Deep Learning Project

Master 2 MIASHS (2023-2024)

Julien Bastian, **Walid Ghalleb**, **Ugo Zennaro**

Anthropologie, Sciences Sociales et Politiques (ASSP)
Université de Lyon, Université Lumière Lyon 2

Enseignant : Irina Proskurina
Classe : Atelier Data Science Deep Learning

1 Image Classification with Deep Learning

1.1 Image Classification

As human we can it can be easy, knowing characteristics of any object we want to classify to tell what an image is picturing. Indeed we are able to see in an image what the outlines are drawing in many perspectives and in a large range of distances. Doing image classification with deep learning we aim at doing the same thing. In computational approaches the picture is analyzed as the decomposition in every pixel. Therefore the main difficulty is to get from those individual elements the shapes and all the variations to which that images are subject.

1.2 Fully connected Neural Network for image classification

Fully connected Neural Network can be a good call for image classification if, as in more classical machine learning task every pixel, i.e. every column of the dataset, correspond to a specific feature. So, with standardized image it could be working. But most of image datasets aren't. It results in overfiting and a lack of consideration for image patterns. So we need to be able to structure the patterns recognition by considering the pixels with the context of the neighborhood. To do so we will use convolutional methods.

1.3 Convolution

Convolution aims at finding the pattern that are useful for classification by scanning the picture with, so called filters or kernels. They can will be sequentially multiplied element-wise to every bits of the picture. An illustration of this process would be :

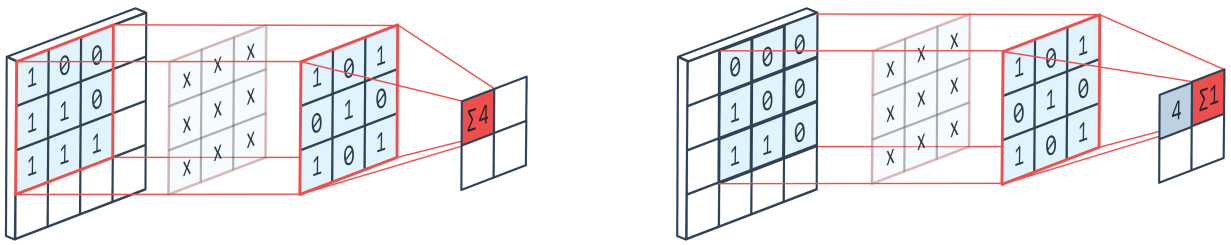


Figure 1: Example of a product of convolution in a convolutional neural network[1].

Formally the product of convolution is :

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(x-t)g(t)\,dt$$

with g the kernel and f the image. Thus the product of convolution allows to find pattern anywhere in the picture and having a max pooling invariant to shifts. The max pooling value being the maximum value of the output matrix.

Then we can then integrate convolutional layers into the architecture of our neuronal network. It can be understood as a search for pattern in the raw picture but when it goes deeper in the architecture we might want to understand how those pattern interacts together meaning the model is now able to understand the structure of what is pictured in the image.

1.4 Some important architectures

There are several famous architectures that mark milestones in the development of deep learning architecture for image recognition and CNN. We chose here three of those architectures to illustrate this, LeNet [LeCun et al., 1989], AlexNet [Krizhevsky et al., 2012] and ResNet [He et al., 2015].

[1] source : peltarion.com

LeNet [LeCun et al., 1989]

In 1989 Yann LeCun et al. proposed the original form of LeNet as one of the earliest convolutional neuronal networks and marks the emergence of the interest for deep learning. This architecture was train on digits extracted from handwritten zip codes. Those digits suffer from many handwriting variation so there was a need for generalization from the classifier. The architecture, illustrated in Figure 2, is quite simple by modern standard but yielded convincing results.

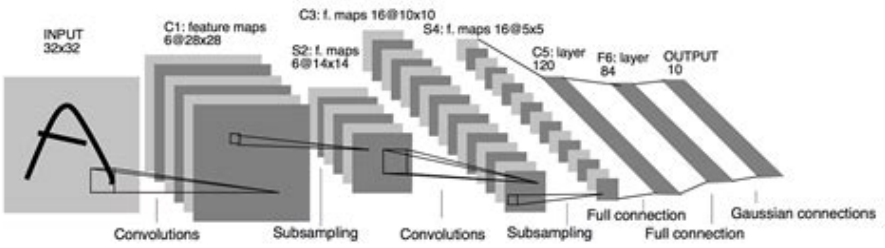


Figure 2: Architecture of LeNet[2].

Indeed, it resulted in an error rate of 1% on the test set which promoted the development of deep learning. The architecture is only made of two convolutional layers, two pooling layers and three dense layers. It allowed the demonstration of the power of convolution and convolutional layers to represent 2d data.

