Exploring the Impact of Deep Learning Data Augmentation on Medical Imagery Classification

Quentin Le Roux

Abstract. We evaluate the impact of data augmentation on the performance of a Convolutional Neural Network on 8 medical datasets. We demonstrate accuracy improvements on 7 of the datasets when using either Geometric or Deep Learning data augmentation methods[1].

1 Introduction

MedMNIST is a collection of 18 labeled biomedical datasets [7], standardized to fit the MNIST format [9]. The authors created MedMNIST to provide an educational benchmark of classification tasks[2], and to compare classifiers in a medical setting. Using MedMNIST, we are interested in studying the impact of data augmentation on the performance of a classifier.

Data augmentation techniques are split in two families: geometric (e.g. rotations, shearing) and Deep Learning generative approaches. In this study, we select 1 geometric data augmentation pipeline and 3 Deep Learning data augmentation techniques to assess whether performance improvements can be observed on a given classifier. We retain 8 2-dimensional MedMNIST datasets for this study (see Table 1).

In the next section, we introduce our implementation methodology, which methods we use and their advantages. Afterwards, and before concluding, we will cover our preliminary results and observations.

Table 1. MedMNIST datasets retained for studying the impact of data augmentation

Dataset	Type (Fig. with montage)	Task (n. classes)	Subset size		
			Training	Validation	Test
PathMNIST	Colon pathology (Fig. 1)	Multi-class (9)	89,996	10,004	7,180
DermaMNIST	Dermatoscope (Fig. 2)	Multi-class (7)	7,007	1,003	2,005
OctMNIST	Retinal OCT (Fig. 3)	Multi-class (4)	97,477	10,832	1,000
PneumoniaMNIST	Chest X-ray (Fig. 4)	Binary-class (2)	4,708	524	624
BreastMNIST	Breast Ultrasound (Fig. 5)	Binary-class (2)	546	78	156
OrganAMNIST	Axial Abdo. CT (Fig. 6)	Multi-class (11)	34,581	6,491	17,778
OrganCMNIST	Coronal Abdo. CT (Fig. 7)	Multi-class (11)	13,000	2,392	8,268
OrganSMNIST	Sagittal Abdo. CT (Fig. 8)	Multi-class (11)	13,940	2,452	8,829

2 Methodology

2.1 A Baseline Classifier

We reuse the Pytorch Convolutional Neural Network (CNN) [13,14] provided by the MedMNIST team on their GitHub page [8]. The CNN depth and hyperparameters

[1] Code and results used in this document are available on Google Colab: `https://colab.research.google.com/drive/1J64flVq0ALWS7JBd8hj5qF1bwdbHlmeR?usp=sharing`

[2] The MedMNIST benchmark is available at `https://medmnist.com/`.

are fixed to control for each model factor besides our selection of data augmentation methods. The only divergence between architectures stems from each dataset's number of classes, which modulates the CNN's output size. The CNN implementation for the PathMNIST dataset is reproduced in Fig. 9.

Each CNN implementation is trained for 20 epochs with a Stochastic Gradient Descent optimizer [1] with a 0.001 learning rate and a 0.9 momentum. The loss is computed via cross-entropy (binary cross-entropy in the case of binary classification). Batch size is set to 64. We reuse the MedMNIST custom evaluator object [8], which provides two output metrics to assess model performance: accuracy and area-under-the-curve (AUC). When assessing the performance of a model on a test set, we use the model weights from the epoch that yields the best validation accuracy.

2.2 Dealing with Imbalancedness

The 8 retained datasets display class imbalances (see Fig. 10, 11, 12, 13, 14, 15, 16, 17). Furthermore, the amount of available datapoints in a dataset can be small (e.g. the BreastMNIST dataset has less than 1,000 images). Such imbalance and lack of datapoints may impact a classifier's convergence towards an optimum [15]. As such, data augmentation may help improve the generalization capability of a model by reducing overfitting to the training data [18].

To further help reduce the effect of class imbalance, we will also include in our process PyTorch's Weighted Random Sampling (WRS) sampler object. This allows us to deal with class imbalance at the dataloader level. We will evaluate all our implementations with and without WRS.

2.3 Augmenting Datasets: Geometric Approach

Using Pytorch Transforms, we build a geometric data augmentation pipeline, conditioned on whether the underlying MedMNIST dataset is grayscale or RGB.

In the case of a grayscale image dataset, we consider a 3-step pipeline: random sharpness adjustment, random contrast, then random horizontal flipping (with default Pytorch parameters). In the RGB case, we add a random color hue jitter of 0.05 after sharpness and contrast adjustments. The pipeline ends with a normalization step.

We do not use shearing or translation as part of this process as we expect medical images to be collected in a controlled manner that would account for such alterations. Of note, the OctMNIST dataset displays evidence of shearing during the data acquisition phase (see Fig. 3). Rotations could be considered for datasets such as PathMNIST or DermaMNIST at a later point.

2.4 Augmenting Datasets: Deep Learning Approaches

The Manifold Hypothesis states that high-dimensional data are representations of lower-dimensional objects [4]. In this context, Variational Autoencoders (VAEs) assume that a random process involving a continuous random variable has generated the data. A latent space representation of the data can thus be learned via a pair of encoder and decoder networks [2].

Similarly, Generative Adversarial Networks (GANs) generate new data using adversarial training, relying on a pair of generator and discriminator networks to learn a new data-generating process [6]. Deep Convolutional GANs have been shown to improve medical data classification [5,20].

We are interested in implementing both types of methods for our Deep Learning data augmentation setups. As such, we settle on 3 techniques: Conditional VAE, Joint VAE, and Conditional GAN.

Conditional VAEs learn a latent space representation of a dataset where the generation process is informed by the data classes [16] as classes are part of the VAE's input (see Fig. 18 for an example representation of a Conditional VAE, and Fig. 19 and Fig. 20 for the encoder and decoder architectures in the PathMNIST case).

Instead of learning the prior distribution $P_\theta(z)$ of the latent space z (with the data x being generated by the distribution $P_\theta(x|z)$), a Conditional VAE learns $P_\theta(z|y)$ with y the data labels conditioning the latent space. As such, Conditional VAEs can generate new labeled data.

Joint VAEs learn both a continuous and discrete representation of some data [3]. They learn the prior distribution $P_\theta(z, c)$, with z the continuous latent space and c the discrete/categorical latent space. The data x is thus generated by the distribution $P_\theta(x|z, c)$ (see Fig. 21 for an example representation of a Joint VAE, and Fig. 22 and Fig. 23 for the encoder and decoder architectures in the PathMNIST case).

A Joint VAE may help better capture the features of medical images that a purely continuous latent space might not. Indeed, medical images are usually obtained in strictly controlled environments where some variables are discrete such as the left-right orientation of a chest CT scan.

The main drawback is the absence of conditioning on classes. We can either train one Joint VAE per dataset class or train a single Joint VAE on a whole dataset. The latter case, though less computationally expensive, implies that we do not have the ability to categorically condition the underlying generative process.

Conditional GANs are similar to Conditional VAEs. They generate data conditioned on the data classes [10] (see Fig. 24 for an example representation of a Conditional GAN, and Fig. 25, and Fig. 26 for the discriminator and generator architectures in the PathMNIST case).

Conditional GANs rely on using labels as inputs to guide the learning of the generator and discriminator networks [11,12,17]. The discriminator is evaluated on the similarity between generated and real data, and the match between the predicted and true labels of the generated data. The rationale behind using a Conditional GAN is similar to that of the Conditional VAE: we want to generate new labeled data as part of a data augmentation process.

2.5 VAE and GAN Training Setup

Our two VAEs rely on existing implementations available online [19]. We use the TorchFusion library for the Conditional GAN [12]. We respectively train the two

VAEs for 200 epochs (with early stopping), and the Conditional GAN for 100. The Conditional VAE is trained with a latent space of dimension 150 and the Joint VAE with continuous and discrete latent spaces of dimensions 100 and 50 respectively. The Conditional GAN is trained with a latent space of dimension 128.

2.6 Data Augmentation Setup

As a result of training (see Fig. 27 for the three methods' loss per epoch in the PathMNIST case), we generate new images for each retained MedMNIST datasets. We only augment the training sets; validation and test sets are untouched.

The Conditional VAE and GAN generate new data in amounts inversely proportional to the scarcity of a class in each original datasets. We add to each dataset (in the order stated in Table 1) the following amounts of images: 4%, 10%, 4%, 10%, 10%, 5%, 10%, 10%, i.e., 3599, 700, 3899, 470, 54, 1729, 1300, and 1394 images (selected due to memory usage issues and restrictions encountered during development). Example generated images for DermaMNIST are shown in Fig. 28.

The Joint VAE augments the MedMNIST datasets by interweaving the original images with their VAE reconstructions. We expect this setup to increase intra-class variability, which may help our implementations yield better results.

3 Results

With no WRS, we find that using a data augmentation method improves model performance over the baseline on all datasets but BreastMNIST (see Tables 2 and 3, best results per dataset in bold). In all such cases but PathMNIST and PneumoniaMNIST, we find that a Deep Learning augmentation approach beats the geometric one.

WRS improves model performance on 7 out of 8 datasets when no data augmentation is involved (see Table 2, best results per dataset in bold). This highlights class imbalance as an existing problem to account for in MedMNIST datasets.

Overall, we can evidence a performance improvement on 7 out of the 8 MedMNIST datasets by using either geometric (2 cases) or Deep Learning (5 cases) data augmentation approaches with or without WRS (see Tables 4, best results per dataset in bold, and 5). Though we can find cases leading to performance improvements, we must highlight that in most tested cases, we did not see any improvement.

4 Conclusion

It is possible to find a data augmentation technique that improves a CNN's accuracy in MedMNIST classification tasks. The use of a Conditional GAN or VAE improved accuracy on a majority of datasets, implying that class-dependent generative processes can be relevant in improving performance.

The second key takeaway is that data augmentation alone does not guarantee improvements. We insist that most of our implementations have shown equivalent or worse results compared to the baseline. This is a case of multiple comparisons problem (the more cases are tested the likelier we are to find an improving one).

Finally, we must recall the MedMNIST authors' insistence on the data not being intended for clinical use. We must indeed beware extending any preliminary conclusion, especially in possibly critical applications in healthcare and medical research.

Table 2. Test Accuracy & AUC obtained with geometric data augmentation.

	no GDA or WRS		WRS, no GDA		no WRS, GDA		WRS & GDA	
Dataset	AUC	Acc.	AUC	Acc.	AUC	Acc.	AUC	Acc.
PathMNIST	97.99	84.36	96.67	79.93	98.94	87.97	**99.14**	**88.29**
DermaMNIST	89.21	65.94	89.62	**69.43**	**89.97**	66.53	89.43	64.39
OctMNIST	93.40	70.40	**94.29**	**76.6**	91.54	68.8	92.98	74.20
PneumoniaMNIST	95.80	85.58	96.07	87.98	95.74	86.38	**96.66**	**90.38**
BreastMNIST	**88.35**	82.69	86.45	**84.62**	82.14	78.21	83.02	67.95
OrganAMNIST	99.12	88.10	**99.14**	**88.49**	98.87	87.03	98.85	87.43
OrganCMNIST	98.72	86.25	**98.74**	**86.32**	98.35	83.19	98.30	81.66
OrganSMNIST	**97.46**	71.92	96.42	**71.94**	95.42	68.64	95.27	63.95

Table 3. Test Accuracy & AUC obtained with DL data augmentation (no WRS).

