Exploring the Landscape: Analysis of Model Results on Various Convolutional Neural Network Architectures for iRESPOND System

Freddie Prianes[1], Kaela Marie Fortuno[2], Rosel Onesa[3], Brenda Benosa[4], Thelma Palaoag[5], Nancy Flores[6]
College of Computer Studies, Camarines Sur Polytechnic Colleges, Nabua, Camarines Sur, Philippines[1, 2, 3, 4]
College of Information Technology and Computer Science, University of the Cordilleras, Baguio City, Philippines[5, 6]

Abstract—**In the era of rapid technological advancement, the integration of cutting-edge technologies plays a pivotal role in enhancing the efficiency and responsiveness of critical systems. iRESPOND, a real-time Geospatial Information and Alert System, stands at the forefront of such innovations, facilitating timely and informed decision-making in dynamic environments. As the demand for accurate and swift responses, the role of CNN models in iRESPOND becomes significant. The study focuses on seven prominent CNN architectures, namely EfficientNet (B0, B7, V2B0, and V2L), InceptionV3, ResNet50, and VGG19 and with the integration of different optimizers and learning rates. The methodology employed a strategic implementation of looping during the training phase. This iterative approach is designed to systematically re-train the CNN models, emphasizing identifying the most suitable architecture among the seven considered variants. The primary objective is to discern the optimal architecture and fine-tune critical parameters, explicitly targeting the optimizer and learning rate values. The differential impact of each model on the system's ability is to discern patterns and anomalies in the image datasets. ResNet50 exhibited robust performance showcasing suitability for real-time processing in dynamic environments with a better accuracy result of 95.02%. However, the EfficientNetV2B0 model, characterized by its advancements in network scaling, presented promising results with a lower loss of 0.187. Generally, the findings not only contribute valuable insights into the optimal selection of architectures for iRESPOND but also highlight the importance of fine-tuning hyperparameters through an iterative training approach, which paves the way for the continued enhancement of iRESPOND as an adaptive system.**

Keywords—Artificial intelligence; image classification; emergency response; model training; optimizers; learning rate

I. Introduction

In the contemporary landscape of rapid technological progress, integrating state-of-the-art technologies has become intrinsic to augmenting the efficiency and responsiveness of critical systems [1]. Within this realm of innovations, iRESPOND stands out as a real-time Geospatial Information and Alert System, assuming a pioneering role in technology to facilitate timely and informed decision-making in dynamic and unpredictable environments. The escalating need for precision and rapidity in addressing geospatial challenges accentuates the crucial role played by Convolutional Neural Network (CNN) models within the iRESPOND system.

As a complex and dynamic system, iRESPOND relies on advanced computational models to process and analyze geospatial data efficiently. CNNs, a specialized class of deep neural networks designed for image analysis, emerge as pivotal components that significantly enhance iRESPOND's capability to discern intricate patterns and anomalies within vast image datasets [2]—recognizing the significance of these EfficientNet (B0, B7, V2B0, and V2L, Google Inception CNN 3rd Edition (InceptionV3), Residual Network – 50 Layers Deep (ResNet50), and Visual Geometry Group – 19 Convolutional Layers (VGG19).

EfficientNet is a revolutionized model scaling that proposes a compound scaling method that balances depth, width, and resolution. EfficientNetB0 represents the baseline model, while EfficientNetB7 is a larger variant [3]. These models achieve state-of-the-art performance with fewer parameters, making them efficient and scalable for various applications [4]. The compound scaling ensures the models are optimized across multiple dimensions, providing a favorable trade-off between accuracy and computational efficiency [5].

Building upon the success of EfficientNet, EfficientNetV2 refines the original architecture. EfficientNetV2B0 and EfficientNetV2L are variants designed for improved performance. The advancements focus on improved training stability and robustness [6]. EfficientNetV2 introduces novel architectural choices, such as a new stem and a more efficient inverted bottleneck structure, contributing to enhanced generalization and efficiency [7].

InceptionV3 is part of the Inception family of CNN architectures. Notable for its inception modules, which incorporate multiple filter sizes within the same layer, InceptionV3 captures hierarchical features at different scales [8]. The inception architecture aims to balance computational efficiency and representation capacity, making it suitable for various computer vision tasks [9].

ResNet introduced the concept of residual learning, addressing the vanishing gradient problem in deep neural networks [10]. ResNet50, a variant with 50 layers, has become a benchmark architecture known for its deep, skip-connection design, allowing for the training of profound networks [11]. The skip connections facilitate the flow of gradients during backpropagation, enabling the successful training of deep networks without degradation in performance [12].

The VGG architecture is characterized by simplicity and uniformity [13]. VGG19, an extended version with 19 layers, features convolutional layers with small 3x3 filters and max-pooling layers [14]. While computationally intensive, VGG architectures are known for their excellent performance in image classification tasks, demonstrating the importance of depth in CNNs [15].

The scope of exploration of this study extends not only to the CNN, as mentioned above architecture, but it also encompasses various optimizers, i.e., Adaptive Moment Estimation (Adam) and Root Mean Squared Propagation (RMSProp) and learning rates, with the overarching goal of identifying the most effective combination to optimize the performance of iRESPOND.

Adam combines the advantages of adaptive learning rate methods and momentum-based optimization [16]. It maintains two moving average estimators: the first moment (mean) of the gradients (similar to momentum) and the second moment (uncentered variance) of the gradients. These estimates are then used to adjust the learning rates for each parameter adaptively [17]. Adam computes individual adaptive learning rates for each parameter, allowing for practical training across different dimensions and reducing the need for manual tuning of the learning rate hyperparameter [18]. This adaptability to different gradients and learning rates makes Adam well-suited for various tasks and architectures [19].

RMSprop addresses some limitations of traditional gradient descent algorithms, particularly the sensitivity of learning rates to the scale of gradients in different dimensions of training [20]. RMSprop modifies the learning rate for each parameter based on the average of recent squared gradients [21]. By scaling the learning rates inversely proportional to the square root of these averages, RMSprop effectively adapts the learning rates for each parameter independently [22]. This adaptive adjustment helps mitigate the exploding and vanishing gradient problems, increasing stability and efficiency [23].

In general, these diverse arrays of architectures and optimizers have significantly impacted the field of computer vision and image analysis. Each brings unique characteristics, design principles, and innovations to deep learning, contributing to various applications, including image recognition, object detection, and many others.

II. Related Works

The deliberate use of deep learning techniques exemplifies a larger trend in disaster management wherein machine learning approaches are becoming more popular due to its ability to handle complex and dynamic datasets [24, 25]. Despite the potential for deep learning algorithms to enhance accuracy, concerns persist regarding their resource-intensive nature and inefficiency in real-time monitoring applications [26]. Rathod et al. study highlights the efficacy of CNN-based models in getting better accuracy for disaster image classification, but it also shows how little foundation has been laid for establishing a robust computerized system for disaster response and recovery management [27].

According to Shah et al., traditional disaster classification methods lack in precision and speed which are essential for quick decision-making and resource allocation during emergencies. Challenges in data protection, latency transport, and unified-controlled data storage make disaster classification system implementation even more difficult [28]. Hence, exploring the efficacy of transfer learning techniques becomes imperative to address data scarcity issues and bolster model performance in deep learning scenarios, particularly where datasets are limited [29].

Moreover, the study of Asif et al. underscores the potential of neural network-based image processing architectures in enhancing crisis-related operations. However, the authors also acknowledge the limitations in evaluating activities, contexts, and related images during emergencies and disasters. Similarly, Tang et al. point out the shortcomings of existing forest classification algorithms based on graphics analysis, while Kallas & Napolitano, and Daly & Thom works also highlight the challenges in sub-classifying complex structural damage types and recognizing fire and smoke in images respectively [26, 30-32]. Subsequently, Mukhopadhyay et al. emphasizes the necessity for future research in emergency prediction to assess the accuracy of prediction models thoroughly, necessitating additional modeling and empirical studies to comprehend method advantages and drawbacks fully [33].

