Air Writing Recognition

Michelle Jin
mjin8@ucsc.edu
University of California, Santa Cruz

Crystal Lin
crklin@ucsc.edu
University of California, Santa Cruz

ABSTRACT

The Air Writing Recognition project is a combination of computer vision object tracking and handwriting recognition machine learning. The air writing recognition system uses the webcam of a computer to track character digits written in the air by the user, then uses a convolutional neural network to classify the character digits into one of 62 classes: 10 digits, 26 uppercase letters, 26 lowercase digits. Many current systems use complex and expensive tracking setups to achieve gesture recognition, but we seek to create a system that can achieve the same with a much more affordable setup.

CCS CONCEPTS

• **Computer vision → Object detection and tracking; Depth Analysis; Image recognition**.

KEYWORDS

handwriting classification, computer vision, machine learning, convolutional neural networks, datasets

ACM Reference Format:
Michelle Jin and Crystal Lin. 2018. Air Writing Recognition. In *Proceedings of Applied Machine Learning '19: Final Project (Applied Machine Learning '19).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Handwriting analysis is a classic, well-explored problem in the realm of introductory machine learning that encompasses many of the important topics of neural networks. When we discussed handwriting analysis during coursework, we realized that handwriting analysis is a project that could be extended with other machine learning concepts for an interesting combination application.

Our project aims to use a combination of computer vision and handwriting recognition to create a system that acts as a virtual whiteboard. Our model recognizes gestures written in the air and converts it into text. Users would be able to write "air words" facing a web-camera either real-time or in-advance and have those gestures translated into letters or digits.

Applied Machine Learning '19, March 17, 2019, Santa Cruz, CA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.1 Motivation

We settled on computer vision and handwriting analysis because we believe air writing recognition is a worthwhile topic to pursue. It allows for an entirely new text input interface that does not require much more than the computer itself.

We were not the first to pioneer the idea. Other systems such as the HTC Vive [5] virtual reality (VR) system have products that follow a similar idea. The Vive has a virtual whiteboard experience that allows users to write in the air while immersed in VR, but the system comes at a high cost that also requires an addition extensive (expensive) tracking system.

The motivation of our project is to achieve a virtual whiteboard system at a cost that is accessible to the average user. We want to introduce alternative interfaces for communication that have high affordability, usability, and accessibility.

1.2 Objective

The objective of our project is to create a system that needs only a computer and a built-in webcam to recognize different letters and digits written in the air.

2 MODELS AND ALGORITHMS

2.1 Object Tracking

In our model, object tracking is realized with detection of the target object through image analysis of each video frame. OpenCV [2], an open source computer vision and machine learning library, provides both classic and state-of-the-art algorithms relevant to object tracking. Therefore, we decided to utilize assets from the OpenCV library to detect, track, and save the trajectory of the target object as its position shifts throughout the video.

After setting the lower and upper color boundaries of our target object, we preprocess each frame by resizing and reducing Gaussian noise. Then, we construct a binary mask around the object(s) and perform morphological transformations to clean up. Finally, we detect the contours and form an enclosing circle around the object, saving the center coordinates of the circle as our tracked points.

Setting Parameter Values. In order to detect the target object, our system filters the image for items of a specific color. Our system searches for a color within a specified range to provide a margin to account for slight variations in object color. Our target object does not necessarily always exhibit the exact same color values, whether due to environmental lighting conditions or to the use of different objects that slightly differ in color values.

Frame Preprocessing. Since our system performs analysis on each frame in the video, we resize each frame while maintaining aspect ratio to increase efficiency and smoothness during run time.

