INF8953CE - KAGGLE COMPETITION

Team Decision Three - Polytechnique Montréal
Bruno Curzi-Laliberté, bruno.curzi-laliberte@polymtl.ca, 1933468
Masoud Karami, masoud.karami@polymtl.ca, 2050716
Thomas Rochefort-Beaudoin, thomas.rochefort-beaudoin@polymtl.ca, 1842175

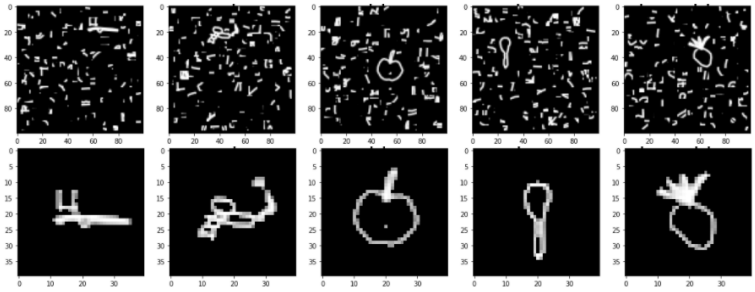
1 INTRODUCTION

This report describes the work done to design a doodle classifier based on a modified version of the *Google Quick, Draw!* dataset, containing hand-drawn doodles of 31 different classes. First, we present the various steps taken during the feature design process in order to remove the artificial noise from the images and crop the images around the doodles. Then, the various algorithms used as well as the ensembling method are presented, followed by a description of the methodology taken to train the various classifiers. Finally, we present the accuracy results of the various classifiers which enabled a final predictions score of 85.471% to achieve 2nd place in the Kaggle competition that took place during the Machine Learning class of Professor Sarath Chandar at Polytechnique Montréal during the Fall semester of 2020. We conclude with a discussion on further potential work and the various advantages and disadvantages of our proposed classifier.

2 FEATURE DESIGN

For this project, we are using a variant of the publicly available *Google Quick, Draw!* dataset[1] which consists of 50 million hand-drawn doodles from 345 different categories (Figure 1a). In this modified version, the training and test set consists of 10000 images each, distributed across 31 different classes. The images were created by appending the drawings to a random location on larger 100×100 pixel blank canvas images. Random artificial noise was also added to make the competition more challenging. The dataset also includes an "empty" class consisting of only noise with no human drawing.

(a) Google's quick draw dataset examples.



(b) Example of images before and after some preprocessing.

Figure 1: a) Original Quick! Draw dataset. b) Denoised and cropped images examples.

To simplify the classification task, two pre-processing steps were taken: noise removal and re-centering on the doodle to reduce the size of the images, thus making training more efficient.

To extract the actual drawings from the artificial noise, we leveraged the continuity feature of hand-drawn doodles to use the contour finding function `findContours` from the Python `OpenCV` library. The function `findContours` returns an image where each closed contour is filled using

[1] The Quick, Draw! Dataset

the *border following algorithm* (Suzuki et al. (1985)). Then, the largest convex hull is found and only the pixels inside the hull are kept, essentially removing every pixel not adjacent to the main contour. For the "Empty" class, this showed to be sufficient since the largest contours extracted are usually very small and the CNN was able to identify it correctly.

To re-center the image on the doodle, the largest bounding box was calculated and used to crop each doodle on its center of mass. This step allowed us to transform the dimensions from 100x100 to 40x40, an 84% reduction of the amount of pixels in each image. This enabled the CNN models to compute more quickly. Figure 1b visually presents the "before-after" of our pre-processing steps. Finally, the pixel values were normalized in the interval $[0, 1]$.

3 ALGORITHMS

It was required to create a baseline linear classifier for the problem. We chose to implement a logistic regression model and its results are presented. Then, the real classifier composed of a stacking ensemble of various CNN architectures with various meta-models is described.

