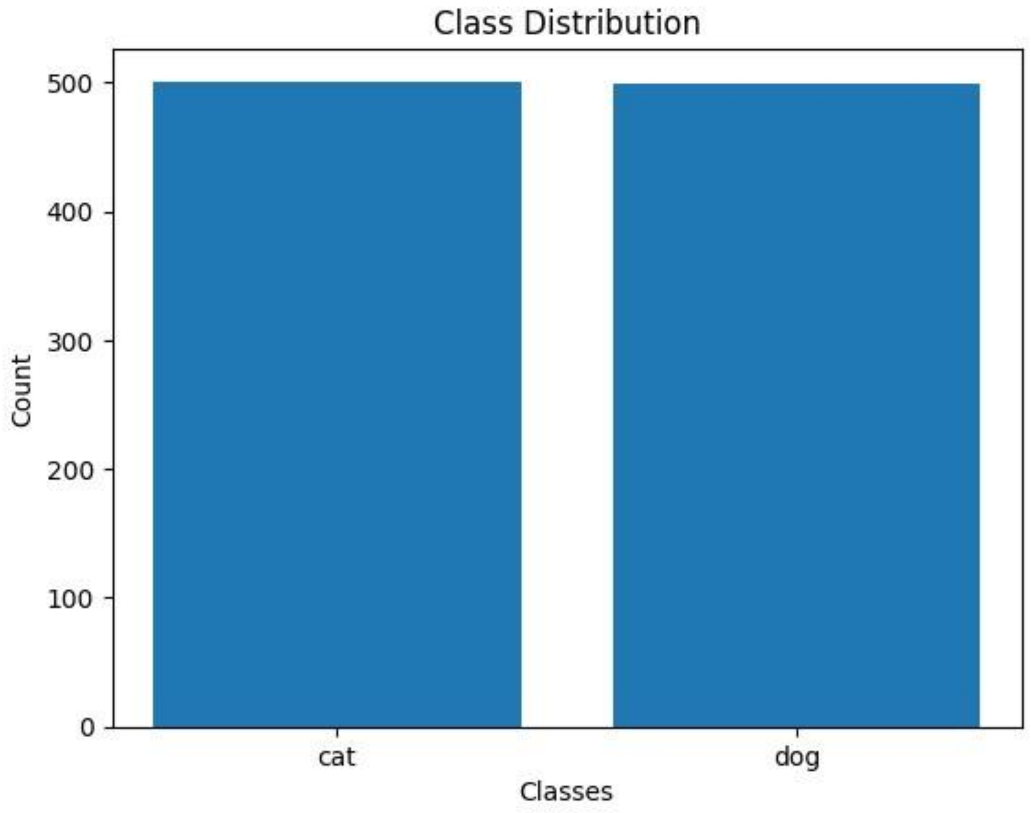


Figure 1. Distribution of cat and dog in training data

5. PCA

Principal Component Analysis (PCA) as part of the data preparation process for an image classification task. Initially, PCA is configured to reduce the dataset to 50 principal components, focusing on capturing the most significant variance within the data. This reduction is first carried out on the training data, where the PCA model learns the necessary transformations to reduce dimensionality while preserving critical information. Subsequently, both the training and validation datasets are transformed based on this model, effectively reducing their dimensions to the top 50 components. This step simplifies the data structure, making the machine learning process more efficient and ensuring consistency between the training and validation phases. Applying PCA in this manner helps streamline data processing, enhances training efficiency, and supports more accurate model evaluation.

3. Training Phase

In this approach, we use two models, SVM (Support Vector Machine) and CNN (Convolutional Neural Network), to train on the provided dataset for the purpose of classifying and predicting whether an image depicts a dog or a cat. Below are the steps and results that were got in this process:

1. **Label Encoding and Data split for training**: The labels for the training data, which are presumably strings indicating 'cat' or 'dog', are converted into a numerical format using a

LabelEncoder. This conversion is necessary because most machine learning algorithms, particularly those implemented in libraries like scikit-learn, require numerical input. The fit_transform method not only fits the label encoder to the labels but also transforms them, resulting in a new array where each label is represented as a unique integer.

The dataset is divided into training and validation subsets using the train_test_split function. This function segregates the images (train_images) and their corresponding encoded labels (train_labels_encoded) into training (80% of the data) and validation (20% of the data) sets. The test_size=0.2 parameter specifies that 20% of the dataset should be reserved for validation. The random_state=42 ensures that the split is reproducible; the same random split will occur each time the code is run, which is helpful for debugging and comparing model performance across different runs.

