CS230

# AlphaNUT: Nut/Screw Classifier via CNN Spring 17-18

**Sean Choi**[*]
Stanford University
Department of Electrical Engineering
yo2seol@stanford.edu

**Ryan Kwon**[†]
Stanford University
Department of Mechanical Engineering
ryankwon@stanford.edu

## Abstract

The conventional method of finding a right screw bits to use for a screw is a simple trial and error process, where the user tries a series of screw bits that seem like an appropriate fit for a target screw. As a result, the process of inserting or removing a screw becomes inefficient due to less than optimal fit of the screw bit to the screw head, and even worse, can result in damaging the screw in use.

In this paper, we present our methods for automating the process of finding the right screw bit for the screw to be used through image classification using convolutional neural network (CNN) model. We obtained a training data consisting of 2-dimensional images of 13 different types of screws, augmented the data through image data augmentation methods, and built a customized CNN models based on existing CNN models that are built on the ImageNet database. In this paper, we compared and analyzed multiple existing CNN models that we build the final model upon, and present a set of models that fit best for different use cases. We name our model AlphaNUT and we have shown that we were able to achieve an accuracy of around X% on a 13 class screw classification task.

## 1 Introduction

From furnitures, picture frames, home appliances, electronics and many more, millions of users are actively assembling and disassembling these household items using household tools such as screw drivers or power drills. The process of building and disassembling household items involves interacting with numerous types and sizes of screws, which ultimately requires finding the right screw bit to use for the screw of interest. This process has been historically performed through trial and error process, where the user simply grabs a screwdriver that seem as if it fits best to the screw of interest. As an example, an iPhone 6 involves 6 different types of screws for full disassembly, and the process of finding the right screw drivers for each type of the screw is extremely painful and erroneous in many cases. Also, engaging the screw with a power drill or a screwdriver while using inadequately fitting screw bit, can not only cause inefficient application but critically damage the screw head as can be seen in Figure 1. The cost and annoyance caused by such problem are significant and can be highly mitigated if proper screws bits are identified for the respective screws.

With the advent of large, accessible data set and increasing ease and availability of processing power, the field of machine learning, especially deep learning, has gained an enormous attention and have made tremendous improvements in recent years [LeCun et al., 2015, Guo et al., 2015]. Among

[*]Website: http://stanford.edu/yo2seol
[†]Website: http://micromachine.stanford.edu/?p=personal&user=ryankwon

Figure 1: Example of broken screws from applying pressure using screw bit with inappropriate fit.

numerous applications of deep learning, image recognition applications using convolutional neural network (CNN) have shown many successes[Simonyan and Zisserman, 2014, Krizhevsky et al., 2012]. Thus, given CNN's proven track records on image recognition, we experimented on applying deep learning to provide a simple, yet, effective solution to the problem of finding the correct screw bits. The goal is classify the type of the screw using a combination of screw image and CNNs, and then recommend a correct screw bit or a screwdriver to the user with high probability. The overall process of detecting the type of the screw is as follows. First of all, the input to our CNN model is a set of 2 dimensional images of the top of the screw. We first take an image of the top of the screw, we then pre-process it to be a specific size and finally augment a single image into a multiple image to be used for classification for higher accuracy. Then, we then use a set of customized CNN models to output a probability of the screw belonging to each of the known class of screws, which we can map to an appropriate screw bit to use and ultimately informs the user to use the correct screw bit. By having such tool on our hand, we significantly increase the effectiveness of how a user assembles any household product.

## 2 Related work

Field of image recognition has been around for decades. However, since 2010, ImageNet [imagenet] database has become the standard on measuring the performance of image recognition tasks for neural networks. Since then, there have been numerous papers, such as Guo et al. [2015], Simonyan and Zisserman [2014], Szegedy et al. [2015], Alemi [2016], Chollet [2016], Howard et al. [2017] that attempted to broaden the field of image classification via CNNs. These papers attempted at creating new CNN architectures to improve performance, reduce training time or model sizes. Besides these works, there are countless works, such as Donahue et al. [2014] that applies these CNNs for performing image classification on various domains. Our work differs from all other previous work, because no one has yet attempted to use image classification on the domain of screw and bolt recognition. Our dataset is completely unique, as it comes from our own customized hardware.

