Color Normalization of Blood Cell Images

Emmy Sjöstrand, Jesper Jönsson(✉), Adam Morell, and Kent Stråhlén

CellaVision AB, Mobilvägen 12, 223 62 Lund, Sweden
jejo@cellavision.se
http://www.cellavision.com

Abstract. Traditional microscopes have been used a long time in hematology for blood analysis, but during the last decade many laboratories have started to replace them with digital microscope systems. The appearance of blood cells in the digital images is very important to the end user, ideally they should be identical to how they would look in a traditional microscope. There are several digital microscope systems on the market today with various optics and illumination, which means that images from different systems do not look the same. This is a cumbersome problem in many ways. For example this means cell classification networks need to be trained for every single system. In this paper we investigate the possibility of using deep learning to transform images between digital systems. The main focus is on a cyclic network setup where it is possible to transform the images between two systems. We present two different networks, a cyclic network with a perceptual loss based on the VGG-16 network and a conditional version of a cyclic generative adversarial network (GAN). With these networks we obtain very good results that are better than previous methods for transforming blood cell images.

Keywords: Deep learning · GAN · Color normalization · Blood cell images

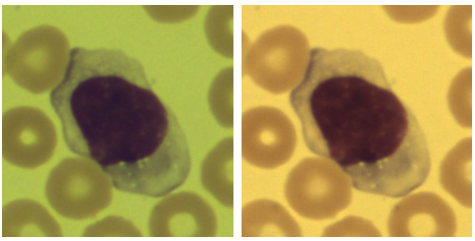
1 Introduction

Modern health care is rapidly changing due to advances in artificial intelligence, and hematology is no exception. Blood analysis is a common tool when screening for different diseases and confirming diagnoses. The conventional way to analyze blood is to run it through a so called cell counter and based on the results, if needed, perform an in-depth analysis of the sample using a traditional microscope. This is time consuming and requires a large number of highly trained laboratory technicians. A more efficient way to analyze blood is to use a digital microscope system. For the end user it is very important that digital images of blood cells look like they would have in a traditional microscope. Digital images that have not been processed in any way are called raw images, and depending

Supported by CellaVision AB.

M. Felsberg et al. (Eds.): SCIA 2019, LNCS 11482, pp. 477–488, 2019.
https://doi.org/10.1007/978-3-030-20205-7_39

on which system the image was captured with they can look quite different. Two digital systems from CellaVision are used in this paper, the DM1200 from their third generation of systems and the DM96 from their second generation of systems. One of the major differences between them is the optics and the illumination. The DM1200 has an LED lamp and the DM96 has a halogen lamp. From now on the DM1200 will be called system X and the DM96 will be called system Y. Figure 1 shows a raw image from system X and a raw image from system Y.

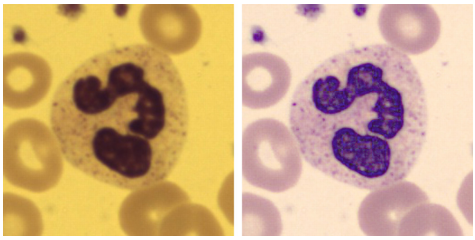


(a) (b)

Fig. 1. Raw images from (a) system X and (b) system Y.

A raw image is not very similar to a microscope image. The process that makes the raw image resemble the microscope image is called normalization. In Fig. 2 a comparison between a raw image and a normalized image is shown. The normalization algorithm first finds a representative background color and uses an affine transform on the entire image to set this background to a predefined value. The dynamic of the image is also changed, for example by increasing sharpness and contrast in the image, especially in the nucleus of the cell, where also some hue adjustments are made. Using this kind of normalization algorithm it is clear that images from different systems cannot be normalized the same way because we want the result to look the same. One solution is to tweak the normalization algorithms so that each system has its own normalization process. It is however very hard to obtain results that are similar enough. Today an algorithm called Hedlund-Morell normalization (below called HM normalization) is used. The main idea of HM normalization is to segment the image and apply different transformations to different parts, see [4]. This works fairly well, but the result is not entirely satisfactory. One problem with the HM normalization is that it does not perform very well on certain cell types, especially those with red colors. A better solution would be to transform images from one system to look like they were captured with another system. Then it would be possible to use the same normalization algorithm for all systems. Unfortunately the transformation is not very simple and a global transformation does not work. The main reason for the complexity of the problem is the different illumination in the systems. An LED and a halogen lamp have different spectral characteristics which can

lead to metameric failure [5]. Experts are often used to working with halogen images and think they look better than LED images.



