Perceptual Losses for Real-Time Style Transfer and Super-Resolution

Iantsa Provost, Lilian Rebiere-Pouyade, Bastien Soucasse, and Alexey Zhukov

Presented as part of the
AMIP
course unit.

Computer Science Master's Degree in Image and Sound
Université de Bordeaux, France
January 5, 2022

Contents

1 Introduction

Johnson et al. (2016) introduced neural network models for style transfer and super-resolution, based on the convolutional neural networks (CNNs) for image transformation introduced by Radford et al. (2015).

The image transformation models take an image as input and produce a transformed version of this image. They are trained by minimizing a loss function between this produced image and a target image.

The key upgrade is the use of perceptual loss functions for a feature-wise similarity score—instead of pixel-wise—when checking the similarity between the generated image and the target image. This technique focuses on the similarity between high-level feature components of the images—instead of their pixels—and therefore maintains the perceptual quality of the generated image.

We propose an implementation of the super-resolution model by Johnson et al. (2016). Super-resolution models are used to generate high-resolution (HR) images from low-resolution (LR) versions. These methods apply to image processing, computer vision, and medical imaging.

To train the model, we use a dataset of HR images and generate the LR version of these images. We use the LR images as input for the model, and HR versions as the ground-truth target.

In this report, we present and explain the method used in the model, describe any implementation difficulties that we encountered, and analyze the qualitative and quantitative results of the experiments. We also discuss the energy consumption used during the project and consider other possible impacts. Finally, we suggest possible extensions and future directions for this work.

2 Method

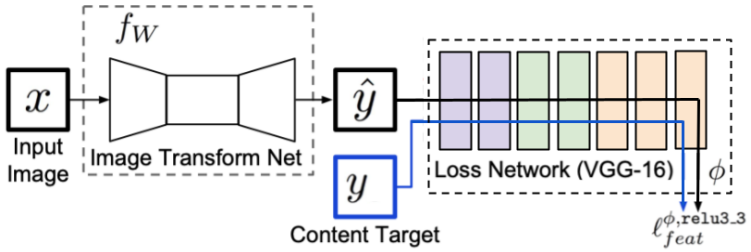

Figure 1: Overview of the architecture of the model proposed by Johnson et al. (2016), focusing on super-resolution.

The proposed model illustrated in Figure 1 consists of two components: the image transformation network f_W and a loss network ϕ. Given a low-resolution input image x, it generates an output image that is expected to be similar to the ground truth high-resolution image y.

The image transformation network is a deep residual convolutional neural network with a set of weights parameter denoted W. From input images x, it provides output images $\hat{y}$.

The loss network defines a feature reconstruction loss l^{ϕ}_{feat}. It delineates several feature reconstruction loss functions $l^{\phi,i}_{feat}(\hat{y}, y_i)$ that each measure the difference in content between output image $\hat{y}$ and content target images y_i. Here, i denotes the layer. Furthermore, Johnson et al. (2016) drew inspiration from several papers such as the one by Gatys et al. (2015) in order to make the most of their feature reconstruction loss. The principle is to use a network pre-trained for image classification as a loss network. This way, the beforehand learning of the perceptual and semantic information will facilitate the measure computation of the feature reconstruction loss.

Gathering both components, the image transformation network adjusts its weights W by minimizing the combination of those loss functions, using stochastic gradient descent (Equation 1). Note that the formula given in the paper (Johnson et al., 2016) mentions a "weighted combination of loss functions" but does not provide more information about the aforementioned weights. As a consequence, we decided to set them all to 1.

$$W^* = \text{argmin}_W \mathbf{E}_{x,y_i} \left[\sum_{i=0} l^{\phi,i}_{feat}(f_W(x), y_i) \right] \tag{1}$$

2.1 Image Transfomation Network

As mentioned previously, the image transformation network is a deep residual convolutional neural network. That is, it includes residual blocks that prevent vanishing gradient issues—when the gradients of the parameters become very small, hindering the ability of the network to learn and improve.