	Cond. VAE		Joint VAE		Cond. GAN		Improved perf. over	
Dataset	AUC	Acc.	AUC	Acc.	AUC	Acc.	no GDA	GDA
PathMNIST	**98.11**	**87.49**	97.90	85.08	97.76	85.52	Yes	No
DermaMNIST	90.32	**74.71**	**90.34**	73.22	90.13	71.67	Yes	Yes
OctMNIST	**93.53**	**72.50**	93.20	73.00	92.93	67.30	Yes	Yes
PneumoniaMNIST	**96.81**	**86.38**	93.52	85.58	96.27	85.42	Yes	id.
BreastMNIST	87.49	78.85	84.02	77.56	**83.31**	**80.77**	No	Yes
OrganAMNIST	99.00	87.87	98.98	88.08	**99.07**	**88.54**	Yes	Yes
OrganCMNIST	**98.66**	**86.59**	98.43	85.87	**98.66**	85.43	Yes	Yes
OrganSMNIST	96.01	70.98	95.86	69.66	**96.46**	**72.27**	Yes	Yes

Table 4. Test Accuracy & AUC obtained with DL data augmentation (with WRS).

	Cond. VAE		Joint VAE		Cond. GAN		Improved perf. over	
Dataset	AUC	Acc.	AUC	Acc.	AUC	Acc.	WRS, no GDA	WRS & GDA
PathMNIST	**97.95**	**86.16**	96.12	74.81	96.28	79.54	Yes	No
DermaMNIST	**90.21**	**73.02**	88.68	66.38	88.54	68.93	Yes	Yes
OctMNIST	91.99	**74.10**	**93.70**	72.30	92.17	70.00	No	No
PneumoniaMNIST	**95.30**	84.78	94.77	**89.42**	95.87	87.02	Yes	No
BreastMNIST	85.69	82.69	86.93	80.13	**89.33**	**85.9**	Yes	Yes
OrganAMNIST	**99.15**	**88.57**	99.06	88.33	99.08	88.20	Yes	Yes
OrganCMNIST	98.53	86.07	98.54	85.27	**98.83**	**86.94**	Yes	Yes
OrganSMNIST	96.01	**71.57**	95.60	68.86	**96.16**	71.47	No	Yes

Table 5. Best model per dataset (based on test Accuracy results)

Dataset	Model	Accuracy gain over		Loss	Confusion
		no WRS or GDA	WRS, no GDA	Convergence	Matrix
PathMNIST	WRS + GDA	+3.93	+8.36	Fig. 29	Fig. 30
DermaMNIST	Conditional VAE	+8.77	+5.28	Fig. 31	Fig. 32
OctMNIST	WRS only	+6.20	id.	Fig. 33	Fig. 34
PneumoniaMNIST	WRS + GDA	+4.80	+2.40	Fig. 35	Fig. 36
BreastMNIST	WRS + Cond. GAN	+3.21	+1.41	Fig. 37	Fig. 38
OrganAMNIST	WRS + Cond. VAE	+0.47	+0.08	Fig. 39	Fig. 40
OrganCMNIST	WRS + Cond. GAN	+0.69	+0.62	Fig. 41	Fig. 42
OrganSMNIST	Cond. GAN	+0.35	+0.33	Fig. 43	Fig. 44

References

1. Bottou, L., Curtis, F.E., Nocedal, J.: "Optimization Methods for Large-Scale Machine Learning." In: SIAM Review, Vol. 60, No.2, pp. 223-311. (2018)
2. Doersch, C.: "Tutorial on Variational Autoencoders." In: eprint arXiv:1606.05908. (2016)
3. Dupont, E.: "Learning Disentangled Joint Continuous and Discrete Representations." arXiv preprint arXiv:1804.00104. (2018)
4. Fefferman, C., Mitter, S., Narayanan, H.: "Testing the manifold hypothesis." In: Journal of the American Mathematical Society, 29(4), pp.983-1049. (2016)
5. Frid-Adar, M., Diamant, I., Klang, E., Amitai, M., Goldberger, J., Greenspan, H.: "GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification." In: Neurocomputing, Volume 321, pp. 321-331. (2018)
6. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y.: "Generative Adversarial Nets." NIPS. (2014)
7. Jiancheng, Y., Rui, S., Donglai, W., Zequan, L., Lin, Z., Bilian, K., Hanspeter, P., Bingbing, N.: "MedMNIST v2: A Large-Scale Lightweight Benchmark for 2D and 3D Biomedical Image Classification". arXiv preprint arXiv:2110.14795. (2021)
8. Jiancheng, Y., Rui, Bingbing, N.: "MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis". In: IEEE 18th International Symposium on Biomedical Imaging (ISBI). (2021)
9. LeCun, Y., Cortes, C.: "MNIST handwritten digit database." (2010)
10. Mirza, M., Osindero, S.: "Conditional Generative Adversarial Nets." In: arXiv eprint arXiv:1411.1784. (2014)
11. Nayak, M.: "An Introduction to Conditional GANs." In: Data Drive Investor (blog). `https://medium.datadriveninvestor.com/an-introduction-to-conditional-gans-cgans-727d1f5bb011`. (2019)
12. Olafenwa, J.: "Simple Introduction to Conditional GANs with TorchFusion and PyTorch." In: Towards Data Science (blog). `https://towardsdatascience.com/simple-intro-to-conditional-gans-with-torchfusion-and-pytorch-404264a3267c`. (2018)
13. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Chintala, S.: "Automatic Differentiation in PyTorch." In: Advances in Neural Information Processing Systems 32 (pp. 8024–8035). Curran Associates, Inc. (2019)
14. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Chintala, S.: "PyTorch: An Imperative Style, High-Performance DL Library." In: NIPS 2017 Workshop Autodiff Submission. (2017)
15. Quasim, A., Ezhov, I., Suprosanna, S., Schoppe, O., Paetzold, J., Anjany, S., Kofler, F., Lipkova, J., Hongwei, L., Bjoern, M.: "Red-GAN: Attacking class imbalance cia conditioned generation. Yet another medical imaging perspective." In: Proceedings of Machine Learning Research, 1-14. (2020)
16. Rahman, M.A.: "Understanding Conditional Variational Autoencoders." `https://theaiacademy.blogspot.com/2020/05/understanding-conditional-variational.html`. (2020)
17. Sayak, P.: "Conditional GAN." In: Keras Documentation. `https://keras.io/examples/generative/conditional_gan/`. (2021)
18. Shorten, C., Khoshgoftaar, T.M.: "A survey on Image Data Augmentation for DL." In: J Big Data 6, 60. (2019)
19. Subramanian, A.K.: "PyTorch-VAE." GitHub repository, `https://github.com/AntixK/PyTorch-VAE`. (2020)
20. Yi, X., Walia, E., Babyn, P.: "Generative Adversarial Network in Medical Imaging: A Review." In: Medical Image Analysis. 58. 101552. (2019)

Annex

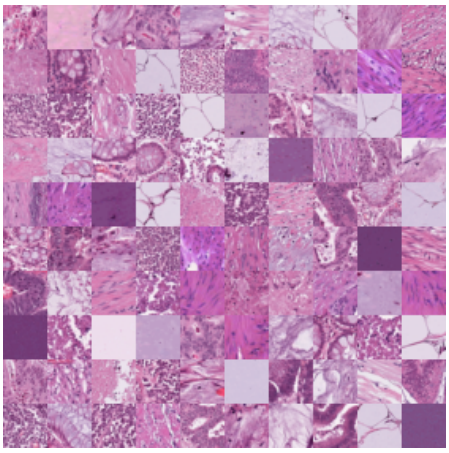

Fig. 1. Montage of training samples from the PathMNIST dataset.

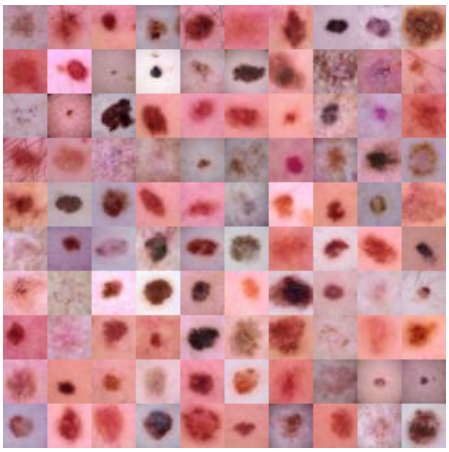

Fig. 2. Montage of training samples from the DermaMNIST dataset.

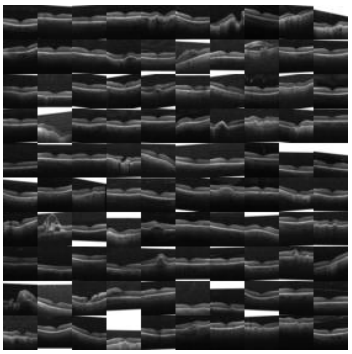

Fig. 3. Montage of training samples from the OctMNIST dataset.

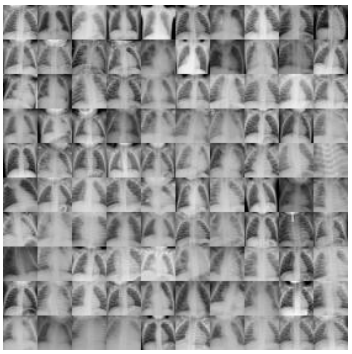

Fig. 4. Montage of training samples from the PneumoniaMNIST dataset.

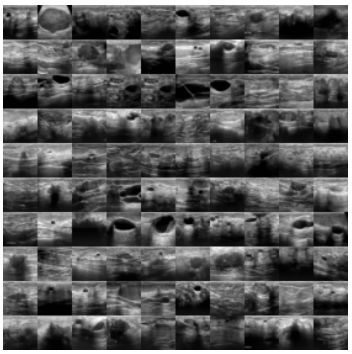

Fig. 5. Montage of training samples from the BreastMNIST dataset.

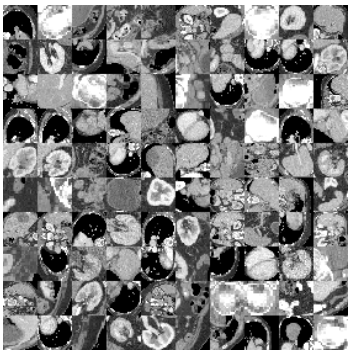

Fig. 6. Montage of training samples from the OrganMNIST_Axial dataset.

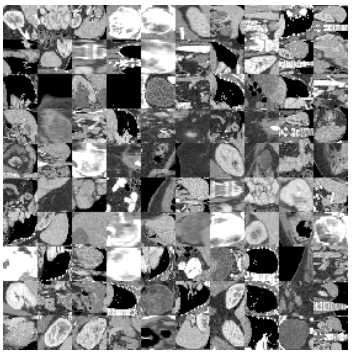

Fig. 7. Montage of training samples from the OrganMNIST_Coronal dataset.

