21.05.2018

Skin Lesion Analysis Towards Melanoma Detection

by Matthias Andreas Fontanellaz,

generative model group,

variational autoencoder model

1. Introduction

In the medical section deep learning algorithm are already widely used. They are used for segmentation tasks in order to generate bone or soft tissue models of the underlying MRI or CT-data[1], or they are used in classification tasks. Under the assumption, that enough training data is available, deep neural networks can at least reach human-expert levels in detecting certain anomalies[2], or they are used to support humans in their diagnostic routines[3]. Within the scope of this project we are training a classifier network to differentiate between three common skin lesions, seborrheic keratosis, melanomas and nevi. In order to improve the classification-networks generalization performance we will try to develop a method next to conventional jittering, in order to augment the relatively small dataset of 2000 labelled images (374 melanoma, 1372 nevus, 254 seborrheic keratosis) provided by the International Skin Imaging Collaboration (ISIC) as part of the ISIC 2017 challenge[4]. The project can be separated into two tasks. The first task is to find a method to generate artificially data, using a combination of a variational autoencoder[5] and a deep convolutional generative adversarial network[6] whereas the second task is to develop precise and reliable skin lesion classifiers in form of ResNets[7] and evaluate their performance on the augmented, jittered and the original datasets.

2. Image pre-processing

In order to use the raw data provided by ISIC the images had to be aligned and scaled to the same resolution of 256 x 256 pixels each. Additionally to RGB images and labels, masks for an additional segmentation task were provided by ISIC which have been used to extract the skin lesion on each image and to align them. The results of this pre-processing is shown in Figure 1.

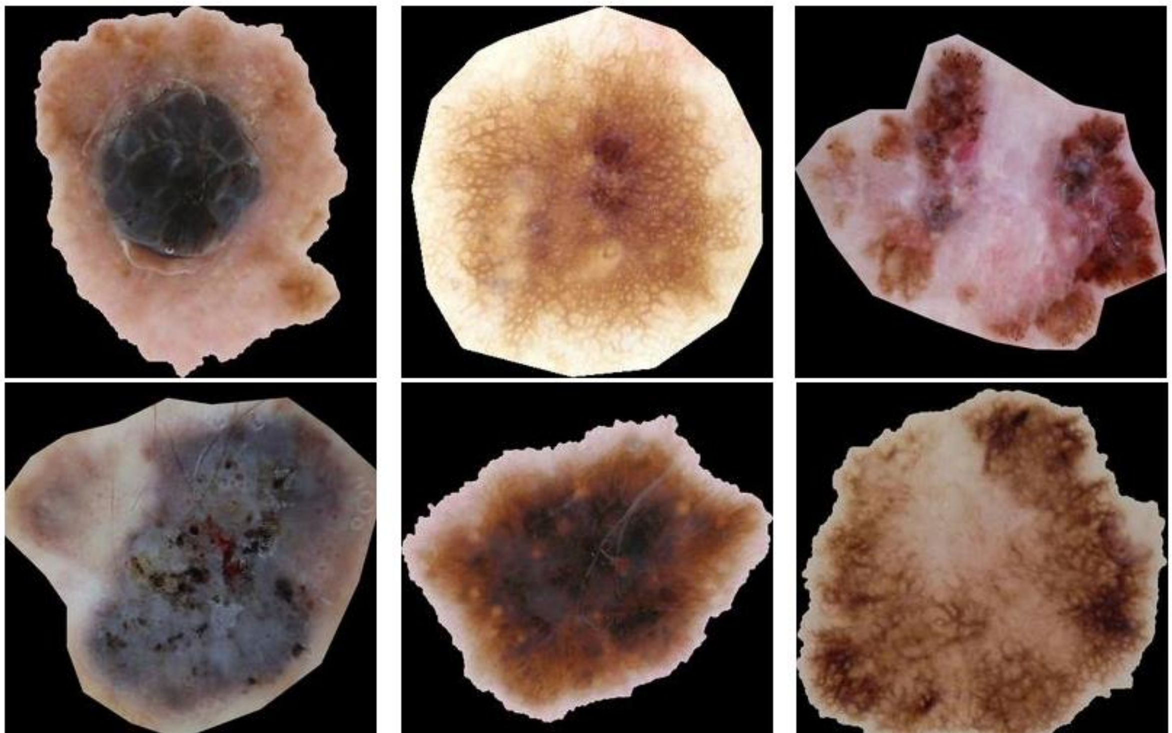

Figure 1 Pre-processed skin lesion images. Left: melanoma, middle: nevi, right: seborrheic keratosis

3. Dataset augmentation task

The most successful neural nets are trained with labelled data. Especially in the medical field labelling data can become very expensive. Therefore we are looking for a method to artificially generate data with a model and using a refinement afterwards in order to make the model images look more real. When using just generated data from a model, the network may learn to overfit on details only present in the synthetic images and failing in generalization on real images. The idea behind this technique came from Apples research on "Improving the Realism of Synthetic Images"[8]. We will try to develop our own models to generate skin lesion images in form of variational autoencoders and we will use a second network in this dataset-augmentation pipeline to improve the realism of these rough skin lesion images created by the variational autoencoder model. The refinement network will be a DCGAN (Deep Convolutional Generative Adversarial Network) where the generator network will be built as autoencoder in order to add a reconstruction loss to the DCGAN loss. The mentioned pipeline is graphically displayed in Figure 2.

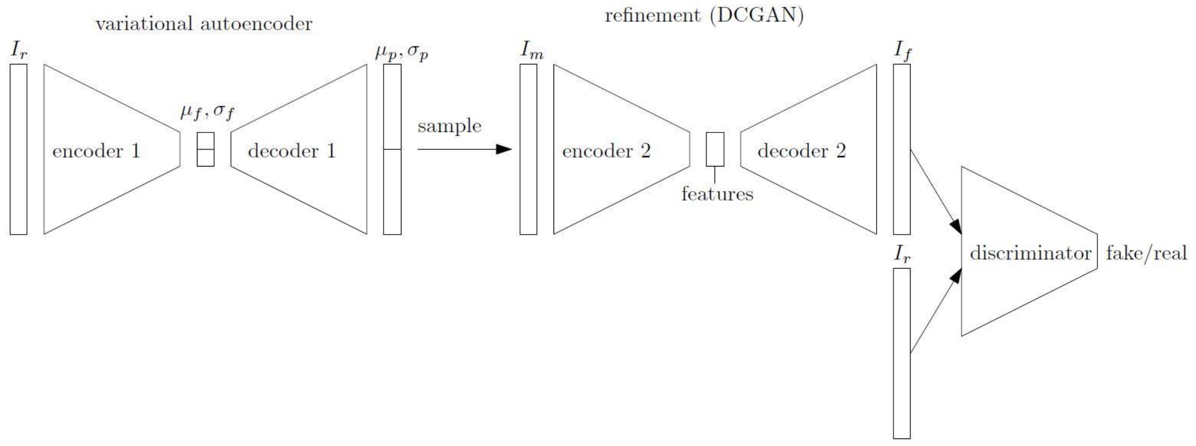

Figure 2 Schematically description of the data augmentation pipeline I_r are real images, I_m are modelled images, , I_f are fake images, μ_f feature mean, σ_f feature standard deviation , μ_p pixel mean, σ_p pixel standard deviation