3.1 LOGISTIC REGRESSION

Logistic regression is a discriminative classification model where a linear decision surface is constructed by modeling the logit of the probability $P(C_k|x)$ with a linear combination of the features X. Let $p_i = P(y_i = 1)$ and $(\mathbf{x}_i \in \mathbb{R}^d)$. The logistic regression function (*logit*) is defined as follow:

$$\log \frac{p_i}{1 - p_i} = \beta^T \mathbf{x}_i \qquad \text{or} \qquad p_i = \frac{\exp\left(\beta^T \mathbf{x}_i\right)}{1 + \exp(\beta^T \mathbf{x}_i)}$$

where $\beta \in \mathbb{R}^d$ are some unknown regression coefficients often estimated using maximum likelihood.

The algorithm implementation from Scikit-learn (Pedregosa et al. (2011)) using the *Limited-memory Broyden-Fletcher-Goldfarb-Shanno* (L-BFGS) solver was used. Hyperparameter tuning of the regularization parameter C was done by randomized search since it has been shown to be more efficient than grid search and manual tuning (Bergstra & Bengio (2012)). The logistic regression model achieved a 51.20% test accuracy (Table 1 and Table 2).

3.2 CONVOLUTIONAL NEURAL NETWORK

Convolutional layers make use of a convolution kernel which can be trained to extract local features in an image. By stacking multiple of these kernels together with pooling layers, a deep convolutional neural network can learn complex image features by exploiting the 2D structure of the input. Deep convolutional neural networks have shown great accuracy on similar classification tasks such as the Fashion MNIST (Xiao et al. (2017)) and EMNIST datasets (Cohen et al. (2017). Those two grayscale image datasets consist respectively of clothing objects and handwritten characters, which are somewhat similar to the grayscale doodle dataset of this challenge. The following architectures were trained from scratch on the preprocessed training set using their respective implementation from the Keras library (Chollet et al. (2015)):

- ResNet50v2 (He et al. (2015))
- ResNet101v2
- DenseNet121 (Huang et al. (2018))
- MobileNetV2 (Howard et al. (2017))
- DenseNet201

We also trained 12 different variations of a CNN built upon a common baseline architecture which is composed of two CNN "blocks" followed by a single dense "block" (Figure 2). Each CNN block combines convolution layers with MaxPooling layers, with batch normalization and dropout layers between each convolution layer for regularization purposes (see section 4.3). The dense block is composed of dense layers and dropout layers.

From the baseline architecture, we generated multiple variants to try to increase the classification accuracy. Those modifications include increasing the number of filters, the number of CNN blocks, changing the kernel size and modifying the activation function. In total, 15 different CNNs were trained and were combined in our stacking ensemble.

Table 1: Logistic Regression Classifier Accuracy

DATASET	ACCURACY
Training	61.18%
Validation	49.55%
Test	51.20%

Table 2: Logistic Regression Hyperparameter tuning

VARIABLE RANGE	BEST VALUE
C : loguniform(1e-5,1e0)	0.01255

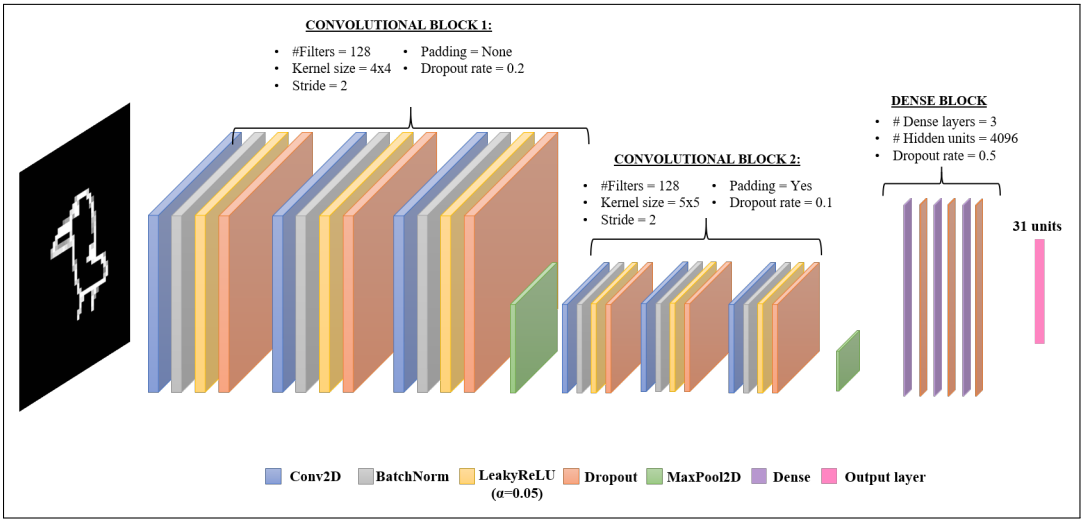


Figure 2: Basic CNN architecture ($cnn3_1$).

