Coin Classification using Convolutional Network

Domantas Meidus Alessandro Pomes

January 4, 2018

Abstract

Classifying ancient coins is a highly complex task that requires years of experience in the entire field of numismatics. As a part coin analysis, coin classification can be supported and facilitated by an automatic image-based. The intent of this project is to classify a coin following the recognition image method. Convolutional Neural Network are used in this approach to identify Roman coins between others types of ancient coins.

Contents

1 Description of the problem **1**

2 Description of our approach **2**
2.1 Research . . . 3
2.2 Preprocessing . . . 3
2.3 Classifiers . . . 3
2.4 Validation . . . 4

3 Implementation **4**
3.1 Dataset expansion . . . 4
3.2 Data Handling . . . 5
3.3 Feature Extraction and Selection . . . 6
3.4 Classification . . . 6
3.5 Convolution Neural Network . . . 7

4 Results **8**

5 Conclusions **10**

1 Description of the problem

Two different Dataset are used to compute the problem:

1. CoinsDataset Scraped dataset of 2600 images of coins.
2. Coin Image Dataset is a dataset of 60 classes of Roman Republican coins. Each class is represented by three coin images of the reverse side acquired at Coin Cabinet of the Museum of Fine Arts in Vienna, Austria.

Finally the two Datasets have been collected as below described:

- Images : 2629 coins images. 180 of 2629 are coins from Roman times.

Considering the distribution of classes into the dataset the training problem result is unbalanced.





2 Description of our approach

The approach presented below is based on th Computer Vision method to classify images of Ancient Coins.
Good results can be archived adopting a computer Vision tool like Convolutional Neural Network

, but improving the results of the "state-of-the-art" still remains a hard challenge .
One of the difficulties that arise using a classification image approach is due to the quality of the images that are saved in the datasets. In fact small differences from coins to other coins should be preserved and in the learning process a good resolution image should be maintained.
Other common issues that can falsify the process from a 2D image can be the state of conservation of the coin and the different ways that the light is reflected in the reliefs and on the border.
Good and precise habits must be applied when a Dataset is built [5]. Another approach can be made by using a 3D coins scanning[4], but these type of 3D acquisitions are more laborious and expensive than traditional 2D imaging. Finally, the approach chosen allows to use a machine learning tool already tested from other fields of studies. This provides a solid implementation of the thesis and also the most common python libraries offer Convolution Network, therefore adapting the problem can be fast and powerful.

2.1 Research

An important but not simple task of numismatic research is to classify coins.
There are millions of coin types from all over the world ranging back far until 700 BC. Classification means to find out the correct reference number of the coin in the reference numismatic literature [1]. In spite of there is no definition of a unique reference literature.
Nowadays ancient coins are also considered as pieces of art which reflect the individualism of the engravers who manually cut the dies used for minting the coins. Roman coins, for instance, often depict portraits of gods and emperors or historical events, in a similar manner as sculptures or paintings from this era do. The main problem of a coin classification problem is time. In fact a Numismatic devotes a lot of time to the task of researching from which era the coins are.
For these reasons many approaches[2][3][4] were used in these days to make the work easier for archaeologists.
Some of these approaches use Machine learning tools. However, others use different methods like SIFT flow that is based on the Scale Invariant Feature Transform who is able to detect differences between images.[3][4]

2.2 Preprocessing

As

2.3 Classifiers

Convolution Neural Network architectures make the explicit assumption that the inputs are images. Using this assumption the approach is based on putting as input several three-dimensional structure arrays (3-dimensions corresponding to the 3 channels Red-Green-Blue) and returning as the output one vector that belongs to the classes: Roman or not Roman coin. The Convolution neural networks characteristics could be summaries as follow:

- Architecture is in the simplest case a list of Layers that transform the image volume into an output volume
- There are a few distinct types of Layers
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a function
- Each Layer may or may not have parameters

- Each Layer may or may not have additional hyperparameters.