3.1. Variational Autoencoder as model and theoretical background

The variational autoencoder is first introduced in 2013 by Diederik P. Kingma and Max Welling [5]. The base idea behind the variational autoencoder is to learn and represent the data generating distribution in its latent space. In order to achieve such a representation the normal autoencoder has to be modified.

To create only images belonging to one class of skin lesions (seborrheic keratosis, melanomas or nevi) we train three different autoencoder networks to generate the raw images.

The goal of the variational autoencoder is to learn the data generating distribution in order to generate new samples.

$$P(X) = \int P(X,z)dz = \int P(X|z)P(z)dz \tag{1}$$

In order to find $P(z)$, $P(z|X)$, the posterior probability given our input data can be used to infer the unknow distribution of the latent variables ($P(z)$). Using a method called variational inference the posterior $P(z|X)$ can be modelled by a simpler distribution $Q(z|X)$. By minimizing the difference between these two distributions using the Kullback-Leibler divergence as measurement the result will be a useful approximation[9].

$$D_{KL}[\,Q(z|X)||P(z|X)] = \sum_z Q(z|X) * log\left(\frac{Q(z|X)}{P(z|X)}\right) = E_{z\sim Q(z|X)}[log\left(\frac{Q(z|X)}{P(z|X)}\right) = E_{z\sim Q(z|X)}[\log(Q(z|X)) - \log(P(z|X))] \tag{2}$$

By applying Bayes rule and move $P(X)$ outside of the expectation since it does not depend on z, Equation 2 can be rewritten as

$$D_{KL}[\, Q(z|X)||P(z|X)] = E_{z\sim Q(z|X)}\left[\log\big(Q(z|X)\big) - \log\left(\frac{P(X|z)*P(z)}{P(X)}\right)\right] = \\ E_{z\sim Q(z|X)}\left[\log\big(Q(z|X)\big) - \log(P(X|z)) - \log\big(P(z)\big) + \log(P(X))\right] = \\ E_{z\sim Q(z|X)}\left[\log\big(Q(z|X)\big) - \log(P(X|z)) - \log\big(P(z)\big)\right] + \log(P(X)) \tag{3}$$

Rearranging leads to

$$\log\big(P(X)\big) - D_{KL}[\, Q(z|X)||P(z|X)] = E_{z\sim Q(z|X)}[\log(P(X|z)) - (\log\big(Q(z|X)\big) - \\ \log\big(P(z)\big))] = E_{z\sim Q(z|X)}\,[\log(P(X|z))] - E_{z\sim Q(z|X)}[\log\big(Q(z|X)\big) - \log\big(P(z)\big)] = \\ E_{z\sim Q(z|X)}\,[\log(P(X|z))] - D_{KL}[\, Q(z|X)||P(z)] \tag{4}$$

Leading to the variational autoencoders objective function

$$\log\big(P(X)\big) - D_{KL}[\, Q(z|X)||P(z|X)] = E_{z\sim Q(z|X)}\,[\log(P(X|z))] - D_{KL}[\, Q(z|X)||P(z)] \tag{5}$$

The left hand side contains the data generating distribution we want to model and a certain error. The variational autoencoder is therefore trying to find the lower bound ($\log\big(P(X)\big) - D_{KL}[\, Q(z|X)||P(z|X)]$) on $\log P(X)$. Since $P(z|X)$ cannot be calculated analytically the right hand side of equation 5 is evaluated.
If we constrain $P(z)\sim N(z|0,1)$ to obtain $Q(z|X)\sim N(\mu(X;\Phi),\sigma(X;\Phi)^2)$ similar to a unit gaussian and Φ as encoder parameters, $\mu(X;\Phi)$ and $\sigma(X;\Phi)^2$ as encoder output we can rewrite the KL-divergence

$$D_{KL}[\, Q(z|X)||P(z)] = -\tfrac{1}{2}\textstyle\sum_k\big(1 + \log(\sigma(X;\Phi)^2) - \mu(X;\Phi)^2 - e^{\log(\sigma(X;\Phi)^2)}\big) \tag{6}$$

With k = number of latent variables

Equation 6 can be interpreted as penalty for the encoder network to produce latent variables, distributed as multivariate gaussian. The reconstruction loss of the decoders output $P(X|z;\theta)$ with θ as decoder parameters, can be evaluated as maximum likelihood estimation $E_{z\sim Q(z|X)}\,[\log(P(X|z))]$.

3.1.1. Applied objective function implemented as torch autograd function

A variational autoencoder can similarly be trained like a conventional autoencoder. As reconstruction loss a custom cost function has been implemented to provide a gaussian output of the decoder network rather than simply use a mean square error loss. Having the output represented as mean and standard deviation for each pixel gives the opportunity to sample from this output distribution in order to generate more diverse data if needed. For calculating the reconstruction loss of the autoencoder a gaussian criterion is used .
It has to be mentioned that the neither the encoder nor the decoder do output $\mu(X;\Phi), \mu_{out}(z;\theta)$ and $\sigma(X;\Phi)^2, \sigma_{out}(z;\theta)^2$ directly as described in the theory section above, but $\mu(X;\Phi)$, $\mu_{out}(z;\theta)$ and $\log(\sigma(X;\Phi)^2)$, $\log(\sigma_{out}(z;\theta)^2)$.

$$P(X|z;\theta) = P(X|\mu_{out}(z;\theta),\sigma_{out}(z;\theta)) = \frac{1}{\sqrt{2*\pi*\sigma_{out}^2}} * e^{\frac{(y-\mu_{out})^2}{2*\sigma_{out}^2}} \tag{7}$$

