AN12954

Porting a Deep Convolutional Generative Adversarial Network on imx8MMini with eIQ

Rev. 0 — 08/2020 Application Note

by: NXP Semiconductors

Contents

1 Introduction.....1
2 DCGAN network.....1
3 Porting DCGANs to i.MX8MMini..... 7
4 Conclusion.....11
5 References.....11
A Appendix..... 11

1 Introduction

1.1 Purpose

This application note introduces a recent discovery in machine learning and artificial intelligence named Deep Convolutional Generative Adversarial Networks (DCGANs) and explains how to port one on an embedded system. Those networks learn to generate data from scratch using a specific training strategy. Indeed, the training is constructed as a two-player game, where two sub-models compete with each other and learn to analyze and copy the diversity and specificity of a dataset.

This application note explains the process of creating an application using a DCGAN for generating handwritten digits and presents an example of solution running on an i.MX8MMini board and based on NXP's BSP and NXP's software eIQ.

1.2 Definitions, acronyms, and abbreviations

Table 1. Definitions, acronyms, and abbreviations

DCGAN	Deep Convolutional Generative Adversarial Network
GAN	Generative Adversarial Network
BSP	Board Support Package
MNIST	Mixed National Institute of Standards and Technology
API	Application Programming Interface
MCU	Microcontroller

2 DCGAN network

The DCGANs (Deep Convolutional Generative Adversarial Networks) were presented in 2015 in the paper of Alec Radfor [1], and they are networks coming from a class of neural networks named GANs. There are many models with different architectures, which include "GAN" in their name, because they use the principle of GANs but include some extensions, such as DCGANs. The GANs (Generative Adversarial Networks) are a recent innovation in machine learning and artificial intelligence. They were invented by Ian Goodfellow and his colleagues in 2014 [2] and lead to a lot of innovation. This section introduces the singularity of GANs and presents details about the model used in this document.

2.1 GANs general presentation

2.1.1 GANs introduction

GANs are neural networks which use deep learning methods to perform generative modeling. The generative modeling represents a class of statistical model in machine learning which can generate new data samples. The particularity of GANs is that they are trained using two sub-models: the generator model which generates the new data and the discriminator model which classifies the data as "real" (coming from the dataset) or "fake (generated by the generator model).

GANs have been researched since their introduction in 2014. Since then, a lot of improvement has been demonstrated, and now these models achieve remarkable results and performance. The figures below show the impressive evolution of GANs results since 2014:

Figure 1. Generated images using a GAN, presented by Ian Goodfellow in his paper in 2014 [2], the format of pictures is 64x64

Figure 2. Generated images using a coupled GAN, presented by Ming-Yu Liu and Oncel Tuzel in their paper in September 2016 [3], the format of pictures is 128x128

Figure 3. Generated images using a progressively growing GAN, presented by Tero Karras, Timo Aila, Samuli Laine and Jaakko Lehtinen in their paper in February 2018 [4], the format of pictures is 1024x1024

The GANs performance has been improved a lot since their introduction, and now it is capable of generating photos that are so realistic that even humans can't tell if they are real or fake. However, the application areas have evolved too, and nowadays GANs are used in a lot of different domains.

2.1.2 GANs application

A lot of researchers work on GANs and discover the wide possibilities of applications. The best known application is the generation of synthetic images similar to the dataset used during the training. This is very impressive and researchers successfully generated convincing results, such as photographs of humans faces, cartoon characters, maps, and so on. This is a very good way to perform data augmentation, which is a big benefit for neural network training. In the machine learning application, it takes long to gather enough data to create a dataset. The more diverse data you have, the better chance you have to train a robust network. The data augmentation is a real chance to improve the performance of a network. Working with GANs allows to use a small amount of data and generate new data.

Lots of new GANs applications have been demonstrated. For example, GANs have shown their capacity to perform anomaly detection, which is a topic highly used in domains such as manufacturing, medical imaging, and cyber security. It has been also used for video prediction, text-to-image translation, 3D object generation, and image high resolution. The possibilities of applications are wide and interesting to more and more people. That is why GANs are a powerful tool for innovation.