Navigating these challenges reveals promising outcomes in developing emergency and disaster-related models. Sharma, Jain, and Mishra stress the importance of testing CNNs across multiple datasets to unveil their true potential and limitations. Although they observed superior performance by GoogLeNet and ResNet50 compared to AlexNet in object recognition precision within images, significant performance variations persist across different object categories [34]. In line with these results, Zainorzoli et al., and Sushma & Lakshmi affirm ResNet50 as the highest accuracy achieved among tested models. Comparative evaluations against popular CNN architectures like AlexNet, GoogLeNet, VGG16, and VGG19 consistently position ResNet50 as a better choice, displaying higher precision and reliability in object recognition across diverse datasets and applications, particularly in emergency incident image classification scenarios. [35, 36]

The collective body of related studies contributes diverse approaches and applications to the disaster prediction and response domain, spanning advanced machine learning models to innovative technological solutions. However, addressing complex challenges and bridging gaps in diverse image datasets for various emergency and disaster classifications, achieving higher prediction accuracy rates, and real-time processing of incident reports in disaster response and mitigation necessitate further research and collaborative efforts. Thus, the contributions of these studies are important in introducing a CNN model custom-made for the iRESPOND system.

III. Methods

Achieving optimal performance for image classification using CNN models requires close attention to detail. A systematic process was used in a specialized repository to construct and optimize a CNN model. Robust experimentation was initiated by first changing global variables essential for training the model, such as seed, epochs, learning rates, and base model selection that has already been trained [37]. The subsequent processes were carried out precisely to guarantee a thorough comprehension of the model's behavior and performance, from dataset preparation to model creation and evaluation.

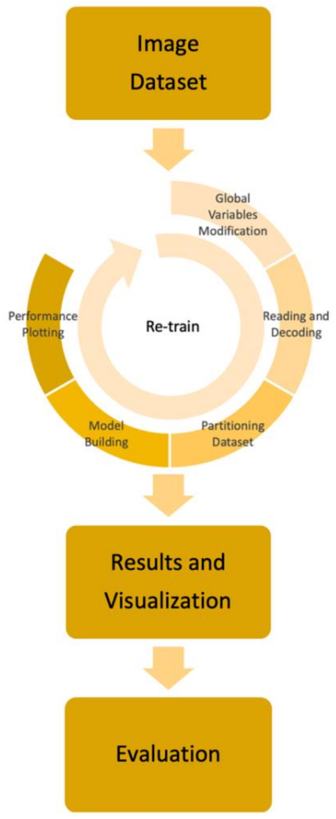


Fig. 1. Re-training process.

In analyzing the performance of the CNN models for image classification, as shown in Fig. 1, a systematic approach was undertaken within a dedicated repository of image datasets of disasters and emergencies [38]. Next, global variables crucial for model training were modified, encompassing parameters such as the generic seed, number of epochs, learning rates, choice of pre-trained base models including EfficientNetB0, B7, V2B0, and V2L, InceptionV3, ResNet50, and VGG19, together with the pre-processing methods and optimization algorithms like Adam and RMSprop [39]. Subsequently, the dataset was read and decoded into pairs, ensuring proper preparation for training [40]. Random partitioning divided the dataset into training, validation, and test sets, providing robust model evaluation [41]. The CNN model was then constructed, featuring specified hyperparameters and layers, including the selected base model, global average pooling, dropout layers for regularization, and softmax activation function for multi-label classification tasks [42]. Performance metrics were plotted, depicting training and validation accuracy and losses over epochs, facilitating insights into model convergence and potential overfitting [43]. Moreover, image samples were visualized, presenting original images alongside the sample prediction result [44]. Then, global variables were adjusted based on observed results, allowing for further optimization and exploration of model configurations and re-train the model [45]. Finally, all results will be compiled to assess and evaluate which architecture provides a better performance. This structured approach enabled a comprehensive understanding of the CNN model's behavior and performance, facilitating iterative improvements toward enhanced accuracy and interpretability in image classification tasks [46].

The process of creating and improving the CNN model for image classification serves as an example of how machine learning operations are iterative. By means of thorough testing, visualization, and modification of global variables, valuable insights were obtained, and advancements were achieved throughout the entire process [47]. The method used in this study promoted a better understanding of the complex principles behind CNN-based image classification in addition to aiding in the optimization of model performance. The study serves as a monument to the commitment and creativity propelling developments in computer vision and artificial intelligence as the years' progress.

IV. RESULTS

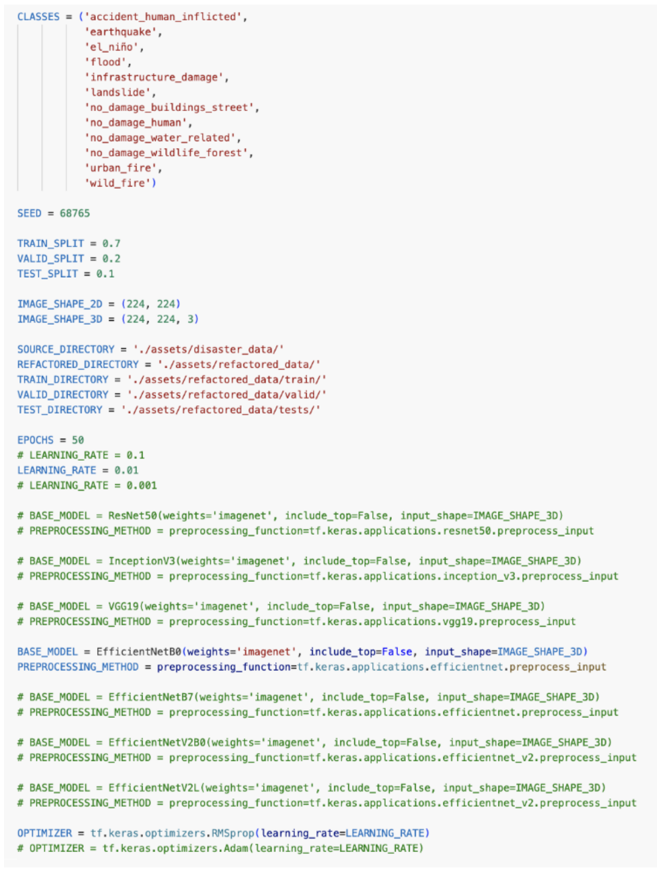
A. Image Repository

The image repository covers a broad range of incidents, from man-made accidents to natural disasters like floods and earthquakes to different types of environmental and infrastructure damage, which composed of 13,578 image datasets. The diversity of the dataset is crucial because it will be the basis for accurately capturing the complex and uncertain character of an emergency report sent in the iRESPOND system. There are a lot of images in each area in the collection, so there is enough coverage and depiction of many situations and settings. The models will be efficiently trained by the availability of these data, which enables them to learn and recognize complex patterns and features related to various emergencies and disasters. It will be trained to perform robustly over a wide range of emergency circumstances and generalize effectively to new data by utilizing this diversified dataset. This will increase the model's usefulness and efficacy in real-world applications.

B. Modify the Global Variables

As shown in Fig. 2, there are various configurations and parameters necessary for training the model for image classification in the iRESPOND system. We define the classes first, representing different categories of emergencies and disasters, such as earthquakes, floods, urban fires, infrastructure damage, etc. This categorization is needed for organizing and labeling the dataset appropriately, ensuring that the model can learn to distinguish between different types of emergency scenarios.

Several global variables are defined, including the random seed for reproducibility, the proportions for splitting the dataset into training, validation, and test sets, and the dimensions of the input images. These variables are for controlling the training process and evaluating the model's performance effectively. We specify the directories for accessing the source dataset and storing the refactored data after pre-processing to ensure proper data management and organization throughout the training pipeline.



```
CLASSES = ('accident_human_inflicted',
           'earthquake',
           'el_niño',
           'flood',
           'infrastructure_damage',
           'landslide',
           'no_damage_buildings_street',
           'no_damage_human',
           'no_damage_water_related',
           'no_damage_wildlife_forest',
           'urban_fire',
           'wild_fire')

SEED = 68765

TRAIN_SPLIT = 0.7
VALID_SPLIT = 0.2
TEST_SPLIT = 0.1

IMAGE_SHAPE_2D = (224, 224)
IMAGE_SHAPE_3D = (224, 224, 3)

SOURCE_DIRECTORY = './assets/disaster_data/'
REFACTORED_DIRECTORY = './assets/refactored_data/'
TRAIN_DIRECTORY = './assets/refactored_data/train/'
VALID_DIRECTORY = './assets/refactored_data/valid/'
TEST_DIRECTORY = './assets/refactored_data/tests/'

EPOCHS = 50
# LEARNING_RATE = 0.1
LEARNING_RATE = 0.01
# LEARNING_RATE = 0.001

# BASE_MODEL = ResNet50(weights='imagenet', include_top=False, input_shape=IMAGE_SHAPE_3D)
# PREPROCESSING_METHOD = preprocessing_function=tf.keras.applications.resnet50.preprocess_input

# BASE_MODEL = InceptionV3(weights='imagenet', include_top=False, input_shape=IMAGE_SHAPE_3D)
# PREPROCESSING_METHOD = preprocessing_function=tf.keras.applications.inception_v3.preprocess_input

# BASE_MODEL = VGG19(weights='imagenet', include_top=False, input_shape=IMAGE_SHAPE_3D)
# PREPROCESSING_METHOD = preprocessing_function=tf.keras.applications.vgg19.preprocess_input

BASE_MODEL = EfficientNetB0(weights='imagenet', include_top=False, input_shape=IMAGE_SHAPE_3D)
PREPROCESSING_METHOD = preprocessing_function=tf.keras.applications.efficientnet.preprocess_input

# BASE_MODEL = EfficientNetB7(weights='imagenet', include_top=False, input_shape=IMAGE_SHAPE_3D)
# PREPROCESSING_METHOD = preprocessing_function=tf.keras.applications.efficientnet.preprocess_input

# BASE_MODEL = EfficientNetV2B0(weights='imagenet', include_top=False, input_shape=IMAGE_SHAPE_3D)
# PREPROCESSING_METHOD = preprocessing_function=tf.keras.applications.efficientnet_v2.preprocess_input

# BASE_MODEL = EfficientNetV2L(weights='imagenet', include_top=False, input_shape=IMAGE_SHAPE_3D)
# PREPROCESSING_METHOD = preprocessing_function=tf.keras.applications.efficientnet_v2.preprocess_input

OPTIMIZER = tf.keras.optimizers.RMSprop(learning_rate=LEARNING_RATE)
# OPTIMIZER = tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE)
```

