What *is* the Best Classifier?
What *is* the Best Regressor?

Timothée Guédon   Antoine Hébert   Jean Lucas de Souza Toniolli

Abstract

This report investigates two questions. First, for a given selection of data sets, can we say what is the 'best' classifier or the 'best' regressor in terms of good predictions? How much does the answer depend on the particular selection of data sets? How much does the answer depend on our computational constraints? We investigate these questions using data sets from the UCI repository. To ensure each algorithm is given an equal hyperparameter tuning budget, we perform automated hyperparameter tuning. Second, we compare the interpretability of a decision tree classifier to that of a convolutional neural network. We compare the decision tree visualization to 'activation maximization' and 'class activation mapping', two techniques to gain insight into the kinds of inputs that deep neural networks respond to.

1. Introduction

Many machine learning algorithms exist and it can sometimes be hard for machine learning practitioners to know which algorithm to use on a new dataset. In this project, we compare some of the main machine learning algorithms for both classification and regression tasks. We evaluate the performances obtained by each of these algorithms on a set of datasets from the UCI repository [5]. To make a fair comparison of the different algorithms, it is necessary to ensure good hyperparameters are used so that performances obtained correctly represent the capabilities of the algorithm. We used automated hyperparameter tuning based on Tree-structured Parzen Estimator [2] to find good hyperparameters for each model and dataset. The prediction performance is not always the only factor to take into account when choosing which algorithm to use, the training time and difficulty of the hyperparameter search can also matter. To account for that, and to ensure a fair comparison, for each dataset, all algorithms are given the same hyperparameter tuning time. We included the following algorithms in our comparison of classifiers: k-nearest neighbors algorithm, support vector machine, decision tree, random forest, adaptive boosting algorithm, logistic regression, Gaussian naive Bayes, gradient boosted trees algorithm and two implementations of a neural network classifier, the basic implementation from the sci-kit learn library [13] and our implementation using PyTorch [12], including more advanced techniques such as dropout and batch normalization. In our comparison of regressors, we included support vector machine, decision tree, random forest, adaptive boosting algorithm, Gaussian process algorithm, linear regression, and two implementations of a neural network regressor, like for classification the sci-kit learn library implementation and a more advanced implementation using PyTorch.

When using machine learning to make important decisions, it is sometimes necessary to be able to understand how the model makes these decisions. We compare the interpretability of two algorithms: the decision tree and the convolutional neural network on a computer vision task using the CIFAR-10 dataset [9]. We show how we can easily visualize a decision tree to try to understand how the model makes its decision. We also present two methods to try to interpret the output of a convolutional neural network: activation maximization [6], and class activation mapping [16]. We designed a small convolutional neural network with decent performances for the needs of this study.

Our source code has been designed to make it easy to include additional datasets or algorithms and to perform a more extensive study on them.

2. Methodology & Experimental Results

For each dataset, we evaluate the performances of the different algorithms using a held-out test set. For datasets that already provide a separate test set we use it directly, for other datasets, we use a random stratified sample preserving the proportion of classes and containing 25% of the examples. For each dataset and each algorithm, in order to find good hyperparameters, we perform automatic hyperparameter tuning using the library Hyperopt [1]. The main advantage of this library is the possibility to parallelize hyperparameter tuning, on one computer or even on a cluster of computers. We used this feature to ran multiple tuning trials in parallel on one powerful machine.

To perform hyperparameter tuning we created for each algorithm a distribution of hyperparameter configurations using our experience with each algorithm and advice gathered online. The hyperparameter tuning algorithm needs an objective function which returns the loss value corresponding to a hyperparameter configuration. We used 7-fold cross-validation, and aggregated the metric values obtained with each fold to compute the loss. The aggregation method used depends on the metric, for the accuracy we simply use the average of accuracies reported for each fold but for other metrics a simple average is sometimes not a good estimator. With the F1 score, we compute the aggregated F1 score from the sum of the confusion matrices obtained for each fold as advised in Forman and Scholz [7]. With root mean squared errors, we compute the aggregated value by taking the root of the mean of the means squared errors obtained for each fold.

