Fine-grained Detection of Apple Leaf Diseases with Recurrent Attention CNN

Gabriele Fantini
Politecnico di Torino
gabrielefantini97@gmail.com

Lorenzo Appendini
Politecnico di Torino
lorenzoappendini@gmail.com

Abstract

*This class project explores the implementation of the paper **"Look Closer to See Better: Recurrent Attention Convolutional Neural Network for Fine-grained Image Recognition"** [1] on the Plant Pathology 2021's dataset, a competition from Kaggle platform. The RA-CNN model will crop and up-sample the attended region in a coarse image to allow the model to recognize more features in details. The training strategy is used to iteratively and alternatively optimize weights in the classifier and APN (Attention Proposal Network) with pre-trained EfficientNet B0 CNN features extractor. An EfficientNet B0 has been used for feature extraction instead of a VGG-19, making training less memory greedy and less computationally expensive.*

1. Introduction

Apples are one of the most common fruit crop in the world, so leaf diseases could be a major threat to productivity and quality for large number of people. While manual scouting is time-consuming and expensive, computer vision-based models are surely a preferred answer to the problem, but there are some limitations to be addressed: there are large variations in visual symptoms of the same disease across different cases, and this is mainly caused by the differences in natural and capturing environments, like leaf color and morphology or the age of the infected tissues, non-uniform backgrounds or different light exposure during image capturing.

In order to achieve this goal we decided to implement a machine learning-based model starting from the already implemented network EfficientNet, and composing a more complex network with it, as it will be described shortly, in order to improve his performances and focus more precisely on the images.

The main idea behind the previously mentioned paper is that an image could have some regions whose features are more effective for the network, so the approach is to extract them and do further classification on these finer regions, training and using some dedicated networks called APN (Attention Proposal Networks) which will specialize on defining the latters.

Figure 1. Random batch from the dataset, showing the differences in terms of quality, illumination and background.

2. Data

The dataset used in this project is provided by Plant Pathology 2021-FGVC8 challenge promoters and consists of approximately 18600 RGB images, depicting apple leaves in lots of different scenarios by representing non-homogeneous backgrounds, different maturity stages and different time of day under different camera settings, as in figure1.

One important aspect that needs attention is that this is a **multi-label** classification problem, as provided by a spreadsheet along with the dataset, and this needed to take certain network design decisions in order to be dealt with.

We decided to split the dataset in a **static** way, taking 90% of it for training and the remainder 10% for validation (after shuffling the data): we didn't use k-fold cross validation because despite using this approach we didn't notice major changes in the performances of the network, probably because of the amount of data we have access to and more importantly because even labels with less amount of cases have enough occurrences, as can be seen in figure2; in other words the dataset seems balanced enough (except from the *scab* pathology which is pretty common).

In order to use these images within the EfficientNet chosen implementation we had to pre-process them, resizing them to *224x224* px, since EfficientNet B0 needs this in-

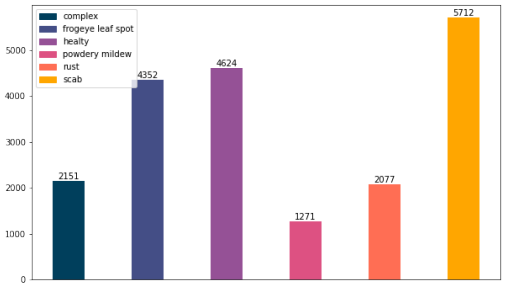

Figure 2. Dataset label distribution.

put format, and normalizing them using *mean* and *standard deviation* of **ImageNet**, since we start with an EfficientNet instance pre-trained with that dataset.

