Data Fusion Project Group 2

i6220668 Ayse Arslan,
i6258699 Louis Meeckers,
i6179318 Ali Al-Saeedi,
i6192579 Yonghui Guo,
GitHub Repo: https://github.com/louismeeckers/data-fusion-g2

March 2023

Contents

1 Introduction

In this project we are developing models that perform binary classification of tomato seedlings based on images taken of these seedlings. In total, we implemented **six approaches: two high-level fusion, three mid-level fusion and one federated learning approach**.

The two high-level fusion approaches follow the same idea: Each approach trains two CNNs to classify a plant, where the input data for the first model is the color images and for the second model it is the side view images. Upon obtaining prediction labels for a single plant from both models, the decisions are combined using various voting techniques.

For the mid-level fusion, we employed three distinct techniques to extract features from the two types of images (color and side), to combine features from different data sources. This combined data was used as input for a classifier to classify the plants. The first two feature extraction methods involved the use of randomly initialized CNN and Local Binary Patterns (LBP). The third approach for mid-level fusion incorporated multiple methods, including Oriented FAST and Rotated BRIEF (ORB), Binary Robust Invariant Scalable Keypoints (BRISK), and Scale Invariant Feature Transform (SIFT). The extracted features then were merged to form input data for an additional classifier.

For federated learning, we implemented horizontal federated learning, where each folder of the data set is a client. The model used for the classification is again a CNN.

As mentioned during the lecture, there is room for improvement when determining labels directly based on the average of the labels provided by the four experts, primarily due to the potential differences in credibility among the experts' judgments, or in other words, the lack of agreement among the labelers. To solve this problem, we proposed a solution employing Cohen's Kappa method (section 2.2.1).

To gain an overview of all the approaches implemented in this project, consider the following table:

High-Level Fusion		Mid-Level Fusion			FL
Rand. init. CNN	Pre-trained CNN	Rand. init. CNN	Local Binary Pattern + kNN	Mult. Methods + SVC	Horiz. FL + CNN
Cohen's Kappa	Cohen's Kappa	Cohen's Kappa	Cohen's Kappa	Cohen's Kappa	Cohen's Kappa

Table 1: Overview of approaches

The last row represents the labels that were used for the classifiers.

2 Data

2.1 Data description

The data consists of eight folders: A1, A2, A4, A6, B1, B2, B3 and B4. Each folder contains 126 or 127 folders which represents a plant. However, not all plants in the folders are used because some labels are missing for some of the plants. Therefore, a csv file is provided with information about the location of the images in the folders and the experts labels. Each plant is represented by two images: A top image of the plant and a side image as shown in figure 1a and 1b. Colored images have 3 color channels [R,G,B], while side images are grayscales and can be processed using one channel. For this problem four experts gave a score from 1 to 4, where 1 is very good, 2 is good, 3 is abnormal and 4 is not sprouted.

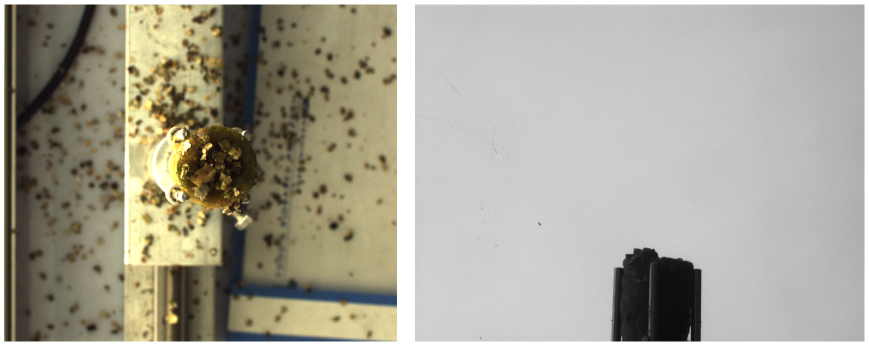

(a) Colored image (b) Side image

Figure 1: Instance example

2.2 Labels

One of the tasks in this project was to convert the labels into binary labels where a label represents a normal plant and the other represents an abnormal one. However, we found out that not all experts are equally reliable, which pushed us to use several techniques to ensure the final calculated label is as representative as possible. Normalization and weighted majority voting were applied on each label. In this section, we will further explain how our final label is generated.