With $y = Target\ pixel\ values$. By taking the negative log we obtain

$$-\log(P(X|\mu_{out}(z;\theta),\sigma_{out}(z;\theta))) = \frac{1}{2}\log(2\pi) + \frac{1}{2}\log(\sigma_{out}^2) + \frac{(y-\mu_{out})^2}{2\sigma_{out}^2} \quad (8)$$

The gradients form the two output layer $\mu_{out}(z;\theta)$ *and* $\log\left(\left(\sigma_{out}(z;\theta)\right)^2\right)$ then can be calculated as follows.

$$\frac{\partial}{\partial\mu_{out}}\left(\frac{1}{2}\log(2\pi) + \frac{1}{2}\log(\sigma_{out}^2) + \frac{(y-\mu_{out})^2}{2\sigma_{out}^2}\right) = -(y-\mu_{out}) * e^{-log(\sigma_{out}^2)} \quad (9)$$

$$\frac{\partial}{\partial\log(\sigma_{out}^2)}\left(\frac{1}{2}\log(2\pi) + \frac{1}{2}\log(\sigma_{out}^2) + \frac{(y-\mu_{out})^2}{2\sigma_{out}^2}\right) = \frac{\partial}{\partial\log(\sigma_{out}^2)}\left(\frac{1}{2}\log(2\pi) + \frac{1}{2}\log(\sigma_{out}^2) + \frac{1}{2} * (y-\mu_{out})^2 * e^{-log(\sigma_{out}^2)}\right) = \frac{1}{2} - \frac{1}{2} * \frac{(y-\mu_{out})^2}{\sigma_{out}^2} \quad (10)$$

The property of this gaussian output compared to a mean square error on just one output layer is that the loss will be asymptotically to negative infinity and not to zero.

3.1.2. Reparameterization trick implementation

For the gradients to be tractable we cannot just sample from the encoders output $N(\mu(X;\Phi),\sigma(X;\Phi))$. Using a reparameterization gradients w.r.t to all the networks parameter can be evaluated[5].

Instead of $z \sim N(\mu(X;\Phi),\sigma(X;\Phi)^2)$ we introduce an additional random variable $\varepsilon \sim N(0,1)$. The latent variables z can then be obtained as follows

$$z = \mu(X;\Phi) + \varepsilon * e^{0.5*\log(\sigma(X;\Phi)^2)} \quad (11)$$

Where $\mu(X;\Phi)$ and $\log(\sigma(X;\Phi)^2)$ are the encoders output. In contrast to the gaussian objective function Equation 11 uses PyTorchs variables in order to calculate the gradients automatically.

3.1.3. Architecture of the VE model

The encoder and the decoder are build up inversely (Figure 3) with some minor differences. The encoders architecture is as follows: Seven convolutional layers using all a leaky ReLUs with a negative slope of 0.2 are followed by the fully connected μ_f and σ_f layers. The first convolutional layer unit differs from the other since it does not down sample the images (stride is set to one), uses e kernel of size 3 x 3 and no batch normalization layer is added. All the following units are build out of convolutional layers with stride set to two, kernel size to 4 x 4 and padding to one with leaky ReLU-activation followed by batch normalization layer. Using this architecture the 768 feature maps of the last convolutional layer have a height and width of just 4 pixels each. The output layers of the encoder network (μ_f and σ_f) own 64 units each. During training the input to the decoder part is sampled as described in 3.1.2 resulting in a 64 dimensional densely connected input layer. This layer then is connected to a mirrored structure of the encoder network. In the decoder standard ReLU activations instead of the leaky ReLU are used. The 7th convolutional unit again differs from the previous ones and is a mirroring of the encoders first convolutional unit with kernel size of 3 x 3 and stride of one, but no nonlinear activation. In order to model an output distribution, two of these units are generate in parallel leading to the μ_p and σ_p output of dimension 3X256x256 each, so that every reconstructed pixel is represented as its mean value and standard deviation.

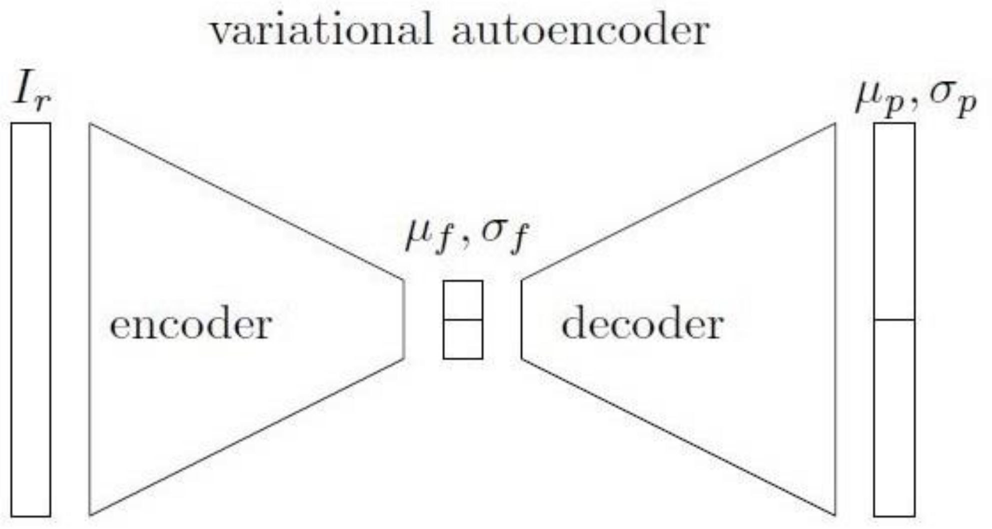

Figure 3 Variational autoencoder architecture. I_r are real images, μ_f feature mean, σ_f feature standard deviation , μ_p pixel mean, σ_p pixel standard deviation

3.1.4. Training

As described in the theory section to the variational autoencoder, the network is trained by optimizing the right hand side of Equation 5 where $E_{z\sim Q(z|X)}\,[\log(P(X|z))]$ is performed by the customized autograd function introduced in 3.1.1. For the optimisation an Adam optimizer with default β-coefficients ($\beta_1 = 0.9,\ \beta_2 = 0.999$) and a learning rate of 10^{-4} has been used. No additional weigh initialization strategy has been considered. Further a simple early stopping has been implemented for additional regularization next to the Kullback-Leibler-Penalty.