3. Methods

Identifying apple leaf diseases can be led back to a multi-label classification problem. In fact multiple diseases can affect an apple plant and manifest on the same leaf. Usually the symptoms occur in a random and restricted area of the leaf, so different from general recognition, the fine-grained image recognition should be capable of localizing and representing the very marginal visual differences within subordinate categories by gradually zooming the affected region. The chosen data set does not have supervised bounding box/part annotations, so part-based recognition framework must identify possible object regions by analyzing convolutional responses from a neural network in an unsupervised fashion. Yet, two frameworks stand out of the crowd [4], one from the paper "Look Closer to See Better: Recurrent Attention Convolutional Neural Network for Fine-grained Image Recognition" [1] and one from the paper "Learning Multi-Attention Convolutional Neural Network for Fine-Grained Image Recognition" [2]. The Multi-Attention network focus on part-localization, based on the fact that objects have a well defined structure that can be decomposed on multiple parts. Since leaf are quite "uniform" and diseases can manifest in any place, the recurrent-attention network (RA-CNN) is a more suitable solution. It recursively learns discriminative region attention and region-based feature representation at multiple scales in a mutually reinforced way. The learning at each scale consists of a classification sub-network and an attention proposal sub-network (APN). The APN starts from full images, and iteratively generates region attention from coarse to fine by taking previous predictions as a reference, while a finer scale network takes as input an amplified attention region from previous scales in a recurrent way. The paper achieved the state of art results better than most existing models in different fine-grained data set, Stanford Dogs(120 classes), Stanford Cars (196 classes) and CUB-200-2011(200 classes) with a recurrent structure of three scales.

3.1. Model Overview

3.1.1 High Level Changes

In the original paper the RA-CNN uses a VGG-19 for feature extraction. Each network scale must have his own VGG-19, making training unfeasible with limited resources, since a full VGG is made of 144 Million parameters. So we shift to an EfficientNet-B0 which has 5.3 Millions parameters, allowing us to train the entire RA-CNN. EfficientNet networks are based on Inverted Residual Block, also called MBConv Block, a type of residual block that used an inverted structure for efficiency reason. In the original paper the feature extracted are passed to a fully-connected layer for classification. Instead we implement the classifier as shown in figure 4 by keeping the remaining part of the EfficientNet, thus avoiding loosing the precious contribution of skip-connections.

3.1.2 Model Structure

The overall network can be divided in three subparts. The first two parts take as input an image, extract its features using EfficientNet convolutional layers and than feed them into an Attention Proposal Network and into the classifier. The last part instead just take an image and classifies it using EfficientNet. The Attention Proposal Networks (APN) predict a set of box coordinates of an attended region for the next finer scale, similar to a bounding box proposal but in an unsupervised fashion. The related formulae are:

$$p(\mathbf{X}) = f(\mathbf{W}_c * \mathbf{X}) \tag{1}$$

$$[t_x, t_y, t_l] = g(\mathbf{W}_c * \mathbf{X}) \tag{2}$$

where $\mathbf{X}$ is the input image, $\mathbf{W_c}$ denotes the convolution layer operations, f is the classifier, and g is the APN. More specifically, the input of the APN will be the flattened feature output of EfficientNet network on stage 8 as shown in Figure 5, which is of size 320*7*7. Then it will run through a linear layer, a sigmoid layer, another linear layer and a tanh layer, so that the final output will be three values including the 2D position of the center location of the next region as well as the diameter of the new region. The three values are then fed into the AttentionCropResize layer, which takes them together with an image and returns a cropped image. The AttentionCropResize layer's forward and backward paths require a customized implementation, writing a class that inherits from `torch.autograd.Function`.

The forward paths consist of two parts, the first one handles the cropping and the second one applies an up-sample

