# Drawing: A New Way To Search
## (Computer Vision)

Nguyet Minh Phu
*Department of Computer Science*
*Stanford University*
*minhphu@stanford.edu*

Connie Xiao
*Department of Computer Science*
*Stanford University*
*coxiao@stanford.edu*

Jervis Muindi
*Department of Computer Science*
*Stanford University*
*jmuindi@stanford.edu*

*Abstract*—**Computers having the ability to understand our quick line drawings will allow for broader forms of expression and communication. In our project, we aim to use machine learning techniques to efficiently recognize labels of hand-drawn images. After implementing and analyzing classical and deep learning models, we found that a simplified Convolutional Neural Network was best for this task.**

## 1. Introduction

### 1.1. Motivation

Using words can be limited when communicating across cultures and literacy levels. Drawing images is a shared communication method that can bridge those divides. If successful, this model can be applied for a variety of interesting tasks, including a new search interface where someone can draw what they need and search for it, or an app where a language learner can draw an image and get the translation immediately. These applications require computers to understand our quick line drawings or doodles.

### 1.2. Goal

Our goal is to develop an efficient system to recognize labels of hand-drawn images from Google's QuickDraw dataset. The input to our algorithm is an image. We use Logistic Regression, Support Vector Machines (SVMs), Convolutional Neural Networks (CNNs), and Transfer Learning to output a predicted class.

## 2. Related Work

### 2.1. Image Recognition

Deep learning has proven to be very successful in general image classification. Past ImageNet [1] competition winners have used deep learning techniques, CNNs in particular, to achieve high accuracies ever since 2012 [11, 16]. Our interest in CNNs is motivated by the favorable outcomes as demonstrated in the area of computer vision.

**2.1.1. Transfer Learning.** The idea behind Transfer Learning is that we can apply knowledge from a generalized area to a novel task [15]. There are two broad strategies [6]. The first is to use the pre-trained model as a fixed feature extractor. New candidate inputs would be run through the pre-trained model, but stopped at the penultimate fully-connected layer to get a feature vector. The other approach is fine-tuning which uses back-propagation to update the weights of the pre-trained model. When layers are fine-tuned to the new tasks, the model generally does well on that new task [10].

While pre-trained ImageNet models have been widely used in Transfer Learning for other natural image classification tasks, they have not commonly been used for hand-drawn images. However, researchers, Lagunas and Garces, have successfully used Transfer Learning with VGG [17] for artistic illustration [12]. Artistic illustrations could be considered as high-quality doodles. We wanted to explore if Transfer Learning would also be good for quickly-drawn doodles of varying qualities.

### 2.2. Our Contributions

We are working on a relatively new dataset (released under two years ago) with a focus on efficiency. Our project could be seen as within the domain of image recognition. However, doodles only consist of lines and have two colors, black and white, which could render complex features learned by image recognition models useless. We want to focus on not just accuracy but also efficiency. Efficiency (i.e model size, training time) is critical to allow deployment of the system in real-life applications, yet it has not received sufficient attention in research.

## 3. Dataset

Google's QuickDraw [3] is the worlds largest doodling dataset, consisting of 50 million hand-drawn images across 345 categories. This dataset is highly applicable to our goal because doodles are what we will draw when we need to communicate something quickly via image. Some examples are given in figure 1.

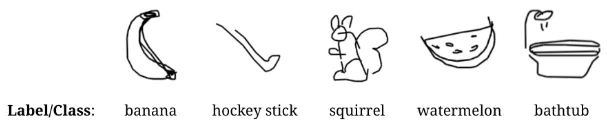

Figure 1. Examples of the data

We used the bitmap version of the data. Each drawing consists of 28 by 28 raw pixel inputs with values from 0 to 255. We took advantage of the fact that each image has only two colors, black and white, to binarize the pixels for a more compact representation.

To make training more tractable on modest computing resources, we elected to work with a subset of the data. The classes selected were chosen randomly from the overall pool of classes and were fixed throughout our experimentation. The number of examples per class is 20,000. We picked 3, 10, and 50 classes to train. For each of the classes, we split our data into training, validation, and test sets with the ratio of 80:10:10.