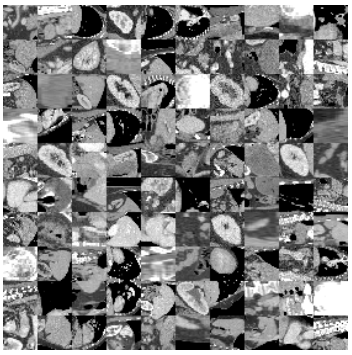

Fig. 8. Montage of training samples from the OrganMNIST_Sagittal dataset.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 26, 26]             448
       BatchNorm2d-2          [-1, 16, 26, 26]              32
              ReLU-3          [-1, 16, 26, 26]               0
            Conv2d-4          [-1, 16, 24, 24]           2,320
       BatchNorm2d-5          [-1, 16, 24, 24]              32
              ReLU-6          [-1, 16, 24, 24]               0
         MaxPool2d-7          [-1, 16, 12, 12]               0
            Conv2d-8          [-1, 64, 10, 10]           9,280
       BatchNorm2d-9          [-1, 64, 10, 10]             128
             ReLU-10          [-1, 64, 10, 10]               0
           Conv2d-11            [-1, 64, 8, 8]          36,928
      BatchNorm2d-12            [-1, 64, 8, 8]             128
             ReLU-13            [-1, 64, 8, 8]               0
           Conv2d-14            [-1, 64, 8, 8]          36,928
      BatchNorm2d-15            [-1, 64, 8, 8]             128
             ReLU-16            [-1, 64, 8, 8]               0
        MaxPool2d-17            [-1, 64, 4, 4]               0
           Linear-18                 [-1, 128]         131,200
             ReLU-19                 [-1, 128]               0
           Linear-20                 [-1, 128]          16,512
             ReLU-21                 [-1, 128]               0
           Linear-22                   [-1, 9]           1,161
================================================================
Total params: 235,225
Trainable params: 235,225
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.82
Params size (MB): 0.90
Estimated Total Size (MB): 1.73
----------------------------------------------------------------
```

Fig. 9. Convolutional Neural Network classifier architecture used with the PathMNIST dataset.

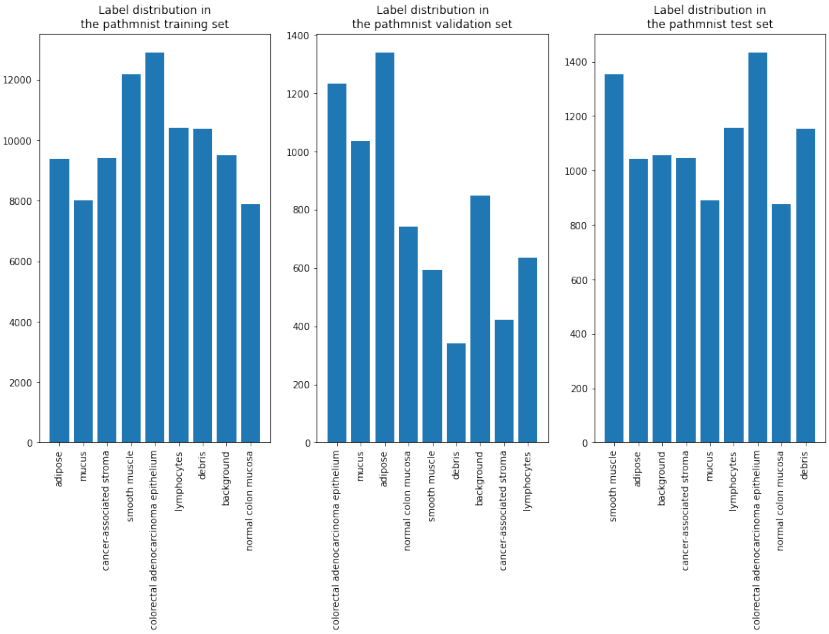

Fig. 10. Class distribution of the PathMNIST dataset.

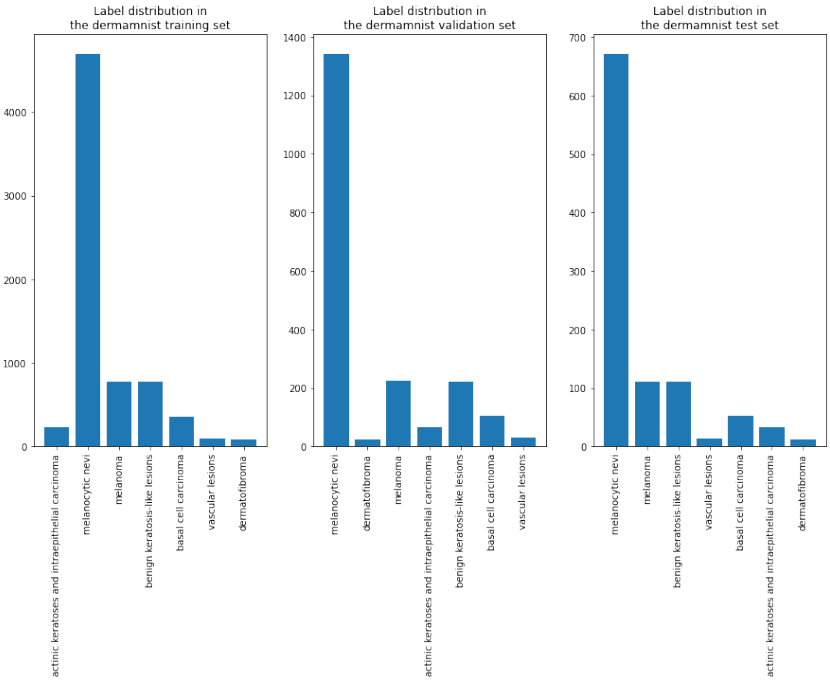

Fig. 11. Class distribution of the DermaMNIST dataset.

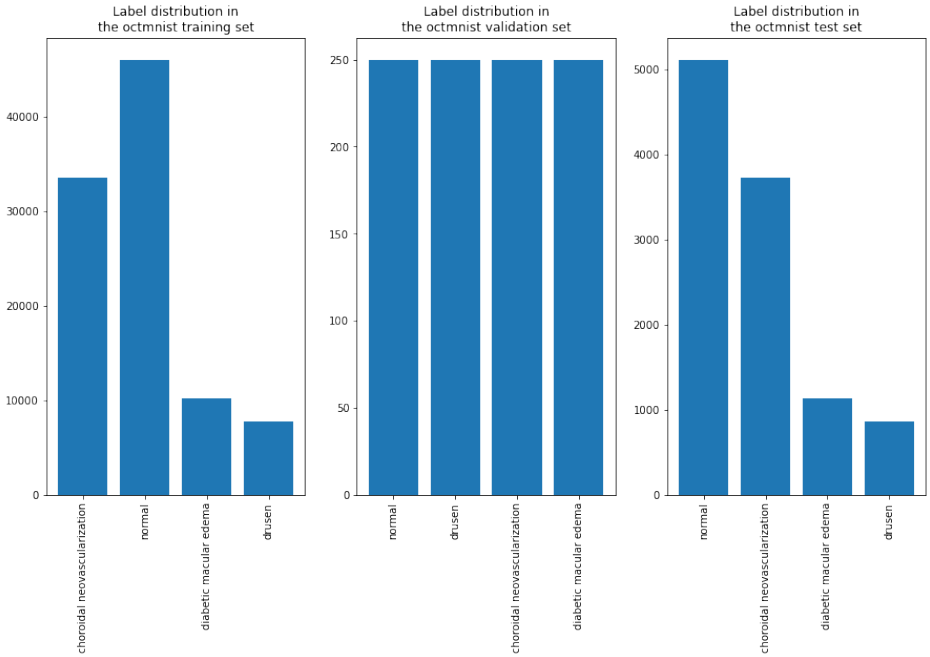

Fig. 12. Class distribution of the OctMNIST dataset.

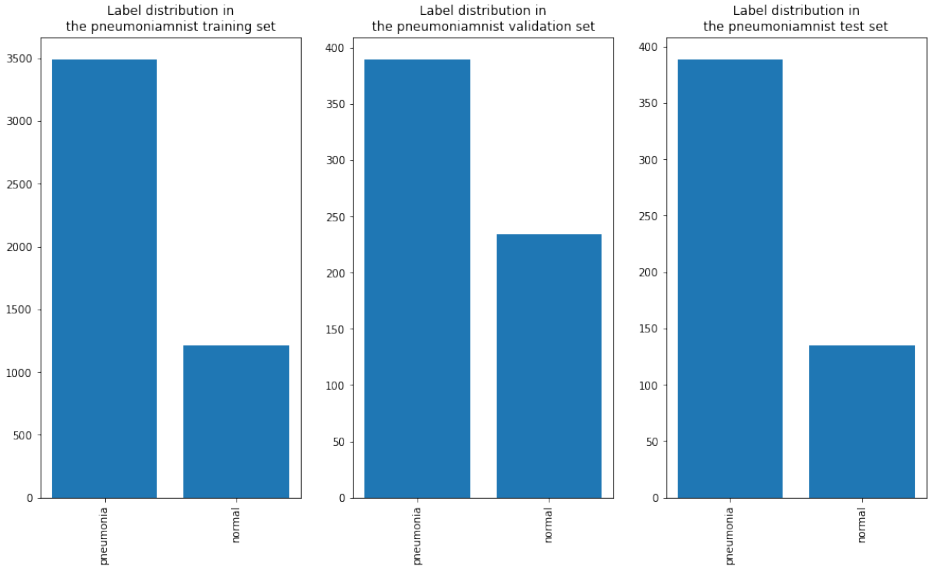

Fig. 13. Class distribution of the PneumoniaMNIST dataset.

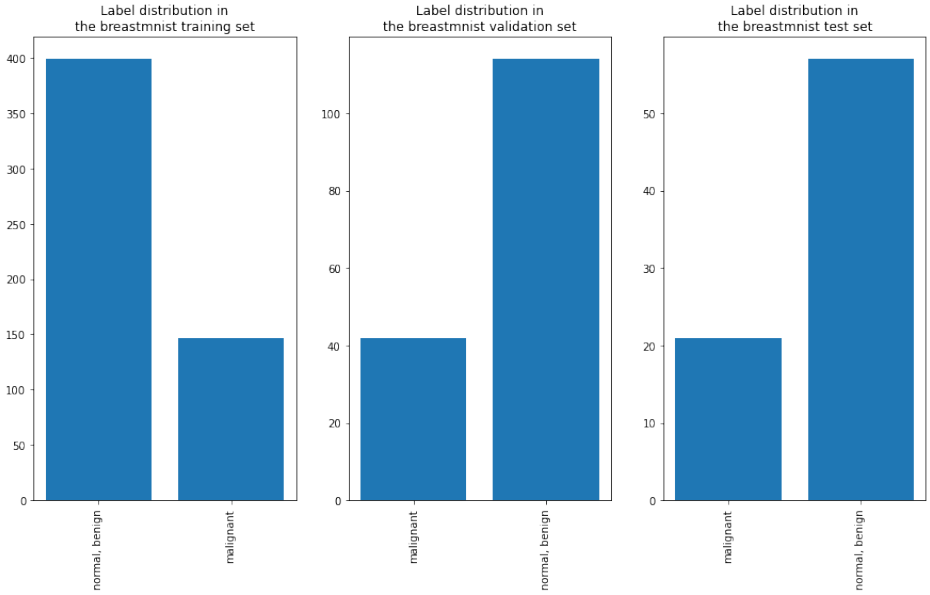

Fig. 14. Class distribution of the BreastMNIST dataset.

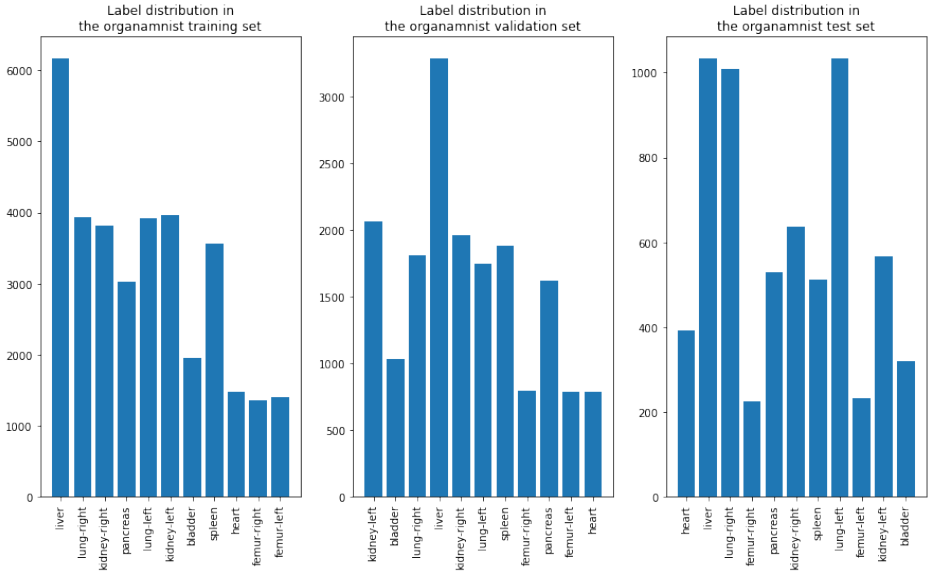

Fig. 15. Class distribution of the OrganMNIST_Axial dataset.

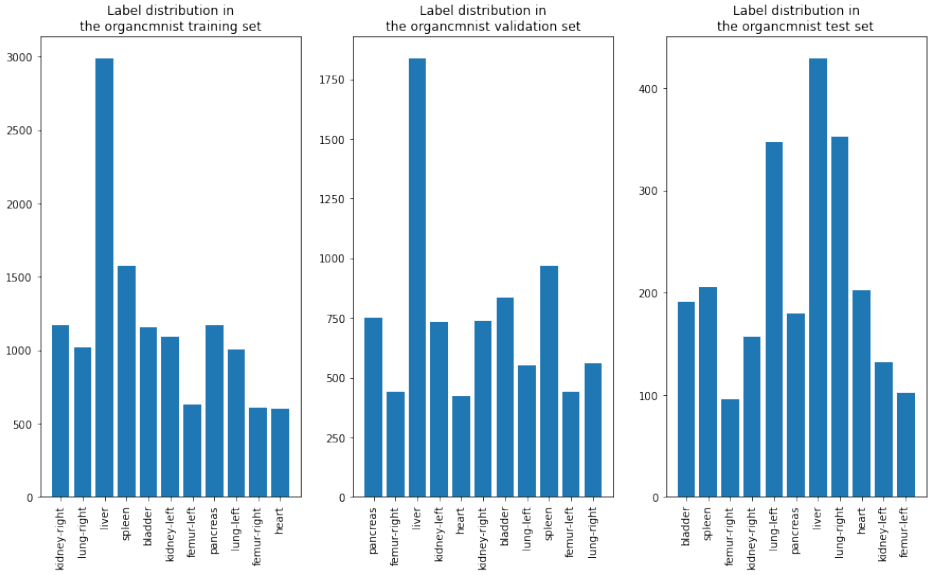

Fig. 16. Class distribution of the OrganMNIST_Coronal dataset.

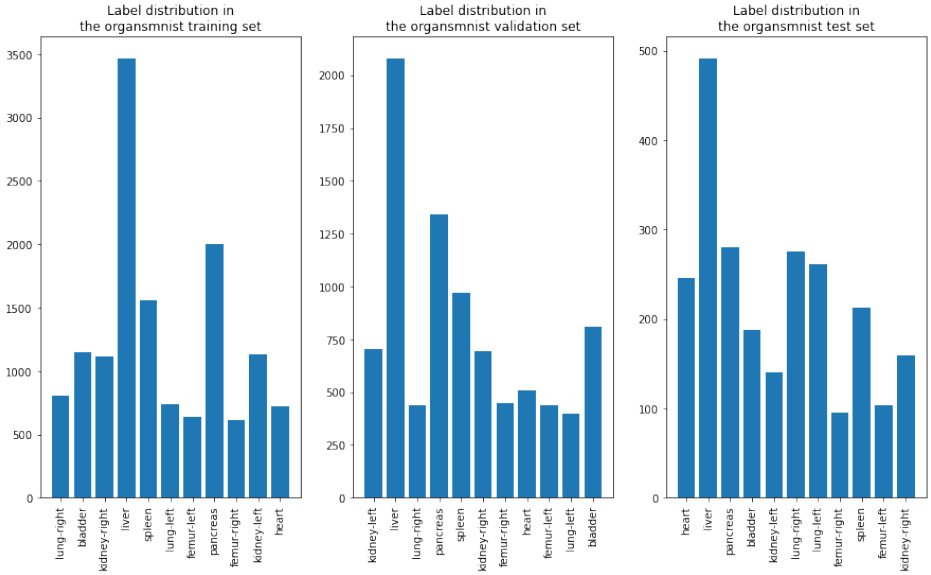

Fig. 17. Class distribution of the OrganMNIST_Sagittal dataset.

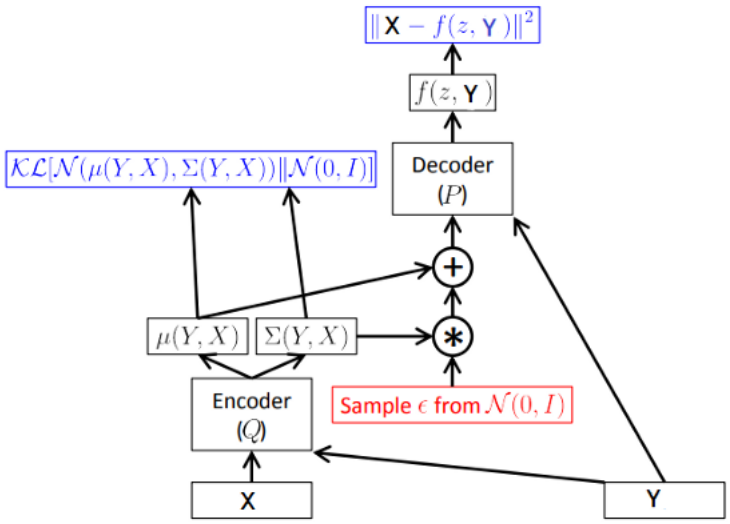

Fig. 18. Reproduction of the graphical representation of a Conditional Variational Auto-Encoder [2].