Fig. 2. Global variables (Code Snippet).

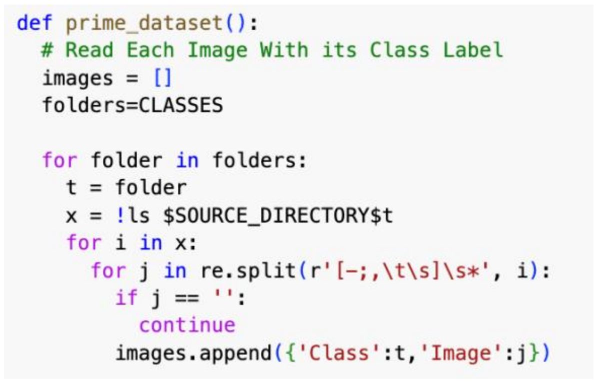
Key components of the model are configured next; including the choice of a base model pre-trained on ImageNet, in this case, EfficientNet (B0, B7, V2B0, and V2L), InceptionV3 ResNet50, and VGG19, along with the corresponding pre-processing method. The choice of base model and pre-processing technique significantly influences the model's performance and ability to extract meaningful features from input images.

Additionally, we define the optimizer used during zmodel training, with options for RMSprop or Adam optimization algorithms. The learning rate, an essential hyperparameter affecting the convergence and stability of the training process, is also specified.

C. Read and Decode

For this section, we initialized a function called "prime_dataset()" designed to facilitate the reading and decoding of the dataset. Based on Fig. 3, this function operates iteratively through each class folder within the dataset, where each folder corresponds to a distinct category of emergency or disaster-related scenarios.

Within each class folder, the function iterates over the images contained within, capturing both the image file name and its associated class label. This is achieved by utilizing shell commands, with the "ls" command listing all files within the specified directory. The resulting list of file names is then parsed using regular expressions to extract individual image filenames.



```
def prime_dataset():
  # Read Each Image With its Class Label
  images = []
  folders=CLASSES

  for folder in folders:
    t = folder
    x = !ls $SOURCE_DIRECTORY$t
    for i in x:
      for j in re.split(r'[-;,\t\s]\s*', i):
        if j == '':
          continue
        images.append({'Class':t,'Image':j})
```

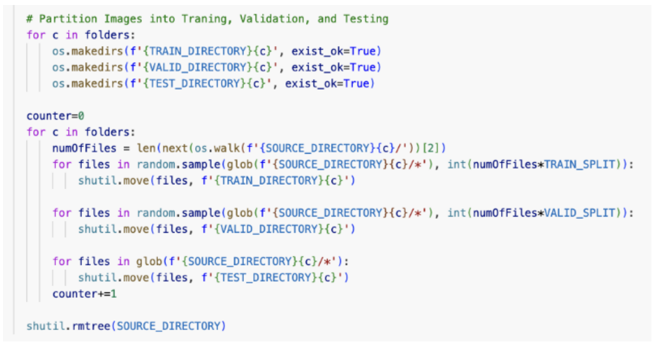
Fig. 3. Reading and decoding (Code Snippet).

Throughout this process, each image file is associated with its corresponding class label and added to a list named "images", ensuring that the dataset is structured appropriately for subsequent processing and model training. However, it's worth noting that the exact method for reading and loading images may vary depending on the specific dataset format and requirements. Therefore, additional pre-processing steps, such as image resizing or normalization, may be necessary to prepare the data adequately for model training.

D. Partition the Dataset

The dataset is partitioned into training, validation, and test sets with proportions of 70%, 20%, and 10%, respectively. This partitioning ensures that the models are trained on a sufficiently large portion of the data while also having separate datasets for validation and final evaluation. In order to do so, we created directories for each class within the training, validation, and testing directories, ensuring proper organization of the partitioned data. This organizational structure facilitates subsequent data loading and model training processes.

As the code segment presented in Fig. 4, it iterates through each class folder in the dataset, determining the number of files present in each class using the "os.walk()" function. For each class, a portion of the images is randomly selected based on the specified split ratios (TRAIN_SPLIT and VALID_SPLIT). Using the "random.sample()" function, files are randomly sampled from the class folder, with the number of files sampled proportional to the respective split ratio. These sampled files are then moved to the corresponding directories within the training and validation sets.



```
# Partition Images into Traning, Validation, and Testing
for c in folders:
    os.makedirs(f'{TRAIN_DIRECTORY}{c}', exist_ok=True)
    os.makedirs(f'{VALID_DIRECTORY}{c}', exist_ok=True)
    os.makedirs(f'{TEST_DIRECTORY}{c}', exist_ok=True)

counter=0
for c in folders:
    numOfFiles = len(next(os.walk(f'{SOURCE_DIRECTORY}{c}/'))[2])
    for files in random.sample(glob(f'{SOURCE_DIRECTORY}{c}/*'), int(numOfFiles*TRAIN_SPLIT)):
        shutil.move(files, f'{TRAIN_DIRECTORY}{c}')

    for files in random.sample(glob(f'{SOURCE_DIRECTORY}{c}/*'), int(numOfFiles*VALID_SPLIT)):
        shutil.move(files, f'{VALID_DIRECTORY}{c}')

    for files in glob(f'{SOURCE_DIRECTORY}{c}/*'):
        shutil.move(files, f'{TEST_DIRECTORY}{c}')
    counter+=1

shutil.rmtree(SOURCE_DIRECTORY)
```

