Team 17

Name	Section	BN
Mostafa Ahmed Sobhy Ahmed	2	24
Michael Aziz Fahim	2	10
Mostafa Magdy Abdelrahman	2	25
Mostafa Mahmoud Kamal	2	27

Project Pipeline

- Image preprocessing
- Feature extraction and selection
- Classification (per pixel classification and classical models)
- evaluation (the performance of the model is evaluated using metrics such as accuracy, precision, recall, and F1 score)

A detailed description of each phase in your pipeline

1- Data preprocessing

In the preprocessing phase of the project, the images are resized to a consistent size using the preprocess_image function. The function takes an image path as input and performs several preprocessing steps on the image.

First, the image is read using the OpenCV library's cv2.imread function. This step ensures that the image data is loaded into memory for further processing.

Next, the image is resized using the cv2.resize function. In this specific case, the image is resized to a fixed size of 256x256 pixels. Resizing the images to a uniform size is important for ensuring compatibility and consistency in subsequent steps of the project.

Finally, the pixel values of the resized image are normalized to a specific range, typically ranging from 0 to 1. This normalization step ensures that the pixel values are within a standardized range, making it easier for machine learning algorithms to process and analyze the images.

By performing image resizing and normalization in the preprocessing phase, the project ensures that the input images are standardized and ready for further analysis and modeling.

2- Feature extraction

CNN Features

In the second phase of the project, feature extraction is performed using a Convolutional Neural Network (CNN). The extract_cnn_features function is utilized to extract CNN features from an input image.

The function begins by preprocessing the image for compatibility with the ResNet-50 model. The image is transformed using a series of operations, including conversion to a PIL Image object, conversion to a tensor, and normalization using specific mean and standard deviation values.

To ensure compatibility with the model, the image is converted to the uint8 data type by multiplying the pixel values by 255 and casting them to np.uint8.

Next, the pre-trained ResNet-50 model is loaded. This model is widely used for image classification tasks due to its effectiveness in learning rich hierarchical features.

To extract the CNN features, the last fully connected layer of the ResNet-50 model is removed, resulting in a feature extraction model. This model is created by sequentially stacking all the layers of ResNet-50 except the last fully connected layer.

The feature extraction model is set to evaluation mode using model.eval(), indicating that no training is required and the model should only perform inference.

The preprocessed image is passed through the feature extraction model using features_model(preprocessed_image.unsqueeze(0)). The unsqueeze(0) function is used to add an extra dimension to the image tensor to match the expected input shape of the model.

With torch.no_grad(), the features are extracted from the preprocessed image. The resulting features are a 3-dimensional tensor.

To make the features compatible with further analysis, they are flattened using features.view(features.size(0), -1). This step reshapes the tensor into a 2-dimensional array.

Finally, the flattened features are converted to a NumPy array using .numpy() and returned as the output of the function.

By extracting CNN features from the input images, the project aims to capture high-level representations of the image content, which can be used for subsequent analysis, such as classification or clustering.

Local Binary Patterns (LBP)

In the project's second phase, the Local Binary Patterns (LBP) algorithm is employed to extract features from the input images. The `extract_lbp_features` function is responsible for performing this feature extraction process.

By incorporating LBP features into the project's pipeline, the goal is to capture local texture information from the images, which can provide valuable insights for various image analysis tasks, such as texture classification or segmentation.

Histogram of Oriented Gradients (HOG)

The project's second phase involves extracting features using the Histogram of Oriented Gradients (HOG) algorithm. The `extract_hog_features` function is responsible for performing this feature extraction process.

By incorporating HOG features into the project's pipeline, the aim is to capture information about the image's edges, corners, and overall structure, which can be beneficial for tasks such as object detection, pedestrian recognition, or other computer vision applications.

Gray-Level Co-occurrence Matrix (GLCM)

The project's second phase involves extracting features using the Gray-Level Co-occurrence Matrix (GLCM). However, it has been observed that this feature extraction technique does not perform well for the project's requirements.