(a) (b)

Fig. 2. (a) Raw image. (b) Normalized image.

Digital systems often come with analyzing software that uses artificial intelligence to perform different types of analyses. A differential count of white blood cells (WBCs) is a common analysis. A differential count means counting a predefined number of cells, classifying them into their cell type and the result is the proportions of each cell type. One way to do the classification of the WBCs is to use a neural network. Since training neural networks is time consuming and requires large sets of data labeled by experts it is desirable to be able to use the same network for images from several systems.

The aim of this paper is to develop a transformation, a color normalization, between system X and system Y. This should not be confused with the normalization process that makes a raw image look like it would in a microscope. Our color transformation will be a neural network. The goal is that transformed images from system X should be visually indistinguishable from images that were captured with system Y. A secondary goal is that a cell classification network should classify the transformed cell images the same way as their original counterpart, thus eliminating the need to retrain the classification network. This paper is based on a Master's thesis [13].

2 Generative Adversarial Networks (GANs)

In 2014 Goodfellow et al. presented their article "Generative Adversarial Nets" [2] where they introduce a framework consisting of two networks. One of the networks is called the discriminator and is used to define a loss function. The other network is called the generator and this is the network that produces the results. During training the two networks will compete against each other, hence the name "adversarial nets". This paper deals with image transformation so from now on it is assumed that the generator outputs an image. Given a set of images the generator's task is to produce images resembling these images. The discriminator's task is to decide if an image is from the real set of images or

if it is a generated image. The discriminator is shown labeled examples of the real images as well as images from the generator in order to learn the difference. Then the generator tries to fool the discriminator by generating images which are similar to the real images. The generator's loss function is defined using the discriminator, if the generated image is assigned a high probability of being real then the generator's loss is low and vice versa. The two networks are trained alternately, competing against each other, forcing both networks to become better and better. Figure 3 illustrates the GAN framework, the generator is called G and the discriminator is called D. A simple application of a GAN would be to use the well known MNIST dataset to generate images of handwritten digits. For the case with blood cell images we want the generator to produce images that looks like they were captured with system Y.

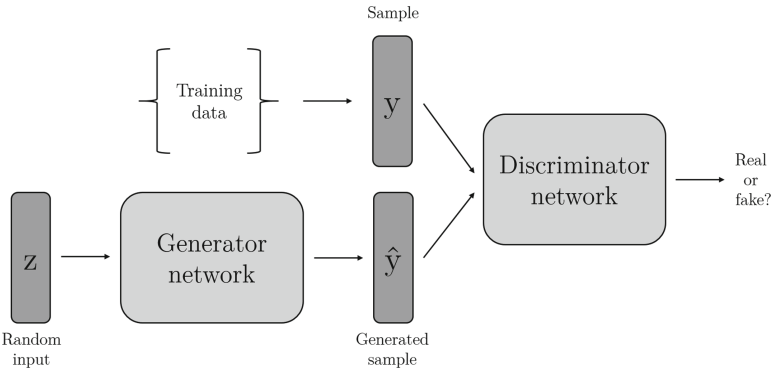


Fig. 3. Illustration of the GAN framework. It consists of two networks, the generator and the discriminator. The input to the generator is a sample from some noise prior. The output of the generator is passed along to the discriminator. The discriminator is trained to separate fake samples from real samples and the generator is trained to fool the discriminator.

