Understanding the Amazon Rainforest from Space using Neural Networks*

Christian Koguchi ckoguchi@eng.ucsd.edu
Naveen Ketagoda nketagod@eng.ucsd.edu
Niral Pathak nlpathak@eng.ucsd.edu
Samuel Sunarjo ssunarjo@eng.ucsd.edu

***Abstract*— We apply modern machine learning techniques for multi-label classification of satellite imagery. Using custom convolutional neural networks and popular architectures with transfer learning, we participate in a Kaggle competition that aims to fight deforestation.**

I. INTRODUCTION

Deforestation in the Amazon Rainforest accounts for reduced biodiversity, habitat loss, climate change, and other devastating ecological and environmental effects. Understanding and tracking human deforestation and ecological changes over time can better help environmentalist and government efforts in responding to both human and natural forest loss. Using modern computer vision techniques, we attempt to aid in this effort by using satellite imagery data of the Amazon Rainforest to track the expansion of human deforestation efforts. The ability to classify satellite data will allow us to better understand where, how, and why deforestation happens all over the world - and ultimately how to respond [1].

We utilize several convolutional neural networks (CNNs) whose inputs are images of the rainforest and whose outputs are predicted labels of the condition of the rainforest. Class labels can be categorized into 3 groups: atmospheric conditions, common land cover, and rare land cover. Weather conditions such as overcast can complicate accurate labeling of the data, which must be taken into careful consideration. Our experimentation included the following methods:

- ***Custom CNN Architectures***: We create our own custom neural network architectures using modern techniques in machine learning and computer vision.
- ***Transfer Learning using Various Successful CNN Architectures***: We wish to utilize pretrained features from the successful VGG16 and Xception architectures and fine-tune the weights through backpropagation to classify the satellite images in our dataset.

We implemented these in the Python programming language using the high-level neural networks API Keras with TensorFlow backend [2], [3].

II. RELATED WORKS

Convolutional Neural Networks work well for image classification tasks partly due to their rich hierarchical representation of features derived from an input image. Prior to CNNs however, there have been several similar approaches for extracting features for computer vision and image processing tasks.

Finite-Impulse Response (FIR) filter design laid the foundations for low-level feature extraction for things such as edges, lines, and colors of an image. Laplacian, Prewitt, and Gabor filters are examples of primitive feature extractors that can detect lines and edges in various orientations [4]. These filters were applied using convolution and were some of the first methods in computer vision and image processing for developing "hand-crafted" features for image classification. It is pleasing that the first-layer filters learned through backpropagation of a convolutional neural network are typically variants of Laplacian and Gabor filters.

Wavelet representations were an early development in creating a hierarchical representation of features [5]. A wavelet representation can be obtained by recursively applying an orthogonal "filter bank" followed by a decimation step over various stages creating a multi-scale hierarchical representation of the image. This is similar to CNNs in that the various output channels of a convolutional layer can be interpreted as a filter bank followed by a downsampling or "pooling" stage. However, these methods used hand-crafted filters, while the philosophy of deep learning attempts to learn the best representation (or kernels) through backpropagation.

An early convolutional network called the Neocognitron was designed for optical character recognition of handwritten digits from 0 to 9 in the 1980's by Fukushima et. al [6]. They alternated between convolution and average-pooling without backpropagation. LeCun et. al. created LeNet-5 for reading hand-written digits using the hyperbolic tangent function after convolutions trained using backpropagation on the famous MNIST dataset [7]. In 2012, AlexNet won the ImageNet challenge using a deeper CNN using ReLU and introduced the concepts of using specific GPU architectures, normalization, and overlapping pooling [8]. Since this landslide victory of the ImageNet challenge, many variants of CNNs have appeared such as ZFNet [9], VGG[10], Inception/GoogLeNet[11], ResNet[12], DenseNet[13], and Xception[14].

III. DATA

We obtain the dataset from Kaggle's "Planet: Understanding the Amazon from Space" [1].

