Implementation of a Coin Recognition System for Mobile Devices with Deep Learning

Nicola Capece, Ugo Erra, Antonio Vito Ciliberto
Dipartimento di Matematica, Informatica ed Economia
Università della Basilicata,
Potenza, Italy
nicola.capece@unibas.it, ugo.erra@unibas.it, antonio.ciliberto@unibas.it

***Abstract*—This paper examines the application of a deep learning approach to automatic coin recognition, via a mobile device and client-server architecture. We show that a convolutional neural network is effective for coin identification. During the training phase, we determine the optimum size of the training dataset necessary to achieve high classification accuracy with low variance. In addition, we propose a client-server architecture that enables a user to identify coins by photographing it with a smartphone. The image provided by the user is matched with the neural network on a remote server. A high correlation suggests that the image is a match. The application is a first step towards the automatic identification of coins and may help coin experts in their study of coins and reduce the associated expense of numismatic applications.**

***Keywords*-deep learning; coin recognition; numismatic;**

I. Introduction

Numismatics is the scientific study of all forms of currency. Various criteria are used to classify a coin, including its history, geography, and market value. In recent decades, image recognition techniques have been investigated for the identification and classification of coins, as currently these procedures still rely on expensive human intervention. In [1], the authors survey numismatic research into the classification, identification, and segmentation of coins based on image recognition. Numismatics research can be divided into different historical periods: ancient, medieval, modern, and contemporary. Here, we consider contemporary numismatics, which focuses on coins from the 17th century onward. The advantage of studying this category is that any engravings can still be clearly seen by the human eye, and there is a plentiful supply of coins, which is useful in machine learning approaches.

Recent studies of coin classification are based on image recognition techniques. Several families of algorithms have been developed, based on neural networks (NN) [2], decision trees [3], edge detection [4], gradient directions [5], and contour and texture features [6] [7]. Although many algorithms are NN variants, Deep Learning (DL) is often used for image recognition. DL is a branch of artificial intelligence (AI) where the aim is to develop computational models that are composed of multiple processing layers and can learn a representation of the data at multiple levels of abstraction [8]. DL algorithms need complex structures and huge datasets, while back-propagation techniques enable internal parameters to be changed (training) to improve prediction accuracy. These internal parameters are called weights, and are used to represent the data from one level to the next. Image recognition and DL form the boundary of convolutional neural networks (CNN). A CNN is a special type of NN whose functioning is inspired by the organization of the visual cortex of the human brain. In general, each NN layer is composed of several neurons that are connected with upper and lower layers via arches that are called synapses.

There are two categories of learning: unsupervised and supervised. In unsupervised learning, the NN takes the inputs that are provided and reclassifies and organizes them according to a set of shared features, in order to reason and make predictions about any subsequent input. In supervised learning, the NN is provided with a series of inputs for which the output is known; the output is also provided to the NN so that it can resolve the task autonomously. The aim of supervised learning is to develop a function that can replicate results obtained in the training phase using similar, but new, structured input.

In this paper, we present an implementation of a system for automatic coin recognition based on DL. The representation of coins is learned in a training phase that is based on a supervised learning approach. We show that a well-known CNN can be effective for coin identification. In particular, we determine the optimum size of the training dataset that is necessary to achieve high classification accuracy with low variance. Based on a set of 8320 images of euro coins, we trained the CNN using different-sized training samples and tested the resulting system. Using this data, we employ a learning curve approach to predict classification accuracy for a given training sample size. Furthermore, we propose a client-server architecture that makes it possible to query the classification model obtained from the NN training set, and allows a user to identify a coin by photographing it with, for instance, a mobile device camera. A coin is identified when the image provided by the user can be matched with the NN on a remote server.

The remainder of this paper is structured as follows. Section II, provides an overview of related works. Section III provides some background information about the NN and software tools used to perform training. Section IV is a detailed presentation of the framework. Section V describes the training phase. We end with some final remarks and future directions for our research in Section VI.

II. Related Works

Among the different coin recognition methods, Bremananth [9] proposes a system that focuses on numerals rather than images on the front or back. The idea is to capture an image of the coin and extract the numeral using a technique based on pattern matching. The approach uses several techniques, such as statistical color, threshold method, Gabor filtering, and back propagation networks for accurate recognition of these numbers.