## 4. Methods

Broadly speaking we looked at two classes of algorithms for our task: traditional machine learning approaches and deep learning techniques. The link to our Github with our code is here: `https://github.com/jervisfm/cs229-project`.

### 4.1. Classical Machine Learning

**4.1.1. Logistic Regression.** For our baseline, we used Logistic Regression, a simple and fast model to train.

The log likelihood for our logistic model [13] to be optimized is given by:

$$l(\theta) = \sum_{i=1}^{m} y^{(i)} log\, h_\theta(x^{(i)}) + (1 - y^{(i)}) log\,(1 - h_\theta(x^{(i)}))$$

where

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

**4.1.2. Support Vector Machine.** Support Vector Machines are optimal margin classifiers and the optimization objective for these models [14] is given below:

$$min_{w,b}\frac{1}{2}\,\|w\|\; s.t\; y^{(i)}(w^T x^{(i)} + b) >= 1, i = 1, ..., m$$

We explored using Support Vector Machines with various kernels to find empirically the kernel most suited for the task of doodle classification. The types of kernels we experimented with are Linear, RBF (Radial Basis Function), Polynomial, and Sigmoid.

### 4.2. Deep Learning

**4.2.1. Convolutional Neural Network (CNN).** We started out with a CNN, a natural candidate for image recognition given the convolutional layer's ability to capture spatial dependency. A key insight is that since a doodle is a simple image, some components of the CNN may be unnecessary. By identifying and removing these layers, we developed a compact model that is both fast to train and still accurate.

The CNN architecture is given in figure 2. We used a simple architecture because we wanted to find a time efficient model that can perform well. Much research has focused on developing more complex models (as mentioned in section 2.1). However, these models are also more expensive to train and do inference on. After implementing the CNN, we simplified it by progressively removing layers and dense units to analyze the impact on accuracy and runtime (figure 3).

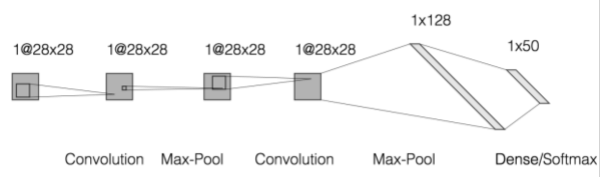

Figure 2. CNN architecture

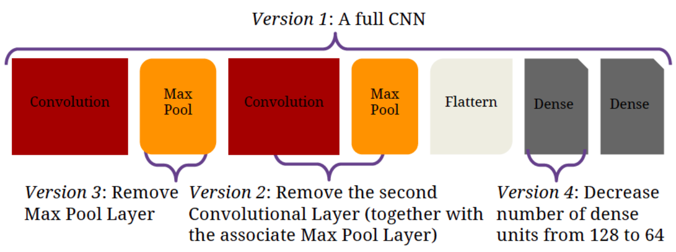
*Version 1*: A full CNN
Convolution
Max Pool
Convolution
Max Pool
Flattern
Dense
Dense
*Version 3*: Remove Max Pool Layer
*Version 2*: Remove the second Convolutional Layer (together with the associate Max Pool Layer)
*Version 4*: Decrease number of dense units from 128 to 64

Figure 3. CNN Versions: a sketch of how we progressively simplified our CNN for doodle classification

**4.2.2. Transfer Learning.** Even with a simple CNN, we noticed that training a deep-learning model from the ground up can be time and resource intensive. Since a doodle is also an image, we explored if it is feasible to transfer knowledge from winning ImageNet architectures to our specific problem of doodle classification via Transfer Learning. We used four different baseline models, namely Inception V3, VGG, MobileNet and ResNet50 from the ImageNet competition, and extended them for doodle classification (figure 4). More specifically, we added a global spatial average pooling layer after the original architecture, followed by a dense layer of 128 units with ReLu activation, and finally a softmax layer for classification. We used stochastic gradient descent with an Adam optimizer to fine-tune the added layers. Due to constraints on computational resources, we optionally fine-tuned the top two-layers of the original network.