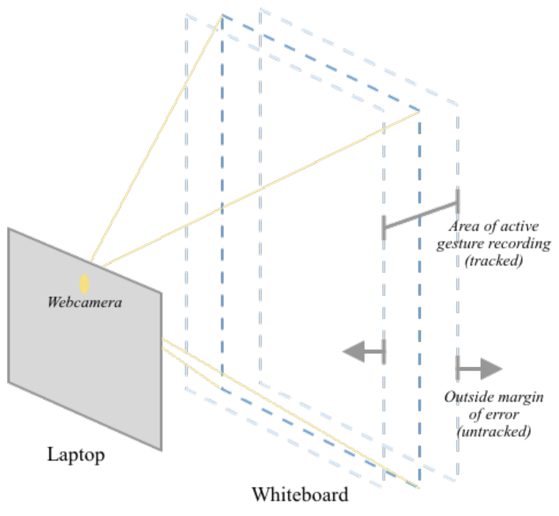
Area of active gesture recording (tracked)
Webcamera
Outside margin of error (untracked)
Laptop
Whiteboard

Figure 1: Depth Diagram

Next, we apply a Gaussian Blur to reduce the Gaussian noise, statistical noise with a probability density function equal to that of a Gaussian distribution, of the image. Finally, we convert the image from BGR to the Hue-Saturation-Value (HSV) color space to match our color boundaries that are set to HSV values. We chose to use the HSV color space, because the BGR color space values correlate with the amount of light hitting the object, whereas, image descriptions in terms of hue, saturation, and lightness are more relevant.

Mask Construction. To simplify detection and to match the training images in the EMNIST [4] dataset, which uses binary images, we create a binary mask over the objects found to be in the target color range. A series of morphological transformations, non-linear operations related to the shape of the object in the binary image, are then performed on the binary mask to clean up the image and prepare it for contour detection.

We first perform an erosion operation that shrinks the boundaries of the object and removes small white noise. Next, we perform a dilation operation to expand the boundaries of the object. Although erosion and dilation might seem like a counter-intuitive sequence, as the former shrinks the object while the latter expands it, it actually is very reasonable. While erosion shrinks the object boundaries, it also removes noise and cleans up the image. Dilation then simply re-expands the boundaries without the noise.

Contour Detection. We find the contours using an OpenCV algorithm which calculates the the hierarchy of contours in the image and compresses it. After obtaining a list of contours in the mask, we select the largest one and compute the minimum enclosing circle around that contour. If the enclosing circle has a radius larger than a predefined size and exists within the correct whiteboard margins, which will be further explained in the Depth Analysis section, then we update our list of tracked points. The tracked points

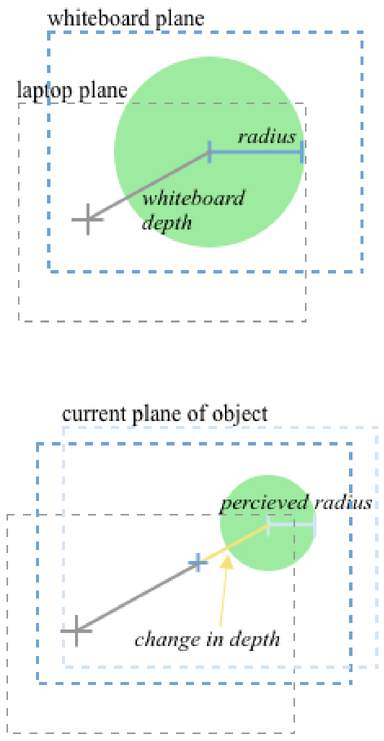

Figure 2: Triangle Similarity

are saved and overlaid onto a binary image that will be preprocessed and then passed as input into the handwriting classification model.

3 DEPTH ANALYSIS

3.1 Segmentation Problem

For our project, we were ambitious in the types of characters we wanted to detect. Rather than stick with the classic 10 digits or 26 lowercase letters, we wanted to create a system that was trained on 62 classes: 10 digits, 26 lowercase letters, and 26 uppercase letters. When working on gesture tracking, we realized that many of the characters we wanted to recognize required multiple "pen strokes" or segmented lines to write. If we were doing only lowercase letters and digits we could have been able to stick with continuous input, but many uppercase letters (such as letters A, E, and I) require segmented lines to be accurately placed for accurate differentiation– for both computer and human perception.