2.2.1 Cohen's Kappa

To measure the reliability of each expert, we used Cohen's Kappa. This statistic measures the agreement between two experts. Cohen's Kappa is a continuous value that ranges from 0 to 1, where 0 is no agreement between the two experts and 1 is perfect agreement. To measure the Cohen's Kappa, we first need to measure what an expert says compared to the other. Therefore, we start by creating a contingency table, which shows what expert 1 says when expert 2 votes on a specific label. As an example, consider the contingency of experts 1 and 2 in table 2.

		Expert 2			
		1	2	3	4
Expert 1	1	0	10	10	0
	2	10	20	10	10
	3	5	10	15	10
	4	0	10	10	10

Table 2: Experts contingency table

Rows represents what expert 1 voted while columns represent what expert 2 voted. For instance, row 2 column 3 means that in 10 instances, expert 1 voted label 2. Meanwhile, expert 2 voted label 3. The next step would be to calculate the observed agreement, p_o, which is the number of time the experts agreed on a certain label divided by the total number of votes. Then we calculate the probability of the two experts agreeing by chance which is the probability of random agreement. To calculate this, consider label 2. Expert 1 voted 2 50 times in 140 times, which means this expert votes label 2, 41% of the time. On the other hand, expert 1 voted 2, 50 times in 140 which is again 41%. This means that the expected probability of both experts choosing label 2 by chance is the product of the two probabilities, or 41% $\times$ 41% which is about 17%. We calculate this value for each label and take the sum which gives us p_e, the overall random agreement probability. The Cohen Kappa between two experts is calculated as shown in equation 1.

$$\kappa_{a,b} = \frac{p_o - p_e}{1 - p_e} \tag{1}$$

where $\kappa_{a,b}$ = Cohen Kappa value between expert **a** and **b**

p_o = Probability of the observed agreement

p_e = Overall random agreement probability

2.2.2 Expert weights

As we wanted to measure how reliable each expert is, we calculated the Cohen's kappa coefficient for each pair of experts, and then we summed the coefficients for each expert and divided it by 3 (number of relations between the expert and the other experts) to give us a weight for an expert. The formula for calculating experts weights is shown in equation 2. For instance, if we want to measure the weight for expert 1, we first have to calculate the Cohen kappa between expert 1 and all the other experts: $\kappa_{1,2}$, $\kappa_{1,3}$, $\kappa_{1,4}$, then we divide it by n-1 which makes the value less than 1, this value is the Cohen's kappa of expert 1, this part is done in the numerator of equation 2. Then we need to calculate the weight of each expert with one constraint, which is: the sum of all weights should be equal to 1. To do that we need to divide the Cohen's kappa of each expert by the sum of all expert's Cohen Kappas which is done in the denominator of equation 2.

$$w_i = \frac{\frac{\sum_{j=1}^{n} 1_{i \neq j} \kappa_{i,j}}{n-1}}{\sum_{k=1}^{n} \frac{\sum_{j=1}^{n} 1_{k \neq j} \kappa_{j,k}}{n-1}} \tag{2}$$

$$\text{where} \qquad w_i = \text{Weight of expert } \mathbf{i}$$
$$n = \text{Number of experts}$$