As mentioned in the introduction, for each dataset we give the same hyperparameter tuning time to all algorithms. Some algorithms with more hyperparameters or longer training time probably need more time than others to find the optimal hyperparameter configuration. However, we decided to not only evaluate the prediction performances of each algorithm but also its practical interest when a limited hyperparameter tuning budget is available. For most datasets we tune each algorithm for a thousand seconds, for the adult dataset and the default of credit card clients dataset which are significantly bigger in size, we tuned each algorithm for three thousand seconds. In order to prevent a pathological hyperparameter configuration to waste too much tuning time, we set a maximum time after which a tuning step is interrupted.

Once good hyperparameters have been found, we train a final model with the whole training set. We evaluate the performances of this model on the held-out test set using the same metric as the hyperparameter tuning. In order to compare the global performances of the different algorithms across datasets, we compute the average rank obtained by each algorithm on the various datasets. We summarize the result using a critical difference diagram [4].

For each dataset, we consider each feature either as a numerical feature or as a categorical feature. Categories for which there is a clear ordering are encoded as numerical features. Other categorical variables are one-hot encoded except when doing binary classification and using a tree-based model. In this case, we encoded them as suggested in The Elements of Statistical Learning [8] in Section 9.2.4. Instead of using one-hot encoding which would create an exponential number of possible splits, we index the categorical variable ordered by the proportion of positive examples among the examples belonging to the given category. This encoding guarantees optimal splits on these categorical variables [8]. For datasets with missing values, we use the iterative imputer from the sci-kit learn library [13] which makes use of a Bayesian ridge model to estimate the missing values based on the other values. For models that are not based on trees or when we use the iterative imputer, we normalize the features using standardization.

2.1. Classification Experiments

For some of the datasets, we made some small preprocessings. For the retinopathy dataset, we decide to ignore the quality binary feature because its value is true for almost all examples. For the Yeast dataset, duplicate examples were present in the dataset, we removed them. For the thoraric surgery dataset, we encoded the size of the original tumor feature as a numerical feature even though it was presented as a categorical feature in the dataset. We believe it is can be considered as an ordinal feature. For the diagnosis categorical feature, we merged the categories with less than 15 examples. To evaluate the classification performances, we used either the accuracy or the F1 score. We used the accuracy for datasets for which classes are balanced. When one of the class is less frequent and when the class of interest is the minority class, we used the F1 score. We used the F1 score for the default of credit card clients dataset, the breast cancer dataset, German Credit Data dataset, the adult dataset, the yeast dataset, the thoraric surgery dataset, the seismic bumps dataset, and the steel plates faults dataset.

2.2. Regression Experiments

For some of the datasets, we did some pre-processing. For the Merck molecular activity dataset, we ignored features for which more than half of the examples had a missing value. The iterative imputer was not able to impute the remaining values efficiently, so we imputed them with the mean value of the feature in the training set. For the Parkinson sound recordings dataset, we ensured examples from the same subject were either present in the test set, the validation set or the training set, in order to evaluate the performances of the model on new subjects. For the Facebook metrics dataset, there are many possible dependent variables, we split the dataset into 4 different regression tasks: the prediction of the number of likes, shares, and comments. We encoded the post month and post hour variables as cyclical features, that is to say, that we derive two features from each of these features using the cosine and sinus functions in order to make extreme values close to each other in the resulting feature space. For the bike-sharing dataset, we encode the features hour and month as cyclical features. For the GPU kernel performance dataset, we apply the binary logarithm function on all variables but the four ones so that they variate linearly. We derive four different tasks from the student performances dataset: the prediction of the math grades with or without the grades of the two first periods and the prediction of the Portuguese grades with or without the grades of the two first periods. To evaluate the regression performances, we used the root mean squared errors for all datasets except the Merck molecular activity dataset for which we used the coefficient of determination score as recommended in the Kaggle competition instructions.

2.3. Results of the algorithms comparison

For both classification and regression tasks, we look at the global ranking, the ranking on small datasets and the ranking on big datasets. For the classification task, we chose to consider as big datasets the Default Credit Card and the Adult datasets. Both have more than 30,000 samples while others only have from 470 to 2500 samples. For the regression task, the Merck Molecular activity, the Bike Sharing and the GPU Kernel Performances datasets are considered as big datasets, they have more than 10 thousand samples while the other ones have up to 2 thousand samples. Note that the model comparison on big datasets is less statistically significant than the comparison on the small datasets due to the small number of big datasets.