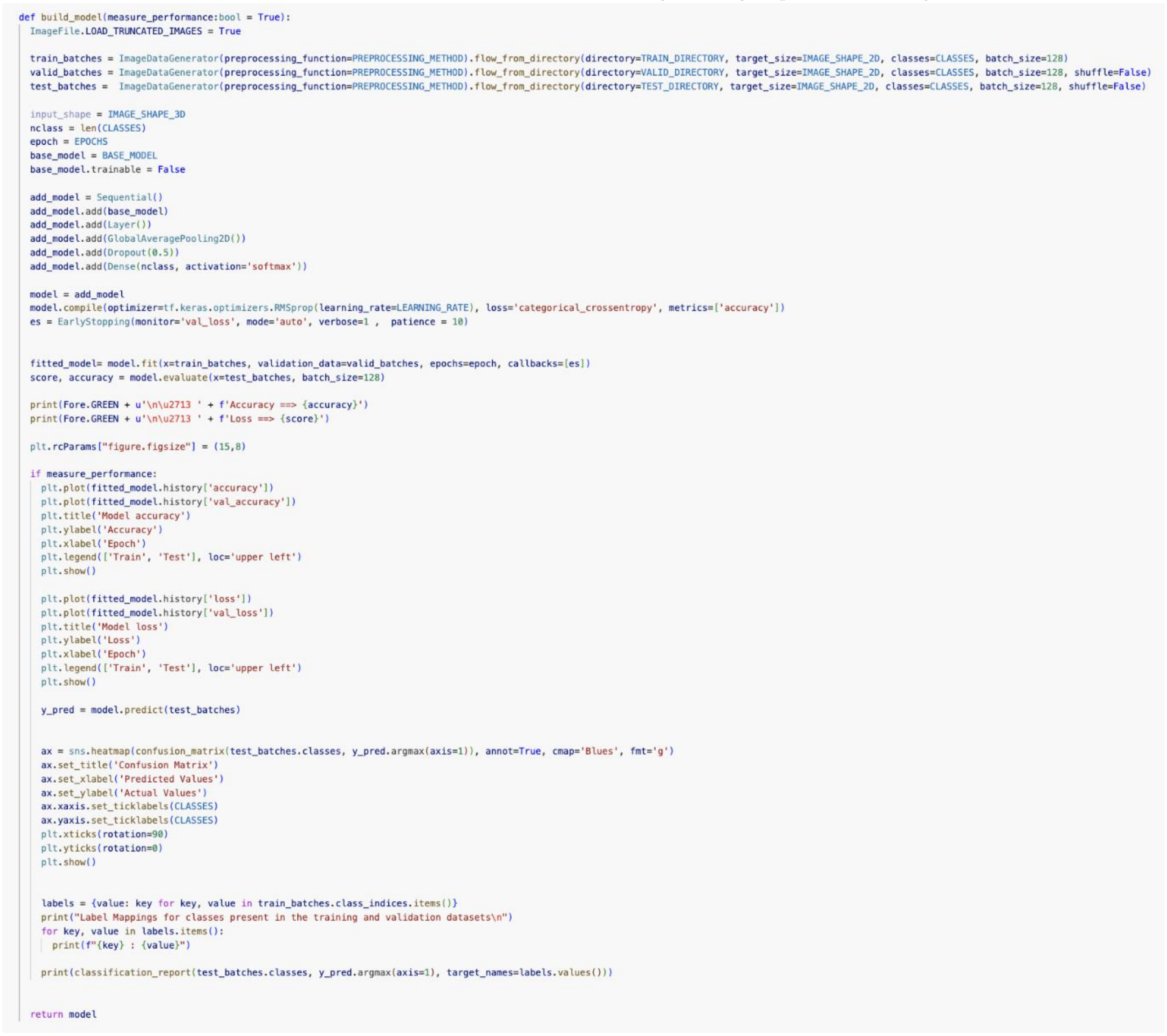
Fig. 4. Partitioning the dataset (Code Snippet).

After moving files to the training and validation directories, the remaining files within each class folder are moved to the testing directory. This ensures that every image in the dataset is accounted for and partitioned appropriately across the three sets.

Moreover, the original source directory containing the entire dataset is removed using "shutil.rmtree()", as the data has been successfully partitioned and relocated to the respective training, validation, and testing directories. This cleanup step helps maintain a clean and organized directory structure, reducing clutter and facilitating easier management of the dataset during subsequent stages of the model development process.

E. Build the CNN Model

The process of building the CNN model for image classification is set to start by initializing the "build_model()" function and setting "ImageFile.LOAD_TRUNCATED_IMAGES" to "True", ensuring that truncated images can be loaded without error during training as presented in Fig. 5.



```python
def build_model(measure_performance:bool = True):
  ImageFile.LOAD_TRUNCATED_IMAGES = True

  train_batches = ImageDataGenerator(preprocessing_function=PREPROCESSING_METHOD).flow_from_directory(directory=TRAIN_DIRECTORY, target_size=IMAGE_SHAPE_2D, classes=CLASSES, batch_size=128)
  valid_batches = ImageDataGenerator(preprocessing_function=PREPROCESSING_METHOD).flow_from_directory(directory=VALID_DIRECTORY, target_size=IMAGE_SHAPE_2D, classes=CLASSES, batch_size=128, shuffle=False)
  test_batches =  ImageDataGenerator(preprocessing_function=PREPROCESSING_METHOD).flow_from_directory(directory=TEST_DIRECTORY, target_size=IMAGE_SHAPE_2D, classes=CLASSES, batch_size=128, shuffle=False)

  input_shape = IMAGE_SHAPE_3D
  nclass = len(CLASSES)
  epoch = EPOCHS
  base_model = BASE_MODEL
  base_model.trainable = False

  add_model = Sequential()
  add_model.add(base_model)
  add_model.add(Layer())
  add_model.add(GlobalAveragePooling2D())
  add_model.add(Dropout(0.5))
  add_model.add(Dense(nclass, activation='softmax'))

  model = add_model
  model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=LEARNING_RATE), loss='categorical_crossentropy', metrics=['accuracy'])
  es = EarlyStopping(monitor='val_loss', mode='auto', verbose=1 ,  patience = 10)


  fitted_model= model.fit(x=train_batches, validation_data=valid_batches, epochs=epoch, callbacks=[es])
  score, accuracy = model.evaluate(x=test_batches, batch_size=128)

  print(Fore.GREEN + u'\n\u2713 ' + f'Accuracy ==> {accuracy}')
  print(Fore.GREEN + u'\n\u2713 ' + f'Loss ==> {score}')

  plt.rcParams["figure.figsize"] = (15,8)

  if measure_performance:
    plt.plot(fitted_model.history['accuracy'])
    plt.plot(fitted_model.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

    plt.plot(fitted_model.history['loss'])
    plt.plot(fitted_model.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

    y_pred = model.predict(test_batches)


    ax = sns.heatmap(confusion_matrix(test_batches.classes, y_pred.argmax(axis=1)), annot=True, cmap='Blues', fmt='g')
    ax.set_title('Confusion Matrix')
    ax.set_xlabel('Predicted Values')
    ax.set_ylabel('Actual Values')
    ax.xaxis.set_ticklabels(CLASSES)
    ax.yaxis.set_ticklabels(CLASSES)
    plt.xticks(rotation=90)
    plt.yticks(rotation=0)
    plt.show()


    labels = {value: key for key, value in train_batches.class_indices.items()}
    print("Label Mappings for classes present in the training and validation datasets\n")
    for key, value in labels.items():
      print(f"{key} : {value}")

    print(classification_report(test_batches.classes, y_pred.argmax(axis=1), target_names=labels.values()))


  return model
```

Fig. 5. Building the CNN model (Code Snippet).

The function prepares data batches for training, validation, and testing using the "ImageDataGenerator" class from TensorFlow's Keras API. Images are loaded from their respective directories ("TRAIN_DIRECTORY", "VALID_DIRECTORY", "TEST_DIRECTORY") and resized to the specified target size ("IMAGE_SHAPE_2D"). Additionally, the images undergo pre-processing using the "PREPROCESSING_METHOD" function to ensure consistency and compatibility with the chosen base model.

The architecture of the CNN model is then constructed, starting with the pre-trained base model with its top layers removed. Following the base model, as shown in Table I, a custom sequence of layers is added, including a global average pooling layer, a dropout layer for regularization, and a dense layer with softmax activation for multi-class classification.

The model is compiled with the specified optimizer, loss function, and evaluation metrics. During model training, early stopping is implemented, as a sample shown in Fig. 6, to prevent overfitting, with training progress monitored using the validation data.

F. Plotting the Model's Performance

Performance metrics such as training accuracy, validation accuracy, training loss, and validation loss are plotted over epochs to monitor the model's convergence and potential overfitting as a result of building the CNN model from the previous phase. Once training is complete, the model's performance is evaluated using the test data, and metrics such as accuracy and loss are printed to the console. As "measure_performance" is set to "True", additional visualizations and performance evaluations are conducted. This includes plotting the model's accuracy and loss curves over epochs, generating a confusion matrix to visualize the model's performance across different classes, and a classification report summarizing the model's performance metrics. See a sample model accuracy, loss, confusion matrix, and classification report using EfficientNetB0 with RMSprop Optimizer and 1% Learning Rate in Fig. 7, 8, 9, and Table II, respectively.

