Hunting for Naval Mines with Deep Neural Networks

Daniel Gebhardt, Keyur Parikh, Iryna Dzieciuch,
Michael Walton, and Nhut Anh Vo Hoang
Space and Naval Warfare Systems Center Pacific
Email: {daniel.j.gebhardt, keyur.parikh, iryna.dzieciuch, michael.walton, nhutanh.n.vohoang}@navy.mil

***Abstract*—Explosive naval mines pose a threat to ocean and sea faring vessels, both military and civilian. This work applies deep neural network (DNN) methods to the problem of detecting mine-like objects (MLO) on the seafloor in side-scan sonar imagery. We explored how the DNN depth, memory requirements, calculation requirements, and training data distribution affect detection efficacy. A visualization technique (class activation map) was incorporated that aids a user in interpreting the model's behavior.**

We found that modest DNN model sizes yielded better accuracy (98%) than very simple DNN models (93%) and a support vector machine (78%). The largest DNN models achieved <1% efficacy increase at a cost of a 17x increase of trainable parameter count and computation requirements. In contrast to DNNs popularized for many-class image recognition tasks, the models for this task require far fewer computational resources (0.3% of parameters), and are suitable for embedded use within an autonomous unmanned underwater vehicle.

I. Introduction

Mine clearing operations often use side-scan sonar imagery from unmanned underwater vehicles (UUV) to locate mine-like objects (MLO) on the seafloor. A human operator is employed to analyze the collected imagery, which can incur significant time and financial cost, and be susceptible to limitations of human vigilance such as drowsiness or nervousness. An eventual goal is to replace the operator attention-intensive methodologies with autonomous systems. Sensor data transmission rates are severely limited in an undersea environment, and thus an autonomous system would require sensor processing to be completed on-board the vehicle, rather than on a remote high-performance computing system, if operators require real-time sensor results.

There have been many methods proposed to perform MLO identification. These methods often apply signal processing and from computer vision (CV) algorithms, which may make use of local visual features, hand-crafted features, or features learned from data.

Model-based approaches often use *a priori* knowledge of the mine, sonar return characteristics, and shapes of the resulting highlight and shadow regions. One such method uses a discrete shadow/highlight segmentation step with Markov random fields [1] before classifying a target based on those features. Another [2] uses geometric associations between segmented shadows and highlights to perform classification.

Data-driven approaches combine learned low-level features, such as Haar-like filters, with a classification model which learns its parameters from training data. One such method used the AdaBoost algorithm to select features and build an ensemble model of weighted weak classifiers using boosting. [3]. Another method uses a sparse autoencoder to learn a latent space representation of Haar features [4]. The combination of Haar features and learned features from a human operator's brain electroencephalogram (EEG) has been evaluated [5].

This work explores automatic MLO detection using recent advances in deep neural networks (DNN), both in efficacy and computational requirements. In recent years, DNNs have risen to the state-of-the art on a variety of object detection and classification tasks. For example, human-level performance was surpassed by a DNN on ImageNet, a difficult 1000-class dataset consisting of natural images [6]. In visual domains, the input of such models is most often raw pixel data from images; usually pre-defined features [7], [8] are not used.

DNNs along with many other machine learning methods, often necessitate training on voluminous datasets. These data-intensive methods are in contrast to specific model-based approaches, such as hand-selected features relying on expert-knowledge to craft distance metrics, signal-to-noise ratio, area, Haar-like features, or spectral matched filters. With enough data available, a DNN will learn what features of the input data are important to produce the proper output, and their "deep" nature (more than 2 layers) enables the representation of more complex compositions of features. There are disadvantages to data-driven machine learning, such as the need of many examples, but the advantage is that retraining with additional data (e.g., a new seafloor texture or clutter) does not require expert-knowledge to refactor a custom model.

While DNNs produce very high efficacy, the computation required can be quite high compared to some other methods. Training the model is certainly the most demanding on compute hardware, but even *inference* mode (post-training) can tax an embedded system limited by size, weight, and power (SWaP) requirements. Therefore, we evaluate the computation required for various model configurations to help inform a system architect.