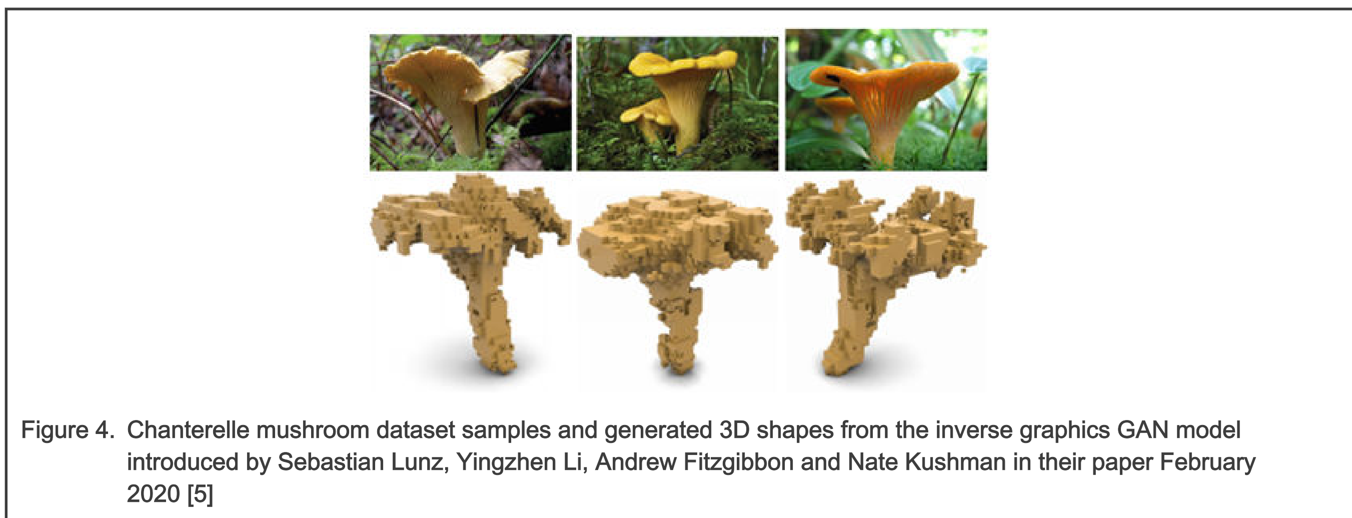


Figure 4. Chanterelle mushroom dataset samples and generated 3D shapes from the inverse graphics GAN model introduced by Sebastian Lunz, Yingzhen Li, Andrew Fitzgibbon and Nate Kushman in their paper February 2020 [5]

2.1.3 Training GANs

The training of GANs is very singular and uncommon because it involves two sub-models: the generator model, which generates new data, and the discriminator model, which classifies the data as "real" (coming from the dataset) or "fake" (generated by the generator model). The two sub-models play a "game" which can be compared to a forger and the police, where the forger is the generator and the police is the discriminator. The generator generates samples which resemble the train data and try to fool the discriminator. The discriminator tries to not be fooled by classifying the samples coming from the generator (fake data) and those coming from the dataset (real data). It can be compared to a forger wanting to counterfeit money by making it as legitimate as possible, while the police try to identify the real and fake money. The generator input is random noise and the discriminator takes the samples created by the generator or the dataset. The GANs architecture is shown in Figure 5.

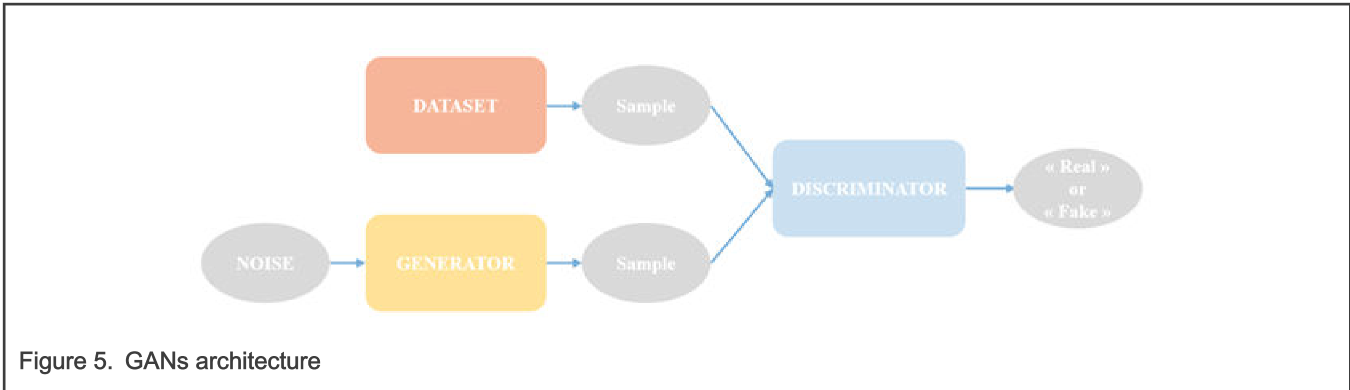


Figure 5. GANs architecture

The difficulty of this training is the fact that the two sub-models must be trained simultaneously and the improvement of one model must lead to the improvement of the other. If the generator is not good enough, the discriminator has no problem to classify the "real" and "fake" data and the model does not converge. If the discriminator is not good enough, a badly generated image is classified as real, and the model does not produce the desired data. To avoid these situations, the generator and discriminator losses are used during the training. The discriminator loss quantifies how well the model achieves the differentiation between real and fake images, and the generator loss quantifies how well the model can trick the discriminator. The training strategy lies in the fact that the two losses are calculated for each sub-models and used to update the weights of the generator and discriminator. The GANs training is done by alternating between these two parts:

- Firstly, the training of the discriminator is executed while the generator is idle. The discriminator is trained on two batches of data: a batch of real data and a batch of fake data. Two losses are then calculated for each batch and the discriminator loss is measured by computing the average.
- Secondly, the training of the generator is executed while the discriminator is idle. The predictions of the discriminator are used to train the generator: the loss of the generator is computed based on the probability that the discriminator predicts that the data generated by the generator are real.