2.2.3 Labels normalization

During our research, we found that we cannot use the average of the expert labels for the following reason: Assume the experts voted as follows: [1,1,1,4]. When calculating the average, we get a label of 1.75, which is closer to 2 than to 1. This is not valid since 4 in this case gets a higher weight than 1 just for being a big number, although three experts agreed that the plant is very good. To overcome this issue, we decided to normalize the labels in a way that 1 gets an equivalent weight to 4. We found that the best option is to normalize labels around 0 from [1,2,3,4] to [-3,-1,1,3]. It ensures that there is an equal difference between each label. For the conversion, the following formula was used:

$$\hat{l} = ((l-1) \times 2) - 3 \tag{3}$$

$$\text{where} \qquad l = \text{Label of an expert [1, 2, 3, 4]}$$
$$\hat{l} = \text{Normalized label [-3, -1, 1, 3]}$$

2.2.4 Labels with weighted majority voting

After generating the expert weights and the normalized labels, the next step is to generate the final label where the value can range between 0 and 1. However, it can be rounded such that 0 indicates that the plant is normal and where 1 indicates that the plant is abnormal. The continuous value of the label is used for the training of the neural network and is computed as shown in equation 4. The rounded value is used to calculate the accuracy of the classification of the plant.

$$L = \frac{\sum_{i=1}^{n} (\hat{l}_i \times w_i + 3)}{6} \tag{4}$$

where $\hat{l}_i$ = Normalized label of expert **i**

L = Calculated label based on all expert labels $[0 \rightarrow 1]$

After computing the final label of each instance of the dataset, we are ready to train the models.

3 Methodology

3.1 High-Level Data Fusion

3.1.1 Randomly initialized CNN

To perform high-level fusion we implemented a Convolutional-Neural-Network (CNN). One instance is classifying the color images and a second one is classifying the side images. The chosen architecture is a simplified version of VGG-16. It has been simplified to reduce the number of parameters of our network. The architecture is described in table 3. Those deep learning models learn during training to extract features from the input images to classify the instances.

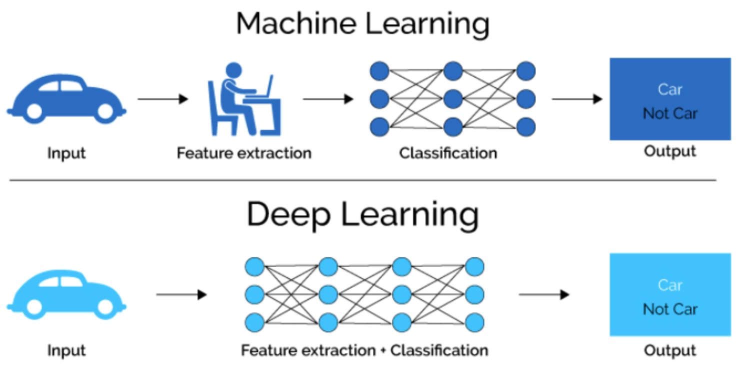


Figure 2: Machine Learning and Deep Learning comparison

The model has been trained with the hyperparameters described in table 4 and uses data augmentations to improve the performance and prevent overfitting:

- Random brightness and contrast (probability: 0.3)
- Rotation (angle limit: 35; probability: 1)
- Horizontal flip (probability: 0.5)
- Random resized crop (intensity: small; probability: 0.5)
- Resize (height: 480; width: 640)

Layer	Details	Output size
input	torch file	480 x 640 x 3
conv2d_1	3 x 3 x 16; batch norm; relu	480 x 640 x 16
max_pooling_1	2 x 2 max pool stride 2	240 x 320 x 16
conv2d_2	3 x 3 x 32; batch norm; relu	240 x 320 x 32
max_pooling_2	2 x 2 max pool stride 2	120 x 160 x 32
conv2d_3	3 x 3 x 64; batch norm; relu	120 x 160 x 64
max_pooling_3	2 x 2 max pool stride 2	60 x 80 x 64
conv2d_4	3 x 3 x 128; batch norm; relu	60 x 80 x 128
max_pooling_4	2 x 2 max pool stride 2	30 x 40 x 128
conv2d_5	3 x 3 x 256; batch norm; relu	60 x 80 x 256
max_pooling_5	2 x 2 max pool stride 2	15 x 20 x 256
linear	dropout 0.5; relu	512
linear_1	dropout 0.5; relu	512
linear_2		1

Table 3: CNN architecture

Label	Value
epochs	30
batch_size	8
criterion	BinaryCrossEntropy
optimizer	SGD
learning_rate	0.001
momentum	0.9
data_augmentation	True

Table 4: Hyperparameters

To predict the binary class of the instance, we take the result of both classifiers and use weighted majority voting to determine the final label. The weight given to each classifier is relative to the accuracy returned by each model.

