Deep Learning Final Project Report

Wildfire Prediction with Deep Learning

Introduction

Wildfires have been a global issue, with the recent increase in the frequency of wildfires it has become a devastating issue and has major issues on our ecosystems, human lives, wildlife, and economy. With the high risk of air pollution, health risks, economic loss, infrastructural damage, threat to human safety and displacement, and long-term social and psychological impacts, wildfires have been a global crisis. For the sake of our health, economy, and future, addressing the root cause of this is imperative. Keeping these effects in mind, understanding and predicting wildfire behavior is crucial for minimizing their impact and protecting lives and property.

The Wildfire Crisis: A Global Perspective

Recent statistics highlight the alarming rise in the incidence and intensity of wildfires. According to the latest data available, there were over 5,000 wildfires recorded worldwide in 2023, burning an estimated 2.5 million hectares of land. These fires resulted in the loss of 176 human lives and caused extensive damage to homes, infrastructure, and natural habitats. Prominent incidents like the 2023 Anvil Fire in Oregon, the Mililani Mauka Fire in Hawaii, the 2023 Black Bear Fire, and the 2023 Collett Ridge Fire in North Carolina serve as stark reminders of the urgency to develop effective wildfire prediction and management strategies. The devastating impacts of these wildfires shows the critical need for advanced techniques and technologies, including deep learning, to predict and mitigate the destructive effects of such natural disasters.

The Role of Deep Learning in Wildfire Prediction

Deep learning, has emerged as a powerful tool in various domains due to its ability to extract intricate patterns and make predictions from vast datasets. When applied to wildfire prediction, deep learning models exhibit significant promise in providing early warnings and reducing response times. The critical advantage of deep learning lies in its ability to analyze and interpret large-scale satellite imagery and sensor data in near-real-time, enabling the rapid detection of fire outbreaks.

The Power of Real-Time Image Analysis

One of the most exciting applications of deep learning in wildfire prediction is the analysis of real-time images uploaded to the model. With the proliferation of smartphones and the constant connectivity of the modern world, individuals can capture and share images of potential fire incidents instantaneously. By incorporating deep learning algorithms, we can harness this collective visual data to supplement traditional wildfire monitoring systems.

This project aims to leverage deep learning to develop an image classification model capable of differentiating between images of normal landscapes and those depicting wildfire events. By training the model on a vast dataset of wildfire-related images, it can learn to identify critical visual cues such as smoke plumes, flames, and scorched terrain. When users upload images via a user-friendly interface, the model can analyze them in real-time, issuing alerts and providing valuable information to authorities and first responders.

Problem Statement

Can deep learning model accurately predict wildfire growth and spread patterns from real-time images, enabling faster response and resource allocation.

Objective

The objective is to develop a deep learning model that effectively predicts wildfire behaviour based on images, aiding in proactive mitigations strategies and reducing associated damages.

Data Collection and Pre-processing

The brainstorming of the project idea started with a dataset search, first I came up with the idea of using different datasets and correlating forest loss with factors like wildfires, climate change, or human activities using the Global forest change dataset or MODIS dataset where we can use additional weather data, to include information on temperature, wind, and humidity, which affect fire behavior or vegetation and land cover data to understand fuel conditions, but due to limited time and to save the time being spent of cleaning the raw data available, the team decided to go with Kaggle dataset.

After inspecting and finalizing the dataset with the team, I considered two real-time datasets from Kaggle, exceeding 13,000 diverse images. The image diversity covers wildfires across various types of terrain, including forests, bushes, and grasslands, enhancing model generalizability. These datasets were selected for their potential to pave for highly accurate and versatile wildfire prediction.

Merging two datasets involved folder modification and image pre-processing, below is the code snippet of it.

```
from google.colab import files
files.upload()  # Use this to upload kaggle.json

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

! kaggle datasets download -d elmadafri/the-wildfire-dataset
! kaggle datasets download -d anamibnjafar0/flamevision

! unzip the-wildfire-dataset.zip
! unzip flamevision.zip
```

```
! mv /content/Classification1/the_wildfire_dataset/test/fire/Both_smoke_and_fire/* /content/Classification1/the_wildfire_dataset/test/fire
! mv /content/Classification1/the_wildfire_dataset/test/fire/Smoke_from_fires/* /content/Classification1/the_wildfire_dataset/test/fire
! mv /content/Classification1/the_wildfire_dataset/test/nofire/Fire_confounding_elements/* /content/Classification1/the_wildfire_dataset/te
! mv /content/Classification1/the_wildfire_dataset/test/nofire/Forested_areas_without_confounding_elements/* /content/Classification1/the_w
! mv /content/Classification1/the_wildfire_dataset/test/nofire/Smoke_confounding_elements/* /content/Classification1/the_wildfire_dataset/t

! mv /content/flames/flamesvision/* /content

! rm -r /content/flames
```