Layers of a CNN are Convolutional, Pooling layers and fully connected Layers.
Convolutional Layers are in charge to learn the weight of the process and are composed of a fixed number of filter **K** ,**K** a number of spatial extent **F**, the stride **S** and a amount of zero padding **P**. Number **K** correspond to the final depth dimension of output volume while **F** is the dimension of the spacial extent that is computed in every convolutional step when the final output volume is produced. Instead, the number of shift step made in the convolutional calculations is represented by **S**.
So the output Volume should set with the follow parameters:

- $Weight = (W - F + 2P)/S + 1$
- $Height = (H - F + 2P)/S + 1$
- $Depth = K$

Where W and H are the Weight and height of the input Volume.
MaxPool Layer is placed in successive Convolutional layers in a CNN architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network.
Using the $max()$ function in a previously set spacial extent it compute the max value of each kernel size in the input volume. So like in the conv layers we have to set the kernel size and the stride value.
The final Fully connection layer is a general neural network layer that compute the matrix product from the input (all activations in the previous layer) and the weights, and it returns the final output(that has the dimensions of the final labels).

2.4 Validation

Due to the problem unbalance classes to in the Dataset the team splitted it in two parts as usual (Training set and Test set) but leaving the Test set on 40% of total. Doing this, the percentage of the two classes, turn out to be well-balance.

3 Implementation

Roman coins dataset consist of **2629 coins images**. Only 180 of 2629 (6,85 %) images are from Roman times coins. In project implementation part there are 3 goals to achieve:

1. Expand coins dataset
2. Prepare images dataset for Convolution Neural Network.
3. Classify images using Convolution Neural Network.

3.1 Dataset expansion

Original coins dataset has 2629 coins examples. For the better results, project coins image dataset is expanded to 21032 images by **flipping and rotating** each image:

1. Original dataset : 2629 coins images. 180 of 2629 are coins from Roman times.

2. Flipping: 10516 with flipped coins images. - For each coin image adds 4 images of flipped coin versions.

3. Rotating: 21032 with rotated coins images. - For each flipped coin image adds 2 images of rotated coin version.

Expanded dataset consist of 21032 images number representation which is fallowing this order: 2629 (Base coin images) * 4 (Each original coin image is flipped 4 times) * 2 (Each flipped image is rotated two times). Images in the original dataset is constructed based these rules:

- Each non Roman coin have 2 images: one image represents front of the coin, other image represents back of the coin.
- Each Roman coin have 3 images which represents the same coin appearance with different quality coins. The reason behind that Romans coins are old and the picture on the top is usually faded.
- Roman coin name starts with substring **"class"**.
- Each coin image has **".png"** or **".jpg"** extension.

For rotating and flipping coin images cv2 python library was used. Project follow this order of operations:

1. Images are flipped. Each coin image with extension ".png" or ".jpg" from original coins dataset are traversed and flipped using cv2 library flip function. Each coin image is flipped into 4 additional images: image of flipped coin right, image of flipped coin left, image of flipped coin bottom left, image of flipped coin bottom right.

2. Images are rotated. Each coin image with extension ".png" or ".jpg" from flipped coins dataset are traversed and flipped using cv2 library flip function. Each coin image is flipped into 8 addition images - for each flipped coin created 2 rotated image instances: flipped coin image and flipped coin image rotated by 90 degree.

During flipping and rotating procession, all images are created and resized to **64x64** pixels. Folders named "Flipped" and "Rotated" should be **existing in the same file system** with this project, because flipped and rotated images are saved into separate folders.

3.2 Data Handling

Data for project is saved into H5 dataset as hdf5 format. The dataset named **"coins.h5"** appears in the same file directory as python code of the project was executed. Dataset is created when "coins.h5" dataset **is not existing** in the project file directory. Each data compressed to the h5 dataset explanation:

1. **X_train** - 4-rd dimensional vector of Coins images RGB representation for training data. Training data consist first 60% (12619) part of total generated coins images. Shape of training data vector is (12619, 64, 64, 3).