II. Methods

A. Dataset

Our dataset is derived from prior MLO detection work [3]. Briefly, the data is generated from REMUS vehicles equipped with 900-kHz Marine Sonic side-scan sonar sensors. An out-and-back path is followed by the vehicle where one or more

MLOs are present. From each path, a 1024 x 1000 pixel image is generated, and MLOs were labeled by a trained operator to generate ground truth.

We took each of these labeled large images and cropped out the MLO into a 84 x 84 pixel patch to generate *positive* examples, and used neighboring patches of seafloor background as *negative* examples. An image patch is the input vector, as raw pixel intensities are scaled to the range $[0, 1]$. Figure 1 illustrates the patch creation from a labeled mine location and surrounding not-mine background. We do not preprocess the images, and allow the artifacts of collection to be present. These artifacts include the "black" area directly under the vehicle, noise induced by movement, and 180-degree turn discontinuities. Preprocessing may result in better results, but we wish to investigate the DNN's capabilities in learning these artifacts natively.

The ratio of background patches to MLO patches in the training dataset can be specified, where additional *negative* (background) examples are chosen around the MLO. For a 1:1 ratio, the training dataset contains 2163 patches each of positive and negative labels, while the test dataset contains 2569 patches each. Additional studies may vary this ratio to provide a greater diversity of negative (background/clutter) examples to the training process.

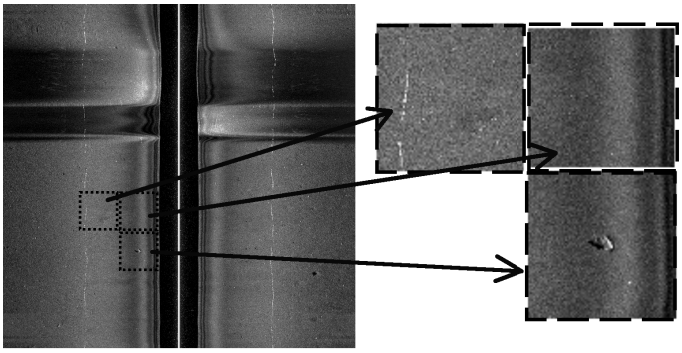

Fig. 1. Sonar imagery patch generation for a MLO and two background patches.

B. Deep Neural Network Model

A deep neural network is fundamentally a non-linear function approximator, where the model parameters (weights) are fit to training data. The connection to biological neural operation is weak, at best. The space of possible DNN configurations is extremely large, but generally a convolutional neural network (CNN) has been shown to be very effective for object classification in the visual domain [9]. We refer to referenced material for detailed descriptions of a CNN's arrangement and function.

Optimal configurations for a particular application are highly dependent on the details of that application. We examine one axis of adjustment—network depth and its growth of parameters and computation requirements. With a two-label classification problem of relatively simple features, we suspect a much smaller network compared to those for the Imagenet competition will be sufficient. Layers are based loosely on the *VGG* [10] arrangement, which emphasizes small filter spatial dimensions of many repeated layers (as opposed to larger windows and fewer layers), with a growth of filter-count per layer as depth increases. These CNN configurations are shown in Table I, where each is given an identifier such as *L3*. Layer descriptions are shorthand for layer types; *c16* is a convolutional layer with 16 filters with a 3x3 pixel window; *mp* is a max-pool with 2x2 pixel window; *gap* is global average pool; *fc2* is a fully-connected softmax function with 2 output classes. Briefly, the softmax function outputs a probability between $[0, 1]$ for each class, where the sum of class probabilities is 1. The nonlinearities after each layer (activation function) were rectified linear units.

Instead of flattening the output of the last convolution to a feature vector, as in *VGG* and others, our model averages across the spatial dimension of the last convolution. This method is commonly referred to as Global Average Pooling (GAP) [11]. GAP is a simple technique to providing regularization of the model (to prevent overfitting, and improve generalization). It is also provides a convenient means of visualization of network features on the input by generating a class activation map (CAM) [12]. A CAM is a heatmap over the pixel space of an input image which indicates regions that most strongly influence the class prediction. CAMs may also be used as a form of weakly-supervised semantic-segmentation; that is, the CAM can be thought of as a pixel-level class membership likelihood function that may be trained to identify regions of interest through using only whole-image annotated images. A CAM overlaid on the input sample is shown in four examples in Fig. 2, where a rough measure of confidence in the prediction as a MLO or not-MLO is shown above each sample. The right two samples contain MLO, where the colored portions clearly illustrate where a MLO is present. The two left samples do not contain a MLO; the colored portions do not necessarily reflect a particular object, only showing areas most strongly indicating not-MLO.