3.3 Stacking ensemble

Stacked generalization is an ensemble method where predictions from multiple classifiers are combined to form a training set for a meta-model which is trained to combine optimally the various predictions together in order to generate final predictions (Wolpert (1992)). We used 5 different meta-models which are described in Table 3. Final predictions were generated by majority-voting between the meta-models. Although it may seem counter-intuitive to use the stacking classifier at the same level as the other meta-models, it resulted in better validation accuracy and allowed us to generate our final score. An illustration of our stacking architecture is presented in (Figure 3).

4 Methodology

4.1 Training-Validation Split

We decided to generate a validation set from our training set using an $80-20$ split. The validation set consisting of 2000 samples was then also split into an additional validation set for the meta-model training. From the initial 10000 images training set, 8000 were selected for CNN training, 2000 for validation of the CNN training. Of that validation set, 1600 are for meta-model training, and 400 images are used for meta-model validation. The small size of the meta-model validation set showed to be a problem during meta-model selection, causing variance between the validation accuracy and test accuracy.

4.2 Data Augmentation

To reduce overfitting during training, and increase the generalization capacity of our CNNs, various data augmentation techniques were used. Simple techniques used are: randomly shifting the height

Table 3: Description of meta-models.

Meta-model	DESCRIPTION
A	Logistic Regression ($C = 0.26$)
B	Gaussian Naive-Bayes($\alpha = 4.4$)
C	KNN ($k = 7$)
D	Random Forest ($Maxdepth = 49$)
Stacking Classifier	Logistic Regression ($C = 1.0$)

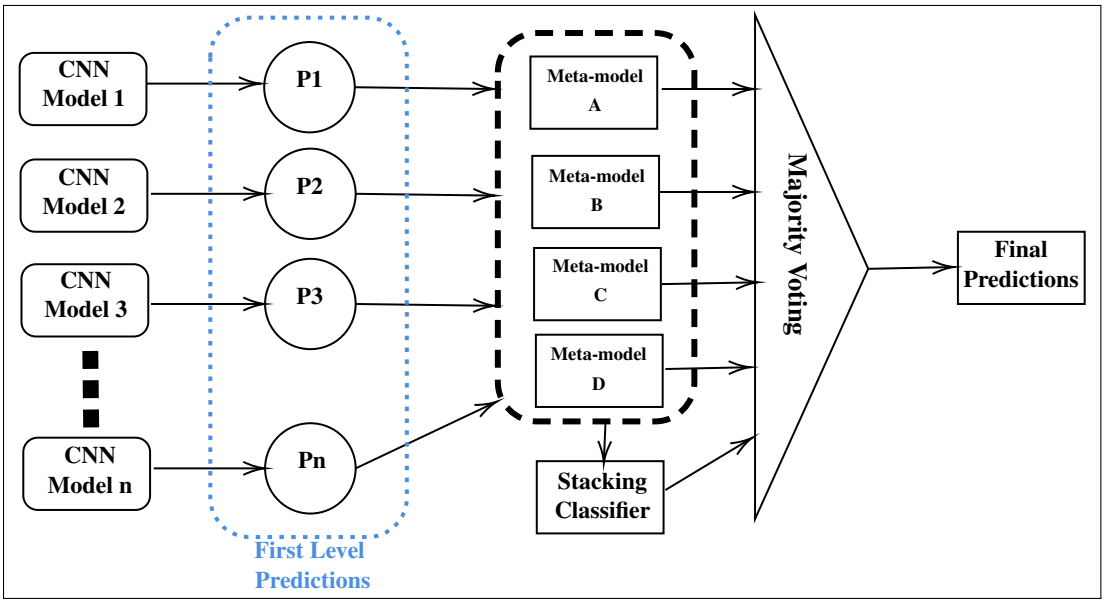


Figure 3: Stacking ensemble architecture.

and width of the image, as well as horizontal flips. More modern techniques such as mixup and random erasing were also used.