For the classification task, the most effective model seems to be the random forest algorithm. It had the highest global rank with a global mean ranking of 2.9 over 9 models evaluated. It also has the best mean rank on the big datasets, while having the third-best rank for small datasets. The SVM and logistic regression models also seem to be particularly effective: they both are in the top 3 models in the global ranking and the small dataset ranking. However, they seem to have limited capacity with handling big datasets: they have been ranked 6 and 7. On the other hand, more complex models like the advanced neural network, AdaBoost and the gradient boosting models are performing better on big datasets. Their respective ranks on big datasets are 2nd, 3rd and 5th. On smallest datasets, those complex models are at the bottom of the rank list. Interestingly, the basic neural network model performs better than the advanced one on small datasets, but its performance is worse than the advanced neural network on the big datasets. This results in the advanced one having a better global mean rank than the simple one. Our neural networks also seems to be affected by the presence of many categorical variables in the dataset, it would be interesting to see how the use of embeddings could help them.

For the regression task, gradient boosting seems to be the best algorithm, overall, but the random forest algorithm is still the best on big datasets and second overall. Also consistent with the classification part, the SVM model is in the top 3 for the global and small datasets rankings. For bigger datasets, the basic neural network is the second-best followed by the gradient boosting algorithm.

As we can see from the results, caution must be taken with the global ranking metric as there are not as many big datasets as small datasets and we found that the dataset size has a great impact on the performance of a model. The random forest algorithm seems to be a very interesting model,

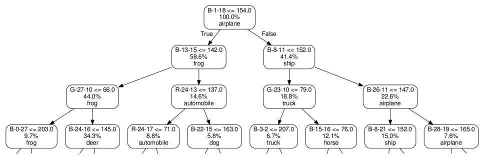

Figure 1. Decision Tree Interpretability

performing well on both classification and regression, for both small and big datasets. Simple models seem preferable on small datasets, while more complex models like neural networks seem to have more capacity to handle bigger datasets.

2.4. Interpretability Experiments

We compare the interpretability of a decision tree and a convolutional neural network (CNN) on a computer vision task: classification of images using the CIFAR-10 dataset [9]. We leveraged our hyperparameter tuning code to tune the decision tree. The hyperparameters we used are presented in table 4.

The decision tree reaches an accuracy of 31% on the test set. This result shows the decision tree is not suitable for computer vision tasks when no features are available. Figure 1 shows a visualization of the first 3 depth of the decision tree. This visualization does not provide much insight into how the model makes its prediction.

We have built our own small convolutional neural network for this experiment. We carried out the calculations using "Google Colaboratory" to speedup our experiments using an efficient GPU. We measured the accuracy on a separate validation set for model selection and early stopping.

We first started with a simple model called LeNet5 [11]. With small adjustments so that it can accept 32x32x3 input images and the addition of two fully connected layers before the output layer, we obtained 57% accuracy on the test set after 10 epochs. Using the ReLU activation function first introduced in the AlexNet architecture [10], increased the accuracy to 62%. Inspired by the spatial pyramid pooling architecture used in semantic segmentation [3], we decided to try using multiple resolutions in the CNN and to combine their output activation maps (2). This further increased the accuracy. Then we added dropout [10] and increased the number of feature maps. We later switched to channel dropout modules like recommended with CNN [15]. We remarked that retraining the network 2 or 3 times during 10 epochs was better than training it for 20 or 30 epochs directly. We, therefore, decided to use a cyclical learning rate policy [14] which increased the accuracy to 73% in 45 epochs. The paper introducing this learning rate policy [14] also introduced a method to easily find the optimal learning rate. This method works by training a model starting from a small learning rate and increasing it gradually, in

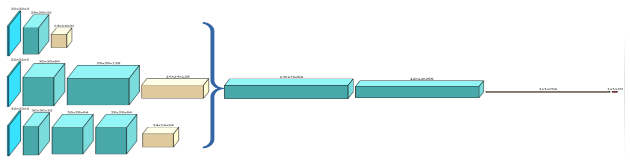

Figure 2. Final architecture of our CNN, teal boxes correspond to convolutional layers, white ones to pooling layers and the red one to a fully connected layer.