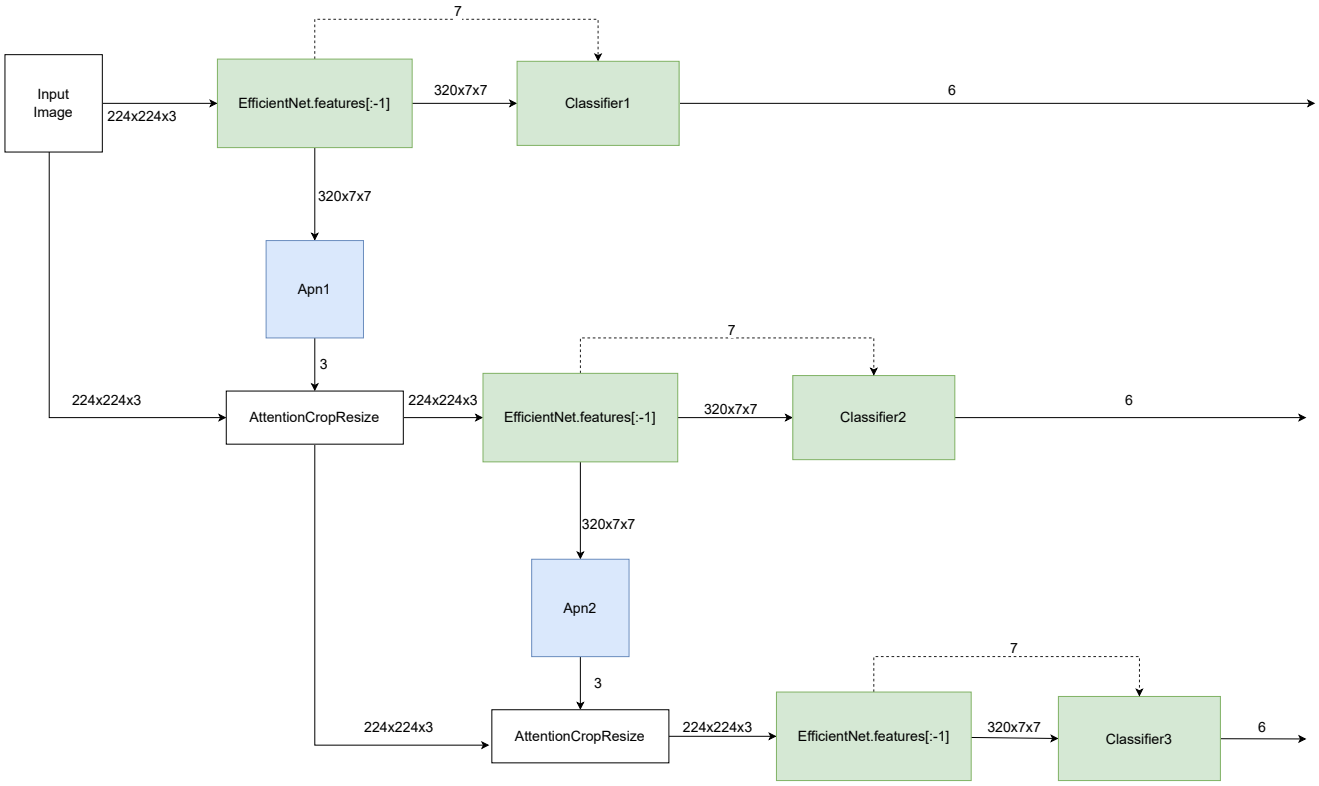

Figure 3. Overall implemented network architecture.

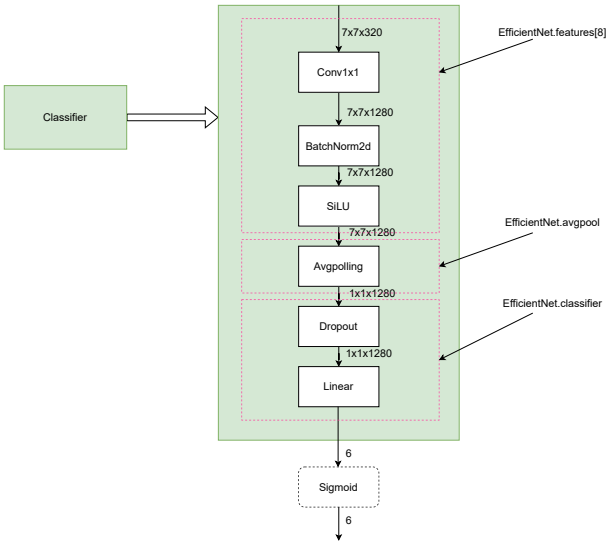

Figure 4. Classifier's structure used in our custom RA-CNN implementation.

layer provided by `PyTorch`. In the first cropping process, the range of t_x and t_y from the APN is bounded in the range $[-1, 1]$, since the last layer of the latter is the $tanh$ activation function. Starting from this range, we want to have both x, y coordinates between 56 and 168 (the center region of the image), so we multiply it by 56 and then add 112 to it; similarly, we set the range of t_l from 38 to 56, making sure the cropped image can't go out of bounds.