AlexNet [Krizhevsky et al., 2012]

In 2009 the ImageNet dataset [Deng et al., 2009] was introduce to act and promptly became a classical benchmark for image recognition methods and between 2010 and 2017 a annual contest was organized. The dataset was a reference for visual object recognition software researchers with large values of classes (≈ 20000) and of images (≈ 14 *millions*). In the year 2012 AlexNet [Krizhevsky et al., 2012] was introduced achieved a top-5 error of 15.3%, that was 10.8% points lower than that of the runner up. The original paper's primary result was that the depth of the model was essential for its high performance, which was computationally expensive, but made feasible due to the utilization of graphics processing units (GPUs) during training. Its architecture is quite similar with the one of LeNet. It mainly expands by a convolutional block of three convolutional layers and by being way wider at every step. Furthermore, the activation function are also different LeNet uses the sigmoid whereas AlexNet uses the more moder Rectfied Linear Unit (ReLu),

$$ReLu(x) = \max(0, x).$$

A comparison of the two architectures is presented in Figure 3.

ResNet [He et al., 2015]

In 2015, ResNet [He et al., 2015], short for Residual Network, was introduced. It is a deep neural network architecture that achieved top performance in the ImageNet dataset and won the associated contest in 2015. ResNet differs from past CNN by introducing skip connections, or "residual connections". These skip connections address the vanishing gradient problem commonly encountered in deep networks by providing alternative paths for gradient propagation allowing for deeper models. By stacking more layers and building a deeper model that was previously possible ResNet achieved new standards of performance in the image recognition realm.

1.5 Optimization Procedures

Optimization procedures design the different way by which the models of the parameters are inferred during the learning step. They usually are a variation of gradient descent and make the basis of

[2]source : medium.com
[3]source : wikipedia.com

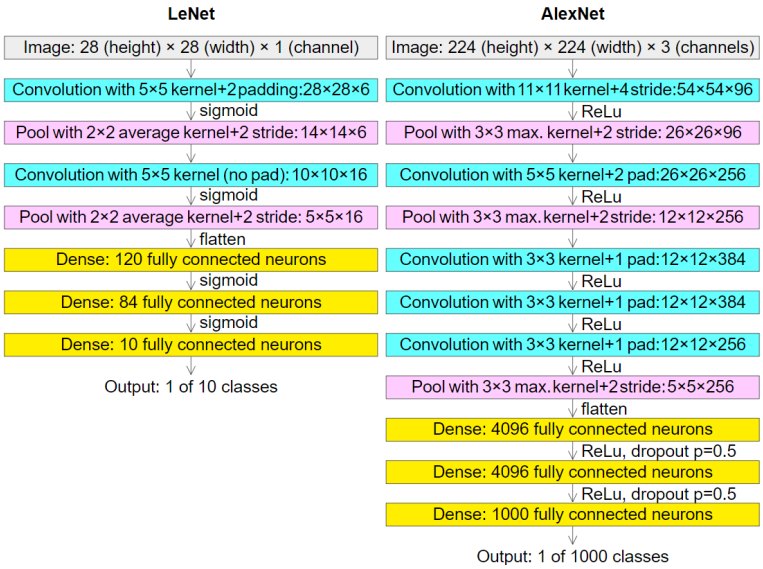


Figure 3: Comparison of LeNet and AlexNet architecture. We can see that the latter introduce more convolution layers and switch from sigmoid to ReLu for its activation function.[3].

the loss minimization. There exist a wide variety of methods, such as but not exclusively, Stochastic Gradient Descent (SGD), Momentum based gradient descent, RMSprop, AdaGrad [Duchi et al., 2011] or Adam [Kingma and Ba, 2017]. We will mainly focus here on SGD and Adam. The first because it is fundamental in deep learning optimization for its simplicity and similarity with the simple gradient descent (and still widely used). The second because it is probably the most used optimization algorithm at the moment (and it is the one we are using too).
At each iteration, SGD computes the gradient of the loss function with respect to a randomly chosen mini-batch of training samples. It then updates the model parameters in the opposite direction of the gradient scaled by a learning rate η:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \mathcal{L}(\theta; x_i, y_i)$$

where $\nabla_\theta \mathcal{L}(\theta; x_i, y_i)$ is the gradient of the loss function $\mathcal{L}$ with respect to the parameters θ evaluated on the mini-batch (x_i, y_i) of training data. This process is repeated for a prespecified number of epochs. Distinctly from SGD Adam adapts the learning rate for each parameter based on estimates of the first and second moments of the gradients. This adaptivity allows Adam to achieve more efficient optimization by automatically adjusting the learning rates according to the gradients magnitudes and directions. This adjustments allows a more efficient navigation of the loss landscape. This is done by using two changing parameters : the first moment, or moving average of the gradients, m_t and the second moment, or uncentered variance of the gradients, v_t. At each iteration t, they are updated in the following way :

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

where g_t is the gradient at iteration t, and β_1 and β_2 are the decay rates for the first and second moments, typically set to 0.9 and 0.999 respectively. Then the bias-corrected estimates of the moments are computed as:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Finally, the parameters θ are updated using the computed moments and a learning rate η:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where ϵ is a small constant added to avoid multiplicating by 0.