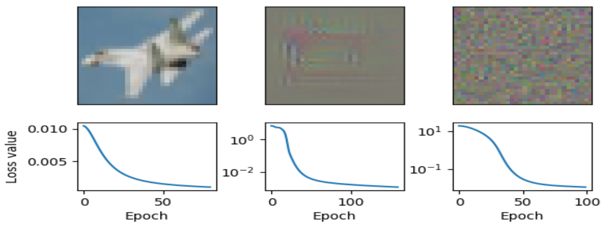

Figure 3. Activation maximization results and learning curve when using image initialisation, zero initialisation and uniform initialisation and when maximizing the airplane class activation

order to generate a graph like the one in Figure 10. The optimal learning rate is the one for which the loss decreases the fastest. We late switched to the cosine annealing with warm restart policy, added data augmentation and increase the model size to achieve 83% accuracy. Once we had a neural network architecture with good performances, we transferred it from Google Colaboratory to the repository of our project. We used the "Skorch" library in order to leverage the tuning framework we developed for the algorithm comparison. This library allows training a PyTorch neural network using the same API than Sci-kit learn. After integration, we continued the tuning of the hyperparameters and added an additional convolutional layer. We reached 91% accuracy. Finally, we also added a global average pooling layer after the last convolutional layer and kept only one final fully connected layer in order to make it possible to use the class activation mapping interpretation technique.

We used activation maximization [6] in an attempt to understand how our model identifies an image as belonging to one class. Activation maximization consists of finding the input data which maximizes the probability for the model to identify this data as belonging to one class. We did not manage to make this method useful for the interpretation of the results of a CNN. We used momentum in an attempt to improve the optimization process, but it did not produce better visualizations. We initialized the input data randomly with a normal distribution of mean zero and several different standard deviations without finding a better one. We also tried initializing the input data with zeros, and to use an example from the test dataset.

No clear pattern appears in the results of activation maximization as we can see for example with the class airplane in Figure3. We noticed that when the class whose activation is maximized match the class of the image the loss value cannot be reduced much, which is logical.

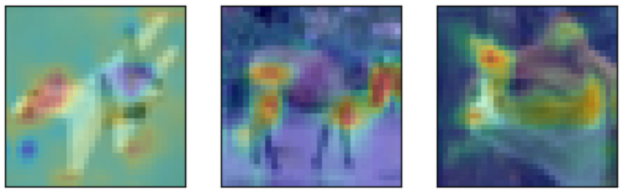

Figure 4. Class activation mapping results for 3 correctly identified images from the test set. The redder the more this part of the image is used by the model

Since the activation maximization method did not produce interesting results with our model, we tried another method proposed to interpret the result of a convolutional network. Class activation mapping [16] make it possible to see which parts of the image have contributed the most to the output of the model. This method is quite simple, since the last convolutional layer is followed by a global average pooling, the weights of the fully connected layer directly give the contribution of each feature map to the prediction of each class. Therefore, we can get a map of the total activation of each pixel by multiplying the weights of the last layer corresponding to the predicted class to the activations of the last convolutional layer.

As we can see in Figure 4, the class activation mapping method gives much more interesting results. We see that the model seems to identify a plane using the angular shapes of its wings and of its nose, to identify a horse using the shape of its head and its legs, and to identify a cat using the shape of its ears and its muzzle.

3. Conclusions

To conclude, there is no clear winner. Some algorithms indeed seem to perform better on average, but the best algorithms are only third on average. Therefore, we recommend trying several algorithms when starting working on a new dataset, we would recommend experimenting at least with random forest and SVM with a small dataset, and adding an advanced neural network when the dataset is quite big. Automatic hyperparameter tuning can help to effortlessly try several algorithms on a new dataset.

As we have seen, despite the simplicity of the decision tree algorithm, it can be very hard to interpret when the decision tree is very deep. On the opposite, despite deep convolutional neural networks (CNN) complexity, some methods are successful to understand how the model makes its prediction. Therefore, we believe CNNs are more interpretable than decision trees for computer vision.