- `train.csv` - a list of training file names and their labels; the labels are space-delimited

*This work was supported by ECE 228 - ML for Physical Applications

- `sample_submission.csv` - standard format of submission according to Kaggle, which contains all the file names in the test set
- `[train/test]-tif-v2.tar.7z` - tif files for the training/test set (updated: May 5th, 2017)
- `[train/test]-jpg[-additional].tar.7z` - jpg files for the training/test set (updated: May 5th, 2017)

According to Kaggle, the data is provided in a "chips" image format. Planet, the company sponsoring the competition, used 4-band satellites in sun-synchronous orbit (SSO) and International Space Station (ISS) orbit. The chips use the "GeoTiff" format characterized by four bands of data: red, green, blue, and near infrared, each of which is in 16-bit digital number format. Kaggle has stripped "geotiff information regarding the chip footprint and ground control points" [1]. The data, collected between January 1, 2016 and February 1, 2017, centers around the Amazon basin which includes Brazil, Peru, Uruguay, Colombia, Venezuela, Guyana, Bolivia, and Ecuador (Figure 1).

Fig. 1. Map of Amazon River Basin

Kaggle further provides the JPEG format of the data with only three channels (rgb). The training dataset consisted of 40479 labeled files, and the test dataset consisted of 61191 files. The images are 256 x 256 x 3 pixels. There are 17 unique labels provided in the training labels CSV:

- *four weather labels*: clear, partly cloudy, haze, cloudy
- *six land labels*: primary, agriculture, water, cultivation, habitation, road
- *seven rarer labels*: slash burn, conventional mine, bare ground, artisinal mine, blooming, selective logging, blow down

This allows us to frame our task as a multi-class classification problem. We provide example training data with respective labels in Figure 2.

Fig. 2. Examples of Labeled Data

A. *Pre-Processing*

We opted to use the JPEG formatted data. We randomly shuffle data and use a 80/20 training and validation split. We resize from $256x256x3$ to $96x96x3$ for faster training. We do note that this down-sampling may result in a loss of information. We normalize the image pixel values from 0-255 to 0-1 then perform mean subtraction over the entire training set. We record this mean and apply it to the validation and testing set. We further perform the following functions randomly on the training images:

- rotate the images from 0°-90°
- flipping horizontally
- flipping vertically
- shift images horizontally up to 0.2 of image width
- shift images vertically up to 0.2 of image height

IV. METHODS

A. *Convolutional Neural Networks*

Convolutional neural networks (CNNs) have been shown to successfully learn a hierarchy of features for a variety of computer vision and image processing tasks. CNNs are a "specialized kind of neural network for processing data that has a known grid-like topology" [15]. Basic neural networks do not scale well to images, whereas convolutional neural networks can "take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way" [16]. Figure 3 visualizes a Basic neural network.

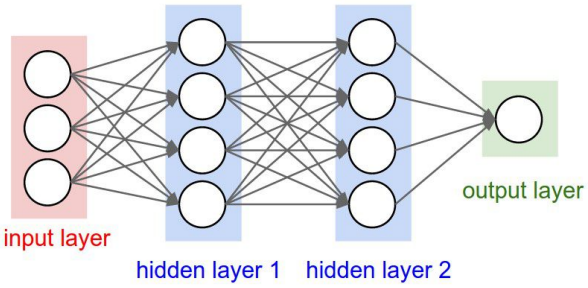

Fig. 3. A basic 3-layer Neural Network [16]

Figure 4 visualizes a typical convolutional neural network where layers of the CNN transforms the 3D input volume to a 3D output volume of neuron activations.

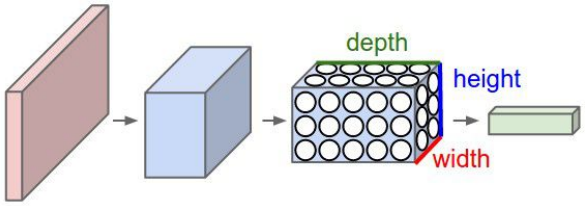

Fig. 4. A Convolutional Neural Network [16]

We suspect that a convolutional neural network would be useful for this type of problem where we need to take into account texture, color, and shape information to classify classes of data. To justify the use of convolutional neural networks, we explain what CNNs are.

"Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers" [15]. In the most basic of terms, convolution is an operation on two functions of a real-valued argument. The convolution operator, usually denoted by an asterisk, takes an input x and applies a kernel w to produce a feature map s

$$s[n] = (x * w)[n] = \sum_{k=-\infty}^{\infty} x[k]w[n-k]$$

In machine learning applications, the inputs and kernels are multi-dimensional and are usually called tensors. To motivate the use of the convolution, we note several key ideas: sparse interactions, parameter sharing, and equivariant representations [15].