1.6 Some special layers

Apart from classical convolutional, dense or pooling layer it appears important to us in this report to present some additional layers that we are going to use and make important building blocks of state-of-the-art models. We consider here two of these layers : the Dropout layer [Hinton et al., 2012] and the Batch normalization (BatchNorm) layer [Ioffe and Szegedy, 2015].

Dropout is a regularization technique used to prevent overfitting. At each training iteration, dropout layers randomly set a fraction of input units to zero with a probability p, typically set between 0.2 and 0.5. This temporarily removes these units along with all their connections. Therefore, during each training iteration, a different random set of units are dropped out, leading the network to learn more robust features to represent the data and thus reducing overfitting. In this way it can be seen as training multiple subnetworks with shared parameters, which can lead to better generalization performance. During inference, dropout is typically turned off or scaled down to allow all units to contribute to the predictions.

BatchNorm is used to improve the training speed and stability of neural networks. Batch normalization calculates the mean and variance of each feature dimension within a mini-batch of training examples. It then normalizes each feature dimension using these batch statistics, followed by a scale and shift operation using learnable parameters θ_1 and θ_2. The normalized output $\hat{x}$ for a feature x is given by:

$$\hat{x} = \frac{x - E(x)}{\sqrt{V(x) + \epsilon}} \cdot \theta_1 + \theta_2$$

Batch normalization is typically applied before to the input of a layer, allowing the network to adaptively adjust the distribution of inputs, which can improve the training speed and stability.

2 Learning problem

2.1 Data Origin

The data for this image classification project originates from a collaboration between the Aarhus University Signal Processing group and the University of Southern Denmark [Giselsson et al., 2017]. It has been built to address an agricultural challenge: the ability to effectively differentiate between weed and crop seedlings. This capability has direct implications on optimizing agricultural yields and on more responsible environmental management. This dataset was released as part of a Kaggle competition, aimed at encouraging the exploration of various image recognition techniques and facilitating the exchange of innovative ideas.

2.2 Composition and Species Diversity

The dataset consists of images representing approximately 960 unique plants, classified into 12 species at various growth stages, for a total number of around 5000 images. These species have been carefully selected to cover a range of common crops as well as weeds frequently encountered in agricultural settings from which it is important to discriminate. The included species are : Black-grass, Charlock, Cleavers, Common Chickweed, Common wheat, Fat Hen, Loose Silky-bent, Maize, Scentless Mayweed, Shepherds Purse, Small-flowered Cranesbill, Sugar beet.

This diversity offers a wealth of visual data, from shades of green to the varied shapes and textures of leaves and stems, posing both a challenge in terms of classification and an opportunity for deep learning and in particular CNNs. Those models indeed are very pertinent for image recognition tasks through their ability to represent adequately 2d datas where the spatial component is meaningful as we pointed in Subsection 1.3

