Instructor : Pr Hanna Abi Akl

Djihene Beladjine
Yilun DU
Christy Lazar

Deep Learning project: Image Classification

I. INTRODUCTION

Image classification is one of the most important applications of computer vision, which involves predicting the class of an image.

For our deep learning project, we used a rice type dataset from Kaggle.

In this study, we will create, train and evaluate different Deep learning and machine learning models (KNN and SVM), and thus for predicting the class of a rice image.

GitHub Repo: Jupyter notebook and detailed analysis are available here

II. LOADING THE DATA

Dataset home page on Kaggle: Rice Image Dataset (kaggle.com)

Since the dataset contains a large number of images, to facilitate our development environment, we chose to host our Jupyter notebooks on Kaggle so that we can load the whole dataset once run all codes remotely.

Here are the links to our notebooks, each of which implement a distinct model to do image classification:

CNN: Image_classification_CNN (kaggle.com)
KNN: Image_classification_KNN (kaggle.com)
SVM: Image_classification_SVM (kaggle.com)

III. DATA ANALYSIS

Our dataset contains 75 000 color images (RGB) of size 250x250 pixels, with 5 files representing a type of rice (5 classes) , with 15 000 images per class.

In this study, we will implement an image classification by using different models and Framework:

- ❖ Convolutional Neural Network using Pytorch : The most popular deep learning model, particularly in the area of image classification, with their ability to automatically learn features from raw pixel data.
- ❖ Support Vector Machines (SVM) proves efficacious for image classification due to its inherent ability to delineate complex decision boundaries. By leveraging a linear kernel, SVM effectively separates features in high-dimensional space, accommodating the flattened image data. SVM's capacity to handle high-dimensional feature spaces aligns with image data's inherent complexity.
- ❖ K-nearest Neighbors(KNN): It is a straightforward algorithm used for classification and regression. It works by finding the most similar data points to make predictions. However, it's sensitive to the number of neighbors chosen and the distance measure used.

IV. IPRE-PROCESSING

The preprocessing part is done differently for each model.

1. For the CNN model using Pytorch

a. Importing the libraries :

For the CNN model, we will mostly use Torch and Torchvision libraries for handling, building the model, training and evaluating it. We also need sklearn.metrics , seaborn and matplotlib for plotting and visualizing some images and results, NumPy for data transformation.

b. Dataset preparation :

First, we will define a transformation to resizing and normalizing the dataset :

- Using the **transform.Compose** function to apply the transformation,
- We have 3 transformations: resizing the size of the image, from 250x250 pixels to 128x128 pixels. Then, we convert the type images from the dataset into Tensors. And finally, we normalize to help CNN perform better . Since our images are RGB, which means they have three channels -Red, Green and Blue - we pass in 3 parameters for both the mean and standard deviation sequence. We choose the mean and std values `[0.485, 0.456, 0.406]` and `[0.229, 0.224, 0.225]` respectivelly, which are derived from the ImageNet dataset, a large-scale dataset commonly used for training models for an image classification. By using the values of these types of dataset, we can ensure consistency with normalization, stability, and transferability.
- We use the **ImageFolder** class to create the dataset, which automatically arranges images into classes based on the directory structure. Each class contains images of that class. The final dataset will be **rice_dat**, with 75 000 images

Second, the splitting the dataset into train, validation, and test sets :

We split the loaded dataset **rice_dat** into three sets : training, validation and testing. This step is crucial for training and evaluating deep learning (or machine learning) models effectively. The splitting is done following the proportions :

- 70% for the training set,
- 15% for the validation set,
- The remaining 15% for the testing set.

To perform this splitting, we use the **random_split** function from Pytorch, by assigning the data points to each subset based on the specified proportion. The results we have are : **train_dataset, valid_dataset,** and **test_dataset.**

At last, we will create data loaders :

The creation of data loaders allows us to efficiently load and batch the training, validation and testing set during the training process. They are responsible for loading the image datasets and providing them in batches to models. We will precise the Tuning parameters of these data loaders :

- **Batch_size**: The number of training samples in one iteration, or one forward/backward pass. We give the argument 64, which means we are getting 64 images at every iteration of training the network. Each batch is training the network successively and considering the updated weights from the previous batches.
- **Num_workers:** it allows us to speed up the training process by using machines with multiple cores. We set it to 4, which means there are 4 workers simultaneously putting the data into the computer's RAM.

In our case, we created three data loaders, one for each subset: **train_loader, valid_loader, test_loader.**

2. *For the KNN model*

- For each rice type, a subset of training images is randomly selected and displayed, allowing for visual inspection of the dataset's diversity and quality.
- The number of samples per class is controlled to ensure a manageable visualization size, with each class having five samples displayed in the visualization grid.

3. *For the SVM model*

To ready the dataset for SVM classification, RGB images were converted to PyTorch 3D tensors, then flattened into 1D arrays. Standardization of pixel intensities was applied for consistency. These steps ensure compatibility with SVM's linear kernel and preserved spatial information. This preprocessing streamlined data representation,

enhancing SVM's effectiveness in accurately classifying the five rice types based on images.

V. MODEL : BUILDING, TRAINING AND EVALUATION

CONVOLUTIONAL NEURAL NETWORK (CNN) MODEL

- ***building the Neural network model***

We will define the architecture of a convolutional neural network. **ConvNeuralNet** is our custom model class which has three convolutional layers and two fully connected (fc) layers. Each convolutional layer has a ReLU activation function and Max Pooling, as well as the first fully connected layer has a ReLU activation function.

