Pooja Chandrashekara 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.

Description and Results

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
```

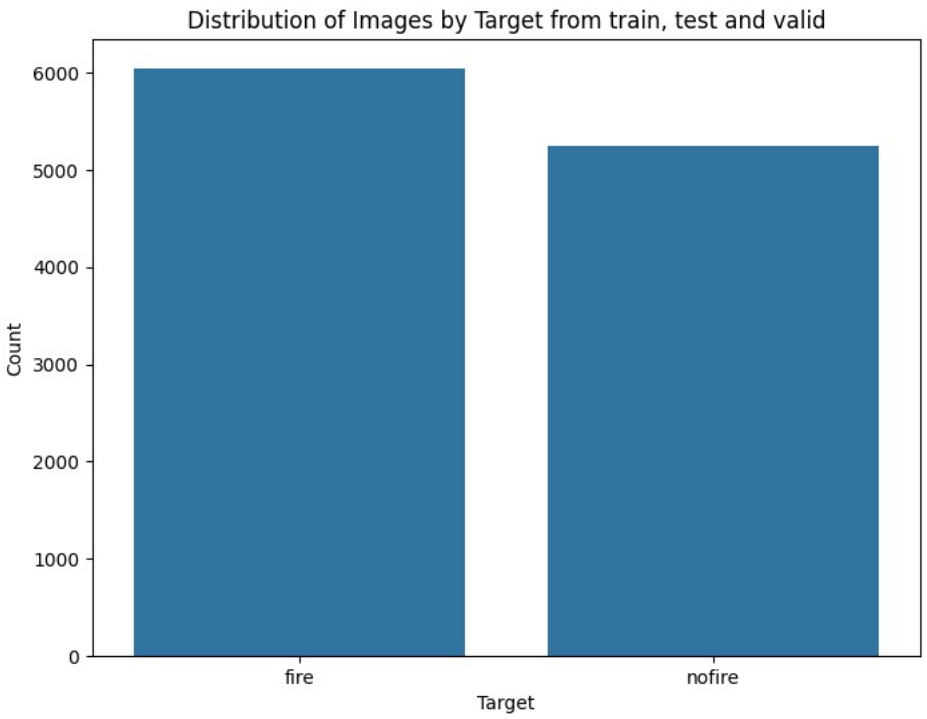
```
! mv /content/Classification1/the_wildfire_dataset/test/fire/Both_smoke_and_fire/* /content/Classification1/the_wildfire_dataset/test/fire
! mv /content/Classification1/the_wildfire_dataset/test/fire/Smoke_from_fires/* /content/Classification1/the_wildfire_dataset/test/fire
! mv /content/Classification1/the_wildfire_dataset/test/nofire/Fire_confounding_elements/* /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.

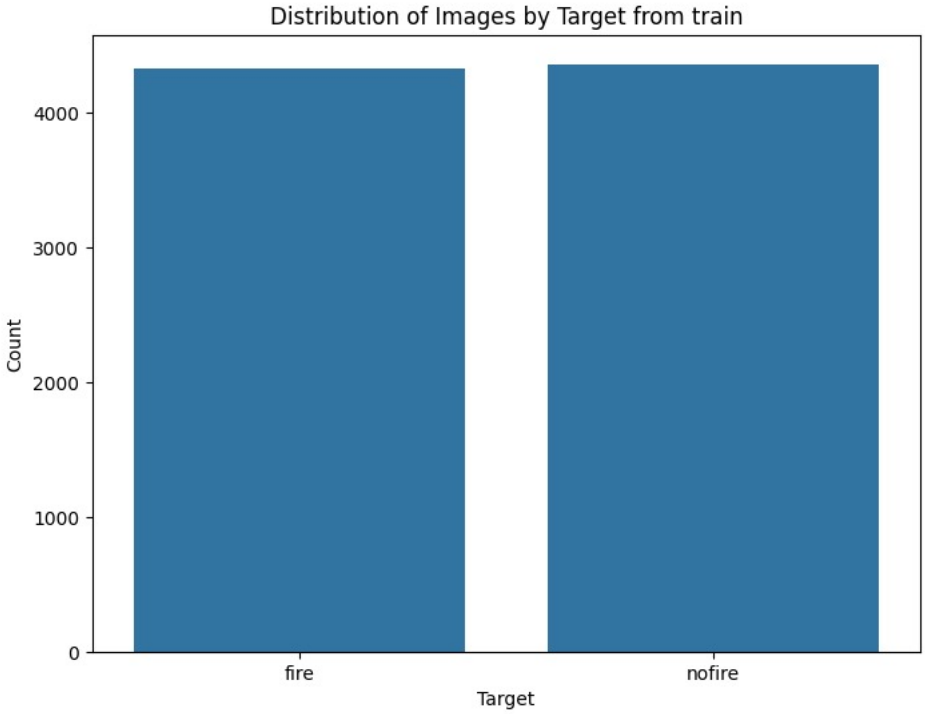
```
def make_excel():
    # Getting the current working directory
    cwd = os.getcwd()

    # 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
```