## 3 Dataset and Features

Identification of screws involve not only the shape (e.g. Philips, Flat, Hex) but also the exact size of the screws (e.g. PH0, PH1, PH2). The size aspect provides a unique challenge to this work by necessitating a new dataset from scratch. Importing screw/nut images only gives the shape information but does not contain the exact size information due to the nature of the shots taken such as inconsistent focal length and sizing. To overcome this problem, we built a custom camera solution that has a fixed focal length and a casing that holds the camera module in the fixed distance from our target - screws/nuts. This platform allowed us to take images that contains not only the shape of the screws but also the dimensions. Moreover, screws are found in diverse backgrounds such as wood, metal, edges/corners and to make our model learning such conditions, we mimicked certain scenarios. As for the overall dataset, we included 12 different types of screws with varying shape/size as shown in Figure 2.

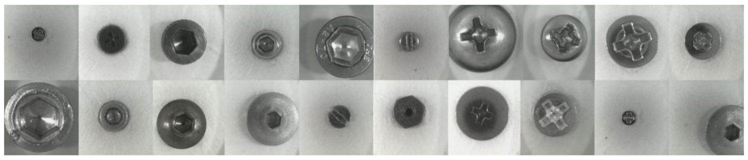
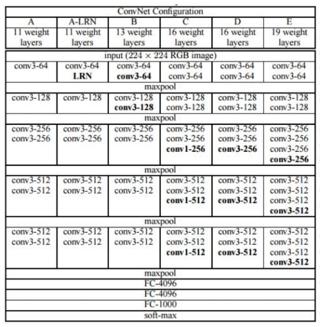
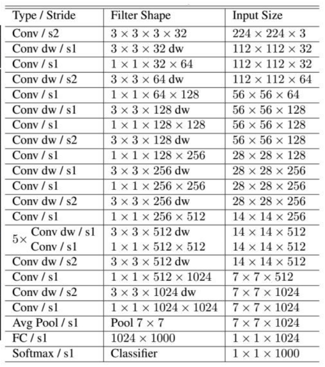


Figure 2: 12 Classes of Screws/Nuts.



| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |



| Type / Stride | Filter Shape | Input Size |
|---|---|---|
| Conv / s2 | $3 \times 3 \times 3 \times 32$ | $224 \times 224 \times 3$ |
| Conv dw / s1 | $3 \times 3 \times 32$ dw | $112 \times 112 \times 32$ |
| Conv / s1 | $1 \times 1 \times 32 \times 64$ | $112 \times 112 \times 32$ |
| Conv dw / s2 | $3 \times 3 \times 64$ dw | $112 \times 112 \times 64$ |
| Conv / s1 | $1 \times 1 \times 64 \times 128$ | $56 \times 56 \times 64$ |
| Conv dw / s1 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 128$ | $56 \times 56 \times 128$ |
| Conv dw / s2 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 256$ | $28 \times 28 \times 128$ |
| Conv dw / s1 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 256$ | $28 \times 28 \times 256$ |
| Conv dw / s2 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 512$ | $14 \times 14 \times 256$ |
| $5\times$ Conv dw / s1<br>Conv / s1 | $3 \times 3 \times 512$ dw<br>$1 \times 1 \times 512 \times 512$ | $14 \times 14 \times 512$<br>$14 \times 14 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 512$ dw | $14 \times 14 \times 512$ |
| Conv / s1 | $1 \times 1 \times 512 \times 1024$ | $7 \times 7 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 1024$ dw | $7 \times 7 \times 1024$ |
| Conv / s1 | $1 \times 1 \times 1024 \times 1024$ | $7 \times 7 \times 1024$ |
| Avg Pool / s1 | Pool $7 \times 7$ | $7 \times 7 \times 1024$ |
| FC / s1 | $1024 \times 1000$ | $1 \times 1 \times 1024$ |
| Softmax / s1 | Classifier | $1 \times 1 \times 1000$ |

