Pri-Matrix Factorization: Automated Species Tagging in Video Clips

Niranjan Balachandar
Department of Computer Science
Stanford University
niranja9@stanford.edu

Andrew Wei Chen
Department of Computer Science
Stanford University
awchen@cs.stanford.edu

Jiwoo Lee
Department of Computer Science
Stanford University
jlee29@stanford.edu

Abstract

Wildlife camera traps, which capture video clips when triggered by activity, are fundamental tools in wildlife ecology and conservation. However, with the vast amount of data produced by these cameras, discriminating among various animal species and misfired, animal-less videos requires substantial human overview, which renders camera traps infeasible for many ecological applications. We implement and evaluate a number of machine learning approaches to automate the classification of video clips captured from camera traps. We demonstrate that a deep long-term recurrent convolutional network, formed by a combination of a convolutional neural network and a recurrent long short-term memory network, achieves the highest classification performance on a large dataset of camera trap video clips, with a Top-1 accuracy of 72.6%.

1. Introduction

Research in wildlife ecology provides insight into the evolutionary origins of human behavior and physiology, and enables improved conservation efforts. Human understanding of primates and other wildlife is driven through observation – however, direct human surveillance can disrupt natural behaviors and ecosystem dynamics, and is difficult to scale.

Recently, automated cameras have been used to address these issues. These automated cameras, also known as camera traps, are used to capture wildlife on film with minimal disturbance. These systems are triggered by motion or heat, and do not discriminate between different species nor false triggers. As such, labeling of the species found in the resulting video recordings requires large crowdsourcing efforts or substantial input from wildlife experts, which renders the usage of camera traps untenable to many research institutions [1].

Here we develop models to automatically identify the presence and absence of certain animal species in video recordings from camera traps. We will be using the dataset from the Pri-matrix Factorization competition hosted by the Max Planck Institute for Evolutionary Anthropology [2].

In this research, the input to our algorithms are downsampled wildlife video clips, and the output is a 24-element vector corresponding to 23 animal categories plus one category corresponding to no animal. The value for each output category is the model's confidence that the respective label is present in the video clip.

2. Related Work

A multilayer perceptron is a vanilla feedforward neural network in which non-linear transformations project the input data into a linearly-separable format. Convolutional Neural Networks (CNNs) are variations on multilayer perceptrons that have enabled recent advances in computer vision, especially for large-scale image classification tasks [3–6].

Recurrent Neural Networks (RNNs) are another type of deep learning model that have been useful for sequence learning tasks, such as learning with time-dependent data [7]. Videos are time-dependent sequences of images, so it would make sense to combine CNNs and RNNs for video classification. In fact, these approaches are concurrently leveraged in the Long-Term Recurrent Convolutional Neural Network (LRCN), a deep learning architecture designed for label classification in videos [8]. This architecture feeds each frame of a video into a CNN, the output of which is jointly fed into a stack of Long Short Term Memory (LSTM) networks. A summary of this architecture is provided in Figure 3.

LSTMs are a type of RNN that enable long-range learning. Normally, backpropagated error evolves exponentially due to the repeated rounds of multiplication involved in long-range backpropagation. This tends to result in vanishing/exploding errors, which render learning untenable. LSTMs enforce additive error flow across temporal steps, and therefore enable long-range learning [9].

3. Dataset/Features

The competition provides training and testing sets [2]. Model training and calibration is performed on the training set, and the competition evaluates predictions on the testing set. The training set consists of 204,130 labeled video clips, and the testing set consists of 87,484 unlabeled video clips. Each video clip is 15 seconds long, is in color, and includes an audio channel. Each video clip in the training dataset also comes with a set of 24 binary labels; each of 23 labels indicates the ground truth presence of a particular species, and the 24th label indicates whether there are no species present. We use 30% of the training set (61,239 video clips) for the validation set and the rest (142,891 video clips) for training. The amount of raw data is huge (> 1 TB), so the competition provides significantly downsampled versions of the data, in which the framerate and resolution of the images are decreased, and the audio channel is excluded. We utilize the `micro` dataset, in which the video clips have been downsampled to 30 frames and $64\text{px} \times 64\text{px}$ resolution. Examples of the data are shown in Figure 1. We used code provided by the competition to fill in missing frames and split the data into training and validation sets [10,11]. These data are input into our deep-learning models as four-dimensional arrays of size $(30, 64, 64, 3)$ containing RGB intensities for each pixel. For logistic regression, our input was vectorized. The data was zero-centered and scale-normalized.

