Technische Universität Wien

184.702 Machine Learning

Exercise 3
Topic 3.4: Generation and evaluation of unstructured synthetic datasets

Group 12

Students:
Majlinda Llugiqi 11931216
Lukas Lehner 01126793
Mal Kurteshi 11924480

Table of contents

1. The Dataset

We chose the fruit image dataset that was provided (FIDS30). It is a multiclass classification dataset. It consists of 971 images of common fruits. Images are classified into 30 different fruit classes. Every fruit class contains about 32 different images. The images are in JPG format, from a few kB to a few MB in size.

The dataset is quite diverse, meaning it contains images from different perspectives, some contain just a single fruit and others contain dozens, moreover some of the images contain a lot of noise (such as leaves, plates, trees and other backgrounds).

Before training we preprocess the data by scaling the shorter edge to 64 pixel in length and then applying a center crop to receive a 64 by 64 rgb image, which is then normalized with mean and std of 0.5. Figure 1.1 shows example images after preprocessing.

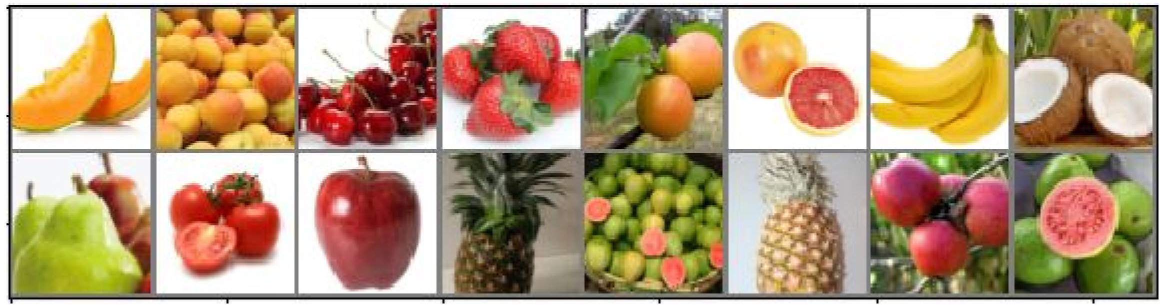


Figure 1.1: Example images from the FIDS30 dataset after preprocessing.

2. The Setup

For this project we have used Python 3.7 with the following libraries with respective versions:

- Torchvision 0.5.0
- Torch 1.4.0
- Conda 4.7.12
- Image 1.5.28
- Pillow 7.0.0
- Cuda 10.2.89

The full list of all the python libraries used you can find in the requirements.txt.
Moreover experiments were performed on the following machines:

- Lenovo flex 5 (i7, 16GB RAM, 512SSD DDR3, NVIDIA 2GB GPU)
- Desktop PC (Intel 1151 i7-97005, NVIDIA GeForce GTX 1070 Ti 8GB, 16 GB DDR4)

3. The Discriminator

The Discriminator is a neural network used for evaluation of the data instances for their authenticity. The Discriminator decides whether each instance of data that it reviews belongs to the actual dataset or not. To look at it from another perspective the Discriminator is a classifier for the input data instances.

The Discriminator's training data comes from two sources:

- **Real data** instances, such as real pictures of people. The Discriminator uses these instances as positive examples during training.
- **Fake data** instances created by the generator. The Discriminator uses these instances as negative examples during training.

We have trained a discriminator, but before going into details with our trained discriminator lets first analyse some points of training a discriminator.

When you train the discriminator, hold the generator values constant; and when you train the generator, hold the discriminator constant. Each should train against a static adversary.

In our case we have trained the discriminator which is able to differentiate the data instances from fake and real ones from the dataset. One of the main and strongest challenges in order to build a strong discriminator was the part of gaining as much accuracy as it would be possible, the higher the value of the discriminator accuracy the better the discriminator. In the discriminator some of the parameters that were adjusted were: depth of the model (18/34), batch size, used loss function and the number of epochs.