```
Decoder input layer:
 Linear(in_features=109, out_features=2048, bias=True)

Decoder hidden layers:
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1            [-1, 256, 4, 4]       1,179,904
       BatchNorm2d-2            [-1, 256, 4, 4]             512
         LeakyReLU-3            [-1, 256, 4, 4]               0
   ConvTranspose2d-4            [-1, 128, 8, 8]         295,040
       BatchNorm2d-5            [-1, 128, 8, 8]             256
         LeakyReLU-6            [-1, 128, 8, 8]               0
   ConvTranspose2d-7           [-1, 64, 16, 16]          73,792
       BatchNorm2d-8           [-1, 64, 16, 16]             128
         LeakyReLU-9           [-1, 64, 16, 16]               0
  ConvTranspose2d-10           [-1, 32, 32, 32]          18,464
      BatchNorm2d-11           [-1, 32, 32, 32]              64
        LeakyReLU-12           [-1, 32, 32, 32]               0
================================================================
Total params: 1,568,160
Trainable params: 1,568,160
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.41
Params size (MB): 5.98
Estimated Total Size (MB): 7.40
----------------------------------------------------------------

Decoder output layer:

----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1           [-1, 32, 61, 61]           9,248
       BatchNorm2d-2           [-1, 32, 61, 61]              64
         LeakyReLU-3           [-1, 32, 61, 61]               0
            Conv2d-4           [-1, 32, 30, 30]           9,248
       BatchNorm2d-5           [-1, 32, 30, 30]              64
         LeakyReLU-6           [-1, 32, 30, 30]               0
            Conv2d-7            [-1, 3, 28, 28]             867
              Tanh-8            [-1, 3, 28, 28]               0
================================================================
Total params: 19,491
Trainable params: 19,491
Non-trainable params: 0
```

Fig. 19. Decoder architecture of the Conditional VAE used with the PathMNIST dataset.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 14, 14]           1,184
       BatchNorm2d-2          [-1, 32, 14, 14]              64
         LeakyReLU-3          [-1, 32, 14, 14]               0
            Conv2d-4            [-1, 64, 7, 7]          18,496
       BatchNorm2d-5            [-1, 64, 7, 7]             128
         LeakyReLU-6            [-1, 64, 7, 7]               0
            Conv2d-7           [-1, 128, 4, 4]          73,856
       BatchNorm2d-8           [-1, 128, 4, 4]             256
         LeakyReLU-9           [-1, 128, 4, 4]               0
           Conv2d-10           [-1, 256, 2, 2]         295,168
      BatchNorm2d-11           [-1, 256, 2, 2]             512
        LeakyReLU-12           [-1, 256, 2, 2]               0
           Conv2d-13           [-1, 512, 1, 1]       1,180,160
      BatchNorm2d-14           [-1, 512, 1, 1]           1,024
        LeakyReLU-15           [-1, 512, 1, 1]               0
================================================================
Total params: 1,570,848
Trainable params: 1,570,848
Non-trainable params: 0
```

Fig. 20. Encoder architecture of the Conditional VAE used with the PathMNIST dataset.

