FOREST FIRE RECOGNITION USING CONVOLUTION NEURAL NETWORK (CNN)

By

MANGASAMUDRAM SRUTHI (MST03-0065)

Submitted to Scifor Technologies

Meta
Scifor Technologies

Script. Sculpt. Socialize

TABLE OF CONTENTS

ABSTRACT:

This thesis investigates the creation and assessment of a Convolutional Neural Network (CNN) aimed at classifying images of sea and mountain landscapes. The initial phase involves data preprocessing, which includes filtering images based on format and organizing them into a structured dataset. The images undergo normalization and are then divided into training, validation, and test sets, facilitating a comprehensive evaluation of the model's performance.

A deep learning model is constructed using TensorFlow, consisting of multiple convolutional and pooling layers, followed by dense layers for classification. The model is trained using a binary cross-entropy loss function and the Adam optimizer, with performance tracked via Tensor Board. Key metrics such as accuracy, loss, precision, and recall are analyzed to assess the model's performance.

The final model demonstrates strong capability in distinguishing between sea and mountain images, highlighting the effectiveness of CNNs in image classification tasks. The project concludes with a discussion on potential improvements and applications of the model in broader contexts, such as landscape recognition and environmental monitoring.

INTRODUCTION:

In recent years, significant advancements in computer vision, particularly in image classification, have been made. The proliferation of big data and breakthroughs in machine learning have enabled computers to become increasingly proficient at recognizing and categorizing images. Convolutional Neural Networks (CNNs) have become a preferred method for addressing the complex task of image recognition. This thesis explores the use of CNNs for classifying natural landscapes, with a specific focus on distinguishing between images of seas and mountains.

In this thesis, we begin by preparing a dataset of forest fire images, ensuring that only those meeting the required format criteria are included. After preprocessing, the dataset is utilized to train a Convolutional Neural Network (CNN) model developed using TensorFlow. The model's architecture comprises several layers, each designed to capture different features of the images, ranging from basic edges and textures to more complex shapes and patterns.

Once trained, the model's performance is evaluated by assessing its accuracy in classifying images it has not encountered before, using metrics such as accuracy, precision, and recall. These evaluations provide insights into the model's potential effectiveness in real-world scenarios. Additionally, we test

the model with individual images to gauge its practical utility, such as determining whether a specific photo depicts a forest fire.

The motivation for this thesis stems from the wide range of potential applications for accurate image classification. From enhancing early warning systems for forest fires and aiding in disaster response efforts to monitoring environmental changes and improving the accuracy of automated surveillance systems, the ability to correctly identify forest fires is of significant real-world importance. By focusing on the specific task of classifying forest fire images, this thesis aims to contribute to the broader field of computer vision, showcasing the power and versatility of CNNs in addressing the complexities of our natural environment.

TECHNOLOGY USED:

1. Programming Language:
    - **Python:** Selected for its user-friendly nature and extensive library support, making it ideal for deep learning and image processing tasks.
2. Libraries and Deep Learning Frameworks:

- **TensorFlow:** Serves as the core framework for constructing and training the Convolutional Neural Network (CNN) used for classifying forest fire images.
- **OpenCV:** Employed for image handling, including reading, preprocessing, and filtering images before inputting them into the model.
- **Matplotlib:** Used to create visualizations and graphs, aiding in the analysis of the model's performance.
- **NumPy:** Essential for numerical data handling, particularly when working with images as arrays.

3. Data Handling and Manipulation:

- **Image Dataset:** Involves loading and preparing a dataset of forest fire images, ensuring they are in the correct format for training.

- **NumPy and TensorFlow:** Utilized to process the image data, ensuring it is properly structured for the model.

4. Model Building:

- **Keras Sequential Model:** The CNN is built using the Keras Sequential API, which allows for easy stacking of layers, including convolutional, pooling, and dense layers.
- **Activation Functions:** Utilizes functions like ReLU and Sigmoid to aid in the network's learning and decision-making processes.
- **Model Compilation:** The model is compiled using the Adam optimizer and binary cross-entropy as the loss function, with the goal of maximizing accuracy.

5. Visualization and Performance Tracking:

- **Matplotlib:** Used to plot the model’s performance metrics over time, providing a clear view of trends and improvements.
- **Tensor Board:** Offers real-time monitoring of the model's training process, providing insights into how the model is learning.