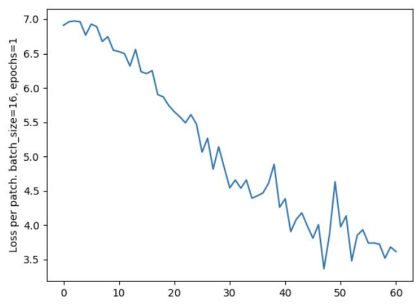
A residual network (ResNet) differs from other architectures in Deep Learning in the fact that it is able to jump over layers, as well as return to previously visited layers while training. This helps tackle the issue of vanishing gradient, where it has been shown that adding too many layers in a network might actually deter the performance. Since ResNets, uses the same layers over and over it is not required to add new layers in order to try and improve performance. For the discriminator we have mostly used the resnet34 model provided by pytorch library in python. Also some tests were made with the resnet18 but due to hardware limitations we did not use deeper models for resnet, the most efficient one was the resnet34.

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. A perfect model would have a log loss of 0.

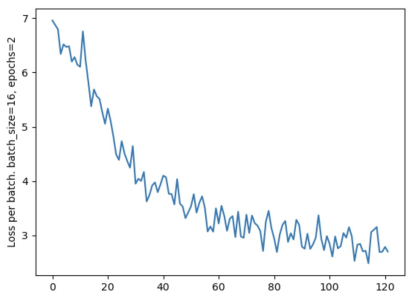
The batch size defines the number of samples to work through before updating the internal model parameters. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model.

The number of epochs defines the number times that the learning algorithm will work through the entire training dataset.

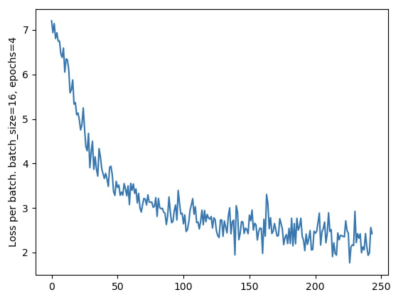
As we can see from the figures below, it is plotted the loss function for different batch size parameters and epochs parameters. As expected, the results started to improve with the raise of the number of epochs, also batch size played an important role, different batch sizes were tested and the best results were gathered. Best results were gathered using a batch size of 16. The accuracy of the discriminator with the specified parameters was able to reach 90% of accuracy.