Kim [10] proposes an automatic recognition method for Imperial Roman coins using CNN. The study implements a hierarchical framework that employs CNN models for coin classification tasks. The aim is to find a landmark on coin images. An optimization problem is formulated as the selection of a set of the coin's elements that represents the class. The selected parts are considered to be elements that can be used to discriminate the coin. These landmarks can be critical for the analysis of the features of coins for numismatic experts.

Modi [11] developed an artificial NN based on the automatic recognition of Indian coins. The system is able to recognize both sides of coins. Information is extracted from images using methods such as the Hough Transformation, Pattern Averaging, etc. The extracted information is used as input to train a NN.

The main difference between these methods and the work we present here is that we consider the entire coin, and classify it according to membership classes. The information is extracted directly from the coin and is passed to the NN in the training phase. In this way, we obtain a classification model that can predict the classification of other coins.

III. Background

Any discussion of Deep Learning (DL), also indirectly refers to Neural Networks (NN). A standard NN consists of connected, artificial neurons that are modelled on biological neurons. These simple nodes (neurons) are often called processing elements or units. Each neuron produces a sequence of real-value activations [13]. Within the NN, artificial neurons (AN) are organized into layers. ANs in each layer are connected to upper and lower layers through weighed arcs called synapses. There are three types of layer: input, hidden, and output.

A layer is a container that receives a weighted input, transforms it with a set of (mostly non-linear) functions, and passes these values as input to the next layer [14]. The first and last layers in a network are the input and output layers, respectively, while all layers in between are hidden layers. Input neurons are activated by data provided by the external system. In contrast, output and hidden neurons are activated by data provided from already activated neurons [13]. Each neuron is activated by an activation function, which receives weighted data (matrix multiplication between input data and weights) and outputs a non-linear transformation of it. In the context of DL, neural networks are often called Deep Neural Networks (DNN), as they are distinguished from the more common neural networks by their depth. In DNN, each layer of neurons is trained on a distinct set of features based on the previous layer's output. Deeper layers of the NN can recognize more complex features, as they reprocess features from the previous layer. This is known as the feature hierarchy [14]. Image recognition requires a very deep neural network composed of multiple layers. It must be able to extract non-linear features and pass them to a classifier that can combine all of the features and make predictions.

[14] show mathematically that for image processing, the best features of a single layer are edges and blobs. This is because they contain the most information that can be extracted from a single, non-linear transformation. It has been shown that the human brain does the same thing. The first hierarchy of visual cortex neurons is sensitive to specific edges and blobs, while deeper regions of the brain that are further down the visual pipeline are sensitive to more complex structures, such as faces.

One type of DNN that is often used in image recognition is the convolutional neural network (CNN). The CNN is a type of feed-forward neural network in which the connectivity pattern between neurons is inspired by the specific organization of the brain's visual cortex. The architecture of a CNN is designed to take advantage of the 2D structure of an input image. Every image is a matrix of pixel values that describe intensity at that point. The range of values that can be encoded in each pixel are a function of its bit size. CNNs are given an array of intensities as input, and the output is numbers that describe the probability of the image belonging to a certain class. One benefit of CNNs is that they are easier to train and have fewer parameters than fully-connected networks with the same number of hidden units [14].

For coin recognition, we used a CNN created by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, which is called AlexNet [12]. The input to the AlexNet is a resized image, while the output is a class-label probability. AlexNet includes object recognition steps such as local feature extraction, feature coding, and learning (see Figure 1). The advantage of such a CNN is that it can adaptively estimate optimal feature representations for datasets, which is a feature that is lacking in the conventional, handcrafted approach. The effectiveness of the AlexNet was proven for large-scale object recognition at the ImageNet Large-Scale Visual Recognition Challenge 2012. In addition, AlexNet has already been tested on image recognition in several contexts including, for instance, face detection [15], maritime vessel identification [16], food recognition [17], and playing video games [18].