3.1.2 Pre-trained CNN

We also implemented another approach of high-level fusion, where we utilized a pre-trained model. The MobileNetV2 model is commonly used for image classification and object detection tasks. It is built using depth-wise separable convolutions, which significantly reduce the number of parameters and computational cost compared to traditional convolutions [1]. It uses a feature called ”linear bottlenecks”, which helps to preserve information flow [2].

The idea for this approach is the same as for the randomly initialized CNN:

Two CNNs are used to classify a plant, using the colored images for one model and the side view images for the other model. The decisions of each model for a certain plant are then fused at the end.

We used transfer learning with the MobileNetV2 as the feature extractor by adding it as the first layer of a custom sequential neural network. The output of the model is flattened and then connected to a dense layer with 128 units and a ReLU activation function. A dropout layer is added to prevent overfitting, and a final dense layer with a sigmoid activation function is used to output the binary classification prediction. The last few layers of the model are unfrozen for fine-tuning and a custom fully connected neural network is added on top of it for the final classification task. The model is fine tuned during four epochs with the use of the Adam optimizer and the binary cross-entropy loss function.

To combine the decisions of both models, three different voting methods were implemented: Weighted voting (by giving the model with higher accuracy higher weight), majority voting, and Bayesian Consensus.

3.2 Mid-Level Data Fusion

3.2.1 Randomly initialized CNN

For one approach of mid-level fusion, we decided to use a CNN model with two input branches (as illustrated on figure 3) responsible to extract the features of their respective input. The first branch takes as input the color image, and the second branch the side image. Then the features resulted by both branches are merged and processed to predict the final label of the instance with the use of linear layers.

We used the same architecture described in table 3, but duplicated the first tenth layers to create the second branch. The features of both branches are merged before the first linear layer.

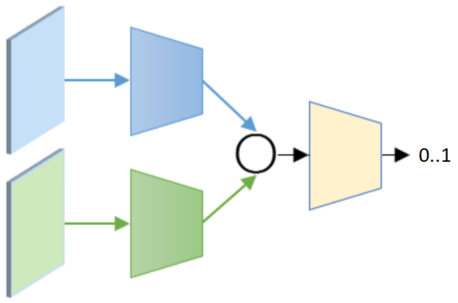


Figure 3: Mid level fusion

Source: "RFBNet: Deep Multimodal Networks with Residual Fusion Blocks for RGB-D Semantic Segmentation"

3.2.2 Local Binary Patterns

Another technique we implemented for mid-level fusion was extracting features using Local Binary Patterns. Local Binary Patterns(LBP) is a computer vision algorithm that extracts numerical features from images. The way LBP works is that a pixel is compared with other neighbouring pixels that are in the range of a specified radius. Using LBP, we extracted features for both colored and side images and used K-nearest-neighbours to build a classifier. We used the classifier to classify images from both side and colored images in separate models. Then we used mid-level fusion to concatenate features from both types of images and we built a model for the concatenated data.

3.2.3 Feature Extraction Using Multiple Methods

Since mid-level data fusion is a technique that combines data at the feature level, our goal is to improve the performance of the model by enhancing the efficiency of feature extraction. For the third mid-level fusion apparoach, we utilize three different algorithms—Oriented FAST and Rotated BRIEF (ORB), Binary Robust Invariant Scalable Keypoints (BRISK), and Scale Invariant Feature Transform (SIFT)—to extract features from images. Considering the accuracy of these algorithms, we experimented with two feature combination methods: Concatenation and weighted average. By employing these methods, we successfully achieved our goal of improving the stability and accuracy of the model for classification tasks.