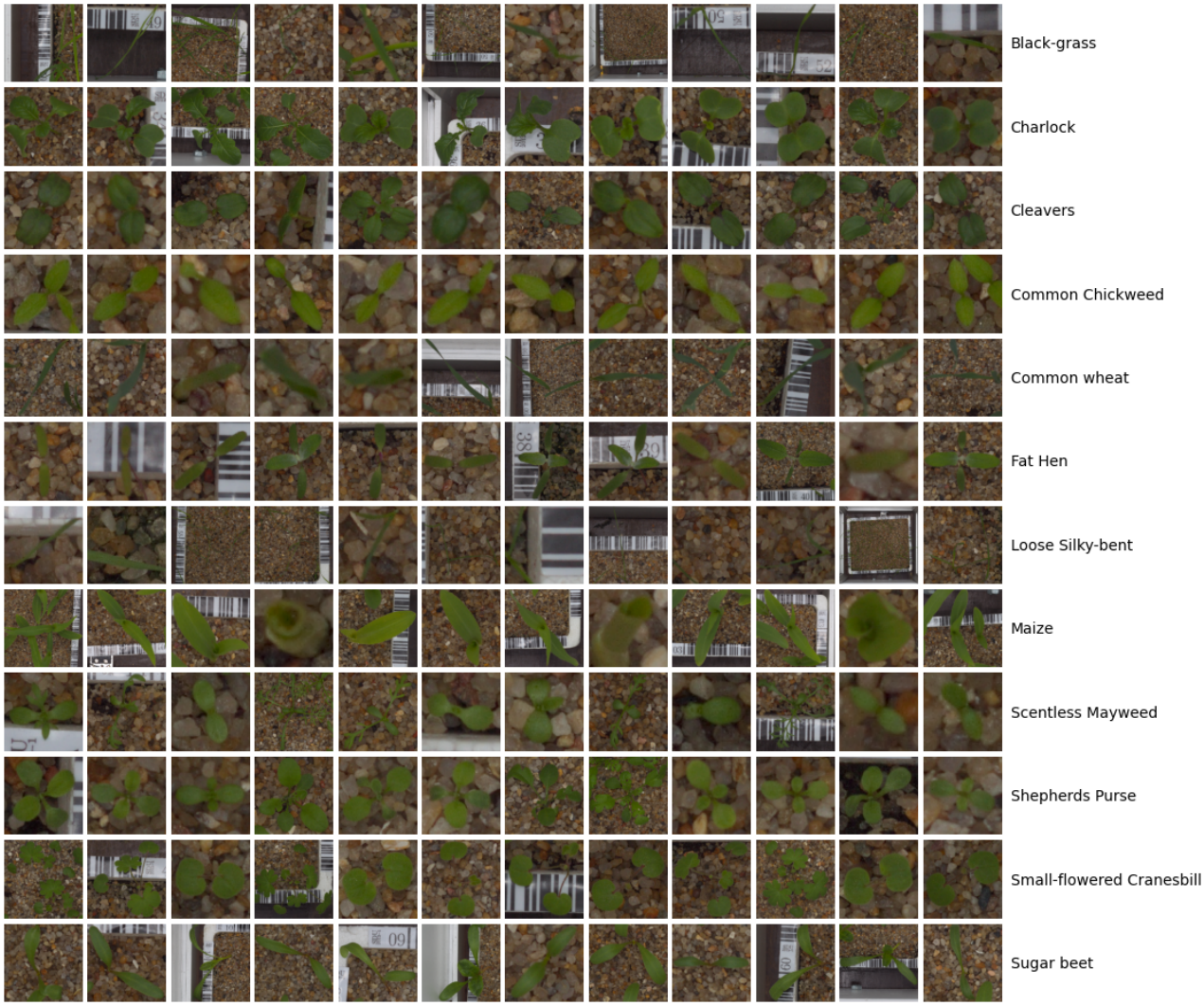


Figure 4: Sample of seedling for each category

This diversity is illustrated if Figure 4, which also demonstrates what makes the classification task so challenging. Indeed, seedling from different categories can appear very similar even between crops and seedlings. This may be an issue for the ability of the model to learn but also for its generalization abilities. Furthermore, the variety of growth stages represented in the images introduces significant variations in terms of plant shape, size, and texture, implies an even harder classification task.

Class distribution

Another important characteristic of the dataset when considering a classification task is class distribution. This distribution is illustrated if Figure 5 and illustrates a mild class imbalance. For instance, the Loose Silky-bent and Common Chickweed species have the highest counts of around 600 samples. This suggests that these species are more frequently present or that more samples are available for these species in the dataset. On the other end of the spectrum, Shepherd's Purse seems to have the lowest count of approximately 200 samples.

The unequal distribution could have implications for classification, where class imbalance might affect the performance of the model. It might be useful to consider resampling or imbalanced learning method if the model shows bad test performance on specific under represented classes. However, the imbalance is not large enough for us to build a model with this issue in mind first hand.