4.2.1 MIXUP

Mixup is a simple data augmentation technique to increase the generalization capacities of neural networks (Zhang et al. (2018)). This technique has been used to improve state-of-the-art accuracy on various datasets such as ImageNet-2012, CIFAR-10 and CIFAR-100. A variant of mixup is also used in the second best performing classifier on Fashion-MNIST (Harris et al. (2020)). Mixup generates virtual training examples by randomly selecting two images from the dataset and *blending* them using the following linear interpolation:

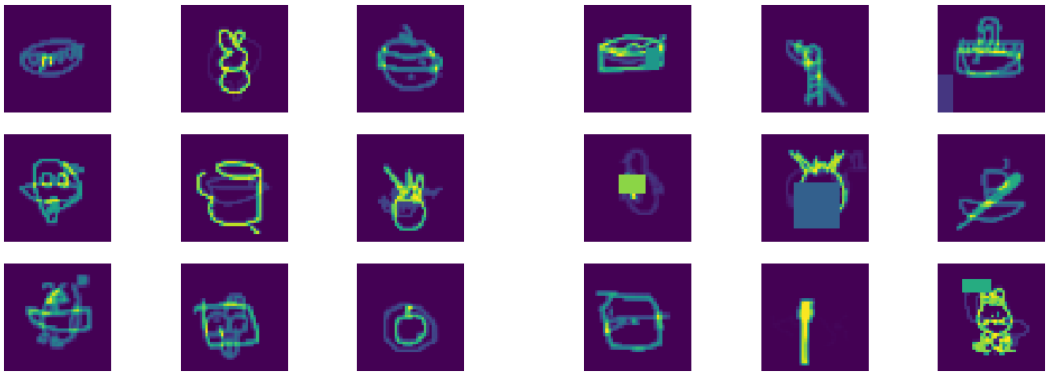
$$\tilde{x} = \lambda x_i + (1 - \lambda)x_j \tag{1}$$

$$\tilde{y} = \lambda y_i + (1 - \lambda)y_j \tag{2}$$

x_i, x_j are input images, and y_i, y_j are the associated one-hot image labels. The two samples i, j are randomly sampled from the dataset. Examples of "mixed-up" images are presented in Figure 4a.

4.2.2 RANDOM ERASING

Random erasing is another data augmentation technique used to increase the robustness of CNN and reduce overfitting during training (Zhong et al. (2017)). The technique consists of randomly selecting a rectangle of pixels in the image and replacing it with random pixel values, thus "erasing"



(a) Mixed-up training images.

(b) Mixed-up with random erasing training images.

Figure 4: Samples of augmented images using mixup and random erasing.

a portion of the image, and making it harder for the algorithm to ”memorize” images in the training set. This technique is used in the third best performing classifier for the Fashion-MNIST dataset. Examples of images augmented with mixup and random erasing are shown in Figure 4b.

4.3 Regularization Strategy

Our CNNs make use of batch normalization layers. Those layers apply a transformation to its inputs to center the mean near 0 and scale the data to achieve a standard deviation of 1. The layer works best when inferring data similar to that of training data to calibrate the transformation. During the learning process, the normalization helps stabilise the learning, especially for deep neural networks. This helps us in reducing the total number of epochs and achieving more stable results at each epoch.

To further help the training process, our CNNs are also composed of dropout layers that only apply during training. Most of our layers use a dropout rate of 0.1, meaning the weights between the layers where dropout is applied has a 10% chance of being set to 0. This technique helps avoid overfitting to certain features specific to our training examples.

4.4 IMPROVING THE CNN ARCHITECTURE

During our experiments with CNNs, it was clear that the validation accuracy varied by a margin of $\pm 2\%$ due to early stopping or slow convergence. Our approach was to note a model and its accuracy, modify an aspect of the network and train it again to see if the validation accuracy improves or decreases, keeping the versions which increased validation accuracy. This was not done extremely rigorously since a slight decrease or increase in accuracy is meaningless when compared to the possible error margin. This meant that we could only really assume we were on the right track every time the accuracy increased by at least 1% while tuning a parameter.