3.1.5. Generating new data

Since the trained variational autoencoder will be used as model to generate new data for the refinement via a generative adversarial neural network, we now look at how this new data is generated.

The Kullback-Leibler divergence term in the objective function (Equation 5) forces the encoder to represent the learned data in a multivariate gaussian. To generate new data the latent variables z as input for the decoder are just sampled from a unit gaussian. The decoders output is then $P(X|\mu_{out}(z\sim N(0,1);\theta), \sigma_{out}(z\sim N(0,1);\theta))$. In Figure 4 two generated examples for each class are displayed.

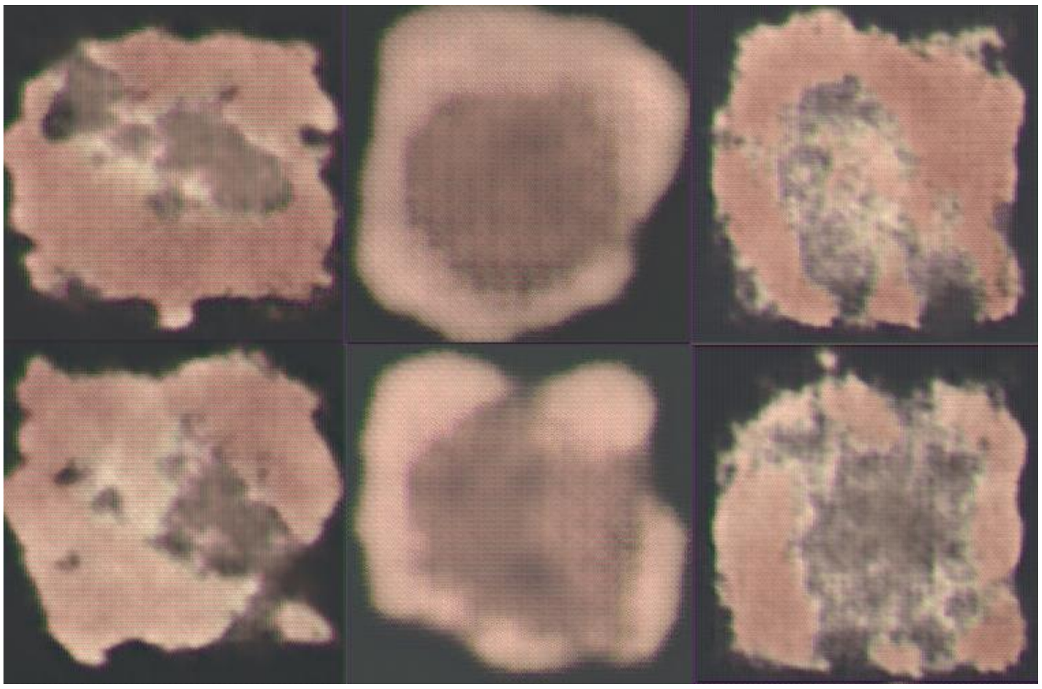
Figure 4 Generated data, left: generate Melanoma, middle: generate nevi, right: generated seborrheic keratosis

It has to be mentioned that the variance in the obtained images was enough so that sampling from the output distribution has not been used for the evaluation of the ResNet classifier.

3.2. Deep Convolutional Generative Adversarial Network (DCGAN) for refinement

Like the GAN introduced by Goodfellow[10] the DCGAN[6] is a composition of two different networks, the generator network in form of a standard autoencoder and the discriminator network as shown in Figure 5.

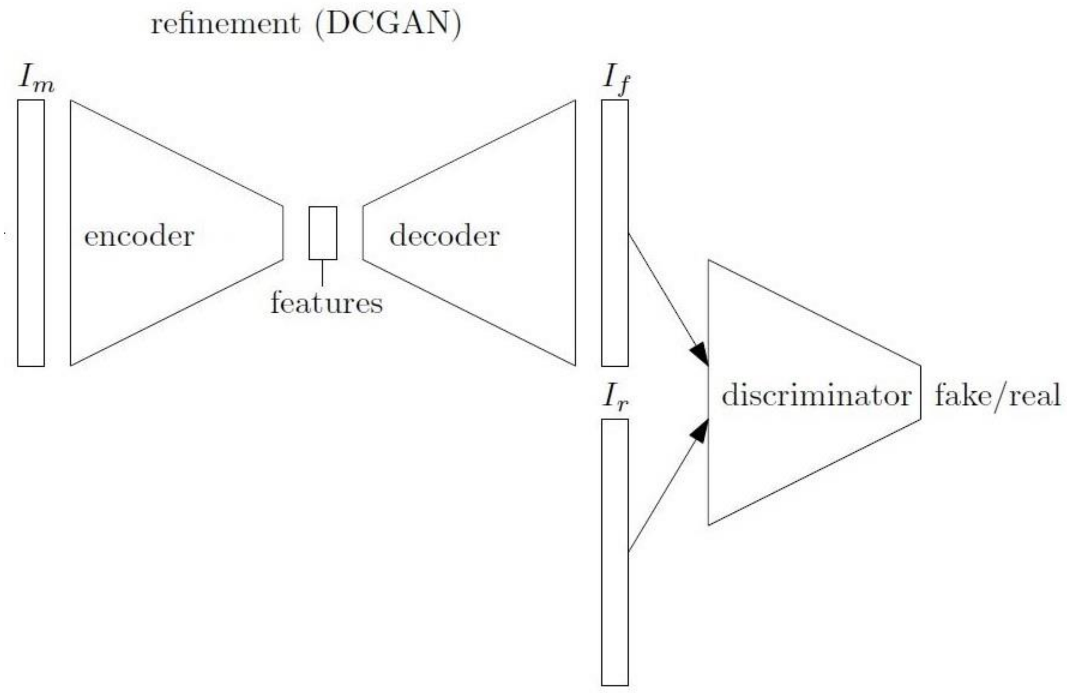

Figure 5 DCGAN composition. I_r are real images, I_m are modelled images, , I_f are fake images

Usually for DCGAN the generator would only be the decoder part from Figure 5, however this GAN's task is not to generate fake images from gaussian noise but refine model images I_m generated by the VA model. Therefore we use an encoder to generate the feature vector, based on the model data. The decoder then has to learn to reconstruct the model images more realistic than the VA on its own would be able to. In order to fulfil this task the generator and the discriminator have to compete against each other. Through training the generator will try to reconstruct the model images as realistic as possible in order to make the discriminator unable to predict the fake images I_f as such, whereas the discriminators goal is to correctly predict the true origin of his input images.
The main difficulty in training such a system is to finetune hyperparameters so that the generator and the discriminator converge simultaneously. When not finding this balance, it is usual that the discriminator converges very fast to its optimum causing a vanishing gradient, leading to stagnant in the generators optimization procedure.
Mathematically the adversarial game can be described with following equation.

