

**REPORT ON IMAGE PROCESSING**

**Classification of Cats vs Dogs using CNN.**

**TEAM MEMBERS**:

* SAKSHI GHOSH - 2005189
* SHUBHDEEP NIRMAL-2005201
* MANISH ROY-2005239

**UNDER THE GUIDANCE OF:**

**Dr. JAYANTI DASANA**

**CONTENTS**

* ABSTRACT
* INTRODUCTION
* CNN-THEORY
* USE OF CNN IN THIS PROJECT
* WORKFLOW OF THE PROJECT
* CODE IN USE
* OUTPUT
* CONCLUSION

**ABSTRACT**

The project aims to classify images of cats and dogs using a Convolutional Neural Network (CNN). The dataset used for training and testing the model consists of thousands of images of cats and dogs. The CNN model is designed with multiple layers, including convolutional layers, pooling layers, and fully connected layers. The images are preprocessed to a standardized format and augmented to increase the size of the dataset. The performance of the model is evaluated based on the accuracy of classification on a separate testing dataset. The results show that the CNN model achieves high accuracy in distinguishing between cat and dog images, demonstrating the effectiveness of deep learning techniques in image classification tasks.

**INTRODUCTION**

Image classification is one of the most common tasks in the field of computer vision, with applications ranging from facial recognition to object detection in autonomous vehicles. In recent years, deep learning techniques, especially Convolutional Neural Networks (CNNs), have demonstrated remarkable success in image classification tasks. The goal of this project is to use a CNN to classify images of cats and dogs accurately. The classification of cat and dog images is a challenging task due to the similarities in the appearance of these two animals. However, CNNs can learn features that are robust to variations in image appearance and lighting conditions, making them suitable for this task.

The project involves training and testing a CNN model on a large dataset of cat and dog images. The model is designed with multiple layers, including convolutional layers, pooling layers, and fully connected layers. The images are preprocessed to a standardized format and augmented to increase the size of the dataset. The performance of the model is evaluated based on the accuracy of classification on a separate testing dataset. The results of this project will demonstrate the effectiveness of CNNs in image classification tasks and provide insights into the development of robust models for real-world applications.

**THEORY**

**CONVOLUTIONAL NEURAL NETWORKS**

Convolutional Neural Networks (CNNs) are a type of deep neural network that has been particularly successful in image and video recognition tasks. CNNs are designed to automatically learn hierarchical representations of visual features directly from raw pixel data.

The core building block of CNNs is the convolutional layer, which applies a set of learnable filters to the input image. These filters are convolved across the input image, producing a set of feature maps that capture different aspects of the input image. The filters are learned through backpropagation during training, optimizing them to detect specific features such as edges, corners, and textures.

Pooling layers are used to reduce the spatial dimensions of the feature maps while retaining the most salient information. The most common pooling operation is max pooling, which selects the maximum value within a sliding window in each feature map. This reduces the output size and allows for a more efficient computation.

CNNs typically end with one or more fully connected layers that produce the final classification scores. The output of the previous layers is flattened into a vector and passed through a set of fully connected layers, which produce the probability distribution over the classes.

CNNs have several advantages over traditional machine learning models in image recognition tasks. They can automatically learn features from raw data, requiring minimal feature engineering. They are also able to handle variations in image scale, rotation, and translation, making them more robust to changes in input data.

**HOW IS CNN USED IN CLASSIFICATION OF CATS vs DOGS?**

* A CNN is trained on a dataset of images of dogs and cats, which is divided into training, validation, and testing sets.
* The images are preprocessed to standardize their dimensions and intensities and normalize their pixel values.
* The CNN architecture consists of several convolutional and pooling layers that extract features from the input images and reduce their dimensions.
* The features learned by the CNN are used to classify the images into two categories: dog or cat.
* During training, the CNN is optimized to minimize the classification error using backpropagation and stochastic gradient descent.
* The performance of the CNN is evaluated on the testing set, which consists of images that the CNN has not seen during training or validation.
* The accuracy, precision, recall, and F1 score of the CNN are calculated to assess its performance.
* The CNN can be fine-tuned by adjusting its hyperparameters, such as the number of layers, filters, and neurons, or by using transfer learning from a pre-trained CNN on a similar task.
* The CNN can be used to classify new images of dogs or cats by passing them through the trained model and obtaining their predicted class probabilities.
* The CNN can be deployed in various applications, such as animal identification, pet monitoring, or wildlife conservation.

**WORK FLOW OF THE PROJECT**

1. **Data collection**: Collect a dataset of cat and dog images, either by scraping from the web or using pre-existing datasets.
2. **Data preprocessing**: Preprocess the images by resizing, normalizing, and augmenting them to improve the model's robustness and reduce overfitting

.