1) Sparse Interactions: Whereas traditional neural networks use matrix multiplication to describe the interaction between each input neuron and each output neuron, CNNs use kernels, which are of smaller size than the input, in order to detect smaller, meaningful features. This allows for more efficient parameter storage and faster output computation since there are naturally fewer operations.

2) Parameter Sharing: In traditional neural networks, each element of the weight matrix is used exactly once when computing the output of a layer [15]. In convolutional neural networks, rather than learning a separate set of parameters for every location, we learn only one set for every kernel [15]. Since the kernel size is drastically smaller than the input image size, convolutions are "dramatically more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency" [15].

3) Equivariance: A function is equivariant if when the input changes, the output changes in the same way. With images, the convolution creates a 2-D map of where certain features appear in the input. If an object in the input is shifted, its representation will be shifted by the same amount. With convolutional neural networks, it is common to detect edges in the first layer. The same edges appear more or less everywhere in the image, so it is practical to share parameters across the entire image. It is important to note that convolution is not equivariant to scale or rotations.

Now that we have a motivation for using convolutions, we explore the basic layers that combine to form a CNN.

- **Convolutional layers** make up the most computationally intensive part of the network. The parameters of a convolutional layer include learnable filters or kernels. Recall, these filters are of smaller size than the input size dimension. These filters are convolved to obtain a multi-dimensional activation map. "Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color" [16].
- **Pooling layers** replace the output of the net at a certain location with a summary statistic of the nearby outputs [15]. Variations of pooling downsamples the input to "progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network" which also helps control overfitting [16]. Pooling operations tend to help with spatial invariance as well.
- **Dropout** is a form of regularization by only keeping a neuron active with some probability p or setting it to zero otherwise [16]. This technique forces the network neurons to generalize by randomly dropping neuron connections.
- **Batch Normalization** forces the activations throughout a network to take on a unit gaussian distribution [16]. Normalization is a differentiable operation, and networks that use Batch Normalization are more robust to bad initialization [16]. This technique "can be interpreted as doing preprocessing at every layer of the network" [16].
- **Fully Connected layers** are also known as Dense layers. These types of layers make up traditional neural networks. They have full connections to all activations in the previous layer, and are used towards the ends of CNNs in order to calculate class scores.

We combine the aforementioned layers in order to create custom CNN models. We name the models Custom 0 and Custom 1 for easy reference. The architectures are drawn in Figure 5.

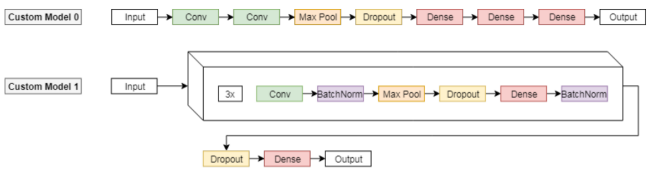

Fig. 5. Custom CNN Architectures

B. Transfer Learning

We also explore the use of Transfer Learning with the popular architectures VGG16 and XCeption whose architectures are shown in Figures 6 and Figure 7 respectively. Transfer Learning is a technique in deep learning in which we use the convolutional layers of a pre-trained deep learning model as

a form of "feature extraction" for a different classification problem. In theory, a convolutional neural network learns a rich hierarchy of features which may be useful for multiple problems. Many modern deep learning frameworks come with pre-trained CNN architectures. With transfer learning, we can "freeze" the convolutional layers during backpropagation. This means the network weights will not change during backpropagation, which is akin to not learning. However, we then append a non-frozen dense linear layer after the convolutional layers in the architecture to satisfy our output requirements (in our case, seventeen outputs). We would only retrain this final layer in order to better suit our classification problem. Despite being trained on different datasets, popular CNNs have shown to be powerful feature extractors that are capable of generalizing to new data and can thus be re-purposed for our new image classification problem. In our experimentation, however, we implement a variant of the aforementioned Transfer learning called "fine-tuning". With this technique, we do not freeze any layers, but instead we allow all the transferred weights in the network to update with backpropagation. We found this yielded better results with small samples of data, so our report will focus on this "fine-tuning" approach. Furthermore, we simply modify the architectures input and output layers of VGG16 and XCeption to fit the input image dimension and the output label dimension.