G. Visualize Image Samples

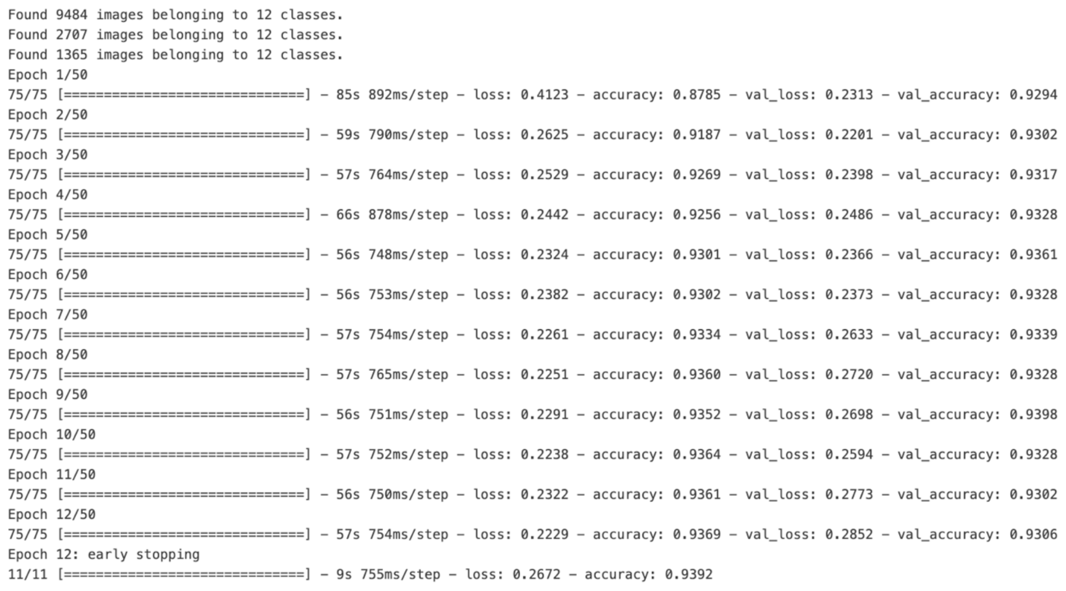
After the trained model is returned and has provided a comprehensive framework for building, training, and evaluating CNN models for image classification, we have invoked a sample image for visualization to see the certainty of how the model correctly predicts. In this phase, we include displaying the fed original image as well as the predicted label of the sample image, as show in Fig. 10.

TABLE I. MODEL SUMMARY (SAMPLE)

Model: "sequential"		
Layer (type)	Output Shape	Param #
efficientnetb0(Functional)	(None, 7, 7, 1280)	4049571
layer(Layer)	(None, 7, 7, 1280)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout(Dropout)	(None, 1280)	0
dense(Dense)	(None, 12)	15372
Total params: 4064943 (15.51 MB) Trainable params: 15372 (60.05 KB) Non-trainable params: 4049571 (15.45 MB)		

TABLE II. SAMPLE CLASSIFICATION RESULT

	precision	recall	f1-score	support
accident_human_inflicted	0.94	0.67	0.78	24
earthquake	0.00	0.00	0.00	4
el_niño	0.83	0.95	0.89	21
flood	0.92	0.80	0.86	104
infrastructure_damage	0.86	0.94	0.90	143
landslide	0.67	0.78	0.72	46
no_damage_buildings_street	0.99	0.99	0.99	458
no_damage_human	0.80	1.00	0.89	12
no_damage_water_related	1.00	0.97	0.98	229
no_damage_wildlife_forest	0.99	1.00	0.99	228
urban_fire	0.81	0.67	0.73	43
wild_fire	0.82	0.89	0.85	53
accuracy			0.94	1365
macro avg	0.80	0.81	0.80	1365
weighted avg	0.94	0.94	0.94	1365



```
Found 9484 images belonging to 12 classes.
Found 2707 images belonging to 12 classes.
Found 1365 images belonging to 12 classes.
Epoch 1/50
75/75 [==============================] - 85s 892ms/step - loss: 0.4123 - accuracy: 0.8785 - val_loss: 0.2313 - val_accuracy: 0.9294
Epoch 2/50
75/75 [==============================] - 59s 790ms/step - loss: 0.2625 - accuracy: 0.9187 - val_loss: 0.2201 - val_accuracy: 0.9302
Epoch 3/50
75/75 [==============================] - 57s 764ms/step - loss: 0.2529 - accuracy: 0.9269 - val_loss: 0.2398 - val_accuracy: 0.9317
Epoch 4/50
75/75 [==============================] - 66s 878ms/step - loss: 0.2442 - accuracy: 0.9256 - val_loss: 0.2486 - val_accuracy: 0.9328
Epoch 5/50
75/75 [==============================] - 56s 748ms/step - loss: 0.2324 - accuracy: 0.9301 - val_loss: 0.2366 - val_accuracy: 0.9361
Epoch 6/50
75/75 [==============================] - 56s 753ms/step - loss: 0.2382 - accuracy: 0.9302 - val_loss: 0.2373 - val_accuracy: 0.9328
Epoch 7/50
75/75 [==============================] - 57s 754ms/step - loss: 0.2261 - accuracy: 0.9334 - val_loss: 0.2633 - val_accuracy: 0.9339
Epoch 8/50
75/75 [==============================] - 57s 765ms/step - loss: 0.2251 - accuracy: 0.9360 - val_loss: 0.2720 - val_accuracy: 0.9328
Epoch 9/50
75/75 [==============================] - 56s 751ms/step - loss: 0.2291 - accuracy: 0.9352 - val_loss: 0.2698 - val_accuracy: 0.9398
Epoch 10/50
75/75 [==============================] - 57s 752ms/step - loss: 0.2238 - accuracy: 0.9364 - val_loss: 0.2594 - val_accuracy: 0.9328
Epoch 11/50
75/75 [==============================] - 56s 750ms/step - loss: 0.2322 - accuracy: 0.9361 - val_loss: 0.2773 - val_accuracy: 0.9302
Epoch 12/50
75/75 [==============================] - 57s 754ms/step - loss: 0.2229 - accuracy: 0.9369 - val_loss: 0.2852 - val_accuracy: 0.9306
Epoch 12: early stopping
11/11 [==============================] - 9s 755ms/step - loss: 0.2672 - accuracy: 0.9392
```

Fig. 6. Sample early stopping (EfficientNetB0, RMSprop Optimizer, and 1% Learning Rate.

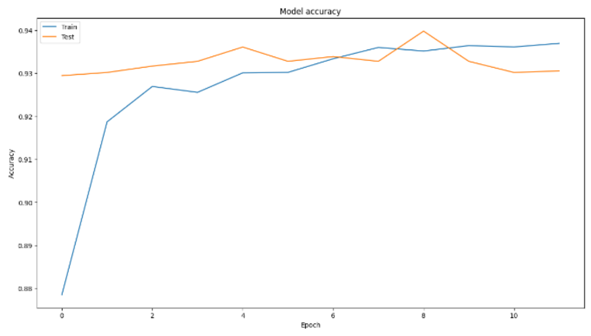


Fig. 7. Sample accuracy graph.

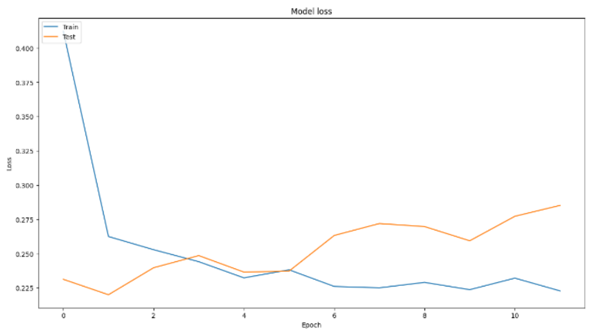
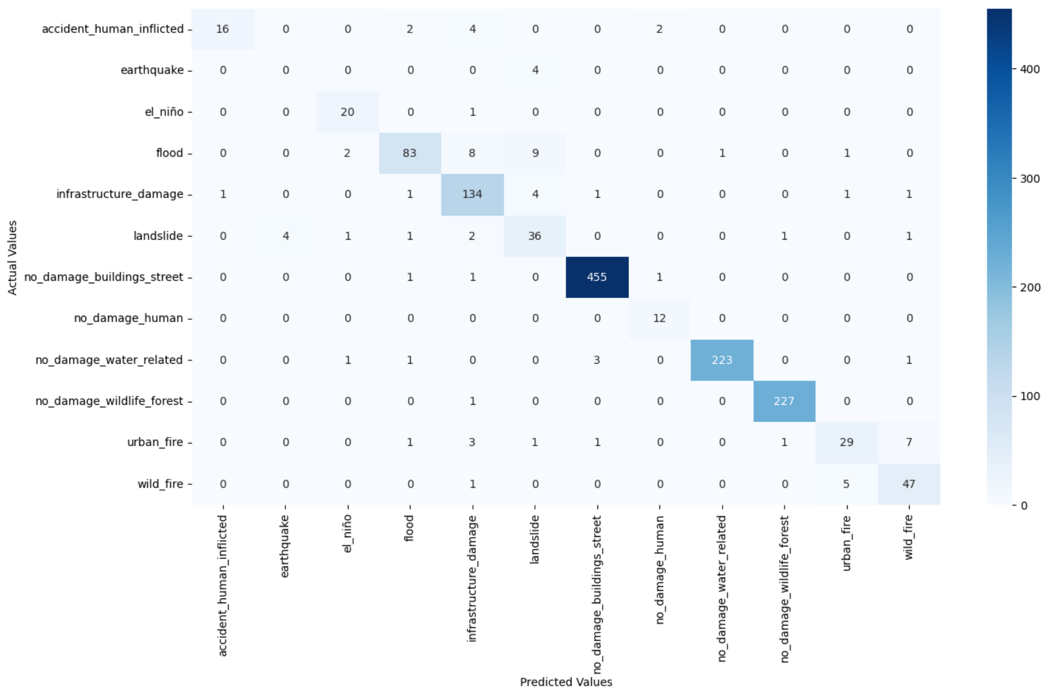
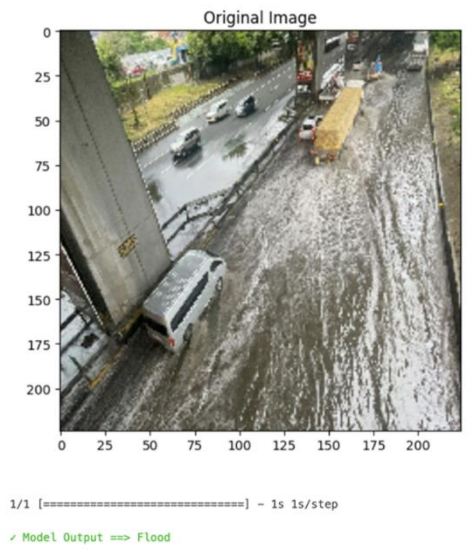