$$\min_G \max_D V(D,G) = \frac{1}{2} E_{x \sim p_{data}}\left[\log\left(D(x)\right)\right] + \frac{1}{2} E_{z \sim p_z(z)}\left[\log\left(1 - D\left(G(Z)\right)\right)\right] \quad (12)$$

3.2.1. Adversarial learning technique

In order to train this ensemble an Adam optimizer was used with initial parameter $\beta_1 = 0.5$ and $\beta_2 = 0.999$ with a learning rate of 10^{-4}. The weights of all subsystems of the implemented DCGAN are initialized with a sampling from a gaussian distribution with zero mean and variance of 0.04. Further pretraining the generator-autoencoder on real images for 8000 epochs as a specialized weight initialization is expected to be a valuable strategy in respect to prevent too fast convergence of the discriminator during adversarial training. The training procedure for the generator and discriminator are described in detail below. The overall training of the DCGAN for each epoch is composed of training the generator four times while the discriminator is only trained once. With this sanction neither the generator nor the discriminator converged to fast, leading to a successful refinement of the model images, whereby the quality of these images has to be evaluated visually and not via the loss values. The results are presented in 3.2.2 Results of the refinement.

3.2.1.1. Discriminator training

Before training the whole composition of generator-autoencoder and discriminator the latter of whom has to be trained first on real and fake images. The learning strategy for the discriminator is schematically displayed in Figure 6. The training is a compilation of epochs in which the discriminator has to learn how to output one for real images and zero for fake images. To calculate the gradients during backpropagation the sum of the two losses $L_f + L_r$ is used.

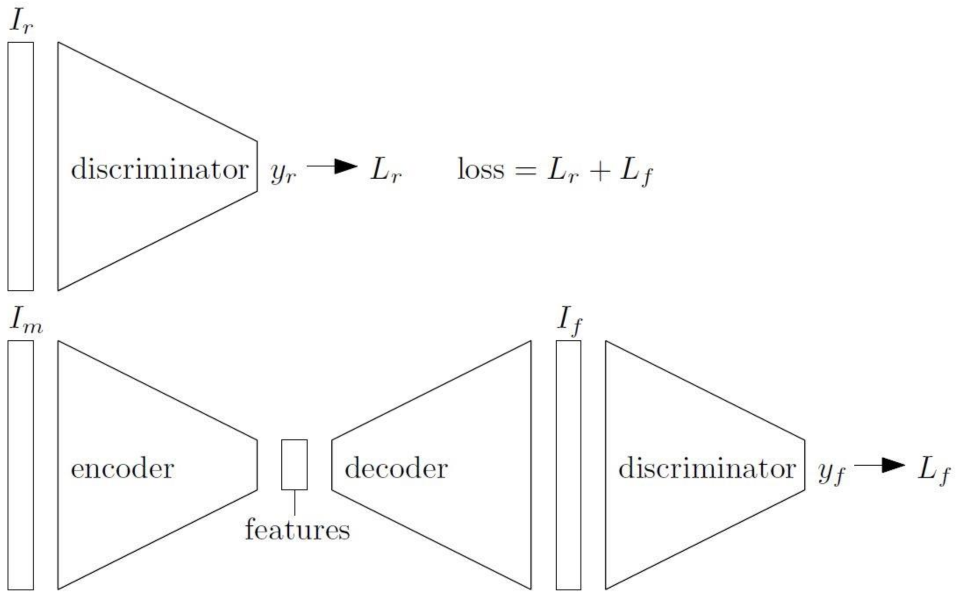

Figure 6 Learning procedure for the discriminator network. I_r are real images, I_m are modelled images, , I_f are fake images, y_r are the labels for real input images (=1), y_f are labels for fake images (=0), L_r is the BCE-loss for real images and L_f is the BCE- loss for fake images.

The discriminator is a network with seven convolutional layers with leaky ReLU activations (negative slope of 0.2), where the middle five layers are supplemented with batch normalization layers. In order to distinguish between real and fake images the network uses one output unit with a sigmoid activation and binary cross entropy loss.

3.2.1.2. Generator training

The training procedure for the generator-autoencoder differs a little bit from the one for conventional DCGAN's. The similarity between both strategies is, that the output of the decoder, obtained by a model image input in the encoder, is passed to the discriminator. The labels for the discriminator input images are now set to one to calculate the fake loss, L_f . What differs from conventional training strategies is the additional autoencoder mean square error loss (MSE-loss) used to force the generator to reconstruct images as real as possible even though the training data is limited. The MSE-loss is calculated between the model images provided by the VAE model and the real images. The overall loss for the training is a combination of the loss backpropagated through the discriminator (L_f) and the reconstruction loss from the autoencoder L_a.

$$loss = (1-\gamma) * L_f + \gamma * L_a \tag{13}$$

With γ as hyperparameter, weighting the amount of the contribution of the individual losses (L_f and $L_{a)}$ to the overall loss. $\gamma \in [0,1]$
During the evaluation of the GAN, the best results have been obtained with γ set to 0.8. In Figure 7 the training strategy is displayed schematically.