The crop procedure is actually implemented by and element-wise multiplication between the original image received as input and an attention mask, which can be computed from t_x, t_y, t_l generated by a logistic function, as stated by [2]. Specifically, let $\mathbf{X}$ denote the image, $\mathbf{M}$ the attention mask and h the logistic function, the first step is to generate an attention region, which has the formula:

$$\mathbf{X}^{att} = \mathbf{X} \cdot \mathbf{M}(t_x, t_y, t_l) \tag{3}$$

where M is determined by t_x, t_y, t_l as:

$$\begin{aligned} tx(tl) &= t_x - tl, \quad ty(tl) = t_y - tl \\ tx(tl) &= t_x + tl, \quad ty(tl) = t_y + tl \end{aligned} \tag{4}$$

$$\begin{aligned} \mathbf{M}(\cdot) = &[h(x - tx(tl)) - h(x - tx(br))] \\ &\cdot [h(y - ty(tl)) - h(y - ty(br))] \end{aligned} \tag{5}$$

The generated image is then up-sampled again to 224x224 using `nn.Functional.upsample`, to feed into the next unit.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 5. EfficientNet B0 structure.

The gradient of the attention crop layer is completely unknown to the auto gradient mechanism, so we need to specify the entire backward path; specifically said, it's necessary to define the gradient for both the crop and the bilinear upsample operation. From [2], the gradient from the ranking loss is proportional to the element-wise multiplication of the gradient output from the second CNN layer and the gradient of the APN, which can be indicated as:

$$\frac{\partial L_{rank}}{\partial t_x} \propto \mathbf{D}_{top} \odot \frac{\partial \mathbf{M}(t_x, t_y, t_l)}{\partial t_x} \tag{6}$$

Besides, the paper [1] specifies also that the gradient of $\mathbf{M}$, or the APN, should follow these qualitative results:

$$\mathbf{M}'(t_x) = \begin{cases} < 0 & x \rightarrow t_{x_{tl}} \\ > 0 & x \rightarrow t_{x_{br}} \\ = 0 & otherwise \end{cases} \tag{7}$$

$$\mathbf{M}'(t_y) = \begin{cases} < 0 & x \rightarrow t_{y_{tl}} \\ > 0 & x \rightarrow t_{y_{br}} \\ = 0 & otherwise \end{cases} \tag{8}$$

$$\mathbf{M}'(t_l) = \begin{cases} > 0 & \{x, y\} \rightarrow t_{x,y_{tl}} or\ t_{x,y_{br}} \\ < 0 & otherwise \end{cases} \tag{9}$$

In the above expressions, "$\rightarrow$" means "tends to". Since the images have size 224x224, we set a metric of 56 pixels to evaluate this condition: a pixel "tends to" (comes near) the specific border if it is within 56 pixels from that.

To compute the result, we first calculate the negative norm of the gradient output, displayed as an "energy map" of the cropped image. Then we incorporate equations [7]-[9] by applying a pre-calculated mask with values 0,1 or -1 to the norm. Finally, since t_x, t_y and t_l are scalars, the masked values for each pixel is summed up to propagate the gradient into the APN.