There are several frameworks for the training and use of DNNs. Most exploit GPU acceleration and include: Caffe, Torch7, Theano, and CUDA-Convnet2. Here, we used the NVIDIA Deep Learning GPU Training System (DIGITS) [14]. This system is the first interactive software for DNN training using the GPU. It makes it possible to develop, train, and visualize the DNN. The interface is browser-based, and NN behavior can be viewed in real-time. For training, DIGITS uses the popular Caffé Deep Learning

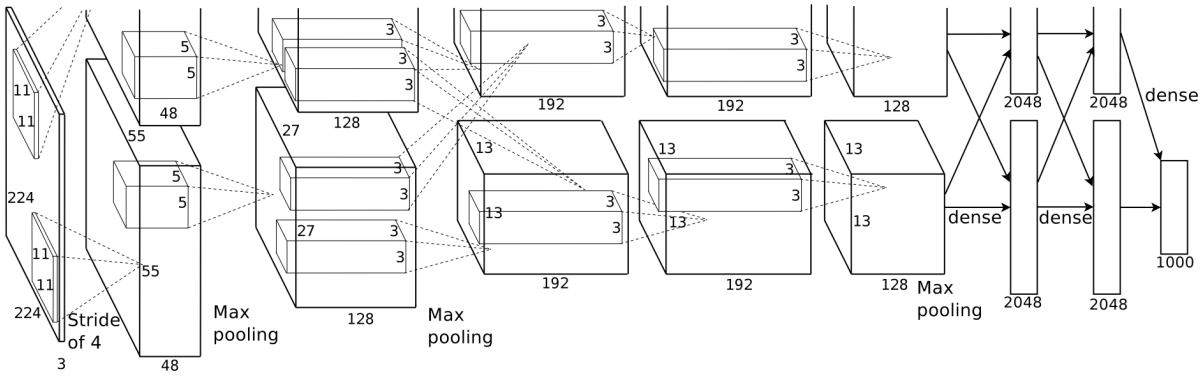

Figure 1. The AlexNet architecture, which consists of an 8-layer CNN [12].

Framework and GPU capabilities to reduce training time [19].

Using DIGITS requires two steps. The first is the creation of the dataset and the classification model through the training process. A set of folders are provided to the system; each contains a series of images of the same class. Then, a text file is defined that contains the labels belonging to each class. The overall dataset is divided into three subsets: training, validation, and testing. The training set is a collection of data for which both the input and the output are known. It is used in the NN training phase and for the creation of the related classification model. The validation set is similar to the dataset used in the validation phase, but designed to avoid overfitting and compare NN performance. This set is used to assess features such as prediction accuracy, loss training and loss validation. Accuracy is the reliability of the prediction based on validation data provided in the training phase. Loss training is the error on the training set. Loss validation is the error after running the validation set through the trained NN. As epochs increase, both validation and training error fall. Training error can continue to fall even after many epochs, allowing the NN to improve its learning. In this case, the validation error increases, eventually leading to overfitting. Finally, the test set is a collection of data used to test and evaluate the performance of the classification model after the training phase.

The second step consists of the creation of a classification model and defining the parameters of the model itself. In particular, this consists of the dataset that was created in the previous step, and the training epochs that represent the number of times the data is passed to the NN. In more specific terms, back-propagation learning algorithms involve two steps. The first step (forward propagation), consists of passing the training data to the NN in order to generate the activation output. The second step (backward propagation), consists of passing the activation output to the NN using the target data to generate a value that is the difference between the output target and NN's current output values for all hidden and output neurons. The aim of backward propagation is to minimize the loss function. The epoch is a forward and backward pass of all training data in the NN.

The snapshot interval represents the number of epochs of training between two snapshots. The validation interval represents the number of epochs of training, running through one pass of the validation data. If a random seed is provided and the same model and dataset are running, the same results should be obtained. The batch size measures the number of inputs processed. The solver type represents the gradient descent optimization method and can be the stochastic gradient descent, the adaptive gradient, or Nasterov's accelerated gradient.

DIGITS makes it possible to edit the pre-configured NN; it is possible to change parameters, add layers, change the bias, etc. Once the dataset has been created, training can begin. In the training phase, DIGITS provides a visualization of the data used and the training state. It also provides an accuracy chart and loss value in real time.

