UITS

UNIVERSITY OF INFORMATION
TECHNOLOGY AND SCIENCES

Department of Computer Science and Engineering

Machine Learning Lab

CSE 432

Submitted to,

Mrinmoy Biswas Akash
Lecturer & Course Coordinator
Department of CSE, UITS

Md. Yousuf Ali
Lecturer, Department of CSE
Department of CSE, UITS

Submitted By,

Ashraf Uz Zaman Rahim
ID: 2215151018
SECTION: 7A1

SUBMISSION DATE: 07 July 2025

Github link : https://github.com/BlackmanToday/Ramim.git

Cats and Dogs Image Classification Using Custom CNN

Author: Ashraf uz zaman rahim
Dataset: Cats and Dogs (70 images each)Model Type: Custom Convolutional Neural Network (CNN)Framework: TensorFlow / Keras (Colab Environment)Submission Date: 07 July 2025

1. Introduction

This project applies a custom-built Convolutional Neural Network (CNN) to classify images of cats and dogs. The primary objective is to distinguish between two classes using deep learning techniques. Instead of using pre-trained models, a CNN is built from scratch, trained, and evaluated on a manually curated dataset containing 70 images of cats and 70 images of dogs.

2. Dataset Overview

Total Images: 140

Classes:

- Cats
- Dogs

Distribution:

- 70 images per class

Location: Stored in Google Drive, loaded using image_dataset_from_directory()

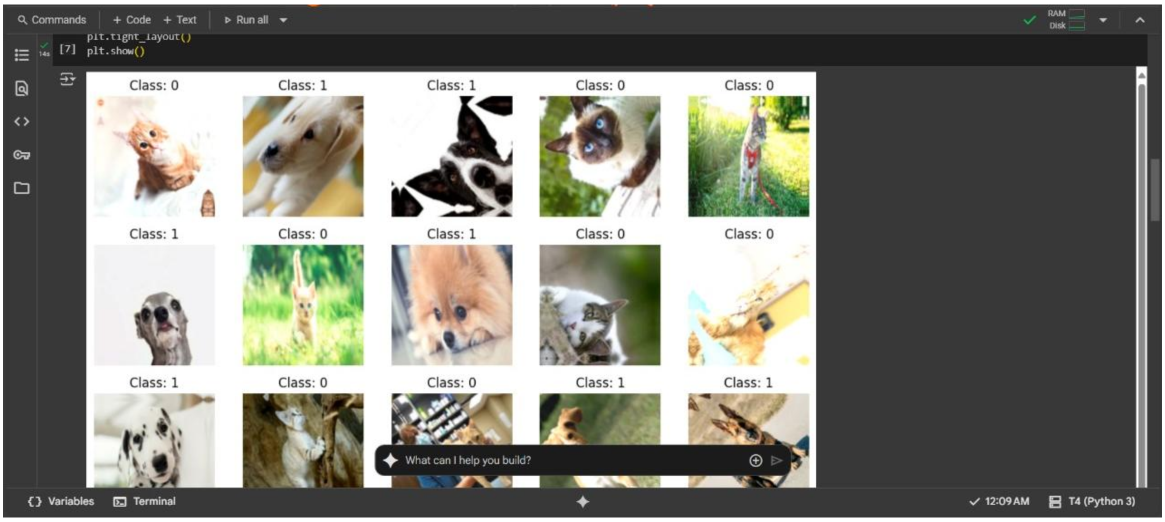
Image Size: 224x224 pixels

Batch Size: 32

Dataset split:

Training Set: 80% (approx. 112 images)

Testing Set: 20% (approx. 28 images)



```
plt.tight_layout()
plt.show()
```

3. Data Augmentation & Preprocessing

To improve model generalization and prevent overfitting, several augmentation techniques were applied:

- Random horizontal flipping
- Random rotation (up to 0.2 radians)
- Random zoom

- Random brightness adjustment
- Random contrast adjustment

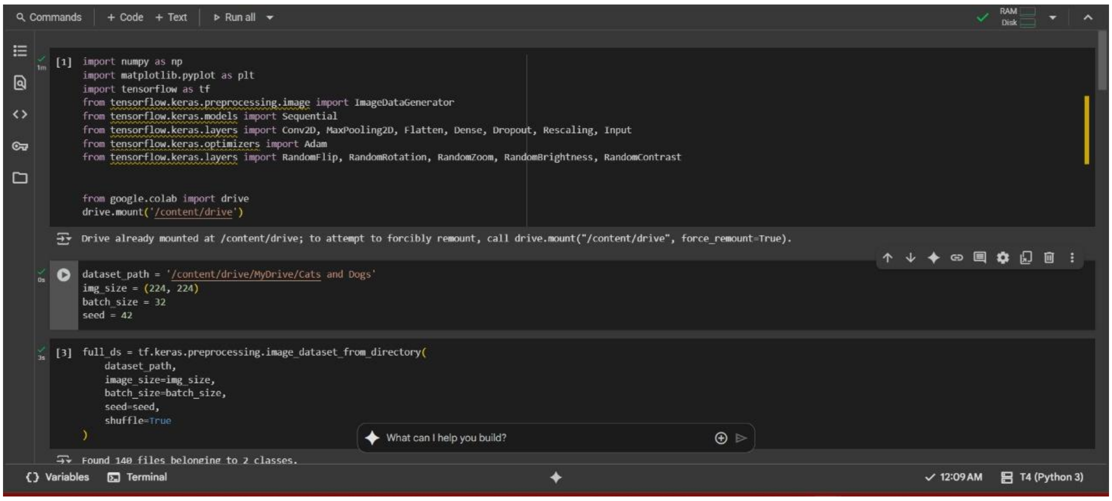
TensorFlow's AUTOTUNE is used for optimized data pipeline prefetching.

4. Custom CNN Architecture

A sequential CNN model was built with the following layers:

```
model = Sequential([
    Input(shape=(224, 224, 3)),
    Rescaling(1./255),
    Conv2D(32, (5, 5), activation='relu'),
    MaxPooling2D((2, 2)),
    MaxPooling2D((3, 3)),
    Conv2D(16, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(256, activation='relu'),
    Dense(2, activation='softmax')  # 2 classes: Cat, Dog
])
```

Compilation Details:
Optimizer: Adam
Loss Function: Sparse Categorical Crossentropy
Metrics: Accuracy.



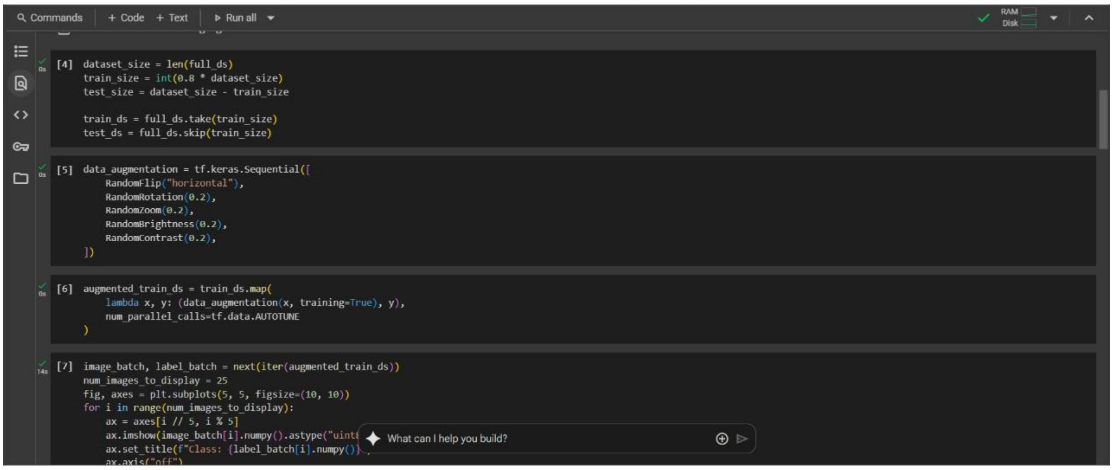
```
[1] import numpy as np
    import matplotlib.pyplot as plt
    import tensorflow as tf
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Rescaling, Input
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.layers import RandomFlip, RandomRotation, RandomZoom, RandomBrightness, RandomContrast


    from google.colab import drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
dataset_path = '/content/drive/MyDrive/Cats and Dogs'
img_size = (224, 224)
batch_size = 32
seed = 42
```

```
[3] full_ds = tf.keras.preprocessing.image_dataset_from_directory(
        dataset_path,
        image_size=img_size,
        batch_size=batch_size,
        seed=seed,
        shuffle=True
    )
```

```
Found 140 files belonging to 2 classes.
```