TABLE I
CNN CONFIGURATIONS WITH VARYING LAYER DEPTH

Ident.	Layer Desc.
L1	c16-gap-fc2
L2	c16-c16-gap-fc2
L3	c16-c16-mp-c32-gap-fc2
L4	c16-c16-mp-c32-c32-gap-fc2
L5	c16-c16-mp-c32-c32-mp-c64-gap-fc2
L6	c16-c16-mp-c32-c32-mp-c64-c64-gap-fc2
L7	c16-c16-mp-c32-c32-mp-c64-c64-c128-gap-fc2
L8	c16-c16-mp-c32-c32-mp-c64-c64-c128-c128-gap-fc2
L9	c16-c16-mp-c32-c32-mp-c64-c64-c128-c128-c256-gap-fc2

These networks were implemented with the Keras [13] software framework. Training was done using backpropagation to adjust the weights, specifically mini-batch (32 examples) stochastic gradient descent (SGD) with the Adam [14] learning

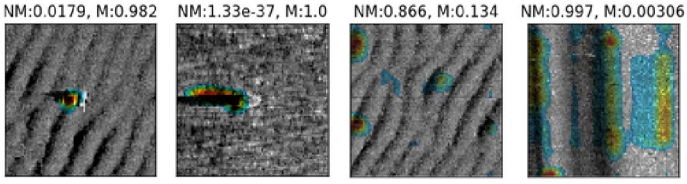

Fig. 2. Colored CAM overlay (best viewed in color) with correct *MLO* and *Not-MLO* classifications.

rate optimizer.

A concern among any machine learning approach is overfitting, and we can verify this does not happen by observing the accuracy and loss function values on both the training and validation datasets, as training progresses. The validation dataset is a disjoint selection of examples from the training dataset, and is used to evaluate the model on examples never used in SGD backpropagation. Figure 3 shows these trends (the validation set is labeled "test"). If the validation loss (and especially accuracy) diverged greatly from the training loss, it would indicate the network memorized only specific examples used for training, rather than "learning" features that generalize well, allowing the model to correctly classify unseen data. In this case, there is only slight divergence in loss, and both training and generalization error converge within the 50 epochs (observations of the entire training set) considered.

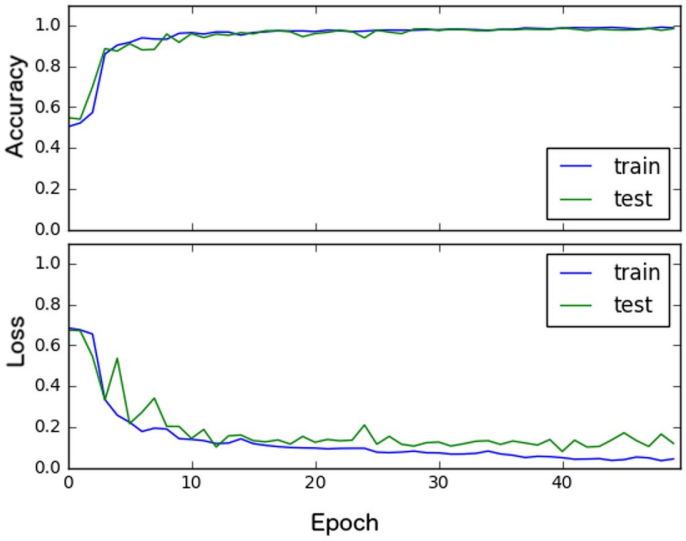

Fig. 3. Accuracy and loss function values over the course of training for the L5 model configuration.

III. Results

A. Accuracy vs. Model Size

We show a summary of results in Table II comparing the different layer configurations. Efficacy for each configuration is indicated by the accuracy and F1-score (harmonic mean of precision and recall) columns. The rates of false positives and false negatives are similar, as observed by inspection of the confusion matrices. The L7 configuration indicated the highest accuracy and F1 scores of 0.989, but may be a noise-induced peak and not indicative of a generally superior model.