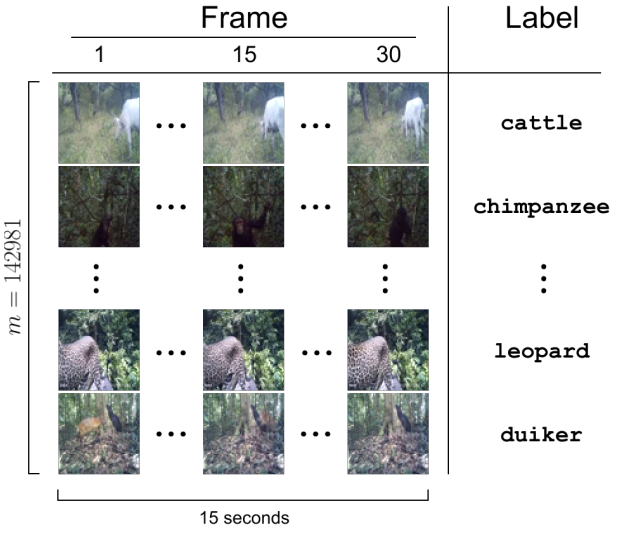

Figure 1: A sample of data from our training set.

During training, we perform real-time data augmentation with random horizontal flipping with probability 0.5, and random shading and contrast adjustment, given by $x^a + b$ for each pixel RGB value x, where $a \sim \mathsf{normal}(1, 0.05)$ and $b \sim \mathsf{normal}(0, 16)$. An example of an image augmentation transformation is shown in Figure 2.

Finally, the final feature vector used in k-means segmentation is (sx, sy, H, S, V), where (H, S, V) is the hue, saturation, value color at a point (x, y), and s is an optimized scaling factor. This is discussed in more detail in Section 4.1.

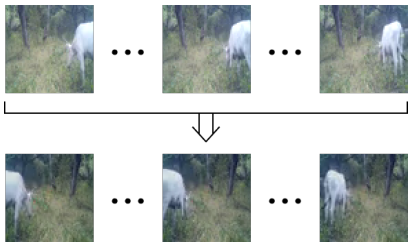

Figure 2: An example of an image pre- and post-augmentation.

4. Methods

First we use K-means for segmentation to obtain additional input features for our classification models. Then we designed and implemented a number of models that perform 24-way classification. We first implemented a baseline logistic regression model to establish performance benchmarks. We did not expect the baseline to perform well relative to human performance. We then developed deep learning models that incorporate CNNs and MLPs or LSTMs to serve as our main models.

Batch normalization ensures zero-centering and unit variance and has been shown to accelerate training, so we included a batch normalization layer as the first layer of each model as well as before or after each convolutional layer for the CNNs [12]. Each convolutional layer uses the rectified linear unit (ReLU) activation function whose equation is given by: $\text{ReLU}(x) = \max(x, 0)$. We use ReLU activation for the convolutional layers to avoid gradient saturation that has been shown to result from sigmoid and tanh activations. The final transformation layer of each of these models are a dense layer with output vector size of 24. For the deep models, a dropout layer was included before the final dense layer for regularization. A softmax layer is applied at the very end of each of the deep learning models to generate class probabilities. For the LRCN models, we apply gradient clipping to a max norm of 5 to prevent exploding gradients.

4.1. K-Means for Segmentation Features

Many videos contain visually-distinct animals, with the typically green forest background contrasting with the natural colors of the animals. To take advantage of this, we investigated the efficacy of k-means applied to the segmentation of animals in our video dataset, with the goal of using segmented images as additional classification features.

In our initial approach, we read each frame from the video clip through Imageio [13], and naively clustered through pixel location (x, y) and RGB values, where (R, G, B) are the 0-255 RGB color values at the pixel location. Centroid number was optimized for $k \in \{2, 3\}$, and a scaling factor s was included to scale the position terms and the RGB values, resulting in a final feature vector (sx, sy, R, G, B). We then performed k-means with scikit-learn [14]. Qualitative evaluation of the resulting figures showed low segmentation efficacy (Figure 5).