The training schema is shown in Figure 6.

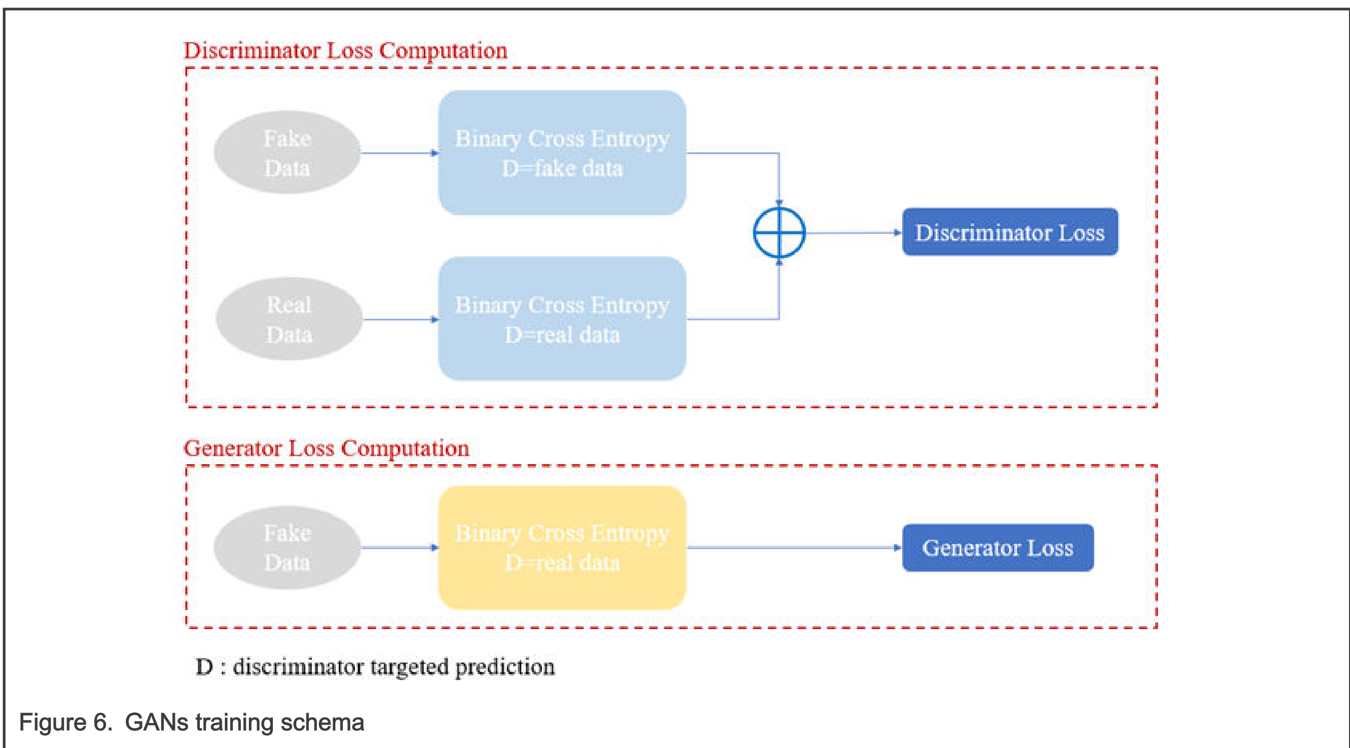


Figure 6. GANs training schema

After the training is done, GANs have a particularity for their evaluation. The performance of trained GANs is challenging to evaluate because GANs use two loss functions unlike other deep-learning neural networks. This means that the model evaluation must be estimated using the quality of the generated data. The common evaluation is made by manual inspection. It enables you to check the quality using human eyes that can verify the precision and details of generated outputs. However, few quantitative measures are introduced and they can provide a robust evaluation, such as the inception score offered by Tim Salimans, et al. in their 2016 paper titled “Improved Techniques for Training GANs” [6] and the Frechet inception distance introduced by Martin Heusel, et al. in their 2017 paper titled “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium” [7].

2.2 DCGANs used in this AN

2.2.1 Model

In this document, the model used is a Deep Convolutional Generative Adversarial Network (DCGAN) based on the paper of Alec Radfor presented in 2015 [1]. Figure 7 shows the generator model (on the left) and the discriminator model (on the right), with all their layers. The particularity of DCGANs is that they use deep convolutional neural networks for the generator and discriminator, whereas GANs usually use fully-connected networks. There are other distinctions, such as the fact that maxpool layers are replaced by learnable upsampling, the fully connected layer of the discriminator’s output is substituted with a flatten and sigmoid layer, the application of Batch Normalization to layers, and other. The advantage of using convolutional neural networks and the other modifications is that it ensures a more stable architecture. By forcing these constraints on the model, it enables you to train a high-quality generator model and that is why DCGANs is the basis of a substantial number of GANs extensions.