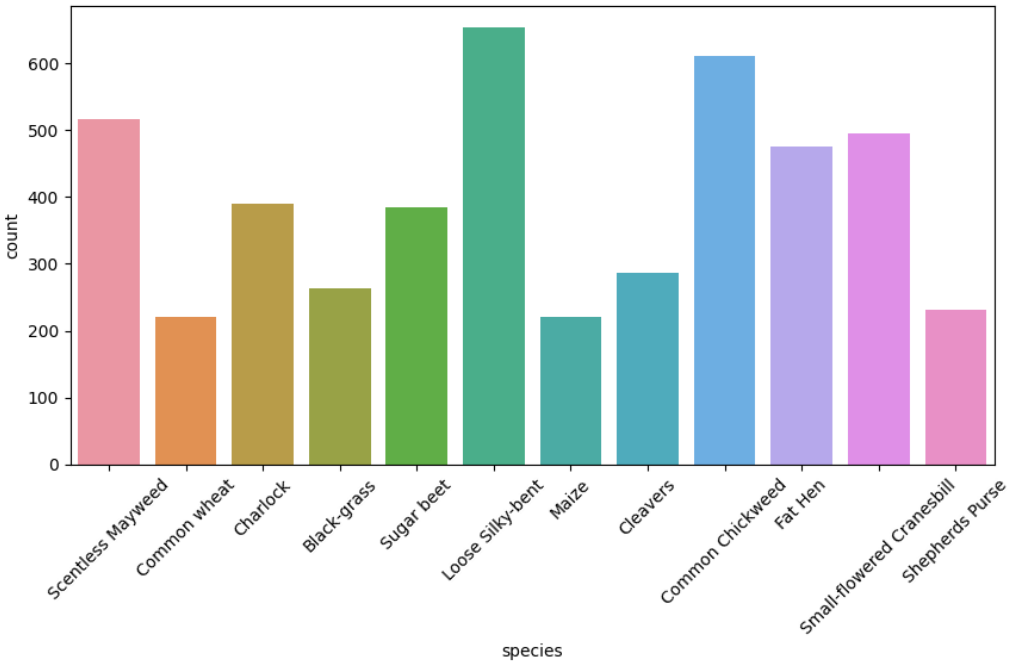


Figure 5: Visualization of Category Distribution in the Training Dataset

3 Models definition and experiments

3.1 Models

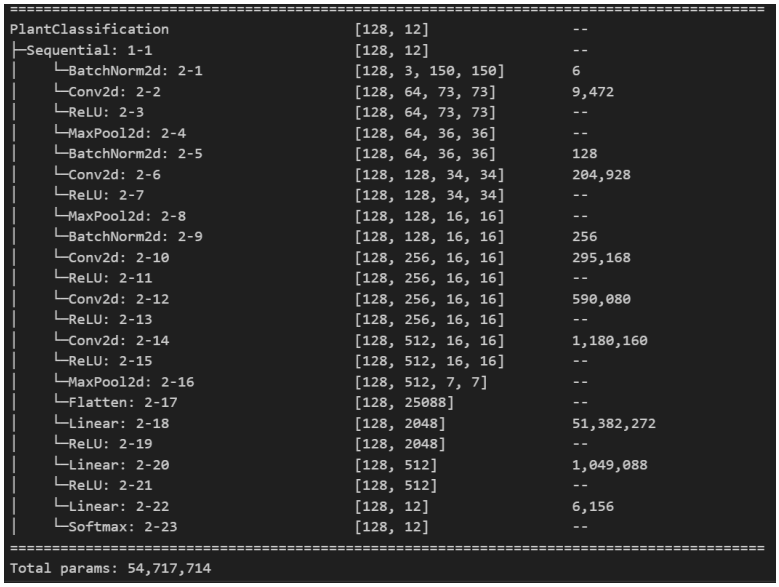
The first defining choice we made regarding our model(s) is to build and train it from scratch instead of fine-tuning an existing one. The motivation for this choice was to be in control of the whole architecture in the way that every modeling decision was our own and motivated by personal reflection. While wanting to develop our personal architecture we also wanted to stay faithful to the historical models that we presented in Subsection 1.4. The main characteristic we wanted for it was 1. to be similar to historical models in its skeleton, 2. to be as simple as possible, i.e. as few layers and parameters as possible and 3. to incorporate simple innovations from Subsection 1.6 and 1.5. Therefore, we decided not to build a residual network because one of its main advantages is to allow the construction of deeper model which would contradict point 2..

Overall, the network skeleton we choose is similar to the one of AlexNet with 3 convolutional blocks each followed by a max-pooling layer and a final classification block composed of dense layers, the two first convolutional blocks consists of a single convolutional layer and the third contains 3 convolutional layer. Of course the kernel size, padding and stride of each layer was purposefully selected, as can be seen in our code, though we will not enter into the details of this parameters here for the sake of conciseness. This skeleton is entirely compatible with the first two characteristics we define before.

Regarding the third characteristic we consider two kind of innovation. First, the addition of new layers, we added BatchNorm layer before each convolutional block to try and improve the learning ability and generalization performance of our network. We also added a dropout layer with $p = 0.2$ at the end of each convolutional block, more precisely we first built a model without those dropout layers that we call our *base model* and another with them that we call the *dropout model* so we can asses the potential advantages of this modification. Note that dropout layers were already used in the AlexNet network but only in the classification block composed of dense layers, we extend from it and incorporate them even at convolutional steps. The last layer we added is a softmax function at the very end of the classification block. The softmax function for C classes is defined as:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}, \quad \text{for} \quad i = 1, 2, \ldots, C$$

where $\mathbf{z} = (z_1, z_2, \ldots, z_C)$ is a vector of raw scores or logits for each class, with the $\text{softmax}(\mathbf{z})_i$