For GANs the loss functions of the networks will not only depend on its own parameters. Let x_i and y_i be samples from system X and Y respectively. The discriminator will try to minimize

$$\mathcal{L}_{disc} = -\frac{1}{2N}\sum_{i}^{N}\log\big(D(y_i)\big) - \frac{1}{2N}\sum_{i}^{N}\log\big(1 - D(G(x_i))\big), \tag{1}$$

but can only do so by manipulating its own weights, not the generator's weights. Similarly, the generator will try to minimize

$$\mathcal{L}_{gen} = -\frac{1}{N}\sum_{i}^{N}\log\big(D(G(x_i))\big), \tag{2}$$

but can only do so by manipulating its own weights. This means training the networks will be more like a game than a traditional optimization problem. The solution will be a Nash equilibrium which is a local minimum of $\mathcal{L}_{disc}$ with respect to the weights of the discriminator and a local minimum of $\mathcal{L}_{gen}$ with respect to the weights of the generator [3].

The original GAN gives no control over what type of image is being generated, it just generates an image that looks like it came from the real set of images. In our case we want to transform the content of an image from system X so that it looks like it was captured with system Y, but the generator would just generate an image that looks like it came from system Y with no regards of the content. To be able to condition the generator on some information Mirza et al. created the conditional GAN in 2014 [9]. In the MNIST case this additional information could simply be what digit the generator should produce. In the case of cell images the generator is conditioned on the image from system X. The generator is not allowed to change the shape or structure of the cell in the image, so we do not send any noise which gives us a deterministic transformation from system X to system Y.

3 Cyclic GANs

A cyclic version of the GAN called cycleGAN is original work by Zhu et al. [15]. In 2017 they proposed a new type of framework which goal was to transform images between two different domains, without the need of paired data. Figure 4 illustrates the framework. It consists of two generators, G and F, and two discriminators, D_x and D_y. Generator G is a mapping from domain X to Y and generator F is a mapping from domain Y to X. Discriminator D_x is trained to separate true images from domain X from fake ones i.e. images transformed by generator F from domain Y. D_y is trained in a similar fashion but on images in domain Y. By sending an image through both generators it is possible to explore a loss based on the difference of the input and the output of the full cycle. Figure 5 illustrates the idea. The networks are trained to minimize this loss, called $\mathcal{L}_{cycle}$ defined in (3). This loss combined with the discriminators is the backbone of this framework. The authors do however experiment with using this loss in combination with another loss function, called $\mathcal{L}_{identity}$. It is defined as (4) and ensures the generators manage to perform identity mappings. If G is given an image from domain Y it should preferably output the same image that was given as input. Same argument with generator F if given input from domain X. By introducing this loss they could improve the colors of the transformed images.

$$\mathcal{L}_{cycle} = \|y - G(F(y))\|_1 + \|x - F(G(x))\|_1 \tag{3}$$

$$\mathcal{L}_{identity} = \|y - G(y)\|_1 + \|x - F(x)\|_1 \tag{4}$$

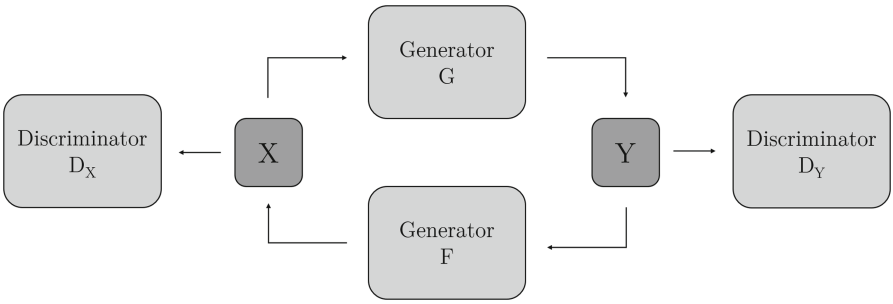


Fig. 4. Overview of the cycleGAN framework.

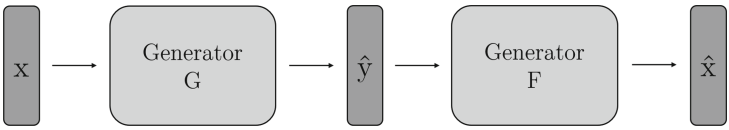


Fig. 5. Image describing the idea of a forward cycle loss. A sample from domain X is sent through both generators and is then evaluated how much in l^1-norm it differs from the input.