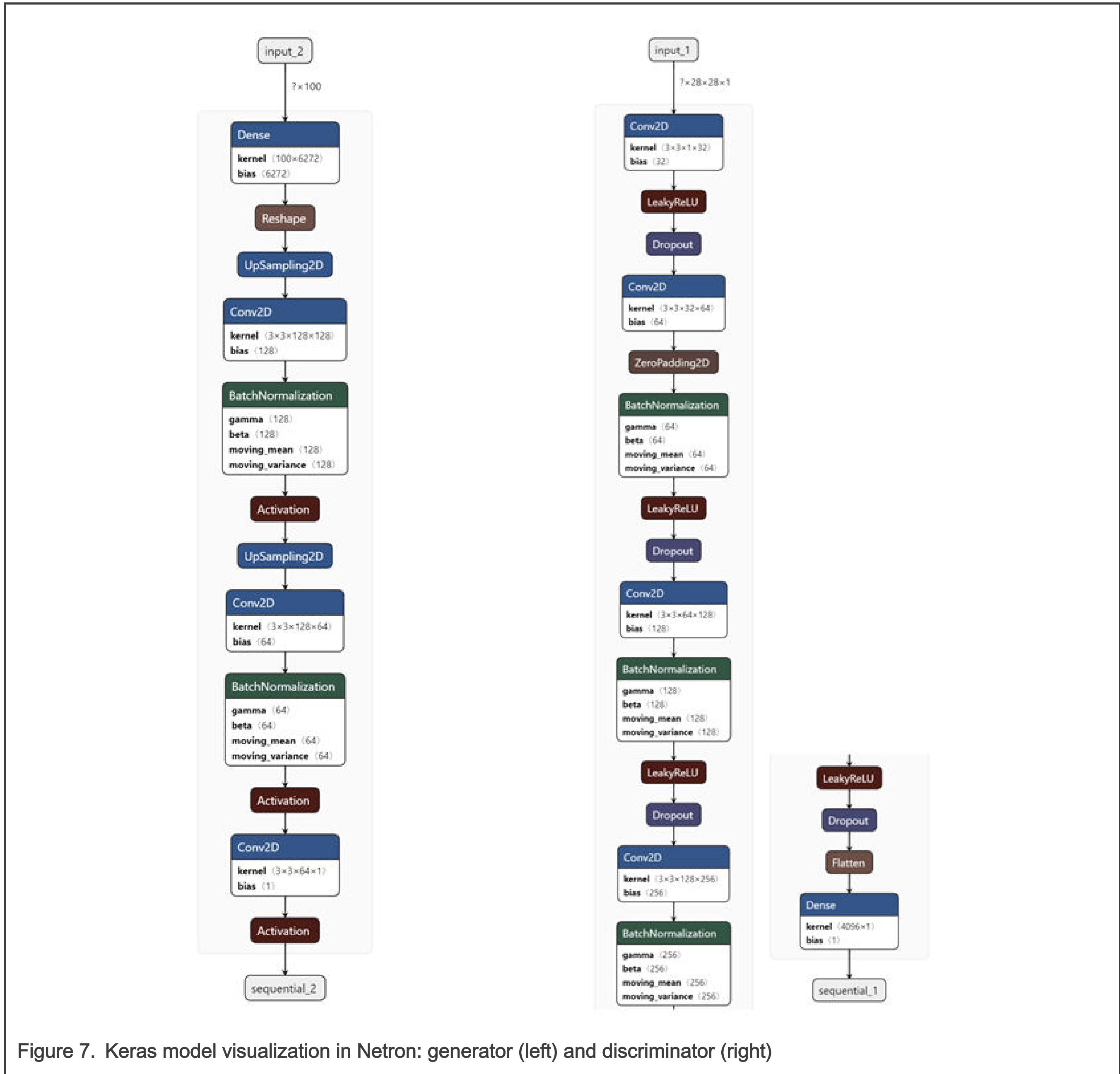


Figure 7. Keras model visualization in Netron: generator (left) and discriminator (right)

2.2.2 MNIST dataset

The Mixed National Institute of Standards and Technolog MNIST(MNIST) dataset [8] is a large dataset which gathers 70,000 images of handwritten digits and their labels, where 60,000 are used for training and 10,000 are used for testing. The digits are centered, grayscale, between 0 to 9, and their format is 28x28.