In rectifying this result, we noted that human-level segmentation is not accurately represented by the RGB color model. As a result, we re-implemented k-means using the Hue, Saturation, Value (HSV) color model, which is designed to reflect human color perception. We converted the RGB values of each feature vector to the corresponding HSV values, such that the final feature vector was given by (sx, sy, H, S, V), where s is re-optimized for the HSV feature vectors.

4.2. Logistic Regression Baseline

As a baseline, we implemented logistic regression on our videos by flattening the video inputs into vectors in $x \in \mathbb{R}^{\text{frames} \times \text{height} \times \text{width} \times \text{colors}}$. The output is given by $\sigma(Wx)$ where x is the input video vector, σ is the logistic function, and $W \in \mathbb{R}^{(\text{frames} \times \text{height} \times \text{width} \times \text{colors}, 24)}$

4.3. Vanilla CNN + MLP

Motivated by the LRCN architecture, we implemented a CNN to extract feature vectors of size 1024 from each frame of the input. The CNN architecture is very simple, so we refer to it as a vanilla CNN; it consists of 4 blocks of the form [batch normalization layer, convolutional layer, max pooling layer], and a 5th block of the form [batch normalization layer, convolutional layer]. We wanted to determine whether an MLP could substitute LSTM in LRCNs, so we then concatenated frame features into one vector, and input them into an MLP with two hidden layers of size 256 and 64 respectively. While there is no time-dependency encoded into an MLP, the MLP may be able to learn this relationship through sufficient training.

4.3.1 Vanilla CNN + LSTM

To encode an explicit time dependency in our model, we used an LSTM in place of the MLP. A summary of our LRCN is shown in Figure 3. The frame-wise CNN feature extractor is the same as the one for the MLP- output vectors of size 1024 for each frame. These output vectors are input into an LSTM with a single hidden layer. A dense layer with softmax activation is applied to the output of the final timestep of the LSTM to generate predictions of label probabilities.

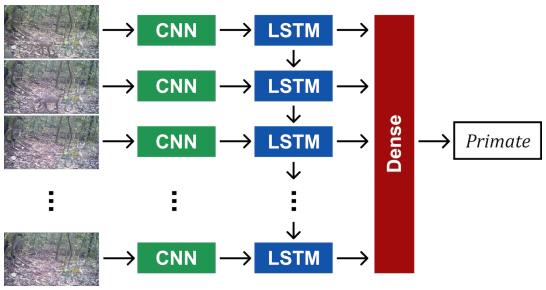

Figure 3: High-level summary of LRCN architecture.

4.3.2 Inception v3 + LSTM

After we noticed underfitting in our previous models, we increased the complexity of our CNN architecture. We explore various complex CNN architectures, including MobileNet and VGG19 [4, 15]. Ultimately we replaced our vanilla CNN with a modified version of Inception v3, a high-performing CNN architecture designed for ImageNet [6]. Inception v3 has inception modules, as shown in Figure 4, that increase network depth without exploding the number of parameters, allowing massive depth without limitations in computational complexity and model size. Our modified implementation removes 9 of the 159 original layers as our 64px $\times$ 64px frames are smaller than typical inputs to Inception v3 (usually images of size 299px $\times$ 299px). We modified the Keras implementation of Inception v3 [16].

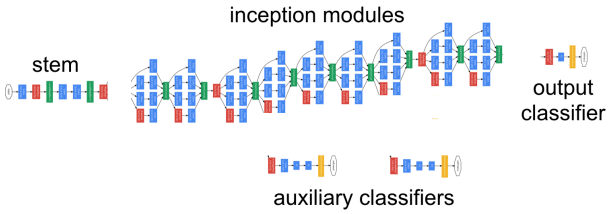

Figure 4: High-level summary of Inception v3 components.