As DIGITS runs on a web server, the dataset and the network configuration can be easily shared with the client. Once the NN has been trained, its (related) model can be queried by supplying an image in the form of an HTTP request. The DIGITS web service responds with predictions that take the form of couple labels (classes) and percentage accuracy. Messages can be exchanged between a client (e.g., a web or mobile application) and DIGITS using the Representational State Transfer (REST) architecture.

IV. The Architecture

The software uses a client-server architecture. DIGITS is installed on the server side and provides services through the *Digit Rest API* programming interface. The interface makes it possible to query a classification or regression model that has been trained on a NN. The API can be used to create a dataset and models, or make predictions using a trained model. The interface is based on the REST architecture. In formal terms, REST consists of a set of components, connectors, and other data within a distributed system, where the focus is on the role of components rather than implementation details. REST

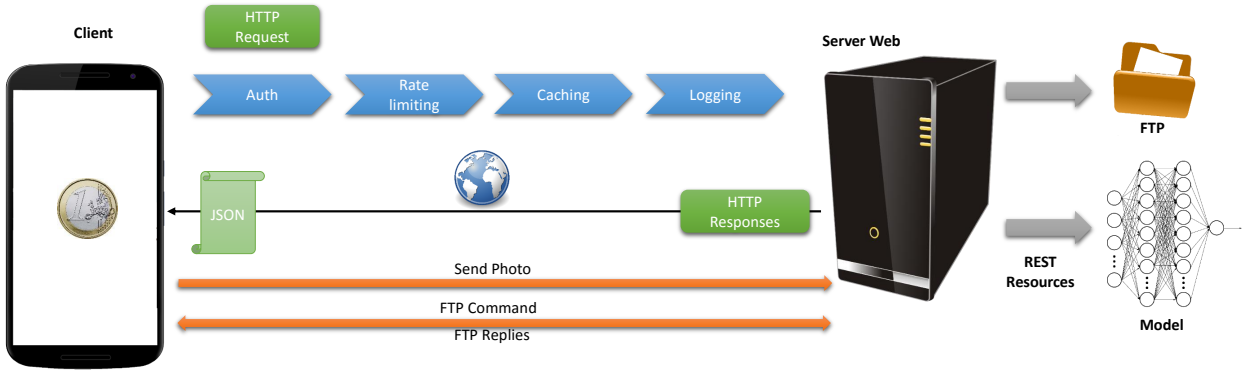

Figure 2. The client-server architecture.

uses the HTTP protocol [20] to transmit data, while the network allows components to exchange representations of resources [21] [22]. Connection elements are connectors that handle communication between components. A component can be a client or a server, and it acts as a mediator, making it possible for the application to interact with a resource given the identifier of the resource itself, and the action to perform. The application must interpret the response, and be able to represent the information. DIGITS includes a server-side component that, once a request is accepted by a client, provides a response in JavaScript Object Notation (JSON)format. In particular, it makes it possible to create a classification (or regression) model and a dataset, to classify an element or an inference using a previously-trained model, and to remove a classification or regression model.

In our work, we focus on the classification of an image of a coin using a trained model. To exploit the remote classification model, we developed a client-side mobile application for Android devices (see Figure 2). This application uses the device's camera to take a photograph of a coin to classify. The resulting image is sent by the client over FTP to the server, then an HTTP POST request that contains the query string is sent to the web service (DIGITS). This query contains the address of the server, the id of the trained classification model, and the location and name of the image to classify. Although it is possible to use a HTTP POST multipart request [23] to send the image directly to the server (together with the parameters specified above), FTP was chosen because the protocol is generally considered faster for sending individual files such as photographs.

Next, the web service's response (in JSON format) is interpreted by the client. The response consists of a list of predictions, each composed of a pair of labels (class), and a percentage accuracy (reliability) of the prediction. This response is displayed on the mobile device running the client application.

An example will help to demonstrate the procedure: the user takes a photograph of the coin and the client software sends it to the server via FTP; it sends an HTTP request specifying the query string name and the location of the image, together with the classification model's id that should be used. The server processes the image given in the query string, classifies it using the named model, and sends the client a list of predictions in JSON format. These predictions include the coin's membership classes and prediction accuracy percentage ordered from the most to least accurate. The application presents the user with the results of the classification.