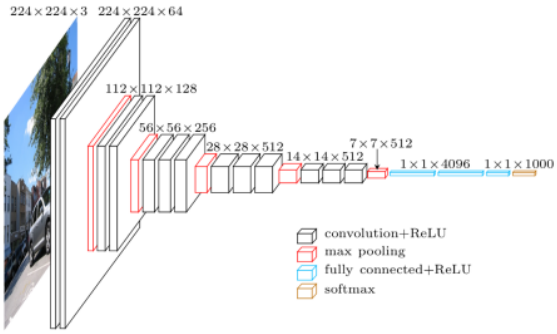

Fig. 6. VGG16 Architecture [17]

The architectures VGG16 and XCeption were chosen due to their popularality in image classification. The VGG16 network is built using 3x3 size Convolutional layers that are stacked on top of each other in the order of increasing depth. To reduce the volume size, 2x2 size Max pooling layer is applied. Afterwards, the network contains two fully-connected layers of 4,096 nodes, which is followed by a sofmax classifier. Again, we make minor changes to the architecture to fit our classification problem. Similar to VGG16 is the VGG19 network which instead contains 19 layers.

The XCeption network is an extension of the Inception network, where the Inception network extracts multi-level features by computing 1x1, 3x3 and 5x5 convolutions and output of these convolutions are stacked along the channel dimensions before feeding into the next layer in the network. In the XCeption network, the Inception module is replaced with depthwise separable convolutions, as shown in the "Middle flow" section in Figure 7. This allows the network to map spatial correlations for each output channel separately. XCeption network tends to outperform the Inception network on Image classification datasets [18].

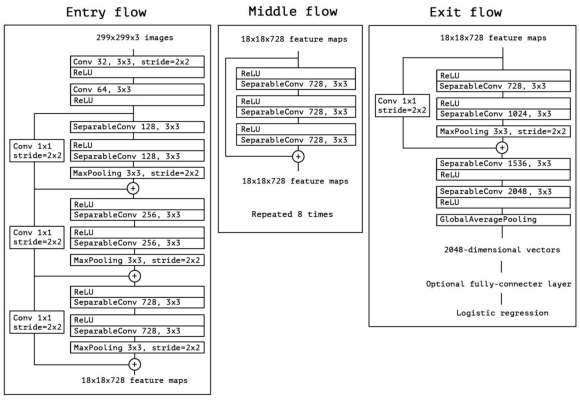

Fig. 7. XCeption Architecture [19]

V. EXPERIMENTS

Since this is a multi-label classification problem, we must use binary cross-entropy loss with individual sigmoid activations on the last layers. Typically, one uses softmax on the final layer, but this does not work with multi-label classification as the labels are not mutually exclusive. We would like to train a network that can recognize the labels individually, which can be satisfied by using the sigmoid activation function at the final layer.

We use a batch size of 128 images. Furthermore, recall the training/validation split is 80/20. In practice, we tested convergence (on custom 0) using batch sizes of 512, 256, and 128 and observed minimal difference. Using the smallest batch size tended to converge quicker so we stick with batch size of 128. We use the Keras implementation of the Adam optimization algorithm which has a default learning rate of 0.001 and no decay [2]. In order to prevent overfitting, we utilize a patience parameter of 3 defined in Keras callbacks in order to stop training. Patience sets the number of epochs with no improvement in validation loss after which training will be stopped. We use validation loss for our metric during training. However, Kaggle utilizes Fβ score with $\beta = 2$ in order to rank the competitors.

$$f\beta = (1+\beta^2)\frac{p \times r}{\beta^2 p + r}$$

$$p = \frac{\text{true positives}}{\text{true positives + false positives}}$$

$$r = \frac{\text{true positives}}{\text{true positives + false negatives}}$$

To see why F2 score is important, we observe the data distribution of our training set in Figure 8.

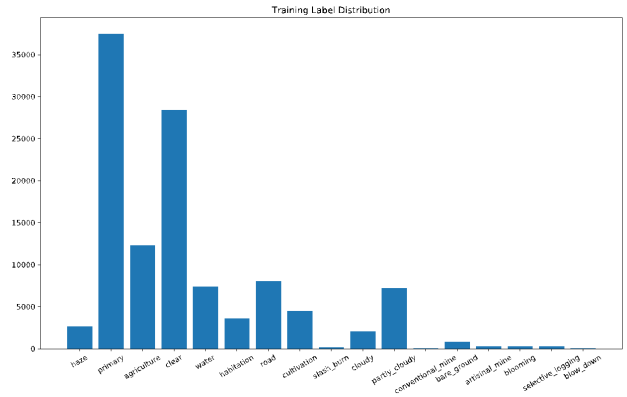

Fig. 8. Training Label Distribution

Overfitting via always predicting the majority classes of "primary" and "clear" must be avoided and could indicate artificially low training error and high training accuracy; therefore, the F2 metric is much more meaningful than accuracy. Using a patience parameter based on F2 score would not work well since Keras implements the callback using batches and is approximated global-wise. [1] It is misleading to implement a batch-wise F2 score, so we only check F2 score on the validation set.