The cost of increasing the accuracy with additional layers is shown in the columns for number of parameters (e.g., network weights), number of calculations (approximate multiplications and additions per input patch), and throughput of input patches per second. Parameter count is a representation of how much memory is required to hold the network definition, and calculation count is a representation of the numerical operations that must be performed using that memory. A system designer may wish to consider these characteristics when choosing or designing a computer architecture to match this particular domain (sonar imagery analysis). In comparison, the VGG network [10] used as a benchmark for many-class visual identification contains 138 million parameters, so these models require far fewer resources.

As the network depth increases (along with parameter count), increased efficacy is generally observed. However, there is a point of diminishing return around the L5 configuration—a 5x increase in parameter count (to L7) results in a 0.7% increase in accuracy. The "best" configuration would be chosen by taking into account the particular platform, whether an autonomous vehicle constrained by computing resources or a ship-based system with fewer constraints.

The throughput was measured by timing the inference of the training set on a laptop-class CPU processor (Intel® i7-4770HQ), and dividing the training set size by this time. All configurations ran at greater than 100 inputs/second, which is enough for real-time operation in many embedded systems. Throughput is highly dependent on the hardware used, e.g., a simpler embedded CPU processor would have much lower throughput, while a general-purpose graphics processing unit (GPGPU) would have far greater throughput. The size of input image size will greatly affect throughput, and thus these results are only valid for patches of 80 x 80 pixels.

TABLE II
DNN Model Configuration Comparison

Ident.	No. Param.	No. Calc.	Throughput	Acc.	F1
L1	194	2e6	3094	0.535	0.672
L2	2,514	33e6	646	0.932	0.929
L3	7,186	48e6	433	0.961	0.960
L4	16,434	79e6	321	0.972	0.971
L5	34,994	94e6	287	0.982	0.982
L6	71,922	125e6	250	0.986	0.986
L7	145,906	186e6	212	0.989	0.989
L8	293,490	309e6	156	0.985	0.985
L9	588,914	555e6	111	0.988	0.988

B. Sensitivity Tradeoff

We can glean more insight into our model's efficacy by evaluating how different mine-detection threshold levels affect probability of detection and probability of false alarm. When the softmax layer output for class *mine* is ≥ 0.5, by default

this is interpreted as a positive *mine*. However, this threshold can be varied to yield different levels of probability of detection. The resulting plots of false positive rates (FPR) and true positive rates (TPR) form a receiver operating characteristic (ROC) curve, which is shown for selected configurations in Figure 4. The ROC comparison is critical, as it illustrates the tradeoff between false positives and true positives, and provides a "knob" of control to tune based on system needs and requirements. For instance, higher sensitivity to mines may be achieved at the expense of increased false positives. If the system demands the lowest FPR, then L9 is optimal among the studied configurations, and Pareto-dominates the others. However, among FPRs of 2% or greater, there is little difference between L6-L9. This result seems to indicate that the networks were not overfit to the training data, and instead were able to learn features that generalized to the test set.

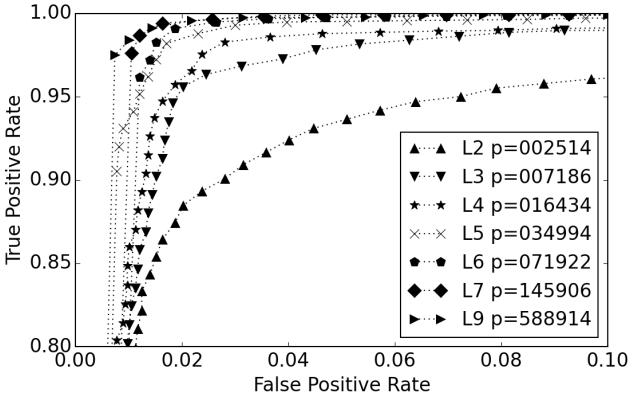

Fig. 4. ROC comparison of models of varying layers and parameter size.