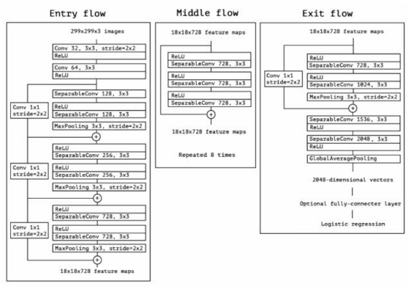


Figure 3: Architectures of VGG, MobileNet and Xception (left to right).

## 4 Methods

We built multiple models with varying properties, enabling us to choose a specific model depending on a specific use case. The models employ transfer learning [Wikipedia] extensively thus are built upon the weights obtained from various well-known CNNs models that are pre-trained using ImageNet dataset. The set of CNN models used to obtain the weights are as follows: VGG16, VGG19 [Simonyan and Zisserman, 2014], MobileNet [Howard et al., 2017], InceptionResnetV2 [Alemi, 2016], InceptionV3 [Szegedy et al., 2015], and XCeption Chollet [2016]. For the interest of space, the architecture for the subset of these models can be seen in Figure 3. Each models is characterized by what metrics it wants to optimize. VGG 16 and VGG 19 is built for accuracy, but it is very computationally expensive. In order to solve the problem Inception networks were built to reduce computational requirements by adding sparser connections between layer and also added different type of convolutions to capture different aspects of the input. Xception is an improvement over Inception with the aim to split the cross-channel correlation and the spatial correlation of an image. MobileNet is an optimized version of Xception to run on mobile.

The main reason for using the pre-built and existing models are as follows. First of all, given that we have a limited amount of manually obtained data, transfer learning helps us to bootstrap our model. Secondly, many of the earlier layers in the CNN models tend to learn the generic features of an image, such as where the boundary exists or spatial correlations within a channel, thus we can avoid creating new models to learn the same features again. Lastly, training deep models require large computational power and lengthy training time, which we hoped to avoid.

Given each of the CNN models, we removed final set of layers that are responsible for completing the final image classification tasks for the ImageNet database. The effect of this is that the output of each model now consists of the weights that can be used as an input to a final layer of choice. Thus, we pass our training into the pre-trained model to obtain the weights for each of the images. Given the pre-trained weights, they are used as an input vector to our softmax logistic regression model [] with various hyperparameters, such as type of regularization, regularization parameter and stopping criteria. We also experimented with two different types of regularization, L1 and L2, where the L2 regularization adds $||\theta||_2^2$ to the loss function, making the loss function

$$||\theta||_2^2 + C \sum_{i=1}^{n} \log \prod_{l=1}^{k} \left( \frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \right)^{1\{y^{(i)}=l\}}$$

and L1 regularization adds the $||\theta||_1$ regularization term instead. Softmax regression then finds the final weights that minimizes the given loss function, which we use as our final model. Note that larger $C$ here has an effect of reducing regularization.

| | vgg16 | vgg19 | resnet | inceptionv3 | inception-resnet-v2 | mobilenet | xception |
|---|---|---|---|---|---|---|---|
| Input Size | 4096 | 4096 | 2048 | 51200 | 38400 | 50176 | 2048 |

Figure 4: Size of the feature output from each mode.

Finally, Figure 4 we show the size of the weights that are outputted from each model. This acts as our input ans well as the features to our final model.