3.2. Loss

The RA-CNN is optimized by two types of supervision: intra-scale classification loss and inter-scale pairwise margin ranking loss, for alternatively generating accurate attention and learning more fine grained features. The total loss of the model is defined as:

$$L(X) = \sum_{s=1}^{3} L_{cls}(Y^{(s)}, Y^*) + \sum_{s=1}^{2} L_{rank}(p_t^{(s)}, p_t^{(s+1)}) \tag{10}$$

where s denotes each scale, $Y^{(s)}$ and Y^* denotes the predicted label vector from a specific scale and the ground truth label vector, respectively. L_{cls} represents classification loss, which optimizes the parameters of convolution and classification layers, ensuring adequate discrimination ability at each scale. Because our task is a multi-label classification problem, instead of passing the output of classification layers to a Softmax function and than use Cross Entropy Loss, we use `torch.nn.BCEWithLogitsLoss`, a Pytorch class that combines a Sigmoid layer and a Binary Cross Entropy Loss. Given that $p_t^{(s)}$ from L_{rank} denotes the prediction probability on the correct categories label t, the margin ranking loss is:

$$L_{rank}(p_t^{(s)}, p_t^{(s+1)}) = max\{0, p_t^{(s)} - p_t^{(s+1)} + margin\} \tag{11}$$

and it has been implemented using Pytorch class `torch.nn.MarginRankingLoss`, with `reduction` parameter set to `"sum"` and `margin` set to 0.05 as is in the original paper. The L_{rank} loss enforces

$$p_t^{(s+1)} > p_t^{(s)} + margin$$

which enable network to take the prediction from coarse scale as references and gradually approach the most discriminative region by enforcing the finer-scale network to generate more confident predictions.

3.3. Problems Encountered

We started from a non working implementation of RA-CNN found on GitHub [3]. The crop function didn't work, the APN network loss didn't converge at all and the network performances were very bad, but it was a good starting point since it has a lot of pre-built utilities, such as making GIFs. Addressing the problems and correcting them has been a difficult task due to the articulate network structure.

3.3.1 Attention Crop Function Implementation

Since the starting network had very big problem with the Attention Proposal Network, we started from rethinking both the structure of it, including the attention crop function an it's backward function. The original implementation for the APN layers uses two `nn.Sequential` blocks, made of Linear layer, Tanh activation function, Linear layer and Sigmoid activation function at the end. This gives, starting from the input features, three outputs in the $(0, 1)$ range. Then t_x, t_y, t_l were multiplied by respectively by 1, 1, 0.5 and than by 224, so their range became $(0, 224)$ for t_x, t_y

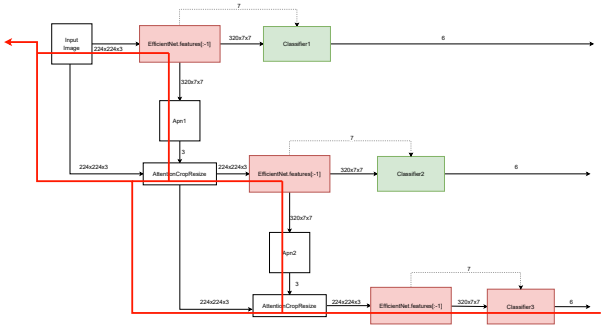

Figure 6. Backward propagation of scale 3 BCE loss.

and $(0, 112)$ for t_l. The next steps that follow in the attention crop function didn't put an upper limit on t_l so in the early part of training the network was encouraged to maximize it, since the best predictions initially come from the first scale, leading to a network that predicts from the same regions of the first scale. So we decided to change the Sigmoid with a Tanh and redesign the logic of attention crop function as already explained in the 3.1.2, simplifying it and putting an upper limit of 56 px on t_l.

3.3.2 Network Initialization

Starting from a pretrained EfficientNet B0 on the ImageNet dataset, we fine-tuned it on our data set for better feature extraction, early stopping the training before the network starts to overfit on train data. We saw that this solution improves the APN initialization and speed up the training but does not affect much the ending results of it. For APN initialization, we take the feature output of scale 1 and 2, we upscale the 7x7 to 224x224, taking the max from all the 320 channels. Then we take the indices of the maximum value of the 224x224 matrix and we re-conduct them in the $(-1, 1)$ range. We pretrain the APN with `torch.nn.smooth_l1_loss` from the obtained values and the APN's output.