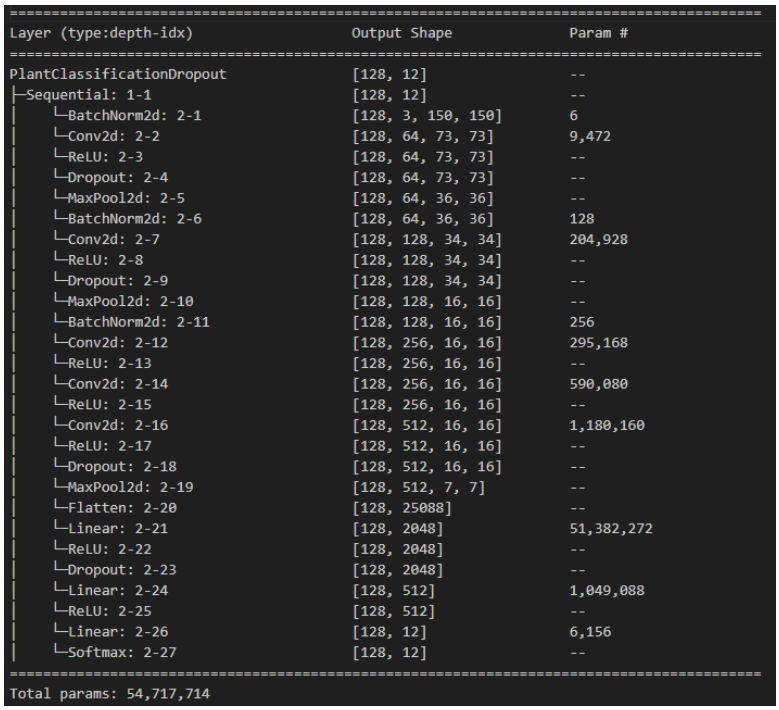


```
==========================================================================================
PlantClassification                      [128, 12]                 --
├─Sequential: 1-1                        [128, 12]                 --
│    └─BatchNorm2d: 2-1                  [128, 3, 150, 150]        6
│    └─Conv2d: 2-2                       [128, 64, 73, 73]         9,472
│    └─ReLU: 2-3                         [128, 64, 73, 73]         --
│    └─MaxPool2d: 2-4                    [128, 64, 36, 36]         --
│    └─BatchNorm2d: 2-5                  [128, 64, 36, 36]         128
│    └─Conv2d: 2-6                       [128, 128, 34, 34]        204,928
│    └─ReLU: 2-7                         [128, 128, 34, 34]        --
│    └─MaxPool2d: 2-8                    [128, 128, 16, 16]        --
│    └─BatchNorm2d: 2-9                  [128, 128, 16, 16]        256
│    └─Conv2d: 2-10                      [128, 256, 16, 16]        295,168
│    └─ReLU: 2-11                        [128, 256, 16, 16]        --
│    └─Conv2d: 2-12                      [128, 256, 16, 16]        590,080
│    └─ReLU: 2-13                        [128, 256, 16, 16]        --
│    └─Conv2d: 2-14                      [128, 512, 16, 16]        1,180,160
│    └─ReLU: 2-15                        [128, 512, 16, 16]        --
│    └─MaxPool2d: 2-16                   [128, 512, 7, 7]          --
│    └─Flatten: 2-17                     [128, 25088]              --
│    └─Linear: 2-18                      [128, 2048]               51,382,272
│    └─ReLU: 2-19                        [128, 2048]               --
│    └─Linear: 2-20                      [128, 512]                1,049,088
│    └─ReLU: 2-21                        [128, 512]                --
│    └─Linear: 2-22                      [128, 12]                 6,156
│    └─Softmax: 2-23                     [128, 12]                 --
==========================================================================================
Total params: 54,717,714
```

Figure 6: Base model architecture

corresponding to the final prediction. Even though it is parameter free this softmax layer is very usual in classification context and we believe it might be useful to allow great performances. The second kind of addition we made regards the optimization procedure itself. While AlexNet used SGD we wanted to improve on this basis by implementing Adam as presented in Subsection 1.5 using the usual parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We believe it may improve the ability of the model to navigate the loss landscape efficiently even though it comes at the cost of a less time efficient optimization compared to SGD.

Both the *base model* and *dropout model* architectures with all characteristics we presented before are illustrated in Figure 6 and 7. The philosophy of our model is therefore to "modernize" the AlexNet skeleton while selecting, kernel sizes, stide, padding and layer width adapted to our learning setup.