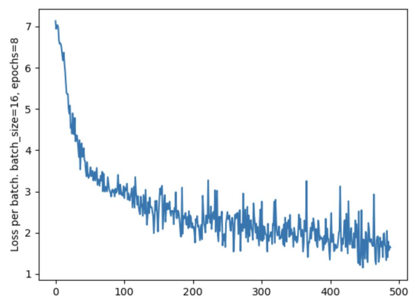
Resnet18, 1 epochs, batch size = 16



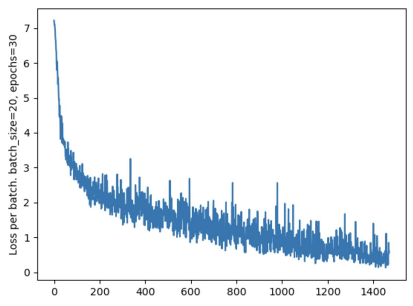
Resnet18, 2 epochs, batch size = 16



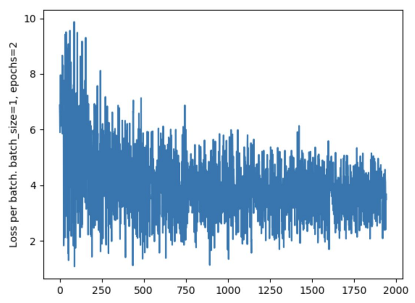
Resnet 32, 4 epochs, batch size = 16



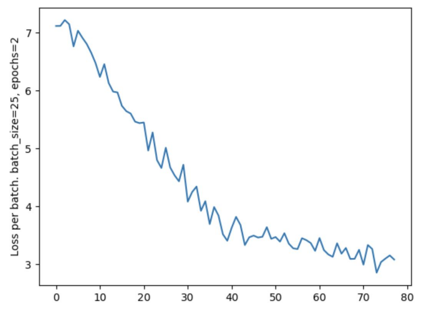
Resnet 32, 8 epochs, batch size = 16



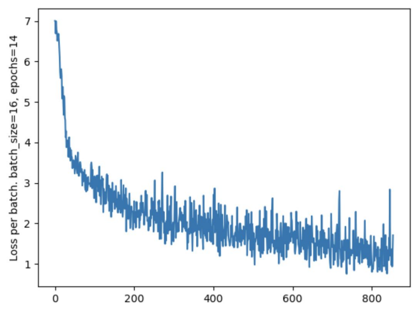
Resnet 32, 30 epochs, batch size = 20



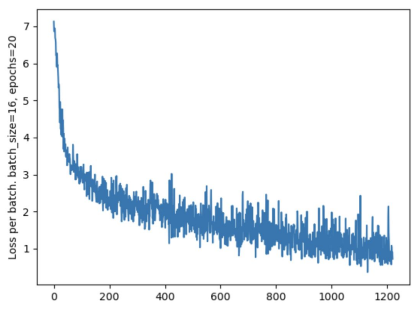
Resnet 32, 2 epochs, batch size = 1



Resnet 32, 2 epochs, batch size = 25



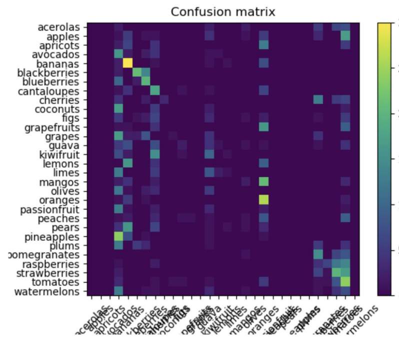
Resnet 32, 14 epochs, batch size = 16



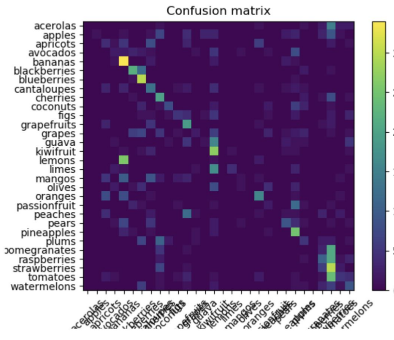
Resnet 32, 20 epochs, batch size = 16

Figure 3.1. Loss function for different number of batched and epochs

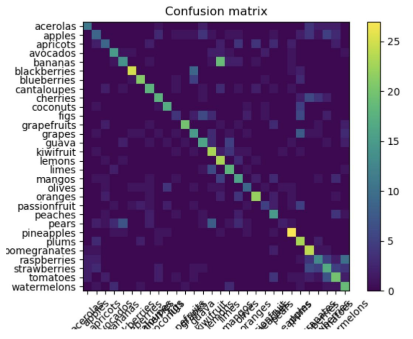
During the analyzation and parameter testings for the accuracy of the discriminator the confusion matrix was taken also in the consideration, in the figure below it is shown the progress in the confusion matrix starting from the testing parameters which in the beginning where bad as like the number of batches 1 and number of epochs 1 until 5, by varying with parameters its was obtained that the confusion matrix started to get more regular and clear, the last matrix is shown for the best accuracy taken from the testings, which contained accuracy up to 90%, this was also gained with number of batches 16 and number of epochs 30.