Residual blocks in the model are defined using the architecture design of Gross and Wilber (2016), with the exception of the deletion of the last activation ReLU.

The network takes as input a low resolution image of shape $3 \times 288/f \times 288/f$, and ouputs an image of shape $3 \times 288 \times 288$, which is a generated estimation of the high-resolution version of the input image.

2.2 Loss Network

What propose Johnson et al. (2016) is to try and make the images have similar feature representations, instead of forcing a pixel-per-pixel match. To do so and to make it more performant, the loss network is here defined as a pre-trained network for image classification. More specifically, they use the 16-layer VGG network (Simonyan and Zisserman, 2014) pre-trained on the ImageNet dataset (Russakovsky et al., 2014).

Now to compute the feature reconstruction loss $l^{\phi,i}_{feat}$ in Equation 1 between the output $\hat{y}$ and the content target y at a layer i, we use the mean square Euclidean distance (Equation 2).

$$l^{\phi,i}_{feat}(f_W(x), y_i) = \frac{1}{C_i H_i W_i} \|\phi_i(\hat{y}) - \phi_i(y)\|_2^2 \tag{2}$$

Note that C_i, H_i, and W_i are respectively the number of channels, height, and width of the input image. And, $\phi_i(x)$ is the feature representation of x at layer i (here, x does not mean the input image, but an arbitrary image).

3 Implementation

3.1 Image Transformation Network

The super-resolution model was implemented following the original paper (Johnson et al., 2016), which one is based on the image transformation model class defined by Radford et al. (2015).

First of all, the model is constructed with an input convolutional layer with a kernel size of 9, followed by four successive residual blocks (He et al., 2015).

In addition, each residual block is composed of two convolution layers with a kernel size of 3. The first one is activated with the ReLU function. Finally, the block's initial input (before the convolutional layers) is added to the output of the block—this is the residual part.

After that, just before the output convolutional layer, with a kernel of size 9, there are other convolutional layers. The number of these convolutional layers, with a kernel, of size 4 in our case, instead of 3, and a

stride of 1/2, depends on the upsampling factor. Indeed, two layers are used if the upsampling factor is set to 4, whereas three layers are set up if upsampling factor is set to 8.

Finally, on both layers of the network, each of the convolutional layers have the same number of filters with is 64, followed by spatial batch normalization and a ReLU non-linearity, excepted on the output layer, without batch normalization, and which uses a scaled Tanh to respect the pixels range.

3.2 Loss Network

As previously mentioned, the loss network is based on VGG-16 (Simonyan and Zisserman, 2014). We use pre-trained weights on ImageNet (Russakovsky et al., 2014) automatically downloaded from the PyTorch website. This VGG-16 instance is frozen and set to evaluation mode.

When called with generated and output images, the model retrieves the features from the two images. It can then compute the sum of the mean squared Euclidian distances (MSEs). This sum is the actual loss between the two images, used for backpropagation in the image transformation model.

3.3 Suggested Optimization

The goal of using the feature loss instead of the pixel-wise MSE loss was to compute the perceptual similarity score instead of the difference between each pixel, as previously established. But we wondered what could be the result when combining such a perceptual loss and the original pixel loss, by summing the two of them. We implemented a simple option to activate this potential optimization. The results will be put in comparison in Section 4.

4 Experiments

4.1 Setup

In order to conduct experiments, we first need a dataset to train our model. Johnson et al. (2016) used the Microsoft Common Objects in Context (MS-COCO) dataset (Lin et al., 2014) for training, but due to its large size and the associated training time, we had to find a more adequate dataset.

As an alternative, we chose DIV2K, a dataset introduced by Agustsson and Timofte (2017) in the technical report *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. DIV2K is a widely-used dataset for super-resolution tasks containing 1000 pairs of low-resolution (LR) and high-resolution (HR) images. Compared to the MS-COCO dataset, it has a reasonable size, making it more feasible for use in this study. More specifically, we only used the HR images dedicated to training: 800 images of variable sizes. An extract is visualizable in Figure 2.