To solve this problem, we have come up with several solutions including flipping the object to a non-detected color, covering the object, and using predetermined keystrokes to indicate stroke beginnings and ends. Ultimately, we decided the implement the method we thought would be the most intuitive solution for the user the use– the pen lift. After observing the natural writing habits of users, we realized that whether writing on paper or on vertical whiteboards, users naturally lifted their pens at the end of each stroke and replaced it at the beginning of the next stroke. We translated

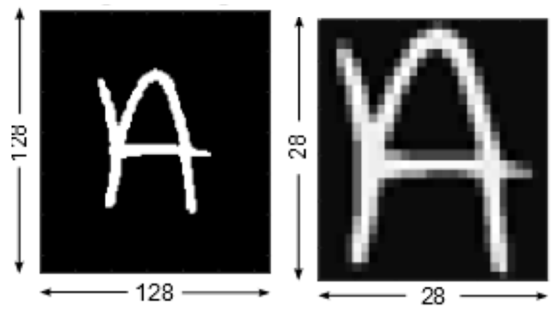

Figure 3: EMNIST Dataset

that action into a change in depth from the camera. We wanted users to be able to lift their "pen" off an invisible whiteboard to signal the end of a stroke, and a significant change in depth created by the "lifting" action would be the indicator to the system that the stroke was over (Figure 1).

Implementing depth analysis turned out to be one of the most interesting challenges of this project. In the domain of computer vision, depth analysis is being researched on many fronts, the most popular being the stereoscopic approach. This method, however, requires two different angles of the same field of vision to use optic geometry to triangulate depth. When realized, this approach requires the minimum two different cameras to be mounted at a specific height– which is not a setup the average consumer would have. Since our core goal was to increase usability and accessibility, we thought that it would be interesting to find a solution to calculating depth using only one sensing device (such as a computer web camera).

3.2 Whiteboard Approach

In considering different monocular methods, we found that using depth analysis to mark stroke segmentation did not require a terribly robust depth detection system. We were not trying to avoid pedestrians or catch a flying object– we only wanted to know if the user's "pen" was still on the invisible whiteboard plane.

Our solution was built on two assumptions: one, that the user would write on a vertical plane parallel to the computer screen and two, that the physical size of the tracked object, "the pen", would not change mid-stroke. Because of these two assumptions, we found that we were able to implement the simple law of proportions between the relative depth and relative size of the object (Figure 2).

If we are given the original radius of the tracked object and the distance of the whiteboard (writing) plane from the camera, we can use the following relation to calculate the change in depth:

$$\frac{radius_{original}}{depth_{whiteboard}} = \frac{radius_{percieved}}{depth_{whiteboard} + \Delta depth}$$

By solving for $\Delta depth$ at each frame, we are able to track the changes in depth as they occur.

4 HANDWRITING CLASSIFICATION

After obtaining the pre-processed, 28x28 sized image of the written character, our next step is to run it through a trained convolutional neural net (CNN). Our CNN architecture is rather simple, containing two convolutional layers followed by a max pooling layer, a dropout layer, and a flatten method to produce an intermediate image. The intermediate image is immediately fed into a feedforward network of one hidden layer, one dropout layer, and one softmax output layer (Figure 4).

