Colour Image Synthesis using CNNs

Santosh Vangapelli Satya Kuppam

Abstract

We propose a novel approach to colorization with a fully CNN, inspired by the image transformation techniques that use perceptual loss[3]. We present training details, conduct a Turing Test to determine our network's performance and compare it with results obtained by Baldassarre et al. [1]. In the end we reflect on the network's performance and provide further possible improvements.

1. Introduction

The goal of the project is to create a colorized version of a gray scale image. The gray scale version of the image is devoid of any information regarding the color. Hence the colorized image that is generated is only one out of many possible colorized versions. The goal is to create a colorized version, which when presented to people, they cannot ascertain whether the image is real or generated. A number of techniques exist currently that address this problem. Techniques that do not leverage deep learning tend to rely overwhelmingly on human input and tend to be very time consuming as well. Deep learning based models especially after training on a large number of images generates better results compared to others. Hence they have become the most widely used technique for colorization.

In our project we take into consideration the existing methods in colorization that use convolutional neural networks(CNNs). We improve upon these methods by leveraging the high level image feature representations extracted from the activations of a pre-trained classifier. We developed two different networks that use two different pre-trained classifiers viz VGG16[8] and Inception-Resnet-v2[9], we shall refer to them as VGG and RESNET respectively for the rest of the paper. We train one of the networks against *perceptual loss* using VGG, which is more robust at determining image similarities than a per pixel loss. The other network uses the last block of activations from RESENET to augment the activations from the CNN layers. The block from RESNET will contain the high level feature representations of the image. To the best of our knowledge colorization with *perceptual loss* has not been done before. Finally we will compare the results obtained from both these networks. Due to time and resource constraints we could only train our networks on 25K and test them on 5K images. We have presented a random sample of this 5K images to 28 people during the poster session. In total these 28 people viewed 118 images and they could determine that 67% were generated images. The rest of the paper briefly describes the existing state of the art for colorization in deep neural networks, details the experiments we conducted, the problems we encountered, the resolutions we devised to circumvent these problems, presents the results we obtained and proposes further improvements.

2. Related Work

Colourization of gray scale images is a well studied topic of research. Hence a number of techniques exist in literature that are capable of accomplishing this task. Existing methods can be broadly classified into two, hint based colorization and deep colorization. Hint based colorization as the name suggests takes cues from the user and presented with a reference image will try to colour the gray scale image. However these methods are time consuming. Hence research in colorization with deep learning techniques has garnered steam in the fast few years. The idea of using a pre-trained classifier to color images was first demonstrated by Larsson et al [4], where a fully convolutional version of VGG-16 was used to create a probability distribution of colors over each pixel in the image. Baldassarre et al. [1], use this idea and improve on it using a fully CNN with the activations from RESNET to extract features that will aid the CNN in coloring the image. We also use their idea of using the Lab color space to train the CNN network.

Johnson et al[3], use the activations from a VGG for training a fully CNN for an image transformation task. They train the fully CNN to create images of a certain style, for example generating images in the style of Vincent Van Gogh, using the activations from a VGG network to calculate a loss function viz. *perceptual loss*. We take this idea of *perceptual loss* and apply the technique to the image colorization problem. We will also compare the results obtained with Baldassarre et al's. [1] network. To make the comparison fair we use the same set of images for testing both the networks.

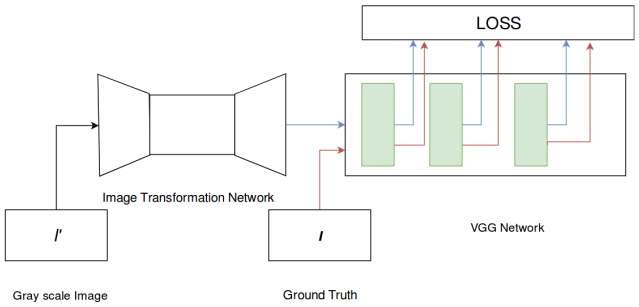

Figure 1. High level view of Image transformation and loss networks

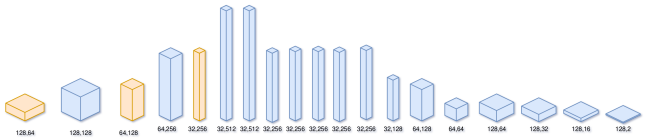

Figure 2. Activation volumes of Image transformation network after each convolutional layer. As you can see, depth of the volume increases in the encoder and gradually decreases to 2 in the decoder. Accordingly the hight and width decrease in the encoder while they increase to original input size in the decoder