Similarly, all other train and valid folder manipulation was for both the dataset and merged into to single folder named classification.
The data visualization and data analysis involved creating an Excel file, the code snippet below shows the creation of an Excel file.

```
def make_excel():
    # Getting the current working directory
    cwd = os.getcwd()

    # Navigate up to the 'Classification' directory
    curr_path = os.path.dirname(cwd)
    classification_path = os.path.join(curr_path, 'Classification')

    # Creating excel folder
    excel_folder = os.path.join(classification_path, 'excel')
    os.makedirs(excel_folder, exist_ok=True)

    # Excel file path
    excel_path = os.path.join(excel_folder, 'image_dataset_info.xlsx')

    # Check if Excel file already exists
    if not os.path.exists(excel_path):
        # DataFrame to hold the information
        data = {'ID': [], 'Split': [], 'Target': []}

        # Iterating through each split and class
        for split in ['train', 'test', 'val']:
            for target in ['fire', 'nofire']:
                folder_path = os.path.join(classification_path, split, target)
                for image in os.listdir(folder_path):
                    data['ID'].append(image)
                    data['Split'].append(split)
                    data['Target'].append(target)

        # Creating a DataFrame
        df = pd.DataFrame(data)

        # Saving to Excel
        df.to_excel(excel_path, index=False)
        print("Excel file created at:", excel_path)
    else:
        print("Excel file already exists at:", excel_path)
```

```
    return excel_path
```

Data Cleaning and Pre-processing

Data cleaning is a crucial step in the data preprocessing pipeline, and it aims to ensure the dataset is accurate, consistent, and ready for analysis.

```
def data_cleaning():
    # 1. Check for Missing Values
    print("Missing values before cleaning:")
    print(data.isnull().sum())

    # Drop rows with any missing values
    data.dropna(inplace=True)

    # 2. Removing Duplicates
    data.drop_duplicates(inplace=True)

    # 3. Data Type Conversion
    # Convert 'ID' and 'Target' to string if they're not already
    data['ID'] = data['ID'].astype(str)
    data['Target'] = data['Target'].astype(str)

    # 4. Text Data Cleaning
    # Example: Ensuring all 'Target' entries are lowercase
    data['Target'] = data['Target'].str.lower()

    # 5. Data Consistency
    # Example: Standardize category names
    data['Target'].replace({'fire': 'Fire', 'nofire': 'NoFire'}, inplace=True)

    print("\nMissing values after cleaning:")
    print(data.isnull().sum())

    print("\nCleaned DataFrame:")
    print(data.head())
```

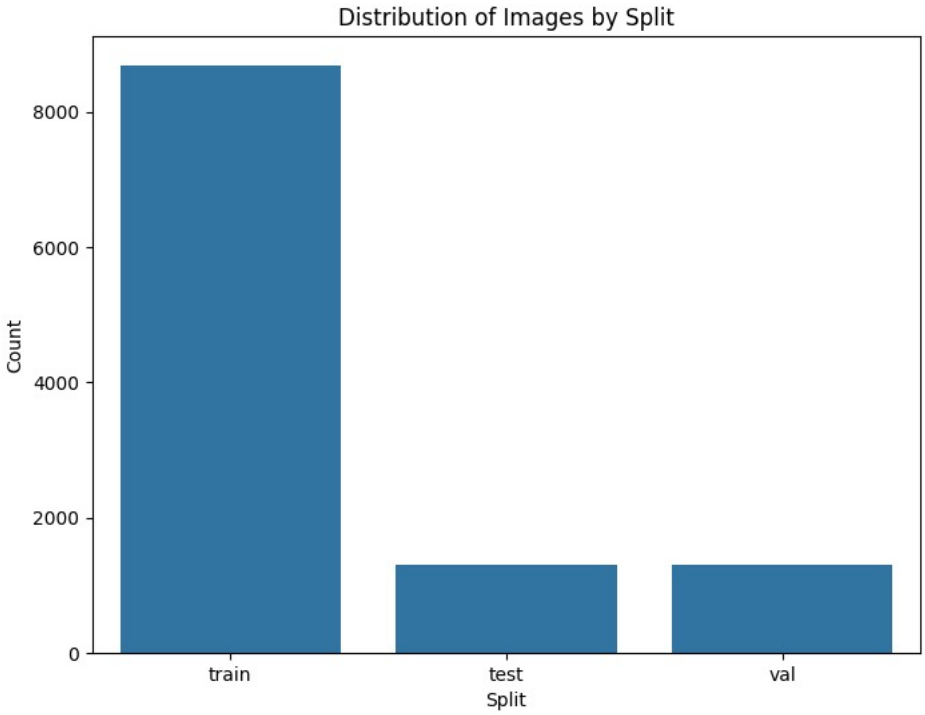
This code snippet demonstrates common data cleaning steps to prepare a dataset for analysis or modelling, including handling missing values, duplicates, data type conversion, text cleaning, and ensuring data consistency.

In the data pre-processing step, every image resized in to 224, 224, 3(colour image) according to the standard required size for the model. In our case, we used resnet50, so we resized it in to 224, 224 and normalised the every imaged passed in to the model.