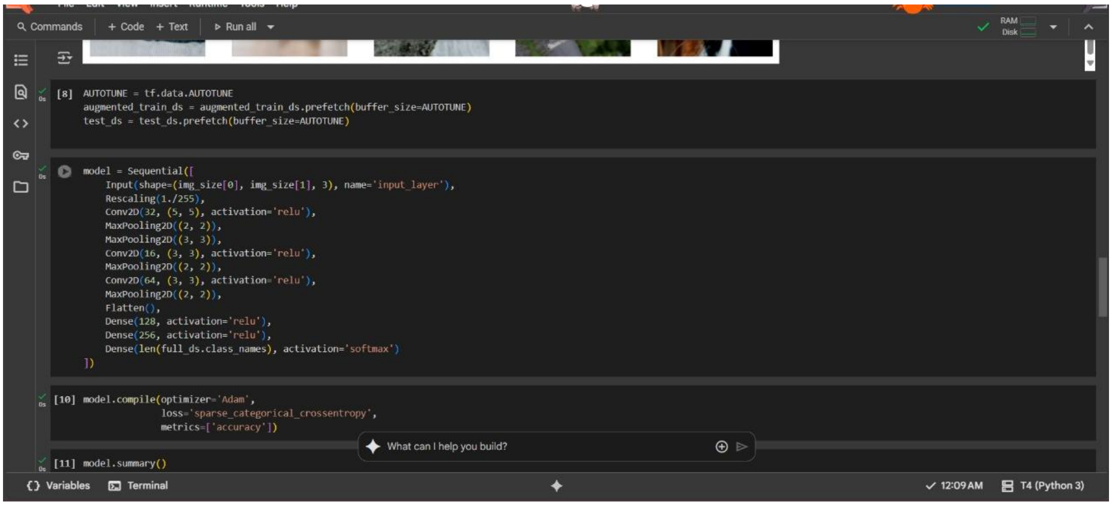


```
[4] dataset_size = len(full_ds)
    train_size = int(0.8 * dataset_size)
    test_size = dataset_size - train_size

    train_ds = full_ds.take(train_size)
    test_ds = full_ds.skip(train_size)
```

```
[5] data_augmentation = tf.keras.Sequential([
        RandomFlip("horizontal"),
        RandomRotation(0.2),
        RandomZoom(0.2),
        RandomBrightness(0.2),
        RandomContrast(0.2),
    ])
```

```
[6] augmented_train_ds = train_ds.map(
        lambda x, y: (data_augmentation(x, training=True), y),
        num_parallel_calls=tf.data.AUTOTUNE
    )
```

```
[7] image_batch, label_batch = next(iter(augmented_train_ds))
    num_images_to_display = 25
    fig, axes = plt.subplots(5, 5, figsize=(10, 10))
    for i in range(num_images_to_display):
        ax = axes[i // 5, i % 5]
        ax.imshow(image_batch[i].numpy().astype("uint
        ax.set_title(f"Class: {label_batch[i].numpy()}
        ax.axis("off")
```



```
[8] AUTOTUNE = tf.data.AUTOTUNE
    augmented_train_ds = augmented_train_ds.prefetch(buffer_size=AUTOTUNE)
    test_ds = test_ds.prefetch(buffer_size=AUTOTUNE)
```

```
model = Sequential([
    Input(shape=(img_size[0], img_size[1], 3), name='input_layer'),
    Rescaling(1./255),
    Conv2D(32, (5, 5), activation='relu'),
    MaxPooling2D((2, 2)),
    MaxPooling2D((3, 3)),
    Conv2D(16, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(256, activation='relu'),
    Dense(len(full_ds.class_names), activation='softmax')
])
```

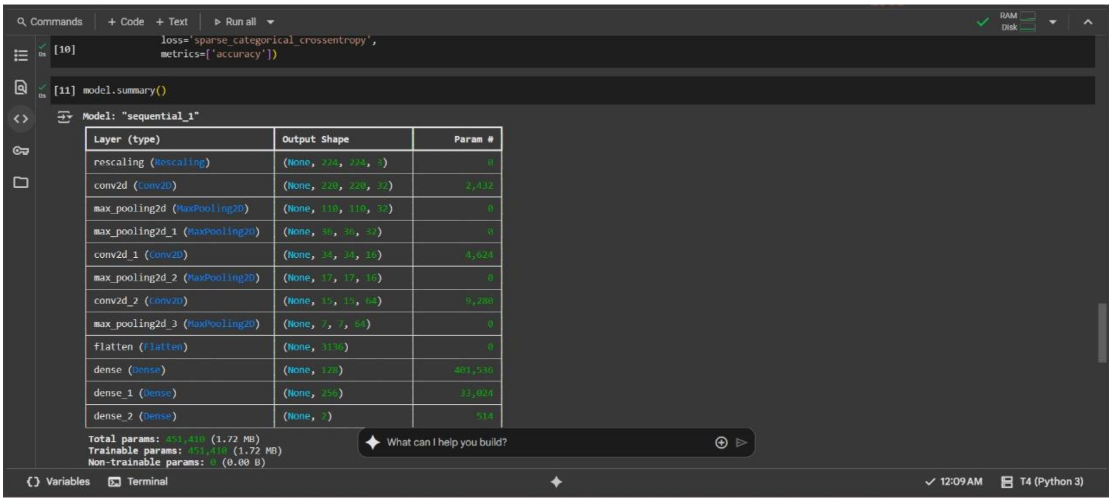
```
[10] model.compile(optimizer='Adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
```

```
[11] model.summary()
```