DATASET	Euro1	Euro2	Euro3	Euro4	Euro5
MODEL	M1	M2	M3	M4	M5
TRAINING	60%	50%	55%	60%	59%
VALIDATION	35%	45%	37%	37%	38%
TEST	5%	5%	8%	3%	3%
EPOCH	10	10	10	10	10
ACCURACY	62.93%	42.65%	71.09%	75%	77.21%
LOSS TRAIN	0.63%	1.11%	0.51%	0.47%	0.30%
LOSS VAL	0.65%	1.10%	0.53%	0.45%	0.41%

Table I
TRAINING DATASET AND CLASSIFICATION MODEL WITH RELATED PARAMETERS.

V. THE TRAINING PHASE

Input images consist of 500×500 pixels, giving an input dimensionality of 250,000. There are eight classes (corresponding to 0.01€, 0.02€, 0.05€, 0.10€, 0.20€, 0.50€, 1€, and 2€coins). It should be noted that here we only take into account the reverse side of the coin (portraying a map of Europe). For each of the eight classes, the training dataset consisted of 80 images taken with a camera at a distance of 10cm. To increase the number of coins, we performed 13, 2D transformations. In particular, coins were rotated by $60°$ around their center, mirrored, then and rotated again by $60°$. The final dataset consisted of 1,040 images for each coin, and a total of 8,320 coin images.

To obtain the best model, we performed several experiments on the input images using five different datasets. Each dataset had a different percentage of training, validation, and test data. The number of epochs was set to 10, and the batch size was set to 24 (the default). This approach is a useful way to establish the minimum number of input images needed to produce a useful model. Table I shows how the datasets and classification models were divided. The first and second row specify the name

COIN	M1	M2	M3	M4	M5
0.01€	0%	0%	100%	100%	100%
0.02€	100%	100%	100%	100%	100%
0.05€	100%	0%	100%	100%	90.3%
0.10€	5.7%	21.2%	48.2%	32.3%	96.8%
0.20€	50%	24.7%	89.2%	19.3%	77.4%
0.50€	0%	53.8%	62.6%	22.6%	80.6%
1.00€	100%	90.4%	90.4%	90.3%	100%
2.00€	92.31%	90.4%	84.3%	100%	100%

Table II
TEST PERFORMED ON THE TRAINED CLASSIFICATION MODELS. MODEL M5 PERFORMS BEST IN THE CLASSIFICATION OF TEST IMAGES.

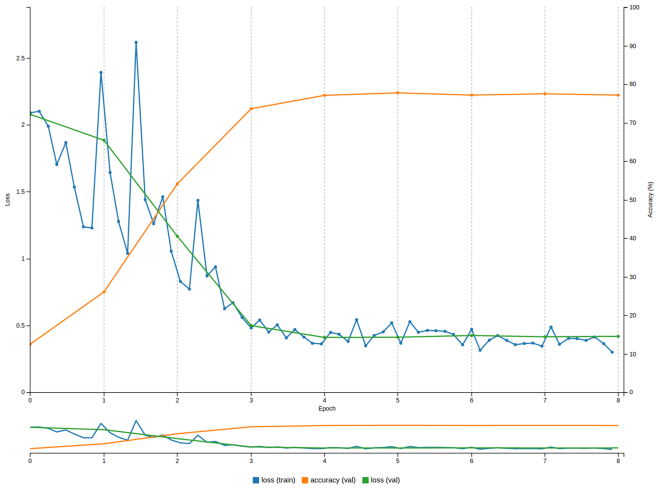

Figure 3. Loss training, accuracy validation (Test) and loss validation for model M5.

of the dataset and model (respectively), while the other rows represent, in order: the percentage of training data, validation data, test data, the number of epochs, percentage accuracy, loss training, and loss validation.

Table II shows the tests that were performed on all of the classification models. These tests consisted of passing the test image folder to the respective classification model. Each test image folder is composed of a percentage of images defined in the dataset creation step (see Table I). As Table II shows, each test is associated with a class of coin that is defined in the first column. If the percentage for each coin is greater than 50% the test is passed, otherwise it is failed. Table II shows that the model M5 provided the most accurate classification. Therefore, for our input image dataset, this is the optimal recognition model.

