CAT V.s. DOG Classification using CNN

Shreerang Mhatre – 52
Sarvesh Gurav – 44
Tanvi Kariyappa – 56

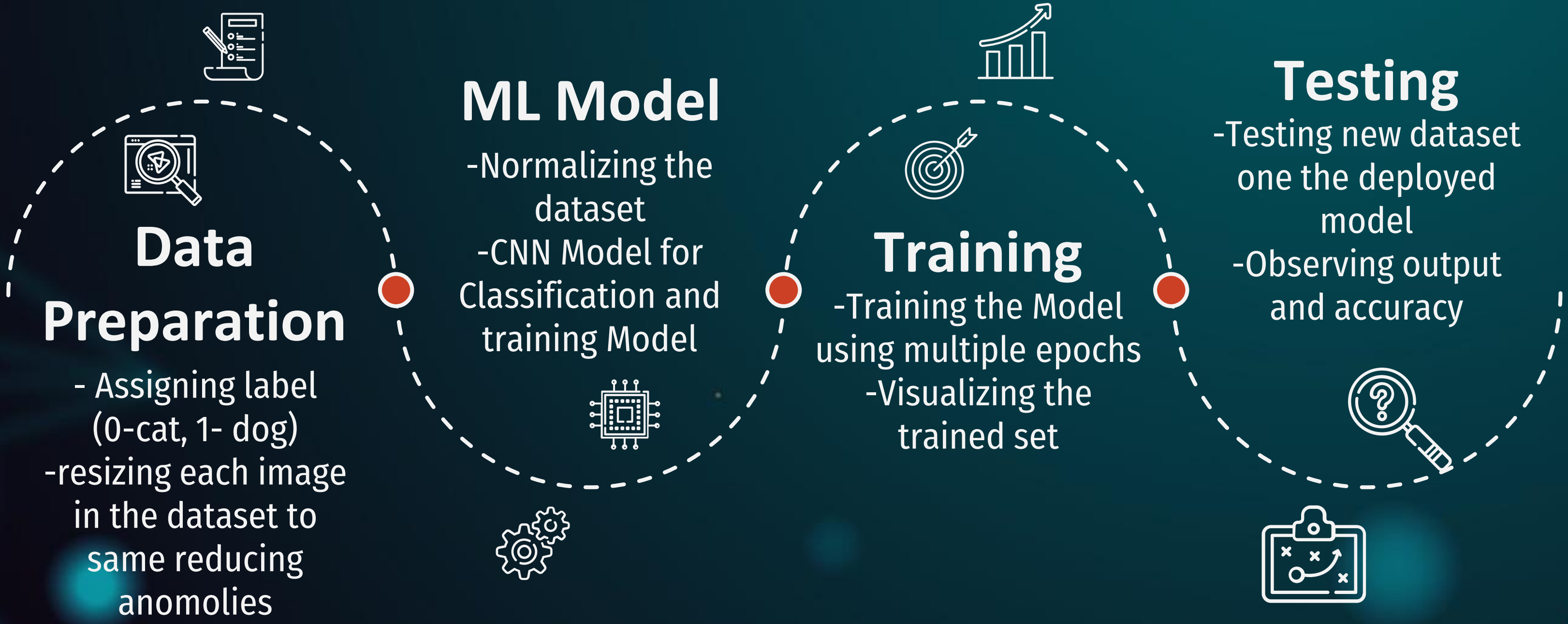
Guide- Prof. Anjali Purohit
Course - AIML

Introduction

In recent years, machine learning and deep learning techniques have made remarkable strides in image classification tasks, enabling computers to recognize and categorize objects in images with high accuracy. One popular and interesting application of these technologies is the classification of images into different categories, such as distinguishing between cats and dogs.

This project focuses on building a Cat vs. Dog classification system using Convolutional Neural Networks (CNNs) in Google Collab. The dataset for this project is sourced from Kaggle, a well-known platform for machine learning competitions and datasets. The dataset consists of a large collection of labeled images of cats and dogs, providing a diverse set of examples for training and evaluating the model.

Workflow of the Project



Data Preparation

- Assigning label (0-cat, 1- dog)
-resizing each image in the dataset to same reducing anomolies

ML Model

-Normalizing the dataset
-CNN Model for Classification and training Model

Training

-Training the Model using multiple epochs
-Visualizing the trained set

Testing

-Testing new dataset one the deployed model
-Observing output and accuracy

Objectives

Data Preparation:

Load the dataset from Kaggle into Google Colab.
Split the dataset into training and testing sets.

Building the CNN Model:

Design a Convolutional Neural Network architecture suitable for image classification.
Include convolutional layers to capture spatial hierarchies and pooling layers for down sampling.
Add fully connected layers for classification.

