**Artificial Neural Network and Deep Learning**

Homework 1: Binary Image Classification

Team name: YellowSister

Giulia Huang (10832369) Hao Chen (10854895) Yizhou Wu (10926468) Hao Zhou (10748918)

**Abstract**

This case study focuses on implementing artificial neural networks for binary image classification of plant health. The dataset includes images categorized into two classes based on plant condition. We utilized pretrained models customized for binary classification, employing effective techniques in data preprocessing, training, and prediction phases to enhance performance. Results, measured by test dataset accuracy, indicate the EfficientNetv2 model achieved the highest accuracy of 0.82 in Phase 1 and the accuracy of 0.771 in Phase 2. We discuss model performance, efficiency, methodologies, and potential improvements.

1. **Introduction**

Given a dataset of 5200 plant images labeled as "healthy" or "unhealthy," our objective is to develop a model achieving the highest accuracy in predicting the correct class label [0,1] for images in a hidden test set. Initial examination revealed non-plant images in two categories throughout the dataset, impacting model training by introducing confusion. Another challenge is the uneven distribution of class samples, leading to a bias towards classifying as "healthy." To address these issues, we employ data augmentation during training to maximize dataset utility. This technique, applicable during testing as well, enhances the model's robustness and generalization to diverse input variations, improving overall performance. Instead of training a model from scratch, we opt for Transfer Learning, using a pre-trained model and leveraging knowledge from similar tasks. Throughout training, we incorporate callback functions to ensure a smoother and more efficient completion of the training phase.

1. **Methodology**

- Dataset Preprocessing

First of all, once loaded the dataset labeled with categorical values [“healthy”, “unhealthy”], we proceed with converting them to numerical values [0,1] that are accepted by architecture and in line with expected output format. Next, we removed from dataset all occurrences of two kinds of outliers manually, an alternative way is to compute the mean and the standard deviation of images on pixel values assuming a normal distribution, and exclude the ones with values standing outside the 3 standard deviations of the mean for the three-sigma rule.

Later we dealt with the problem of unbalanced classes sample, by preparing a data augmentation function, consisting of a series of random transformations to the data given as input, and feeding into it a set of less frequent class’s sample.

While implementing data augmentation to increase the volume of images within the 'unhealthy' class, we encountered certain challenges. Despite observing an improvement in validation accuracy and seemingly impressive test accuracy in local, the final submission results were disappointing. Several potential reasons for this discrepancy have been considered:

1. Augmented data in validation and test sets: The inclusion of augmented data in these sets might contribute to imprecise predictions during training and final evaluation on test set.

2. Excessive number of augmented images: We’ve generated a substantial number of augmented images, comprising about 39% of the 'unhealthy' class. Moreover, applying excessive augmentation may lead to loss of critical information from the original images. This extensive augmentation might have introduced noise and adversely affected the model's ability to generalize.

To rectify these issues, we implemented the following strategy:

1. Creation of test and validation sets from the original dataset: We initially formed these sets with a fixed number of samples and ensured an equal distribution of classes. This step aimed to provide a more reliable evaluation environment.

2. Reduction of class distribution disparity: Instead of aiming for a 50% balance in class distribution, we focused on minimizing the difference between the proportions of the two classes.

- Data Splitting

At early stages of development phase, we split the given dataset into 3 subsets, train, validation and test, in order to assess model’s performance locally resulting in immediate feedback. We tried two different data split approaches: splitting the dataset randomly with proportion train:0.81, validation:0.09, test:0.1, or manually splitting them based on class distribution for more accurate training and evaluation.

Later on, we split it into only train and validation dataset in order to exploit the potentiality of all available data, and indeed this brings a marginal improvement on performance.

* Learning Rate

We tested the model with different learning rates and it turns out that high learning rate leads the model to faster convergence during training but at the same time the model might oscillate and the training process becomes unstable. It is not convenient either to choose a low learning rate: even though it provides a more stable training process and it takes longer to find the optimal values. Moreover, we observed that it has the risk to get stuck in a local minima. We finally found out that the learning rate in range [1e-4, 1e-5] would be optimal in the case of EfficientNetV2.

* Optimizer

We adopted the Adam optimizer which offers advantages in terms of faster convergence and adaptability to varying learning rates.

* Callback functions

EarlyStopping: we used it to monitor the metric of validation accuracy during training and interrupts the training process if the metric stops improving.

ReduceOnPlateau: dynamically adjusts the learning rate when the monitored metric stops improving, helping the model to converge more effectively.

ModelCheckpoint: it preserves the best-performing model throughout the training process. In this scenario, we allow the model to undergo an extensive number of epochs without the imposition of early stopping to have a clear view of model’s progress and performance.

* Transfer learning and Fine tuning

We used the pretrained model EfficientNetV2 from Keras initialized with Imagenet weights and integrated to our fully connected layers. We tried different approaches:

1. Train the model having all the layers of the pretrained model frozen and using a lower learning rate to fine tune part of them
2. Freezing part of the layers of the pretrained model and train only those weights
3. Setting the entire model as trainable.

Finally, we find out that the latest approach has better results. In this case we can keep the model structure without using the trained weights, so that the model could fit the distribution of our own data.