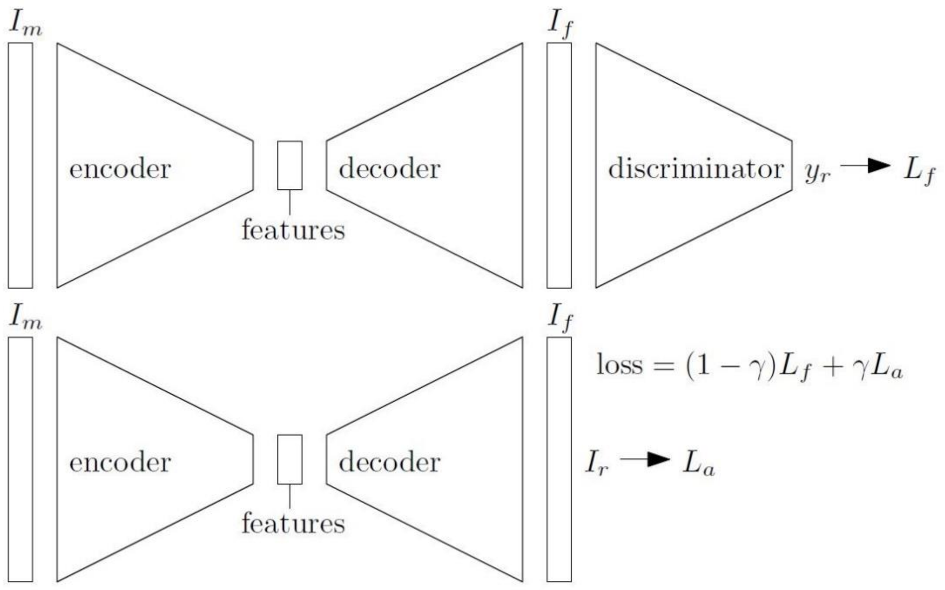

Figure 7 Learning procedure for the generator network. I_m are modelled images, , I_f are fake images, y_r are the labels for the generator training(=1), L_a is the MSE-reconstruction loss, L_f is the BCE- loss for fake images and γ as factor for the overall loss.

The generator-autoencoder is composed of following layer combination. The encoder is built by 6 convolutional layers with leaky ReLU activation each. The middle four layers are supplemented with batch normalization layers similar to the discriminator. The feature vectors dimension is set to 100. The decoder is a combination of seven convolutional layers where the first six are followed by standard ReLU activations and batch normalization layers. The last convolutional layer has a tanh activation and outputs a 3 x 256 x 256 output.

3.2.2. Results of the refinement

In Figure 8 the difference between the model images created by the VAE models and the refined images are visualized. Typical for GAN are the artefacts in the reconstruction. Using additional features in the generator-autoencoder's bottleneck did not lead to any improvement of the image quality.

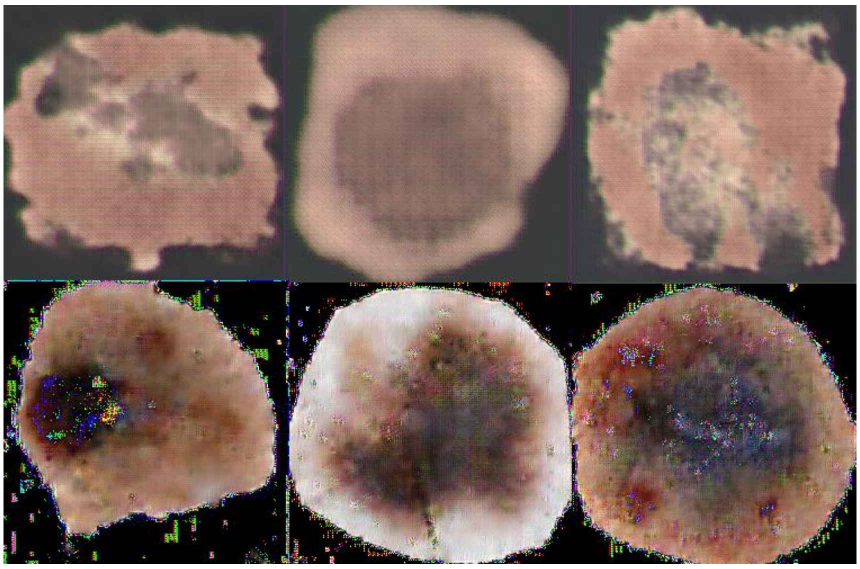

Figure 8 Results of the refinement procedure. Top row: model images generated by VA model, bottom row: refined images

4. Image Classification

As mentioned in the introduction, the classification networks are implemented as deep residual network inspired by the work of He, Zhang, Ren and Sun[7]. ResNets obtain good results at the ImageNet challenge by providing a strategy to train deeper networks and gain accuracy through their increased depth. Residual networks behave like an ensemble of relatively small shallow networks[11]. This effect is shown by Veit, Wilber and Belongie by deleting some layers and perform the trained task. When deleting a layer in network such as VGG it will fail whereas ResNets will handle such an alteration much better since other sub-networks of the ensemble will still perform the trained task. In Figure 9, provided by[11] the results of deleting one layer in a certain network is displayed.

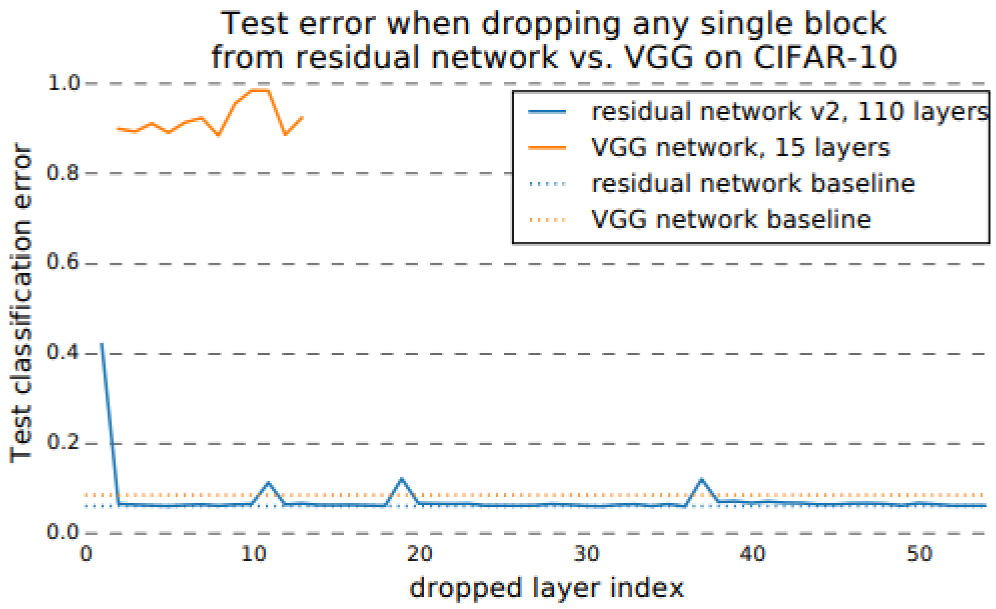

Figure 9 Effect on the learned task when deleting a layer from the network[11]

4.1. Residual Learning and Network Architecture

Residual learning is introduced by adding a shortcut connection, where the key invention was to skip two layers[7]. With this technique we assume that each layer doesn't only need information of the previous one.