The GLCM-based feature extraction is typically used to capture texture information in an image by analyzing the spatial relationship between pixel intensities. It calculates the co-occurrence of pixel values at various offsets and directions within an image.

It is important to note that different feature extraction techniques may have varying effectiveness depending on the specific characteristics of the dataset and the nature of the problem being addressed. In this case, the GLCM-based features did not yield satisfactory results, and alternative approaches may be explored for improved performance.

3- Model training

In classification task, it was required to types
1- per pixel classification
2- flooded image or not flooded

1- per pixel classification

In addition to the feature-based classification approach discussed earlier, per pixel classification can also be performed using clustering techniques such as K-means and Gaussian Mixture Models (GMM). This approach aims to assign each pixel in an image to be flooded or not flooded based on its similarity to other pixels in the dataset.

2- flooded image or not flooded

Model Training using Deep Learning

In the model training phase, a deep learning approach is employed to train a Convolutional Neural Network (CNN) for image classification. The code provided outlines the architecture of the CNN, the optimization process, and the evaluation of the trained model.

1. Model Architecture:
   - The CNN model consists of multiple convolutional layers (conv1 to conv4) followed by ReLU activation functions and max-pooling layers (pool1 to pool4).
   - The output of the last pooling layer is flattened and passed through fully connected layers (fc1 and fc3) with ReLU activation functions.
   - A dropout layer with a dropout rate of 0.5 is applied to prevent overfitting.
   - The final layer (fc3) has a single output unit for binary classification, predicting the probability of belonging to a specific class.

2. Device Configuration:
   - The model is trained on either a GPU (CUDA) if available or the CPU.
   - The device is set using `torch.device` to ensure efficient computation.

3. Loss Function and Optimizer:
   - The loss function used is the Binary Cross-Entropy with Logits Loss (BCEWithLogitsLoss).
   - This loss function combines sigmoid activation and binary cross-entropy, suitable for binary classification tasks.
   - The Adam optimizer is utilized with a learning rate of 0.0001 to update the model parameters.

4. Training Loop:
   - The model is trained over a specified number of epochs (num_epochs), iterating through the training dataset.
   - Within each epoch, the training data is loaded in batches using a data loader (train_loader).
   - The optimizer's gradients are reset, and the model's forward pass is computed to obtain the predicted outputs.
   - The loss is calculated by comparing the predicted outputs with the true labels.

- Backpropagation is performed to compute the gradients, and the optimizer updates the model parameters.
- The training loss is recorded for each epoch.

5. Evaluation:
   - After each epoch, the model's performance is evaluated on the test set.
   - The model is switched to evaluation mode using `model.eval()` to disable gradient computation and dropout.
   - The predictions are obtained by applying the model to the test set, and the predicted labels are rounded using sigmoid activation.
   - Accuracy, precision, recall, and F1 score are calculated using the predicted labels and true labels.
   - The evaluation metrics are printed for each epoch.

6. Best Model Selection:
   - The best model is determined based on the highest accuracy achieved on the test set.
   - If the current model surpasses the previous best accuracy, the model's state is saved as "best_model.pt".

The weights and hyperparameters used in the model are as follows:

- CNN Architecture:
  - Number of Convolutional Layers: 4
  - Number of Fully Connected Layers: 2
  - Dropout Rate: 0.5

- Optimization:
  - Learning Rate: 0.0001
  - Optimizer: Adam

The training process aims to optimize the model's parameters to minimize the loss function and improve classification accuracy on the test set. The best model obtained during training is saved for further use.

```
Best Model Accuracy =  81.622 %
```

<u>Model Training using classical technique</u>
In addition to the deep learning approach, classical machine learning techniques were also employed in the model training phase. The following classical techniques were utilized for image classification:

1. Support Vector Machine (SVM) with Linear Kernel:

2. K-Nearest Neighbors (KNN):
3. Random Forest:
4. Naive Bayes:
5. Logistic Regression:

These classical techniques leverage the extracted features to learn classification patterns and make predictions on unseen images.

