Unsupervised Visual Representation Learning by Context Prediction

Valentina Signor
valentina.signor@studenti.unipd.it

Marco Andrea Limongelli
marcoandrea.limongelli@studenti.unipd.it

Abstract

In this work we study the problem of image representation learning without human annotation. By following the principle of self-supervision we used several neural network architectures previously trained to solve a context prediction task. This task, which no requires manual labeling, consists in the use of the spatial context information through the extraction of a central patch and one of the surrounding patches from each image, and train a convolutional neural network to predict the position of the second patch relative to the first. We tested two different grid sizes for the patch extraction: 3x3 and 5x5. Furthermore, once this was done, we used these pre-trained models to perform the image classification task. Finally, for each architecture used, we compared the results of the pre-trained model against the corresponding model trained from scratch. To do all this, the datasets we worked on were Tiny ImageNet and PascalVOC2012.

1. Introduction

Nowadays, one of the biggest obstacles that *supervised learning* faces is the cost of manually annotating target variables in order to label each data item or image. In addition, this critical issue is becoming increasingly evident in the field of *Computer Vision*, where working with datasets containing hundreds of thousands of images is a crucial aspect. The search for alternative strategies to classical learning by supervision therefore becomes essential.

It is in this context that *unsupervised learning* takes its place. The attempt to use data without any annotation is indeed a natural response to the limitations posed by *supervised learning*. However, despite several decades of sustained effort, unsupervised methods have not yet been shown to extract useful information from large collections of life-size real images[2]. In particular,

"Without labels, how to understand what is represented in the image?" or even, *"How to write an objective function to encourage a representation to capture, for instance, objects, if none of the objects are labeled?"*.

From these questions comes our work. Through the use of *Convolutional Neural Networks (CNNs)* trained to identify the spatial location of one randomly extracted patch relative to another one (central) within an image[2], we attempt to capture its semantically relevant features. This representation is then through the use of *transfer learning*, used as a supervisory signal by *CNNs* to solve a *classification task*.
What we went to test is the importance of reasoning about the relative spatial relationships of objects and their parts, and how an understanding of context in some cases may be a possible alternative to human labeling.

2. Related Work

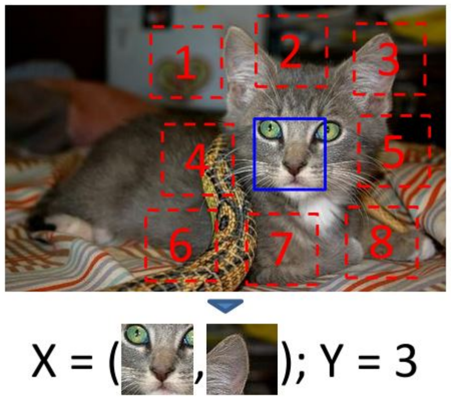

Figure 1. *Context prediction task* - patch grid *3x3*. *Source:[2] , pag. 2.*

To develop our project, we took inspiration from previous works shown in the papers [2] and [6].
Although both were a clear guidelines for figuring out what we needed to do, from a practical point of view we preferred to focus on developing something more similar to what was covered in the first [2], considering the second [6] an in-depth study to be carried out eventually in a next time.

The two papers deal the use of spatial context as a free source of a supervisory signal in order to obtain

a visual representation that is as rich and powerful as possible, to be reused for understanding image content.
In particular, in [2], the task at least in the initial part, corresponds to what we did. First it deals with the reconstruction of the supervisory signal (*context prediction task*) which is then used for the resolution of the *final task*. As shown in the ***Figure 1***, the image is divided into 9 patches, and by sampling a central patch and a random patch in one of 8 possible spatial configurations, a *machine learning model* tries to guess the spatial location of the second patch in relation to the first. Once the context representation is obtained, the purpose is then to solve the *final task*. With these premises, it is important to highlight that by the term *"final task"* we mean the final goal for which the *machine learning model* is trained, regardless of how the supervisory signal is obtained (i.e. through manual annotation or context representation). Some examples of these tasks are for instance *objects detection, image segmentation, image classification* and so on. The parameters obtained thanks to the *context prediction task* are therefore reused within a subsequent *machine learning model* for solve these tasks. In our project, we focused only on the task of *image classification*.