2. **X_test** - 4-rd dimensional vector of Coins images RGB representation for testing data. Testing data consist first 40% (8413) part of total generated coins images. Shape of testing data vector is (8413, 64, 64, 3).

3. **roman_coins** - List of all coins indexes which was recognized as Roman coins.

Each generated coin image is converted to the RGB representation using cv2 library method: astype and saved as a list - python data structure. After all coins images RGB representation is stored to the list, list is converted to the numpy array list data structure.

3.3 Feature Extraction and Selection

Features of the project are each coin image pixel RGB representation which have to be extracted from the h5 dataset. Features in the dataset are saved in names: **X_train** and **X_test**. Features are extracted from the h5 dataset to a multidimensional, homogeneous array of fixed-size items - **numpy.ndarray**. H5 dataset is iterable, so each coin image representation of X_train and X_test datasets was traversed and stored to the numpy multidimensional arrays: **x_train** for training data and **x_test** for testing data. x_train shape is equal to **(12619, 64, 64, 3)** and x_test shapes is equal to **(8413, 64, 64, 3)**:

- First shape element represents a number of total coins images RGB representation stored in the data structure.
- Second shape element represents a number of coin image **height** by pixels.
- Third shape element represents a number of coin image **width** by pixels.
- Four shape element represents a number of RGB coin image pixel.

Extracted feature data is used for Convolution Neural Network.

3.4 Classification

Coins images for Convolution Neural Network are classified into categories: Roman coins and non Roman coins. Classification part follows these steps:

1. During dataset creation by executing flip, rotate and save functions all coins images indexes which applies Roman times coins features are saved into a list (more information about Roman coins features: Roman coins categorization features
2. Roman coins indexes are compressed into h5 dataset.
3. Roman coins indexes from h5 dataset are converted to the python friendly list data structure.
4. Created python lists to store labels training and testing data: **y_train** and **y_test**.
5. Training labels are created by iterating each features training data. If the index of features training coins image data is in the Roman coin index list, the value of labels list: y_train is equal **1**, otherwise it equals to **0**.
6. Testing labels are created by iterating each features training data. If the index of features training coins image data is in the Roman coin index list, the value of labels list: y_train is equal **1**, otherwise it equals to **0**.
7. Labels training and testing lists are converted to numpy arrays.

Roman coins contribution in the labels testing and training data:

- Labels training data **y_train** contains **1668 Roman coins** of 12619 total coins.
- Labels testing data **y_test** contains **599 Roman coins** of 8413 total coins.

3.5 Convolution Neural Network

Parameters short explanation for Convolution Network:

- **Height and Width** - Image height and weight. Our images are made of 64*64 pixels, so as a height and width numbers are the same.
- **Channels** - Represents image color set. In our project images consist of combination of three colors: red, green, blue(RGB). In this case channels value is 3.
- **n_inputs** - Number of inputs which are calculated by multiplying height, width and channel number. In our project inputs value is 12288(64*64*3).
- **ntrain** - Number of all instances of training set (18928).
- **ntest** - Number of all instances of testing set (2104).
- **conv1_fmaps, conv2_fmaps** - Feature map for the first(conv1_fmaps) and second(conv2_fmaps) layers. The feature map is the output of one filter applied to the previous layer. A given filter is drawn across the entire previous layer, moved one pixel at a time. Each position results in an activation of the neuron and the output is collected in the feature map.
- **conv1_ksize, conv2_ksize** - Kernel size for first and second layers. Kernels are used for feature detection.
- **conv1_stride, conv2_stride** - The amount of shifts.
- **conv1_pad, conv2_pad** - Padding parameters. Parameter value is equal 'Same' which is Trying to pad evenly left and right, but if the amount of columns to be added is odd, it will add the extra column to the right
- **pool3_fmaps** - Pooling feature map.
- **n_fc1** - Fully connected layer. Fully connected layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.
- **n_outputs** - Number of classifiers. Outputs value equals to 2, because there are two possible prediction outputs: 0(Non Roman coin) and 1(Roman coin).