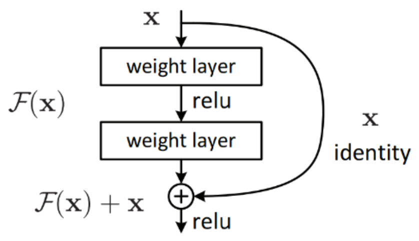
x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
x
identity

Figure 10 Residual learning: a building block[7]

The building block displayed in Figure 10 has been modified by He, Zhang, Ren and Sun for ResNets with more than 50 layers. They introduced revised a building block called bottleneck that is displayed in Figure 11.

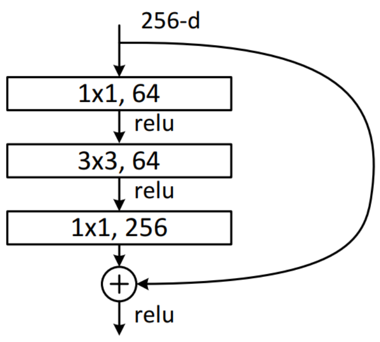

Figure 11 bottleneck building block[7]

The 1 x 1 convolutions are used to reduce and restore the dimensions. The 3 x 3 convolution is used from the bottleneck with smaller input and output dimensions. Further the parameter-free identity shortcut is crucial for the bottleneck architecture since using a projection (using shortcut with learnable parameters) would double the time complexity[7].

For the classification of the skin lesions a palette of ResNets with depth 14, 18, 34, 50, 101 supplemented with networks of a slightly different architecture, wide residual networks (WRNs)[12], has been evaluated. WRNs differ from conventional ResNets in how the basic building block and the bottleneck block are structured. Main differences between both architectures are the use of dropout layers in the WRNs bottleneck blocks and the amount of feature planes, convolutional layers and increased filter size per block. Wide residual networks can be indicated through a widening factor k which multiplies the number of features in convolutional layers and the amount of blocks used n (WRN-k-n).

Dataset	**Subset**	**Melanoma**	**Nevi**	**Seborrheic Keratosis**	**Total**
Original	Train	326	1151	233	1700
	Validation	48	221	31	300
	Test	147	471	132	750
Original + jittered	Train	1240	4684	876	7100
	Validation	64	201	35	300
	Test	147	471	132	750
Original + generated	Train	710	2527	479	3716
	Validation	48	221	31	300
	Test	147	471	132	750

Table 1 Datasets used for the evaluation

In Figure 12 the test accuracies of the different networks for each combination of dataset is displayed.

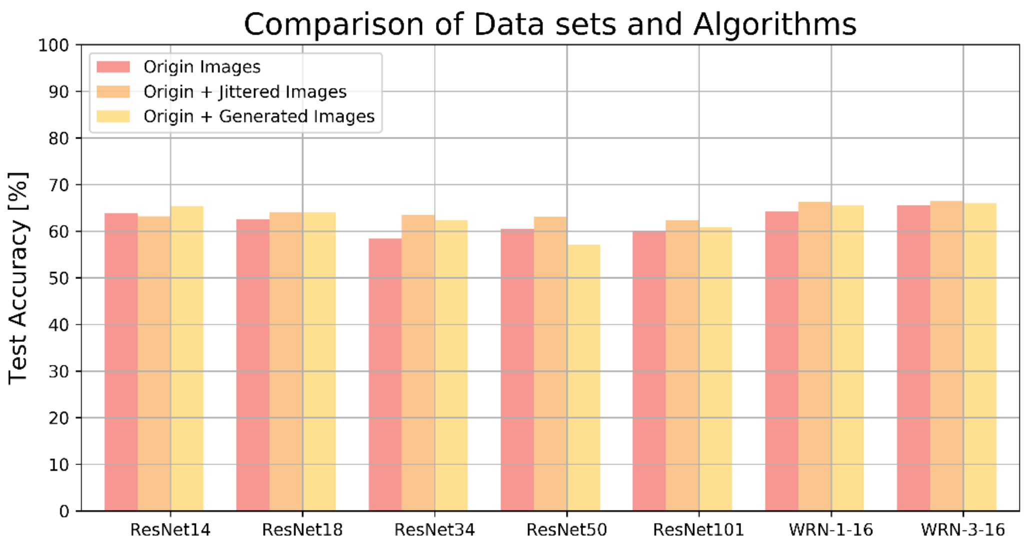

Figure 12 Test Accuracy of different network depth and architectures

For the training of the classification networks PyTorchs Adam-optimizer with a learning rate of 10^{-4} and default initialisation ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) has been used. Additionally an improved version of the mentioned optimizer has been used to train the conventional ResNets. Nadam is an optimizer incorporating Nestrov Momentum into Adam[13], parametrised with a learning rate of 10^{-4} , $\beta_1 = 0.9$ and $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, $weightdecay = 0$ and $scheduledecay = 4 * 10^{-3}$. The results of this strategy is displayed in Figure 13.

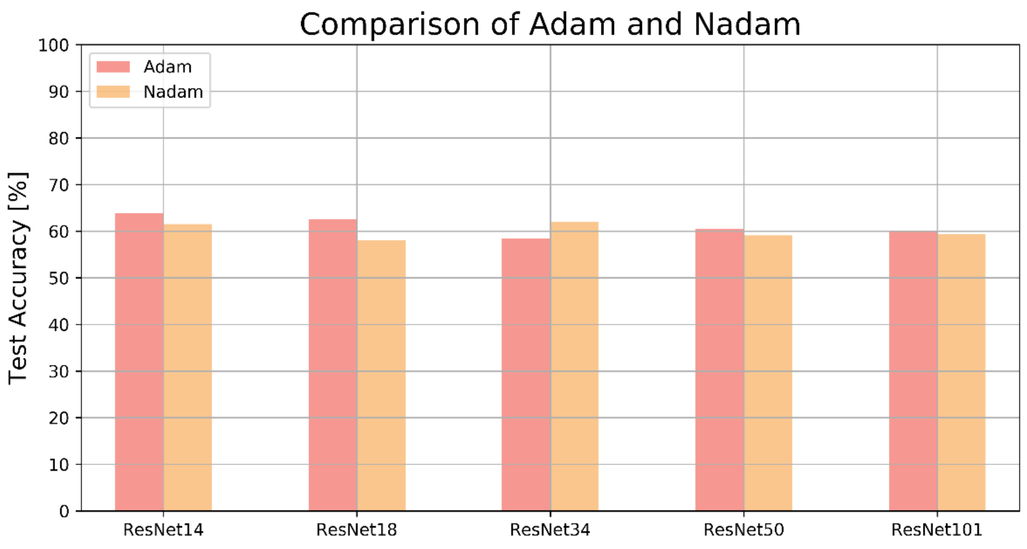

Figure 13 Adam vs. Nadam optimizer used for training the classification ResNets

4.2. Interpretation of the results

Using Nadam instead of the conventional Adam-optimizer does not justify the introduction of additional hyper parameters and the increase in training complexity since the overall accuracy does not improve (Figure 13).