In [6] what is discussed is viewed more broadly than in the work described above. In practice, the *context prediction task* is no longer applied to just two patches but expanded through the localization of all patches in the image (like a puzzle).

Whether we talk about the *context prediction task* or the *final task*, everything is done by training and using different types of deep networks in order to compare the results. However, what we were most curious and tried to replicate in our work was the way in which these networks were organized: a dual architecture composed of two identical *CNNs*, one for each patch given as input (***Figure 2***).

3. Datasets

In our project we performed the same tasks (*context prediction* and *image classification*), in two different datasets, first using *Tiny ImageNet* [5] and then *PascalVOC2012*[3]. We chose these two datasets because they allowed us to have enough training images for each class [8].

3.1. Tiny ImageNet

Tiny ImageNet is a subset of the ILSVRC-2012 classification dataset. It consists of 200 object classes and for each object class it provides 500 training images, 50 validation images and 50 test images. All images were downsampled to 64x64 pixels. Training and validation images are provided with the class labels, while test images are released without class labels.

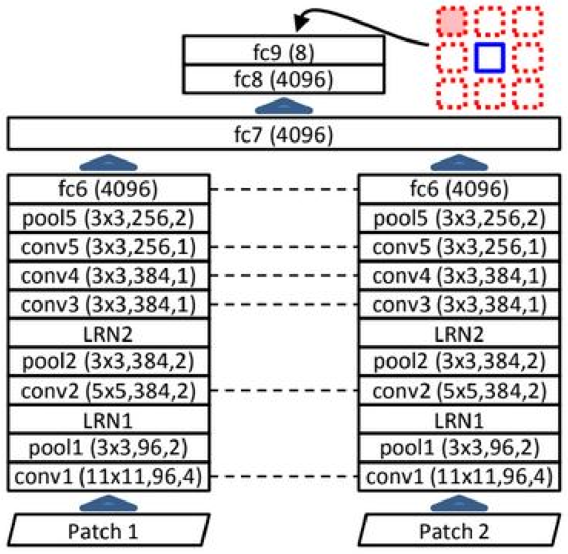

Figure 2. Framework used for solve the *context prediction task*. *Source:[2], pag. 3.* The figure shows a parallel architecture composed by two identical *CNNs*. Dotted lines indicate shared weights, *'conv'* stands for a convolution layer, *'fc'* stands for a fully-connected one, *'pool'* is a max-pooling layer, and *'LRN'* is a local response normalization layer. Moreover, the numbers in parentheses are kernel size, number of outputs, and stride.

In particular, with *Tiny ImageNet*, we used 10 classes and for each class we took all the training images and all the validation images, so a total of 550 images per class. Then, for each class we took 150 images for the *context prediction task* and 400 images for the *image classification task*. Thus, at the end, we worked with 1500 images for the first task and 4000 images for the second.

3.2. PascalVOC2012

PascalVOC2012 is an image classification dataset that consists of 20 object classes and 11'530 train and validation images. The classes in this dataset are unbalanced and the images sizes are variable but much larger than 64x64 of *Tiny ImageNet*.

In particular, with *PascalVOC2012*, we again used 10 classes and for each class we took all the training and validation images, reaching a total of 7654 images. Nevertheless, since the classes are not equally distributed among the data, we used a percentage approach to split the data between the two tasks. In particular, for each class the 30% of the images were assigned to *context prediction* and the other 70% to *image classification*. Thus, at the end, we worked with 2291 images for the first task and 5363 images

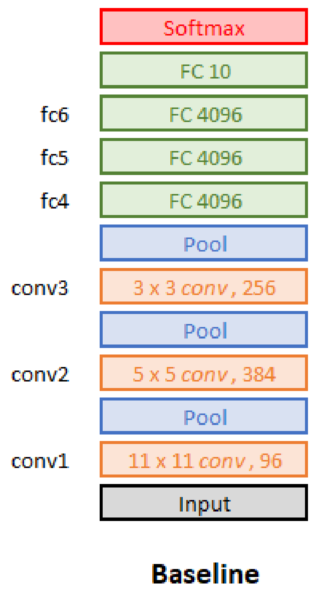

Figure 3. *Our Baseline* architecture for the *image classification task*. For the *context prediction tasks 3x3* and *5x5* the last fully connected layer size changes respectively to 8 and 24.

for the second.