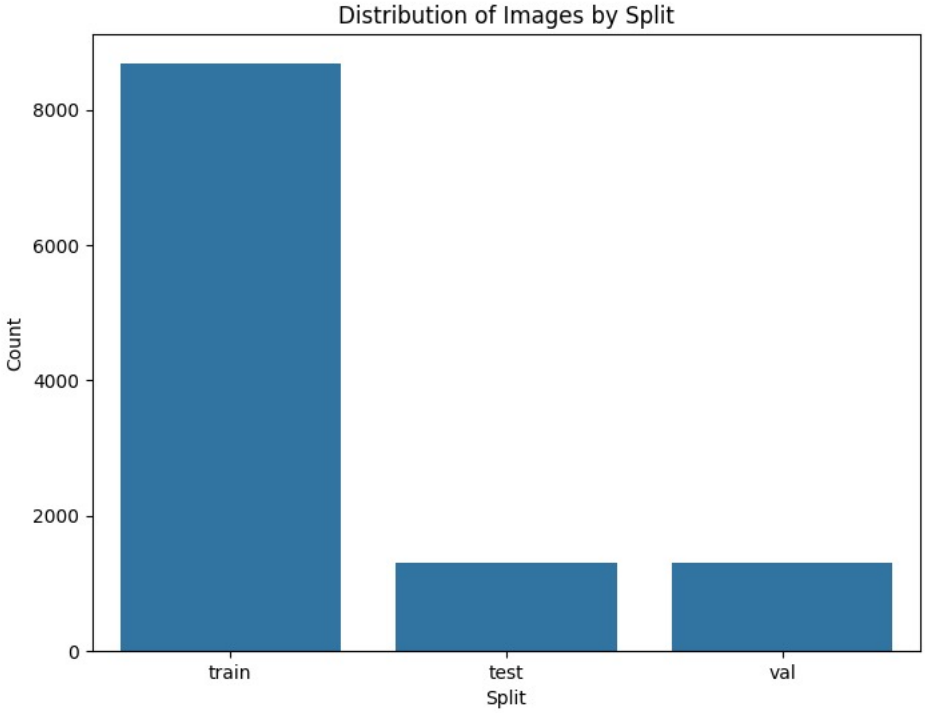
Once, the Excel file was generated, the below visualization graphs were plotted to understand the data and to go ahead with the better approach according to the nature of the data.