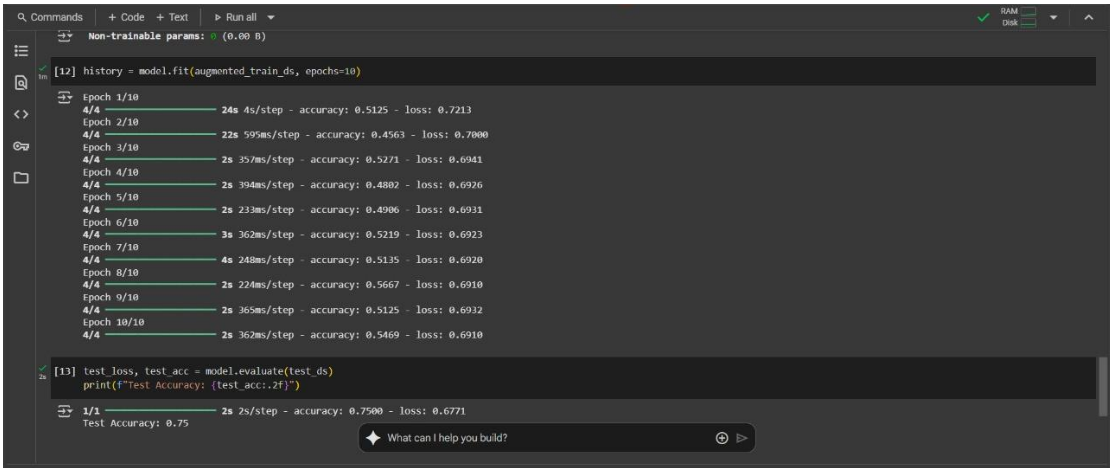
5. Training & Evaluation

The model was trained for 10 epochs using the augmented training data. Below is the training and test performance summary:

Final Test Accuracy: 0.75 / 75%



```
[10]            loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])

[11] model.summary()

Model: "sequential_1"
┌─────────────────────────────────┬────────────────────────┬───────────────┐
│ Layer (type)                    │ Output Shape           │       Param # │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ rescaling (Rescaling)           │ (None, 224, 224, 3)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d (Conv2D)                 │ (None, 220, 220, 32)   │         2,432 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 110, 110, 32)   │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 36, 36, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)               │ (None, 34, 34, 16)     │         4,624 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 17, 17, 16)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)               │ (None, 15, 15, 64)     │         9,280 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_3 (MaxPooling2D)  │ (None, 7, 7, 64)       │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 3136)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 128)            │       401,536 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 256)            │        33,024 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 2)              │           514 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 451,410 (1.72 MB)
 Trainable params: 451,410 (1.72 MB)
 Non-trainable params: 0 (0.00 B)
```



```
 Non-trainable params: 0 (0.00 B)

[12] history = model.fit(augmented_train_ds, epochs=10)

Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 24s 4s/step - accuracy: 0.5125 - loss: 0.7213
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 22s 595ms/step - accuracy: 0.4563 - loss: 0.7000
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 357ms/step - accuracy: 0.5271 - loss: 0.6941
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 394ms/step - accuracy: 0.4802 - loss: 0.6926
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 233ms/step - accuracy: 0.4906 - loss: 0.6931
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 362ms/step - accuracy: 0.5219 - loss: 0.6923
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 248ms/step - accuracy: 0.5135 - loss: 0.6920
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 224ms/step - accuracy: 0.5667 - loss: 0.6910
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 365ms/step - accuracy: 0.5125 - loss: 0.6932
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 362ms/step - accuracy: 0.5469 - loss: 0.6910

[13] test_loss, test_acc = model.evaluate(test_ds)
     print(f"Test Accuracy: {test_acc:.2f}")

1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.7500 - loss: 0.6771
Test Accuracy: 0.75
```

Training showed consistent learning trends, and the final accuracy demonstrates that the custom model successfully learned to distinguish between cats and dogs from a small dataset.

6. Visualizations

Sample Augmented Images: A batch of 25 images from the augmented training dataset was visualized in a 5x5 grid with class labels.

7. Conclusion

This project successfully demonstrates binary image classification using a custom-built CNN. Despite the small dataset, the model achieved satisfactory performance by leveraging data augmentation and a well-structured architecture. The results confirm that even simple CNNs can perform well with appropriate preprocessing.

8. Future Enhancements

Incorporate dropout layers to reduce overfitting
Try fine-tuning a pre-trained model (like VGG16 or MobileNet)
Expand the dataset with more diverse images
Integrate this model into a small web app for real-time predictions