DEEP LEARNING FOR SOURCE DIGITAL CAMERA IDENTIFICATION

Bernat G. Škrabec
Student, M.Sc. DSE
Eurecom, Sophia Antipolis
Bernat.Gene-Skrabec@eurecom.fr

Sourish Ghosh
Student M.Sc DSE
Eurecom, Sophia Antipolis
Sourish.Ghosh@eurecom.fr

February 15, 2021

Supervisors: Prof. Jean-Luc Dugelay, Dr. Chiara Galdi.

Students: Bernat G. Škrabec, Sourish Ghosh.

1 Introduction

1.1 Source Digital Camera Identification

The problem of Source Digital Camera Identification (or Recognition) can be formulated with the following question: *Given a picture, which camera captured it?*

One of the main approaches to solve this problem is based on the existence of a "Fixed-Pattern Noise" (FPN). This noise is like a fingerprint that is unique, not only among different camera models, but even between different cameras of the same model. The deterministic component of the Sensor Pattern Noise (SPN) is mainly caused by imperfections during the sensor manufacturing process and different sensitivity of pixels to light due to the in-homogeneity of silicon wafers. It is because of the inconsistency and the uniqueness of manufacturing imperfections and the variable sensitivity of each pixel to light that even sensors made from the same silicon wafer would possess uncorrelated pattern noise, which can be extracted from the images produced by the devices. This property makes SPN a robust fingerprint for identifying and linking source devices and verifying the integrity of images.

1.2 Addressed problem

The main objective of this project was to familiarize ourselves with the domain of digital source camera identification and explore modern Deep Learning (DL) solutions to solve it. We started by reproducing known methods to understand the underlying principles. Then, we focused on exploring new DL methods and compared them with the baseline established a by widely used and known non-DL method described below. More specifically, we wanted to explore modern Deep Learning (DL) techniques and compare them with the Reference Sensor Pattern Noise (RSPN) described by Lukas et al. [1], by measuring their performance on the SOCRatES dataset [2].

1.2.1 Baseline performances

Baseline performances were computed on the SOCRatES database in the article "SOCRatES: a database of realistic data for SOurce Camera REcognitionon Smartphones" by Galdi *et al.*[2] Performances are assessed in terms of Equal Error Rate (EER), Receiver Operating Characteristic curve (ROC) and Area Under the ROC curve (AUC), and summarised in Table 1.

The performances of the method proposed by Lukas et al. in [3], are summarized in this section. The Reference Sensor Pattern Noise (RSPN) is extracted from the “background pictures” for each device using the code made publicly available by the authors. Then the SPN is computed for each ”foreground picture” and associated to the most correlated RSPN.

Li’s approach proposes an enhancing process to mitigate the impact of scene details in the computation of the SPN. The Enhanced SPN (ESPN in the following) n_e is obtained as follows:

$$n_e(i,j) = \begin{cases} e^{-0.5n^2(i,j)/\alpha^2}, & \text{if } 0 <= n(i,j) \\ -e^{-0.5n^2(i,j)/\alpha^2}, & \text{otherwise} \end{cases}$$

where n_e is the ESPN, n is the SPN, i and j are the indices of the components of n and n_e, and α is a parameter that is set to 7, as indicated in [4].

As in the first experiment, the RSPN is extracted from the “background pictures”. Then the ESPN is computed for each “foreground picture” and associated to the most correlated RSPN, i.e. each ”foreground picture” is associated to the most correlated camera.

The ROC curves obtained by the two tested methods are compared in Fig. 1.

Table 1: Performances of Lukas et al. and Li on SOCRatES.

	EER	AUC	RR
Lukas et al.	0.0894	0.9106	0.9191
Li	0.0921	0.9079	0.9164

1.2.2 SOCRatES

SOCRatES: SOurce Camera REcognition on Smartphones[2], is an image and video database especially designed for source digital camera recognition on mobile devices. It answers to two specific needs, the need of wider pools of data for developing and benchmarking of image forensic techniques, and the need to move the application of those techniques on smartphones, since, nowadays, they are the most employed devices for image capturing and video recording.