3. Approach

We consider images of size $H \times W$ in the Lab color space. [7]. Starting from the L, the luminance part of the image $X_L \in \mathbb{R}^{H \times W}$, our model generates the remaining components, and adds them to the L component to create a colored image $X \in \mathbb{R}^{H \times W \times 3}$ corresponding to the input. Let the mapping from L component to colored image be[3]

$$\mathcal{F} : X_L \rightarrow (X_a, X_b)$$

Where X_a, X_b are the a, b components [7] predicted by the network for the given luminance component X_L.We then combine the a,b components with L component to generate fully colored image.

$$\mathcal{G} : (X_L, X_a, X_b) \rightarrow I'$$

In order to be fully independent of input image size, our architecture is fully based on CNNs, a model that has been well studied and employed in the literature [2]. Briefly, each convolutional neural network(CNN) layer is a set of small filters that fit the specific local patterns in the input image. These filters are learnt by the network during training.

3.1. Architecture

We use perceptual loss with custom Image transformation network to train and generate color images from gray

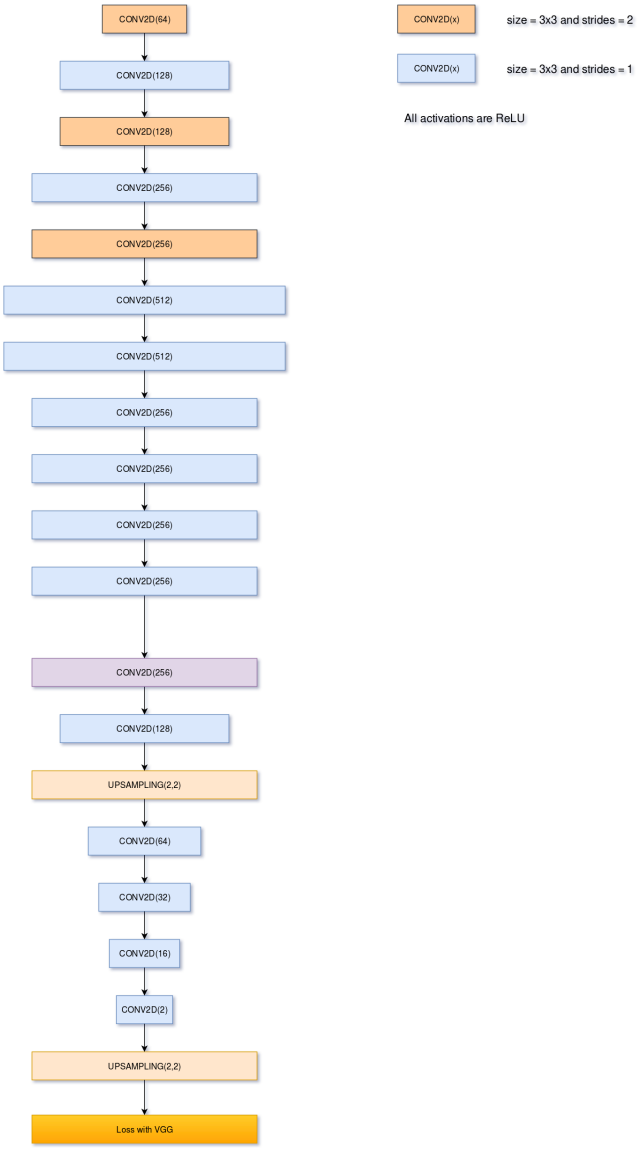

Figure 3. Image transformation network with various layers specified. Encoder includes the layers present before the purple block and the blocks after that are the decoder.

scale images. As shown in figure 3, our system consists of two different components: an image transformation network f_w and a loss network ϕ that is used to define several loss functions $l_1, l_2.....l_k$.

$$l_i = \phi_i(I, f_w(I_L))$$

$\phi_i(I, I')$ indicates the sum of squared differences of each activation of i^{th} layer of our loss network.