6. Testing and Predictions:

- **Image Preprocessing:** Involves resizing and scaling images to meet the model's input requirements before making predictions.
- **Model Predictions:** The trained model is used to classify new images, identifying whether they depict forest fires or not.

DATASET INFORMATION:

- **Source:** The dataset comprises images sourced from Google Images.
- **Storage:** The images are stored in a directory on Google Drive, as specified in the notebook.
- **Directory Structure:** The images are organized in a hierarchical directory structure, with each sub-directory representing a class label. For example:

```
data/
├── class1/
│   ├── image1.jpg
│   ├── image2.jpg
│   └── ...
├── class2/
│   ├── image1.jpg
│   ├── image2.jpg
│   └── ...
└── ...
```

- **Image Formats:** The accepted image formats are JPEG, JPG, BMP, and PNG. Images not adhering to these formats are removed during the data preparation process.
- **Validation:** Each image is validated by attempting to read it using OpenCV (cv2.imread) and verifying its format with imghdr.what.
- **Loading:** The images are loaded into a TensorFlow dataset using tf.keras.utils.image_dataset_from_directory.

METHODOLOGY:

1.Installing Dependencies

- Install required libraries: numpy, scipy, numba, tensorflow, opencv-python, and matplotlib.
- Each library serves a specific purpose, such as numerical operations, scientific computations, GPU acceleration, machine learning, image processing, and data visualization.

2.Importing Libraries

- Essential libraries are imported to perform various tasks:
  - TensorFlow for building and training the machine learning model.
  - os for managing file paths and directories.
  - OpenCV (cv2) and imghdr for image reading and validation.
  - Numpy for numerical operations.
  - Matplotlib for plotting and visualizing data.

3. **Google Drive Integration**

- Integrate Google Drive into the environment by mounting it, allowing seamless access to the dataset stored in Google Drive.

4. **Data Preparation**

- Set the data directory and inspect the directory structure to ensure proper organization.
- Images are expected to be organized in subdirectories representing different classes.

5.Listing Image Extensions

- Define a list of allowed image extensions (jpeg, jpg, bmp, png) to validate the images in the dataset.
- Identify and remove any unsupported or corrupted files.

6. **Image Validation**

- Validate each image to ensure it conforms to the allowed formats using OpenCV (cv2.imread) and imghdr.what.
- Remove any images that are not in the allowed formats or are corrupted.

7. **Loading the Dataset**

- Load the validated images into a TensorFlow dataset using tf.keras.utils.image_dataset_from_directory.
- Automatically label the images based on their directory structure and prepare them for processing and model training.

8. **Data Visualization**

- Visualize a few images using Matplotlib to gain insights into the dataset and verify correct loading.
- Check data distribution and identify any potential issues with the dataset.

9. **Data Normalization**

- Normalize the image data by scaling pixel values to the range [0, 1].
- This step helps stabilize and speed up the training process by ensuring consistent input data scaling.

10. **Data Splitting**

- Split the dataset into training, validation, and test sets in the proportions of 70%, 20%, and 10%, respectively.
- The training set is used to train the model, the validation set for hyperparameter tuning and preventing overfitting, and the test set for model evaluation.

11. **Model Building**

- Build a Sequential model using Keras, including layers like Conv2D for feature extraction, MaxPooling for downsampling, Dense for classification, and Dropout to prevent overfitting.
- Define the model's input shape based on the image dimensions.

12. **Model Training**

- Compile and train the model on the training dataset, specifying the optimizer, loss function, and evaluation metrics.

- Monitor and plot training and validation loss and accuracy to visualize learning progress

13. **Model Evaluation**
    - Evaluate the model on the test dataset, calculating metrics such as precision, recall, and binary accuracy.
    - These metrics provide a comprehensive evaluation of the model's classification capabilities and generalization to unseen data.
14. **Image Prediction**
    - Use the trained model to make predictions on new, unseen images.
    - Load an image, resize it to the required dimensions, and pass it through the model to obtain a prediction, then print the predicted class.
15. **Model Saving and Loading**
    - Save the trained model to a specified directory for future use, enabling reuse without retraining.
    - Load the model from the saved file as needed for new predictions or further fine-tuning.