SOCRatES is currently made up of about 9.700 images and 1000 videos captured with 103 different smartphones of 15 different makes and about 60 different models. The acquisition has been performed in uncontrolled conditions. In order to collect the database, many people were involved and asked to use their personal smartphone to collect a set of pictures. Instructions were given to the participants and they collected the set of pictures in complete autonomy. The reason behind this choice is, on the one hand, to collect a database of heterogeneous pictures and to maximize the number of devices employed, and, on the other hand, to carefully replicate the real scenario of application of the techniques that will use this database as benchmark. In fact, by selecting a “population” of smartphone users and letting them capturing the pictures, we automatically select a set of smartphones representing the current market. In this first acquisition session, a population of Master students from EURECOM has been involved.

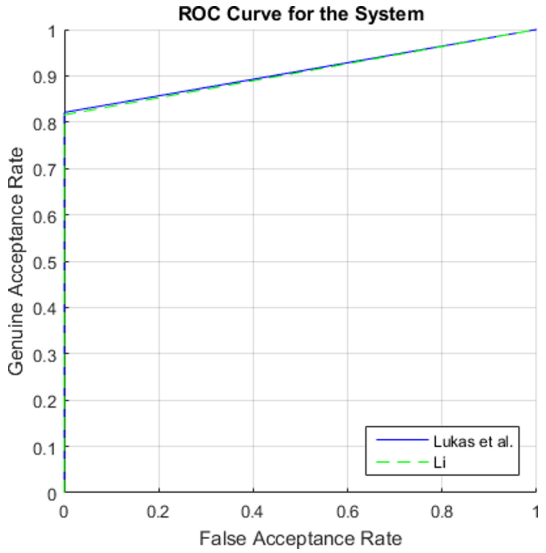

Figure 1: ROC curve illustrating the baselines assessment on SOCRatES.

2 StudProj Škrabec: Siamese Networks for Source Digital Camera Identification

2.1 Proposed solution

2.1.1 Motivation

When thinking about applying known solutions to a real life scenario we may be faced by two problems: the eventual limited availability of test images and the "open set" problem. These can be introduced by formulating one simple question; What if we only have 4 images and those images are from previously unseen devices? Neither the original RSPN method nor a conventional Deep Learning Classification have an answer, since the former recommends at least 50 background images [3] and the latter cannot deal with classes undefined prior to training.

Our proposed solution tries to combine the best of both worlds to solve the stated problems. First, we use the denosing techniques proposed by the original method [3] which help remove the influence of the objects on the image. Second, we leverage the power of deep learning to find patterns that go beyond what a simple correlation measure could achieve with a limited dataset. Third, instead of doing a classification, we rephrase the question to: *"Given two sets of images, are both sets taken by the same device?"*; agnostic to both the sets sizes and the previous knowledge of the device's existence. To answer this question, we propose a one-shot learning solution based on Siamese Networks.

2.1.2 Related work

We first attempted to reproduce the approaches described by Tauma et al. [5] and Freire-Obregon et al. [6]. However, since they are both based on classification, they do not work, by definition, with unseen camera models. Moreover, given the limited number of examples per class in our dataset of interest (~ 100) [2], compared to theirs ($\sim 20,000$), it is almost guaranteed that the approach won't perform as good. The open set problem for camera model identification using deep learning is extensively described by Bayar et al. [7], together with proposed solutions.

Siamese Networks, originally described by Chopra et al. [1], attempt to learn a similarity function, rather than a class, from pairs of positive and negative examples. The network receives two inputs which are put through a twin CNN with the same weights, thus the name. The distance between the two outputs serves as a decision on the similarity of the inputs.

They are used, for example, in face recognition [8], where it is unfeasible to re-train a model every time we want to be able to recognise a new person, taking thousands of images of the newcomer. Instead, the model is trained to tell if two images are from the same person or not, even if it has never before seen said person. This is exactly the functionality we may wish for a useful camera identification tool. We attempted to reproduce first the network described by Sameer et al. [9], using ideas from Mayer et al. [10] as well, which both address this method.

However, the previous failed to perform with our dataset so we decided to write our own version which included a pre-processing step as done in other DL methods, e.g. [5], but more closely resembling the original denoising filter and residual averaging on the Lukas paper [3], specifically the one described on Appendix A. We based some characteristics of our architecture on Koch et al. [11], which uses Siamese Networks for a different domain, but with extensive changes product of our experimentation.