Model Training:

Train the CNN model using the training dataset.
Monitor the model's performance on the validation set to prevent overfitting.

Model Evaluation:

Evaluate the trained model on the test set to assess its generalization performance.
Analyze metrics such as accuracy, precision, recall, and F1 score.

Model Testing:

Test the model on new, unseen images to assess its real-world applicability.

Tech Stacks

Tech	Description
Kaggle	For Dataset
CNN	Deep learning architecture for image tasks
Tensorflow	Open-source machine learning framework
Keras	High-level API for neural networks
matplotlib	Python library for data visualization
openCV	Open-source computer vision library

What is CNN?

Convolutional Neural Networks (CNNs) are a powerful class of deep learning models that have revolutionized the field of computer vision. Unlike traditional neural networks that process data in a linear fashion, CNNs exploit the inherent spatial structure of images to extract meaningful features. They achieve this through the use of filters, also known as kernels, which are small matrices that slide across the input image, identifying local patterns and features. This process allows CNNs to learn hierarchical representations of images, starting from basic edges and textures to more complex objects and scenes.

A key characteristic of CNNs is their shared weights architecture. Each kernel is applied to the entire image, regardless of its position, which helps to reduce the number of parameters and promotes translation invariance – the ability of the model to recognize objects regardless of their location within the image. Additionally, CNNs often incorporate pooling layers, which downsample the feature maps by summarizing information from smaller regions, further increasing their efficiency and robustness.

Explanation of deployed CNN Model

Input Layer:
The input layer is configured with an input shape of (256, 256, 3), indicating that it accepts images of dimensions 256x256 pixels with three color channels (RGB).

Convolutional Layers:

1st Layer:

- Utilizes 32 filters, each with a kernel size of (3,3).
- Employs the Rectified Linear Unit (ReLU) activation function to introduce non-linearity.
- Applies batch normalization to normalize the input and accelerate training.
- Adopts valid padding to maintain spatial dimensions.
- Incorporates a max-pooling layer with a pool size of (2,2) and a stride of 2 for down-sampling.

2nd Layer:

- Employs 64 filters with a kernel size of (3,3).
- Utilizes ReLU activation, batch normalization, valid padding, and max-pooling similar to the 1st layer.

3rd Layer:

- Consists of 128 filters with a kernel size of (3,3).
- Applies ReLU activation, batch normalization, valid padding, and max-pooling similar to previous layers.

Explanation of deployed CNN Model

Flatten Layer:
The flatten layer transforms the output from the convolutional layers into a 1D array, preparing it for input into the subsequent dense layers.

Dense Layers:
1st Dense Layer:

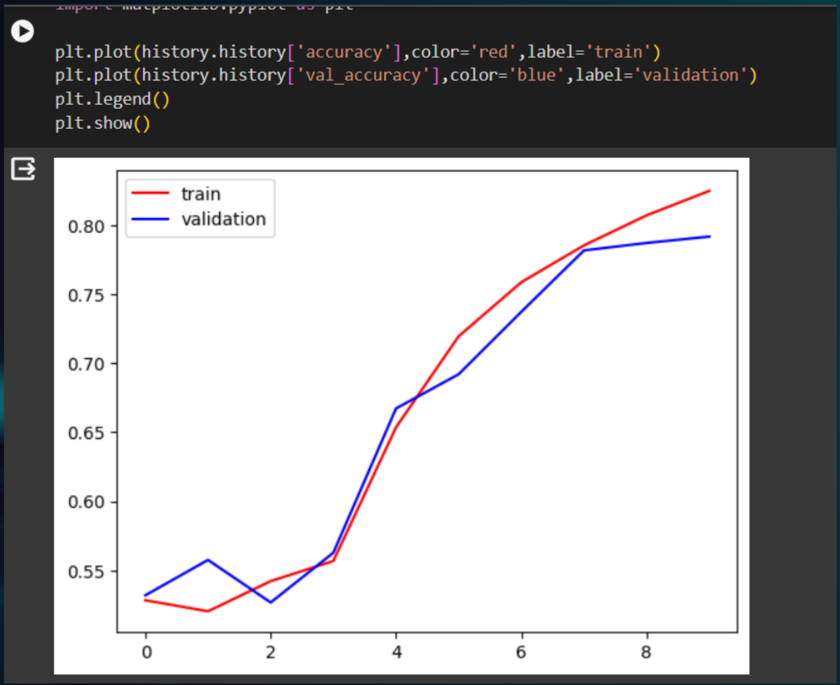
- Contains 128 neurons activated by ReLU.
- Incorporates a dropout layer with a 0.1 dropout rate to mitigate overfitting.