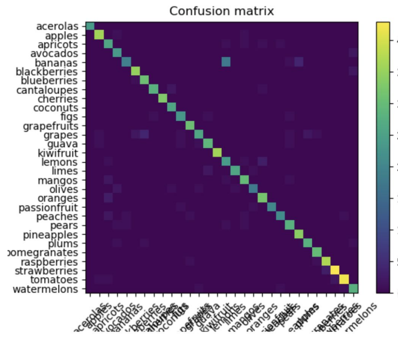
Resnet 32, 4 epochs, batch size = 16



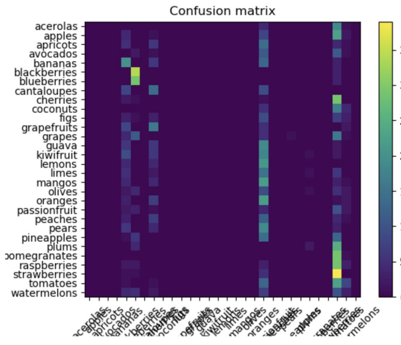
Resnet 32, 2 epochs, batch size = 16



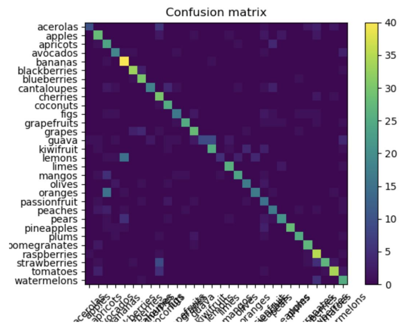
Resnet 32, 14 epochs, batch size = 16



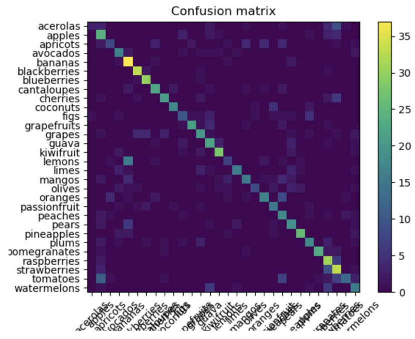
Resnet 32, 30 epochs, batch size = 20



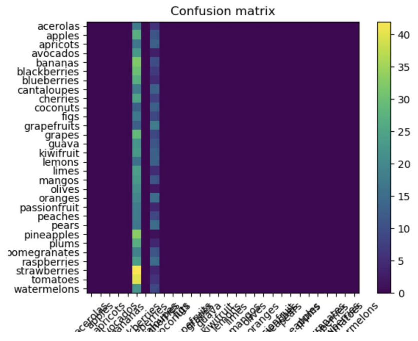
Resnet 32, 2 epochs, batch size = 25



Resnet 32, 20 epochs, batch size = 16



Resnet 32, 8 epochs, batch size = 16



Resnet 32, 2 epochs, batch size = 1

Figure 3.2. Confusion matrix for different numbers of batches and epochs.

4. The Networks

After realizing that a classical generative adversarial network trains both the discriminator and the generator at the same time, we restructured our code to reflect that fact. For the discriminator we mostly stuck with what was tried before:
D34: Residual neural net based on resnet34. The last layer in all resnets is a linear transformation to the number of output features, which we changed to the number of classes in our dataset.
We tried out a few generator setups (more context provided later):
G1: A basic generator neural network. It consists of five ‘deconvolution’ layers (nn.ConvTranspose2d) with batch normalization (nn.BatchNorm2d) and application of the rectified linear unit function (nn.ReLU) in between. Given a batch of 100x1x1 noise, G1 outputs a batch of 3-channel 64x64 images.
G2: Builds on G1. Five convolutional layers are inserted before the deconvolution layers of G1. G2 takes a 3x8x64 input, applies convolutions until the data has size 100x1x1 and then continues like G1.
G2a: Operates similarly to G2 but has only three convolutional layers with different kernel sizes and strides. Input size is 3x10x64.
Unless otherwise specified, these parameters were used during training for both D, and G:
Optimizer: Stochastic gradient descent (optim.SDG) with momentum 0.95
Loss function: nn.CrossEntropyLoss
Learning rate: 0.001

5. Training the GAN

We started out training a GAN using resnet34 and G1. After getting everything to work properly we trained the model for 600 epochs (roughly 5h30m runtime). Figure 5.1 shows generated images after 600 epochs.