2.2 Implementation

2.2.1 Data preparation

Different devices have different image sizes but our Neural Network needs a fixed input size. Additionally, the desired input size is around 128 to 256 pixels, otherwise the network size becomes unmanageable. To address this, we wrote a script to cut and transform the Socrates dataset into a *strips dataset* (patches). Each original image is cut into patches of 256x256px, starting from the top-left and into as many as can fit. Because features of SPN must depend on the region of the image, we build a meta-data file that allows us to navigate the dataset. It maps the set of student id's to their respective sets of images, and then each unique image id to its respective set of strips, preserving the strip location. Thus, for a given strip, we know the device that took it, the image were it comes from and the pixel coordinates were it fits.

To reduce the influence of the physical objects present in the images, we pre-process each strip by applying a denoising filter on the wavelet domain following basically the same algorithm as described in Appendix A of Lukas et al. [3], but we re-implemented using the Scipy library [12] (particularly the wiener filter implementation) and PyWavelets [13].

2.2.2 Network Architecture

Our implementation is developed using the Keras API [14], using Tensorflow 2.4 [15].

The training pipeline works as follows. First, a batch is generated from the training split of the dataset. Each batch consists of two equal sized sets of positive and negative examples. The positive and negative examples are defined as:

- **Positive** : $\{A = \{s_a \equiv strip_{ikxy}\}, B = \{s_b \equiv strip_{jlwz}\}\}$ where $i, j \in \{Students\}$, $k \in \{Images_i\}$, $l \in \{Images_j\}$, $xy, wz \in \{Coordinates\}$ and $(\forall s_{a1}, s_{a2} \in A$ and $\forall s_{b1}, s_{b2} \in B)[(i1 = i2 = j1 = j2)$ and $\{k1, k2, l1, l2\}_{\neq}$ and $(xy1 = xy2 = wz1 = wz2)]$
- **Negative** : $\{A = \{s_a \equiv strip_{ikxy}\}, B = \{s_b \equiv strip_{jlwz}\}\}$ where $i, j \in \{Students\}$, $k \in \{Images_i\}$, $l \in \{Images_j\}$, $xy, wz \in \{Coordinates\}$ and $(\forall s_{a1}, s_{a2} \in A$ and $\forall s_{b1}, s_{b2} \in B)[(i1 = i2)$ and $(j1 = j2)$ and $(i1 \neq j1)$ and $\{k1, k2, l1, l2\}_{\neq}$ and $(xy1 = xy2)$ and $(wz1 = wz2)]$

That is, a positive example is a pair of sets of strips, where each strip comes from a different image, but all are taken by the same device and belong to the same coordinates. In contrast a negative example consists of a pair of sets of strips where each strip comes from a different image, but the strips on each set belong to different devices.

The strips on each side of the pair are averaged, and a random crop of 128x128 px is selected (same crop for both sides), for regularization purposes. Those are fed to the two inputs of the Siamese network. Then, each input goes through a sequential Convolutional Feature Extractor (CFE). The CFE outputs a feature vector of size 1024 for each input. After that, we take the absolute difference between both vectors and feed it into a Dense layer with a Sigmoid activation function that gives the result, a single real number. Values close to 0 indicate that the images on each side of the pair are taken by the same device, while values close to 1 indicate that they are not. The model is then trained by using binary cross-entropy loss between the true labels (positive = 1 or negative = 0) and the predicted values. Figure 2 summarizes the pipeline.