3.3.3 Inter Scale Classification and Pairwise Ranking Losses

After fixing the APN, the attention crop function and the network initialization, we saw that a strange phenomenon occurred with the different scales validation accuracies: the last two scale kept getting better and better prediction accuracies, while the first one was tending to 0. Also the accuracies of the last two scale were very close and by passing images to the network we see that also the attention region were quite similar. Further analyzing the backward path of the losses we found that our our approach was incorrect: we were using a single loss and optimizer for all of the scales output and a single loss and optimizer for both the APNs. This approach led to the update of parameters present in the other scale's convolutional layers (as shown in Figure 6) and to incorrect update of APN parameters. So, starting from scale 3 and going down to 1, we split the BCE loss in 3 losses and the same for the SGD optimizer, for each scale we calculate the gradient with the `.backward()` method invoked on the respective loss and then we update only the parameters of the scale's convolutional and classification layers. In a similar fashion we modified the APNs's losses and optimizers.

3.4. Performance Evaluation

4. Experiments

Lots of experiments have been done before achieving a working network. After that, we run some tests to fine tune the network. Since each training lasted at least 12h, the fine tuning phase has been limited by this technical factor. First we tried two different activation function in between the two fully connected layer of APNs. The Sigmoid's losses function seemed to converge faster than the ReLU's ones as can be seen from Figure 11 and Figure 12. Instead the two activation functions perform differently at each scale, with ReLU performing better at scale 2 and Sigmoid at scale 3 (Figure 7, Figure 8). Based on this fact we decided to adopt the Sigmoid. The last experiment was about the zoom boundaries of the AttentionCropFunction. This hyperparameter hugely affect the performance of the RACNN. In particular, since the leaf's diseases can affect both a small or a large area, we set the min zoom radius to 90px and the max zoom radius to 148px. This improves the accuracies of each scale by a lot (Figure 9) and leads the losses to converge faster (Figure 11, Figure 12). For each test the APNs's optimizers has been set with learning rate of 1e-6 and the Cls's optimizers with learning rate of 0.001 and momentum of 0.9.

4.0.1 Dimension of Zoom's Radius

Experimentally we saw that the boundaries of zoom's radius highly affect the performance at scale 2 and scale 3. As the paper suggests we initially set the lower boundary to $1/3$ of the image size. We saw that without an upper boundary the network didn't zoom at all and, during training, scale 2 and 3 flattened to scale 1, so we set it to $1/2$ of image size. We tried this configuration in Experiment 1 and 2, then we enlarged the zoom giving it $(2/5, 2/3)$ boundaries and we saw a further improvement in performances, corroborating the fact that zoom's radius is one of the most important parameter to fine tune for network's performance.

4.1. Fusion Layer

As in the original paper, to leverage the benefit of feature ensemble, we concatenate the feature descriptor, generated from the classification layers in the RA-CNN, into a

fully-connected fusion layer with sigmoid function for the final prediction. In particular we concatenate scale 1 and 2 and scale 1, 2 and 3. We set all the network's parameters to `require_grad=false` except for the fully connected fusion layers, then we set the RACNN to `eval()` mode in order to disable random behaviour such as dropout and we trained both fusion layers for 5 epoch, with batch size of 32 and learning rate of 0,001 with momentum at 0,9. We achieved on the validation set an accuracy of $0,8374$ for scale 1-2 and an accuracy of 0.8423 for scale 1-2-3. This validates the fact that the superior result benefits from the complementary advantages from multiple scales.

EfficientNetB0				
	Precision	Recall	F1-score	Support
complex	0.70	0.66	0.68	219
frogeye leaf spot	0.88	0.84	0.86	445
healthy	0.97	0.98	0.98	481
powdery mildew	0.96	0.90	0.93	120
rust	0.92	0.93	0.92	215
scab	0.92	0.85	0.88	569
micro avg	0.90	0.87	0.89	2049
macro avg	0.89	0.86	0.88	2049
weighted avg	0.90	0.87	0.89	2049
samples avg	0.90	0.90	0.90	2049