- The 3 convolutional layers apply a 4x4 convolution to the input images, and the padding of 1 which maintains the input size.
- The first **conv1** has the input features set to 3 channels due to the fact that we are working with RGB images. The output features have 16 channels, which allows the network to start capturing the basic, low-level features like gradients and edges. The choice of the input features corresponds to the number of output features of the previous layer, to facilitate the flow of information through the network. The output features increase allows the network to learn increasingly complex and abstract features.
- Applying the Rectified Linear Unit (ReLU) activation function to introduce non-linearity.
- Applying the max pooling with a 2x2 kernel and a stride of 2, which means we are turning our images into 2x2 dimensions while retaining important features.
- The first fully connected layer **fc3** flattened the last convolutional layer (after max pooling, which has 15x15 feature maps). The latter is then mapped to a higher-dimensional space (512 dimensions). The second fully connected layer **fc4** produces the final class predictions
- The **forward** method defines the forward pass of the model.

Define a Loss function and optimizer

The **CrossEntropyLoss** loss function is the most used when training classification problems. It combines the log softmax and the log-likelihood.
We are using the popular optimizer: **Adam.** it adapts the learning rate for each parameter, making it suitable for a wide range of tasks.

- ***Training and validation of the model***

In this section, we implement the training and the validation loop,which runs for a specified number of epochs 10, to train the CNN model on the rice image dataset.

For each epoch, the following steps are performed:

Training : We set the model to training mode, and we iterate through the training loader in batches. Then, we perform a forward propagation through the model, clearing the optimizer's gradients to prevent accumulation from previous batches, computing the gradients of the loss function and updating the model parameters using the optimizer. After that, we calculate the training losses.

Validation: We set the model to evaluation mode. The validation loss as well as the accuracy are computed using the validation loader. There is also the loss which is accumulated, and the correctly predicted samples are counted. The calculation of the validation accuracy is done by dividing the correct predictions by the total number of validation samples.

- ***Result :*** we can see from the performance of the modem that the accuracy of prediction is very good, where the accuracy in the 10th epoch is 99.2%. From the plotting, we can see that the training loss is close to zero which means that the model can fit the training data well. The validation losses are decreasing which suggests that the model is learning useful information from the training data and is generalizing well to unseen data.

- ***Evaluating the model on the test dataset***

With our model trained and ready to go, let's test it on 11 250 images from the test dataset by calling the test loader.

- ***Result :*** the model achieved an overall test accuracy of 99%, which is an impressive performance. This indicates the ability of the model to accurately classify unseen data, which in our case are rice images, and also generalize well to new data.

- ***Prediction on images*** **:** we applied this testing code on 5 images. The result was obviously good since it predicted the right label corresponding to each image.

- ***The confusion Matrix Analysis***

Evaluating the model's performance on the test dataset by using the confusion matrix, which helps us assess how well the model classified each class, as well as where

misclassification occurred. Observing our confusion Matrix, each class has almost all of them the argument 1, which is 100% correctly predicted.

- ***Saving the model***

  We will save the model with the **save( )** mode. By doing this step, we will preserve the learned weights and biases that the model has acquired during the training.

K-NEAREST NEIGHBORS (KNN)

- **Training the kNN Classifier**
  - ❖ The kNN operates based on the principle of similarity, where the class (or value) of a new data point is determined by the majority class (or average value) of its k nearest neighbors in the feature space.
  - ❖ We train the kNN classifier using the fit method, providing the flattened training images and corresponding labels as input.
- **Evaluation**
  - ❖ We calculated the accuracy of the classifier by comparing the predicted labels with the ground truth labels.
- **Accuracy Assessment**
  - ❖ The overall accuracy of the kNN classifier on the test dataset is 95.3%, indicating that it correctly predicted the class labels for the majority of the samples

From the results obtained by using the KNN model, we observed that the overall accuracy of the kNN classifier on the test dataset is **94.3%**, indicating that it correctly predicted the class labels for the majority of the samples.

Additionally, we generated a classification report to get a detailed evaluation of the classifier's performance, including metrics such as for each class : *Precision*, *Recall* and *F1-score.*

SUPPORT VECTOR MACHINES (SVM)

In training the SVM for image classification, the process involves finding the optimal hyperplane that maximizes the margin between different rice types in the feature space. Using the flattened image data as input, SVM iteratively adjusts the hyperplane parameters to correctly classify training samples while maximizing the margin.

Through mathematical optimization techniques, SVM identifies the hyperplane that best separates the rice types, ensuring robust classification. This training process ensures that the SVM model effectively learns the discriminatory patterns present in the image data, facilitating accurate classification of rice types during inference.

VI. CONCLUSION

Our work covered the different steps of image classification, using Deep learning model, as well as two machine learning for the comparison, which are : Convolutional Neural Network model (CNN), Support Vector Machines (SVM) model and K-nearest neighbors model (KNN) .

Image classification is a fundamental task in vision recognition that aims to understand, as well as to categorize an image as a whole under a specific label.

In our case , for the two Machine Learning models, the KNN model made a prediction with an accuracy of 94.3%, while that for the SVM model was 96.7%. As for the Deep Learning model, we can discern that **the CNN model performs really well with an accuracy of 99%,** using a large image dataset of 75000 color images. This is an expected result, because the Convolutional Neural Networks (CNNs) require less preprocessing, as they can spontaneously learn hierarchical feature representations from raw input images. They are also known for their scalability, exploit spatial relationships in images, as well as effectively process large amounts of visual data, which is an indispensable tool in computer vision applications.