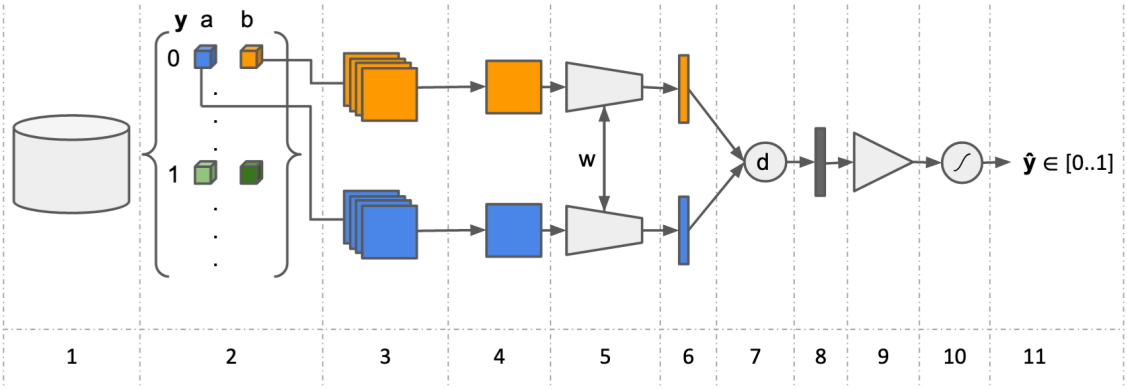

Figure 2: Training diagram. **1** Dataset; **2** Random batch of pairs, y is the label; **3** Individual patches from one pair, `shape=(2,4,256,256,3)`; **4** 'Flattened' averaged image and random crop, `shape=(2,128,128,3)`; **5** Convolutional Feature Extractor, same weights above and below; **6** Feature vectors of each image, `shape=(2,1024)`; **7** Absolute difference; **8** Difference vector; **9** Dense layer $(1024 \to 1)$; **10** Sigmoid activation; **11** Final output, Binary Cross Entropy on $(y, \hat{y})$

The CFE is sequential and consists of 5 consecutive blocks of `[Convolution2D(kernel size=3, activation=relu), MaxPooling 2D(pool size=2, stride=2), BatchNormalization]`, with the number of convolution filters increasing at each block. Then a `Flatten` layer connected to a `Dense` layer with output shape `1024`.

2.3 Experimental results

The model was trained on a subset of the SOCRatEs dataset containing 55 different devices with an average of 95 images per device and an average of 120 strips/patches per image. To guide the hyper-parameter tuning and explore different architectures, we used a separate validation set containing 7 different devices, with about the same distribution of images per device and patches per image.

Finally, once the model was completely built and finished it was tested on a new set of 18 different devices, all unseen until that point. About 20% of the original dataset has not been used due to storage and computational restrictions.

Specifically, the testing set contained 1875 images from 18 devices, with 227,947 different patches. Since the number of total possible pairs is astronomical ($\sim 10^{11}$), testing all of them is clearly unfeasible. Instead, the test was performed by sampling 1000 different pairs of 4 images at random, half of the pairs being positive examples, and the other half, negative.

To boost the final score, we perform an "extended inference". This means that, instead of using a single 128x128px patch from a 14 MP image to make a decision, for each pair, we compare several patches (40 in our experiment) from the same images, and take the average score. This gives a speed-up in accuracy of 1.18x, but is totally reasonable to do, since in a real life scenario we expect to get whole pictures.

The final results are summarised in Table 2: we present the Equal Error Rate (EER), the Area under the ROC (AUC) score and the number of images required to perform one inference (Im. Req. Inf.). This last figure is interpreted as follows: How many pictures of a single device do we need in able to confidently say that they are taken by the same camera? In the case of Lukas et al. method, for

example, we need at least 50 background images, plus the one we wish to predict. In our method, on the other hand, we need 4 images at each side of the pair.

Table 2: Performances of Lukas et al., Li and Škrabec on SOCRatES.

	EER	AUC	Im. Req. Inf.
Lukas et al.	0.0894	0.9106	50+1
Li	0.0921	0.9079	50+1
Škrabec	0.136	0.9447	4+4

3 StudProj Ghosh: Source Camera identification Using Convolutional Neural Network

3.1 Motivation

The main goal of this project was to detect the source camera for mobile devices using a deep learning approach. With the evolution of information technologies and the development of mobile devices have paved the way to an unlimited number of online mobile applications. Further with the availability of these relatively cheap, mobile and convenient digital devices has made it possible to capture image and audio data without time or network related restraints.
The images captured by a camera have a huge array of applications. For example, in digital forensics it is important to identify the source of an image in order to link it with a specific digital camera or link images with a common source. It is also important to prove the authenticity of the images, as they might be computer generated.
Each camera has its own photo-response non-uniformity (PRNU) noise patterns which are unique to it. Convolutional Neural Networks (CNNs) have shown excellent performance on several tasks such as image recognition, video analysis or natural language processing. Thus, these can be exploited to detect the features of the images to detect the source camera.