Fig. 8. Sample loss graph.



	accident_human_inflicted	earthquake	el_niño	flood	infrastructure_damage	landslide	no_damage_buildings_street	no_damage_human	no_damage_water_related	no_damage_wildlife_forest	urban_fire	wild_fire
accident_human_inflicted	16	0	0	2	4	0	0	2	0	0	0	0
earthquake	0	0	0	0	0	4	0	0	0	0	0	0
el_niño	0	0	20	0	1	0	0	0	0	0	0	0
flood	0	0	2	83	8	9	0	0	1	0	1	0
infrastructure_damage	1	0	0	1	134	4	1	0	0	0	1	1
landslide	0	4	1	1	2	36	0	0	0	1	0	1
no_damage_buildings_street	0	0	0	1	1	0	455	1	0	0	0	0
no_damage_human	0	0	0	0	0	0	0	12	0	0	0	0
no_damage_water_related	0	0	1	1	0	0	3	0	223	0	0	1
no_damage_wildlife_forest	0	0	0	0	1	0	0	0	0	227	0	0
urban_fire	0	0	0	1	3	1	1	0	0	1	29	7
wild_fire	0	0	0	0	1	0	0	0	0	0	5	47

Fig. 9. Sample confusion matrix.



```
1/1 [==============================] - 1s 1s/step
✓ Model Output ==> Flood
```

Fig. 10. Model Prediction Output Sample from EfficientNetB0 with RMSprop Optimizer and 1% Learning Rate

H. Re-training

As the previous phase concludes and observations are made regarding the model's performance, the global variables are adjusted accordingly to explore different configurations and optimize the model further. This adjustment process may involve modifying parameters such as the optimizer and learning rate to experiment with alternative optimization strategies and fine-tune the model's performance. By allowing for the re-training of the models with different optimizers and learning rates, this phase enables us to conduct systematic experimentation and exploration of various hyperparameter configurations.

I. Evaluation

Based on the re-training of the CNN architectures with Adam and RMSprop optimizers and different learning rates, this section presents the culmination of the iterative process of re-training CNN architectures with Adam and RMSprop optimizers, along with various learning rates. This stage involves compiling and analyzing the results obtained from the re-trained models to compare their performance comprehensively. The metrics include measures of accuracy and loss, which collectively provide insights into the model's classification performance across different classes. We have also included prediction result on a sample image (same sample image in Fig. 10) used across the re-training process.

TABLE III. SUMMARY OF RESULTS ON ACCURACY AND LOSS PER OPTIMIZERS AND LEARNING RATE

CNN Architecture	Learning Rate (%)	Optimizers			
		Adam		RMSprop	
		Accuracy (%)	Loss	Accuracy (%)	Loss
EfficientNetB0	10	92.8937733	2.228926181793213	93.2600737	1.8809784650802612
	1	93.9194143	0.267235666513443	93.9194143	0.267235666513443
	0.1	94.3589747	0.1868174523115158 [b]	94.3589747	0.1868174523115158
EfficientNetB7	10	90.6959713	3.016308546066284	91.6483521	2.9703691005706787
	1	91.501832	0.39473479986190796	91.2820518	0.41184505820274353
	0.1	91.4285719	0.270229309797287	91.4285719	0.270229309797287
EfficientNetV2B0	10	92.8937733	2.0541882514953613	93.4065938	2.1451196670532227
	1	94.1391945	0.3025626838207245	93.7728941	0.29636630415916443
	0.1	93.9926744	0.1890583485364914	93.9926744	0.1890583485364914
EfficientNetV2L	10	89.4505501	1.9358875751495361	89.3040299	1.9205394983291626
	1	90.402931	0.36925235390663147	90.402931	0.36925235390663147
	0.1	90.6959713	0.2923825979232788	90.6959713	0.2923825979232788
InceptionV3	10	87.6923084	10.626327514648438	88.351649	9.081643104553223
	1	89.1575098	1.0289123058319092	89.5238101	1.1351069211959839
	0.1	90.8424914	0.3351040780544281	90.8424914	0.3351040780544281
ResNet50	10	93.1135535	8.164137840270996	92.1611726	8.595427513122559
	1	92.0879126	0.9594696164131165	92.3076928	0.8893887400627136
	0.1	95.0219631 [a]	0.22417350113391876	93.8461542	0.21442490816116333
VGG19	10	88.2051289	12.45382308959961	90.5494511	7.596359729766846
	1	89.37729	1.0163341760635376	91.4285719	0.7855556607246399
	0.1	90.3296709	0.35073262453079224	90.3296709	0.35073262453079224

[a]. Highest model accuracy result

b. Lowest model loss result

TABLE IV. SUMMARY OF PREDICTION RESULT

CNN Architecture	Learning Rate (%)	Optimizers	
		Adam	RMSprop
EfficientNetB0	10	Infrastructure Damage c	Landslide c
	1	Flood	Flood
	0.1	Flood	Flood
EfficientNetB7	10	Flood	Flood
	1	Flood	Flood
	0.1	Flood	Flood
EfficientNetV2B0	10	Flood	Landslide c
	1	Landslide c	Landslide c
	0.1	Flood	Flood
EfficientNetV2L	10	Earthquake c	Infrastructure Damage c
	1	Landslide c	Landslide c
	0.1	Infrastructure Damage c	Infrastructure Damage c
InceptionV3	10	Flood	Flood
	1	Flood	Flood
	0.1	Flood	Flood
ResNet50	10	Flood	Flood
	1	Flood	Flood
	0.1	Flood	Flood
VGG19	10	Flood	No Damage (Building / Street) c
	1	Flood	Flood
	0.1	Flood	Flood

c. Wrong Prediction

Table III highlights the performance of different architectures trained with varying learning rates, focusing specifically on accuracy and loss metrics. Among the architectures evaluated, ResNet50 achieved the highest accuracy of approximately 95% when trained with a learning rate of 0.001. On the other hand, EfficientNetB0 exhibited the lowest loss of 0.1868174523115158 when trained with the same learning rate of 0.001.

The observation summarized in Table IV consistently exhibits that some models have relatively higher error rates in distinguishing between certain pairs of disaster scenarios, such as flood and landslide, as well as infrastructure damage and earthquake, which can be attributed to the shared characteristics and visual similarities between these classes. Moreover, the similarity between urban fire and wildfire scenarios further exacerbates also the difficulty in classification.

V. DISCUSSION

With the result on the performance of the architectures based on Table III, it suggests that ResNet50, a well-established and widely-used architecture, was particularly effective in capturing and learning the intricate patterns and features present in the image dataset which also been supported in the study of Balavani et al. [48] And according to Wu et al., the choice of a lower learning rate of 0.001 likely facilitated more stable and precise updates to the model's parameters during training, leading to improved accuracy [49]. While accuracy measures the proportion of correctly classified instances, loss quantifies the difference between the predicted and actual values, serving as a measure of how well the model is performing overall [50]. As stated in the study of Chandrasekhar & Peddakrishna, a lowest loss indicates that the model's predictions are closer to the true values on average, suggesting better overall performance [51]. In this case, EfficientNetB0, known for its efficient architecture design and superior performance, demonstrated effectiveness in minimizing prediction errors and achieving optimal performance in terms of loss.