C. Training Dataset Size

One of the criticisms of certain machine learning methods, including DNNs, is their reliance on large labeled datasets for training. We examined the sensitivity of our method to this dataset size by training the L5 configuration using various amounts of training data, ranging from 250 to 4000 patches, split evenly into *mine* and *not mine* categories. The resulting ROC curve is shown in Figure 5, which indicates a strong improvement in efficacy as training data size increases, especially at low FPRs. Even at a high FPR or 10%, using the smallest training dataset (250 patches) yielded a TPR substantially below the others (95% vs. 98% or more). This result indicates that additional *mine* examples are beneficial to improve TPR, and not only in reducing FPR through additional background (*not mine*)exposure.

D. Comparison with Support Vector Machine

We performed a comparison with another machine learning method, the support vector machine (SVM), as implemented by the function *SVC* in scikit-learn [15]. The training size was 2000 examples of an even split between mine and not-mine, where the pixel intensities are used as the feature vector (same as with the DNN model) . The SVM used default parameters. The L5 DNN with the same training size of 2000 examples was chosen as the point of comparison. The probability of false alarm (FPR) was slightly better than L5, however, the miss rate (FNR) was much worse.

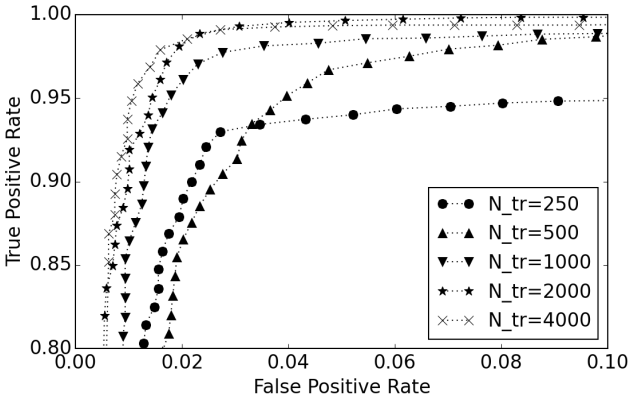

Fig. 5. ROC of the L5 model trained with different sizes of the dataset.

TABLE III
EFFICACY OF L5 CONF. AND SVM

Model Type	Acc.	F1	FPR	FNR
DNN L5	0.98	0.98	0.015	0.021
SVM	0.78	0.75	0.006	0.468

E. Failed Classifications

We investigate improper classifications to get better insight as to why those failures occurred. Figure 6 is a sample of false positive detections from the L6 configuration, the total of which was 50 out of 2569 negative (not-mine) examples. The colored CAM overlays show what part of the image contributed most to the classification (mine). Qualitative inspection suggests these examples do seem "mine-like" in many ways. Possibly contributing to the misclassification is that some of the objects are near the edge of the image patch, and thus may be missing "context" of features around the MLO. Another possibility is that the original labels for the patches are incorrect, caused by user error in the manual-labeling process.

False negatives examples are shown in Figure 7, which totaled 23 out of 2569 positive (mine) examples. These may be more of a concern to an operation, as they constitute a missed detection of a real mine. However, we see that the DNN did identify features that constitute a MLO, as shown by the CAM overlay, even if the overall classification of the image was *not-MLO*. In these cases, the "evidence" of not-MLO was weighted higher than the evidence for MLO. This fact is encouraging in that the core function of the algorithm is seemingly sensitive to these MLO features. Additional methods could potentially be incorporated to reduce the false negative rate.

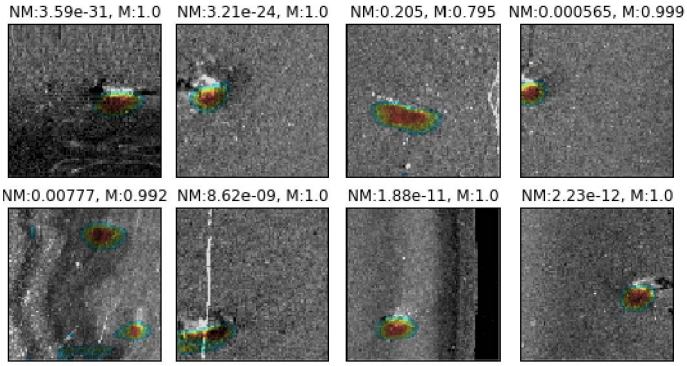

Fig. 6. False positive detections with CAM overlay for *mine* (best viewed in color).