4. Method

Our project is composed by two main tasks: *context prediction* and *image classification*. Both were carried out for *Tiny ImageNet* and *PascalVOC2012* datasets.
In practice, since in [2] the architecture used to build the *parallel CNNs framework* is only *AlexNet* [4], we decided to expand this idea also to other types of architectures, comparing then the obtained performance.
First of all, we built *Our Baseline*, which, as shown in ***Figure 3*** , is very similar to *AlexNet*, but with less depth (i.e. fewer *convolutional layers*). Moreover, we used a third architecture, more complex than the other two: *VGG16* [9]. Being deeper and *"heavier"* in terms of necessary parameters and computations, we indeed thought that it could be a useful counter-model, considering *Our Baseline* and *AlexNet* easier to handle.

For each of these architectures - *Our Baseline, AlexNet* and *VGG16* - we thus trained the correspondent model to solve the *context prediction task* and then used this *pre-trained model* to perform the *image classification task*. In particular, during the development of our project, the two tasks were structured in the same way for both datasets, net of changes related to the nature of the data itself.
The approach employed is discussed in detail below.

4.1. Context Prediction task

Once created the training and validation set, as explained in **§3**, we took care of creating the input to be given to the *parallel CNNs framework*. In practice, since the framework needs 2 patches of equal size to work, we divided the images into many small *local areas* and in particular, for the *Tiny ImageNet* dataset, we decided to test two different patches subdivision policies: 3x3 and 5x5 grids. Our aim was indeed to compare them to understand if, considering more or less large local areas, these would have influenced in some way the type of spatial information captured by the networks.

The images in *Tiny ImageNet* dataset are 64x64 pixels. We used a *patch size* of 15x15 pixels for the 3x3 grid task and a *patch size* of 12x12 pixels for the 5x5 grid task. Furthermore, within each input, in addition to the pixels reserved for the patch itself, we dedicated some pixels for the *gap* - implementation details. the minimum distance between two patches so that they are not adjacent - to prevent the algorithm from using some *"trivial shortcuts"* to provide the results. For both grid sizes, once the patches were extracted, they were resized to 96x96 pixels.
We operated in a similar way also on *PascalVOC2012* dataset but using only the 3x3 grid. Since this dataset contains images of different sizes, we first resized each image to 222x222 pixels and then we extracted the patches with a *patch size* of 60x60 pixels. Once extracted, the patches were resized to 222x222 pixels.

For this task, the number of epochs we considered for training our models was 60 for *Tiny ImageNet* and 20 for *PascalVOC2012*, with a batch size of 64 for both. Net of our resources and given the amount of images we had to work with, it seemed like a suitable choice considering also the effort to train a dual framework for each architecture.

4.2. Image Classification task

Once we obtained the *pre-trained models* in the way described above, we moved on to the real purpose of our project - understanding the content of images thanks to a *"self-produced"* supervision signal. In particular, we focused exclusively on understanding how this technique could be used in *classification tasks*, considering the latter a good starting point that can then eventually be extended to more complex tasks (e.g. *object detection*), which due to time constraints we were unable to deal with. Through the *transfer learning* method we therefore went to reuse the weights obtained from the resolution of the *context prediction task*.
In addition, it is important to observe that in [6] is show how completely re-train the whole net, instead of freezing weights in the *convolutional layers*, allow us to obtained

better results. Given these fact and for reasons of time, we decided to always completely re-train the network.
For training on *Tiny ImageNet* dataset we used 30 epochs, while for training on *PascalVOC2012* dataset we used 20, with a batch size of 64 for both.

4.3. Implementation details

We used *PyTorch* [7] to implement and train all the *CNNs* architectures and we developed our code using the *Google Colab Platform* [1] with integrated *GPU*.
Using *Tiny ImageNet* dataset, for each architecture and for both grid sizes, the training procedure took $\sim$15 minutes for both *context prediction task* and *image classification task*.
Instead, regarding *PascalVOC2012* dataset, for each architecture the training procedure took $\sim$45 minutes for both *context prediction task* and *image classification task*.
Note that the training time for *PascalVOC2012* dataset is greater than *Tiny ImageNet*, although it uses fewer epochs, because the architectures for *PascalVOC2012* use more parameters for the *classification part* of the *CNNs* and take more time in the *feature extraction part* since they process larger images (222x222).