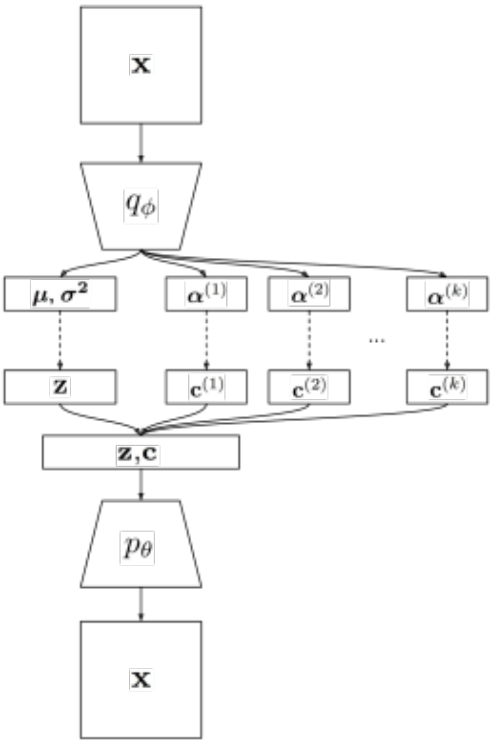

Fig. 21. Reproduction of the graphical representation of a Joint Variational Auto-Encoder [3].

```
Decoder input layer:
 Linear(in_features=110, out_features=2048, bias=True)

Decoder hidden layers:
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1            [-1, 256, 4, 4]       1,179,904
       BatchNorm2d-2            [-1, 256, 4, 4]             512
         LeakyReLU-3            [-1, 256, 4, 4]               0
   ConvTranspose2d-4            [-1, 128, 8, 8]         295,040
       BatchNorm2d-5            [-1, 128, 8, 8]             256
         LeakyReLU-6            [-1, 128, 8, 8]               0
   ConvTranspose2d-7           [-1, 64, 16, 16]          73,792
       BatchNorm2d-8           [-1, 64, 16, 16]             128
         LeakyReLU-9           [-1, 64, 16, 16]               0
  ConvTranspose2d-10           [-1, 32, 32, 32]          18,464
      BatchNorm2d-11           [-1, 32, 32, 32]              64
        LeakyReLU-12           [-1, 32, 32, 32]               0
================================================================
Total params: 1,568,160
Trainable params: 1,568,160
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.41
Params size (MB): 5.98
Estimated Total Size (MB): 7.40
----------------------------------------------------------------

Decoder output layer:

----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1           [-1, 32, 61, 61]           9,248
       BatchNorm2d-2           [-1, 32, 61, 61]              64
         LeakyReLU-3           [-1, 32, 61, 61]               0
            Conv2d-4           [-1, 32, 30, 30]           9,248
       BatchNorm2d-5           [-1, 32, 30, 30]              64
         LeakyReLU-6           [-1, 32, 30, 30]               0
            Conv2d-7            [-1, 3, 28, 28]             867
              Tanh-8            [-1, 3, 28, 28]               0
================================================================
Total params: 19,491
Trainable params: 19,491
Non-trainable params: 0
```

Fig. 22. Decoder architecture of the Joint VAE used with the PathMNIST dataset.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 14, 14]             896
       BatchNorm2d-2          [-1, 32, 14, 14]              64
         LeakyReLU-3          [-1, 32, 14, 14]               0
            Conv2d-4            [-1, 64, 7, 7]          18,496
       BatchNorm2d-5            [-1, 64, 7, 7]             128
         LeakyReLU-6            [-1, 64, 7, 7]               0
            Conv2d-7           [-1, 128, 4, 4]          73,856
       BatchNorm2d-8           [-1, 128, 4, 4]             256
         LeakyReLU-9           [-1, 128, 4, 4]               0
           Conv2d-10           [-1, 256, 2, 2]         295,168
      BatchNorm2d-11           [-1, 256, 2, 2]             512
        LeakyReLU-12           [-1, 256, 2, 2]               0
           Conv2d-13           [-1, 512, 1, 1]       1,180,160
      BatchNorm2d-14           [-1, 512, 1, 1]           1,024
        LeakyReLU-15           [-1, 512, 1, 1]               0
================================================================
Total params: 1,570,560
Trainable params: 1,570,560
Non-trainable params: 0
```

Fig. 23. Encoder architecture of the Joint VAE used with the PathMNIST dataset.

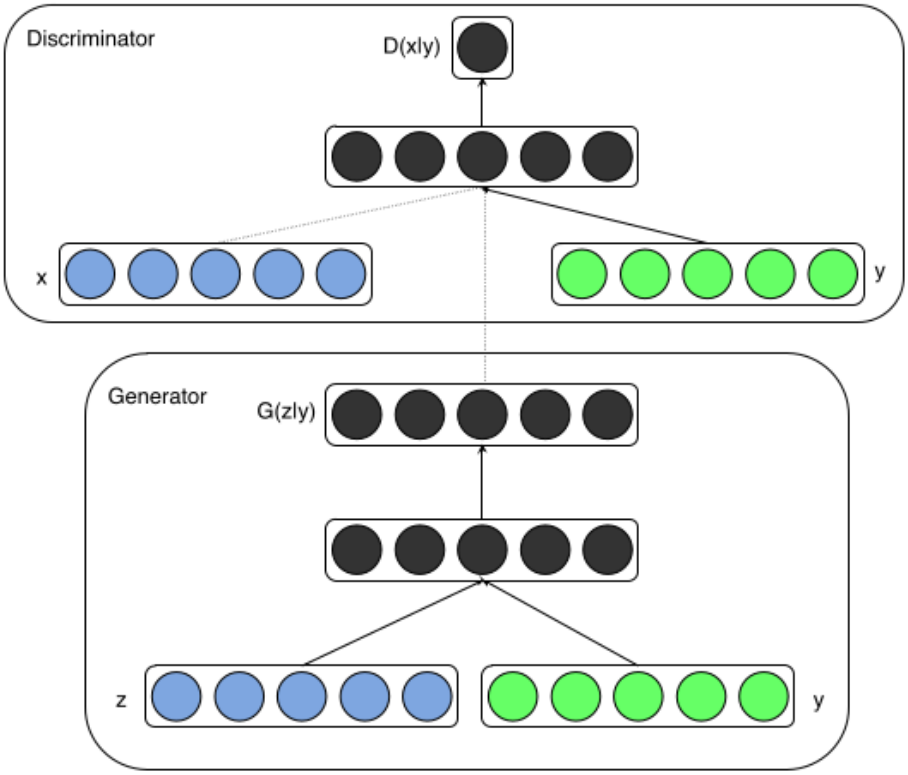

Fig. 24. Reproduction of the graphical representation of a Conditional Generative Adversarial Network [2].

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 32, 32]             896
         LeakyReLU-2          [-1, 32, 32, 32]               0
StandardDiscriminatorBlock-3          [-1, 32, 32, 32]
            Conv2d-4          [-1, 64, 16, 16]          32,832
         LeakyReLU-5          [-1, 64, 16, 16]               0
         LeakyReLU-6          [-1, 64, 16, 16]               0
         LeakyReLU-7          [-1, 64, 16, 16]               0
StandardDiscriminatorBlock-8          [-1, 64, 16, 16]
           Dropout-9          [-1, 64, 16, 16]               0
           Conv2d-10           [-1, 8, 16, 16]             512
           Conv2d-11           [-1, 8, 16, 16]             512
           Conv2d-12          [-1, 64, 16, 16]           4,096
          Softmax-13            [-1, 256, 256]               0
    SelfAttention-14          [-1, 64, 16, 16]               0
           Conv2d-15           [-1, 128, 8, 8]         131,200
        LeakyReLU-16           [-1, 128, 8, 8]               0
        LeakyReLU-17           [-1, 128, 8, 8]               0
        LeakyReLU-18           [-1, 128, 8, 8]               0
StandardDiscriminatorBlock-19           [-1, 128, 8, 8]
          Dropout-20           [-1, 128, 8, 8]               0
           Conv2d-21           [-1, 256, 4, 4]         524,544
        LeakyReLU-22           [-1, 256, 4, 4]               0
        LeakyReLU-23           [-1, 256, 4, 4]               0
        LeakyReLU-24           [-1, 256, 4, 4]               0
StandardDiscriminatorBlock-25           [-1, 256, 4, 4]
          Dropout-26           [-1, 256, 4, 4]               0
          Flatten-27                [-1, 4096]               0
           Linear-28                   [-1, 1]           4,097
        Embedding-29             [-1, 1, 4096]          36,864
           Linear-28                   [-1, 9]          36,873
================================================================
Total params: 772,426
Trainable params: 772,426
Non-trainable params: 0
```

Fig. 25. Discriminator architecture of the Conditional GAN† used with the PathMNIST dataset.

†To work with the TorchFusion library, the width and height of the datasets' images are upscaled from 28 to 32

