Project Report: Fire, Smoke, and Non-Fire Detection Using Deep Learning

1. Project Overview

The goal of this project is to build and deploy a deep learning-based model that can detect and classify images of fire, smoke, and non-fire. The model can be utilized in real-world applications like monitoring forest fires or detecting smoke-related hazards in real-time.

---

2. Dataset

The dataset used in this project is the **FOREST_FIRE_SMOKE_AND_NON_FIRE_DATASET**, which consists of labeled images of three categories:

- **Fire**
- **Smoke**
- **Non-fire (Neutral)**

The dataset is organized into subfolders for each class (fire, smoke, non-fire), and the images are stored in a readable format such as JPEG or PNG.

3. Data Preprocessing

The following preprocessing tasks were performed to ensure the data is ready for training:

- **Corrupted Image Removal:** Images that could not be loaded or were not in the JFIF format were removed.
- **Image Augmentation:** To increase the diversity of the dataset and reduce overfitting, augmentation techniques were applied, including random rotations, shifts, zooms, and horizontal flips.

4. Model Architecture

The architecture of the model was designed using Convolutional Neural Networks (CNNs). A custom CNN model was constructed with multiple convolutional layers followed by max-pooling layers. Dropout layers were introduced to mitigate overfitting.

The architecture of the model is as follows:

1. **Conv2D Layer:** 96 filters with a 3x3 kernel size.
2. **MaxPooling Layer:** Pooling with a 2x2 kernel.
3. **Conv2D Layers:** Additional convolution layers with 256 filters, followed by max-pooling layers.
4. **Fully Connected Layer (Dense):** With 256 units.

5. **Output Layer:** A final dense layer with 3 units, corresponding to the 3 classes (Fire, Smoke, Non-fire), using a softmax activation function.

The model was compiled using **categorical cross-entropy** as the loss function and **accuracy** as the evaluation metric.

5. Hyperparameter Tuning

To improve model performance, **hyperparameter optimization** was carried out using **Keras Tuner** and **Random Search**. The main hyperparameters tuned were:

- Number of filters in each convolutional layer
- Learning rate
- Dropout rate
- Number of units in the final dense layer

6. Transfer Learning (ResNet50)

In addition to training a custom CNN model, transfer learning was employed to leverage pre-trained models. Specifically, **ResNet50** was fine-tuned for the fire and smoke classification task. The ResNet model was used with the pre-trained weights and modified for three output classes.

The final architecture used for ResNet50 consisted of:

- **Pretrained ResNet50 Model:** For feature extraction.
- **Fully Connected Layer:** A dense layer with 128 units followed by an output layer with 3 units for classification.

7. Model Training and Evaluation

The models were trained using the preprocessed dataset, with training and validation sets separated. The training was performed with the following settings:

- **Batch Size:** 16 for CNN models, 64 for ResNet50.
- **Epochs:** 20 epochs for the custom CNN model, 100 epochs for ResNet50.
- **Optimizer:** Adam optimizer with a learning rate of 0.0001.

The model's performance was evaluated using **accuracy** and **loss** metrics, which were plotted for training and validation data. The models were saved after training to retain the best performing versions.

8. Model Deployment

After training the models, they were deployed for real-time inference. The models can be integrated into systems that monitor video feeds or images for fire and smoke detection. The following deployment methods were demonstrated:

- **Image Prediction:** The models were used to classify individual images from a dataset.
- **Video Prediction:** The models were applied to video frames, and predictions were displayed in real-time.

9. Model Results

The results showed high performance in terms of classification accuracy, particularly after the fine-tuning of hyperparameters. The **ResNet50** model achieved a validation accuracy of around 93%.

10. Code Walkthrough

a. Data Preprocessing

- **Corrupted Image Removal:** The code iterates over training and validation directories and checks if each image is in the correct format (JFIF). Corrupted images are removed automatically.

b. Model Definition

- A CNN model is built using Keras layers like `Conv2D`, `MaxPool2D`, `Flatten`, `Dropout`, and `Dense`. The model is then compiled using the Adam optimizer and categorical cross-entropy loss function.

c. Data Augmentation

- The `ImageDataGenerator` is used to apply random transformations to images during training to improve model generalization.

d. Training and Hyperparameter Tuning

- The model is trained using the `fit()` function, with callbacks for early stopping and model checkpointing to save the best model.
- Hyperparameter tuning is conducted using **Keras Tuner** with a **RandomSearch** approach.

e. Transfer Learning

- **ResNet50** is used as a pre-trained model. The fully connected layers are modified to accommodate the fire, smoke, and neutral categories.

f. Real-Time Prediction

- For image and video prediction, a preprocessing pipeline transforms input images into the required format for the model, followed by classification using the trained model.

11. Challenges and Limitations

- **Dataset Imbalance:** If the dataset has more images of one class (e.g., Fire), the model might be biased towards predicting that class. Balancing the dataset or applying class weights could help.
- **Real-time Inference:** While the model performs well on static images, processing video frames in real-time can require optimization of both the model and the infrastructure (e.g., using GPUs).

12. Conclusion

The project successfully built a model capable of classifying images and videos into fire, smoke, and neutral categories. Transfer learning with **ResNet50** helped improve accuracy and performance. The model is now ready for deployment in real-world applications, such as automated surveillance systems for fire detection.

---

13. Future Work

- **Improve Data Augmentation:** More augmentation techniques, such as brightness and contrast adjustments, could be explored to improve generalization.
- **Model Optimization:** Further model optimization can be done using techniques like quantization or pruning for faster inference in real-time applications.
- **Deploy on Edge Devices:** The model can be deployed on edge devices for real-time monitoring and classification, using frameworks like TensorFlow Lite or PyTorch Mobile.

14. References

- TensorFlow Documentation: https://www.tensorflow.org/api_docs
- Keras Documentation: https://keras.io/
- PyTorch Documentation: https://pytorch.org/docs/stable/
- Keras Tuner Documentation: https://keras-team.github.io/keras-tuner/

‘