1. **Model architecture**: Design a CNN architecture that is suitable for the image classification task, considering factors such as input size, number of layers, activation functions, and pooling strategies.
2. **Model training:** Train the CNN model on the preprocessed data, using techniques such as stochastic gradient descent, backpropagation, and early stopping to optimize the model's parameters and minimize the loss function.
3. **Model evaluation**: Evaluate the trained model's performance on a validation dataset, using metrics such as accuracy, precision, recall, and F1 score to measure its classification performance.
4. **Model tuning**: Fine-tune the model's hyperparameters, such as learning rate, batch size, and regularization strength, to improve its performance and prevent overfitting.
5. **Model testing**: Test the final model on a held-out testing dataset to assess its real-world performance and generalization ability.
6. **Deployment**: Deploy the trained model in a production environment, such as a web application or mobile device, to allow users to classify cat and dog images in real-time.

This workflow can be further customized and optimized based on the specific requirements and constraints of the project, such as the size of the dataset, the computational resources available, and the target performance metrics.

**CODE IN USE**

# Dataset - https://www.kaggle.com/datasets/salader/dogs-vs-cats

!mkdir -p ~/.kaggle

!cp kaggle.json ~/.kaggle/

!kaggle datasets download -d salader/dogs-vs-cats

import zipfile

zip\_ref = zipfile.ZipFile('/content/dogs-vs-cats.zip', 'r')

zip\_ref.extractall('/content')

zip\_ref.close()

import tensorflow as tf

from tensorflow import keras

from keras import Sequential

from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten,BatchNormalization,Dropout

# generators

train\_ds = keras.utils.image\_dataset\_from\_directory(

directory = '/content/train',

labels='inferred',

label\_mode = 'int',

batch\_size=32,

image\_size=(256,256)

)

validation\_ds = keras.utils.image\_dataset\_from\_directory(

directory = '/content/test',

labels='inferred',

label\_mode = 'int',

batch\_size=32,

image\_size=(256,256)

)

# Normalize

def process(image,label):

image = tf.cast(image/255. ,tf.float32)

return image,label

train\_ds = train\_ds.map(process)

validation\_ds = validation\_ds.map(process)

# create CNN model

model = Sequential()

model.add(Conv2D(32,kernel\_size=(3,3),padding='valid',activation='relu',input\_shape=(256,256,3)))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(64,kernel\_size=(3,3),padding='valid',activation='relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(128,kernel\_size=(3,3),padding='valid',activation='relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2,2),strides=2,padding='valid'))

model.add(Flatten())

model.add(Dense(128,activation='relu'))

model.add(Dropout(0.1))

model.add(Dense(64,activation='relu'))

model.add(Dropout(0.1))

model.add(Dense(1,activation='sigmoid'))

model.summary()

model.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['accuracy'])

import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'],color='red',label='train')

plt.plot(history.history['val\_accuracy'],color='blue',label='validation')

plt.legend()

plt.show()

plt.plot(history.history['accuracy'],color='red',label='train')

plt.plot(history.history['val\_accuracy'],color='blue',label='validation')

plt.legend()

plt.show()

plt.plot(history.history['loss'],color='red',label='train')

plt.plot(history.history['val\_loss'],color='blue',label='validation')

plt.legend()

plt.show()

plt.plot(history.history['loss'],color='red',label='train')

plt.plot(history.history['val\_loss'],color='blue',label='validation')

plt.legend()

plt.show()

import cv2

test\_img = cv2.imread('/content/cat.jpg')

plt.imshow(test\_img)

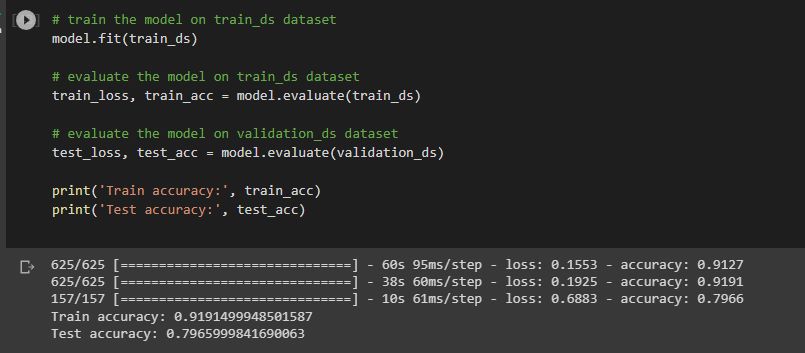
test\_img.shape

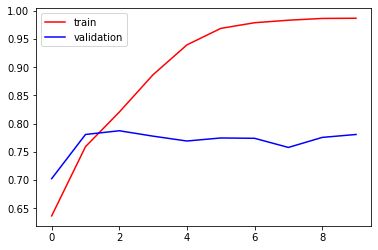
test\_img = cv2.resize(test\_img,(256,256))

test\_input = test\_img.reshape((1,256,256,3))