We provide the final data of our training, validation, and testing in Table I. Kaggle F2 scores are obtained by using late submission entries on the Kaggle competition.

TABLE I
F2-SCORE RESULTS

Networks	Trainable Parameters	Validation F2	Kaggle F2
Custom 0	6,510,289	0.87931	0.87766
Custom 1	826,257	0.87987	0.87736
VGG16	14,749,527	0.91042	0.90845
XCeption	21,120,319	0.92201	0.91918

We can see that XCeption and VGG16 perform the best. Our XCeption model performance is fairly close to the top Kaggle score, which is 0.933. For reference, the Spring 2017 ECE 228 Paper on this same topic had achieved a Kaggle F2 score of 0.90003 [20]. We expect good performance from transfer learning due to use of pre-trained features obtained from the ImageNet dataset which contains 1000 classes. Fine-tuning to our dataset proves to be the best approach. We show the training process using metrics such as training loss, validation loss, and validation accuracy in Figures 9, 10, 11, 12. We note the decrease in validation loss over epochs in all figures. We can observe variability in validation loss in Figure 10, which would suggest that our learning rate is too high, and we should use decay. Furthermore, note that VGG16 trained for more than 25 epochs whereas XCeption trained for 6 epochs. The early termination is a result of the aforementioned patience parameter. We can also see the accuracy of all the models starts above 95%. This corroborates our claim that accuracy does not give us meaningful information about our model, so we use the F2 metric.

Our custom models would likely never do better than the fine-tuned VGG16 and XCeption. We used our general knowledge of CNN architectures to construct them. While our custom networks train from scracth, transfer learning with fine-tuning leverages pre-trained weights that need to be modified slightly in order to fit new datasets. In other words, they start near a minima and require little effort to achieve great performance. This is evident by the nearly flat validation loss data in Figures 11 and 12.