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
   ConvTranspose2d-1            [-1, 256, 4, 4]         409,856
       BatchNorm2d-2            [-1, 256, 4, 4]               0
         Embedding-3               [-1, 1, 256]           2,304
         Embedding-4               [-1, 1, 256]           2,304
ConditionalBatchNorm2d-5            [-1, 256, 4, 4]
         LeakyReLU-6            [-1, 256, 4, 4]               0
         LeakyReLU-7            [-1, 256, 4, 4]               0
         LeakyReLU-8            [-1, 256, 4, 4]               0
StandardGeneratorBlock-9            [-1, 256, 4, 4]
  ConvTranspose2d-10            [-1, 128, 8, 8]         524,416
      BatchNorm2d-11            [-1, 128, 8, 8]               0
        Embedding-12               [-1, 1, 128]           1,152
        Embedding-13               [-1, 1, 128]           1,152
ConditionalBatchNorm2d-14            [-1, 128, 8, 8]
        LeakyReLU-15            [-1, 128, 8, 8]               0
        LeakyReLU-16            [-1, 128, 8, 8]               0
        LeakyReLU-17            [-1, 128, 8, 8]               0
StandardGeneratorBlock-18            [-1, 128, 8, 8]
          Dropout-19            [-1, 128, 8, 8]               0
  ConvTranspose2d-20            [-1, 64, 16, 16]        131,136
      BatchNorm2d-21            [-1, 64, 16, 16]              0
        Embedding-22               [-1, 1, 64]              576
        Embedding-23               [-1, 1, 64]              576
ConditionalBatchNorm2d-24            [-1, 64, 16, 16]
        LeakyReLU-25            [-1, 64, 16, 16]              0
        LeakyReLU-26            [-1, 64, 16, 16]              0