4 Perceptual Losses

Imagine an image where every other column is black and the other columns are white, like a zebra pattern. Suppose this is the target image for the generator. If the generator would produce an image that has the same zebra pattern as the target image, but shifted one column, then it has clearly captured the content of the image. Assuming the application is not very sensitive to shifts then this is a very good result, and the loss function should reflect that. In this case loss functions like the l^1-loss, which is the l^1-norm of the difference of the images, are bad choices since they do not capture perceptual differences between images. The l^1-loss between the target image and the generated image would be the highest possible value, despite the fact that the generator actually has captured the stylistic features of the image.

A perceptual loss function can be defined by using a pre-trained classification network. These kinds of losses have been used in many different applications, for example removing visual signs of rain and snow [14], generating an image with higher resolution than the original image [8] and to combine the content of one image with the style of another [7]. All of these applications have a loss function that uses the output from an intermediate layer of a pre-trained classification network. It is a type of loss function which has proven to give high quality images. Figure 6 shows an illustration of this loss with the VGG network [12] as the pre-trained network. The idea is to send the generated image as well as the target image through the classification network and compare higher level features. Some intermediate layer ϕ_j is chosen as the output layer. This drives

the generator towards producing images with similar feature representation as the target images [14]. One way to define the actual loss is to take some norm of the difference between the feature representations. In [7] they make use of the Gram matrix to define a perceptual loss that captures stylistic features of the image.

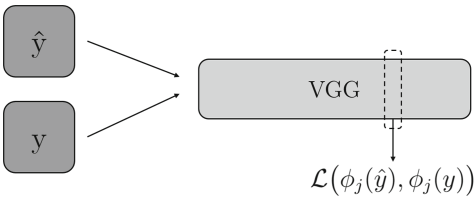


Fig. 6. Illustration of the idea of having a perceptual loss function based on a pre-trained network, in this case VGG.

5 Methodology

5.1 Pre-processing of Data

Data was augmented by rotating some of the images 90°, 180° and 270°. The rotated images look exactly like real data since there is no specific way a cell is positioned on a slide. Our data consists of 9810 cells that were captured with both system X and system Y and then paired and cropped in order to match as closely as possible pixel-wise. The number of each cell type in our data is approximately the same. The data was split into a training set (80%), a validation set (10%) and a test set (10%).

5.2 Conditional CycleGAN (ccGAN)

We have tried a conditional version of the cycleGAN which we call ccGAN (short for conditional cycleGAN). The difference from the original cycleGAN is that we define a loss function (5) that uses paired data. The complete loss function (6) for one generator in ccGAN consists of a loss coming from the discriminator (2), the cycle loss (3), the identity loss (4) and the paired loss (5), where the λ_i's are the weights of the loss functions. In our training we used $\lambda_1 = 1$ and $\lambda_i = 10$ for $i = 2, 3, 4$. The choice of λ's was based on experiments where we got better results when focusing less on the loss coming from the discriminator.

$$\mathcal{L}_{paired}(x_p, y_p) = \|y_p - G(x_p)\|_1 + \|x_p - F(y_p)\|_1 \tag{5}$$

$$\mathcal{L}_{tot} = \lambda_1 \mathcal{L}_{gen} + \lambda_2 \mathcal{L}_{cycle} + \lambda_3 \mathcal{L}_{identity} + \lambda_4 \mathcal{L}_{paired} \tag{6}$$

5.3 Perceptual Cycle Network (pcNet)

We have defined a new framework called a perceptual cycle network (pcNet) and it is based on the cyclic idea from the cycleGAN and a perceptual loss based on the VGG network. We only keep the cyclic part from the cycleGAN, i.e. two generators but no discriminators. The pre-trained VGG-16 network is also used, but it is not part of the training in the sense of getting its weights updated, it only serves as an evaluation network for the perceptual loss function. In this implementation ϕ_j (an intermediate layer in VGG-16) is extracted after the fifth max-pooling layer, see Table 1 in [12] for a full overview of the VGG-16 network. In Fig. 7 the setup of the framework is shown. The loss function which generator F is trained against is defined in (7) where the λ_i's are the loss weights. The loss for generator G is constructed in a similar way. The cycle loss, identity loss and the paired loss are the same as for the ccGAN. The loss $\mathcal{L}_{feat}$ defined in (8) is the feature reconstruction loss using the Gram matrix from [7]. For this network we used the same loss weights for all losses, $\lambda_i = 10$ for $i = 1, 2, 3, 4$.