2. **Model**: In this we are discussing about the performance of each model for the provided data set.

- SVM Model: For the SVM model we are using PCA training data for better optimisation and below are the details regarding the performance of it.
- Validation Accuracy: The SVM model achieves a validation accuracy of 51%, indicating that it correctly predicts the category (dog or cat) for slightly more than half of the images in the validation set. This level of accuracy suggests that the model performs only slightly better than a random guess in this binary classification task.
- Precision: The precision of 53.26% indicates that when the model predicts an image as belonging to a particular class (dog or cat), it is correct about 53.26% of the time. This metric is particularly useful when the cost of a false positive is high.
- Recall: The recall of 47.12% shows that the model successfully identifies 47.12% of all actual positives (either cats or dogs) correctly. This is an important measure when the cost of missing a true positive is critical.
- F1 Score: An F1 score of 50% is the harmonic mean of precision and recall, suggesting a balance between the precision and recall in the model. This score is helpful in cases where it's crucial to maintain a balance between precision and recall.
- ROC AUC Score: The ROC AUC score of approximately 53.99% indicates a very slight improvement over random guessing. The ROC curve (Receiver Operating Characteristic curve) plots the true positive rate against the false positive rate at various threshold settings. The closer this score is to 50%, the closer the model is to a random guess; a score closer to 100% indicates an excellent model performance.

Below is the graphical representation of confusion matrix of the SVM model.

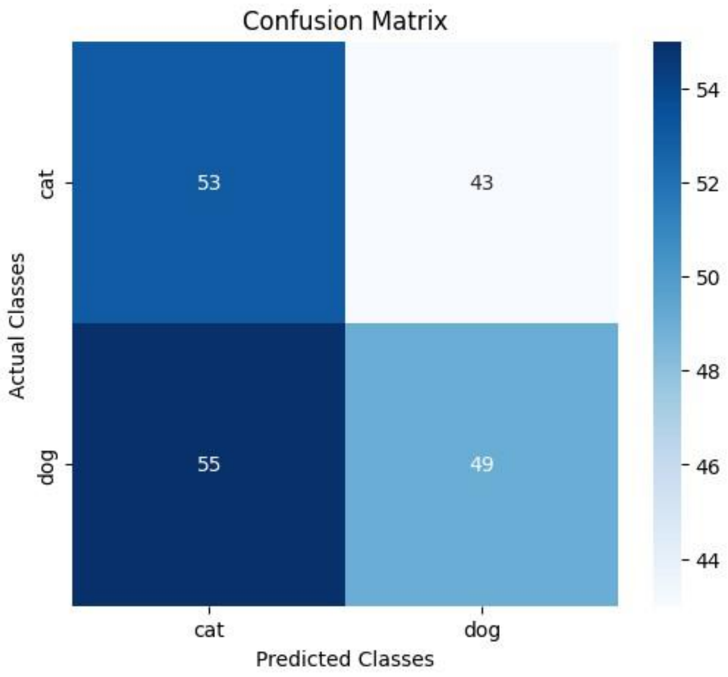


Figure 2. Confusion matrix for SVM model

- CNN: Convolutional Neural Network (CNN) designed for binary image classification, distinguishing between cats and dogs. It consists of multiple convolutional layers with ReLU activation, interleaved with MaxPooling layers to extract and downsample features. A Flatten layer transitions from convolutional to dense layers, introducing a Dropout layer to combat overfitting by randomly disabling neurons during training. The network ends with a sigmoid-activated output layer for binary prediction. Compiled with Adam optimizer and binary crossentropy loss, the model trains over ten epochs, using accuracy as a metric and validating against a separate dataset to ensure generalizability. Below are the performance details of the model:

The training progression of the Convolutional Neural Network (CNN) for image classification over ten epochs shows significant improvement in accuracy on the training set but less stable performance on the validation set. Here's a detailed interpretation:

The training progression of a Convolutional Neural Network for image classification shows a sharp improvement in accuracy from 50.90% to 98.42% over ten epochs, reflecting the model's enhanced capability to recognize training data patterns. However, the validation accuracy tells a different story, starting at 52.00%, peaking at 64.00%, and then dropping to 60.50% by the tenth epoch. This pattern, combined with increasing validation loss after the fifth epoch, suggests the model is overfitting the training data—learning details and noise that do not generalize to unseen data. The final validation on a separate test set confirms a consistent accuracy around 60.50%, underscoring the need for strategies like increased dropout, data augmentation, or regularization to improve the model's generalization capabilities and manage overfitting.