In more detail, the accuracy of model M5 is 72.21%. Figure 3 shows trends for each epoch in the training phase. The blue line indicates loss value trends, the orange line indicates accuracy trends, while the green line presents the loss value trend for the validation set. It should be noted that from the fourth epoch onward, accuracy percentage is stable at around 80%, and loss value remains low. Training and testing were performed on a NVIDIA 6GB TITAN GPU, which significantly improved the performance of the deep learning classifier.

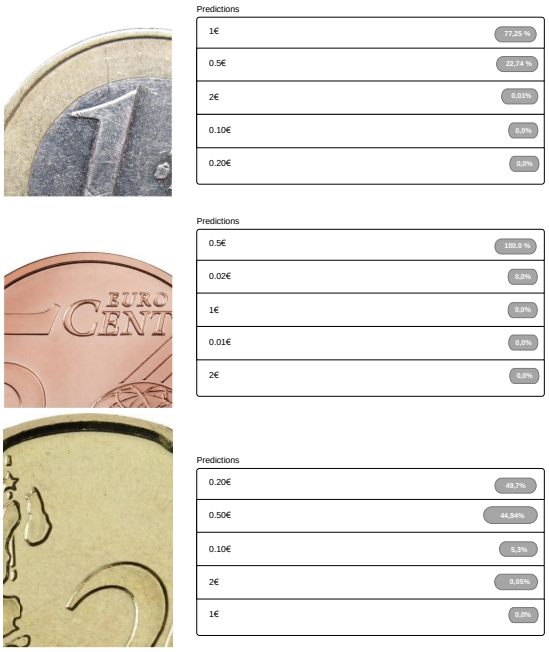

Figure 4. The partial euro coins used to test the neural network.

We also evaluated the response of the optimal model M5 on partial images of 1.0€, 0.05€, and 0.20€ coins as shown in Figure 4. This test was not exhaustive, but did give an indication of performance in the wide range of production conditions associated with coin recognition. The percentage accuracy of the prediction was 77.25%, 100%, and 49.7% for 1.0€, 0.05€, and 0.20€ coins respectively. It should be noted that the accuracy of the test on 0.20€coins was 49.7%, which is highest percentage compared to other accuracies obtained from the same image.

Finally, Figure 5 shows the mobile app in use. The interface is very simple and enables the user to take a photograph using the smartphone's camera, or select an image from a local folder and send it to the server. Once the server-side model performs the recognition, the best five relevant results are highlighted.

VI. CONCLUSIONS

DCNN cannot be applied to all datasets because it requires a lot of training images in order to be able to achieve comparable (or better) performance than conventional, local-feature-based methods. In our preliminary coin recognition experiments, we trained a DCNN on a pre-prepared dataset of images, which confirmed that the amount of training data was enough. The performance of the best model showed that 70-80 coin images are needed in the training phase. More specifically, coins had medium to light overall wear, and all details were visible.

The software architecture that we implemented proved to have substantial advantages: these include an immediate response to the client, and the ability to use GPUs for training, validation and testing of the NN in order to classify models quickly. We developed an application for

Figure 5. Mobile coin recognition application. On the left, the best five results.

Android-based mobile devices that provide a visualization of the results of the prediction based on an image of a coin, obtained using the device's camera. In the future, we will use the Android-based application and a serious game to create a large training dataset [24].

We argue that by using embedded devices (such as the NVIDIA Jetson TX2 System) this approach can be used in other contexts. These could include, for instance, a currency detector for retail kiosks, self-checkout machines, or gaming machines to detect counterfeit coins. The basic principle of the application is to test the coin's physical properties. With respect to future work, we plan to use the approach for recognizing ancient coins. The challenge in this field mainly concerns the reconstruction of worn coins. In this case, the NN can be applied to the partial recognition of images, by isolating the best-preserved parts of an ancient image. As deep learning techniques continue to progress, we are confident that this technology could be applied to ancient numismatics with important results.