3.2 Proposed solution

As mentioned in [6], the project aims to use a CNN architecture which is able to infer the noise pattern of mobile camera sensors (fingerprint) with the aim of detecting and identifying the mobile device used to capture an image. The proposed solution in [6] uses the MICHE-1 dataset containing images taken from 3 devices and 5 cameras in total. However in the project we have implemented it on Socrates[2] dataset containing a wide array of mobile devices.

3.3 Data Preparation

The Socrates [2] data-set currently contains data from 103 devices containing both background as well foreground images. But we have used 15 classes (devices) containing equal number of background and foreground images in each. The images are further arranged in hierarchical directory structure so that the accessibility of the images in the input pipeline can be made easily using the Keras [14]. Also the images taken by various cameras were of various sizes and shapes. But our network needed a fixed input shape, thus accordingly we fixed the shape of the images to an intermediate size of 640x640 px. Further, since we are using the [6] paper as reference, we split each of the images into non-overlapping patches of 32x32 from each of the images, which were then fed to the network for training.

3.4 Model Architecture

As proposed in [6], we will be working with a Convolutional Neural Network in this project. A CNN is similar to an ordinary neural network, which is made up of neurons that have parameters (weights and biases) that can be modified during the training process by means of forward and backward propagation [16]. CNNs only considers images as input contrary to the ordinary neural networks. The schematic diagram of the network can be seen in 3. In the project we have used 3 types of layers: 1) convolutional, 2) pooling, and 3) Dense (fully connected).
The parameters of a convolutional layer consist of a set of trainable kernels. For the 1st layer, the kernel size is fixed to 8x8 pixels with relu as the activation function. For the 2nd layer, the kernel

size is fixed to 3x3 pixels with activation function. According to [17], the formal computation of the convolutional layer can be described as:

$$S_t(i,j) = (K * I)(i,j) = \sum_{m=0}^{2} \sum_{n=0}^{2} I(i-m, j-n)K(m,n)$$

Next comes the pooling layer. Usually pooling layers are inserted in-between successive convolutional layers. But in this case the maximum number in every sub-region that the filter convolves around is extracted from the input volume. It basically reduces the spatial dimension of the image. Here The max pool is included between the convolutional and dense layer with a kernel size of 2x2. Finally, the three fully connected dense layers are considered at the end of the process, where the last layer outputs an N dimensional vector where N is the number of classes for which the architecture is trained for. The fully-connected layers 1 and 2 have 256 and 512 neurons respectively with Relu as activation function. The final output layer uses Softmax activation function.

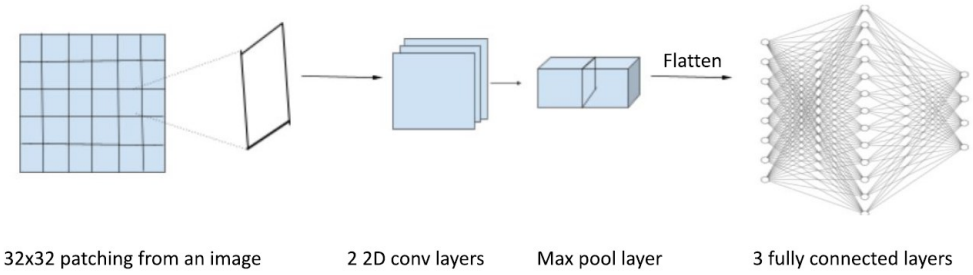

Figure 3: **Model architecture**

3.5 Experimental results

Due to memory management issues, the output class was stripped down to 15 classes. The model was initially ran for 10 epochs. It gave an initial validation accuracy of 39% without any patching done on the images. After patching the images, the validation accuracy increased to close to 50%. However, if we still go on the compare the result with the reference paper [6], there is still a large gap in the accuracy. The inference can be drawn from the fact that in the reference paper they have trained the model on photographs taken from 5 cameras only and on all similar types of images. But in this case, the model was tested on [2] data set, which consisted on diverse set of images on 15 different cameras. As a result of which the accuracy is low. Further changes were made to the baseline proposed model. To help the model more robust by increasing the testing accuracy a dropout layer was included in between the first hidden layer i.e between the two convolutional layers with a dropout rate of 20%. Another dropout of 10% was used in the visible layer which means 10% of inputs will be randomly excluded from each update cycle. This improved the model performance. Further now on an unseen test data it gave an accuracy of 50.1% which an increase of $\pm 7\,\%$ from the earlier one.