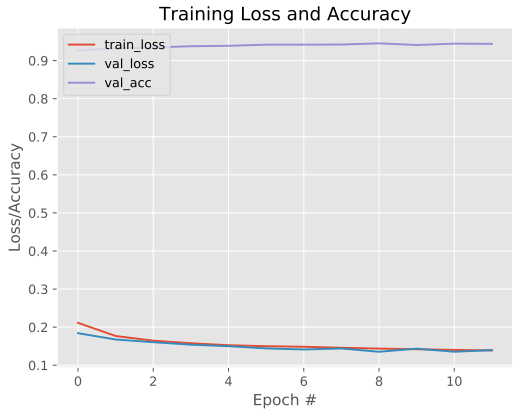

Fig. 9. Custom 0 - Learning

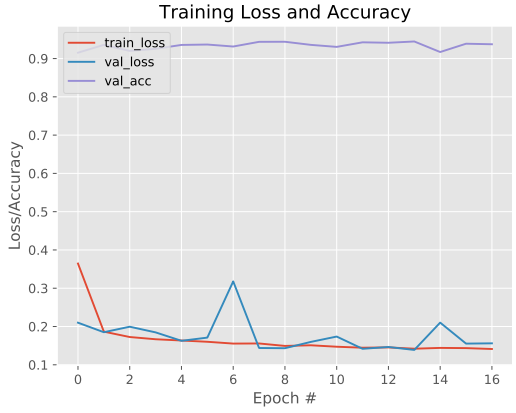

Fig. 10. Custom 1 - Learning

[1] https://github.com/keras-team/keras/issues/5794

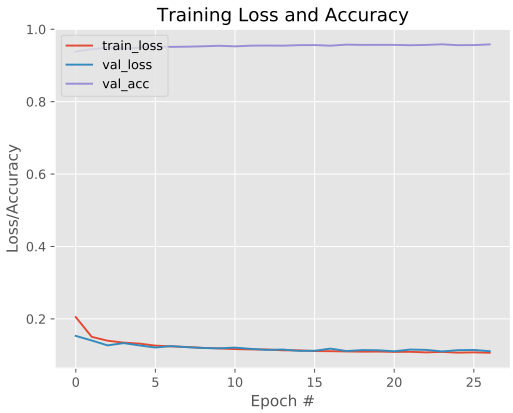

Fig. 11. VGG16 - Learning

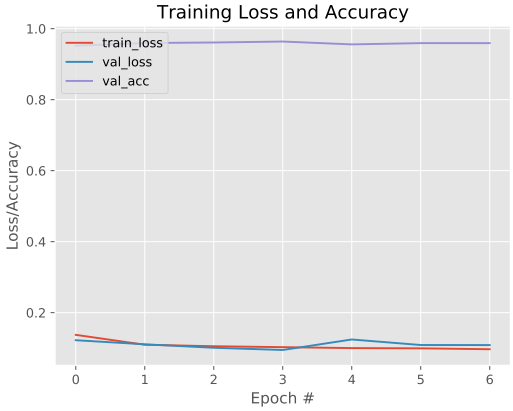

Fig. 12. XCeption - Learning

VI. CONCLUSIONS

We found that transfer learning with fine-tuning on XCeption resulted in the best F2 score. We expected our custom networks not to perform as well as VGG16 and XCeption. The popular architectures came with features pre-trained on 1000 classes with the ImageNet dataset. These architectures are state-of-the-art and can be expected to generalize well to unseen data. Initially, we were unsure how well the pre-trained features would generalize, but the results speak for themselves. The Kaggle top scores are in the range of 0.93, and we were close to 0.92. The custom networks also corroborate our thoughts about convolutional neural networks performing well for images. Custom 0 had few convolutional layers and had many dense layers whereas Custom 1 emphasize convolutional layers with pooling. Despite having less than 13% of the parameters of Custom 0, the deeper Custom 1 achieved nearly the same F2 score. This indicates that convolutional layers along with pooling can leverage certain properties encoded in images and that dense layers have parameters that are wasteful.

VII. FUTURE WORK

With more time and resources, we would explore the use of the fully sampled dataset and could test our models on the 4-channel GeoTiff format. Furthermore, ensemble methods in which we train several models and have them vote on the output labels typically lead to better generalization. This type of model averaging is also a form of regularization [15]. We would also like to explore Hintons Capsule Networks as an alternative to CNNs and their applicability to this domain [21]. We initially explored this type of network, but quickly found that this is a young technology with few educational resources.

ACKNOWLEDGMENTS

We would like to thank Professor Peter Gerstoft and TAs Mark Wagner, Paolo Gabriel, and Nima Mirzaee for their continued advice and encouragement in the completion of this project. We further thank Amazon Web Services for free student tier for GPU computation which allowed us to train and test models.

REFERENCES

[1] "Planet: Understanding the amazon from space — kaggle," https://www.kaggle.com/c/planet-understanding-the-amazon-from-space.

[2] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[4] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov 1986.

[5] S. Mallat and S. Zhong, "Characterization of signals from multiscale edges," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 7, pp. 710–732, Jul 1992.

[6] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, Apr 1980. [Online]. Available: https://doi.org/10.1007/BF00344251

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

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

[9] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. [Online]. Available: http://arxiv.org/abs/1311.2901

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[13] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[14] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR*, vol. abs/1610.02357, 2016. [Online]. Available: http://arxiv.org/abs/1610.02357

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[16] A. Karpathy, "Convolutional neural networks for visual recognition," https://cs231n.github.io/convolutional-networks/.

[17] D. Frossard, "Vgg in tensorflow," Jun 2016. [Online]. Available: https://www.cs.toronto.edu/ frossard/post/vgg16/

[18] A. Rosebrock, "ImageNet: Vggnet, resnet, inception, and xception with keras," March 2017. [Online]. Available: https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

[19] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR*, vol. abs/1610.02357, 2016. [Online]. Available: http://arxiv.org/abs/1610.02357

[20] G. Shi and Z. Gu, "Use satellite data to track the human footprint in the amazon rainforest." [Online]. Available: http://noiselab.ucsd.edu/ECE285/FinalProjects/Group18.pdf

[21] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *CoRR*, vol. abs/1710.09829, 2017. [Online]. Available: http://arxiv.org/abs/1710.09829