Table 1. EfficientNetB0 F1-scores

Scale 1				
	Precision	Recall	F1-score	Support
complex	0.71	0.67	0.69	219
frogeye leaf spot	0.89	0.84	0.87	445
healthy	0.98	0.97	0.97	481
powdery mildew	0.90	0.94	0.92	120
rust	0.93	0.92	0.92	215
scab	0.90	0.89	0.89	569
micro avg	0.90	0.88	0.89	2049
macro avg	0.88	0.87	0.88	2049
weighted avg	0.90	0.88	0.89	2049
samples avg	0.91	0.91	0.90	2049

Table 2. RACNN scale1 F1-scores

Scale 2				
	Precision	Recall	F1-score	Support
complex	0.68	0.66	0.67	219
frogeye leaf spot	0.87	0.82	0.84	445
healthy	0.95	0.97	0.96	481
powdery mildew	0.92	0.94	0.93	120
rust	0.93	0.87	0.89	215
scab	0.92	0.85	0.88	569
micro avg	0.89	0.86	0.87	2049
macro avg	0.88	0.85	0.86	2049
weighted avg	0.89	0.86	0.87	2049
samples avg	0.89	0.89	0.88	2049

Table 3. RACNN scale2 F1-scores

Scale 3				
	Precision	Recall	F1-score	Support
complex	0.67	0.55	0.60	219
frogeye leaf spot	0.80	0.64	0.71	445
healthy	0.88	0.93	0.90	481
powdery mildew	0.94	0.88	0.91	120
rust	0.90	0.79	0.84	215
scab	0.84	0.85	0.84	569
micro avg	0.84	0.78	0.81	2049
macro avg	0.84	0.77	0.80	2049
weighted avg	0.84	0.78	0.81	2049
samples avg	0.82	0.82	0.81	2049

Table 4. RACNN scale3 F1-scores

Scale 1+2				
	Precision	Recall	F1-score	Support
complex	0.73	0.67	0.70	219
frogeye leaf spot	0.89	0.86	0.88	445
healthy	0.97	0.98	0.97	481
powdery mildew	0.96	0.92	0.94	120
rust	0.94	0.92	0.93	215
scab	0.93	0.88	0.91	569
micro avg	0.91	0.88	0.90	2049
macro avg	0.90	0.87	0.89	2049
weighted avg	0.91	0.88	0.90	2049
samples avg	0.91	0.91	0.91	2049

Table 5. RACNN scale1+2 (fusion layer) F1-scores

5. Conclusion

The RACNN network obtains similar results at scale 1 [tab2] as in EfficientNetB0 [tab1]. This is because they are essentially the same networks and have the same inputs. Both performed well in each label except in complex one with an f1-score of 0.69 and 0.68 respectively. The performances at scale 2 [tab3] and then at scale 3 [tab4] degrade progressively. The prediction taken combining scale 1-2 [tab5] and 1-2-3 [tab6] reach the highest f1-score over all the solutions, with the scale 1-2-3 that performs slightly better on the complex label. This is in accordance with the results of the original paper. Finally, we think it is possible to further improve the performances of the RACNN with better hyperparameter fine-tuning. The added complexity of the network and the difficulties encountered to fine-tune it led us to think that, with the same memory and computational consumption of a RACNN, it is possible to obtain better results with an efficienNetB1 or higher, that are much

Scale 1+2+3				
	Precision	Recall	F1-score	Support
complex	0.76	0.68	0.72	219
frogeye leaf spot	0.89	0.87	0.88	445
healthy	0.97	0.98	0.97	481
powdery mildew	0.95	0.92	0.93	120
rust	0.93	0.92	0.92	215
scab	0.91	0.89	0.90	569
micro avg	0.91	0.89	0.90	2049
macro avg	0.90	0.87	0.89	2049
weighted avg	0.91	0.89	0.90	2049
samples avg	0.91	0.91	0.91	2049