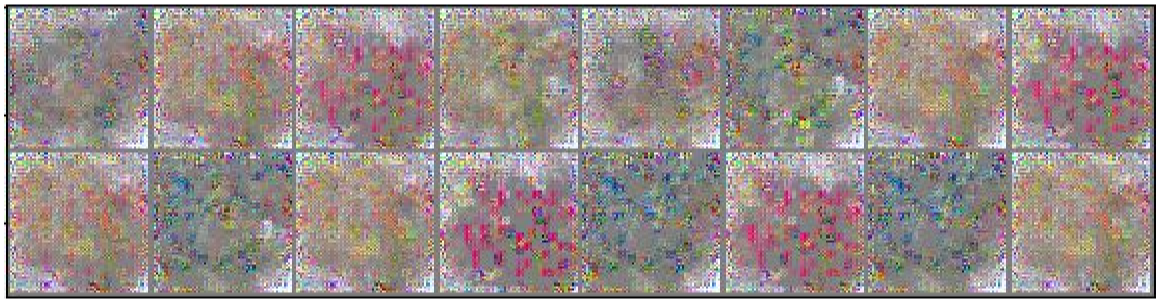
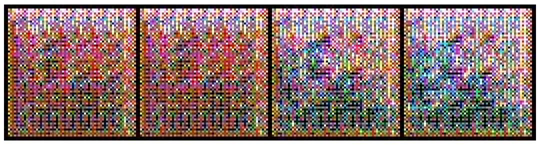
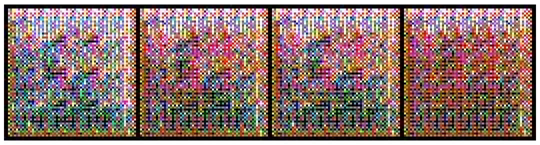


Figure 5.1: Images generated by a generator after 600 epochs of training on the entire dataset.

As you can see, the network learned to somewhat synthesize what an image from our dataset looks like, as most images have a fruit arrangement in the center with white surroundings. However, as the main goal of this project is to create a synthetic dataset on which a new model

can be trained to classify all types of fruit from the original set, we realized the current setup will likely never be able to achieve what it's meant to. Even with enough training and perfect parameter tuning, in a classical GAN approach the generator will probably never learn to generate all ~30 classes, but rather only a few of them who let him reliably fool the discriminator. Additionally, when generating a synthetic dataset we have no control over which classes the generator will output, as its input is completely random.
One idea to easily (i.e. without changing the architecture) amend this issue was to **'spike'** the random input: Instead of a random 100x1x1 input G1 would receive a random 100x1x1 input were the k-th 'pixel' is set to 1, indicating the k-th class is to be trained/generated. But clearly one entry being set to 1 by chance (or close to it) in an array of 100 random entries (generated using normal distribution (mean=0, var=1)) is too high. The first idea to solve this was to simply increase the spike value from 1 to a sufficiently high value (e.g. 10). Results can be seen in Figure 5.2. The second idea was to change the architecture, which led to the creation of G2 and G2a. Spiked noise in G2 is a 3x8x64 input in which the k-th 'column' is set to an 8x1 array of ones for all 3 channels (see Figure 5.3 for an example). This drastically reduces the chances of a spike being generated randomly, while the values themselves are still in a 'normal' range. Results can be seen in Figure 5.4.


4 "Aceolas"


4 "Bananas"

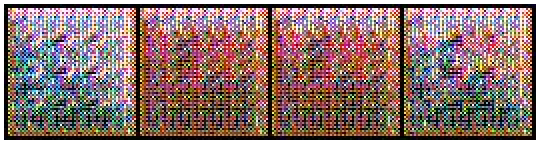

4 "Coconuts"

Figure 5.2: Results from G1 with spiked input after 50 epochs. Spike value = 10.

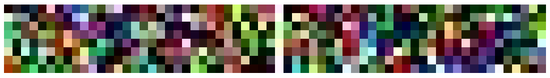
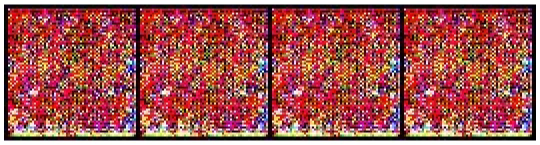
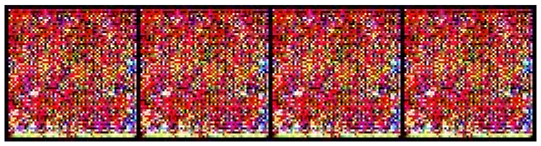


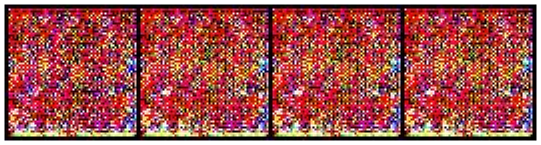
Figure 5.3: Example of a 8 by 64 spiked noise with spike at 32.