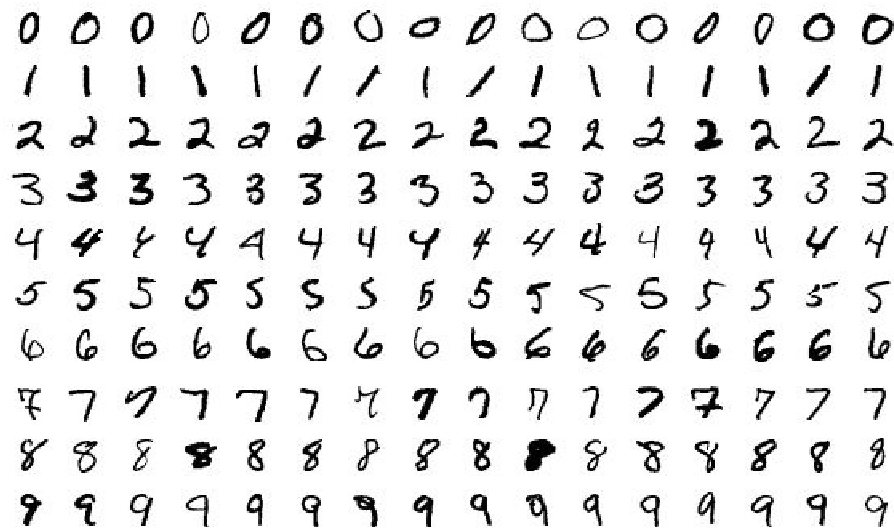


Figure 8. MNIST dataset sample

It was released in 1999 and it has helped to research the benchmarking classification algorithm. Since the explosion of machine learning, the MNIST dataset has been used as the “hello world”. Therefore, it is appropriate to use this dataset as a neutral use case for porting DCGANs to an embedded system.

3 Porting DCGANs to i.MX8MMini

3.1 Network training with Keras

The model’s implementation chosen for this example is available on GitHub and it is the work of Erik Linder-Norén [9], who is a machine learning engineer. GitHub gathers a collection of GANs implementations based on the models described in research papers. The implementation is written in Python and it uses API Keras which is an open-source neural network API written in Python. The feature of this API is that it is user-friendly, high-level, and capable of running on top of different back-ends, such as TensorFlow, CNTK, or Theano.

To train DCGANs, clone the GitHub repository, and go to the DCGAN implementation folder:

$ git clone https://github.com/eriklindernoren/Keras-GAN

$ cd Keras-GAN/dcgan/

In this implementation, the network described in Model is ready to be trained. The training is completed according to this scheme:

For each epoch and for each batch of data:

1. Generate noise vectors of dimension 100.
2. Generate fake data using noise vectors as the generator’s input.
3. Input fake data to the discriminator to get the discriminator’s loss on fake data.
4. Input real data to the discriminator to get the discriminator’s loss on real data.
5. Calculate the generator loss.
6. Calculate the discriminator loss.
7. Update the gradients to minimize the loss values.

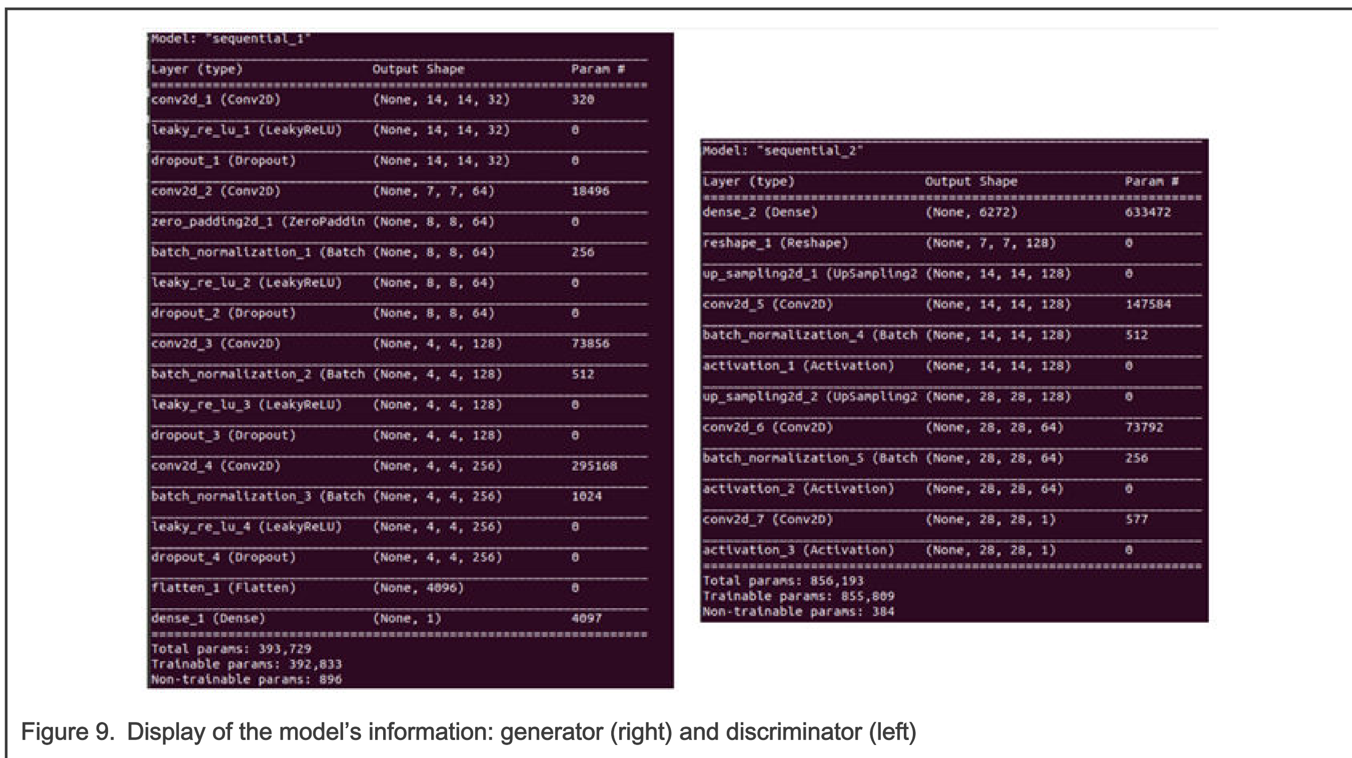
In this code, the “epoch” variable is misleading. In the deep-learning vocabulary, it represents the number of complete passes through the dataset during the training. Here, it is the number of times that the model is trained on one batch. To achieve better results, the number of epochs must be increased. The code does not include saving the model after the training, so few modifications must be done. For that, apply the *dcgan-training.patch* patch available in the repository https://source.codeaurora.org/external/imxsupport/eiq-DCGAN-application-note by executing this command:

```
$ git apply dcgan-training.patch
```