## 5 Experiments/Results/Discussion

In this section, we discuss our methods for experiments, as well as our results of the experiments. We first discuss the experimental setup. We have two set of experimental setup consisting of Google Colaboratory [Google] and a Linux machine. Google Colab notebook was mainly used for retrieving the weights from the existing CNNs models, which was accelerated by utilizing Tesla K80 GPU [Nvidia] offered by Google Colab. The Linux machine, which houses an Intel i7-8700K, 32GB of RAM, Nvidia 1080ti GPU and Ubuntu 18.04, was used to train the last softmax regression layer and also to obtain experimental data. The CNN models trained on ImageNet were obtained via Keras application [Keras] library. Using the Keras, we preprocessed our training set to the appropriate size for each of the model. Then, we passed the training data to the model to obtain the weights. Once we have obtained the weights, we moved it to a local server, where we built the last softmax regression layer using scikit-learn [scikit learn] library with SAGA [Defazio et al., 2014] solver. SAGA solver does not allow us to change the learning rate, so we used a constant learning rate as given. The paper on SAGA provides more detail on what learning rate it chooses based on the data.

We now discuss our initial results for each of the model. First, we show the build time for the weights from each of the model for different regularization method.

| Regularization Type | vgg16 | vgg19 | resnet | inceptionv3 | inception-resnet-v2 | mobilenet | xception |
|---|---|---|---|---|---|---|---|
| L2 | 59s | 73s | 33s | 1459s | 814s | 842s | 17s |
| L1 | 173s | 209s | 83s | 1691s | 1569s | 1213s | 79s |

Figure 5: Build time for each model using L1 and L2 regularization method. The stopping threhold is 0.01 and the regularization parameter is 1.0.

We can see that the build time increases linearly with respect to the size of the weights outputted by each model. We now show the loss curve for training each of the models.

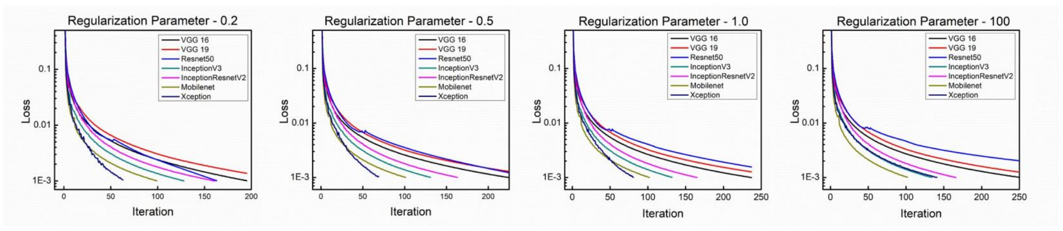
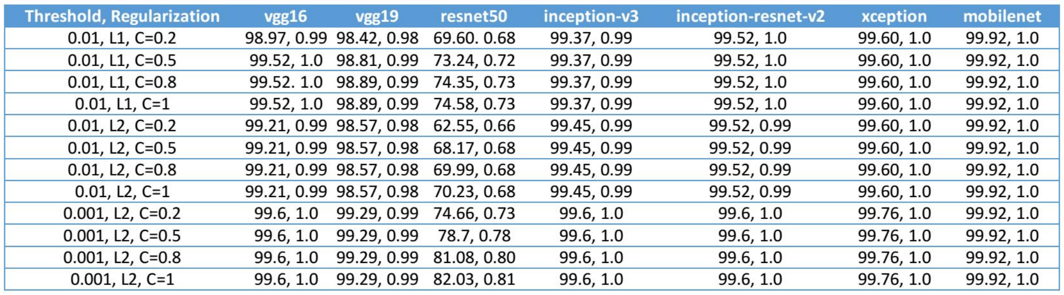


Figure 6: Loss curve of our models based on various regularization parameters.