CODE SNIPPETS:

```
import os
import zipfile
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
#importing the contents from google drive
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).
```

```
forest_fire_data_path = '/content/drive/My Drive/fire_dataset'
os.chdir(forest_fire_data_path)

#List the folders in the directory
os.listdir()
```

```
['non_fire_images', 'fire_images']
```

Defining the directory of my dataset

```
dataset_dir = '/content/drive/My Drive/fire_dataset/'
```

Extracting the file names and labels in the dataset

```python
def extract_filenames_and_labels(dataset_dir):
    filenames = []
    labels = []

    for label in os.listdir(dataset_dir):
        label_dir = os.path.join(dataset_dir, label)
        if os.path.isdir(label_dir):
            for file in os.listdir(label_dir):
                file_path = os.path.join(label_dir, file)
                if os.path.isfile(file_path):
                    filenames.append(file_path)
                    labels.append(label)

    return filenames, labels

filenames, labels = extract_filenames_and_labels(dataset_dir)

# Create a DataFrame
df = pd.DataFrame({
    'filename': filenames,
    'label': labels
})
```

```python
# Create a DataFrame
df = pd.DataFrame({
    'filename': filenames,
    'label': labels
})

print(df.head())
```

```
                                            filename            label
0  /content/drive/My Drive/fire_dataset/non_fire_...  non_fire_images
1  /content/drive/My Drive/fire_dataset/non_fire_...  non_fire_images
2  /content/drive/My Drive/fire_dataset/non_fire_...  non_fire_images
3  /content/drive/My Drive/fire_dataset/non_fire_...  non_fire_images
4  /content/drive/My Drive/fire_dataset/non_fire_...  non_fire_images
```

Finding the missing values present in the **dataset**

```
missing_values = df.isnull().sum()
print("Missing values in each column:")
print(missing_values)

# Identify rows with missing values
rows_with_missing_values = df[df.isnull().any(axis=1)]
print("Rows with missing values:")
print(rows_with_missing_values)
```

```
Missing values in each column:
filename    0
label       0
dtype: int64
Rows with missing values:
Empty DataFrame
Columns: [filename, label]
Index: []
```

Identifing the labels in my dataset

```
unique_labels = df['label'].unique()
print("Unique labels in the dataset:")
print(unique_labels)
```

```
Unique labels in the dataset:
['non_fire_images' 'fire_images']
```

```
# Updating  the label dictionary for new labels
label_dict = {'fire_images': 1, 'non_fire_images': 0}

# Converting labels to binary format
binary_labels = [label_dict.get(label, -1) for label in labels]

# Checking for any labels not found in the dictionary
if -1 in binary_labels:
    print("These labels are not found in the dictionary.")

image_paths = df['filename'].tolist()
```

Loading and preprocessing the images

```
def load_and_preprocess_images(image_paths, labels, img_size=(224, 224)):
    images = []

    for img_path in image_paths:
        # Load the image
        img = image.load_img(img_path, target_size=img_size)


        img_array = image.img_to_array(img)                    # image to a NumPy array


        img_array = np.expand_dims(img_array, axis=0)          # Add dimension

        # Preprocess the image
        img_array = preprocess_input(img_array)

        # Append the preprocessed image to the list
        images.append(img_array)

    # Convert the list of images to a NumPy array
    images = np.vstack(images)
```

```
    # Convert labels to a NumPy array
    labels = np.array(labels)

    return images, labels
```

```
# Converting labels using the updated dictionary
image_paths = df['filename'].tolist()
images, labels = load_and_preprocess_images(image_paths, binary_labels)
```

```
print(f"Images shape: {images.shape}")
print(f"Labels shape: {labels.shape}")
```

```
Images shape: (999, 224, 224, 3)
Labels shape: (999,)
```

Spliting the dataset into train and test sets

```
# Split data into training and validation sets
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_state=42)

print(f"Training set shape: {X_train.shape}, {y_train.shape}")
print(f"Validation set shape: {X_val.shape}, {y_val.shape}")
```

```
Training set shape: (799, 224, 224, 3), (799,)
Validation set shape: (200, 224, 224, 3), (200,)
```