As the model needs a specific type of training data, we did not use the LR images of DIV2K. Instead, we decided to process directly the HR images to create our custom HR-LR pairs of images.

To do so, the original HR images are cropped to 288×288 to create the HR patches. Then they are pre-processed according to the methods described in the paper (Johnson et al., 2016), to create the corresponding LR patches. It consists in blurring the cropped image with a Gaussian kernel of width $\sigma = 1.0$, and downsampling the result with bicubic interpolation.

Now that the dataset is pre-processed (see extract in Figure 3), the model and its dataset are ready to be used for training.

Figure 2: Visualization of 25 DIV2K HR train images.

4.2 Training

To train our model, we have defined some hyperparameters according to the original paper. Indeed, we use the Adam optimizer, defined the learning rate to 1×10^{-3} and also a batch size of 4.

Concerning the loss function, it is implemented as detailed in the section 3.2.

Finally, we experimented different values to determine, in our opinion, what is the best number of epochs to get the best results in the test phase. In our case, and in view of the various results, we have thus defined the number of epochs as 5.

4.3 Testing

To test our model, we focus on his ability to generate high resolution images from the low resolution images created previously. We can compare and discuss the differences between the generated image and the original high resolution one.

Moreover, as mentioned above, we also compare these results obtained with those using the pixel loss.

First of all, let's discuss our results with an upsampling factor of 4 and no pixel loss optimization.