Table 6. RACNN scale1+2+3 (fusion layer) F1-scores

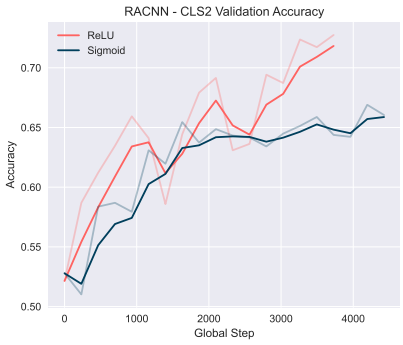

Figure 7. comparison between classifier2 validation accuracies of RACNN using ReLU and Sigmoid

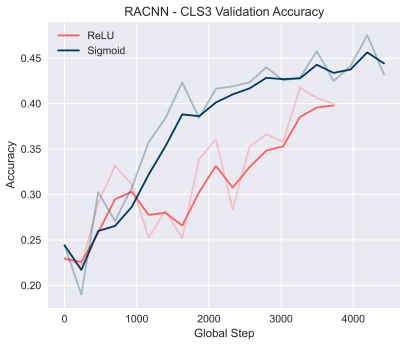

Figure 8. comparison between classifier3 validation accuracies of RACNN using ReLU and Sigmoid

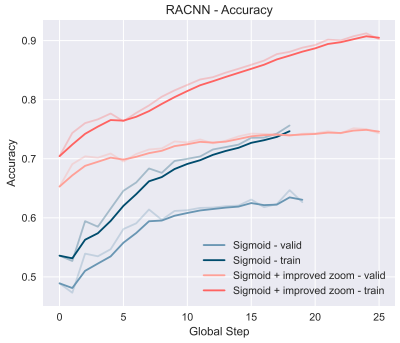

Figure 9. comparison between both training and validation accuracies of RACNN using Sigmoid and Sigmoid with improved zoom

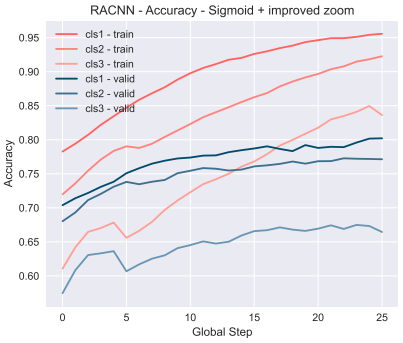

Figure 10. comparison between classifiers training and validation accuracies of RACNN using Sigmoid with improved zoom

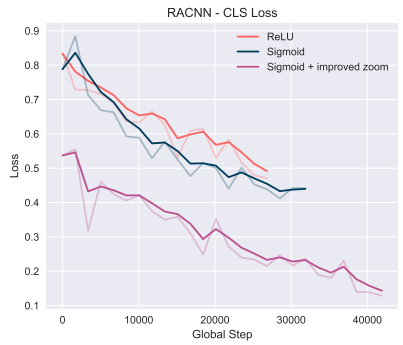

Figure 11. comparison between CLS losses of RACNN using ReLU, Sigmoid and Sigmoid with improved zoom

simpler to fine-tune.

References

[1] Jianlong Fu, Heliang Zheng, and Tao Mei. Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition. *CVPR*, 2017. 1, 2, 4

[2] Jianlong Fu, Heliang Zheng, Tao Mei, and Jiebo Luo. Learning multi-attention convolutional neural network for fine-grained image recognition. *ICCV*, 2017. 2

[3] klrc. Racnn-pytorch. https://github.com/klrc/RACNN-pytorch, 2019. 4

[4] Xiu-Shen Wei. Awesome fine-grained image analysis – papers, codes and datasets. 2

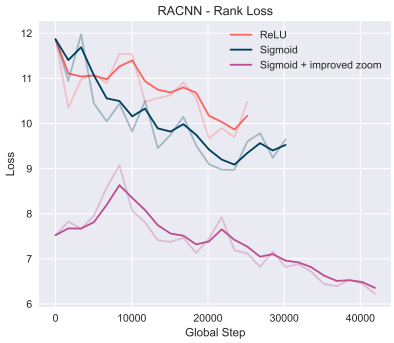

Figure 12. comparison between rank losses of RACNN using ReLU, Sigmoid and Sigmoid with improved zoom

Figure 13. Two samples from dataset and its zoomed results (with both scales).