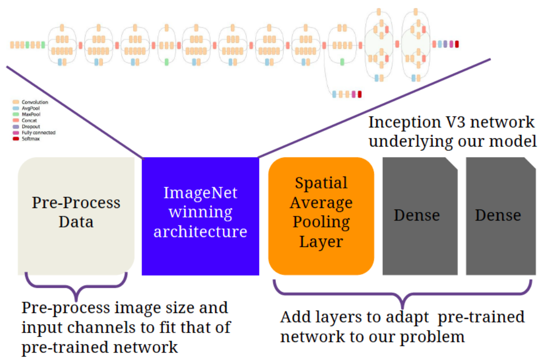
Inception V3 network
underlying our model
Pre-Process Data
ImageNet winning architecture
Spatial Average Pooling Layer
Dense
Dense
Pre-process image size and input channels to fit that of pre-trained network
Add layers to adapt pre-trained network to our problem

Figure 4. A sketch of how we carried out Transfer Learning

## 5. Results and Discussion

### 5.1. Evaluation Metrics

**5.1.1. Accuracy.** Accuracy is the percentage of correctly classified doodles over the total number of doodles. Our reported accuracy is on the validation set.

**5.1.2. Efficiency.** We measured training time in seconds. For Logistic Regression and SVMs, we ran these models locally. Due to our local machines' computational power constraints, we ran the CNNs and transfer learning models on Google Cloud on a virtual machine with 320GB of local disk, 12 Cores of CPU, 64GB of RAM, and an NVIDIA P100 GPU with 16GB of memory. We used Google Cloud Deep Learning machine image [7].

### 5.2. Logistic Regression

Logistic Regression was implemented using Python's SciKit Learn framework [4]. The solver used was `lbfgs` and the `multi_class` setting was multinomial to classify our many classes. To keep training time reasonable on the larger dataset, the maximum number of iterations was set to 100. The results for Logistic Regression are reported in table 1.

| | **Accuracy (%)** | | | **Training Time[1](s)** | | |
|---|---|---|---|---|---|---|
| **Classes** | 3 | 10 | 50 | 3 | 10 | 50 |
| *Baseline* | 79.51 | 64.64 | 43.89 | 25 | 122 | 1089 |

[1] Ran with 100 iterations.

TABLE 1: Logistic Regression Results

The accuracy scores reported in the confusion matrix are normalized with 1 representing 100% accuracy. The best performance for Logistic Regression was with 3 classes where we achieved 79% accuracy and it took less than a minute to train. On the other hand, with a larger dataset of 50 classes, training time increased 40 times to half an hour and the overall accuracy fell to 43%.

Putting these numbers in context, for the 50-class dataset, a classifier that randomly guesses the class would have expected accuracy of 2%. We see that the Logistic Regression classifier did relatively well.

That said, there was still room for improvement and we performed error analysis next to understand the types of errors that the classifier was making.

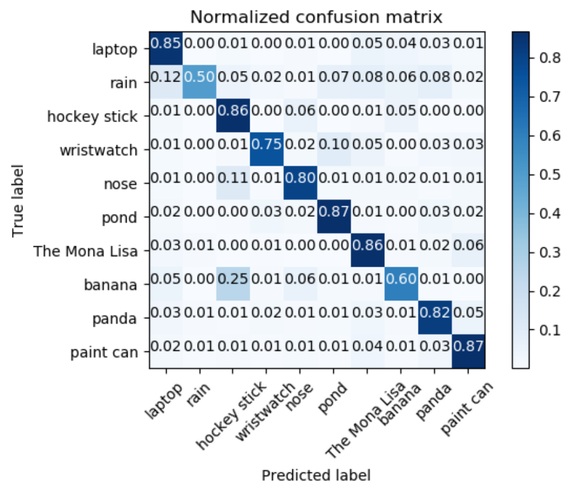

Figure 5. Linear Regression's Confusion Matrix

Looking at the confusion matrix, we can see that Logistic Regression performs relatively well. The diagonal of the confusion matrix carries the most weight, indicating it often makes the correct prediction. We notice that in the wrongly classified regions, the true label *banana* is highly misclassified with *hockey stick*. This is expected as a hand-drawn banana is very similar to a hand-drawn hockey stick as seen in figures 6 and 7. Thus, this experiment suggests that in order to predict hand-drawn doodles, contrary to our initial belief, we may need a more sophisticated model instead of a simpler model because the quality of the drawing may not be very good.