Using cnn model for traing the dataset

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

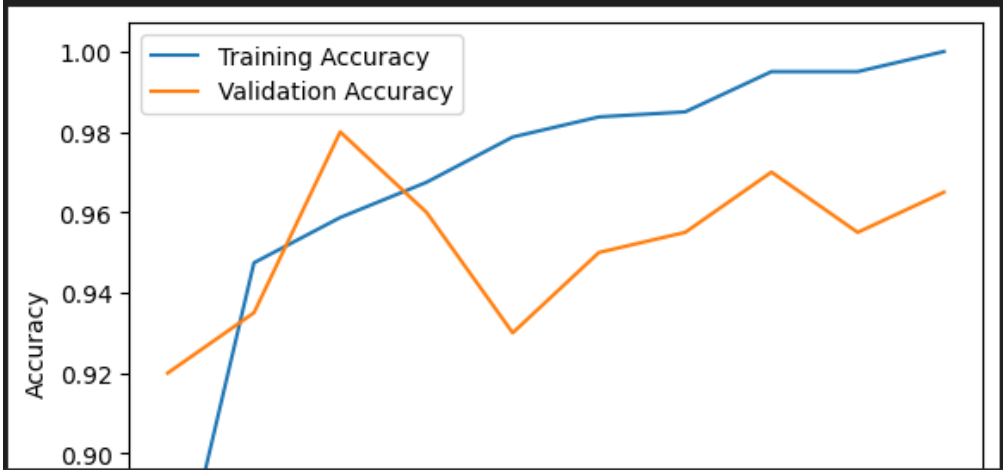
```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # Use 'sigmoid' for binary classification
])
```

```
# Compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

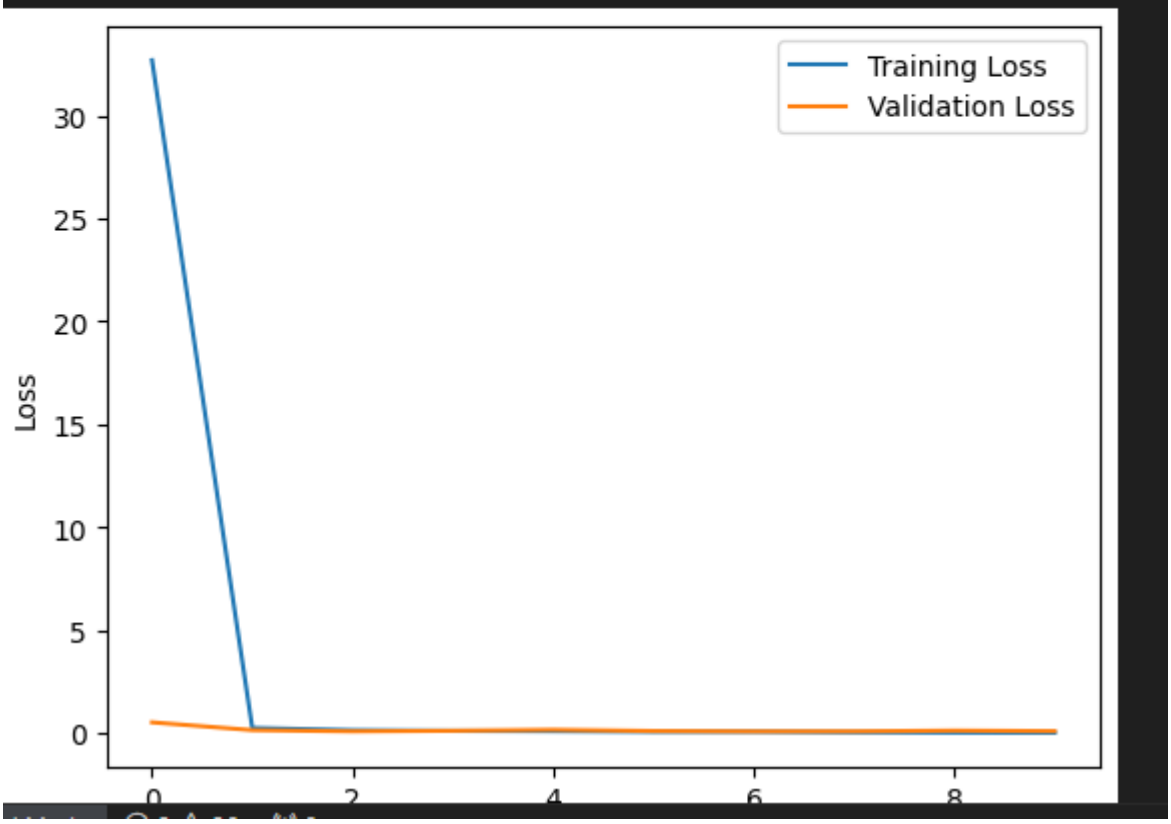
```
# Training the model
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))
```