3.2.3.1 Scale Invariant Feature Transform

SIFT is a widely-used computer vision algorithm designed for the detection and description of local features in images [3]. Its core involves identifying extrema within scale-space, extracting their position, scale, and rotation invariant. SIFT's primary objective is to locate key points, or feature points, across multiple scale-spaces and calculate their orientation. These key points show remarkable stability, remaining consistent in spite of changes in illumination, affine transformations, and noise. The SIFT algorithm can be broken down into four main stages [4]:

1. Scale-space extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint descriptor

3.2.3.2 Oriented FAST and Rotated BRIEF

ORB is an alternative to the SIFT algorithm that offers similar matching performance while demonstrating greater robustness to image noise, making it well-suited for real-time applications without sacrificing computational efficiency [5].

The ORB algorithm integrates the FAST and BRIEF approaches, utilizing a scale pyramid for the image and incorporating orientation measurements to achieve both scale and rotation invariant for key points.

3.2.3.3 Binary Robust Invariant Scalable Keypoints

BRISK represents an enhancement of the BRIEF algorithm and is a feature descriptor based on binary encoding. Exhibiting robustness to noise, it maintains scale invariant and rotation invariant. The BRISK method is capable of efficiently extracting features, demonstrating advantages in terms of processing speed and matching performance. Its framework can be integrated with other key point descriptors and tailored to achieve optimal performance for specific tasks and requirements [6].

3.2.3.4 Experiments

We employed the bag of features approach for feature extraction, followed by SVC for image classification. Firstly, we utilized the ORB, BRISK, and SIFT algorithms to extract features independently. Secondly, we applied k-means clustering to identify representative features. Finally, we employed SVC to classify the images.

We explored two distinct methods for data fusion. The first method, concatenation, involved merging the three feature arrays without taking into account the individual accuracy scores of the three algorithms. The second approach, weighted averaging, assigned weights to the three feature arrays and constructed a new composite array. According to our findings, the weighted averaging method got accuracy scores.

3.3 Federated Learning

We implemented horizontal federated learning with the use of the HFL algorithm. The algorithm instantiates a client for each folder of the dataset. Each client has access to its own private data. The algorithm also instantiates a server, which communicates with the clients to update and retrieve their weights. The clients and the server use the same model and architecture such that the parameters can be shared without compromising the data privacy.

We used the architecture described in table 3 and hyperparameters described in table 4.

The server initializes a global model and sends its weights to each client. Every client instantiates the same model with the weights received and then is trained on its own local data. Each client sends its updated weights to the server which aggregates and averages them to update the global model weights. The model weights are sent back to the clients. The process is repeated for multiple rounds, until our model converges. This allows us to have a global model that learns from several local models without having access to their own local data.

4 Results and Discussion

4.1 High-Level Fusion

To combine the decisions of the models, different voting techniques were implemented. They are listed in the table below.
The following table shows the different accuracies using the two types of models (randomly initalized CNN and pre-trained CNN). First, the accuracies for classifying the different images are listed, then the accuracies when combining the decisions.

	Rand. init. CNN	Pre-trained CNN
Color Cam	0.90	0.92
Side Cam	0.92	0.93
Weighted Voting	0.93	0.62
Majority Voting	-	0.72
Bayesian Consensus	-	0.72

Table 5: Accuracies of the two high-level fusion approaches

For the randomly initialized CNN the accuracy increased when combining the decisions of the two randomly initialized CNN models that were trained on the two types of images, whereas the accuracies for the pre-trained model decreased.

4.2 Mid-Level Fusion

The following table shows the different accuracies using the two classical approaches for mid-level fusion.