2nd Dense Layer:

- Comprises 64 neurons activated by ReLU.
- Features a dropout layer with a 0.1 dropout rate for regularization.

Output Layer:
The output layer consists of a single neuron using a sigmoid activation function, facilitating binary classification for determining whether the input image represents a cat or a dog.

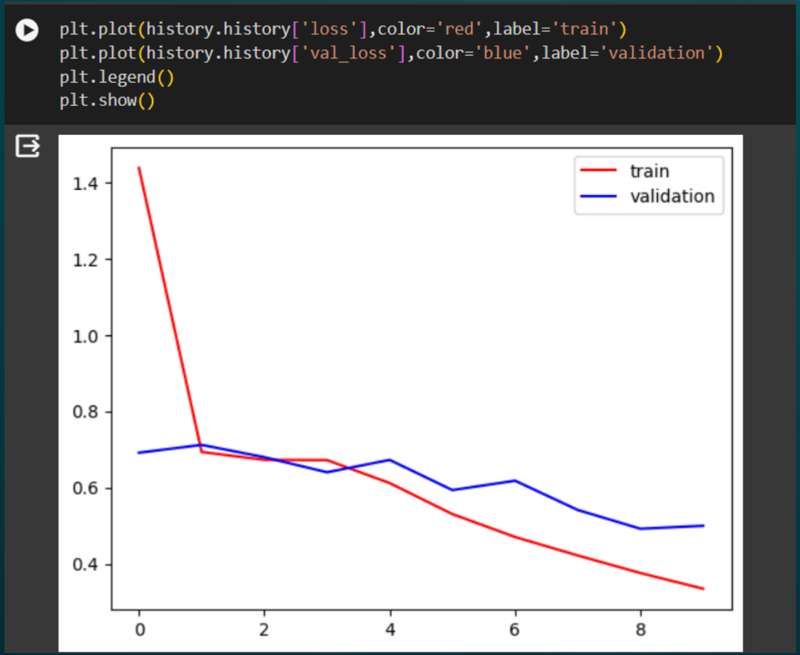
Data Visualization and Validation



```
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()
```

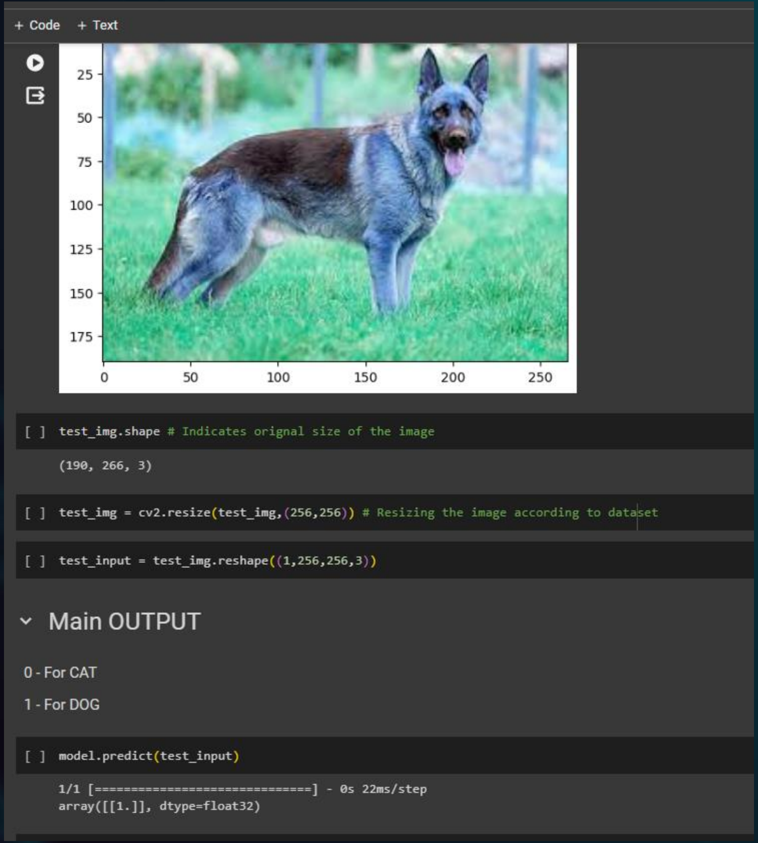
Training accuracy Vs validation accuracy



```
plt.plot(history.history['loss'],color='red',label='train')
plt.plot(history.history['val_loss'],color='blue',label='validation')
plt.legend()
plt.show()
```