5. Analysis

5.1. Experiments

We implemented the learning models in Keras with TensorFlow backend, and trained and evaluated the models on multiple Google Cloud instances with a NVIDIA Tesla K80 GPU [17, 18]. Additionally, we used starter code provided by the competition to imeplement our models [19]. We used Adam to perform gradient descent on randomly sampled minibatches. Adam incorporates adaptive learning rate and parameter updates for faster convergence in practice than standard SGD [20]. We trained models from scratch rather than fine-tune existing models as we were unable to find readily available trained models for our input size, and did not want to significantly alter our data through preprocessing.

We performed hyper-parameter tuning of k and s (the (x, y) scaling factor) for k-means segmentation, and the initial learning rate α, learning rate schedule, minibatch size, gradient clipping norm, LSTM hidden layer size, dropout rate, and L2-regularization coeffient for the learning models. Due to limitations in time and computational resources, we performed hyper-parameter tuning on a small subset of the training set ($N = 3200$). We observed that a learning rate of 0.001 and gradient clipping norm of 5.0 resulted in fastest initial training. We implemented learning rate annealing to prevent overshoot of the minima, so whenever there are 5 epochs of no improvement in validation loss, the learning rate was lowered by a factor of 10. We set minibatch size to 32, as this was the maximum possible minibatch size

permissible under memory constraints. Smaller minibatch sizes resulted in very long epochs (several hours), so they were impractical. We set LSTM hidden layer size to 256, as smaller sizes resulted in underfitting, and larger sizes did not improve performance. For the models that were underfitting (all except for inception LRCN), we set the dropout rate and L2 coefficient to 0, and excluded data augmentation. For the inception lRCN, because it was slightly overfitting, we set the dropout rate to 0.2, the L2 coefficient to 0 as L2 did not seem to improve performance when we trained on a small subset of the training set, and augmented training samples whose true labels are rare (less than 5% of the training set).

Because the competition evaluates submissions on aggregated multi-label binary log-loss, this will be our evaluation metric as well as our loss function during training. The aggregated multi-label binary log-loss equation is given by the following equation:

$$J = -\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\left(y_j^{(i)}\log(\hat{y}_j^{(i)}) + (1-y_j^{(i)})\log(1-\hat{y}_j^{(i)})\right)$$

where M is the total number of inputs videos, N is the number of classes (24 for this task), $y_j^{(i)}$ is the ground truth indicator label for the presence of animal j in video i, and $\hat{y}_j^{(i)}$ is the predicted probability for the presence of animal j in video i. We will also use Top-1 accuracy as another evaluation metric.

5.2. Results

Qualitative results for k-means segmentation are included in Figure 5, model configurations and classification results are included in Table 1, loss curves during training are included in Figure 6, and qualitative examples are included in Figure 7.

Model	Logistic Regression	CNN+ MLP	Vanilla LRCN	Inception LRCN
# Parameters (Mil.)	8.8	11.5	7.6	8.0
Depth[1]	1	8	9	150
Avg. Epoch Time (hrs.)	0.6	1.2	1.9	3.9
Top-1 Acc. (train)	0.611	0.655	0.704	0.749
Top-1 Acc. (val)	0.603	0.634	0.697	0.726
Loss (train)	0.304	0.112	0.082	0.047
Loss (val)	0.298	0.111	0.090	0.057
Loss (test)[2]	0.296	0.121	0.084	—[3]

Table 1: Configuration summary and results of our models.
[1]Depth refers to the total number of convolutional, LSTM, or dense layers.
[2]Test set losses are calculated by the competition upon submission of predictions on test set.
[3]Code for calculating test set predictions on Inception LRCN is still running, so we are not yet able to submit the test set predictions to the competition to receive and report test set loss. However, we anticipate it will be similar to the validation set loss of 0.057.

5.3. Discussion

K-means outputs for some examples in Figure 5 indicate that k-means is a promising approach for segmentation. The

Figure 5: K-means segmentation examples. First column includes frames from sample original videos, the second column includes corresponding output from k-means with RGB features, and the final column includes corresponding output from k-means with HSV features.