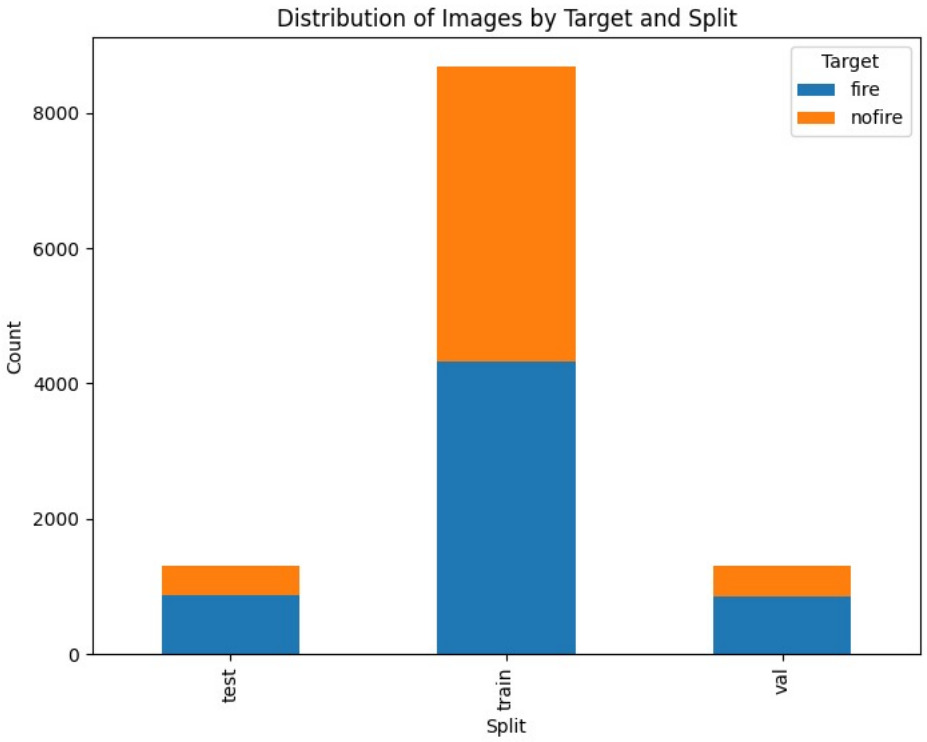
Once, the Excel file was generated, the below visualization graphs were plotted to understand the data and to go ahead with the better approach according to the nature of the data.

The above image shows the distribution of target images in train, test and valid dataset.

The above image shows the distribution of target in train dataset.



The above image shows the count of images in train, test and valid dataset.



The above image shows the count of images by the target in test, train, and valid datasets.
From the visualization plot, it's pretty obvious that the image dataset is very well balanced, hence the balancing image is not required, but to increase the robustness of the model, we have augmented the images and saved the new images in the data directory.

Data Augmentation

Image augmentation helps improve the model's generalization by introducing diversity into the training data. It reduces the risk of overfitting and enhances the model's ability to recognize variations in real-world images.

```
def gaussian_noise(image):
    """
    Add Gaussian noise to an image.
    """
    mean = 0
    var = 10
    sigma = var ** 0.5
    gaussian = np.random.normal(mean, sigma, image.shape)  # Gaussian noise
    noisy_image = np.clip(image + gaussian, 0, 255)  # Add noise and clip the values
    return noisy_image


# ImageDataGenerator with Gaussian noise
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.1,
    #channel_shift_range=20,  # Randomly shift color channels
    preprocessing_function=gaussian_noise
)

def save_augmented_images(class_name, num_images=1):
    """
    Save a specified number of augmented images for each original image in a given class.
    """
    class_dir = os.path.join(train_dir, class_name)
```

```
    images = [img for img in os.listdir(class_dir) if img.endswith((".png", "jpg", "jpeg"))]

    for img_name in images:
        img_path = os.path.join(class_dir, img_name)
        img = load_img(img_path)
        x = img_to_array(img)
        x = x.reshape((1,) + x.shape)

        # Generate and save augmented images
        for _, batch in zip(range(num_images),
                            datagen.flow(x, batch_size=1000, save_to_dir=class_dir, save_prefix='aug_' + class_name,
                                         save_format='png')):
            pass  # This loop will save 'num_images' augmented images for each original image

save_augmented_images('fire')
save_augmented_images('nofire')
```

The code snippet above applies image augmentation techniques using the `ImageDataGenerator` class. Image augmentation is a common technique in deep learning for generating new training examples by applying random transformations to existing images. The applied augmentations include rotation, width and height shifts, horizontal flipping, and the addition of Gaussian noise.

Model Architecture and Summary

The model architecture defines how the deep learning model will learn and make predictions based on the provided data. Using a pre-trained model like ResNet-50 leverages pre-learned features from a large dataset, making it suitable for transfer learning

Resnet50

ResNet-50, short for Residual Network-50, is a deep convolutional neural network architecture that is particularly well-known for its deep structure and use of residual blocks. It was introduced by Microsoft Research in 2015 and has 50 layers, including convolutional and fully connected layers. ResNet-50's innovative residual connections enable training of very deep neural networks by mitigating the vanishing gradient problem, making it a popular choice for various computer vision tasks, including image classification and object detection.

The code below defines a deep learning model architecture. It uses a pre-trained ResNet-50 model as a feature extractor, followed by several fully connected layers for classification. Batch normalization and dropout layers are added to improve model stability and reduce overfitting.

```
ResNet50_model = ResNet50(
    include_top=False,
    weights="imagenet",
    input_shape=input_shape
)

for layer in ResNet50_model.layers:
    layer.trainable = False

model = Sequential()
model.add(ResNet50_model)  # Assuming ResNet50_model is predefined
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dropout(0.7))
model.add(Dense(256, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dropout(0.6))
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dropout(0.6))
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dropout(0.6))
model.add(Dense(1, activation='sigmoid')) # For binary classification, consider using 'sigmoid'

model.summary()
```

Model Training

Model training aims to optimize model parameters to make accurate predictions. The early stopping callback prevents overfitting, and the model checkpoint callback saves the best model during training.