4 Discussion and further work

4.1 Škrabec

Our approach has obtained good results that are comparable with the baseline, while requiring fewer images for the prediction. Thus, we are quite satisfied with our final outcome. However, due to time

and resource constraints, there are several paths that have been left unexplored. The most interesting follow up experiments that could be done are:

- Use the complete SOCRatES dataset (leaving only the necessary subset for validation and testing).
- Randomize the number of examples per pair during training, as to make the required number for inference completely arbitrary.
- Explore other architectures for the Convolutional Feature Extractor.
- Perform a more in-depth Hyper-Parameter tuning.
- Evaluate other denoising filters for pre-processing. Maybe even attempt fore-going it altogether.

All the code used for developing this project is delivered together with this report, containing as well the final weights checkpoint of the model.

4.2 Sourish

On completion of the project if we look back to it, we can conclude there can always be room for improvement. Since, here we replicated the Friere-Obregon paper, there has been a difference in the accuracy of the models. One basic way is to use a deeper CNN with more number of layers like VGG 16 so that deeper and minute features can be extracted. But on experimentation, it can be seen that the model suffers from vanishing gradient problem and it stopped learning. Another suggested method can be to use conventional processing techniques before feeding into the network. Also, due to memory management, we trained on 15 classes only. These can add up to the provided results.

5 Conclusion

In this project we have provided two solutions to tackle the Source Camera Identification problem. In case of the first one, we have used a Siamese Network to decide whether two mini-sets of images were taken from same camera or not. The model is trained to learn a similarity score by training on pairs of positive and negative examples. The final model is comparable in performance to the baseline, while requiring less pictures to make a decision.

The other solution involves a simple CNN Network. Here the the patched images were fed to convolutional layers of the network. The output of the network is categorical cross-entropy loss on the output of the last dense layer which predicts with which camera out of 15 a certain photo was taken from.

In conclusion, we have achieved the main objective of testing modern Deep Learning (DL) techniques against a baseline set by a well established approach. We have seen the advantages that a DL approach may offer, but also its shortcomings. In summary, we are satisfied with the results presented and with the insight gained during the development of this project.

References

References

[1] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546 vol. 1, 2005.

[2] Chiara Galdi, Frank Hartung, and Jean-Luc Dugelay. Socrates: A database of realistic data for source camera recognition on smartphones. In *In Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods*, volume 1, pages 648–655, Volume 1, 2019.

[3] J. Lukas, J. Fridrich, and M. Goljan. Digital camera identification from sensor pattern noise. *IEEE Transactions on Information Forensics and Security*, 1(2):205–214, 2006.

[4] Chang-Tsun Li. Source camera identification using enhanced sensor pattern noise. *IEEE Transactions on Information Forensics and Security*, 5(2):280–287, 2010.

[5] A. Tuama, F. Comby, and M. Chaumont. Camera model identification with the use of deep convolutional neural networks. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, Dec 2016.

[6] David Freire-Obregón, Fabio Narducci, Silvio Barra, and Modesto Castrillón-Santana. Deep learning for source camera identification on mobile devices. *Pattern Recognition Letters*, 126:86 – 91, 2019.

[7] B. Bayar and M. C. Stamm. Towards open set camera model identification using a deep learning framework. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2007–2011, April 2018.

[8] Haoran Wu, Zhiyong Xu, Jianlin Zhang, Wei Yan, and Xiao Ma. Face recognition based on convolution siamese networks. pages 1–5, 10 2017.

[9] Venkata Sameer and Ruchira Naskar. Deep siamese network for limited labels classification in source camera identification. *Multimedia Tools and Applications*, 79, 10 2020.

[10] Owen Mayer and Matthew C. Stamm. Learned Forensic Source Similarity for Unknown Camera Models. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2012–2016, Calgary, AB, April 2018.

[11] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. *ICML Deep Learning Workshop*, 2, 2015.

[12] Pauli Virtanen et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[13] Gregory R. Lee, Ralf Gommers, Filip Waselewski, Kai Wohlfahrt, and Aaron Oamp;8217;Leary. Pywavelets: A python package for wavelet analysis. *Journal of Open Source Software*, 4(36):1237, 2019.

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

[15] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. `https://www.tensorflow.org/`, 2015.

[16] D. Rumelhart, Geoffrey E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.