	Local Binary Pattern	Multiple Methods		
		ORB	BRISK	SIFT
Color Cam	0.89	0.52	0.55	0.71
Side Cam	0.95	0.70	0.70	0.70
Combined	0.96	0.70		

Table 6: Accuracies of the two classical mid-level fusion approaches

Here you can see that combining the features of the two images of plants increased the accuracy of the kNN classifier (Local Binary Pattern method), and for the SVC (multiple methods) the overall accuracy improved as well.

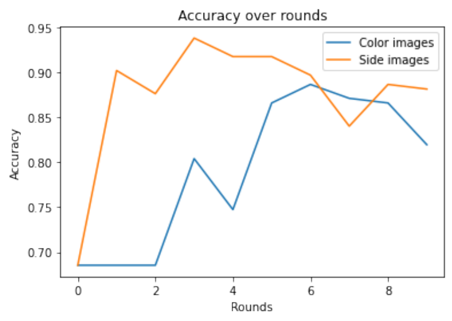
As the randomly initialized CNN does not predict the labels for the colored images and the side view images separately, but rather extracts their features and combines them and uses this bigger data set as input data right after, there is only one accuracy.

Accuracy randomly initialized CNN for mid-level fusion: 0.93

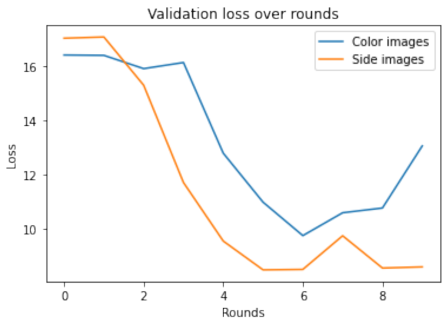
4.3 Federated Learning

Knowing that for each instance of the dataset, we have a color image and a side image, this method is composed of two experiments. The first one is trained on the color images and the second on the side images.

To ensure that our algorithm operates correctly, for each experiment the global model is evaluated on the test set of each dataset at the end of each round.



(a) Accuracy over rounds



(b) Validation loss over rounds

Figure 4: Federated Learning

We can observe on the figure 4a that the experiment using the side images improves faster than the model trained on the color images. Regarding the loss validation, we notice that it does not decrease significantly from the fourth round for the color images experiment and from the sixth round for the side images experiment. The model can thus be stopped at this moment and then returns around 88% and 92% accuracy respectively. Stopping the models before their validation loss increases is very important as it prevents them from overfitting the train set.

In order to test our algorithm, we had to instantiate one CNN per client and one for the server. Knowing that our model architecture is composed of millions of parameters, it is much faster to train each model on a GPU than a CPU. However, the GPU we have at our disposal has 4GB of memory which allows us to fit only one model at a time. To solve this issue our algorithm instantiates each client's model on the CPU, sends it to the GPU when it needs to be trained and sends it back to the CPU at the end of the training before starting to train the next client. This allows us to speed up the training of each client.

5 Conclusion

All in all, in this project we implemented six approaches from different data fusion techniques, namely: two high-level fusion, three mid-level fusion, and one federated learning. As the results show, Local Binary Patterns (LBP) achieved the best results where it classified images with an accuracy of 0.96. The randomly initialized CNN in both high-level and mid-level fusion seem to have the same accuracy of 0.93. However, when including the third decimal, then we can see that the CNN in high-level fusion performs better (high-level: 0.933, mid-level:0.928). The difference is so minimal though, it is not really significant. In this project, we investigated various data fusion techniques that helped in a classification task. Given the expert labels, we were able to create new labels that used different techniques such as Cohen's kappa, normalization and weighted majority voting. Then we used the labels and the different data fusion methods to create the classification methods. Finally, we succeeded in achieving high accuracy in the binary classification task which required us to classify if a plant is normal or abnormal.

References

[1] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[3] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, pp. 1150–1157 vol.2, 1999.

[4] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[5] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*, pp. 2564–2571, Ieee, 2011.

[6] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *2011 International conference on computer vision*, pp. 2548–2555, Ieee, 2011.