$$\mathcal{L}_{tot} = \lambda_1 \mathcal{L}_{cycle} + \lambda_2 \mathcal{L}_{identity} + \lambda_3 \mathcal{L}_{paired} + \lambda_4 \mathcal{L}_{feat} \tag{7}$$

$$\mathcal{L}_{feat}(x_p, y_p) = \|G_j^{\phi}\big(F(y_p)\big) - G_j^{\phi}(x_p)\|_F^2 \tag{8}$$

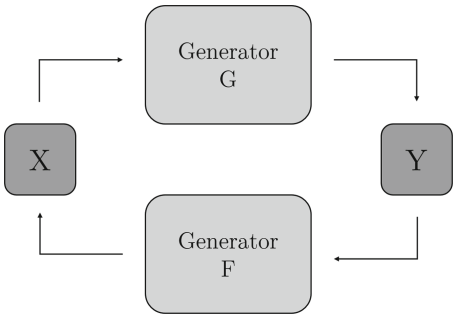


Fig. 7. Cyclic network with two generators and no discriminators.

5.4 Network Architectures

Generator. The architecture of all generators are the same, it is inspired by the Pix2Pix paper [6] where they use a network based on the U-Net [11]. Figure 8 shows an overview of the architecture. The skip connections make it possible to share information between layers which is very common in image mapping when the input image shares structure with the target image. The generator consists of encoding blocks and decoding blocks. An encoding block consists of a convolution followed by a batch normalization and a Leaky ReLU activation function. The size of the convolution kernel is 5×5 and the stride is 2. A decoding

block starts with upsampling, then a transposed convolution, a batch normalization followed by a ReLU activation function. A disadvantage of transposed convolutions is that artifacts such as checkerboard patterns can appear when the kernel size and stride do not match [10]. In our generator we have separated the transposed convolution with stride 2 into an upsampling process followed by a transposed convolution with stride 1. Our generator has approximately 20 million parameters which is almost twice the size of our discriminator.

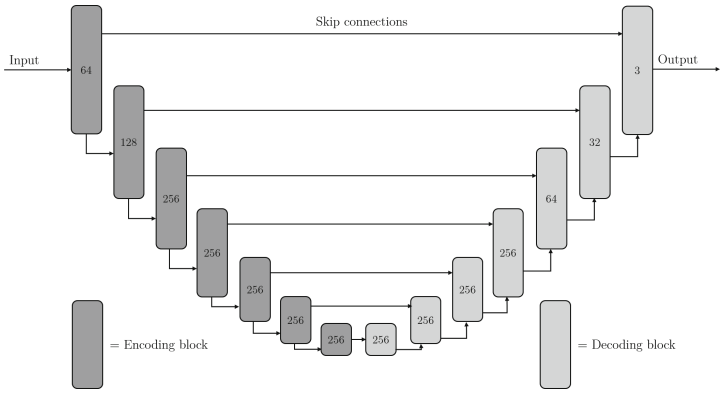


Fig. 8. Illustration of the generator architecture. The number in each block is the number of filters in the convolution layer.

Discriminator. The generators in the ccGAN are never trained solely against the discriminators, the total loss function is a combination of norm-based losses between two images and the loss given by the discriminator. The losses based on norms are good at capturing low frequency data, but might have problems with more general stylistic features of an image. The idea is that the discriminator will encourage the generation of high frequency information in the images by penalizing lack of high frequency data. A discriminator called "PatchGAN" was proposed in [6] and it has a receptive field of 70×70 pixels, small "patches". Each patch should capture the local style of the image. The output from the discriminator is the average of the result of the evaluation of each patch being real or fake. Our patch discriminator has about 11 million parameters and consists of 5 blocks that start with a convolutional layer, then a batch normalization followed by a Leaky ReLU activation. The convolutions have kernel size 4×4 and the stride is 2 for the first 3 blocks and 1 for the last 2 blocks. There is no batch normalization for the first and last block. The final activation function before the mean layer is a sigmoid instead of a Leaky ReLU. Figure 9 shows a simple illustration of our discriminator.