A. Overview of project code and data

To install the prerequisites of the project, the command ‘conda env create -f environment.yml’ can be used to create a conda environment named ‘ml-project’ containing all the necessary prerequisites to run the project code. Once in the conda environement the command ‘conda develop .’ can be used to install the project code.

The entry point of our project code is the file ‘main.py’. When launched it will execute the task corresponding to the first positional argument specified. For each of the project parts, there is a command to run the tuning of models and one to train and evaluate the models. In addition, there is a command to generate interpretation visualizations and one to generate the results tables and diagrams. If no argument is specified the command help is displayed.

All the datasets used are downloaded from their sources if they are not already present in the directory ‘results’. It is necessary to install a Kaggle API key and accept the rules of the competition to be able to download the Merck molecular activity dataset.

All the data produced and used is stored in the ‘data’ directory. The best hyperparameters found as well as the values of the metrics measured on the test and validation sets for each model and each dataset are stored in this directory.

The source code is organized in four main packages, one per project part, and the ‘utils’ module which contains all source code used by several parts of the project.

B. Additional experimental results and illustrations

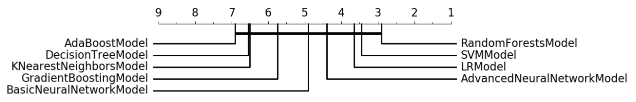

Figure 5. Critical difference diagram for classification

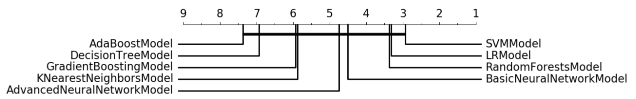

Figure 6. Critical difference diagram for classification with small datasets

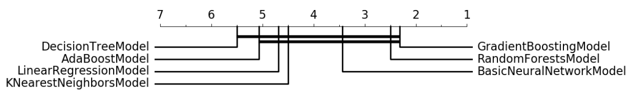

Figure 7. Critical difference diagram for regression. The SVM algorithm is not included because it failed on some datasets.

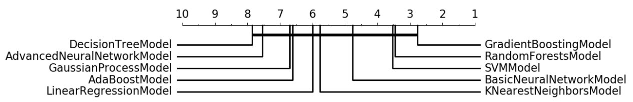

Figure 8. Critical difference diagram for regression with small datasets

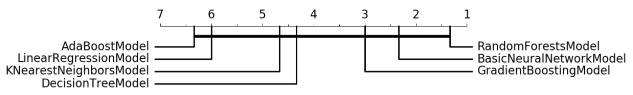

Figure 9. Critical difference diagram for regression with big datasets. The SVM algorithm is not included because it failed on some datasets.

dataset	AB	ANN	BNN	DT	GB	KNN	LR	RF	SVM
Adult	2	4	7	6	3	9	5	**1**	8
BreastCancer	7	3	4	9	5	8	6	**1**	2
DefaultCreditCard	8	2	6	4	7	9	5	**1**	3
Retinopathy	5	3	5	8	4	9	2	7	**1**
SeismicBumps	9	6	3	5	8	7	**1**	4	2
StatlogAustralian	6	9	3	6	6	**1**	3	2	6
StatlogGerman	5	**1**	6	7	8	9	2	4	3
SteelPlatesFaults	9	7	5	6	**1**	4	8	2	3
ThoraricSurgery	9	4	3	5	8	7	**1**	6	2
Yeast	8	5	6	9	7	2	3	**1**	4
Average	6.9	4.4	4.9	6.6	5.8	6.5	3.6	**2.9**	3.4
Average Big Datasets	5.0	3.0	6.5	5.0	5.0	9.0	5.0	**1**	5.5
Average Small Datasets	7.4	4.8	4.5	6.9	5.9	5.9	3.3	3.4	**2.9**