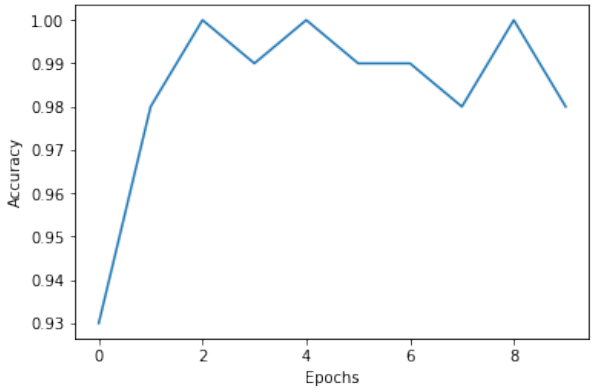
Properties of Convolution Neural Network:

- 2 convolution layers: **conv1 and conv2**. These two layers creates a convolution kernels that are convolved with the layer input to produce a tensor of outputs.
- **The maxpool layer**. Max pool layer is applied to each of the second convolution layer feature map filters, and since the inputs have size: 14x14, Stride=2 and Padding=Valid, after applying max pool. Max pool feature map filters size is 16x16 (Because each coin image height and weight is 64x64 pixels and image is divided to 4 parts (64/4, 64/4) = (16, 16).
- **Full layer**: fc1 = tf.layers.dense(pool3_flat, n_fc1, activation=tf.nn.relu, name="fc1")
- **Output layer**. The output layer is where the network produces a classification for each class. The classification is used using the function softmax.

- **Loss functions** and **Optimizers** function.
- **Correct function** - how many coins images were correctly classified. Correct function representation: tf.nn.in_top_k(logits, y, 1) where logits(predictions) = output layer, y(targets) = Labels placeholder, 1(number of top elements to look at for computing precision.)
- **Accuracy** - Calculates classification accuracy of the training and testing data.

4 Results

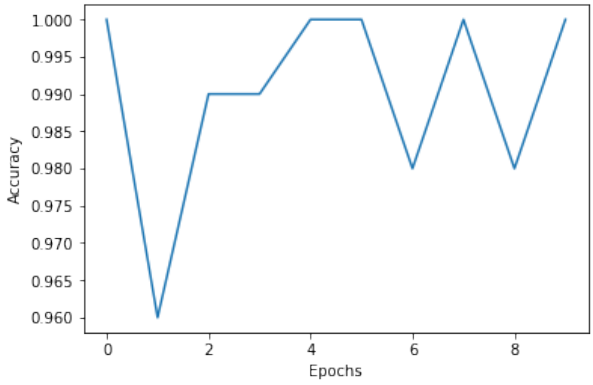
In this section we summarize results for classification of coins and identification of the roman ancient coins.
Before selecting the standard configuration presented above the follow test have been made.
CNN with 1 Convulutional Layer:



Accuracy mean: 0,98

Standard deviation : 0, 015 (1)

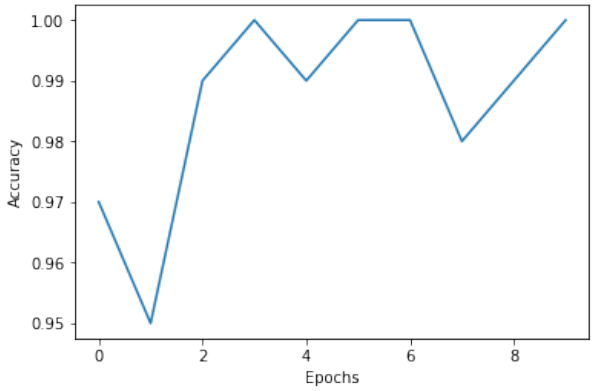
CNN with 2 Convulutional Layer:



Accuracy mean: 0,99

Standard deviation : 0, 012 (2)