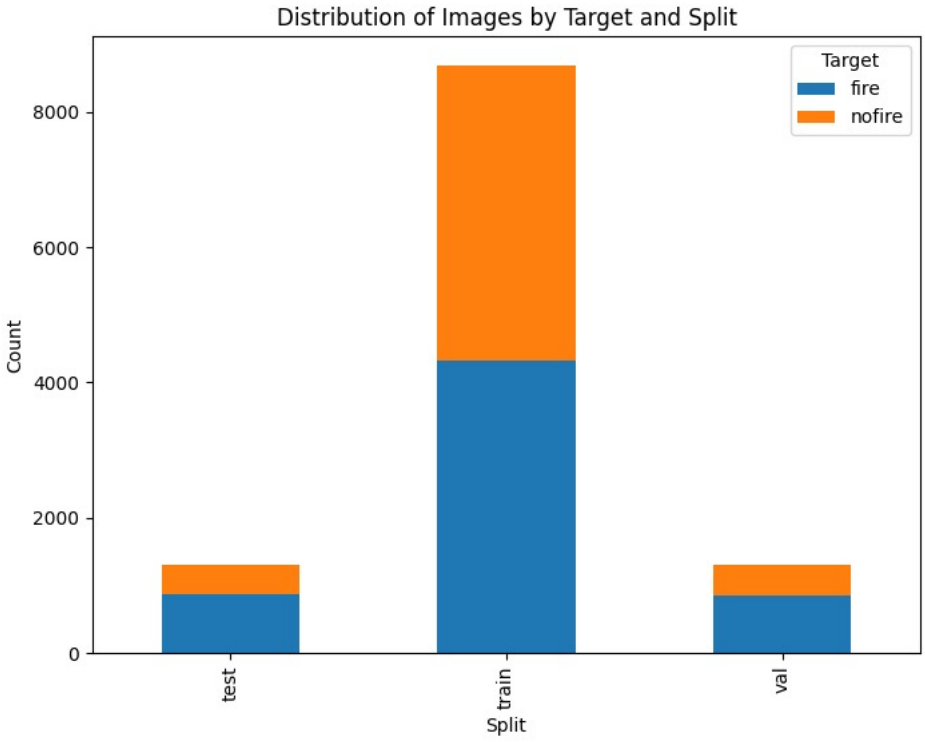
The above image shows the distribution of target images in train, test and valid dataset.



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 I got, 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, I augmented the images and saved the new images in the data directory. The code snippet below shows the data augmentation.

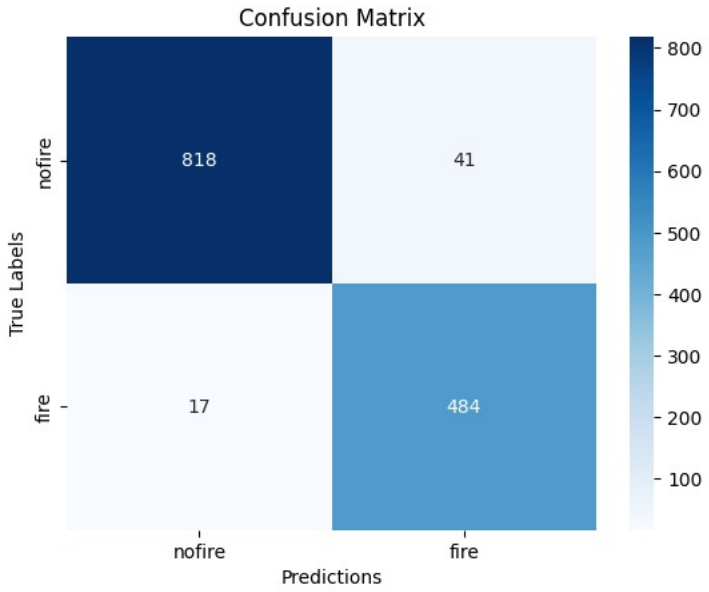
```
def gaussian_noise(image):
    """
    Add Gaussian noise to an image.
    """
    mean = 0
    var = 10
    sigma = var ** 0.5
    gaussian = np.random.normal(mean, sigma, image.shape)  # Gaussian noise
    noisy_image = np.clip(image + gaussian, 0, 255)  # Add noise and clip the values
    return 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 augmented images were then fed pre-processed for an image size of 224, 224 with 3 channels since the images are colored images and then fed to the model of resnet50. The code snippet below shows the resnet50 model implemented on the images, with a dropout of 0.6 at each layer because of the biased results upon overfitting.

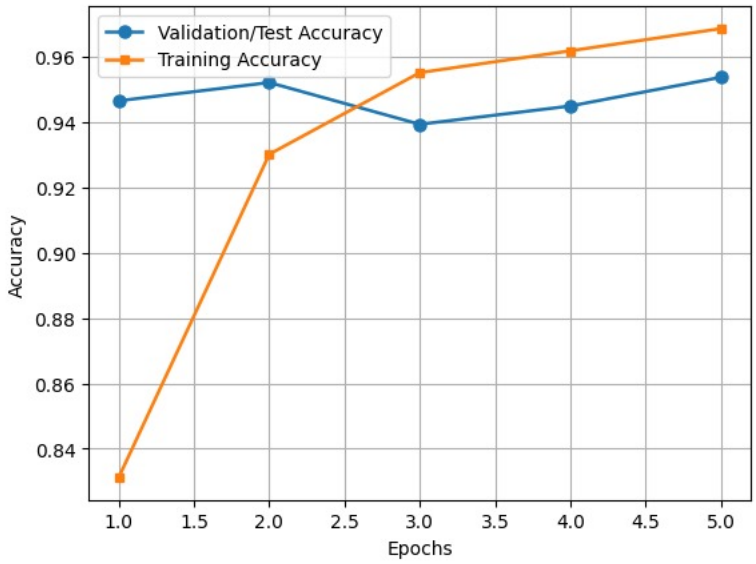
```
ResNet50_model = ResNet50(
    include_top=False,
    weights="imagenet",
    input_shape=input_shape
)