After applying the patch, launch the training using this command:

```
$ python dcgan.py
```

In this document, the version of TensorFlow used is TensorFlow 1.15. After launching the training, the summary of the two sub-models are printed and a few interesting information are displayed, as shown in Figure 9.



Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 14, 14, 32)	320
leaky_re_lu_1 (LeakyReLU)	(None, 14, 14, 32)	0
dropout_1 (Dropout)	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 7, 7, 64)	18496
zero_padding2d_1 (ZeroPaddin	(None, 8, 8, 64)	0
batch_normalization_1 (Batch	(None, 8, 8, 64)	256
leaky_re_lu_2 (LeakyReLU)	(None, 8, 8, 64)	0
dropout_2 (Dropout)	(None, 8, 8, 64)	0
conv2d_3 (Conv2D)	(None, 4, 4, 128)	73856
batch_normalization_2 (Batch	(None, 4, 4, 128)	512
leaky_re_lu_3 (LeakyReLU)	(None, 4, 4, 128)	0
dropout_3 (Dropout)	(None, 4, 4, 128)	0
conv2d_4 (Conv2D)	(None, 4, 4, 256)	295168
batch_normalization_3 (Batch	(None, 4, 4, 256)	1024
leaky_re_lu_4 (LeakyReLU)	(None, 4, 4, 256)	0
dropout_4 (Dropout)	(None, 4, 4, 256)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_1 (Dense)	(None, 1)	4097

Total params: 393,729
Trainable params: 392,833
Non-trainable params: 896

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 6272)	633472
reshape_1 (Reshape)	(None, 7, 7, 128)	0
up_sampling2d_1 (UpSampling2	(None, 14, 14, 128)	0
conv2d_5 (Conv2D)	(None, 14, 14, 128)	147584
batch_normalization_4 (Batch	(None, 14, 14, 128)	512
activation_1 (Activation)	(None, 14, 14, 128)	0
up_sampling2d_2 (UpSampling2	(None, 28, 28, 128)	0
conv2d_6 (Conv2D)	(None, 28, 28, 64)	73792
batch_normalization_5 (Batch	(None, 28, 28, 64)	256
activation_2 (Activation)	(None, 28, 28, 64)	0
conv2d_7 (Conv2D)	(None, 28, 28, 1)	577
activation_3 (Activation)	(None, 28, 28, 1)	0

Total params: 856,193
Trainable params: 855,809
Non-trainable params: 384

Figure 9. Display of the model’s information: generator (right) and discriminator (left)

All the layers, output shapes, and number of trainable and non-trainable parameters are indicated, which provides an idea of the training extension.

During the training, the following information is printed in the terminal:

```
1 [D loss: 0.685027, acc.: 57.81%] [G loss: 0.982729]

2 [D loss: 0.574230, acc.: 67.19%] [G loss: 1.022767]

3 [D loss: 0.356288, acc.: 82.81%] [G loss: 0.921926]
```

For each epoch, which is the first number on the left, the accuracy of the discriminator and the losses of the discriminator and generator are displayed. This is a good way to monitor the evolution of the training and to ensure its good progression.

During the training, samples of generated images are saved at a pre-defined interval and the performance of the training can be observed. The generated images saved at different intervals of the training are shown in Figure 10.