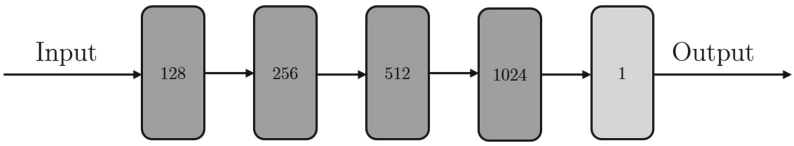


Fig. 9. Illustration of the discriminator architecture. The first blocks are regular convolution - batch normalization - Leaky ReLU blocks. The last block has a sigmoid activation function and a mean layer. The number in each block is the number of filters in the convolution layer.

5.5 Additional Training and Implementation Details

The frameworks used were Tensorflow and Keras. Both networks were trained for 50000 iterations with a batch size of 4 for the ccGAN and a batch size of 2 for the pcNet. The optimizer was Adam with a learning rate of 0.0002, otherwise default Keras parameters [1]. All the training was performed on an NVidia GeForce GTX 1080 Ti GPU. Every iteration of training of the ccGAN starts with training the discriminators, then the weights of the discriminators are frozen and each generator is trained while the other generator's weights are frozen. The pcNet is trained in a similar fashion, but since it does not have any discriminators it just consists of alternately training each generator.

6 Results and Conclusions

The results obtained in this paper are satisfactory and better than previous methods. It is very hard for a person to see any differences between a transformed image and its original counterpart. It is even harder to decide if an image is real or transformed when shown a single image. One problem with human evaluation is that it is subjective, but it is hard to come up with a metric that measures the important characteristics of this type of image. Popular metrics like PSNR (peak signal-to-noise ratio) and SSIM (structural similarity) do not seem to correlate with how good a person thinks the images are. The goal is however that the user should not be able to distinguish between transformed and real images and we think we have achieved that. Figure 10 shows a comparison of the two different networks. Experts think that pcNet is slightly better than ccGAN, but both are better than the current method.

The current method used is the HM normalization. Since the HM normalization does not only transform the image, it also normalizes it, we had to normalize our transformed images to be able to compare. Figure 11 shows the same images as Fig. 10 but normalized. The structure of the HM normalized cells are very good, but the colors could be better, especially for the cell type eosinophils (the cell type in the first row). Our transformation takes approximately 300 ms on a CPU (Intel Core i7-7700 at 3.60 GHz) and 10–20 ms on a GPU (NVidia GeForce GTX 1080 Ti). The advantage of the HM normalization compared to

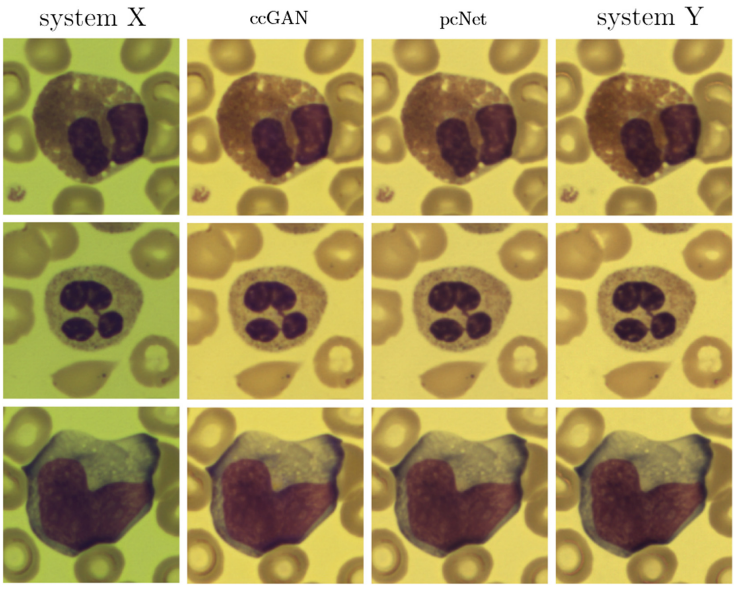


Fig. 10. Comparison of all networks with different raw cell images from the test set.