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
PlantClassificationDropout               [128, 12]                 --
├─Sequential: 1-1                        [128, 12]                 --
│    └─BatchNorm2d: 2-1                  [128, 3, 150, 150]        6
│    └─Conv2d: 2-2                       [128, 64, 73, 73]         9,472
│    └─ReLU: 2-3                         [128, 64, 73, 73]         --
│    └─Dropout: 2-4                      [128, 64, 73, 73]         --
│    └─MaxPool2d: 2-5                    [128, 64, 36, 36]         --
│    └─BatchNorm2d: 2-6                  [128, 64, 36, 36]         128
│    └─Conv2d: 2-7                       [128, 128, 34, 34]        204,928
│    └─ReLU: 2-8                         [128, 128, 34, 34]        --
│    └─Dropout: 2-9                      [128, 128, 34, 34]        --
│    └─MaxPool2d: 2-10                   [128, 128, 16, 16]        --
│    └─BatchNorm2d: 2-11                 [128, 128, 16, 16]        256
│    └─Conv2d: 2-12                      [128, 256, 16, 16]        295,168
│    └─ReLU: 2-13                        [128, 256, 16, 16]        --
│    └─Conv2d: 2-14                      [128, 256, 16, 16]        590,080
│    └─ReLU: 2-15                        [128, 256, 16, 16]        --
│    └─Conv2d: 2-16                      [128, 512, 16, 16]        1,180,160
│    └─ReLU: 2-17                        [128, 512, 16, 16]        --
│    └─Dropout: 2-18                     [128, 512, 16, 16]        --
│    └─MaxPool2d: 2-19                   [128, 512, 7, 7]          --
│    └─Flatten: 2-20                     [128, 25088]              --
│    └─Linear: 2-21                      [128, 2048]               51,382,272
│    └─ReLU: 2-22                        [128, 2048]               --
│    └─Dropout: 2-23                     [128, 2048]               --
│    └─Linear: 2-24                      [128, 512]                1,049,088
│    └─ReLU: 2-25                        [128, 512]                --
│    └─Linear: 2-26                      [128, 12]                 6,156
│    └─Softmax: 2-27                     [128, 12]                 --
==========================================================================================
Total params: 54,717,714
```

Figure 7: Dropout model architecture, the only difference with the base model is the addition of 4 dropout layer supposed to improve the performance and generalization abilities of the model.

3.2 Experiments

Experimental setup

All model building and training was made using the Pytorch Python library [Paszke et al., 2019]. Using the 4580 labeled data we randomly selected a validation set of 500 samples and took $\frac{1}{5}$ of the remaining images to build our test set, giving a validation set of size 500, a train set of size 3264 and a test set of size 816. The training procedure was made using batches of size 128 and a number of 10 epochs, more might have been better but we fixed 10 epoch to mitigate the time consumption of the learning procedure while still potentially leading to good performances. Finally, the learning rate was set to be $\eta = 0.001$

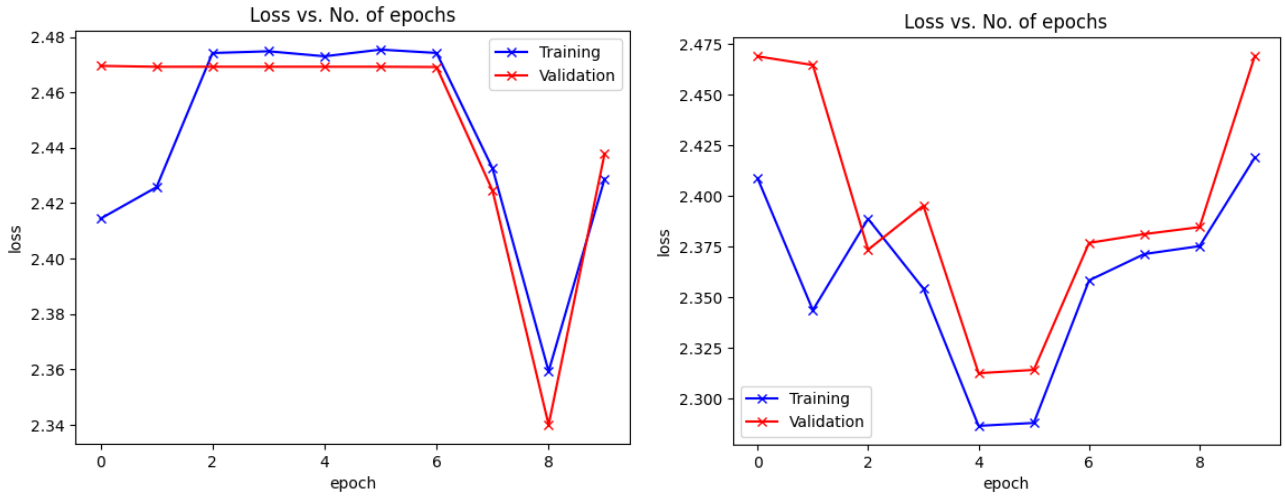
The loss function we chose is the usual cross entropy defined for C classes as,

$$\text{CrossEntropy}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

where $\mathbf{y} = (y_1, y_2, \ldots, y_C)$ is the true probability distribution (one-hot encoded) and $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_C)$ is the predicted probability distribution (output of the softmax function).

Results