The Receiver Operating Characteristic (ROC) curve in the graph evaluates the performance of a binary classification model. The curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. An area under the curve (AUC) of

0.67 indicates that the model has moderate discriminatory ability, performing significantly better than random chance (indicated by the dashed line representing an AUC of 0.5). The graph shows the model's effectiveness at distinguishing between the classes (e.g., cat vs. dog), with higher TPR values for most of the FPR spectrum, suggesting reasonable prediction capability. From below graph you can see this information

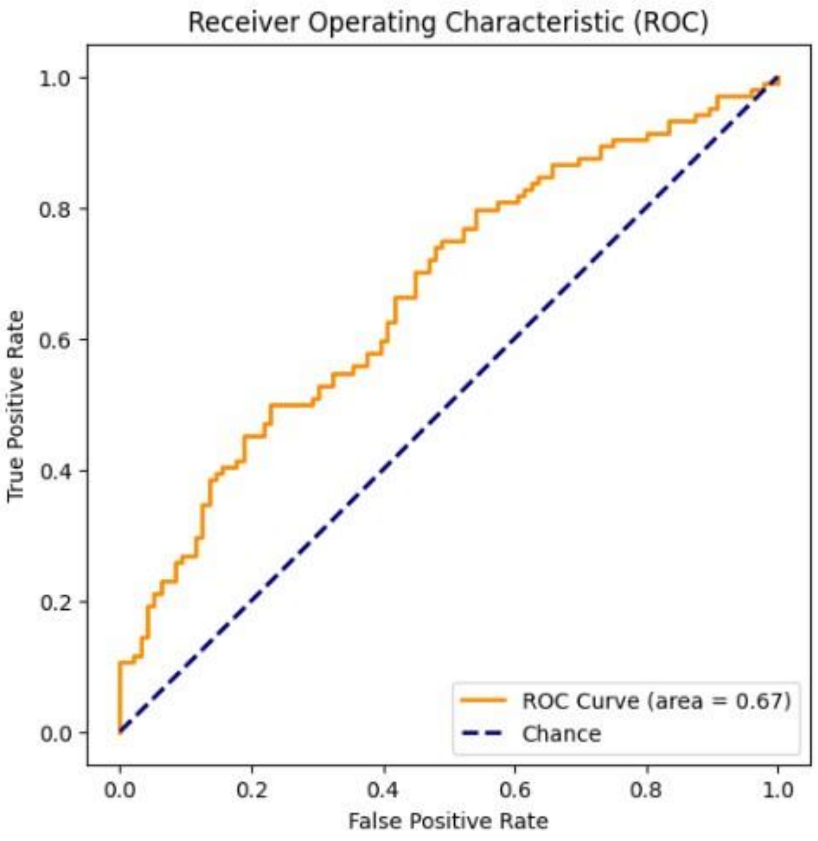


Figure 3. ROC curve of CNN model

So, by comparing both the models we can say that the CNN model has performed better compared to SVM model.

Thus, we are performing the optimisation of the CNN model for better accuracy.

4. Optimisation Phase

the implementation of a hyperparameter tuning process for optimizing a Convolutional Neural Network (CNN) used in binary image classification. This process employs a randomized search strategy using the **kt.RandomSearch** tuner from the Keras Tuner library. Here's how the hyperparameter tuning is structured:

Model Definition with Tuneable Parameters: The build_model function defines the architecture of the CNN with several hyperparameters set to be tuned, including the number of filters and kernel sizes for convolutional layers, the number of units in the dense layer, the dropout rate, and the learning rate of the Adam optimizer. These parameters can significantly affect model performance, as they control aspects like model complexity, capacity to learn from data, and how effectively the model can generalize.

Tuning Setup: A RandomSearch tuner is configured to optimize for validation accuracy (val_accuracy). The search will test three different configurations (max_trials=3), each running for a single execution (executions_per_trial=1). This approach allows the tuner to explore a diverse set of configurations randomly and efficiently identify a promising model configuration.