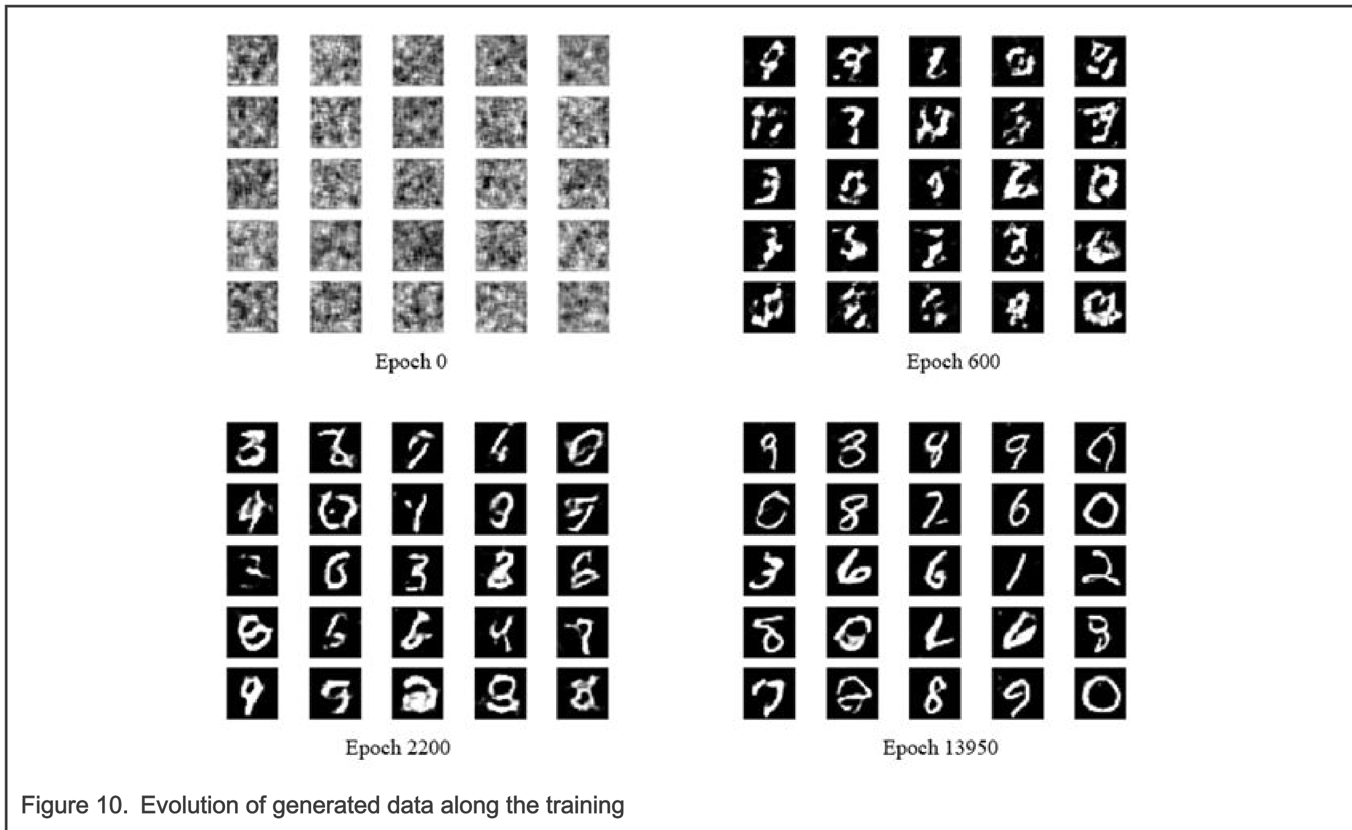


Figure 10. Evolution of generated data along the training

The progression is clearly shown in the generated images at epoch 0, the generated images are made of noise, and the shapes of digits are appearing at epochs 600. They are then refined at epoch 2200 and the generated images look like the MNIST Digits at the last epoch. At the end of the training, the discriminator, generator, and combined models are saved in the **.h5* and **.json* formats. The **.h5* format is used for the conversion in Tflite, which is described in the next section.

3.2 Conversion in TFlite

TensorFlow Lite is a light-weight version of TensorFlow which offers a set of tools and core operators for the floating and quantize data to optimize the integration of TensorFlow models on Android OS and iOS devices, embedded and IoT devices, and MCUs. It enables on-device deep learning inference with low latency and optimization of the trained Keras or TensorFlow models.

A precise guide for the TFlite quantization during the conversion is available on the NXP website https://community.nxp.com/community/eiq/blog/2019/07/15/eiq-sample-apps-tflite-quantization. In this document, the model runs on the CPU of i.MX8M Mini, so during the conversion from Keras to TFlite model, the parameters are kept in float. However, only the generator model is kept for the conversion because the goal of this document is to create an application which generates new images, so the discriminator is not needed. The conversion can be done using the converter command line or using the Python API. To use the python API, create a new Python file and type the following lines into it:

```
import tensorflow as tf

import warnings

warnings.filterwarnings('ignore')

#model=tf.keras.models.load_model("dcgan/saved_model/generator_model.h5")

converter = tf.lite.TFLiteConverter.from_keras_model_file("dcgan/saved_model/generator_model.h5")

tfmodel = converter.convert()

open ("dcgan_generator.tflite" , "wb") .write(tfmodel)
```

After launching the script, the generator model is converted into a TFlite model and ready to be deployed on the i.MX8MMini.

3.3 eIQ

The NXP eIQ machine-learning software development environment is created to facilitate the development of applications using machine-learning algorithms on NXP MCUs, i.MX RT crossover MCUs, and i.MX family SoCs. It provides a set of libraries and development tools which includes inference engines, neural network compilers, and optimized libraries.

However, eIQ is dedicated to achieve inference of networks and standard machine-learning algorithms, so this is why the DCGAN training was previously done on a Linux OS computer.