3.2. Image Transformation Network

The image transformation network is logically divided into two main components, encoder and decoder. The en-

coder reduces the height and width of the input while increasing the depth of the layer output. The Encoder processes $H \times W$ grey-scale images and outputs a $H/8 \times W/8 \times 512$ feature representation. We use 10 convolutional layers with 3×3 kernels. Padding is used to preserve the layer's input size. Moreover stride of 2 is applied on 4^{th} and 6^{th} layers halving the dimension of their output and reducing the number of computations required.

Decoder takes this $H/8 \times W/8 \times 256$ volume and series of convolutional layers are applied on that volume to generate final output of size $H \times W \times 2$. Basic nearest neighbor approach based upsampling is used so that output's height and width are twice the input's.

Although the input and output have same height and width there are few advantages of down sampling and then upsampling as we do in our network. First advantage is computational, number of multiply-adds is proportional to H, W of each layer. We can use more filters with the same amount of computation after we down sample. Another benefit of down sampling is effective receptive field sizes. High quality colorization requires coloring large parts of image coherently, therefore having large effective receptive field in the input for each pixel in the output is advantageous. Without downsampling, each additional 3×3 convolutional layer increases the effective receptive field by 2. When we downsample by a factor of D, effective receptive field of each activation in the resultant layer increases by a factor of D, therefore each further 3×3 convolutional layer increases receptive field size by $2D$ giving larger receptive fields by same number of layers.

3.3. Perceptual Loss Function

We define a loss function that measures the high level semantic and perceptual differences between the images as described by [3]. We use 16-layer VGG network pretrained on the ImageNet dataset as our loss network.

Rather than encouraging the pixels of the generated image $y' = \mathcal{G}(X_L, \mathcal{F}(X_L))$ to exactly match the ground truth y, we instead encourage them to have similar feature representations as computed by the loss network ϕ, VGG16. Activations of later layers of VGG when image I is processed through it are highly correlated to the high level features. Since VGG is trained over large number of colored images, it has right color information for most of the real life objects like grass, sky, winter, spring etc. For objects that have a natural color, VGG will recognize them correctly in the ground truth but will fail to recognize them in the generated color image creating a difference in activation, even though the per pixel difference might not be high.

Let $\phi_j(I)$ be the activations of the j^{th} layer of the network ϕ when image I is processed by ϕ; if j is a convolutional layer then $\phi_j(x)$ will be a feature map of shape $C_j \times H_j \times W_j$. Lets l^j_ϕ denote the eucledian distance between the j^{th} layer activations of ground truth I and generated image I'.

$$l^j_\phi(I, I') = \frac{1}{C_j H_j W_j} ||\phi_j(I) - \phi_j(I')||^2_2$$

As shown in [5] finding an image I' corresponding to I that minimizes the loss of earlier layers will produce visually indistinguishable image while for later layers overall spatial structure is preserved but color, texture and exact shape are not. Using a weighted combination of the multiple layers activation differences, we can achieve image I' that is much closer to I both pixel wise and semantically.

Our final image I' can be generated by solving

$$I' = arg\ min_{I'} \sum_j \lambda_j l^j_\phi(I, I')$$

Where $\lambda_j \in \mathbb{R}^+$ denoted the weight we give to euclidean differences of j^{th} layer activations. We play around different combinations of the weights λ to find the right values.

3.4. Preprocessing

Before passing L component of the image through the image transformation network, we normalize each pixel value to [0, 1]. Once we get the a and b component outputs of the Image transformation network, we add the L component and transform it to RGB format. We then send the ground truth and generated images in RGB format through the loss network to calculate the loss. We also follow the guidelines of VGG16 before sending the generated and ground truth through the loss network. It involves subtracting a fixed pixel from each of our images and then rotating by 180 degrees.

4. Experiments and Training

All the experiments that we conducted can be classified into two types. The first set of experiments are to arrive at a model architecture that performs reasonably well by changing the type of neurons, adding batch normalization layers, various stochastic gradient descent algorithms, upscaling and convolution transpose layers. The second set of experiments are done to improve the results obtained above with the inclusion of the classifier and trying to get the loss to converge.