        LeakyReLU-27            [-1, 64, 16, 16]              0
StandardGeneratorBlock-28            [-1, 64, 16, 16]
          Dropout-29            [-1, 64, 16, 16]              0
  ConvTranspose2d-30             [-1, 3, 32, 32]           3,075
================================================================
Total params: 1,076,547
Trainable params: 1,076,547
Non-trainable params: 0
```

Fig. 26. Generator architecture of the Conditional GAN† used with the PathMNIST dataset.
†To work with the TorchFusion library, the width and height of the generated images are set to 32, and are subsequently downscaled to 28 to work with the baseline classifier

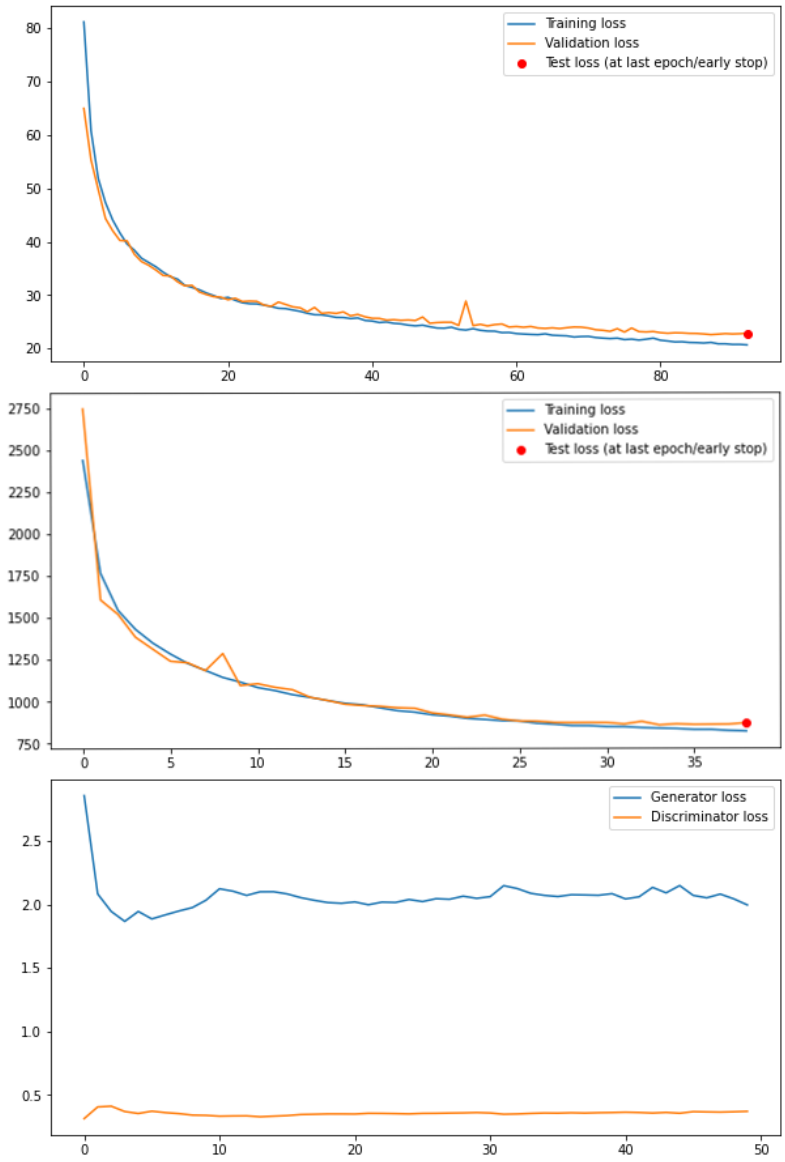

Fig. 27. Training loss of (in order) the conditional VAE, joint VAE, and conditional GAN on the PathMNIST dataset.

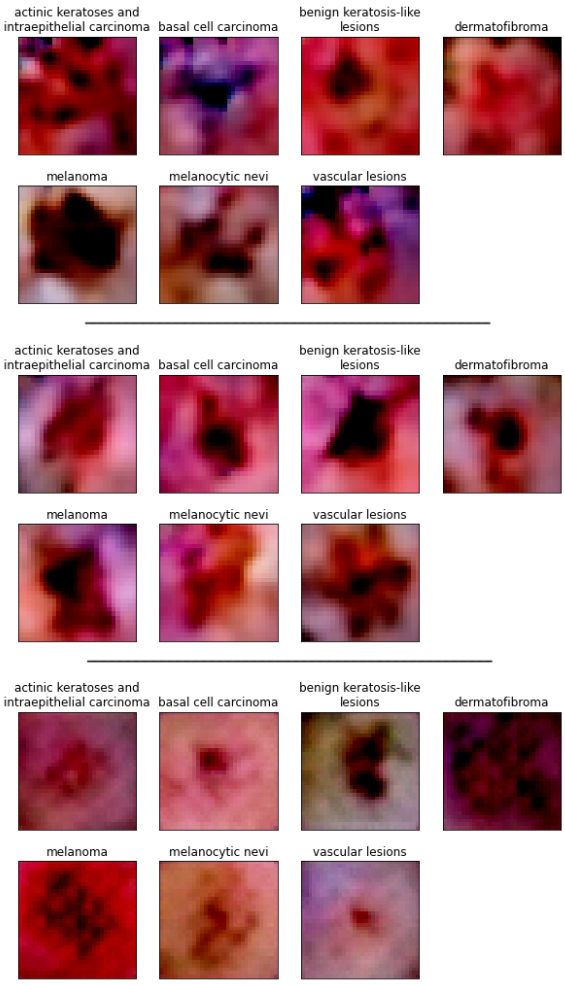

Fig. 28. Example of generated data for the DermaMNIST dataset for each data augmentation models in order: Conditional VAE, Joint VAE, Conditional GAN.

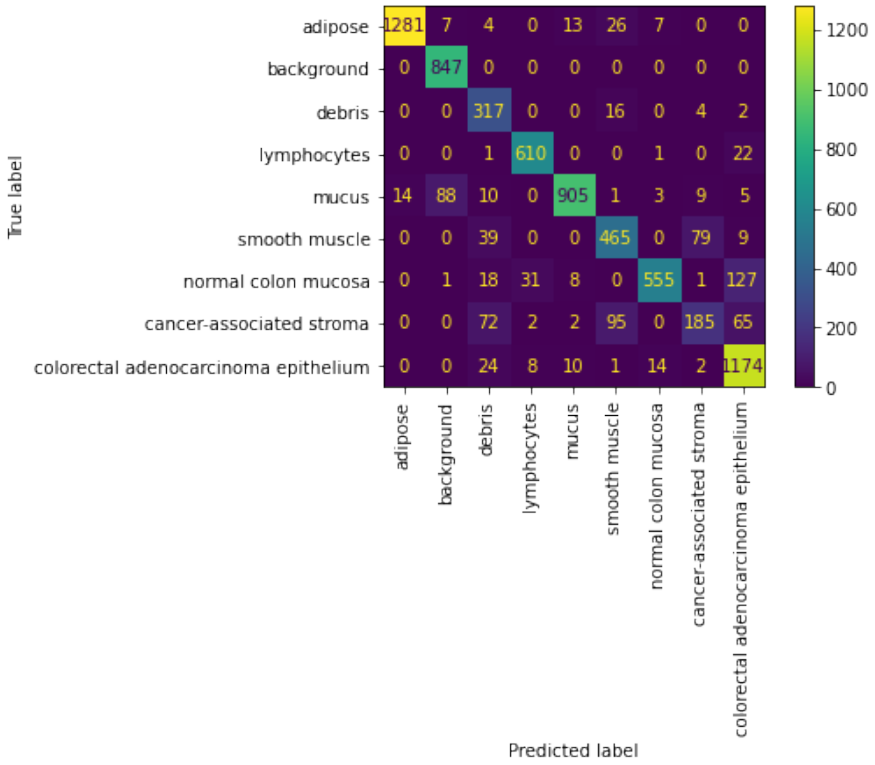

Fig. 29. Confusion Matrix for the best PathMNIST model (Weighted Random Sampling + Geometric Data Augmentation).

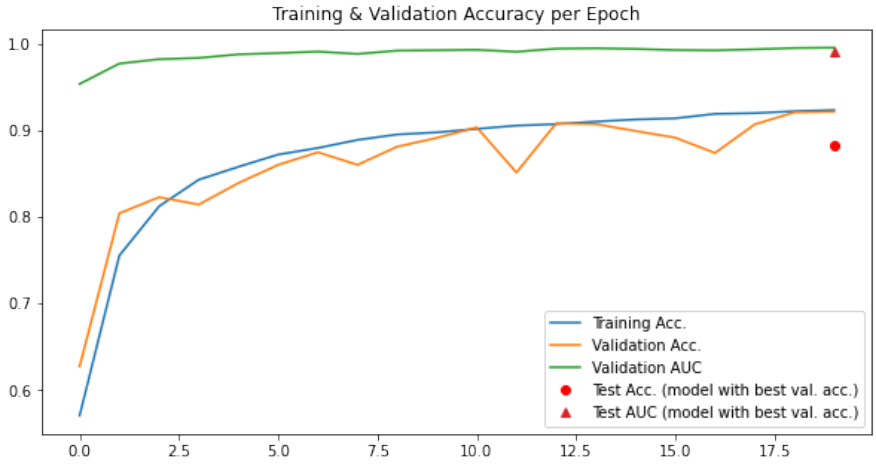

Fig. 30. Accuracy and AUC per epoch for the best PathMNIST model (Weighted Random Sampling + Geometric Data Augmentation).

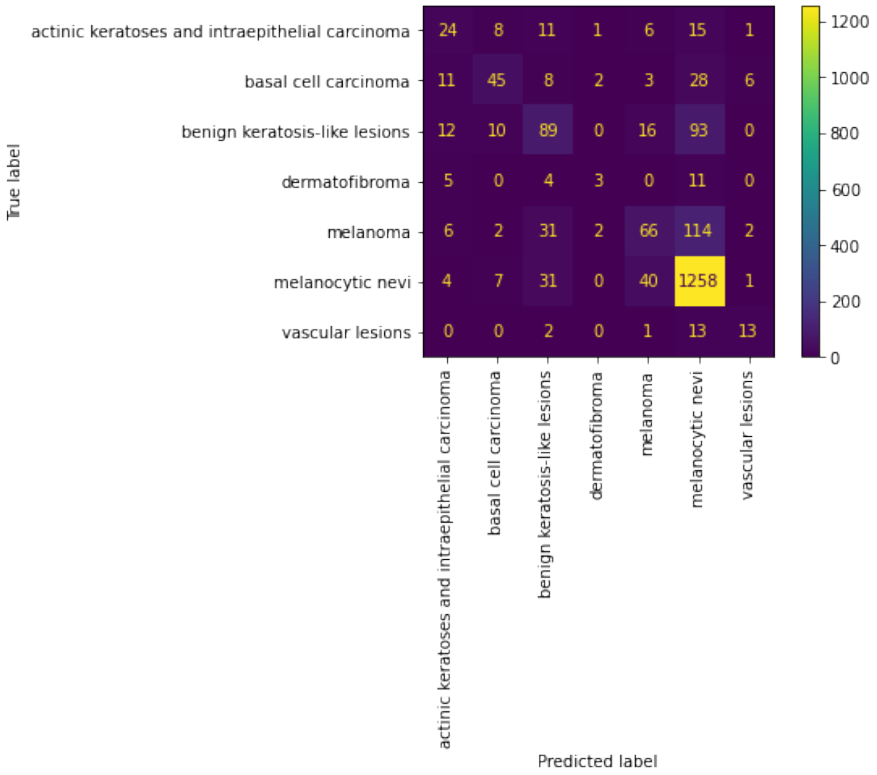

Fig. 31. Confusion Matrix for the best DermaMNIST model (Conditional VAE).

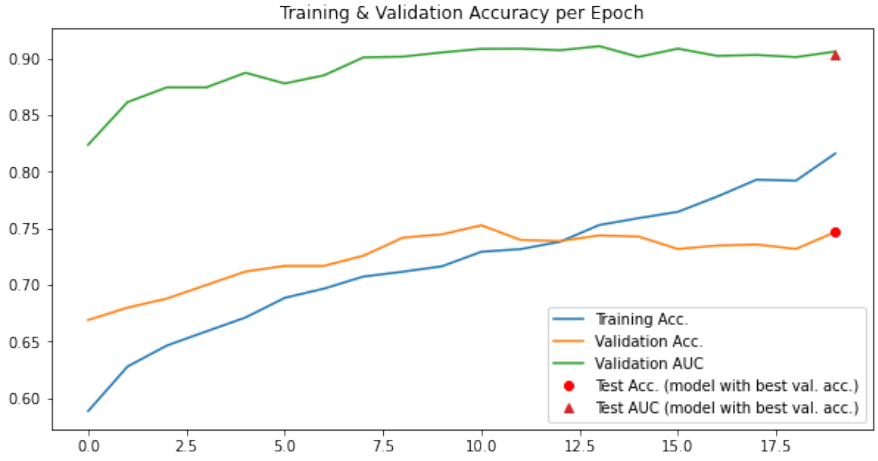

Fig. 32. Accuracy and AUC per epoch for the best DermaMNIST model (Conditional VAE).

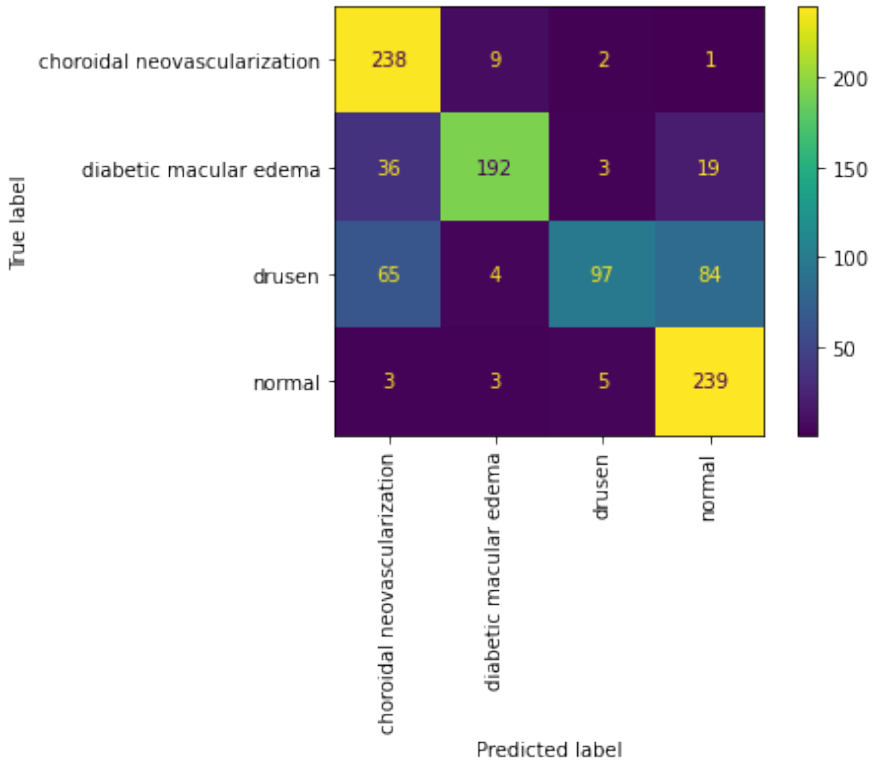