We can see that mobilenet and xception tend to converge the fastest toward the threshold. We now discuss the accuracy and F1 score for each model on different regularization methods. We can see that resnet performs the worst across the board and mobilenet performs the best. To visualize this, here are the confusion matrix for resnet and mobilenet for 0.001 threshold and 1.0 regularization. After analyzing the faulty predicted images in both models, we concluded that they were from similar screws with slightly different sizes. Looking at the accuracy chart, we can see that as regularization increases, accuracy decreases. Finally, we can see that L2 regularization performs a little bit better than L1 regularization for most models, but also results in much higher training time. We hypothesize



| Threshold, Regularization | vgg16 | vgg19 | resnet50 | inception-v3 | inception-resnet-v2 | xception | mobilenet |
|---|---|---|---|---|---|---|---|
| 0.01, L1, C=0.2 | 98.97, 0.99 | 98.42, 0.98 | 69.60. 0.68 | 99.37, 0.99 | 99.52, 1.0 | 99.60, 1.0 | 99.92, 1.0 |
| 0.01, L1, C=0.5 | 99.52, 1.0 | 98.81, 0.99 | 73.24, 0.72 | 99.37, 0.99 | 99.52, 1.0 | 99.60, 1.0 | 99.92, 1.0 |
| 0.01, L1, C=0.8 | 99.52. 1.0 | 98.89, 0.99 | 74.35, 0.73 | 99.37, 0.99 | 99.52, 1.0 | 99.60, 1.0 | 99.92, 1.0 |
| 0.01, L1, C=1 | 99.52, 1.0 | 98.89, 0.99 | 74.58, 0.73 | 99.37, 0.99 | 99.52, 1.0 | 99.60, 1.0 | 99.92, 1.0 |
| 0.01, L2, C=0.2 | 99.21, 0.99 | 98.57, 0.98 | 62.55, 0.66 | 99.45, 0.99 | 99.52, 0.99 | 99.60, 1.0 | 99.92, 1.0 |
| 0.01, L2, C=0.5 | 99.21, 0.99 | 98.57, 0.98 | 68.17, 0.68 | 99.45, 0.99 | 99.52, 0.99 | 99.60, 1.0 | 99.92, 1.0 |
| 0.01, L2, C=0.8 | 99.21, 0.99 | 98.57, 0.98 | 69.99, 0.68 | 99.45, 0.99 | 99.52, 0.99 | 99.60, 1.0 | 99.92, 1.0 |
| 0.01, L2, C=1 | 99.21, 0.99 | 98.57, 0.98 | 70.23, 0.68 | 99.45, 0.99 | 99.52, 0.99 | 99.60, 1.0 | 99.92, 1.0 |
| 0.001, L2, C=0.2 | 99.6, 1.0 | 99.29, 0.99 | 74.66, 0.73 | 99.6, 1.0 | 99.6, 1.0 | 99.76, 1.0 | 99.92, 1.0 |
| 0.001, L2, C=0.5 | 99.6, 1.0 | 99.29, 0.99 | 78.7, 0.78 | 99.6, 1.0 | 99.6, 1.0 | 99.76, 1.0 | 99.92, 1.0 |
| 0.001, L2, C=0.8 | 99.6, 1.0 | 99.29, 0.99 | 81.08, 0.80 | 99.6, 1.0 | 99.6, 1.0 | 99.76, 1.0 | 99.92, 1.0 |
| 0.001, L2, C=1 | 99.6, 1.0 | 99.29, 0.99 | 82.03, 0.81 | 99.6, 1.0 | 99.6, 1.0 | 99.76, 1.0 | 99.92, 1.0 |

Figure 7: Accuracy and F1 score for the models with varying threshold and Regularization.