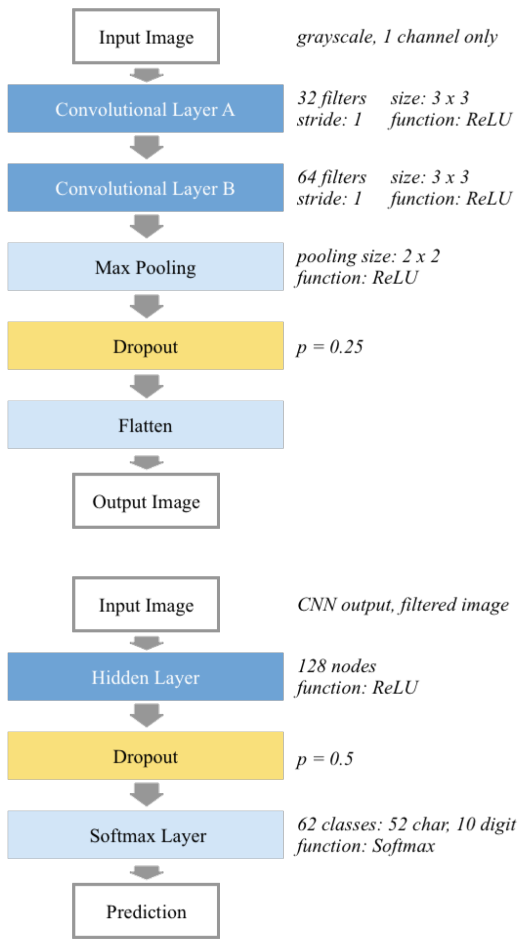

Figure 4: Network Architecture [1] [3]

4.1 Dataset

Our handwriting classification network is trained on the EMNIST (extended MNIST) Dataset of handwritten number digits [4]. This dataset is a popular choice for handwriting classification networks, and we chose to use the "by class" format– which separates between digits, uppercase, and lowercase letters– to train on. The EMNIST dataset of over 814,225 samples is already split 90% training and 10% testing, but we decided to take the training set of data to be both our training and our evaluation data. We split this dataset 8:2, training to evaluation, using the Scikit-Learn train_test_split method [6]. This second separation allowed us to save the original 10% test data for unseen testing.

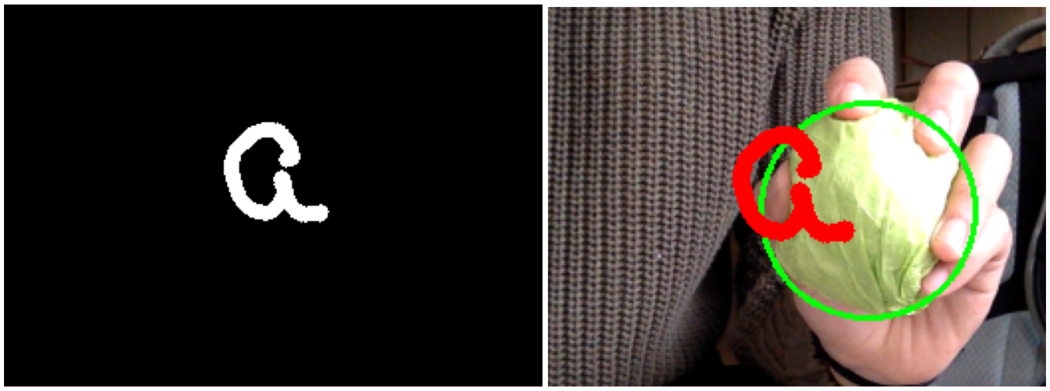
Figure 5: Input Generation

4.2 Preprocessing

When utilizing the EMNIST dataset we needed to reshape the data (EMNIST is in the channels-last format) and cast the booleans values as float32 values. EMNIST has pixel values from $[0, 255]$, and we also remapped those to values from $[0, 1]$. Other than data restructuring, we did not perform any drastic preprocessing as EMNIST is a very clean dataset to begin with.

4.3 Hyperparameters

When choosing our hyperparameters, we started off with just one convolutional layer with 32 filters and a hidden layer of 128 nodes. We figured that because we had a rather large number of output classes it would beneficial for us to increase the number of convolutional layers and filters to check for more differentiating features between the classes (Figure 4). We chose the rectified linear unit (ReLU) function as an activation function because it is a piece-wise function and our images lend well to sharp boundary distinctions. We decided to implement dropout layers at the end, simply to see if it would increase our accuracy.

5 RESULTS AND ANALYSIS

5.1 Object Tracking

We set a lower boundary of $(29, 86, 6)$ and an upper boundary of $(64, 255, 255)$ in the HSV color space to filter for a range that can be described as lime green. Each frame is resized to a width of 600 pixels while maintaining aspect ratio. A Gaussian Blur utilizes a Gaussian kernel and is applied with kernel size of $11x11$ to convolve the image. Strides are calculated automatically given kernel size using tools from OpenCV.