The values given in this section are only those that we ultimately found to be the most suitable in terms of performace and hardware capability. Indeed, the experiments concerning these values will be treated in more detail in the section below.

5. Experiments

The experiments were the core of our project. Getting a good understanding of how this *parallel CNNs framework* worked - where it gave better results and where it gave worse results, and why - was been crucial to being able to understand if these *context pre-trained models* could be a viable technique to use in place of classical *supervised learning*. In details:

- **The main experiments** we performed involved: *Dataset, CNNs architectures, Patch generation grid, comparison between pre-trained models and models from scratch*;
- **Minor experiments** included: *Frozen layers, Number of epochs.*

5.1. Dataset

After a detailed reading of [2] and [6] we decided to address the problem by following a *bottom-up* approach. Once outlining the *goal* of the project, the *tasks* to be executed and the *way* in which everything was to be done, we moved on to the **choice of data** to be used during our work. Since we had to work with large datasets and wanted to find one that was well suited to our problem, we spent a not inconsiderable amount of time on this selection operation. In this preliminary stage, this attention proved necessary to avoid making mistakes due to our inexperience.
Our first choice fell on *Tiny ImageNet*[5]. Indeed, we thought that by presenting relatively simple images it might be a good option given the purpose of solving a *classification task*. Nevertheless, subsequently, given also the results we were getting from our experiments, we decided to extend our work to an additional dataset, *PascalVOC2012*[3]. Thanks its greater size and complexity, we indeed considered it a useful metric for comparison. This therefore led us to develop two paths equal in approach but independent in results, one for each dataset.

5.2. CNNs architectures and patch generation grid

Once we selected the data to work on, we moved on to developing our *CNNs pipeline* deciding to base it on three distinct architectures - **Our Baseline**, **AlexNet** and **VGG16** - in order to solve the *context prediction task*. We also tried to modify the **patch generation grid** from a 3x3 grid to a 5x5 grid, to see if the amount of information contained in the input could somehow affect the final results. Once we got the *pre-trained models* we then moved on to the *classification task*. By modifying the basic architectures in a similar way to what was done for the prediction of the context and thanks to the weights learned in it, we then went on to train our models for the *final task*.

5.3. With and without Transfer Learning

Subsequently, another aspect that we wanted to test was the incidence of the use of *pre-trained models* on the final result. We therefore decided to try to solve the *classification task* also in an alternative way - **without the use of transfer learning**, using simply randomly initialized weights in our *CNNs*. Our aim was indeed to see if and to what extent learning the context brings benefits compared to its version from scratch, in order to understand whether or not it could be a useful alternative to manual annotation.

5.4. Frozen layers and number of epochs

As already anticipated in **§4.2** also made some attempts solving the *final task* **freezing the convolutional layers** of the networks at different levels. However, net of the first results obtained and also thanks to what was stated in [6], we then decided not to further extend this part in favor of more profitable experiments.

The **number of epochs** has also been tested. In particular, as far as *Tiny Imagnet* dataset is concerned, we finally established a number of epochs of 60 for the *context prediction task* and 30 for the *classification task*. Indeed, as show in ***Table 4***, trying to raise the number of context

Architecture	Accuracy	Training Time
Our Baseline	22.8%	13 minutes
AlexNet	**23.2%**	13 minutes
VGG16	15.6%	16 minutes
Our Baseline [120 epochs]	**24.8%**	26 minutes

Table 1. *Context prediction* results, *3x3,Tiny ImageNet* dataset. In all ***Tables*** the models with improved performance are presented in **bold**. When not indicated in brackets, the number of epochs is 60 for *Tiny ImageNet* dataset and 20 for *PascalVOC2012*.

prediction epochs from 60 to 120 in *Our Baseline (patches 3x3)*, leads to a decreasing of the performance.
For *PascalVOC2012* dataset, the epochs were 20 for both tasks. Indeed, due to hardware limitations it was not possible to raise a number of context prediction epochs higher than 60. However, as show also in ***Table 3*** and ***Table 5***, with a greater number of epochs the performance tends to improve. Therefore, we can hypothesize that with more resources we would have obtained better results.