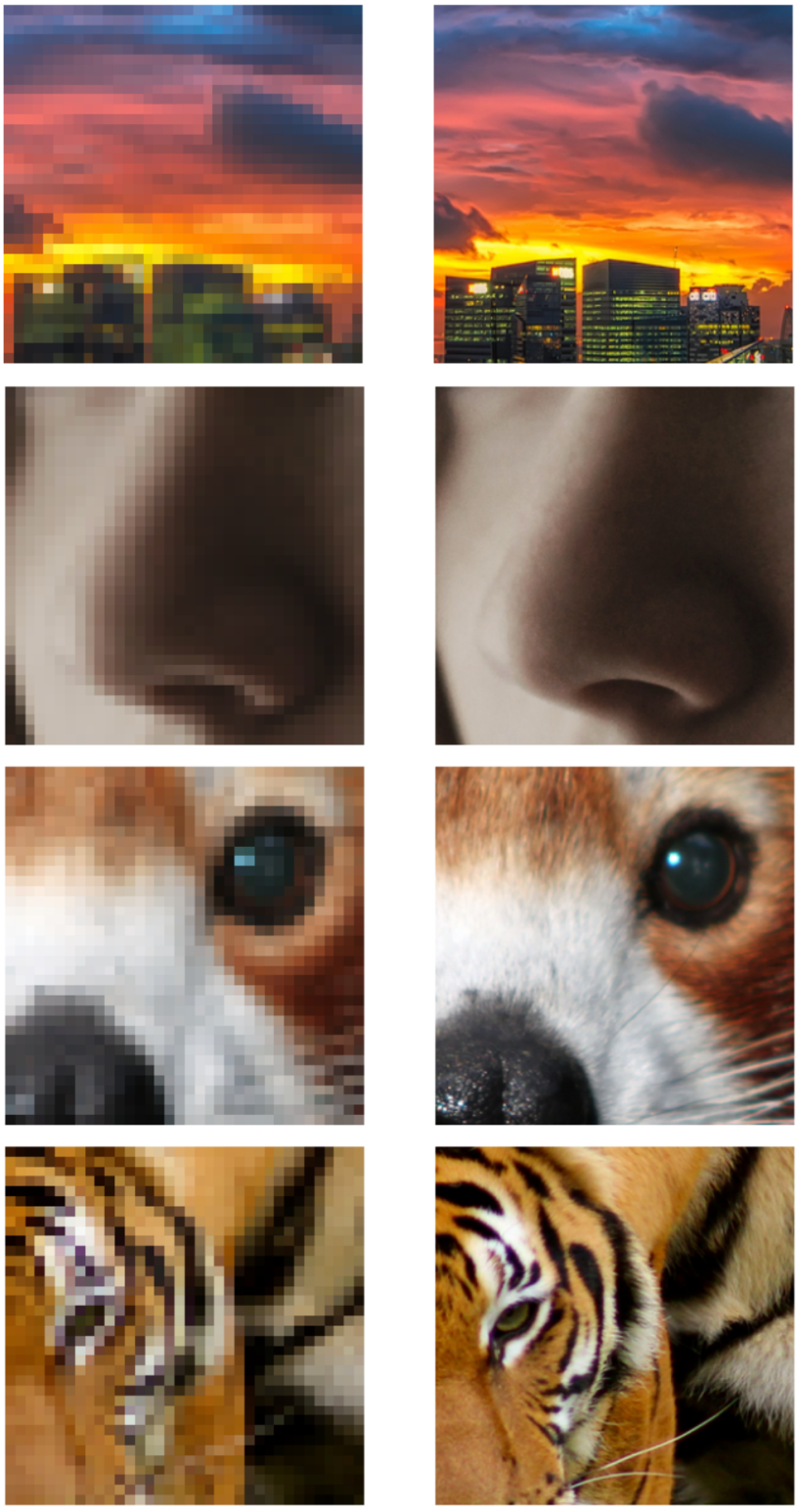

Figure 3: Visualization of 4 pairs of DIV2K HR train images (right) and their LR corresponding images (left), that underwent our pre-processing.