4 “Aceolas”



4 “Bananas”



4 “Coconuts”

Figure 5.4: Results from G2a with spiked input after 52 epochs.

Unfortunately the results were underwhelming. Not necessarily just the image quality but there seems to be no discernible difference between the classes. This is likely due to the ‘spike’ not making enough of a difference, the generator network not being deep enough to learn so many different classes or the training time not being long enough. Due to hardware constraints we decided to abandon this approach. Although it would have been nice to have *one model to generate them all*, since the goal is to generate a synthetic dataset, using multiple generators does not have any functional drawbacks, opposed to a classification task for example, for which having one classifier per class would not make too much sense.

So we went on to rearrange the code to produce discriminator-generator pairs for any subset of classes, but most importantly for single class data. The results were (a little bit) more promising. Figure 5.5 shows images produced by three GANs each trained on either apples, bananas or coconuts.

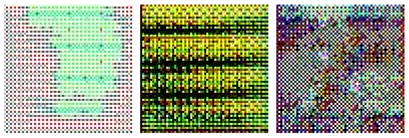


Figure 5.5: “Apples”, “Bananas” and “Coconuts”. D34, G2a, 200 epochs.

Continuing with this setup up to 600 epochs did however only improve results in some cases. Results in general were hard to reproduce and seemed heavily influenced by chance. The ‘moment’ a model performs best, i.e. the number of epochs trained, varies widely. Figure 5.6 shows progress of an ‘orange’-GAN over 600 epochs, performing best between 550 and 560 epochs. Figure 5.7 shows a network trained on ‘kiwifruits’, peeking around epoch 220 only to then revert into what seems like red noise.

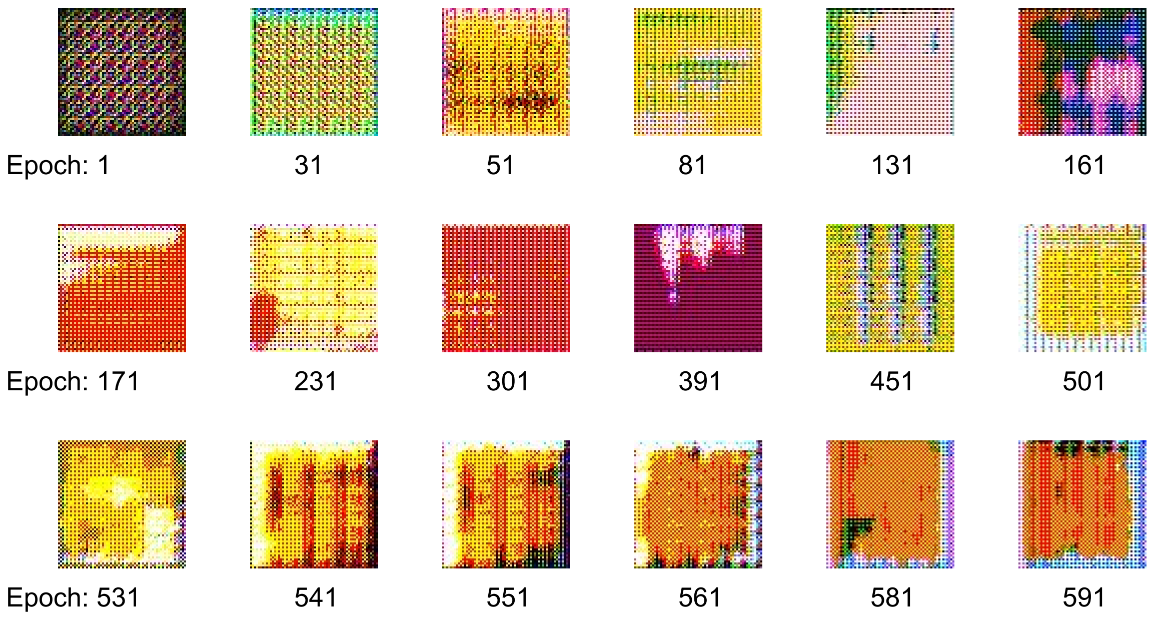


Figure 5.6: “Oranges”. D34, G2a, 1 to 591 epochs.