Table 1. Ranking of classification algorithms

dataset	AB	ANN	BNN	DT	GP	GB	KNN	LR	RF	SVM
BikeSharing	9	5	2	4	6	3	8	10	**1**	7
CommunitiesAndCrime	8	7	4	9	10	2	6	3	5	**1**
ConcreteCompressiveStrength	8	6	3	7	4	**1**	9	10	2	5
FacebookComment	3	7	**1**	10	6	2	8	4	9	5
FacebookLikes	9	10	**1**	8	7	3	4	5	6	2
FacebookShare	10	8	7	9	5	**1**	4	3	2	6
MerckMolecularActivity	9	6	3	8	F	**1**	7	5	2	4
ParkinsonMultipleSoundRecording	3	10	9	7	2	6	**1**	5	8	4
QsarAquaticToxicity	8	4	3	10	5	7	6	9	**1**	2
RedWineQuality	8	7	6	9	4	3	2	10	**1**	5
SGEMMGPUKernelPerformances	6	F	2	3	F	5	4	7	**1**	F
StudentMathPerformance	3	8	5	4	10	2	9	7	**1**	6
StudentMathPerformanceNoPrevGrades	7	5	8	4	10	**1**	9	6	2	3
StudentPortuguesePerformance	4	9	6	7	10	5	8	3	2	**1**
StudentPortuguesePerformanceNoPrevGrades	7	10	3	8	9	**1**	6	4	5	2
WhiteWineQuality	8	7	6	10	5	2	3	9	**1**	4
Average	6.9	7.3	4.3	7.3	6.6	**2.8**	5.9	6.2	3.1	3.8
Average Big Dataset	8	5.5	2.3	5	6	3	6.3	7.3	**1.3**	5.5
Average Small Dataset	6.6	7.5	4.8	7.8	6.7	**2.8**	5.8	6	3.5	3.5

Table 2. Ranking of regression algorithms

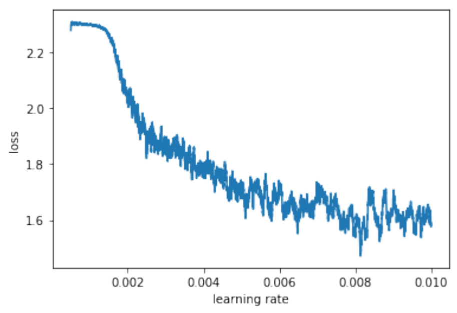

Figure 10. Smoothed loss graph obtained with the learning rate finding method [14]

		AB	ANN	BNN	DT	GB	KNN	LR	RF	SVM
dataset	metric									
Adult	f1	0.70	0.68	0.67	0.67	0.70	0.62	0.67	0.71	0.66
BreastCancer	f1	0.93	0.95	0.94	0.91	0.94	0.92	0.94	0.95	0.95
DefaultCreditCard	f1	0.46	0.53	0.47	0.52	0.47	0.44	0.49	0.54	0.53
Retinopathy	acc	0.71	0.74	0.71	0.67	0.72	0.66	0.74	0.67	0.77
SeismicBumps	f1	0.00	0.25	0.28	0.27	0.10	0.20	0.32	0.28	0.30
StatlogAustralian	acc	0.83	0.82	0.83	0.83	0.83	0.85	0.83	0.84	0.83
StatlogGerman	f1	0.59	0.66	0.57	0.54	0.52	0.48	0.64	0.59	0.63
SteelPlatesFaults	f1	0.61	0.71	0.75	0.72	0.82	0.76	0.68	0.80	0.77
ThoraricSurgery	f1	0.00	0.27	0.30	0.26	0.08	0.16	0.36	0.24	0.35
Yeast	f1	0.38	0.55	0.52	0.35	0.46	0.58	0.57	0.60	0.56

Table 3. Classification results

Hyper-Parameter	Value
class_weight	balanced
max_depth	13
max_features	0.69
min_samples_leaf	48
splitter	best

Table 4. Hyper-parameters for Decision Tree on CIFAR10 dataset.

References

[1] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *Jmlr*, 2013. 1

[2] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011. 1

[3] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 3

[4] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006. 2

[5] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. 1

[6] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009. 1, 4

[7] George Forman and Martin Scholz. Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *ACM SIGKDD Explorations Newsletter*, 12(1):49–57, 2010. 2

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001. 2

[9] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 1, 3

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 3

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NeurIPS Autodiff Workshop*, 2017. 1

[13] 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*, 12:2825–2830, 2011. 1, 2

[14] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017. 3, 5

[15] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656, 2015. 3

[16] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016. 1, 4