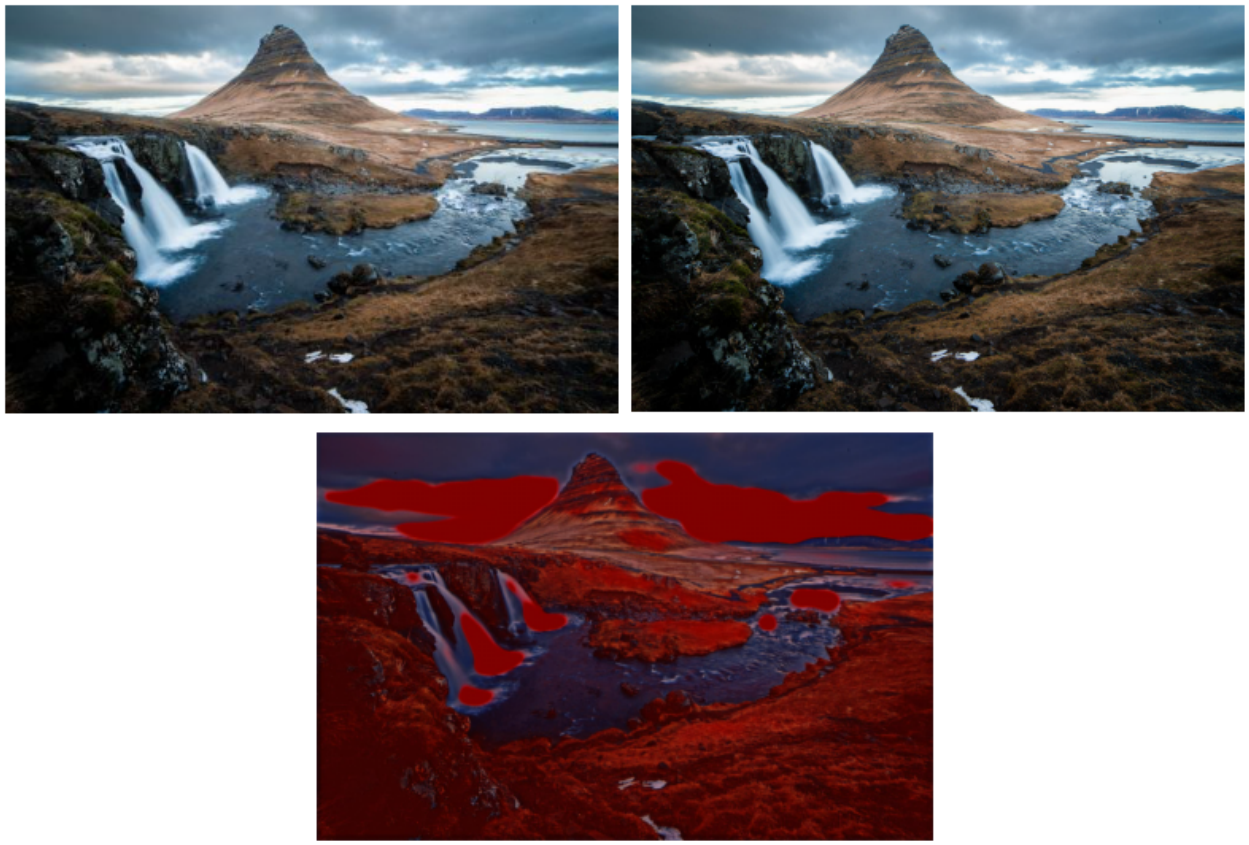

Figure 4: Visualization of a generated image (at the center) without pixel loss optimization, from the low resolution image (on the left), compared to the high resolution image (on the right), with an upsampling factor of 4.

On the first result in Figure 4, we can observe that, even if the generated image is no longer pixelated, as we can see in the associated low resolution version, the colors of the landscape, and the image in general, have changed. For example, some elements are strongly marked, like the waterfalls or part of the sky. On the other hand, the image still retains the content of the image.

In a second step, we can compare it with the pixel loss optimized version.

With the pixel loss optimization activated, we can notice that the new generated image obtained in Figure 5 is better than the previous one. Actually, this one presents colors of the original image and get a little bit closer to the original high resolution image below. However, the image is quite dark and seems rather blurred in relation to the original image.

After that, let's test and analyze the results with, this time, an upsampling factor of 8. We are going to compare the previous obtained image with an upsampling factor of 4 with this one.

Compared to the generated image with an upsampling factor of 4, we can observe, in Figure 6, that this one, with an upsampling factor of 8, has a lower quality. Indeed, we can hardly make out any details in the image. In this case, we loss more information than the previous reconstruction.

Finally, we compared the final generated image, with an upsampling factor of 8, with the one with an upsampling factor of 4, both generated using the pixel loss.

Figure 5: Visualization of 2 generated images : the one on the left obtained without pixel loss, and the one on the right, with pixel loss.

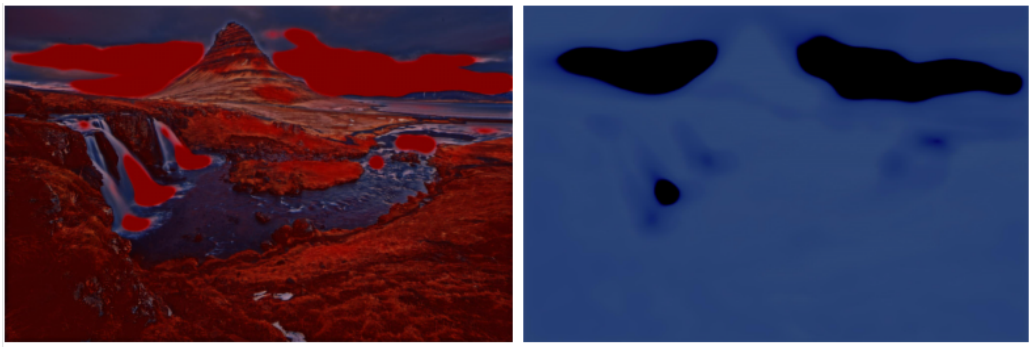

Figure 6: Visualization of 2 images generated without pixel loss : the one on the left obtained with upsampling factor 4, and the one on the right, with upsampling factor 8.