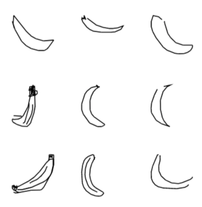

Figure 6. Banana Doodles

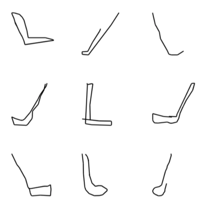

Figure 7. Hockey Stick Doodles

### 5.3. SVM

We implemented SVMs with four different kernels: Linear, RBF, Polynomial, and Sigmoid. The results are reported in table 2

For training, we used non-binarized data and normalized the inputs to the range $[-1, 1]$. We trained for 500 iterations. The initial parameters that we used are:

- Polynomial: degree of 5, coefficient of 1
- RBF: gamma of 1
- Sigmoid: coefficient of 1

| | **Accuracy (%)** | | **Training Time[1](s)** | |
|---|---|---|---|---|
| **Classes** | 3 | 10 | 3 | 10 |
| *Linear Kernel* | 40.8 | 22.22 | 204 | 1831 |
| *RBF Kernel* | 62.68 | 61.01 | 317 | 2842 |
| *Polynomial Kernel* | 51.7 | 50.89 | 448 | 6673 |
| *Sigmoid Kernel* | 33.08 | 11.72 | 478 | 6971 |

[1] Ran with 500 iterations.

TABLE 2: SVM Results

To our surprise, the SVMs performed worse than Linear Regression overall. However, this could be due to the fact that we have not done extensive hyperparameter tuning for SVMs.

Among our different choices of kernels, the RBF kernel performed the best, followed by the Polynomial kernel with degree 5, then the Linear kernel. The Sigmoid kernel performed the worst with an accuracy equitable to assigning a category at random.

This result is consistent with what we expected. The accuracy corresponds to the complexity of the feature space with RBF corresponding to an infinite feature space and Polynomial and Linear having fewer features. Although the Sigmoid corresponds to a higher dimensional feature space, its corresponding kernel matrix is not guaranteed to be positive semi-definite. Therefore, if the chosen parameters are not well-tuned, the algorithm can perform worse than random [5].

## 5.4. Convolutional Neural Network

We tested four different variations of a basic CNN using binarized and non-binarized data. A base V1 model included two convolutions and two max-pool layers from which we then iteratively simplified as shown in figure 3. The performance of these models is given in the tables below:

| | **Accuracy (%)** | | | **Training Time[1](s)** | | |
|---|---|---|---|---|---|---|
| **Classes** | 3 | 10 | 50 | 3 | 10 | 50 |
| *V1:* | 91.43 | 89.34 | 81.79 | 225 | 619 | 2808 |
| *V2:* | 91.33 | 89.22 | 81.83 | 230 | 622 | 2808 |

[1] Ran with 100 iterations.

TABLE 3: CNN Results with non-binarized input features

| | **Accuracy (%)** | | | **Training Time[1](s)** | | |
|---|---|---|---|---|---|---|
| **Classes** | 3 | 10 | 50 | 3 | 10 | 50 |
| *V1:* | 89.02 | 87.22 | 77.46 | 220 | 590 | 2685 |
| *V2:* | 89.12 | 86.67 | 77.38 | 215 | 585 | 2715 |
| *V3:* | 88.2 | 86.6 | 70.3 | 186 | 560 | 2455 |
| *V4:* | 89.1 | 85.6 | 70.3 | 239 | 628 | 3043 |

[1] Ran with 100 iterations.

TABLE 4: CNN Results with binarized input features

Because the binarized input contains less information than the non-binarized input, we saw an analogous drop in accuracy and training time across classes.

We find that the best performing CNN on the larger classification task of 50 classes was V2 which attained a validation set accuracy of 81.83% with the confusion matrix below.