Sadly, the training and evaluation of the models were a complete failure. Figure 8 illustrates the train and validation loss through the epochs and Figure 9 illustrates the validation accuracy through the epochs. It appears that both models basically do not learn anything with final loss and accuracy very close to the initial ones. For the 6 first epochs the base model performance almost do not change at all and finally improve after this before worsening again. The dropout model show a more encouraging behavior at first (around the first 4 epochs) but then goes completely downhill and fails too by getting degraded performance at each epoch. Though this might indicate that this version of the architecture was destined to be more performing it is not possible to conclude anything in this regard since its final performance are so bad.

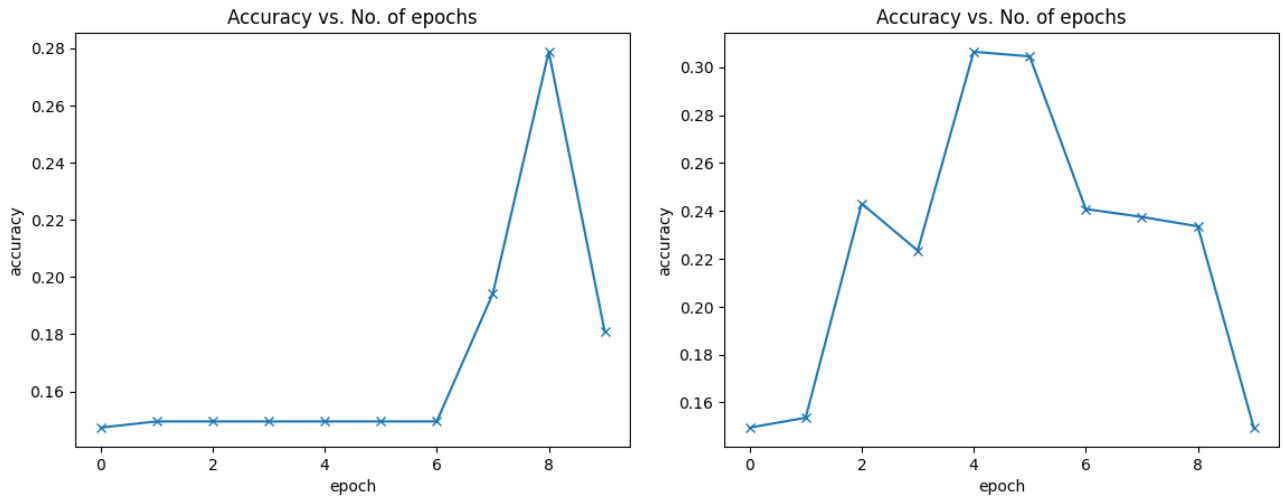


(a) Base model

(b) Dropout model

Figure 8: Evolution of the loss of the models through the epochs.

Without a surprise the same failure appears when considering the test accuracy as shown in Table 1. The base model reach a low 16% accuracy on the test data while the dropout model performs even worse with only 11% accuracy. Combined with the training behavior we presented before this demonstrate how much the model did not learn. Interestingly, what let's us conclude to the absence of learning is that even the training performance did not improve, indicating that the problem is with



(a) Base model

(b) Dropout model

Figure 9: Evolution of the validation accuracy of the models through the epochs.

Data split	Base model	Dropout model
Test	16%	11%
Validation	10%	15%

Table 1: Accuracy of the models on the validation and test data.

the ability of the model to learn a representation of the data and classify not simply a generalization issue.

Even though this failure being caused might be caused by a pure coding error we were not able to spot one. Therefore the explanation for this behavior might be with the models architecturse. It might not be adapted at all for the task at hand. Specifically it is possible that our models are too complex by being too wide and/or too deep. However, in this case we should expect an over fitting issue instead of the mere absence of fitting so this explanation is not satisfying. On the other side, the networks may be too simple for the learning setup, but this too is not a convincing explanation. Indeed, even in the case of a too simple model we should see some kind of learning through the epochs. Furthermore, both networks are quite similar to AlexNet, though a bit less wide, which successfully handled classification one the ImageNet dataset which represents an arguably way harder task than the one we consider here.

4 Conclusion

We constructed two models: a base model and a dropout model, both built with the motivation of making "modern" versions of AlexNet. Despite careful design and implementation, both models failed to learn meaningful representations of the data, as evidenced by their consistently poor performance on both the validation and test sets.

The failure of our models to learn suggests that there may be fundamental issues with the architecture or the training procedure. While it is possible that there are coding errors or implementation issues contributing to this failure, we can only analysis and explain this by considering that the chosen architecture and optimization parameters were not suitable for the task at hand.

One potential avenue for improvement could be to reconsider the complexity of the model architecture. It is possible that our models were either too simple or too complex for the task, leading to difficulties in learning meaningful representations from the data. Experimenting with different architectures, layer sizes, and optimization parameters may help identify a more suitable model for this classification task.

References

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159.

[Giselsson et al., 2017] Giselsson, T. M., Dyrmann, M., Jørgensen, R. N., Jensen, P. K., and Midtiby, H. S. (2017). A Public Image Database for Benchmark of Plant Seedling Classification Algorithms. *arXiv preprint*.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.