The code below compiles and trains the resnet50 model using the training and validation datasets. It specifies the loss function, optimizer, and evaluation metrics. Callbacks for early stopping and model checkpointing are also included.

```
model.compile(
    optimizer='sgd',
    loss='binary_crossentropy', # Note: For binary classification, this is typically used with 'sigmoid' activation
    metrics=['accuracy']
)
callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=5),
    tf.keras.callbacks.ModelCheckpoint('model.h5', save_best_only=True),
    tf.keras.callbacks.TensorBoard(log_dir='logs')
]

results = model.fit(train_data,validation_data=val_data,epochs=5,verbose = 1)
```

Model Evaluation

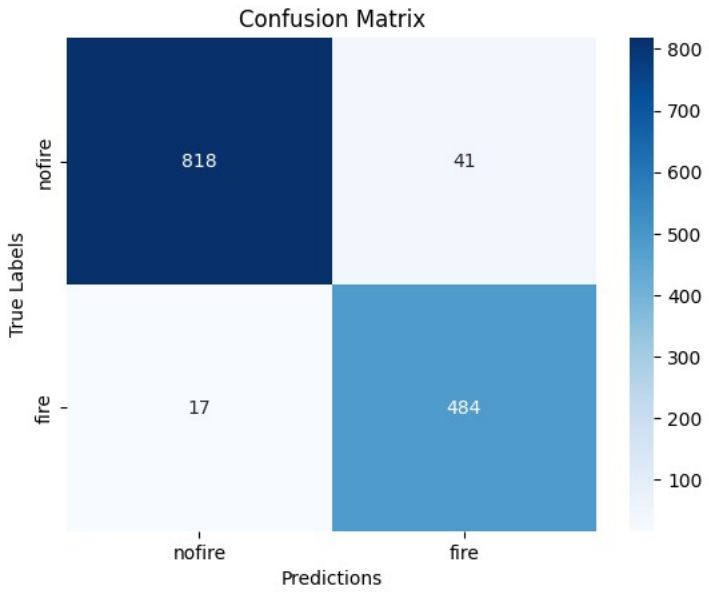
Model evaluation assesses the model's performance on unseen data to determine its effectiveness in making predictions. It provides insights into the model's strengths and weaknesses.

```
loss, acc = model.evaluate(test_data,verbose = 1)
```

Analysis of results

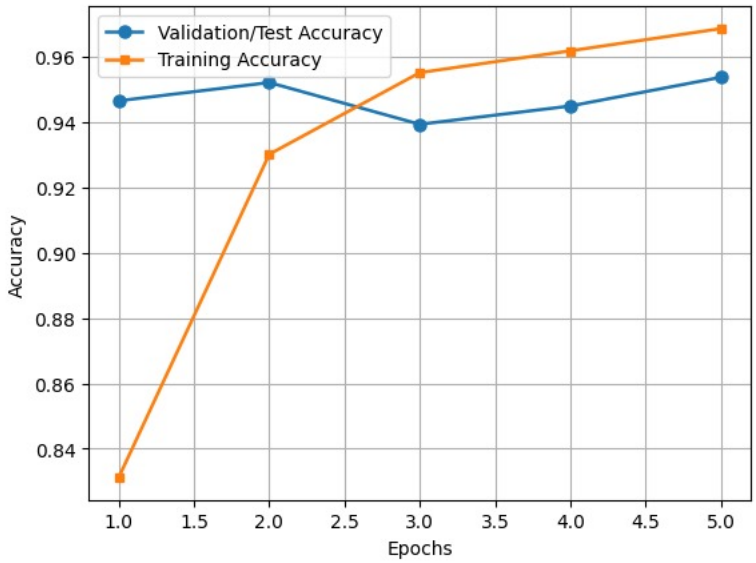
Analysis of results help in interpreting the model results and identify areas for improvement. The classification report provides detailed metrics, while the confusion matrix visualizes the model's ability to classify images correctly.

The following results were produced through modification and improvisation of the code and model over inspecting the results at every iteration.



The above image shows the confusion matrix for a classification of two models “fire” and “nofire”.

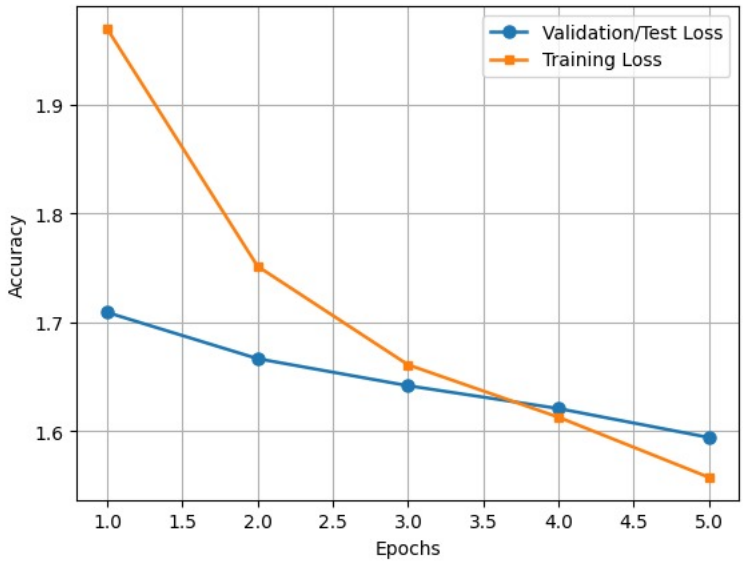
- True Negative (TN): 818 instances correctly predicted as "nofire".
- False Positive (FP): 41 instances wrongly predicted as "fire".
- False Negative (FN): 17 instances wrongly predicted as "nofire".
- True Positive (TP): 484 instances correctly predicted as "fire".