In **§5.5** and **§5.6** all the results of the experiments described above are explained in detail for a clearer understanding of what we have obtained.

5.5. Context Prediction results

5.5.1 Tiny ImageNet

Table 1 summarizes the results of the *3x3 context prediction task* on the *Tiny ImageNet* dataset. It is possible to note that the accuracy on this task is not very high using 60 training epochs, but it is still better than *random chance*: choosing randomly one of the 8 possible location for the patch, you predict correctly (1 / 8) = 12.5% of the times. For this reason we decided to try to train again one of that architecture, in this case *Our Baseline*, using 120 epochs. The results show that this last model perform slightly better although it is trained with the double of the epochs of the preceding ones.

Table 2 summarizes the results of the *5x5 context prediction task*. Also in this case the accuracy is not very good but is still better than *random chance*: choosing randomly 1 of the 24 possible location for the patch, you predict correctly (1 / 24) = 4.16% of the times. For this task we have decided to not continue with further experiments due to time constraints.

5.5.2 PascalVOC2012

Table 3 summarizes the results of the 3x3 *context prediction task* on the *PascalVOC2012* dataset. Also in this case it is possible to note that the accuracy on this task is not very

Architecture	Accuracy	Training Time
Our Baseline	6%	13 minutes
AlexNet	**8%**	13 minutes
VGG16	5.6%	16 minutes

Table 2. *Context prediction* results, *5x5, Tiny ImageNet* dataset.

Architecture	Accuracy	Training Time
Our Baseline	25.1%	38 minutes
AlexNet	**26.19%**	38 minutes
VGG16	21.8%	55 minutes
Our Baseline [60 epochs]	**34.4%**	126 minutes
AlexNet [60 epochs]	**27.6%**	128 minutes

Table 3. *Context prediction* results, *3x3, PascalVOC2012* dataset.

high using 20 training epochs, but it is still better than *random chance*. It is also possible to note that although we used less training epochs w.r.t. *Tiny ImageNet*, the results for *PascalVOC2012* are better for all the architectures. We have also tried to train again *Our Baseline* and *AlexNet*, but using 60 epochs. The results show that these last models perform better than the preceding ones.

5.6. Image Classification results

5.6.1 Tiny ImageNet

Table 4 shows the *image classification task* on *Tiny ImageNet* dataset. It is possible to see that none of the *pre-trained* models performs better than the models trained from scratch. It is curious the fact that the *pre-training*, in this case, not only it is useless but it is also harmful since some *pre-trained models* perform significantly worse than models trained *from scratch*, e.g. *AlexNet* from scratch 61.0% vs 53.6%.

5.6.2 PascalVOC2012

In ***Table 5*** it is possible to see the results of the *image classification task* on *PascalVOC2012* dataset. It is possible to note that, in this case, all the architectures except *Our Baseline* have a better accuracy on the *pre-trained* version than on the *from scratch* one. This implies that the *pre-training* can be useful although it needs more time.
We have also tried to do *transfer learning* using *Our Baseline* and *AlexNet* models pre-trained using 60 epochs in the *context prediction task* and we got a better result than the *from scratch* models. So, the problem with *Our Baseline* architecture was that the *context prediction model* wasn't trained enough to get a good result.

Architecture	Accuracy	Average Precision	Average Recall	Average F-Score	Training Time
Our Baseline scratch	**62.8%**	0.65	0.63	0.63	8 minutes
AlexNet scratch	61.0%	0.62	0.61	0.61	8 minutes
VGG16 scratch	55.4%	0.56	0.55	0.55	10 minutes
Our Baseline 3x3	**62.4%**	0.64	0.62	0.63	9 minutes
AlexNet 3x3	53.6%	0.55	0.54	0.54	8 minutes
VGG16 3x3	46.6%	0.56	0.47	0.44	10 minutes
Our Baseline 5x5	**46.2%**	0.46	0.46	0.46	8 minutes
AlexNet 5x5	45.6%	0.47	0.46	0.46	8 minutes
VGG16 5x5	43.2%	0.54	0.43	0.42	10 minutes
Our Baseline 3x3 [120 epochs]	55.6%	0.56	0.56	0.55	7 minutes

Table 4. *Image classification* results on *Tiny ImageNet* dataset.