References

[1] S. Zambanini, M. Kampel, and M. Schlapke, "On the Use of Computer Vision for Numismatic Research," in *Proceedings of the 9th International Conference on Virtual Reality, Archaeology and Cultural Heritage*, VAST'08, (Aire-la-Ville, Switzerland, Switzerland), pp. 17–24, Eurographics Association, 2008.

[2] M. Fukumi, S. Omatu, F. Takeda, and T. Kosaka, "Rotation-invariant neural pattern recognition system with application to coin recognition," *IEEE Transactions on Neural Networks*, vol. 3, pp. 272–279, Mar 1992.

[3] P. Davidsson, "Coin classification using a novel technique for learning characteristic decision trees by controlling the degree of generalization," in *Ninth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, Gordon and Breach Science Publishers, 1996.

[4] M. Nölle, H. Penz, M. Rubik, K. Mayer, I. Holländer, and R. Granec, "Dagobert-a new coin recognition and sorting system," 2003.

[5] M. Reisert, O. Ronneberger, and H. Burkhardt, "An efficient gradient based registration technique for coin recognition," in *Proc. of the Muscle CIS Coin Competition Workshop, Berlin, Germany*, pp. 19–31, 2006.

[6] L. Van Der Maaten and E. Postma, *Towards automatic coin classification*. na, 2006.

[7] M. Zaharieva, M. Kampel, and S. Zambanini, "Image based recognition of coins – an overview of the COINS project," in *Performance Evaluation for Computer Vision 31st AAPR/OAGM Workshop 2007*, pp. 57–64, 2007.

[8] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 5 2015.

[9] R. Bremananth, B. Balaji, M. Sankari, and A. Chitra, "A New Approach to Coin Recognition using Neural Pattern Analysis," in *2005 Annual IEEE India Conference - Indicon*, pp. 366–370, Dec 2005.

[10] J. Kim and V. Pavlovic, "Discovering Characteristic Landmarks on Ancient Coins using Convolutional Networks," *CoRR*, vol. abs/1506.09174, 2015.

[11] S. Modi and D. S. Bawa, "Article: Automated Coin Recognition System using ANN," *International Journal of Computer Applications*, vol. 26, pp. 13–18, July 2011. Full text available.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[13] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

[14] "NVIDIA Deep Learning." https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts. Accessed: 2016-09-19.

[15] S. S. Farfade, M. J. Saberian, and L.-J. Li, "Multi-view Face Detection Using Deep Convolutional Neural Networks," in *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, ICMR '15, (New York, NY, USA), pp. 643–650, ACM, 2015.

[16] C. Dao-Duc, H. Xiaohui, and O. Morère, "Maritime Vessel Images Classification Using Deep Convolutional Neural Networks," in *Proceedings of the Sixth International Symposium on Information and Communication Technology*, SoICT 2015, (New York, NY, USA), pp. 276–281, ACM, 2015.

[17] Y. Kawano and K. Yanai, "Food Image Recognition with Deep Convolutional Features," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, UbiComp '14 Adjunct, (New York, NY, USA), pp. 589–593, ACM, 2014.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari With Deep Reinforcement Learning," in *NIPS Deep Learning Workshop*, 2013.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, (New York, NY, USA), pp. 675–678, ACM, 2014.

[20] H. Li, "RESTful Web service frameworks in Java," in *Signal Processing, Communications and Computing (ICSPCC), 2011 IEEE International Conference on*, pp. 1–4, Sept 2011.

[21] C. Pautasso, E. Wilde, and R. Alarcon, *REST: Advanced Research Topics and Practical Applications*. Springer Publishing Company, Incorporated, 2014.

[22] N. Capece, R. Agatiello, and U. Erra, "A Client-Server Framework for the Design of Geo-Location Based Augmented Reality Applications," in *2016 20th International Conference Information Visualisation (IV)*, pp. 130–135, July 2016.

[23] L. Masinter, "Returning values from forms: multipart/form-data," 2015.

[24] R. D. Chiara, V. D. Santo, U. Erra, and V. Scarano, "Real positioning in virtual environments using game engines," in *Eurographics Italian Chapter Conference 2007, Trento, Italy, 2007*, pp. 203–208, 2007.