4.5 IMPROVING ENSEMBLING METHODS

To improve the ensembling method, we first had to determine which meta-model improved accuracy. We tested every multi-class classifier from Scikit-learn, with the simplest, and overall good performer being logistic regression classifier. We then proceeded to train it on all of the CNNs meta-model training set predictions. Removing each CNN one at a time, we managed to identify the right combination of CNNs. Once the best subset of CNNs were selected, we re-trained various meta-models and identified the ones that were most promising based on the meta-model validation set accuracy. Using the Kaggle test results, we were able to identify which meta-models to include in our final majority voting stack.

5 RESULTS

In this section we present results from all the convolutional neural networks and the results of every stack ensemble.

5.1 CNN ACCURACY

The following tables present the accuracy of each CNN trained on the 8000 images of the training set against the 2000 images validation set. The first table presents the accuracy of the Keras pre-implemented models initialized with random weights. The second table presents accuracy of our custom CNN architectures.

Table 4: Keras CNN Accuracy on the Validation Set.

NAME	ACCURACY
$resnet50v2$	81.00%
$resnet101v2$	79.90%
$densenet121$	81.65%
$densenet201$	80.35%
$mobilenetv2$	80.05%

Table 5: Our CNN Accuracy on the Validation Set.

NAME	ACCURACY	LEARN. RATE	DEPTH	KERNEL SIZE	FILTERS
$cnn1_1$	80.00%	0.00100	12	5	32, 64, 128
$cnn1_2$	82.30%	0.00100	12	3	32, 64, 128
$cnn1_3$	81.35%	0.00100	12	3	64, 128, 256
$cnn1_4$	83.75%	0.00100	12	3	64, 128, 256
$cnn1_5$	83.30%	0.00033	12	3	64, 128, 256
$cnn1_6$	83.35%	0.00033	12	3	64, 128, 256
$cnn1_7$	82.65%	0.00010	12	3	64, 128, 256
$cnn1_8$	81.25%	0.00033	12	2	64, 128, 256
$cnn1_9$	83.95%	0.00033	12	3	96, 192, 384
$cnn1_{10}$	84.15%	0.00033	12	2	96, 192, 384
$cnn1_{11}$	83.80%	0.00033	12	4, 3, 2	64, 128, 256
$cnn1_{12}$	83.25%	0.00033	12	3, 4, 5	64, 128, 256
$cnn2_1$	81.55%	0.00033	18	3, 3*4, 5	96, 3*192, 64
$cnn2_2$	84.25%	0.00033	21	5*4, 5	2*64, 3*192, 64
$cnn3_1$	84.85%	0.00100	9	4, 5	128, 128

Note: $cnn1_1$ to $cnn1_3$ use Relu activation while other networks use LeakyRelu

5.2 ENSEMBLING ACCURACY

Our initial meta-model selection consisted of 7 types of classifiers made available by Scikit-learn. Table 6 presents each model and the hyper-parameters (their Scikit-learn name) we tuned along with their range and interval. Table 7 presents the accuracy of each of the best configurations against their validation set of 400 samples.

Table 6: Hyper-parameter Tuning Exploration.

MODEL	PARAMETER	RANGE
Logistic Regression	C	[0.01, 10] with intervals of 0.01,
	tol	$10^{[-5,5]}$ with intervals of 0.1
Gaussian Naive-Bayes	var_smoothing	[0.1, 5] with intervals of 0.1
K Nearest Neighbors	n_neighbors	[1, 100] with intervals of 1
Decision Tree	max_depth	[1, 100] with intervals of 1
	min_samples_leaf	[2, 100] with intervals of 1
Extra Trees	max_depth	[1, 100] with intervals of 1
Random Forest	max_depth	[1, 100] with intervals of 1
Stacking Classifier	subset of models	combinations of {LR, GNB, KNN, RndForest,Extra Trees}

Table 7: Hyper-parameters Results.