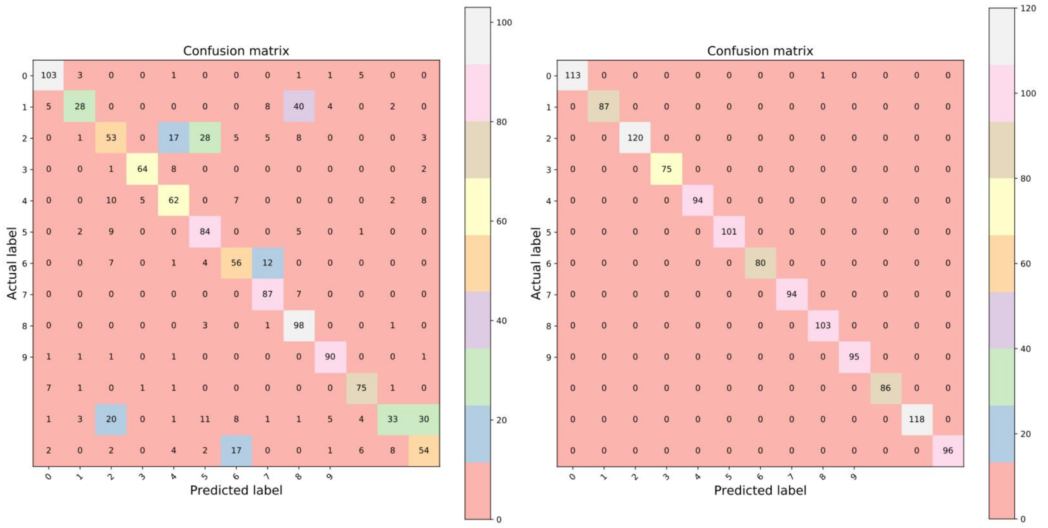


Figure 8: Confusion matrix comparison for resnet and mobilenet

that the high accuracy for all of the models result from our method of collecting data and augmenting it and our model has over fitted the training data. All of training our data comes from black and white images taken with a standardized camera at a standardized focal length. Also, as we increase regularization, we see the accuracy decrease, which shows a sign that training data is similar to test data. We believe that as we increase the type of test images, the accuracy and F1 metrics would most likely to change.

## 6 Conclusion/Future Work

In this paper, we built AlphaNUT, a CNN based screw classifier, to aid millions of users who are actively interacting with screws in their daily lives. AlphaNUT a 12 class classifier based on weights obtained from various existing CNN models. We have shown that we perform at a very high accuracy, precision and recall on our test data set. However, using augmentation on small number of test data to generate more test data may have been problematic and may result in higher result than expected. Thus, for our future work, we plan to gather more training data via manual collection and also fine-tune our model further. Also, we initially tried to use Google Colab engine for building our models with little success due to its slowness in retrieving data from our Google Drive. We can revisit that in the future and try to use other Google storage solutions.

## 7 Contributions

Contributions between Ryan and Sean was fairly equivalent. We both participated in building the camera module to collect training and test data, augmenting data and discussion about building the models. We both participated equally in preparing the poster and the final paper. If we have to emphasize which member was responsible for which deliverables, we summarized it in the list below.

- Ryan: Managed data generation process, class selection, data clean up, and data augmentation process.
- Sean: Managed model selection, writing scripts for building models, experimentation and hyper parameter tuning.

## References

A. Alemi. Improving inception and image classification in tensorflow, 08 2016. URL `https://ai.googleblog.com/2016/08/improving-inception-and-image.html`.

F. Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016. URL `http://arxiv.org/abs/1610.02357`.

A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014. URL `http://arxiv.org/abs/1411.4389`.

Google. Google colab. URL `https://colab.research.google.com/`.

Y. Guo, Y. Liu, A. Oerlemans, S.-Y. Lao, S. Wu, and M. S. Lew. Deep learning for visual understanding: A review. 187, 11 2015.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL `http://arxiv.org/abs/1704.04861`.

imagenet. Imagenet. URL `http://image-net.org/about-overview`.

Keras. Keras applications. URL `https://keras.io/applications/`.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436 EP –, 05 2015. URL `http://dx.doi.org/10.1038/nature14539`.

Nvidia. Tesla k80 gpu. URL `https://www.nvidia.com/en-us/data-center/tesla-k80/`.

scikit learn. scikit-learn. URL `http://scikit-learn.org/stable/index.html`.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL `http://arxiv.org/abs/1512.00567`.

Wikipedia. Transfer learning. URL `https://en.wikipedia.org/wiki/Transfer_learning`.