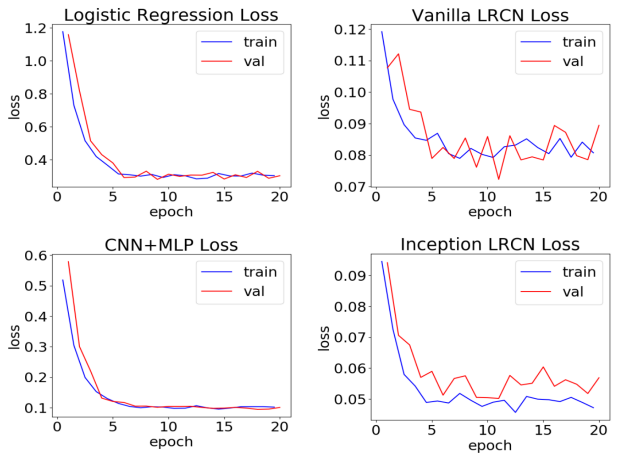

Figure 6: Training and validation losses over the course of training.

top example in Figure 5 is a leopard, and the HSV features clearly yield more visually accurate and less noisy segments than RGB features. Even for the second example in Figure 5, where the chimpanzee is not clearly distinguishable from the background, HSV k-means yields an accurate segmentation. Intuitively, we ought to use $k = 2$ because we are only interested in two segments: animal and not animal. However, for some examples, like the third in Figure 5, $k = 3$ offers better results than $k = 2$, so further investigation needs to be done to determine best value for k, and whether k needs to be different across images. Also for some examples, like for the duiker in the fourth example in Figure 5, HSV and RGB segmentation are both noisy, and it is unclear whether

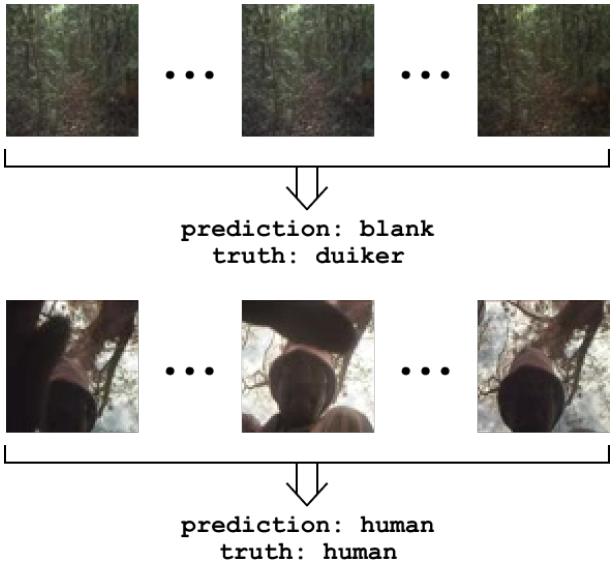

Figure 7: Inception LRCN class predictions for examples from validation set. The true label for the top video is 'duiker', but Inception LRCN misclassified it as 'blank'. The true label for the bottom video is 'human', and Inception LRCN correctly classified it.

the k-means output of blank videos would be helpful. Overall, k-means with HSV segmentation yields decent qualitative segmentation for most animals. We intended on using k-means outputs as additional features for our models. Due to time limitations we were unable to run k-means on the entire dataset, so our current models do not incorporate k-means outputs, but it is a promising future endeavor.

As we can see from losses and accuracies from Table 1 as well as from the loss curves in Figure 6, the logistic regression, CNN + MLP, and LRCN all underfit. This suggests either these models themselves are not suited for fitting our data, our hyperparameter choices were not optimal, or we did not train these models for enough time. We trained each model for 20 epochs, but it is possible that we need to train even longer to improve performance. Because the dataset is large, each epoch takes hours, which limited the extent of our hypermarameter search and training time. Because the true label of 59.8% of the examples in our dataset is 'blank', a classifier that naively predicts 'blank' for all inputs would result in 59.8% accuracy. Logistic regression did not perform much better than this. Based on our results for the three underfitting models, we suspected that it is the case that logistic regression model and CNN+MLP model are not suited for the task, and the vanilla LRCN does not have deep enough CNN. Our suspicions were proven correct when when we replaced the vanilla CNN with Inception v3, as model performance improved.