The above image shows the line graph representing accuracy over the model.

- The orange line represents the training accuracy, which is the model's performance on the data it has learned from.
- The blue line represents the validation/test accuracy, which is how the model performs on new, unseen data.

Both accuracies are plotted over the course of five epochs (a complete pass through the entire training dataset). As the number of epochs increases, both training and validation/test accuracies improve, indicating that the model is learning and generalizing well.



The above image shows the line graph representing accuracy over the model.

- The orange line indicates the training loss, which measures how well the model is fitting the training data. It shows a downward trend, meaning the model is getting better at predicting the training data over time.
- The blue line represents the validation/test loss, which measures the model's performance on a separate dataset that it hasn't learned from. This line also trends downward, suggesting that the model's ability to generalize to new data is improving.

The above results show that even though the accuracy of the resnet50 model is very high, the loss is also very high, which indicates that the model is overfitting, even with the dropout used at every layer. This indicates that the resnet50 model is very complex and a less complex model

would give better results. The model improvisation was later implemented by using vgg16 model instead of resnet50.

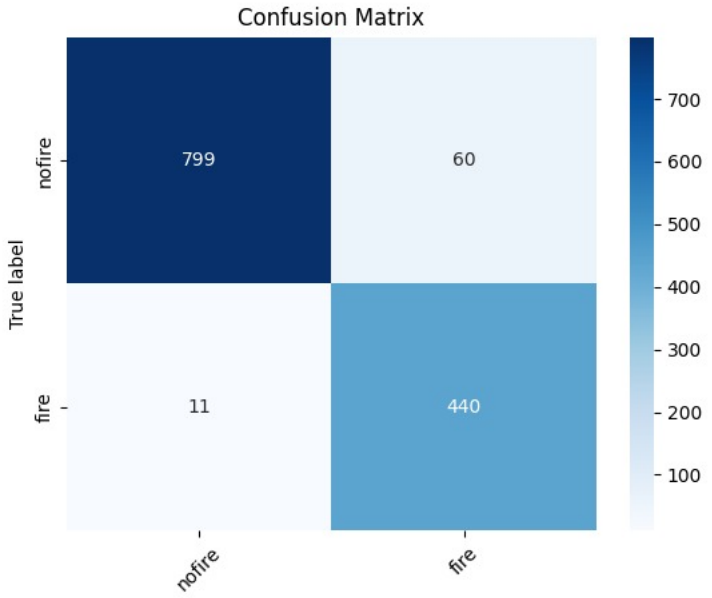
VGG16

VGG16 is a deep convolutional neural network architecture known for its simplicity and effectiveness in image classification tasks. It consists of 16 weight layers, including convolutional and fully connected layers, and is characterized by its use of small 3x3 convolutional filters and max-pooling layers, making it a popular choice for feature extraction in computer vision applications.

The below code shows the implementation of vgg16 model using the Adam optimizer and categorical cross-entropy loss function with callbacks for early stopping, model checkpointing, and TensorBoard logging. The model is trained using the training and validation datasets for a specified number of epochs. Model performance is evaluated on the test dataset, and accuracy and loss metrics are computed.

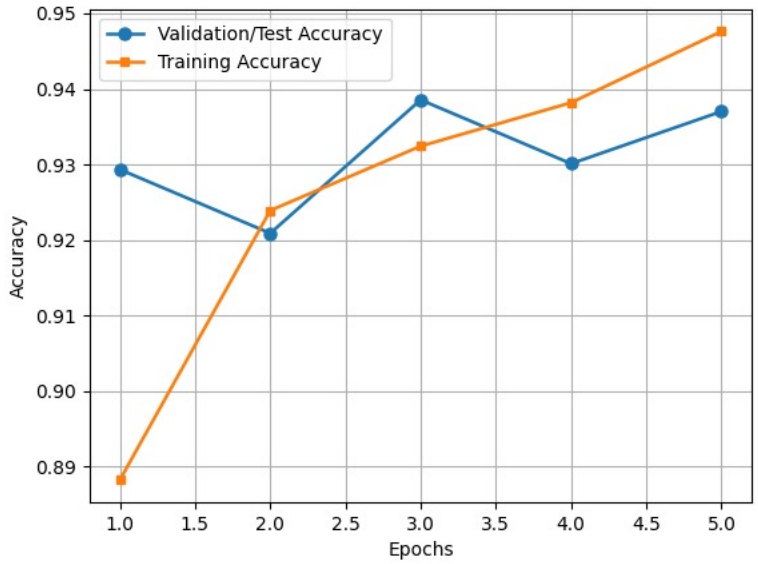
```
VGG16_model = VGG16(
        include_top = False,
        weights="imagenet",
        input_shape = input_shape
    )
    for layer in VGG16_model.layers :
        layer.trainable = False

    from keras import activations
    model = Sequential()
    model.add(VGG16_model)
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2,2)))
    model.add(Flatten())
    model.add(Dense(512,activation = 'relu'))
    model.add(Dropout(0.3))
    model.add(Dense(256,activation = 'relu'))
    model.add(Dense(128,activation = 'relu'))
    model.add(Dense(num_classes,activation = 'softmax'))
    model.summary()

    model.compile(
        optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy']
    )
    callbacks = [
        tf.keras.callbacks.EarlyStopping(patience=5),
        tf.keras.callbacks.ModelCheckpoint('model.h5',save_best_only=True),
        tf.keras.callbacks.TensorBoard(log_dir='logs')
        ]


    results = model.fit(train_data,validation_data=val_data,epochs=5,verbose = 1)
    loss, acc = model.evaluate(test_data,verbose = 1)
    model_save_path = os.path.join(code_path, 'model.h5')
    model.save(model_save_path)
    print(f"Model saved to {model_save_path}")
```

Further the metrics are calculated and confusion matrix is being calculated to understand the correctness of the model.