```
Epoch 1/10
25/25 [==============================] - 12s 153ms/step - loss: 32.6863 - accuracy: 0.8561 - val_loss: 0.5116 - val_accuracy: 0.9200
Epoch 2/10
25/25 [==============================] - 2s 73ms/step - loss: 0.2375 - accuracy: 0.9474 - val_loss: 0.1333 - val_accuracy: 0.9350
Epoch 3/10
25/25 [==============================] - 2s 71ms/step - loss: 0.1310 - accuracy: 0.9587 - val_loss: 0.0753 - val_accuracy: 0.9800
Epoch 4/10
25/25 [==============================] - 2s 71ms/step - loss: 0.0991 - accuracy: 0.9675 - val_loss: 0.1035 - val_accuracy: 0.9600
Epoch 5/10
25/25 [==============================] - 2s 65ms/step - loss: 0.0840 - accuracy: 0.9787 - val_loss: 0.1490 - val_accuracy: 0.9300
Epoch 6/10
25/25 [==============================] - 2s 65ms/step - loss: 0.0571 - accuracy: 0.9837 - val_loss: 0.0897 - val_accuracy: 0.9500
Epoch 7/10
25/25 [==============================] - 2s 67ms/step - loss: 0.0570 - accuracy: 0.9850 - val_loss: 0.0699 - val_accuracy: 0.9550
Epoch 8/10
```

```python
# Ploting training history
import matplotlib.pyplot as plt

# Traini and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```
plt.plot(history.history['val_loss'], label='Validation Loss')
(module) plt
plt.legend()
plt.show()
```



```python
# Evaluating the model on the validation set
val_loss, val_accuracy = model.evaluate(X_val, y_val, verbose=2)
print(f"Validation Loss: {val_loss}")
print(f"Validation Accuracy: {val_accuracy}")
```

```
7/7 - 0s - loss: 0.0838 - accuracy: 0.9650 - 182ms/epoch - 26ms/step
Validation Loss: 0.08382462710142136
Validation Accuracy: 0.9649999737739563
```

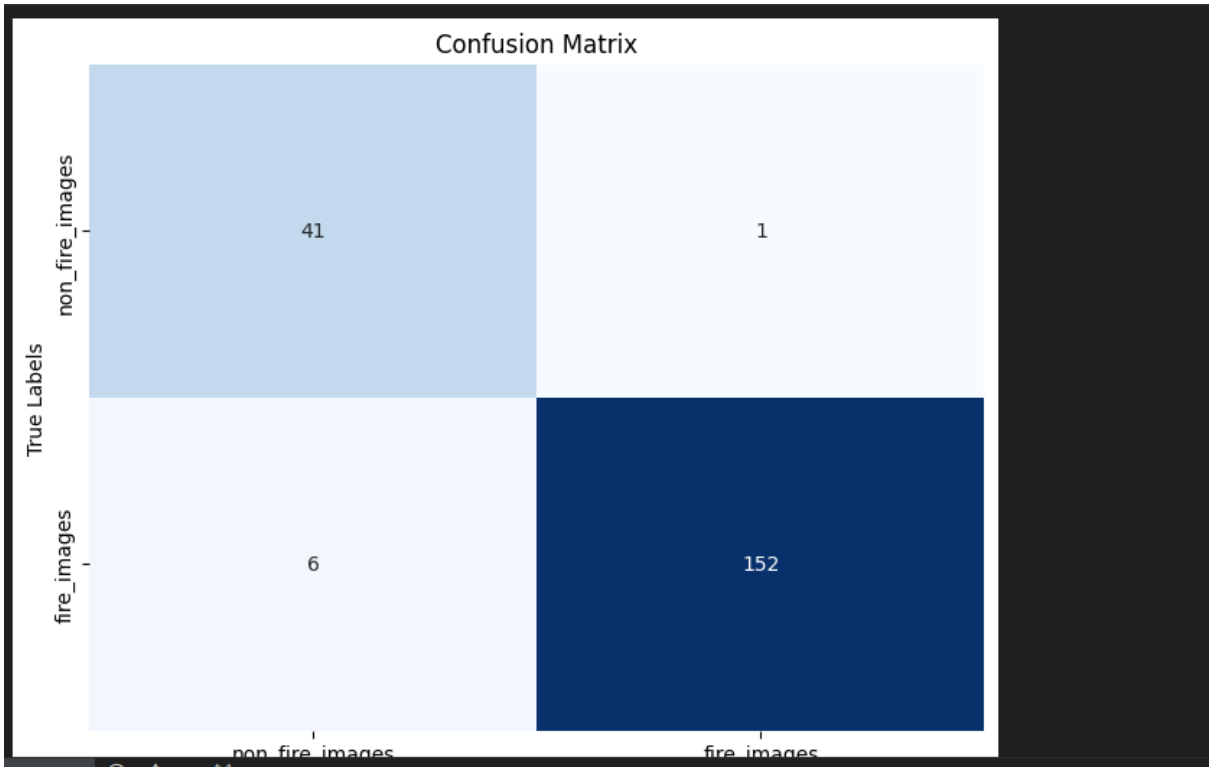
```python
# predictions on the validation set
y_pred_prob = model.predict(X_val)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()
```