Fig. 33. Confusion Matrix for the best OctMNIST model (Weighted Random Sampling).

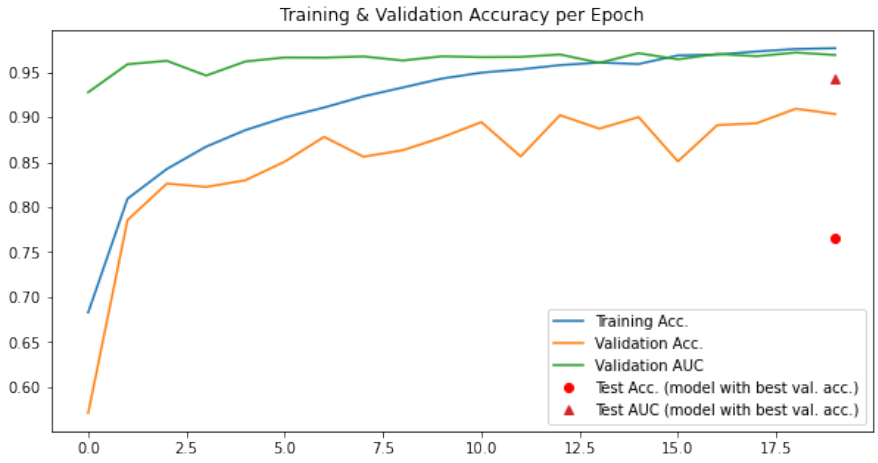

Fig. 34. Accuracy and AUC per epoch for the best OctMNIST model (Weighted Random Sampling).

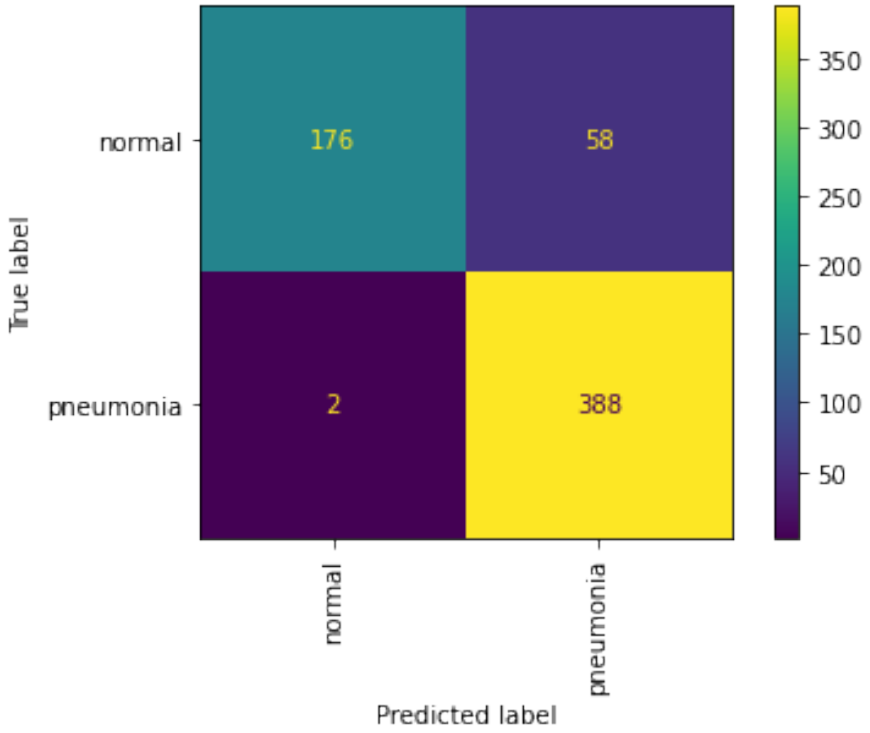

Fig. 35. Confusion Matrix for the best PneumoniaMNIST model (Weighted Random Sampling + geometric data augmentation).

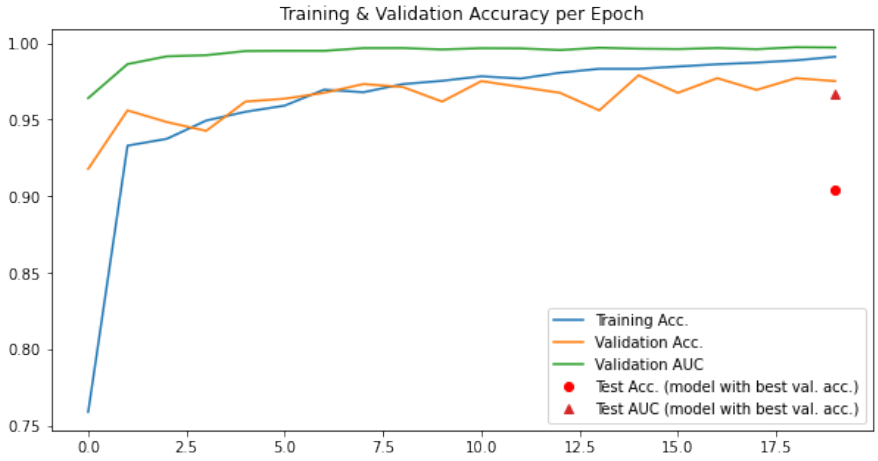

Fig. 36. Accuracy and AUC per epoch for the best PneumoniaMNIST model (Weighted Random Sampling + geometric data augmentation).

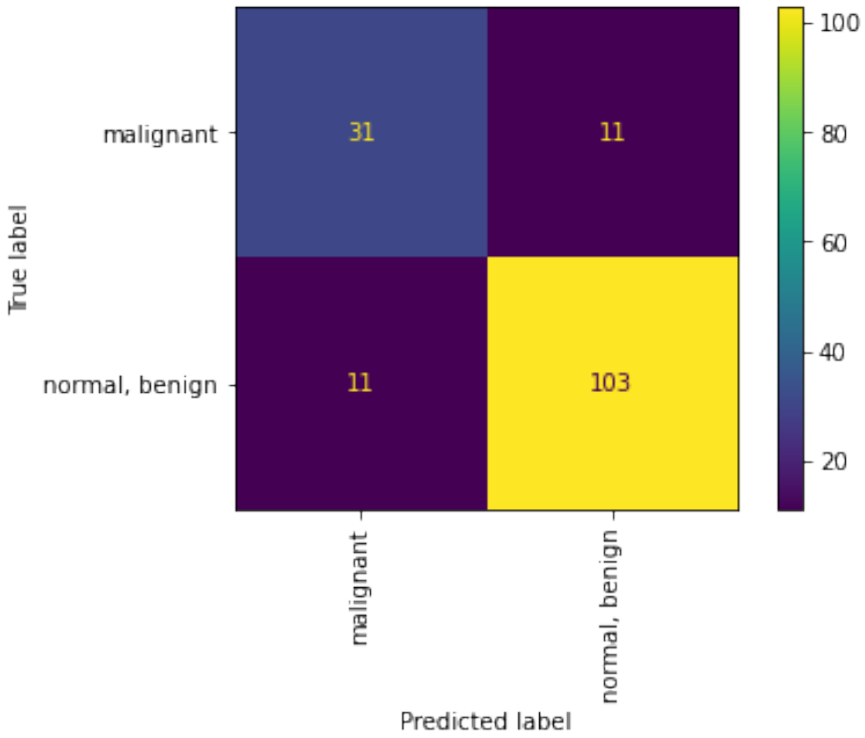

Fig. 37. Confusion Matrix for the best BreastMNIST model (Weighted Random Sampling + Conditional GAN).

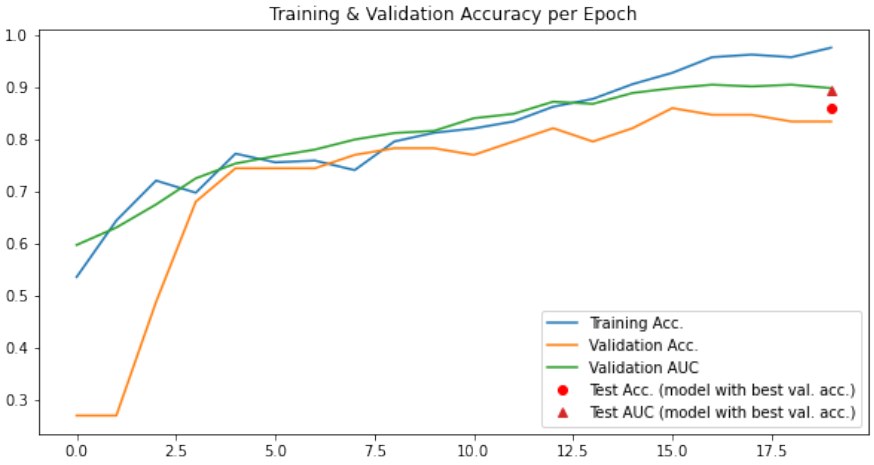

Fig. 38. Accuracy and AUC per epoch for the best BreastMNIST model (Weighted Random Sampling + Conditional GAN).

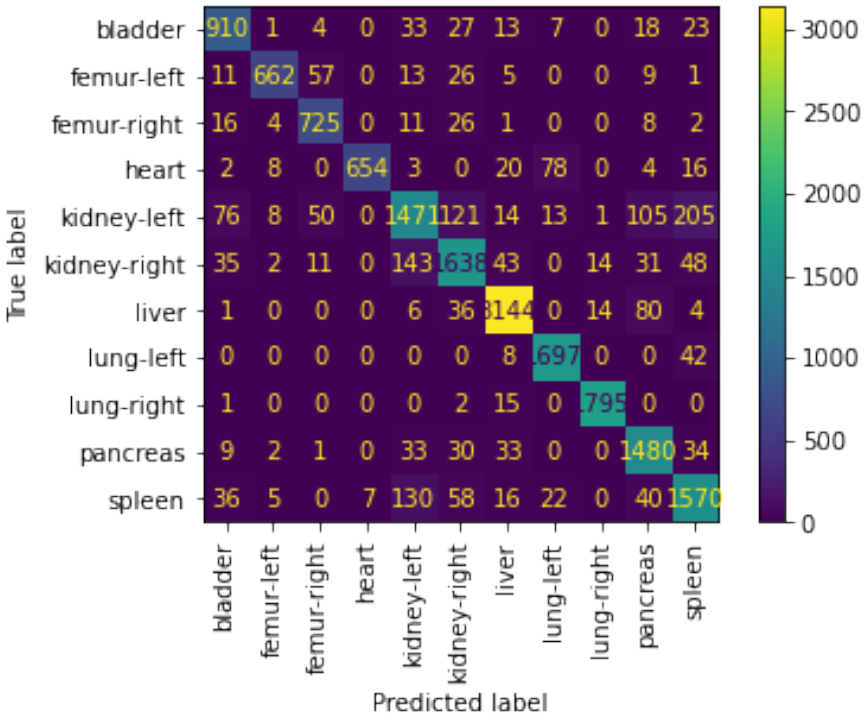

Fig. 39. Confusion Matrix for the best OrganAMNIST model (Weighted Random Sampling + Conditional VAE).

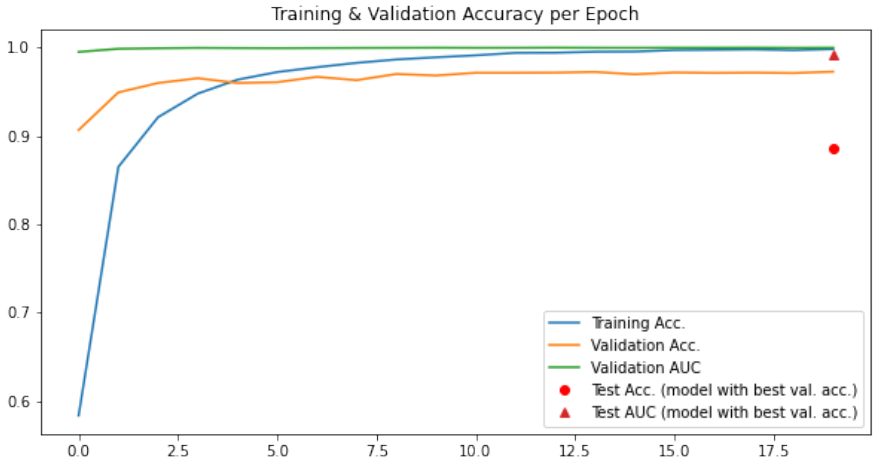

Fig. 40. Accuracy and AUC per epoch for the best OrganAMNIST model (Weighted Random Sampling + Conditional VAE).

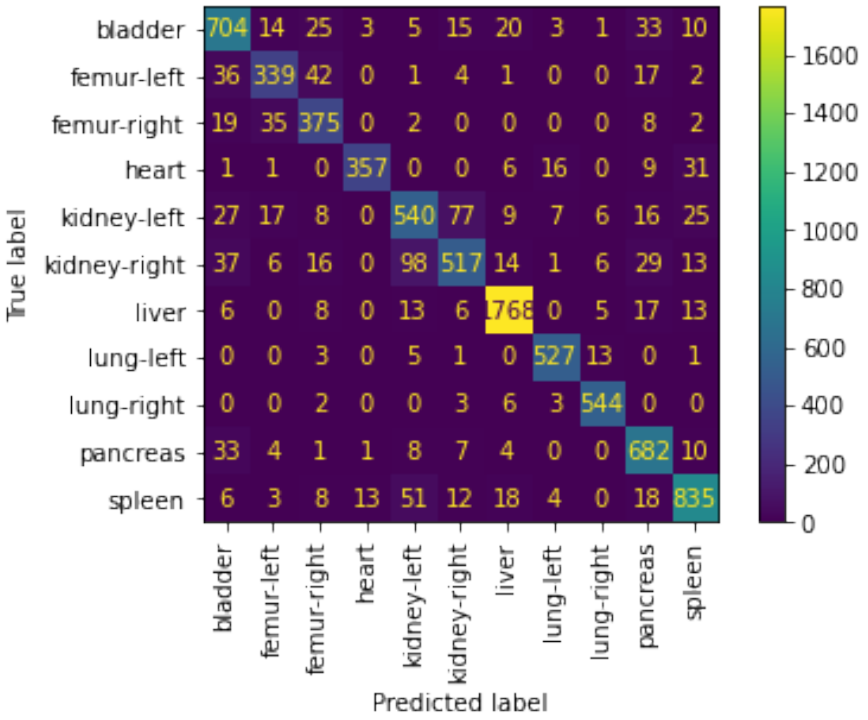

Fig. 41. Confusion Matrix for the best OrganCMNIST model (Weighted Random Sampling + Conditional GAN).

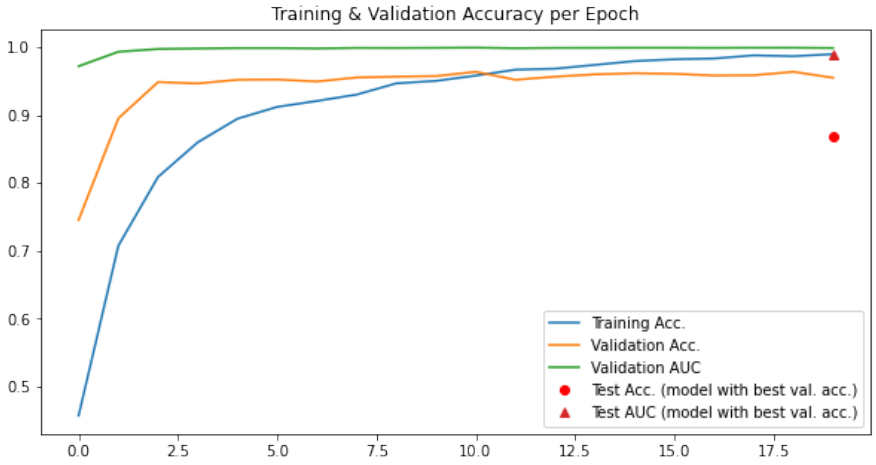

Fig. 42. Accuracy and AUC per epoch for the best OrganCMNIST model (Weighted Random Sampling + Conditional GAN).

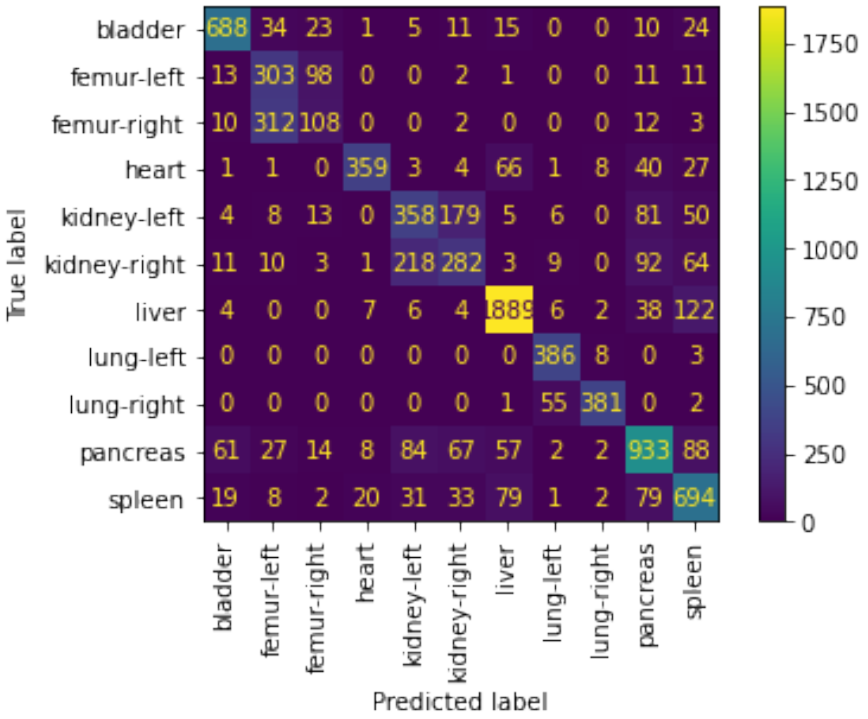

Fig. 43. Confusion Matrix for the best OrganSMNIST model (Conditional GAN).

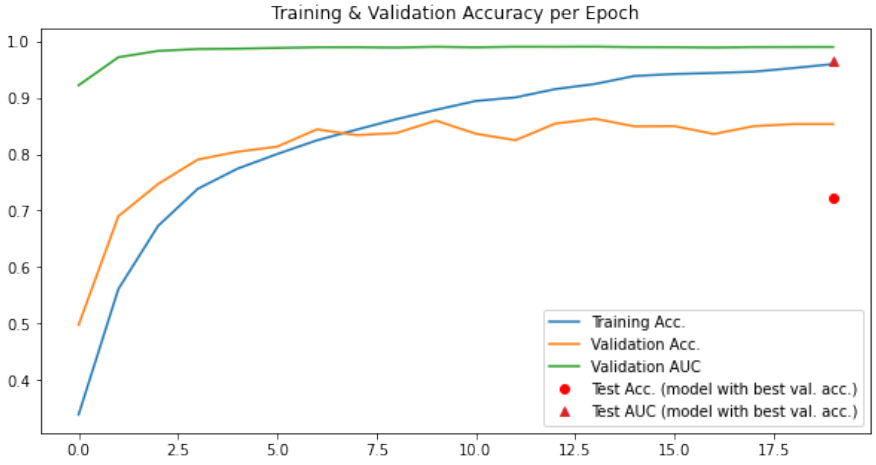

Fig. 44. Accuracy and AUC per epoch for the best OrganSMNIST model (Conditional GAN).