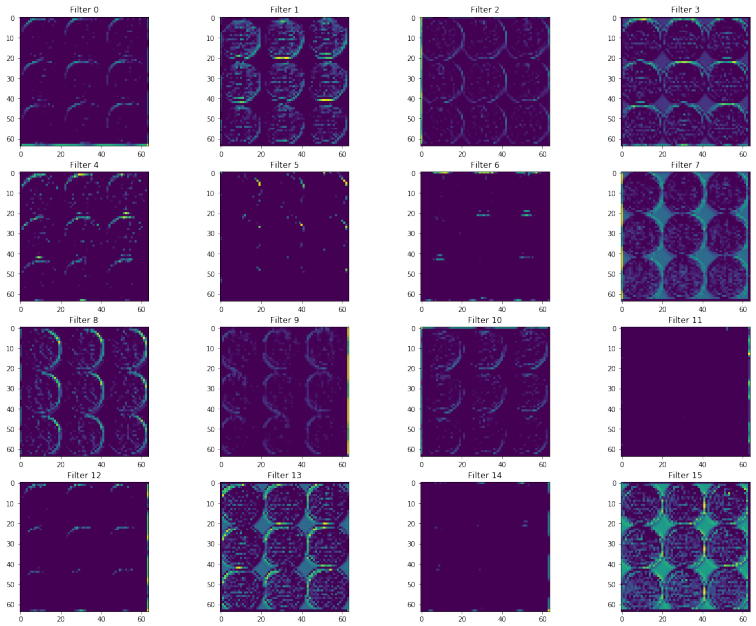
CNN with 3 Convulutional Layer:



Accuracy mean: 0,98

Standard deviation : $0,019$ (3)

As we can see we can not highlight any evident differences growing up the depth of the Network.
To understand what exactly the network is doing when given any specific input is provide here below the plot of the filers of the first a convolutional layer[1]



Different filter learned different characteristics of the images.

[1]This plot is considered for the network presented in the report with 2 ConV layer and for a casual image of the dataset

5 Conclusions

The results obtained to recognize the Roman Coins are very high. The network is able to detect almost all the coins in the Test set.
Checking the datasets is easy to know that the romans coins are not perfectly circular even watching all filters of the network. Even changing the parameters of the network we can not see any different results.
Anyway even if it can be considered a "toy" problem, not only can be extended for a large set of class in the fields of numismatics but it can be a useful approach for any kind of Image classification. The advantage to use is that it isn't very computational heavy run all the network and testing how a Convolution Network work. Add some modifications can be permissive even with a common laptop.
These type of Network can be extended to all the problems of Image recognition and future developments can be done very easily putting in this Network new Database with many different labels although using more powerful Computation.

References

[1] Grierson, P.: Numismatics. Oxford University Press (1975) *Numismatic*.

[2] S. Zambanini, M.Kampel and M. Schlapke *On the Use of Computer Vision for Numismatic Research*. Pattern Recognition and Image Processing Group, Vienna University of Technology, Favoritenstr. 9-183/2, A-1040 Austria Thüringisches Landesamt für Denkmalpflege und Archäologie, Weimar, Germa

[3] Sebastian Zambanini and Martin Kampel *Coarse-to-Fine Correspondence Search for Classifying Ancient Coins* Computer Vision Lab, Vienna University of Technology, Austria

[4] Reinhold Huber-Mörk, Michael Nölle, Michael Rubik Michael Hödlmoser, Martin Kampel and Sebastian Zambanini *Automatic Coin Classification and Identification* Department Safety and Security, Austrian Institute of Technology Computer Vision Lab, Vienna University of Technology Austria

[5] Sebastian ZAMBANINI– Martin KAMPEL *Coin Data Acquisition for Image Recognition* Vienna University of Technology, Institute of Computer Aided Automation, Pattern Recognition and Image Processing Group,Vienna, Austr

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*

[7] Computer vision Stanford CS class
`http://cs231n.github.io/convolutional-networks/`