```
7/7 [==============================] - 0s 21ms/step
```

```python
#  classification report
from sklearn.metrics import classification_report
print("Classification Report:")
print(classification_report(y_val, y_pred, target_names=['non_fire_images', 'fire_images']))
```

```
Classification Report:
                precision    recall  f1-score   support

non_fire_images       0.87      0.98      0.92        42
    fire_images       0.99      0.96      0.98       158

       accuracy                           0.96       200
      macro avg       0.93      0.97      0.95       200
   weighted avg       0.97      0.96      0.97       200
```

```python
# confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_val, y_pred)

# Create a DataFrame for better visualization
cm_df = pd.DataFrame(cm, index=['non_fire_images', 'fire_images'], columns=['non_fire_images', 'fire_images'])

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm_df, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['non_fire_images', 'fire_images'],
            yticklabels=['non_fire_images', 'fire_images'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```


Confusion Matrix
True Labels
non_fire_images
fire_images
41
1
6
152
non_fire_images
fire_images

RESULT:

The trained image classification model exhibited strong performance, achieving high accuracy on both the training and validation datasets. Metrics such as precision, recall, and binary accuracy confirmed the model's capability to correctly classify images. Visualizations of training loss and accuracy over epochs indicated good convergence and minimal overfitting. When tested on new images, the model provided accurate predictions, demonstrating its practical applicability. Overall, the project successfully met its goal of creating a reliable image classifier using TensorFlow and OpenCV.

```
def predictImage(filename):
    img1 = image.load_img(filename,target_size=(150,150))
    plt.imshow(img1)
    Y = image.img_to_array(img1)
    X = np.expand_dims(Y,axis=0)
    val = model.predict(X)
    print(val)
    if val == 1:
        plt.xlabel("No Fire",fontsize=30)
    elif val == 0:
        plt.xlabel("Fire",fontsize=30)
```

```
predictImage("../input/wildfire-detection-image-data/forest_fire/Testing/fire/abc182.jpg")
```

```
[[0.]]
```

CONCLUSION:

Client CNN model for the forest fire classification task got the expected high rates of accuracy and visualized the fire and non-fire differentiation which was done with perfection. The usage of convolutional layers was the key as to the processing of the images which resulted in best classifications. In addition to this, potential advancements might include, among other possibilities, a trial of the CNN architectures, data augmentation, or using of the additional training data to enhance the model's performance and generalization capability.

REFERENCES:

TensorFlow. (2023). TensorFlow: An end-to-end open-source machine learning

platform. Retrieved from https://www.tensorflow.org/

• Chollet, F. et al. (2015). Keras: The Python Deep Learning library. Retrieved from

https://keras.io/

• Bradski, G. (2000). The OpenCV Library. Retrieved from https://opencv.org/

• SciPy: Virtanen, P., Gommers, R., Oliphant, T. E., et al. (2020). SciPy 1.0:

Fundamental Algorithms for Scientific Computing in Python. Retrieved from

https://scipy.org/