4- Evaluation:

we try each feature with all classifiers

1- CNN features with SVM

confusion matrix

	0	1
0 (non flooded)	85	2
1 (flooded)	6	92

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.93	0.98	0.96
1	.98	0.94	0.96

Accuracy : 0.96
Omission Error : 0.04
Commission Error : 0.04

2- CNN features with KNN

confusion matrix

	0	1
0	87	0
1	3	95

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.97	1	0.98
1	1	0.97	0.98

Accuracy : 0.98
Omission Error : 0.02
Commission Error : 0.02

3- CNN features with Random Forest

confusion matrix

	0	1
0	87	0
1	4	94

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.96	1	0.98
1	1	0.96	0.98

Accuracy : 0.98
Omission Error : 0.02
Commission Error : 0.04

<u>4- CNN features with Naive Bias</u>

confusion matrix

	0	1
0	87	0
1	4	94

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.96	1	0.98
1	1	0.96	0.98

Accuracy : 0.98
Omission Error : 0.02
Commission Error : 0.02

<u>5- CNN features with Logistic Regression</u>

confusion matrix

	0	1
0	87	0
1	6	92

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.94	1	0.97
1	1	0.94	0.97

Accuracy : 0.97
Omission Error : 0.06
Commission Error : 0.00

<u>1- Lbp features with SVM</u>

confusion matrix

	0	1
0 (non flooded)	69	18
1 (flooded)	24	74

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.74	0.79	0.77
1	.80	0.76	0.78

Accuracy : 0.77
Omission Error : 0.23
Commission Error : 0.23
Macro Average F1 Score : 0.77

2- Lbp features with KNN

confusion matrix

	0	1
0	78	9
1	16	82

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.83	0.90	0.86
1	0.90	0.84	0.87

Accuracy : 0.86
Omission Error : 0.14
Commission Error : 0.14

3- Lbp features with Random Forest

confusion matrix

	0	1
0	80	7
1	11	87

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.88	0.92	0.90
1	0.93	0.89	0.91

Accuracy : 0.90
Omission Error : 0.1
Commission Error : 0.1

4- Lbp features with Naive Bias

confusion matrix

	0	1
0	69	18
1	23	75

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.74	0.79	0.77
1	0.80	0.76	0.78

Accuracy : 0.77
Omission Error : 0.23
Commission Error : 0.23

5- Lbp features with Logistic Regression

confusion matrix

	0	1
0	72	15
1	31	67

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.70	0.83	0.76
1	0.82	0.68	0.74

Accuracy : 0.75
Omission Error : 0.17
Commission Error : 0.15

<u>1- HOG features with SVM</u>

confusion matrix

	0	1
0 (non flooded)	79	8
1 (flooded)	30	68

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.72	0.91	0.81
1	.89	0.69	0.78

Accuracy : 0.79
Omission Error : 0.16
Commission Error : 0.12
Macro Average F1 Score : 0.79

2- HOG features with KNN

confusion matrix

	0	1
0	87	0
1	98	0

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.47	0.1	0.64
1	0.00	0.00	0.0

Accuracy : 0.47
Omission Error : 0.53
Commission Error : 0.1

3- HOG features with Random Forest

confusion matrix

	0	1
0	71	16
1	16	82

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.82	0.82	0.82
1	0.84	0.84	0.84

Accuracy : 0.83
Omission Error : 0.17
Commission Error : 0.18

<u>4- HOG features with Naive Bias</u>

confusion matrix

	0	1
0	75	12
1	21	77

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.78	0.86	0.82
1	0.87	0.79	0.82

Accuracy : 0.82
Omission Error : 0.18
Commission Error : 0.17

5- HOG features with Logistic Regression

confusion matrix

	0	1
0	80	7
1	30	68

precision , recall and f1 score

	Precision	Recall	F1-score
0	0.73	0.92	0.81
1	0.91	0.69	0.79

Accuracy : 0.80
Omission Error : 0.20
Commission Error : 0.18

5- Choosing Model

based on the results we chose the CNN features with random forest model because it gives us the highest accuracy between all models.