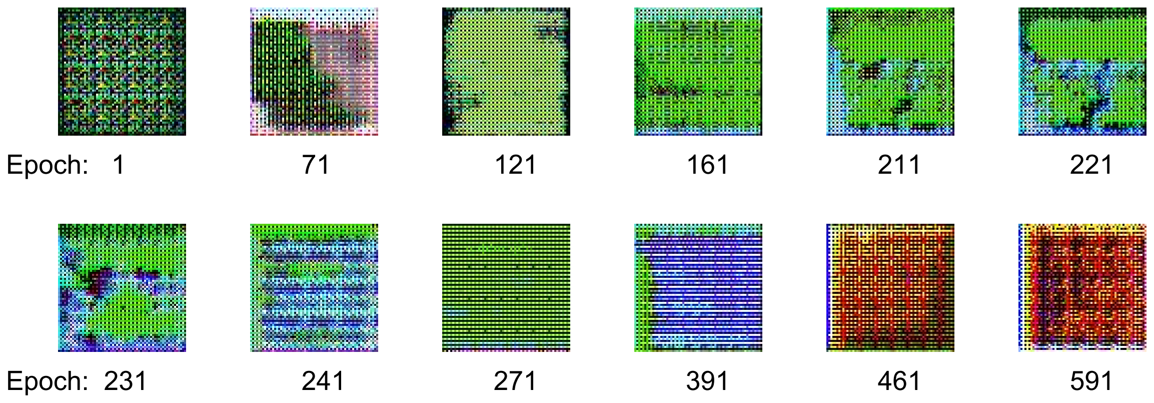
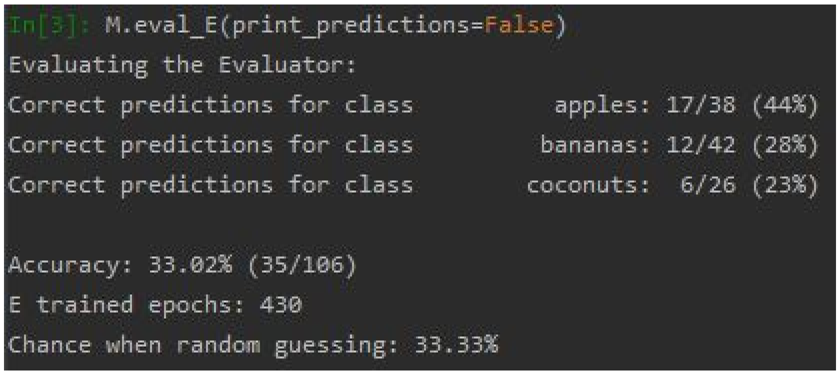


Figure 5.7: “Kiwifruit”. D34, G2a, 1 to 591 epochs.

6. The Evaluator

Evaluating the success of these GANs has so far always been done visually, by simply checking if the output looks anything like a fruit. This is however based on the assumption that the synthetic data needs to look like the original data to the human eye in order to be useful as training data for the new ‘synthetic’ model. However, the only way to determine the actual quality of the generated data, is to train an *evaluator model* on it and test it against the original data.
The evaluator model for this project, i.e. the model trained on generated synthetic data and evaluated on the original data, uses the same parameters as the discriminator. Training the

evaluator on the “Apples”, “Bananas” and “Coconuts” data displayed in Figure 5.5 yielded the following (quite sobering) results:



```
In[3]: M.eval_E(print_predictions=False)
Evaluating the Evaluator:
Correct predictions for class         apples: 17/38 (44%)
Correct predictions for class        bananas: 12/42 (28%)
Correct predictions for class       coconuts:  6/26 (23%)

Accuracy: 33.02% (35/106)
E trained epochs: 430
Chance when random guessing: 33.33%
```

Given that the model was only trained to classify three different classes, an average accuracy of 33.02% equates to the model simply random guessing.

7. Conclusion

Training one pair of models per class generally generates better results than training one GAN on all classes. More work on the generator architecture would be required to achieve more realistic looking results. Deviating from the totally random input for the generator to gain some control over what type of image is created, even within the same class of images, i.e. one red apple, multiple green apples, etc., to the likes of StyleGAN would probably be needed for the evaluator to reliably classify real world input.