The advantage of eIQ is that it can be used with a wide range of frameworks and inference engines like TensorFlow Lite, Arm® NN, and Arm Compute Library. For more information on this software, see the NXP website (https://www.nxp.com/design/software/development-software/eiq-ml-development-environment).

For this project, the TensorFlow Lite library integrated in the eIQ is used. To use this on i.MX8MMini, two things are needed: a BSP containing the eIQ software for the i.MX8MMini board and the toolchain with eIQ on the host machine.

The Yocto BSP and the toolchain can both be generated. For more information, see https://www.nxp.com/design/software/embedded-software/linux-software-and-development-tools/embedded-linux-for-i-mx-applications-processors:IMXLINUX for generating the BSP, and *NXP eIQ™ Machine Learning Software Development Environment for i.MX Applications Processors* (document UM11226) section 3.2.9. Generating the Toolchain” for generating the toolchain.

3.4 Implementation details

Now that the model is trained and converted into an adapted format for an embedded system, the last step is to create an application which performs the inference on i.MX8MMini with the software eIQ. The source code of the application is available on Code Aurora:

https://source.codeaurora.org/external/imxsupport/eiq-DCGAN-application-note/

This application uses the C++ TensorFlow Lite library to load and interpret the TFlite model and to perform inference. The resulting images generated by the DCGAN are displayed on the screen connected to the board.

To compile on the host machine, download the application and add the trained TFlite model into the folder. Then source the toolchain. This toolchain provides a set of tools including compilers, libraries, and header files which allow to cross-compile the code for the image used on the i.MX8MMini board. After sourcing the toolchain, compile the application by running the following:

```
$ make -f Makefile.linux
```

It generates the binary DCGAN that can be executed on the board. Now that the binary of the application is created, power the i.MX8MMini board and connect a display to the board using an IMX-MIPI-HDMI adapter. Create a new folder on the board and copy the binary with the TFlite model and run the application by executing the binary as follows:

```
root@imx8mmevk:dcgan/~# ./DCGAN
```

During the execution, 10 generated images are displayed on the screen and saved on the board in the *result_im_number.png* file. They can be seen afterward using the following command:

```
root@imx8mmevk:dcgan/~# Weston-image result_im_number.png
```

Figure 11 shows examples of three generated images representing the handwritten digit “0” created on the board.

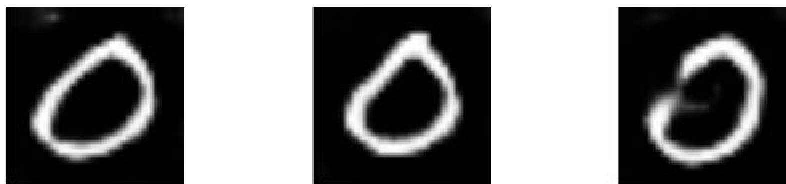


Figure 11. Handwritten digits “0” generated on the i.MX8MMini

4 Conclusion

This application note introduces the DCGAN networks and the challenge of generating new images of handwritten digits using deep learning. It demonstrates an example of DCGAN application running on the i.MX8MMini embedded system i using the NXP software eIQ. This allows to also show a development's demonstration of an application using deep learning on an NXP board, from the training done on an host machine to the deployment on an embedded system. This process is simplified with eIQ, which provides an opportunity to develop and deploy lots of applications using artificial intelligence and machine learning. eIQ keeps getting upgraded, advanced, and optimized, which enhances the possibilities of implementating innovation on an NXP embedded system.

5 References

- Jason Brownlee (2019): introduction to Generative Adversarial Networks https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/
- Jason Brownlee (2019): example of GANs applications https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/
- Open Data Science (2019): example of GANs use cases https://medium.com/@ODSC/6-unique-gans-use-cases-24cab2aa924d

A Appendix

1. Alec Radfor (2015): introduction of DCGAN https://arxiv.org/pdf/1511.06434.pdf
2. Ian J. Goodfellow (2014): introduction of GAN https://arxiv.org/pdf/1406.2661.pdf
3. Ming-Yu Liu, Oncel Tuzel (2016): introduction of Coupled GANhttps://arxiv.org/pdf/1606.07536.pdf
4. Tero Karras, Timo Aila, Samuli Laine and Jaakko Lehtinen (2018): i ntroduction of Progressively Growing GANhttps://arxiv.org/pdf/1710.10196.pdf
5. Sebastian Lunz, Yingzhen Li, Andrew Fitzgibbon and Nate Kushman (2020): introduction of Inverse Graphics GAN model https://arxiv.org/pdf/2002.12674.pdf
6. Tim Salimans , et al. (2016): introduction of inception scorehttps://arxiv.org/pdf/1606.03498.pdf
7. Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler (2018): introduction of Frechet inception https://arxiv.org/pdf/1706.08500.pdf
8. LeCun, Y. (1999) : The Mnist Database http://yann.lecun.com/exdb/mnist/
9. Erik Linder-Norén (2018): implementation of GANs family in Keras https://github.com/eriklindernoren/Keras-GAN

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2020. All rights reserved.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 08/2020
Document identifier: AN12954