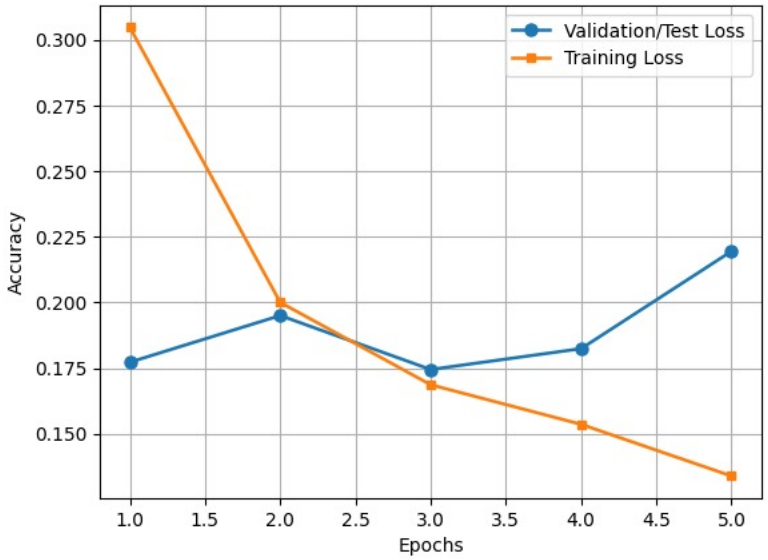


The above image is a confusion matrix from a vgg16 model, showing the classification results for a binary "fire" vs. "nofire" scenario. The matrix indicates 799 true negatives (correct nofire predictions), 440 true positives (correct fire predictions), 60 false positives (fire incorrectly predicted when there was nofire), and 11 false negatives (nofire incorrectly predicted when there was fire). This suggests a model with high accuracy in classifying both fire and nofire instances, although with some errors.

Further, we plotted line graphs to analyse the accuracy and loss of the model for better understanding of the model.



The graph shows the training and validation/test accuracy of vgg16 model across 5 epochs. The orange line represents the training accuracy, and the blue line represents the validation/test accuracy. Both accuracies start around 0.9, but while the training accuracy increases and stabilizes around 0.94, the validation/test accuracy initially drops, then surpasses the training accuracy, peaking around epoch 3 before ending slightly below the training accuracy at epoch 5. This pattern suggests the model is learning and generalizing well without significant overfitting by epoch 5.



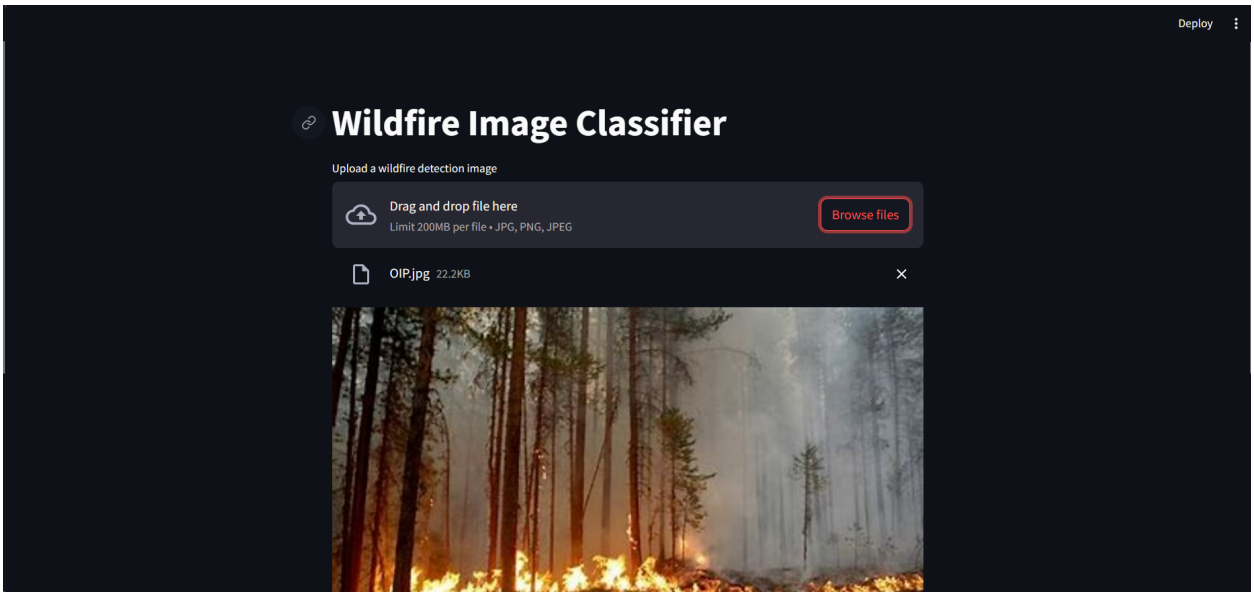
The image displays a graph of training and validation/test loss for vgg16 model over 5 epochs. The orange line indicates the training loss, which consistently decreases, reflecting the model's improving accuracy on the training data. The blue line shows the validation/test loss, which decreases sharply and then levels off, indicating the model's improving generalization to new data. The decreasing trend in loss for both lines suggests effective learning and model improvement over time.