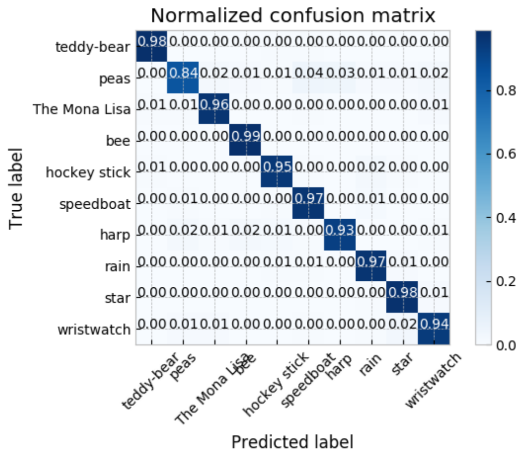

Figure 8. Confusion matrix for V2

Below is a plot of loss and accuracy over time for this model during training:

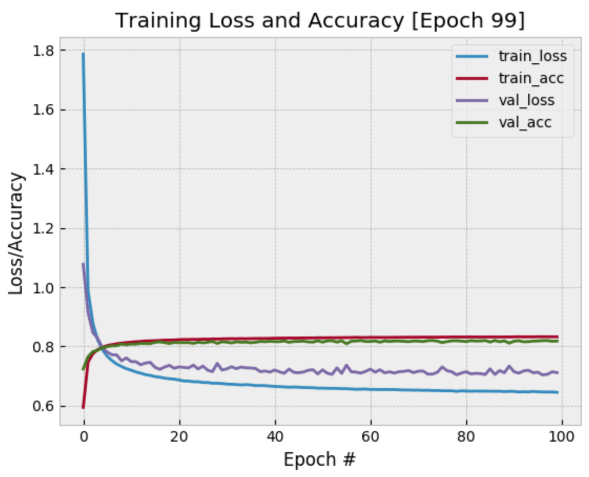

Figure 9. CNN Training plot

There is a close correlation between the training accuracy and validation accuracy. Testing this V2 model on the test set, we obtained a final accuracy score of 81.87%. This shows that our trained model is able to generalize well to unseen data.

In general, simple CNNs performed well for our task. With fewer classes the accuracy is roughly the same when taking away layers, but the training time decreases. With more classes, accuracy decreases with fewer layers as expected, but the lowest accuracy is still significantly greater than the accuracy found using Logistic Regression.

## 5.5. Transfer Learning

We ran Transfer Learning with weights pre-trained on the ImageNet dataset with four different architectures:

- VGG: a model proposed in the ImageNet 2013 challenge which was widely used due to its simple architecture consisting of only repeated units of convolutional layers followed by max-pooling [17]
- Inception V3: also known as GoogLeNet which uses skip connections to form "Inception Modules" that are repeated throughout the network [18]
- ResNet50: the winner of the 2015 ImageNet which uses a very deep architecture with "Residual Blocks" [8]
- MobileNet: a streamlined model optimized for mobile and embedded applications [9]

Due to the large size and number of parameters of these models, we had to train them on the cloud using GPUs. The GPU used was an NVIDIA Tesla P100 with 16GB of memory. Even then we ran into challenges with our GPU running out of memory especially on VGG. To address these, we elected to only tune our last custom layers and also reduce the batch size by an order of magnitude.

The results of Transfer Learning experiments are discussed below:

| **Accuracy (%)** | **ImageNet**[1] | **QuickDraw**[2] |
|---|---|---|
| *Inception v3* | 93.7 | 45.72[3] |
| *MobileNet* | 89.5 | 75.4 |
| *VGG* | 90.1 | 95.58 |
| *ResNet50* | 92.1 | 62.72 |

[1] ImageNet's accuracy refers to the top-5 accuracy on the ImageNet validation dataset [2].
[2] QuickDraw's accuracy refers to the accuracy on 3 classes on our validation set.
[3] Accuracy is calculated after fine-tuning the top-most 2 Inception layers.