Architecture	Accuracy	Average Precision	Average Recall	Average F-Score	Training Time
Our Baseline scratch	**43.4%**	0.44	0.40	0.41	42 minutes
AlexNet scratch	41.6%	0.45	0.41	0.42	45 minutes
VGG16 scratch	19.4%	0.28	0.14	0.08	58 minutes
Our Baseline 3x3	41.8%	0.44	0.38	0.40	42 minutes
AlexNet 3x3	**43.3%**	0.44	0.40	0.42	45 minutes
VGG16 3x3	25.9%	0.15	0.20	0.13	58 minutes
Our Baseline 3x3 [60 epochs]	**45.7%**	0.47	0.46	0.46	42 minutes
AlexNet 3x3 [60 epochs]	**43.62%**	0.44	0.41	0.42	45 minutes

Table 5. *Image classification* results on *PascalVOC2012* dataset.
In both ***Tables*** the models with improved performance are presented in **bold**. When not indicated in brackets, the number of epochs is 30 for *Tiny Imagnet* dataset and 20 for *PascalVOC2012* dataset. Only accuracy is highlighted since the other metrics reflect it.

6. Conclusion

Thanks to this work it was possible for us to deepen the problem of *image representation learning without human annotation*. In particular, there are two aspects that we believe have emerged from this project:

- **Respect of the *Occam's razor principle*.** Indeed, in all our experiments the *VGG16* has always presented lower performances in favor of *Our Baseline* and *AlexNet*;
- **Influence of the *number of epochs* on performance of the models**. In particular, for *Tiny ImageNet* dataset the *pre-training* with the number of epochs we used it proved to be not useful but harmful. In contrast, for *PascalVOC2012* dataset the *pre-training* was useful even if the accuracy grows too slowly.

Unfortunately, due to the tools at our disposal we have not had the opportunity to train the models for more epochs because *Google Colab*[1] once you reach a certain usage time blocks the computation. In any case, it is important to underline what the ***Figure 4*** shows. What can be observed is that when you apply *Our Baseline* for solve the *context prediction task* on *PascalVOC2012* dataset, the gap between train and validation loss tends to not increase as the epochs grow (i.e. the model is not *overfitting* the data). In the light of these results, we can conclude that a possible extension of this work could be to train the *context prediction models* for a bigger number of epochs and make a new comparison of the results. We argue it is possible to obtain better performance than those obtained by us.

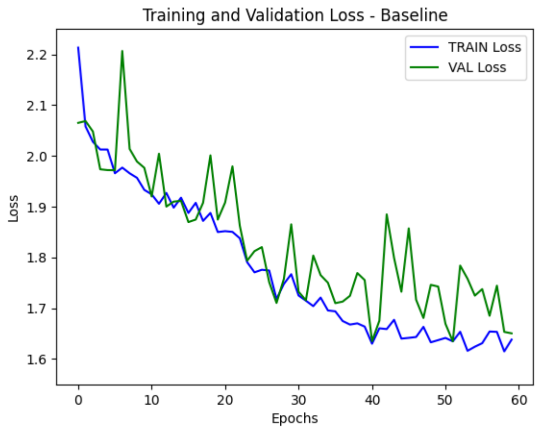

Figure 4. Loss plot for the *context prediction task* obtained from *Our Baseline, 3x3, 60 epochs on PascalVOC2012* dataset. Also *AlexNet model* with this setting presents a very similar behavior.

References

[1] Ekaba Bisong and Ekaba Bisong. Google colaboratory. *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pages 59–64, 2019.

[2] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[3] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[5] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[6] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI*, pages 69–84. Springer, 2016.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[8] Saleh Shahinfar, Paul Meek, and Greg Falzon. "how many images do i need?" understanding how sample size per class affects deep learning model performance metrics for balanced designs in autonomous wildlife monitoring. *Ecological Informatics*, 57:101085, 2020.

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.