4.1. Activation Function

We started off by building a small network with eight convolution layers with the tanh neuron and tried to over fit 100 images. We realized that the tanh neuron had the disadvantage of gradients dying after a certain number of iterations hence we chose to use the ReLU neuron instead

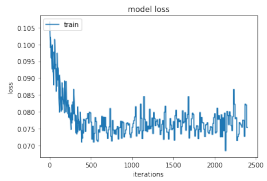

Figure 4. relu activation

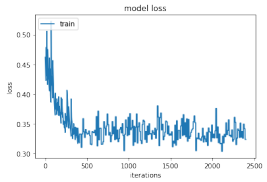

Figure 5. tanh activation

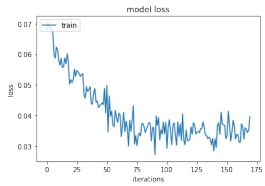

Figure 6. with batchnorm

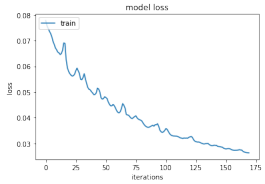

Figure 7. without batchnorm

Figure 8. with stride 2

Figure 9. with maxpooling

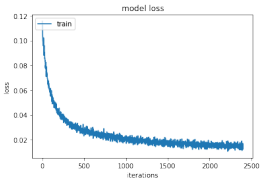

Figure 10. RMSProp

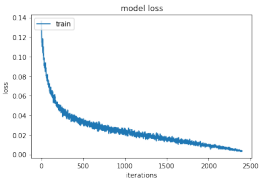

Figure 11. Adam

Figure 12. Bw image

Figure 13. Our result

and this successfully circumvented the problem. We did not experiment with the sigmoid neuron because this will have the problem of dying gradients as tanh neuron. As you can see in Figure 4 and Figure 5, relu leads to lower loss.

4.2. Batch Norm Layers

After over fitting with 100 images we tried to train the network with 500 images. Loss decreased very slowly and did not converge even after running 20 epochs. We tried adding batchnorm after every layer and loss converged much faster.

4.3. Maxpooling Vs stride 2

As we explained in our architecture, we reduce the height and width of the activation volumes in the encoder and then increase them in decoder. We experimented with both maxpooling and stride 2 In our encoder to reduce the height and width of the activation volumes. In the decoder, we experimented with upsampling and deconvolution. We got the best results with stride 2 in the encoder and upsampling in the decoder. We saw better color contrast with stride 2 compared to maxpooling(see Figure 8 and Figure 9).

4.4. RMSProp Vs Adam

With the network we obtained above, we tried training the network with 5000 images using MSE loss. Both the Stochastic Gradient Descent algorithms RMSProp and Adam could make the loss converge. However with rmsprop we are able to converge with much less number of batches compared to Adam. As you can see in Figure 10 and Figure 11, Rmsprop converged within 1000 batches but Adam still has higher loss after 1000 batches.

4.5. Learning Rates with Perceptual Loss

As explained in previous sections we use the VGG network for calculating the loss and this technique is called the perceptual loss. We trained the model with 25k images minimizing the loss function explained above. We started with a learning rate of 0.001 and 100 epochs. We observed that loss stagnated after 50 epochs. To circumvent this problem we put a decay rate of 0.5. This was insufficient to further reduce the network loss. After trying different learning rates at various epochs we got the optimum loss with the following hyper parameters. For the first thirty epochs we use 0.001, for the next 35 we use 10^{-4} and for another 35 epochs we use 10^{-8} as the learning rate. This helped the loss function to converge.

5. Results and Visualizations

We test our network and the the network proposed by Baldassarre et al. [1] with the same test set of 5K images. We have presented some of the results along with the ground truth. See Figures from 14 to 23. Figures from 24 to 27 show the layer 5 activations.

6. Insights

Our network is able to color most common objects like grass and clear sky properly but tends to color conservatively where multiple colors are possible. There are multi-

Figure 14. Colored with Baldassarre et al.

Figure 15. Ground truth

Figure 16. Bw Image

Figure 17. Our result

Figure 18. Colored with Baldassarre et al.