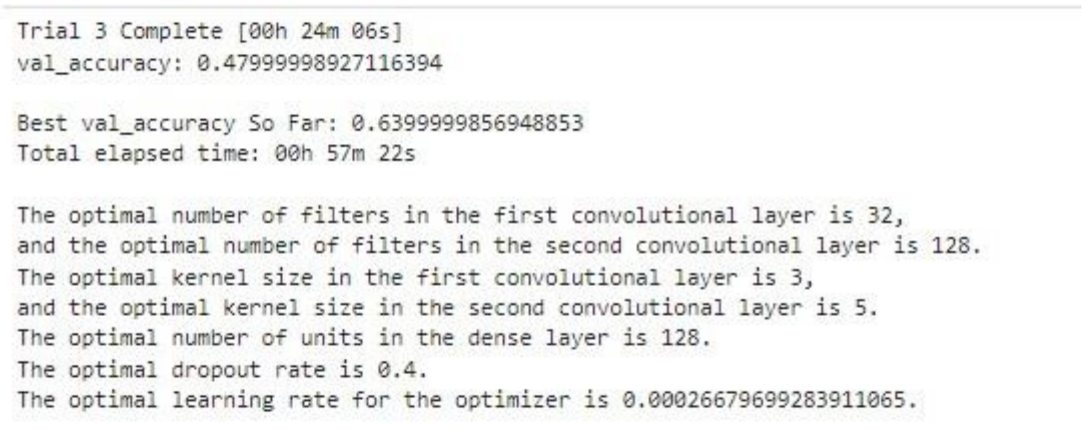
Hyperparameter Search: The search process involves training temporary models on the training data (X_train, y_train) across a specified number of epochs while validating on a separate validation dataset (X_val, y_val). This step is crucial as it evaluates how different hyperparameter settings perform under identical training conditions.

Optimal Hyperparameters: After the search, the best performing hyperparameters are extracted. These include the optimal settings for filters, kernel sizes, dense units, dropout rate, and learning rate, which are determined based on their contribution to achieving the highest validation accuracy.

Model Rebuilding and Training: The optimal hyperparameters are used to build the final model. This model is then trained for an increased number of epochs to thoroughly learn from the full training dataset, using the previously validated settings to potentially enhance accuracy and robustness on unseen data.

This process of hyperparameter tuning is essential for refining the CNN's ability to accurately classify images as either cats or dogs. By systematically exploring different configurations, the model is tailored to best fit the characteristics of the specific dataset, improving its performance and reliability in practical applications.

Below is the performance of the best hyperparameter.



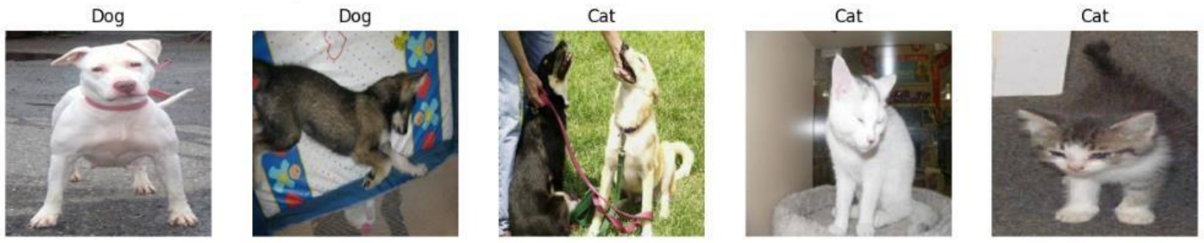
```
Trial 3 Complete [00h 24m 06s]
val_accuracy: 0.47999998927116394

Best val_accuracy So Far: 0.6399999856948853
Total elapsed time: 00h 57m 22s

The optimal number of filters in the first convolutional layer is 32,
and the optimal number of filters in the second convolutional layer is 128.
The optimal kernel size in the first convolutional layer is 3,
and the optimal kernel size in the second convolutional layer is 5.
The optimal number of units in the dense layer is 128.
The optimal dropout rate is 0.4.
The optimal learning rate for the optimizer is 0.0002667969928391106 5.
```

So, the accuracy after increased after hyperparameter from 60% to 64%. Below

is the prediction made



From the above prediction we can see that the model as predicted 4 out of 5 images correctly, which shows a good accuracy.

Note: Given with the better machine we can perform a optimisation to get better results.