Interestingly, the analysis also proves that the choice of optimization algorithm did not significantly impact the model's performance. Both Adam and RMSprop optimizers were used in training the architectures, but neither seemed to contribute significantly to the observed variations in accuracy or loss [52]. This finding implies that other factors, such as the architecture itself and the choice of learning rate, played a more substantial role in determining the model's performance.

As a further observation, classes like flood and landslide may share common visual elements, like water bodies, debris, or altered landscapes, making it challenging for the models to differentiate. Similarly, infrastructure damage and earthquake scenarios may manifest similar visual cues, such as collapsed buildings, rubble, or structural damage, leading to confusion for the models in distinguishing between these classes. Moreover, in urban fire and wildfire scenarios also exhibit same visual characteristics on flames, smoke, or burned landscapes, making it challenging also for the models to distinguish between them accurately [30-32]. This observed behavior intensifies the need of future work that aligns with the inherent complexity and ambiguity present in disaster-related imagery classification tasks.

VI. Conclusion

Re-training allows for the comparison of the performance of multiple models trained with different configurations. By systematically evaluating and comparing the results obtained from these models, we have gained a deeper understanding of the factors influencing model performance and make informed decisions regarding model selection and strategies. The re-training phase made a continuous refinement and optimization of the models for image classification tasks. Through iterative experimentation and adjustment of global variables, we explored a wide range of configurations, identified optimal settings, and ultimately enhanced the effectiveness and robustness of the models that can be deployed in the iRESPOND system.

Moreover, through this re-training phase, we were able to identify that among the tested architectures, ResNet50 and EfficientNetB0 emerged as the top performers, exhibiting the highest accuracy of 95.95% and lowest loss result of 0.187 respectively, when trained with a learning rate of 0.1%. This finding underscores the efficacy of these architectures in effectively capturing and learning the complex features present in the datasets, thereby facilitating accurate classification within the iRESPOND system. Also, the analysis suggests that the choice of optimization algorithm, whether Adam or RMSprop, did not exert a significant impact on the performance of the models in this context. Despite variations in optimization techniques, the observed performance metrics remained consistent across both algorithms. This finding indicates that factor other than the choice of optimizer, such as the architecture itself and the learning rate, played a more influential role in determining model performance and effectiveness.

In addition, the observed higher error rates in distinguishing between certain pairs of disaster scenarios are logical and expected, given the inherent visual similarities and shared characteristics between these classes. Addressing these challenges requires not only fine-tuning hyperparameters but also exploring advanced techniques such as incorporating contextual information, utilizing ensemble learning approaches, or leveraging domain-specific knowledge to enhance model performance and robustness in image classification.

Acknowledgment

The authors would like to express their appreciation to Hon. Fernando V. Gonzales – City Mayor of Ligao, and Mr. Rossauro C. Perillo – Dept. Head I of Local DRRMC for allowing the authors to conduct the study in the local government unit and providing pertinent data for this research.

References

[1] Alam, and A. Mohanty, “Educational technology: Exploring the convergence of technology and pedagogy through mobility, interactivity, AI, and learning tools,” Cogent Engineering, vol. 10, issue 2, November 2023, doi: 10.1080/23311916.2023.2283282.

[2] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” Insights into Imaging Springer Verlag, vol. 9, issue 4, pp. 611–629, June 2018, doi: 10.1007/s13244-018-0639-9.

[3] M. Tan, and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” International Conference on Machine Learning, PMLR, pp. 6105-6114, May 2019, doi: 10.48550/arXiv.1905.11946.

[4] S. F. Ahmed, M. S. B. Alam, M. Hassan, M. R. Rozbu, T. Ishtiak, N. Rafa, et al., “Deep learning modelling techniques: current progress, applications, advantages, and challenges,” Artificial Intelligence Review, vol. 56, pp. 13521–13617, April 2023, doi: 10.1007/s10462-023-10466-8.

[5] C. Lin, P. Yang, Q. Wang, Z. Qiu, W. Lv, and Z. Wang, “Efficient and accurate compound scaling for convolutional neural networks,” Neural Networks, vol. 167, pp. 787-797, October 2023, doi: 10.1016/j.neunet.2023.08.053.

[6] M. Tan, and Q. V. Le, “EfficientNetV2: Smaller Models and Faster Training,” International Conference on Machine Learning, pp. 10096-10106, July 2021, doi: 10.48550/arXiv.2104.00298.

[7] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, “Review of image classification algorithms based on convolutional neural networks,” Remote Sensing, vol. 13(22), November 2021, doi: 10.3390/rs13224712.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818-2826, 2016, doi: 10.48550/arXiv.1512.00567.

[9] E. Barcic, P. Grd, I. Tomicic, E. Barči, and I. Tomiči, “Convolutional Neural Networks for Face Recognition: A Systematic Literature Review,” Research Square (preprint), July 2023, doi: 10.21203/rs.3.rs-3145839/v1.

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778, 2016, doi: 10.48550/arXiv.1512.03385.

[11] A. V. Ikechukwu, S. Murali, R. Deepu, and R. C. Shivamurthy, “ResNet-50 vs VGG-19 vs training from scratch: A comparative analysis of the segmentation and classification of Pneumonia from chest X-ray images,” Global Transitions Proceedings, vol. 2(2), pp. 375–381, November 2021, doi: 10.1016/j.gltp.2021.08.027.

[12] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. A. Dujaili, Y. Duan, O. Al-Shamma, et al., “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” Journal of Big Data, vol. 8(1), March 2021, doi: 10.1186/s40537-021-00444-8.

[13] K. Simonyan, and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv (preprint), 2014, doi: 10.48550/arXiv.1409.1556.

[14] S. Audu, and A. A. Aminu, “Wavelet Attention VGG19 and XGBOOST for Classification of Skin Disease,” International Journal of Computer Science and Information Technology Research, vol. 11(4), pp. 5–13, October 2023, doi: 10.5281/zenodo.8416714.

[15] M. Krichen, “Convolutional Neural Networks: A Survey,” Computers, vol. 12(8), no. 151, June 2023, doi: 10.3390/computers12080151.

[16] D. P. Kingma, and J. Ba, “Adam: A Method for Stochastic Optimization,” arXiv (preprint), 2014, doi: 10.48550/arXiv.1412.6980.

[17] A. Barodi, A. Bajit, M. Benbrahim, and A. Tamtaoui, “Improving the transfer learning performances in the classification of the automotive traffic roads signs,” E3S Web of Conferences, vol. 234, no. 00064, February 2021, doi: 10.1051/e3sconf/202123400064.

[18] M. Reyad, A. M. Sarhan, and M. Arafa, “A modified Adam algorithm for deep neural network optimization,” Neural Computing and Applications, vol. 35(23), pp. 17095–17112, April 2023, doi: 10.1007/s00521-023-08568-z.

[19] Z. Zhang, “Improved Adam Optimizer for Deep Neural Networks,” 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), Banff, AB, Canada, 2018, pp. 1-2, 2018, doi: 10.1109/IWQoS.2018.8624183.

[20] T. Tieleman, G. Hinton, “Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude," COURSERA: Neural Networks for Machine Learning, vol. 4(2), 26-31, 2012.

[21] R. Elshamy, R., O. A. Elnasr, M. Elhoseny, and S. Elmougy, “Improving the efficiency of RMSProp optimizer by utilizing Nestrove in deep learning,” Scientific Reports, vol. 13, no. 8814, May 2023, doi: 10.1038/s41598-023-35663-x.

[22] D. Soydaner, “A Comparison of Optimization Algorithms for Deep Learning,” International Journal of Pattern Recognition and Artificial Intelligence, Deep Learning, vol. 34, no. 13 (2052013), 2020, doi: 10.1142/S0218001420520138.

[23] A. Ghatak, “Optimization,” Deep Learning with R, Springer, Singapore, ISBN: 978-981-13-5849-4, April 2019, doi: 10.1007/978-981-13-5850-0_5

[24] S. Ghaffarian, F. R. Taghikhah, and H. R. Maier, “Explainable Artificial Intelligence in Disaster Risk Management: Achievements and Prospective Futures,” International Journal of Disaster Risk Reduction, vol. 98, no. 104123, November 2023, doi: 10.1016/j.ijdrr.2023.104123.