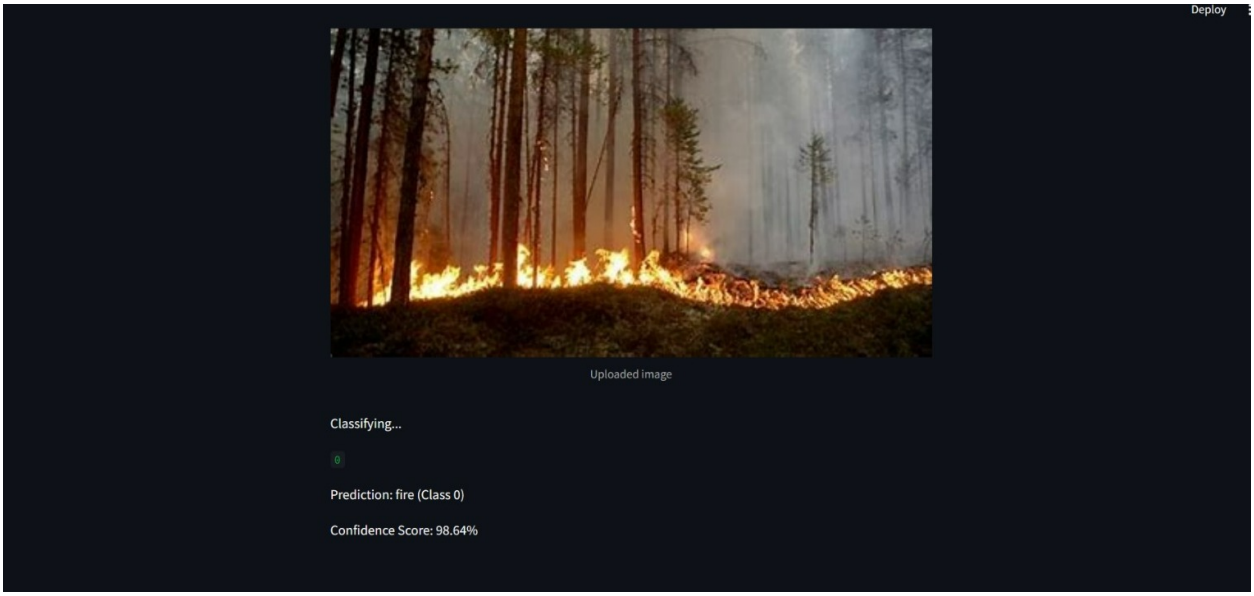
This shows that the resnet50 model shows a decreasing trend in both training and validation/test loss over epochs, which suggests the model is learning effectively. The second graph has a steeper decline in training loss compared to the validation/test loss, which could indicate potential overfitting as the model learns the training data too well and may not generalize as effectively. However, in both cases, the validation/test loss decreases, which is a positive indicator of the model's ability to perform well on unseen data. Overall, the model's loss is reducing as it trains, but the rate of reduction and the gap between training and validation/test loss can inform about its generalization capabilities.

Between the two model, the vgg16 model represents a better model due to the consistently lower values of both training and validation/test loss across all epochs. This indicates that the model is not only learning well but also generalizing better to new data. A lower loss is typically a sign of a better predictive model as it suggests smaller discrepancies between the predicted values and the true values.