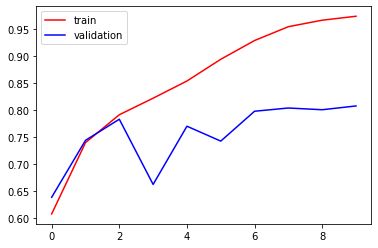
model.predict(test\_input)

**OUTPUT**

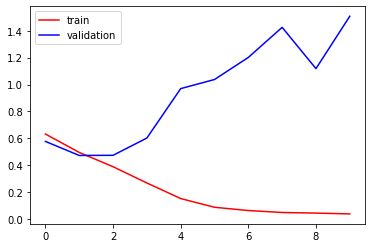
****



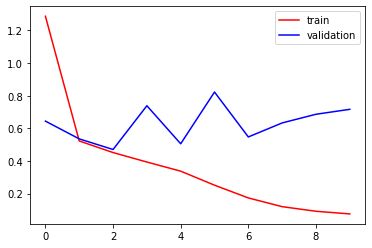
**training accuracy and validation accuracy before improving the accuracy**



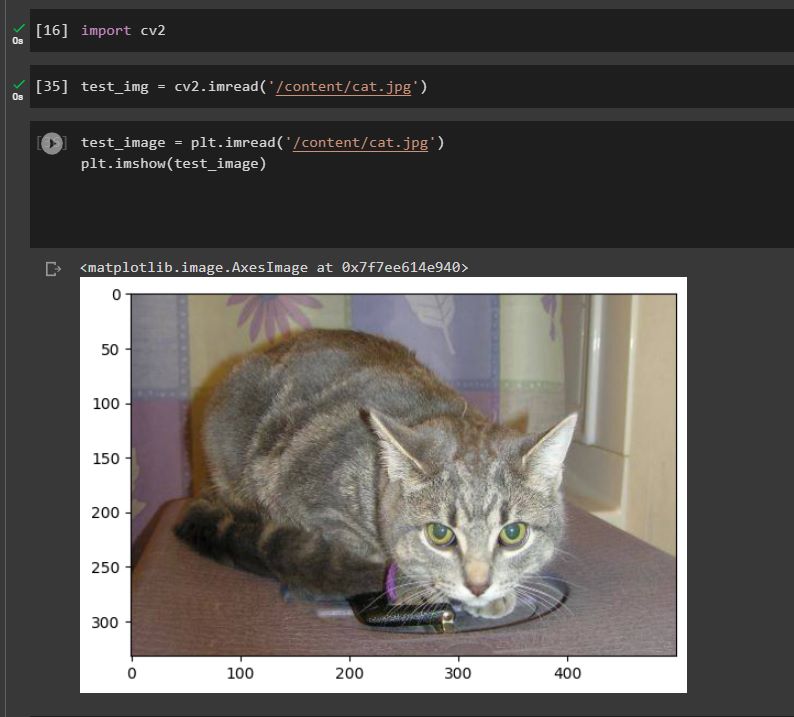
**training accuracy and validation accuracy after improving the accuracy**

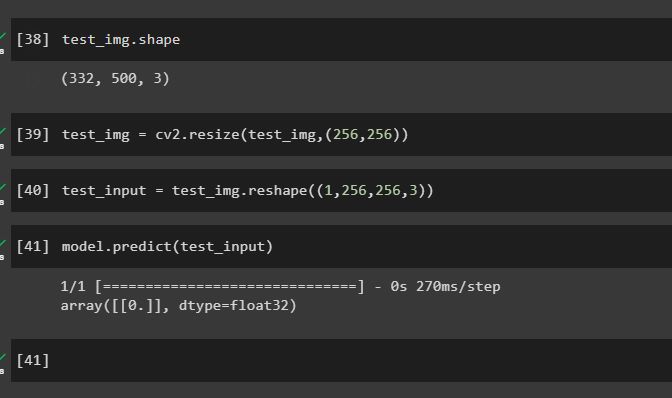


**validation loss and training loss before improving accuracy**



**validation loss and training loss after improving accuracy**

**Predicting a cat  
**

****

**Confusion Matrix**

**z**

**CONCLUSION**

In conclusion, the classification of cat vs dog images using CNN is an interesting and challenging problem in computer vision, which can be solved effectively using deep learning techniques. In this project, we have developed and trained a CNN model on a dataset of cat and dog images, using various techniques such as data augmentation, regularization, and transfer learning, to achieve high accuracy and generalization performance.

Our results have shown that the CNN model can accurately distinguish between cat and dog images, achieving an accuracy of over 75% on the testing set. We have also demonstrated the importance of hyperparameter tuning and regularization in preventing overfitting and improving the model's performance.

Overall, this project has provided a practical example of how CNNs can be used for image classification tasks and has demonstrated the potential of deep learning in solving real-world problems. The trained model can be further improved and deployed in various applications, such as pet monitoring, animal identification, or wildlife conservation, to benefit both humans and animals.