TABLE 5: Transfer Learning Accuracy Results

There is not a strong correlation between the performance of the base model on ImageNet with that of the QuickDraw dataset. In fact, the best performing model on ImageNet, Inception V3, performed poorly on the QuickDraw dataset, obtaining only 45.72% accuracy on 3 classes. We noticed, however, that the best performing model on the QuickDraw dataset, VGG, is also the one with the simplest architecture. This suggests that more complex architectures such as the "Inception Module" used in Inception V3 or the "Residual Block" used in ResNet50 may not be beneficial for the problem of doodle classification. Since doodles are simple drawings, only using the classic convolutional and max-pool layers may be the best. This also corroborates our earlier finding with CNN, where our simple CNN built from the ground up did well on the QuickDraw dataset.

| | **No. Parameters** | **Training Time/Iters** |
|---|---|---|
| *Inception v3* | 23,851,784 | 540[1] |
| *MobileNet* | 4,253,864 | 103 |
| *ResNet50* | 25,636,712 | 152 |
| *VGG* | 138,357,544 | 207 |

[1] Training time includes fine-tuning of the top-most 2 Inception layers

TABLE 6: Transfer Learning Training Time Results

In terms of training time, MobileNet performed the best. This is expected since the model is optimized for efficiency and has the smallest number of parameters. Overall, the trend in training time follows the number of parameters in the base model, which is expected. The exception of Inception v3, whose training time is the largest despite it having the second largest number of parameters of all the four models. This was due to us additionally fine-tuning the parameters of the two top-most layers of Inception V3. This was because Transfer Learning with Inception V3 was performing poorly in terms of accuracy, and we wanted to see if further tuning hyper-parameters would help. Overall, we find that optimizing the model by reducing the number of parameters will help with reducing training time, which further supports our initial push for simplifying the models to achieve higher efficiency.

## 6. Conclusion

Our project aimed to recognize the meaning of doodles, a critical first task in order to build any system that uses hand-drawn images for communication. We focused on doodle recognition with an emphasis on efficiency in conjunction with accuracy. After implementing Logistic Regression, SVMs, CNNs, and Transfer Learning and analyzing our results, we found that a simplified CNN was best for the task, balancing both accuracy and training time. We also found that for simpler images such as doodles, using classic architectures such as a combination of convolutional and max-pool layers can outperform complex architectures.

For future work, we would further develop our most promising approach by performing more extensive experiments to determine the effect of each layer in the CNN. We would also explore using Transfer Learning as a fixed feature extractor for Logistic Regression, our fastest model. Given more time, we would also love to explore working on efficiency in conjunction with smaller datasets.

## 7. Contributions

Each team member contributed equally to this project.

## References

[1] Imagenet. http://www.image-net.org/. Accessed: 2018-12-10.

[2] Models for image classification with weights trained on imagenet. https://keras.io/applications/. Accessed: 2018-12-10.

[3] The quick, draw! dataset. https://github.com/googlecreativelab/quickdraw-dataset. Accessed: 2018-12-10.

[4] scikit-learn: Machine learning in python. https://scikit-learn.org/stable/. Accessed: 2018-12-10.

[5] R. Amami, D. B. Ayed, and N. Ellouze. Practical selection of svm supervised parameters with different feature representations for vowel recognition. *arXiv preprint arXiv:1507.06020*, 2015.

[6] CS231n. Convolutional neural networks for visual recognition: Transfer learning. 2018.

[7] Google. Cloud deep learning vm image. 2018. https://cloud.google.com/deep-learning-vm/.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] 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. *arXiv preprint arXiv:1704.04861*, 2017.

[10] S. Kornblith, J. Shlens, and Q. V. Le. Do better imagenet models transfer better? *CoRR*, abs/1805.08974, 2018.

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

[12] M. Lagunas and E. Garces. Transfer learning for illustration classification. *arXiv preprint arXiv:1806.02682*, 2018.

[13] A. Ng. Supervised learning lecture notes. 2018. http://cs229.stanford.edu/notes/cs229-notes1.pdf.

[14] A. Ng. Support vector machines. 2018. http://cs229.stanford.edu/notes/cs229-notes3.pdf.

[15] S. J. Pan, Q. Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

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

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.