Overall, the LRCN models outperformed all other models in validation and test set loss and accuracy. This implies that the LSTM was effective in capturing the time-dependent nature of our video data. The Inception LRCN performed the best out of all the models, indicating that a deep CNN is the most effective in extracting features from video frames. Interestingly, the number of model parameters did not seem to affect model performance, as all our models had roughly the same number of parameters. Because incorporating a deep CNN and incorporating an LSTM both improved performance, the LRCN was particularly powerful for our video classification task. While the inception LRCN performed the best, the validation accuracy of 72.6% implies there is still room for improvement. Note that the test set loss reported by the competition is still pending due to continuing computation, and test accuracy is not computed by the competition. Based on analysis of correct and incorrect classifications by the Inception LRCN on examples from the validation set, we identified that the model is accurate at classifying humans. This is because in these camera trap videos, the humans are usually researchers who adjust the camera, causing clear frame-to-frame distortions that can be picked up by an LSTM (see bottom video in Figure 7). We also identified that our model often misclassifies videos as 'blank'. For example, the top video in Figure 7 has a barely-visible duiker in the bottom right corner that does not move. With 64×64 resolution, the duiker is barely visible even for a human. This is the case for many of the examples misclassified as 'blank'. Our dataset's 64×64 frames are heavily downsampled from the original raw videos, so it will likely be the case that we will see big gains in performance from using higher resolution videos.

6. Conclusion/Future Work

In summary, we have implemented a model to automate species labeling in video clips. We have demonstrated that the Inception LRCN is the best model, achieving the highest performance with a Top-1 accuracy of 72.6% and loss of 0.057 on the validation set, followed by the vanilla LRCN, the CNN + MLP, and logistic regression, with Top-1 accuracies/losses of 69.7%/0.090, 63.4%/0.111, and 60.3%/0.298 on the validation set, respectively. We have also demonstrated that k-means is highly effective at segmenting animals in wildlife camera trap video clips, particularly when using the HSV color model.

With additional time and computational resources, there are a number of extensions that we'd like to pursue, to further improve our models. First, we intend to utilize the raw videos in training, which we anticipate will effect substantial improvements, due to the high FPS and high resolution native to the raw videos. Additionally, we plan to incorporate the audio channel of the videos as a feature set for our models. Furthermore, we aim to fine-tune our best-performing models; due to our large and general dataset, we expect overfitting from fine-tuning to be less of a concern. Finally, we will incorporate images segmented through k-means as features for our classification models.

Ultimately, we have constructed an accurate video classifier to classify animal species in video clips, and we expect that many of our results are generalizable to important applications in numerous other domains.

7. Contributions

Niranjan Balachandar implemented and performed experiments with logistic regression and the LRCNs, Andrew Chen designed and implemented the k-means modeling, and Jiwoo Lee implemented and performed experiments with CNN+MLP. All three students contributed equally to the paper.

References

[1] C. Boesch, H. Kuehl, M. Arandjelovic, and N. Maldonado, "Chimp & See." https://www.chimpandsee.org.

[2] "Pri-matrix Factorization." https://www.drivendata.org/competitions/49/deep-learning-camera-trap-animals, 2017.

[3] 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.), pp. 1097–1105, Curran Associates, Inc., 2012.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[5] 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 Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June-2015, pp. 1–9, 2015.

[6] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015.

[7] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015.

[8] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 677–691, 2017.

[9] S. Hochreiter and J. J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1–32, 1997.

[10] "multilabel.py." https://github.com/drivendataorg/box-plots-sklearn/blob/master/src/data/multilabel.py, 2017.

[11] "primatrix_dataset_utils.py." https://gist.github.com/drivendata/70638e8a9e6a10fa020623f143259df3, 2017.

[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[13] "Imageio." https://imageio.github.io/, 2014.

[14] "scikit-learn." http://scikit-learn.org/, 2017.

[15] 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*, vol. abs/1704.04861, 2017.

[16] "Keras Inception v3 Implementation." https://github.com/keras-team/keras/blob/master/keras/applications/inception_v3.py, 2017.

[17] "Keras Documentation." http://keras.io/, 2017.

[18] "TensorFlow." http://tensorflow.org/, 2017.

[19] "Pri-Matrix Factorization Benchmark." http://drivendata.co/blog/pri-matrix-factorization-benchmark/, 2017.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.