For the morphological transformations, we first perform erosion, where a kernel slides through the image, setting the pool to 1 only if all the pixels in the kernel are 1 and 0 otherwise. Following that, we perform dilation, where a kernel slides through the image setting the pool to 1 if at least one of the pixels are 1 and 0 otherwise.

Next, we utilize an OpenCV algorithm that finds and returns all contours of the image. We take the largest contour and form an enclosing circle around it. If the radius is greater than 10 pixels and the circle is in the correct whiteboard margin plane, then we save the coordinates of the center of the circle into a list of tracked points. We then plot and join the points into a binary image. We perform similar operations to detect the contours of the binary image and form an enclosing square around our target object. The square which captures our target object gets resized to a $28x28$ pixel image that is fed into our handwriting classification model.

5.2 Depth Analysis

Our implementation of depth analysis sets the original radius and distance of the whiteboard plane when the user determines that they are ready to begin writing and presses the "s" key. We also implemented visual depth feedback where the color of the circle would change to notify the user that they have exited the depth range for tracked movement. The circle starts green, and after the start key is pressed, the circle will become red if the tracked object is too close to the camera or will become yellow if too far. We added an error of margin so users would be allowed some depth change while writing.

After testing the project, we found that the idea was sound, though the implementation had a few hiccups. While experienced air writers could fairly easily write a character at the correct depth, new users had a difficult time keeping a static distance from the camera and would often end up with character images that had jumpy, overly-segmented strokes. The depth calculations are also entirely based on the whiteboard plane– which is perpendicular to the computer screen– so the angle at which the screen was placed heavily impacted the clarity of the output images.

We also discovered that our program sometimes experiences some lag between output frames. This negatively impacts the clarity of the output image because users over are not able to adjust accurately without good feedback from the system. We also hypothesize that the depth calculations at each frame may be the reason. Although we have not tested if this is entirely the case, we have thought about moving the depth calculation to the buffered points rather than at each frame.

5.3 Handwriting Classification

When we initially trained our neural network on 731,668 samples of EMNIST data (split 8:2 between testing and evaluation datasets) we achieved a training accuracy percentage of 84.71% and an evaluation accuracy percentage of 86.43% (Figure 6). We were also at a training

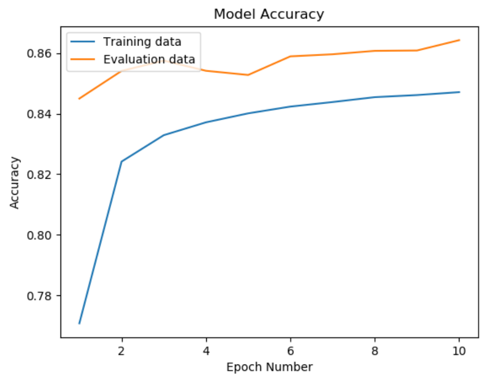

Figure 6: Model Accuracy

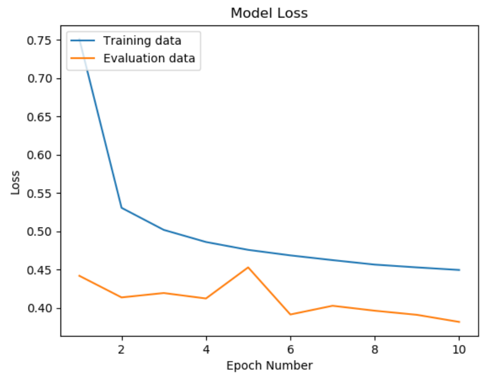

Figure 7: Model Loss

loss percentage of 44.93% and an evaluation loss percentage of 38.14% (Figure 7). Looking at Figures 6 and 7, we were surprised to find that the evaluation data actually performed better than that of the training data, and when tested on the original 10% test data, we were able to achieve 86.33% test accuracy and 38.14% test loss. This could indicate that we should continue training for more epochs to see if we can continue to increase the accuracies. Unfortunately, training 10 epochs of the CNN takes around 5 hours, so we were not able to tune our hyperparameters to the precision we would have liked.