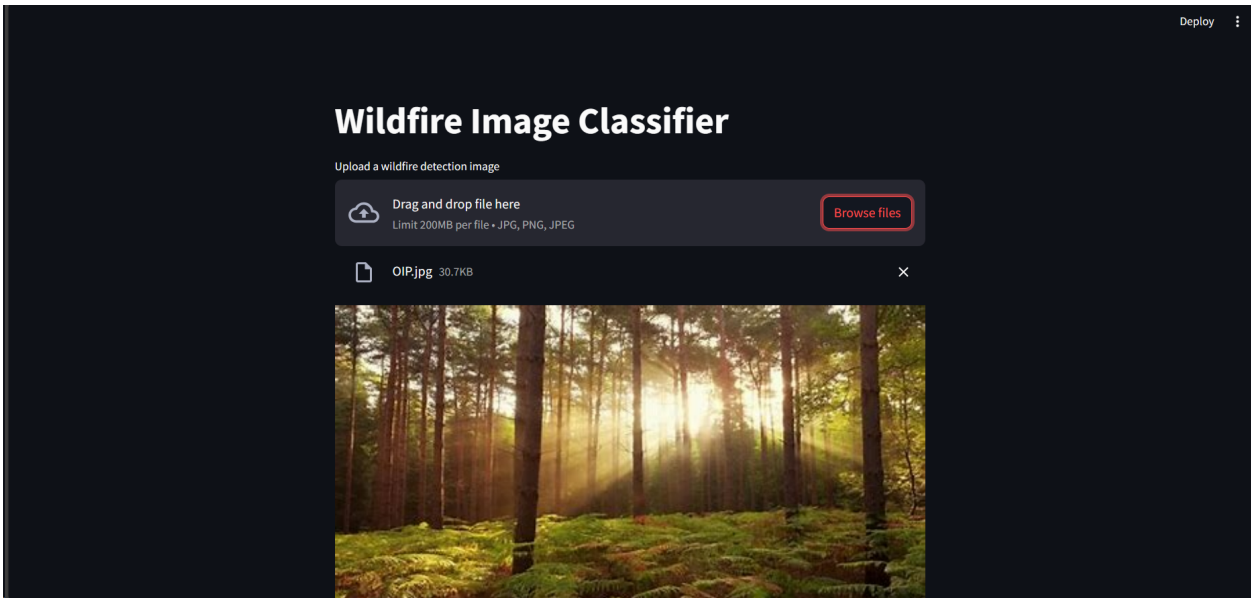
Implementation

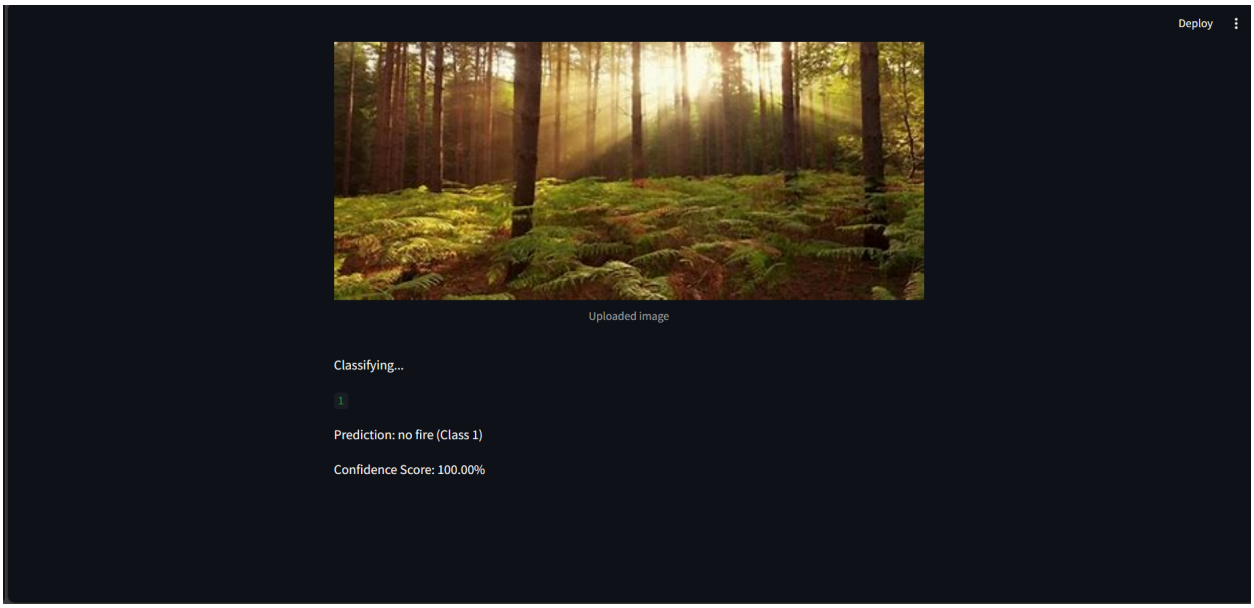
The Streamlit web application developed for the project enables users to upload images and utilizes a trained deep learning model to predict the presence of wildfires. The model discerns between 'fire' and 'no fire' images based on its learned characteristics from the training data, demonstrating the application of AI in real-time environmental monitoring.





In the above image we can see that, the model is predicting fire for the forest **“fire”** image with the confidence score of 98.64%. Similarly in the image below, for the random sunlight image on forest from the google also model is predicting it right as “**nofire**” with the confidence score of 100%.





Summary and Conclusion

This details a deep learning project aimed at predicting wildfires using image classification. It underscores the severity of wildfires and the potential of deep learning, specifically ResNet-50 and VGG16 architectures, to analyze real-time images for early detection. The project involved data collection, pre-processing from Kaggle, and model training, with emphasis on balancing the dataset and augmenting images for robustness. The models were evaluated using accuracy and loss metrics, and VGG16 was identified as the superior model due to its lower loss, indicating better generalization and predictive performance. Also, from the implementation demo we can say that the application's deep learning model showcases high accuracy in its predictions, confirming the presence of fire in forest images with a confidence score of 98.64% and correctly identifying non-fire scenarios with absolute certainty. This reflects the model's robust learning from the training dataset and its capability to generalize well to new, unseen data.

Future Scope and Improvements

Future improvements for this wildfire prediction project could involve integrating multispectral and temporal data analysis to capture a wider range of fire signatures. Enhancements in real-time data processing and edge computing could facilitate quicker response times. Collaboration

with IoT for sensor-based data and application of ensemble learning models may also lead to more robust predictions. Continuous model training with up-to-date and region-specific data will likely improve accuracy, and the development of a decentralized data collection framework could enable community-driven data enrichment.

References

1. **Forest Fire Detection Using Satellite Imagery**, https://www.ijnrd.org/papers/IJNRD2305193.pdf
2. **A survey on Image Data Augmentation for Deep Learning,** https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0
3. **Streamlit Prophet,** https://github.com/artefactory/streamlit_prophet
4. **Fire-Net: A Deep Learning Framework for Active Forest Fire Detection,** https://www.hindawi.com/journals/js/2022/8044390/
5. **Deep Learning Based Forest Fire Classification and Detection in Satellite Images,** https://ieeexplore.ieee.org/document/9087309