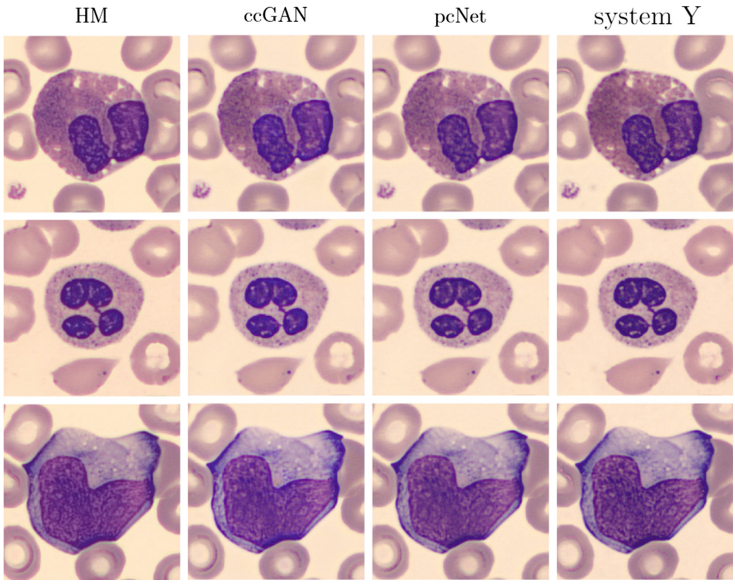


Fig. 11. Comparison of all networks with different normalized cell images from the test set.

our networks is that it is a lot faster, which of course is very important when implementing it in an actual system.

A cell classification network was used to classify both real images and their transformed counterpart. We do not necessarily care whether the classification is correct or not, the goal is that the real images and the transformed images are classified as the same cell type. This evaluation was performed by using 981 image pairs. The results were 88.6% of the pairs were classified the same for pcNet and 85.2% were classified the same for ccGAN. These results do agree with the experts visual evaluation of the network. To improve classification further it could be possible to use a layer from CellaVisions actual classification network instead of or in combination with VGG to define a perceptual loss.

References

1. Optimizers, 1 April 2019. https://keras.io/optimizers/
2. Goodfellow, I.J., et al.: Generative adversarial networks. In: NIPS, pp. 2672–2680 (2014)
3. Goodfellow, I.J.: NIPS 2016 tutorial: generative adversarial networks abs/1701.00160 (2017). http://arxiv.org/abs/1701.00160
4. Hedlund, S., Morell, A.: Segmentation based image transform. US Patent 9,672,447, 14 May 2014. http://www.google.it/patents/US9672447B2
5. Hunt, R.: Measuring Color, 3 edn, p. 114. Fountain Press, Chichester (1998)
6. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. ArXiv e-prints, November 2016. arXiv:1611.07004
7. Johnson, J., Alahi, A., Li, F.: Perceptual losses for real-time style transfer and super-resolution. abs/1603.08155 (2016). http://arxiv.org/abs/1603.08155
8. Ledig, C., et al.: Photo-realistic single image super-resolution using a generative adversarial network. arXiv preprint (2016)
9. Mirza, M., Osindero, S.: Conditional generative adversarial nets. arXiv:1411.1784 [cs.LG] (2014)
10. Odena, A., Dumoulin, V., Olah, C.: Deconvolution and checkerboard artifacts. Distill (2016). https://doi.org/10.23915/distill.00003. http://distill.pub/2016/deconv-checkerboard
11. Ronneberger, O., Fischer, P., Brox, T.: U-Net: convolutional networks for biomedical image segmentation. abs/1505.04597 (2015). http://arxiv.org/abs/1505.04597
12. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. abs/1409.1556 (2014). http://arxiv.org/abs/1409.1556
13. Sjöstrand, E., Jönsson, J.: Cell image transformation using deep learning (2018). https://lup.lub.lu.se/student-papers/search/publication/8945302
14. Wang, C., Xu, C., Wang, C., Tao, D.: Perceptual adversarial networks for image-to-image transformation. abs/1706.09138 (2017). http://arxiv.org/abs/1706.09138
15. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. abs/1703.10593 (2017)