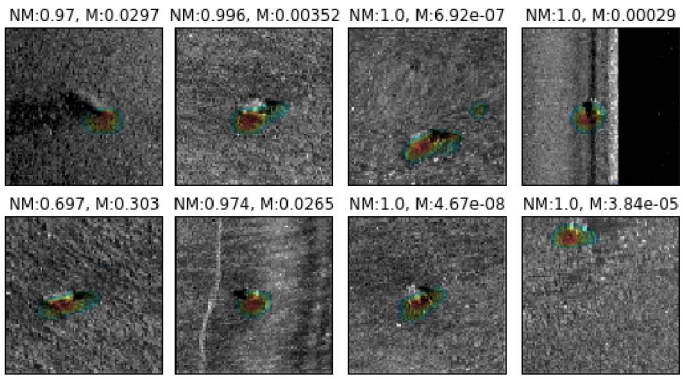

Fig. 7. False negatives (classified not-mine) with CAM overlay for *mine* (best viewed in color).

IV. Conclusion

We present a study showing how convolutional neural networks may be used for MLO detection in side-scan sonar imagery. Many other techniques exist, but our pursuit of CNNs is driven by their demonstrated superiority in related computer vision applications. The design space of CNNs is immensely large, and we analyzed one axis of configuration—parameter count adjusted with convolution layer depth. We presented results showing the computation requirements and efficacy tradeoffs for these model configurations.

Models attain 98% accuracy and run with a throughput of 100-300 images per second on a laptop-class CPU. This throughput is high enough for real-time operation on many embedded systems-class processors, which is important for bringing this capability to autonomous underwater vehicles. The largest models show only incremental improvement, at best, in efficacy, at the cost of increased computation requirements. We conclude for the task of MLO, a DNN-style machine learning algorithm is feasible for achieving high efficacy with on-board processing capabilities.

More broadly, the methods shown in this paper can also be adapted to look for other targets of interest, not only MLOs, but possibly sunken aircraft, shipwrecks, or to catalog natural seafloor features and sealife. For any of these applications, a well-crafted dataset is imperative to achieve quality results. DNNs and other data-intensive methods will benefit from incorporating voluminous data–a fact with advantages and disadvantages. New data can be incorporated easily by retraining the model without inventing and testing new algorithms or hand selected features. On the other hand, there is the requirement for large training sets, which may be difficult to obtain or generate. The systems engineer will need to weight these requirements before deciding if a DNN or similar method is most suitable to a particular application.

References

[1] S. Reed, Y. Petillot, and J. Bell, "An automatic approach to the detection and extraction of mine features in sidescan sonar," *IEEE Journal of Oceanic Engineering*, vol. 28, no. 1, pp. 90–105, 2003.

[2] A. Sinai, A. Amar, and G. Gilboa, "Mine-like objects detection in side-scan sonar images using a shadows-highlights geometrical features space," in *OCEANS 2016 MTS/IEEE Monterey*, Sept 2016, pp. 1–6.

[3] C. Barngrover, R. Kastner, and S. Belongie, "Semisynthetic versus real-world sonar training data for the classification of mine-like objects," *IEEE Journal of Oceanic Engineering*, vol. 40, no. 1, pp. 48–56, Jan 2015.

[4] P. Zhang, L. Wang, and S. Han, "Research on segmentation of sonar image based on features learning," in *2016 IEEE International Conference on Mechatronics and Automation*, Aug 2016, pp. 1897–1901.

[5] C. Barngrover, A. Althoff, P. DeGuzman, and R. Kastner, "A brain-computer interface (bci) for the detection of mine-like objects in sidescan sonar imagery," *IEEE Journal of Oceanic Engineering*, vol. 41, no. 1, pp. 123–138, Jan 2016.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

[7] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, June 2005, pp. 886–893 vol. 1.

[8] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346 – 359, 2008, similarity Matching in Computer Vision and Multimedia. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314207001555

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[11] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2013. [Online]. Available: http://arxiv.org/abs/1312.4400

[12] B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," *CoRR*, vol. abs/1512.04150, 2015. [Online]. Available: http://arxiv.org/abs/1512.04150

[13] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.