Figure 19. Ground truth

Figure 20. Bw image

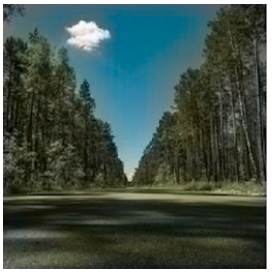

Figure 21. Our result

Figure 22. Colored with Baldassarre et al.

Figure 23. Ground truth

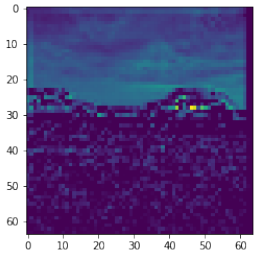

Figure 24. layer 5 activ

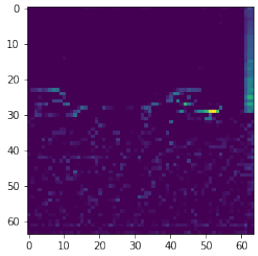

Figure 25. layer 5 activ

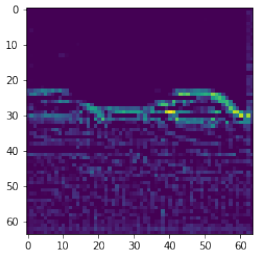

Figure 26. layer 5 activ

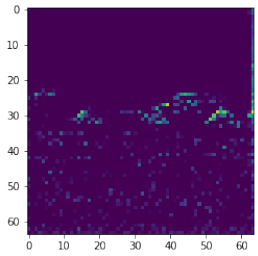

Figure 27. layer 5 activ

Figure 28. Ground truth

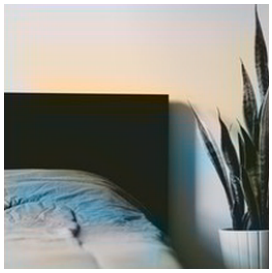

Figure 29.

ple t-shirts with various colors in our training set, network tried to average out the colors it has seen in the training set and often colors the t-shirts with grey or black based on the grayscale in the bw image. As we can see from our results(Figure 36-37), the shirt color hasn't changed even though the background is colored correctly. Sky and trees are guessed properly in most of the results but new and unique objects are not guessed properly(see Figure 46). Another issue we have seen is that color crosses the border and dissolves into object beyond the border of current object.

Most of the images generated on the validation set did not have enough saturation, so we increased the saturation by 40 percent as part of post processing and the images are much more realistic.

When we ran our network with large images or small images color diffusion is rather large. Indicating that number of filters in our network might be insufficient.

One of the possible way to solve the color dissolution problem is to generate RGB components instead of a,b