* Data augmentation

This technique has been partially applied also at later phases in order to reduce overfitting and enhance model’s ability to generalize to a wider variations of the input:

* Train set data augmentation: consist at generating additional training samples by applying random transformations to the existing data. This increases the diversity of the trainings et and helps the model to generalize better. To do so, we added a preprocessing layer after the input layer.
* Test time data augmentation: consist at making the final prediction by averaging the predictions from different augmentations. We found that opting for a more restrained set of augmentation techniques has better results. Our hypothesis is that overly aggressive transformations might pose challenges for the model in accurately predicting the desired outcomes.

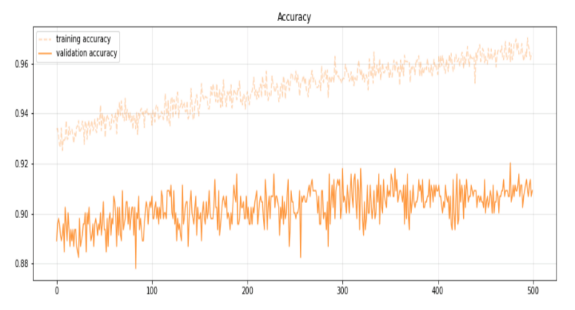
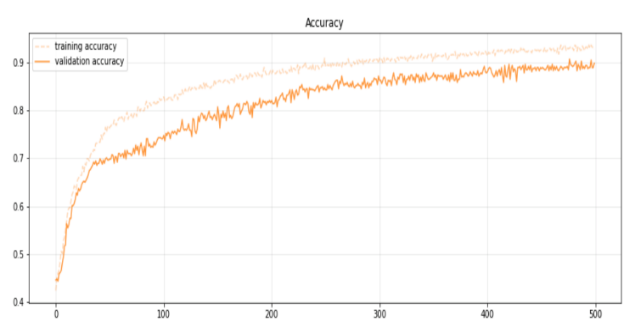
1. **Final adopted architecture**

We used a pretrained model from Keras library which is the EfficientNetV2 neural network for the binary classification task. The architecture is the following:

1. **Input Layer**: it takes inputs data with shape 96x96 pixels and 3 color channels.
2. **Resizing Layer**: it resizes the input images to the specified shape (224x224x3). We made this choice because EfficientNetV2 is pretrained on images of this shape.
3. **Processing Layer**: layer for data augmentation applied to the input, followed by a resizing layer.
4. **EfficientNetV2:** this is the backbone of the network, implementing the EfficientNetV2 architecture and we drop the dense layer at top of the pretrained network.
5. **Global Average Pooling Layer:** it is used for reducing the spatial dimension to 1x1 and channel dimension to 1280.
6. **Dense layer** followed by a **Dropout layer**: in order to introduce further non-linearity and the dropout layer to regularize the model and prevent overfitting.
7. **Output layer**: the final dense layer with 2 units represents the output layer for binary classification. Representing the probabilities of the input image belonging to each of the two classes
8. **Experimental Result**

We applied above mentioned techniques and measured the performance of models based on different ANN architectures, the result we obtained are described as following: EffiientNet reaching accuracy 0.82 during phase 1 and 0.77 during phase 2. MobileNetv1 reaching accuract 0.75 during phase 1. MobileNetV2 reaching accuracy 0.8 during phase 1 and 0.70 during phase 2. Xception reaching accuracy 0.74 during phase 1 and 0.45 during phase 2. The Hybrid model reaching accuracy 0.74 during phase 1.

From which the best performing model is based on EfficientNetV2 architecture, whose accuracy along the training phase on validation set is shown as following



1. **Conclusion**

In this case study, we practiced with various artificial neural network techniques, and experienced firsthand different convolutional neural network architectures. In particular, we observed that not all techniques will improve the performance of models, and some of them do with a significant increase in computational cost. In the end, we achieved an accuracy of 0.82 at our binary classification problem.

1. **Further improvements**

* Feature fusion layer

Feature fusion layer consists of combining features and integrating information extracted from various parts of the network. In our case, we concatenate the outputs from the Xception and MobileNetV1 models, directing this combined output to the next layer. Employing this approach, we observed a modest improvement in our validation accuracy, but not in test accuracy. Further attempts could be made with the EfficientNet model.

* Single output neuron

In our solution, we configured two neurons at output layer mapping to two classes, but as we are affronting a binary classification problem, we could adopt simply one neuron mapped to one of two classes, resulting in fewer parameters and less theoretical probability of overfitting.

* CAM / Grad-CAM

After training the model, we could resort the CAM (in case of the last layer before the output is GAP) or Grad-CAM (suitable for structure with fully connected layers at the end) to make some interpretations on the image area where helps the model make the decision, which also helps us to figure out whether the model is doing the right thing. We’ve tried to implement it but currently fail to extract the output of last convolutional layer, we will further manage to achieve it.

**Contributions**

Spirit leader : Giulia Huang

As the designated coordinator in our project, orchestrating the team's efforts and ensuring seamless collaboration. Her primary responsibilities revolve around organizing tasks, assigning roles, and maintaining a cohesive workflow among team members. Responsible for training EfficientNetV2.

Hao Chen :

Training ResNet (too many parameters which leads to overfitting, so discard it), MobileNetV2 with different structures, conduct the Grad-CAM methods in order to visualize the model decision (But fail).

Yizhou Wu :

Information retrievaling, development assisting, report writing.

Hao Zhou:

Training different Model like MobileNetV1, Xception,Baseline and Hybrid model Xception/MobileNet1 with different model structure, also compute the data balacing approches.