As expected the test accuracy of deeper networks can be increased using dataset augmentation in order to reduce high variance. Since the original dataset supplemented with jittered data provided the most training examples, this dataset led to the best generalization capability in almost all networks except the smallest one. Our proposed method of image generation provides an additional method for improving the generalization performance of the classifier networks. Compared to jittering, the generation of new data via our augmentation pipeline has the advantage of generating as much images as needed, whereas jittering exhibits a confined amount of opportunities. In future work, datasets of the same size should be compared in order to determine which method outperforms the other. Since the nevi class is representant twice as much as the other classes together the test accuracy is based mainly on detecting nevi correctly (Figure 14), what should be considered as well in future work.

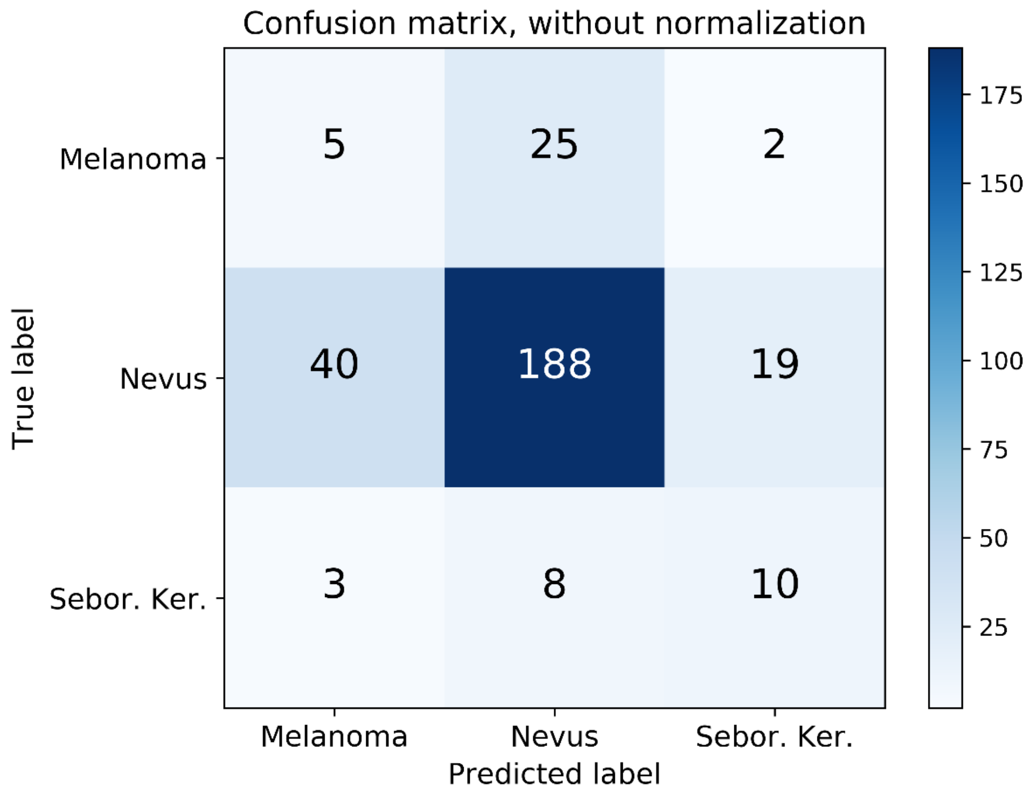

Figure 14 Confusion matrix for ResNet-101 trained with Adam optimizer

5. Acknowledgment

The author gratefully acknowledges the contribution made to this work by Michael Müller and Elias Rüfenacht, who implemented the image classifier, and Jan Riedo, who implemented the generative adversarial network for the refinement.

[1] A. Prasoon, K. Petersen, C. Igel, F. Lauze, E. Dam, and M. Nielsen, “Deep Feature Learning for Knee Cartilage Segmentation Using a Triplanar Convolutional Neural Network,” Springer, Berlin, Heidelberg, 2013, pp. 246–253.

[2] A. Esteva *et al.*, “Dermatologist-level classification of skin cancer with deep neural networks,” *Nature*, vol. 542, no. 7639, pp. 115–118, Feb. 2017.

[3] L. A. Meinel, A. H. Stolpen, K. S. Berbaum, L. L. Fajardo, and J. M. Reinhardt, “Breast MRI lesion classification: Improved performance of human readers with a backpropagation neural network computer-aided diagnosis (CAD) system,” *J. Magn. Reson. Imaging*, vol. 25, no. 1, pp. 89–95, Jan. 2007.

[4] “ISIC 2017 Skin Lesion Analysis.” [Online]. Available: https://challenge.kitware.com/#challenge/n/ISIC_2017%3A_Skin_Lesion_Analysis_Towards_ Melanoma_Detection. [Accessed: 08-May-2018].

[5] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” Dec. 2013.

[6] A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” Nov. 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Dec. 2015.

[8] “Improving the Realism of Synthetic Images - Apple.” [Online]. Available: https://machinelearning.apple.com/2017/07/07/GAN.html. [Accessed: 06-May-2018].

[9] C. Doersch, “Tutorial on Variational Autoencoders,” 2016.

[10] I. J. Goodfellow *et al.*, “Generative Adversarial Networks,” Jun. 2014.

[11] A. Veit, M. Wilber, and S. Belongie, “Residual Networks Behave Like Ensembles of Relatively Shallow Networks,” May 2016.

[12] S. Zagoruyko and N. Komodakis, “Wide Residual Networks,” May 2016.

[13] “Incorporating Nesterov Momentum into Adam - Semantic Scholar.” [Online]. Available: https://www.semanticscholar.org/paper/Incorporating-Nesterov-Momentum-into-Adam-Dozat/82fb61c3701375f04b849ce9240680e349af3dd6. [Accessed: 20-May-2018].