Figure 30. Ground truth

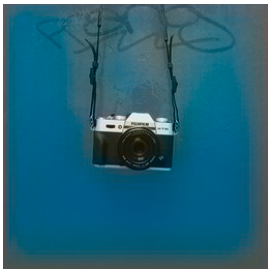

Figure 31. Our result

Figure 32. Ground truth

Figure 33. Our result

Figure 34. Ground truth

Figure 35. Our result

Figure 36. Ground truth

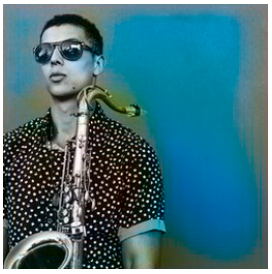

Figure 37. Our result

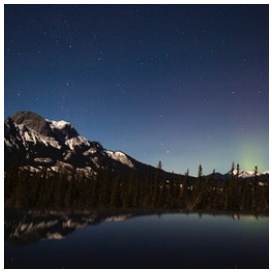

Figure 38. Ground truth

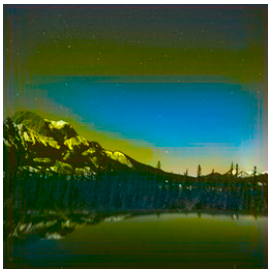

Figure 39. Our result

Figure 40. Ground truth

Figure 41. Our result

Figure 42. Ground truth

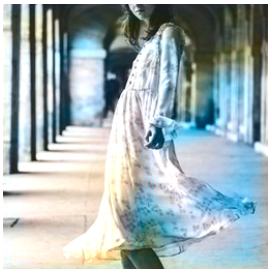

Figure 43. Our result

Figure 44. Ground truth

Figure 45. Our result

Figure 46. Ground truth

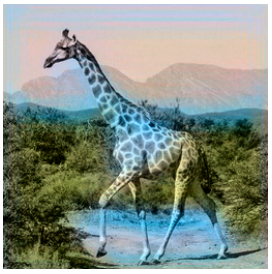

Figure 47. Our result

Figure 48. Ground truth

Figure 49. Our result

Figure 50. Ground truth

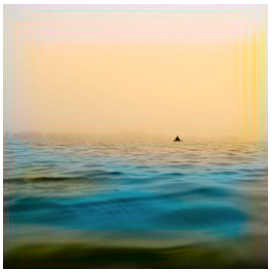

Figure 51. Our result

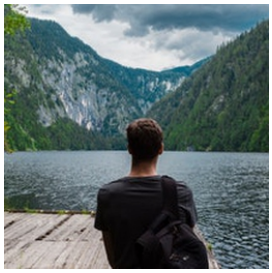

Figure 52. Ground truth

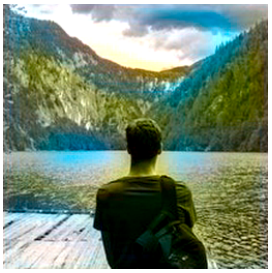

Figure 53. Our result

components by the Image transformation network. Since we merge a and b components generated with L components, color borders are inconsistent with gray scale image borders. Activation differences due to color change are much lower compared to change in contrast. To validate our proposition, we trained on smaller set of 2K images with Image transformation network producing the complete RGB output, color dissolution was much smaller in this network. We plan to retrain this model with a larger set of images and compare the results with our proposed model with Lab components.

7. User Study or Evaluation Metric

We wanted to gauge the public perception of our colorized images. Hence during the poster session we picked images randomly from the 5000 test images and presented it to 28 people. These 28 people viewed a total of 118 images and they could determine that 67% of these were generated images. In their work Baldassarre et al. [1] state that in their usability study only 20% of the test images were identified as generated images. This discrepancy might arise because they have hand picked 9 images from their test set which they felt could fool the user, on the other hand we randomly picked the generated images. Secondly, we have trained our network only on 25000 images and we are confident that when trained on the entire IMAGENET we will achieve better results. Thirdly, most of the users in our study constitute deep learning enthusiasts who would have been more discerning than an average person. Lastly for a more accurate result we need to create two groups of users and treat one of them as the control group.

8. Limitations and Future Work

There is strong training bias to our model. Inherently Colorization is a hard problem and involves the knowledge of subjects' original color and identifying them. Since we use VGG16 as classifier, images that are not similar to IMAGENET performed poorly since objects in them are not identified properly. We have also made an observation for subjects that are not present in our training set, these are colored inaccurately. For example in Figure 13 the puppy is colored green by our network since it could not guess its color. Our network could not identify the color of the car in Figure 17 since such a car might not have been seen by our network during training.

We have also observed that our network performs poorly on images whose dimensions are much larger or smaller than our standard training image size of 256×256

Training on a larger training set can improve our results. Lately GANs have shown promising results in number of image synthesis tasks [6]. Removing classifier and adding Fully CNNs as discriminator in GAN can remove the classifier training bias.

9. Computational Resources

We have trained and tested our model using Google cloud instance with Tesla K80 GPU attached to it. We also used TensorFlow as the underlying framework to utilize the GPU to train our model, download and utilize pre trained VGG16 and InceptionNetV2. Jupyter notebook is used as development environment.

References

[1] F. Baldassarre, D. G. Morín, and L. Rodés-Guirao. Deep koalarization: Image colorization using cnns and inception-resnet-v2. *arXiv preprint arXiv:1712.03400*, 2017.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[3] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[4] G. Larsson, M. Maire, and G. Shakhnarovich. Learning representations for automatic colorization. In *European Conference on Computer Vision*, pages 577–593. Springer, 2016.

[5] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[6] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[7] A. R. Robertson. The cie 1976 color-difference formulae. *Color Research & Application*, 2(1):7–11, 1977.

[8] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[9] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pages 4278–4284, 2017.