[25] S. Ghaffarian, N. Kerle, E. Pasolli, and J. J. Arsanjani, “Post-disaster building database updating using automated deep learning: An integration of pre-disaster OpenStreetMap and multi-temporal satellite data,” Remote Sensing, vol. 11(20), no. 2427, October 2019, doi: 10.3390/rs11202427.

[26] Y. Tang, H. Feng, J. Chen, and Y. Chen, “ForestResNet: A Deep Learning Algorithm for Forest Image Classification,” Journal of Physics, vol. 2024(1), no. 012053, August 2024, doi: 10.1088/1742-6596/2024/1/012053.

[27] A. Rathod, V. Pariawala, M. Surana, and K. Saxena, “Leveraging CNNs and Ensemble Learning for Automated Disaster Image Classification,” International Conference on Sustainable and Innovative Solutions for Current Challenges in Engineering & Technology, vol. 1, November 2023, doi: 10.48550/arXiv.2311.13531.

[28] J. Shah, D. Patel, J. Shah, S. Shah, and V. Sawant, “Proposed Methodology for Disaster Classification Using Computer Vision and Federated Learning,” Research Square ver.1(preprint), July 2023, doi: 10.21203/rs.3.rs-3160125/v1.

[29] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, et al., “ Review of deep learning: concepts, CNN architectures, challenges, applications, future directions, ” Journal of Big Data, vol. 8, no. 53, March 2021, doi: 10.1186/s40537-021-00444-8.

[30] A. Asif, S. Khatoon, M. Hasan, M. A. Alshamari, S. Abdou, K. M. Elsayed, et al., “Automatic analysis of social media images to identify disaster type and infer appropriate emergency response,” Journal of Big Data, vol. 8, no. 53, June 2021, doi: 10.1186/s40537-021-00471-5.

[31] S. Daly, and J. A. Thom, “Mining and Classifying Image Posts on Social Media to Analyse Fires,” ISCRAM 2016 Conference Proceedings - 13th International Conference on Information Systems for Crisis Response and Management, no. 1395, pp. 1-14, May 2016.

[32] J. Kallas, and R. Napolitano, “AUTOMATED LARGE-SCALE DAMAGE DETECTION ON HISTORIC BUILDINGS IN POST-DISASTER AREAS USING IMAGE SEGMENTATION,” The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XLVIII-M-2-2023, pp. 797-804, June 2023, doi: 10.5194/isprs-archives-XLVIII-M-2-2023-797-2023.

[33] A. Mukhopadhyay, G. Pettet, S. M. Vazirizade, D. Lu, A. Jaimes, S. El Said, et al., “A Review of Incident Prediction, Resource Allocation, and Dispatch Models for Emergency Management,” Accident Analysis & Prevention, vol. 165, no. 106501, February 2022, doi: 10.1016/j.aap.2021.106501.

[34] N. Sharma, V. Jain, and A. Mishra, “An Analysis of Convolutional Neural Networks For Image Classification,” Procedia Computer Science, vol. 132, pp. 377-384, June 2018, doi: 10.1016/j.procs.2018.05.198.

[35] S. M. Zainorzuli, S. A. Che Abdullah, H. Z. Abidin and F. A. Ruslan, “Comparison Study on Convolution Neural Network (CNN) Techniques for Image Classification,” Journal of Electrical and Electronic Systems Research, vol. 20, pp. 11-17, 2022, doi: 10.24191/jeesr.v20i1.002.

[36] L. Sushma, and K. P. Lakshmi, “An Analysis of Convolution Neural Network for Image Classification using Different Models,” International Journal of Engineering Research & Technology (IJERT), vol. 9(10), October 2020, doi: 10.17577/IJERTV9IS100294.

[37] S. Tufail, H. Riggs, M. Tariq, and A. I. Sarwat, “Advancements and Challenges in Machine Learning: A Comprehensive Review of Models, Libraries, Applications, and Algorithms,” Electronics (Switzerland), vol. 12(8), April 2023, doi: 10.3390/electronics12081789.

[38] J. Li, G. Zhu, C. Hua, M. Feng, B. Bennamoun, P. Li, et al., “A Systematic Collection of Medical Image Datasets for Deep Learning,” ACM Computing Surveys, vol. 56(5), no. 116, pp. 1–51, November 2023, doi: 10.1145/3615862.

[39] T. I. Götz, S. Göb, S. Sawant, X. F. Erick, T. Wittenberg, C. Schmidkonz, et al., “Number of necessary training examples for Neural Networks with different number of trainable parameters,” Journal of Pathology Informatics, vol. 13, no. 100114, July 2022, doi: 10.1016/j.jpi.2022.100114.

[40] P. Tarasiuk, and P. S. Szczepaniak, “Novel convolutional neural networks for efficient classification of rotated and scaled images,” Neural Computing and Applications, vol. 34(13), pp. 10519–10532, December 2021, doi: 10.1007/s00521-021-06645-9.

[41] V. R. Joseph, and A. Vakayil, “SPlit: An Optimal Method for Data Splitting,” Technometrics, vol. 64(2), pp. 166–176, June 2021, doi: 10.1080/00401706.2021.1921037.

[42] A. Zafar, M. Aamir, N. M. Nawi, A. Arshad, S. Riaz, A. Alruban, et al., “A Comparison of Pooling Methods for Convolutional Neural Networks,” Applied Sciences (Switzerland), vol. 12(17), no. 8643, August 2022, doi: 10.3390/app12178643.

[43] B. Dey, J. Ferdous, R. Ahmed, and J. Hossain, “Assessing deep convolutional neural network models and their comparative performance for automated medicinal plant identification from leaf images,” Heliyon, vol. 10(1), no. E23655, December 2023, doi: 10.1016/j.heliyon.2023.e23655.

[44] J. Yang, and Y. Kwon, “Novel CNN-Based Approach for Reading Urban Form Data in 2D Images: An Application for Predicting Restaurant Location in Seoul, Korea,” ISPRS International Journal of Geo-Information, vol. 12(9), September 2023, doi: 10.3390/ijgi12090373.

[45] U. M. Aseguinolaza, I. F. Iriondo, I. R. Moreno, N. Aginako, and B. Sierra, “Convolutional neural network-based classification and monitoring models for lung cancer detection: 3D perspective approach,” Heliyon, vol. 9(11), no. E21203, October 2023, doi: 10.1016/j.heliyon.2023.e21203.

[46] S. Yeşilmen, and B. Tatar, “Efficiency of convolutional neural networks (CNN) based image classification for monitoring construction related activities: A case study on aggregate mining for concrete production,” Case Studies in Construction Materials vol. 17, no. e01372, December 2022, doi: 10.1016/j.cscm.2022.e01372.

[47] I. H. Sarker, “Machine Learning: Algorithms, Real-World Applications and Research Directions,” SN Computer Science, Springer, vol. 2(3), no. 160, March 2021, doi: 10.1007/s42979-021-00592-x.

[48] K. Balavani, D. Sriram, M. B. Shankar, and D. S. Charan, "An Optimized Plant Disease Classification System Based on Resnet-50 Architecture and Transfer Learning," 2023 4th International Conference

for Emerging Technology (INCET), Belgaum, India, pp. 1-5, July 2023, doi: 10.1109/INCET57972.2023.10170368.

[49] T. Wu, P. Zeng, and C. Song, "An optimization Strategy for Deep Neural Networks Training," 2022 International Conference on Image Processing, Computer Vision and Machine Learning (ICICML), Xi'an, China, January 2013, pp. 596-603, doi: 10.1109/ICICML57342.2022.10009665.

[50] H. Kotta, K. Pardasani, M. Pandya,and R. Ghosh, "Optimization of Loss Functions for Predictive Soil Mapping," Advanced Machine Learning Technologies and Applications: Proceedings of AMLTA 2020, Springer Singapore, vol. 1141, pp. 95-104, doi: 10.1007/978-981-15-3383-9_9.

[51] N. Chandrasekhar, and S. Peddakrishna, "Enhancing Heart Disease Prediction Accuracy through Machine Learning Techniques and Optimization," Processes, vol. 11(4), no. 1210, April 2023, doi: 10.3390/pr11041210

[52] H. Naganuma, K. Ahuja, S. Takagi, T. Motokawa, R. Yokota, K. Ishikawa, et al., "Empirical study on optimizer selection for out-of-distribution generalization," arXiv (preprint), November 2022, doi: 10.48550/arXiv.2211.08583.