In this final comparison in Figure 7, image on the right is darker than the one on the left. This one is still blurred but, as in the comparison of images generated without using the pixel loss, it is difficult to see details that compose the image.

5 Environmental Impact

During the training phase, we have measured the environmental impact of our model. Indeed, as we can see in Figure 8, before the run, we have a 15 W consumption and a 72 MiB VRAM usage, whereas during the training, it increases to 126 W and 2,715 MiB.

6 Conclusion

To conclude, and as we can have discussed in the section 4.3, the high resolution images generated by our model are not exactly similar to the original high resolution images.

A first rational explanation, is in the fact that we do not use the same database. In our work, we have chosen the DIV2K dataset, because it has a reasonable size compared to MS-COCO dataset used in the original model, since we do not have the same devices and definite less computation power. As a result, our generations are not as accurate as their. Indeed, as the model based on the paper is not optimized for the training data we provide, the model doesn't learn the best weights which they may get in the original paper.

Figure 7: Visualization of 2 images generated with pixel loss : the one on the left obtained with upsampling factor 4, and the one on the right, with upsampling factor 8.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.141.03   Driver Version: 470.141.03   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  On   | 00000000:01:00.0 Off |                  N/A |
| 30%   38C    P8    15W / 170W |     72MiB / 12045MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    656727      G   /usr/lib/xorg/Xorg                 70MiB |
+-----------------------------------------------------------------------------+
```

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.141.03   Driver Version: 470.141.03   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  On   | 00000000:01:00.0 Off |                  N/A |
| 38%   66C    P2   126W / 170W |   2715MiB / 12045MiB |     98%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    656727      G   /usr/lib/xorg/Xorg                 70MiB |
|    0   N/A  N/A   2520032      C   python3                          2641MiB |
+-----------------------------------------------------------------------------+
```

Figure 8: Grapgic card consumption and memory usage before and during training.

A second source for these differences may be the definition of weights in the loss network (one for each loss value in the sum). In fact, due to the lack of additional information, we decided to set them all to 1, as explained in Section 2. However, the values chosen for those weights may have consequences in each loss computation. This might influence the training phase, and so make our model less efficient.

Nevertheless, in terms of the results obtained, we that we obtain a better result by using an upsampling factor of 4 and the pixel loss. In fact, the produced image is darker and has a blurred effect, but is of better quality than the other generated images. Moreover, we can consider that it is the one that comes closest to the original high resolution image.

Contributions

Description of each member contributions.

Iantsa Provost

- Model architecture and structure design—based on the paper.
- Image Transformation Network structure implementation.
- Loss Network implementation.

Lilian Rebiere-Pouyade

- Image Transformation Network layer blocks implementation.
- Experiments running and measurements.

Bastien Soucasse

- Model training and testing scripts implementation.
- Custom dataset design, implementation and testing.
- Training data pre-processing implementation.

Alexey Zhukov

- Image Transformation Network layer blocks implementation.
- Experiments running and measurements.

References

E. Agustsson and R. Timofte. NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1122–1131, 2017. URL https://doi.org/10.1109/CVPRW.2017.150.

L. A. Gatys, A. S. Ecker, and M. Bethge. A Neural Algorithm of Artistic Style, 2015. URL https://doi.org/10.48550/arXiv.1508.06576.

S. Gross and M. Wilber. Training and investigating residual nets, 2016. URL http://torch.ch/blog/2016/02/04/resnets.html.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition, 2015. URL https://doi.org/10.48550/arXiv.1512.03385.

J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution, 2016. URL https://doi.org/10.48550/arXiv.1603.08155.

T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft COCO: Common Objects in Context, 2014. URL https://doi.org/10.48550/arXiv.1405.0312.

A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015. URL https://doi.org/10.48550/arXiv.1511.06434.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2014. URL https://doi.org/10.48550/arXiv.1409.0575.

K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014. URL https://doi.org/10.48550/arXiv.1409.1556.