When running the program from beginning to end, we find that actual prediction accuracy is significantly lower than that of just the classification model. This may be because the images created from object tracking have different qualities than those from EMNIST, for the images generated often have more noise and have user errors not often seen when writing on paper. Although we have discussed increasing clarity and reducing errors of generated images earlier, introducing training samples generated from the object tracking part may tailor the model to better recognize and classify those errors.

6 CONCLUSION

This paper introduced our Air Writing Recognition project that was completed over the course of a quarter. Our work took a classic machine learning introduction project, handwriting analysis, and extended it to new interfaces so that it may be used in different ways. We challenged ourselves by implementing a novel solution to the segmentation problem because we believed that our solution provides a more intuitive interface for end users. Although the accuracy is not 100% (few things are), we hope our small contribution draws attention toward the nuances of different implementations of the same machine learning, computer vision problems and encourages other researches to think more deeply about the way users use the technologies they create.

7 FUTURE WORK

If we were to continue forward with this project, we would be interested in increasing both the usability and the accuracy of our model. One, we would continue working to tune the depth analysis implementation. Currently, the program is using basic depth checking to determine whether the gesture would be tracked or untracked, which often leads to rough edges and small skips in a single stroke. Although we already use blurring techniques to reduce noise, we would like to implement a dynamic, weighted line that increases or decreases based on the depth of the tracked object. This may help with both usability and accuracy as users can more easily recognize the depth of which they are writing, and the output images will not have the harsh skips that happen when the object exits the tracked range.

We would also like to conduct more testing with different variations of our CNN. As mentioned in the Results section above, we ended up with a higher test accuracy than training accuracy on our neural network (it happens fairly consistently as well). Although it could be largely attributed to the dropout layers of the network affecting the training accuracies, we also think that this may signal that we can still improve our test accuracy if the model was trained better. In the future, we may implement more hidden layers and convolutional layers to try to capture more of the unique features of the 62 classes. We are also interested in using not only the EMNIST dataset to train on but also the hand-labeled images that were created by the object tracking system. This may allow the accuracies of the CNN to be better reflected when used with the object tracking system.

Beyond bugs, we would want to extend this project to recognize cursive and air-written words. If the accuracy of the current program were to increase, it would be interesting to continue extending the air-writing-recognition to more complex gestures.

8 CONTRIBUTIONS

Implementing air writing recognition requires not only the understanding of the EMNIST dataset and the creation of a model for handwriting classification that uses both a convolutional and feedforward neural network, but also requires the time to learn the skills to utilize OpenCV for object detection, tracking, and generating input that matche the training dataset for our classification neural network.

Our project was completed through 3 phases over the past 10 weeks.

Phase 1. During the first few weeks, we both spent many hours researching different project ideas. After narrowing down our options, we both then spent time brainstorming how we would go about implementing each option and whether or not it would be within the scope of 10 weeks. After selecting a project idea, we spent several hours a week learning about the EMNIST dataset, the different neural networks, and experimenting with OpenCV.

Phase 2. During the intermediate weeks, Michelle researched further into the implementation of an object tracking system while Crystal researched further into the implementation of the classification system. Both parties had an equally demanding workload, and updated one another on their progress regularly.

Phase 3. During the final weeks, Michelle and Crystal spent time combining the systems they built separately into one system. Both parties had to actively work together in order to clarify any ambiguities and inconsistencies between each system. Once the system was put together, both parties worked on improving the smoothness of the program to decrease lag time and accuracy of the predictions.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[3] François Chollet et al. 2015. Keras. https://keras.io.

[4] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. 2017. EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373* (2017).

[5] HTC and Valve Corporation. 2016. HTC Vive. https://www.vive.com/eu/.

[6] 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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.