MODEL	CONFIGURATION	ACCURACY
Logistic Regression	C = 0.26, tol = 1e-4	88.00%
Gaussian Naive-Bayes	var_smoothing = 4.4	88.25%
K Nearest Neighbors	n_neighbors = 7	88.25%
Decision Tree	max_depth = 40, min_samples_leaf = 28	83.50%
Extra Trees	max_depth = 57	87.50%
Random Forest	max_depth = 49	87.00%
Stacking Classifier	subset = {LR, GNB, KNN}	88.00%
	subset = {LR, GNB, KNN, RndForest}	87.75%
	subset = {LR, GNB, KNN, RndForest, ExtraTrees}	87.25%

Note: There are 3 values for the Stacking Classifier since in the context of a competition we get to try different things and we submitted once with each of those configurations.

Final predictions were generated by majority voting over a set of meta-models, using odd-numbers of meta-models (3 and 5) to avoid ties in voting. First, we used majority voting by using the LR, GNB KNN, this achieved a validation accuracy of 89.00%. Then, we added Extra Trees classifier and the Stacking Classifier built upon LR, GNB KNN into the vote, this achieved a validation accuracy of 88.50%. Submitting both, it turns out that the second model achieved the best test result. After the competition we found that also adding Extra Trees into the Stacking Classifier and doing majority voting with the same models performs even better, achieving a final test accuracy of 85.485%, where the previously described second majority vote achieved 85.471%.

6 DISCUSSION

Overall, the proposed classifier achieved excellent accuracy on the test dataset and managed to score second place in the fall 2020 Kaggle competition of the INF8953CE class. The stacking ensemble managed to score better accuracy than single well-known architectures such as MobileNet and ResNet. The preprocessing steps were extremely simple and cost efficient in terms of required computational ressources.

Our solution does not take into account the class imbalance shown in the training set which could affect the performance of the individual CNNs. Also, our method is built on multiple large convolutional neural networks that are (for the most part) very expensive to train, both in terms of required hardware and time.

GPU access being limited to 35 hours per week per user via Kaggle, we did not have full liberty of testing more CNN architectures. Further access to hardware accelerators would allow us to test neural architecture search algorithms such as the *autokeras* open source implementation (Jin et al. (2019)). These algorithms could enable further increase in accuracy by possibly finding a more

optimal network architecture. Also, different kinds of algorithms could be tested. For example, transformer models like Image GPT (Chen et al. (2020)), have shown state-of-the-art accuracy on various classification datasets.

7 STATEMENT OF CONTRIBUTIONS

Masoud managed to obtain a decent preprocessing solution (described in section 2). His pipeline managed a robust denoising and recentering solution which allowed us to remove the artificial noise nearly completely as well as scale down the dimensions from $100x100$ to $40x40$ pixels. This reduction in dimensions allowed us to work with images having 84% fewer pixels, considerably increasing training speed and allowing us to train deeper networks.

Thomas was the first member to achieve a viable solution, placing us very well on the leaderboard with over 73% accuracy. With his great aptitude to adapt, replicate and join solutions he found to be the most promising from his research, he placed us in a very good position to branch off and try to improve on his work. His first attempt at a CNN worked very well and was only improved by adding regularization layers, that version was even used in our final ensemble. All the work that was done branched off from Thomas' initiatives, whenever we proposed to try something new he was always ready to implement it. Without Thomas' contribution our work would amount to much less than we managed.

Bruno worked on hyper-parameter tuning. Most of the work was done after our accuracy had reached 78%. We were looking to further improve and had to try various architectures. His motivation was to do a thorough hyper-parameter search and meta-model experiments, that way we could be sure we didn't miss an easy improvement.

Each member wrote a fair share of the report and contributed to the final solution.

We hereby state that all the work presented in this report is that of the authors

REFERENCES

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, February 2012. ISSN 1532-4435.

Mark Chen, Alec Radford, Rewon Child, Jeff Wu, Heewoo Jun, Prafulla Dhariwal, David Luan, and Ilya Sutskever. Generative pretraining from pixels. 2020.

François Chollet et al. Keras. `https://keras.io`, 2015.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters, 2017.

Ethan Harris, Antonia Marcu, Matthew Painter, Mahesan Niranjan, Adam Prügel-Bennett, and Jonathon Hare. Fmix: Enhancing mixed sample data augmentation, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1946–1956. ACM, 2019.

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.

Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.

David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241 – 259, 1992. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80023-1. URL `http://www.sciencedirect.com/science/article/pii/S0893608005800231`.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2018.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation, 2017.