Training loss Vs validation loss

Output recived from untrained data



```
[ ] test_img.shape # Indicates orignal size of the image

    (190, 266, 3)

[ ] test_img = cv2.resize(test_img,(256,256)) # Resizing the image according to dataset

[ ] test_input = test_img.reshape((1,256,256,3))
```

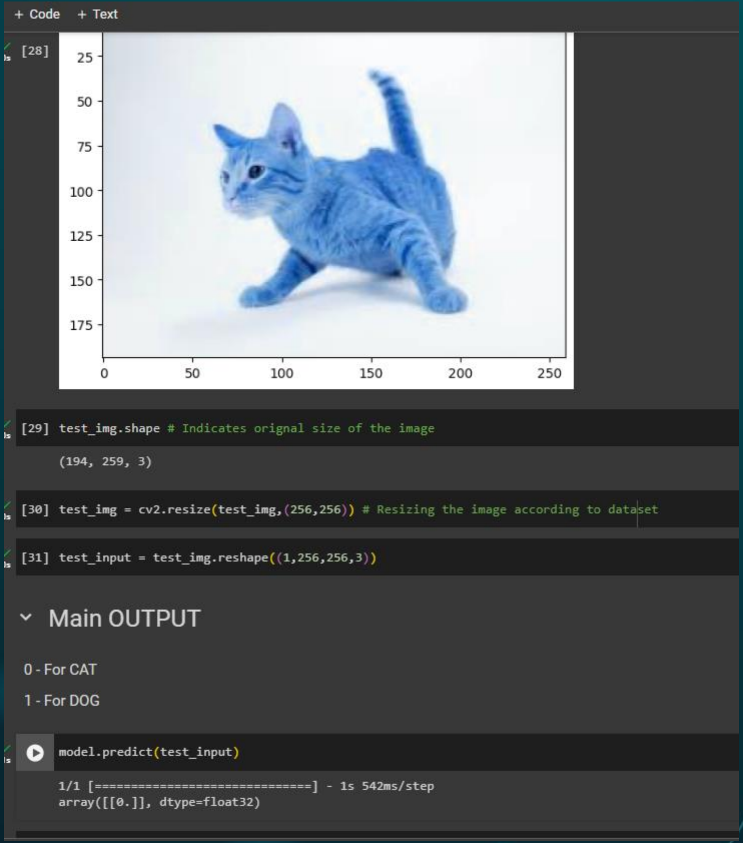
Main OUTPUT

0 - For CAT

1 - For DOG

```
[ ] model.predict(test_input)

    1/1 [==============================] - 0s 22ms/step
    array([[1.]], dtype=float32)
```



```
[28]

[29] test_img.shape # Indicates orignal size of the image

    (194, 259, 3)

[30] test_img = cv2.resize(test_img,(256,256)) # Resizing the image according to dataset

[31] test_input = test_img.reshape((1,256,256,3))
```

Main OUTPUT

0 - For CAT

1 - For DOG

```
model.predict(test_input)

    1/1 [==============================] - 1s 542ms/step
    array([[0.]], dtype=float32)
```

Conclusion

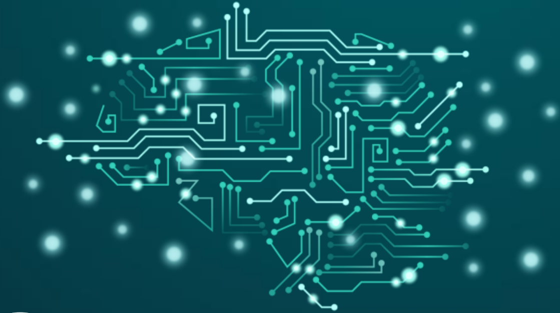
In conclusion, the Convolutional Neural Network (CNN) developed for the Cat vs. Dog classification project represents a powerful solution for image recognition tasks. The meticulously crafted architecture, with its convolutional layers extracting hierarchical features and pooling layers aiding in spatial down-sampling, showcases the effectiveness of CNNs in capturing intricate patterns within images. The incorporation of batch normalization and dropout layers contributes to the model's stability and generalization performance. This project not only leverages essential technologies such as TensorFlow, Keras, and OpenCV but also utilizes the Kaggle platform for access to a rich dataset and collaborative development. As a testament to the success of CNNs, the model demonstrates the ability to discern between cats and dogs with high accuracy. Going forward, the lessons learned from this project can be extended to broader applications, highlighting the versatility and potency of CNNs in diverse image classification endeavors.

Resources:

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

https://www.tensorflow.org/tutorials
https://keras.io/
https://docs.opencv.org/master/d9/df8/tutorial_root.html

https://towardsdatascience.com/
http://cs231n.stanford.edu/



Thankyou