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()
```

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”.

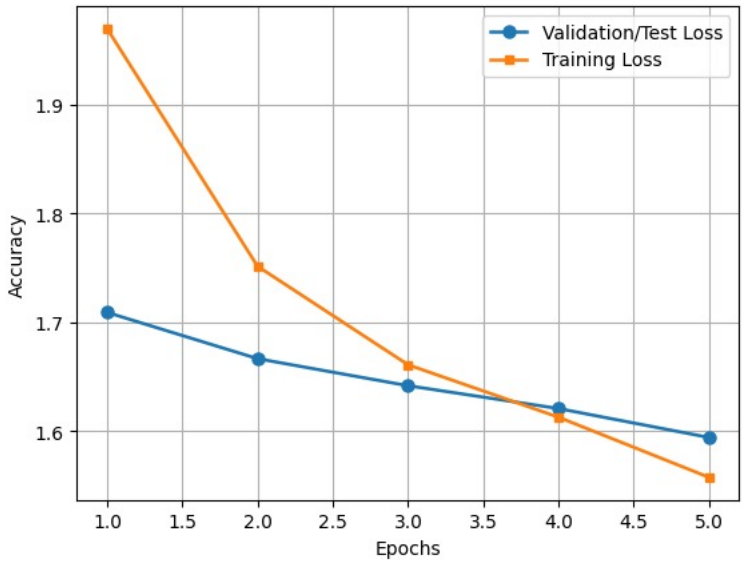
- True Negative (TN): 818 instances correctly predicted as "nofire".
- False Positive (FP): 41 instances wrongly predicted as "fire".
- False Negative (FN): 17 instances wrongly predicted as "nofire".
- True Positive (TP): 484 instances correctly predicted as "fire".



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 my teammate which produced better results.

Summary

The project began with a dataset search. Initial ideas included using datasets correlating forest loss with wildfires, climate change, or human activities, and incorporating weather data to affect fire behaviour predictions. However, due to time constraints and the complexities of data pre-processing, I opted for a readily available and labelled dataset from Kaggle, comprising over 13,000 images of various terrains affected by wildfires.

Two Kaggle datasets were merged, requiring folder modification and image pre-processing, for which the report provides Python code snippets. An excel file was created for data visualization and analysis, and further Python code is provided to generate this file.

The data visualization revealed a well-balanced dataset, making additional balancing unnecessary. To enhance the model's robustness, I performed image augmentation and fed the augmented images into a ResNet50 model with significant dropout to prevent overfitting.

Despite high accuracy, the ResNet50 model exhibited high loss, suggesting overfitting. This concludes that a less complex model might yield better results, and mentions that subsequent improvements by a teammate led to more favourable outcomes.

Conclusion

The outcome was a set of models trained using the ResNet50 architecture, which, despite high accuracy, suffered from overfitting as indicated by the high loss rates. The confusion matrix and accuracy graphs highlighted the model's high true positive and true negative rates, but also underlined the need for a less complex model to reduce overfitting.

The results led to the conclusion that a simpler model may yield better generalizability without compromising accuracy. The report implies that further model improvement and experimentation were undertaken by another team member, which ultimately enhanced the model's performance.