Coin Classification with Machine and Deep Learning Methods

Adi Dahari, Shlomo Glick, Gilad Shotland
Ariel University

Introduction

The field of Numismatics - the study of ancient coins and ancient money is a significant source of knowledge about the ancient world and cultures. To enable the researchers in this field to do their studies, a good and reliable analysis of archaeological findings is needed. For example, if a researcher wants to study coins from a certain era or ruler, he needs to be sure that the coins that he is studying belong to the specific era or ruler.
Throughout the entire development of this field, most of the archaeological findings were classified by the human eye and hand. The amounts of the coins are huge, and these processes can take a very long time - months and even years. Therefore, the use of Machine Learning classification methods can benefit such research by enabling a fast, reliable, and hands-off classification process that will accelerate both the speed of study as well as the resource usage and focus on the important sections the of study.

In some cases, the same type of coin may have been produced by several rulers, and the differences between them are reflected in the name of the ruler that is written on the coin, so reading a text on an ancient coin, sometimes in ancient language or ancient alphabet, is needed. Moreover, another problem that makes the classification mission more difficult, is that sometimes the coin may be corrupted or worn.

This project offers and tries a Deep Learning approach for automating the process of coin classification concentrated on identifying ancient Hebrew coins of 3 Hasmonean rulers: John Hyrcanus I, Alexander Jannaeus, and Judea, who produced the same coin - Half Pruta.
by the supervision of Professor Lee-Ad Gottlieb from the department of Computer Science at Ariel University and the guidance of Mr. Yaniv Levi from the department of Archaeology at Bar-Ilan University.

Building A Data Set

As mentioned above, the coin that was chosen by the guidance of Mr. Levi is Half Prutah. There was no tagged and arranged data set of this coin, therefore building a custom dataset was needed.

For applying learning algorithms a massive amount of data is needed, so downloading images one by one is a Sisyphean task - which was automated by building web scrapers that downloaded all the pictures. The web scrapers were built with BeautifulSoup library and Selenium framework which provide tools for easy parsing of HTML documents, as well as JavaScript-based sites.

After downloading the images, we examined the data and removed some images that were mistakenly downloaded and/or contained a large amount of noise.
Most of the images included the reverse and the obverse sides of the coin together, and since the project concentrated on the reverse side, splitting the images was done using Python's Pillow Module.

Examining Different Approaches

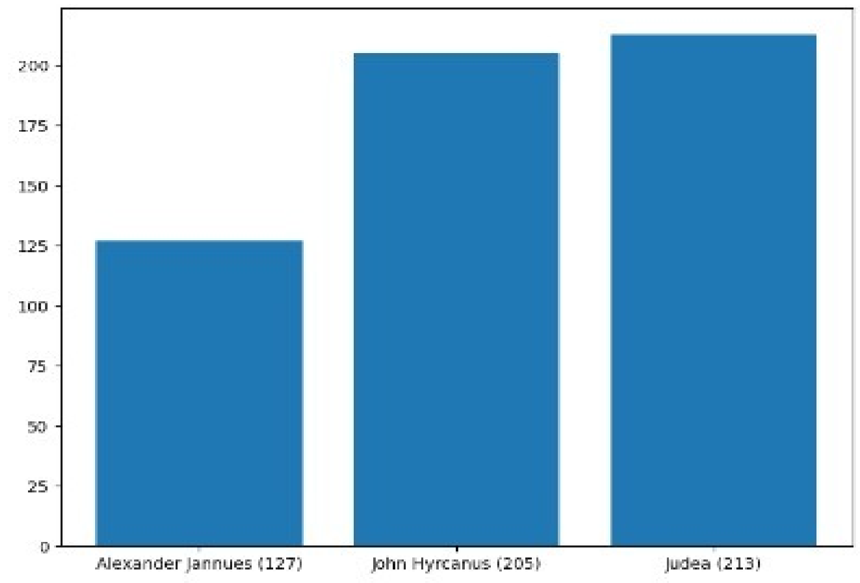
As we finished building a dataset, we were able to start building and training a model.

In the last decade, the most popular approach for image classification is training a Convolutional Neural Network, while the convolution layers are handling the curse of the high dimension - the operation of the convolution provides a dimension reduction, AKA feature extraction from the raw pixel information, and the rest parts, the fully connected layers, handing the classification task. Training a whole CNN from scratch for certain data is a pretty heavy and complicated task - by picking the right architecture, right hyperparameters, etc...
To avoid problems with architecture, we first decided to train the well-known architecture of Alexnet from scratch. Although this architecture proved itself as a reliable form of neural network for image classification, the training sessions weren't easy and ended in a divergence of the loss.
Therefore we decided to try another famous approach of Transfer Learning, by using a pre-trained Convolutional Neural Network.

The pre-trained model can be considered a backbone for the feature extraction procedure. For classifying the new dimension reduced sample we used a custom trainable layer on top of it.

Training

The PyTorch framework provides several pre-trained state-of-the-art CNN models, such as Squeezenet, Resnet, VGG, Alexnet, etc... Resnet and VGG are well known for simplicity and providing good results, therefore we decided to concentrate on these networks. For working with these networks we resized the images to 224x224 pixels.
The new models we built were designed by editing the last layer of the networks.
All along the training sessions, the lack of data (less than 1000 images in total) and the amount of noise in the images were significant difficulties. At first, both of the models gave about 63% of accuracy on the validation set.
For making a more rich and robust data set, we've taken the technique of data augmentation. By rotating each image while loading a training batch by 5 different degrees, which is multiplied by 6 times the size of the training set. With additional usage of momentum during the optimization process, the response of the networks was incremented on the accuracy scale - while the VGG-based network gave about 70% of accuracy, the Resnet gave approximately 75%, although the loss function didn't converge to a small value.



Transfer Learning approach - Different Learning Models

As mentioned, the main role of the backbone network is providing a reliable dimension reduction, which means transforming the raw pixel features into a vector in a latent space that gives a good description of a sample. This fact leads to a question - Will other decision models give good or close results to the Softmax model?

To answer this question we used the convolutional layers of the pre-trained VGG model provided by PyTorch and extracted features for all the data set, which means building a kind of a new data set of vectors with 4096 dimensions.

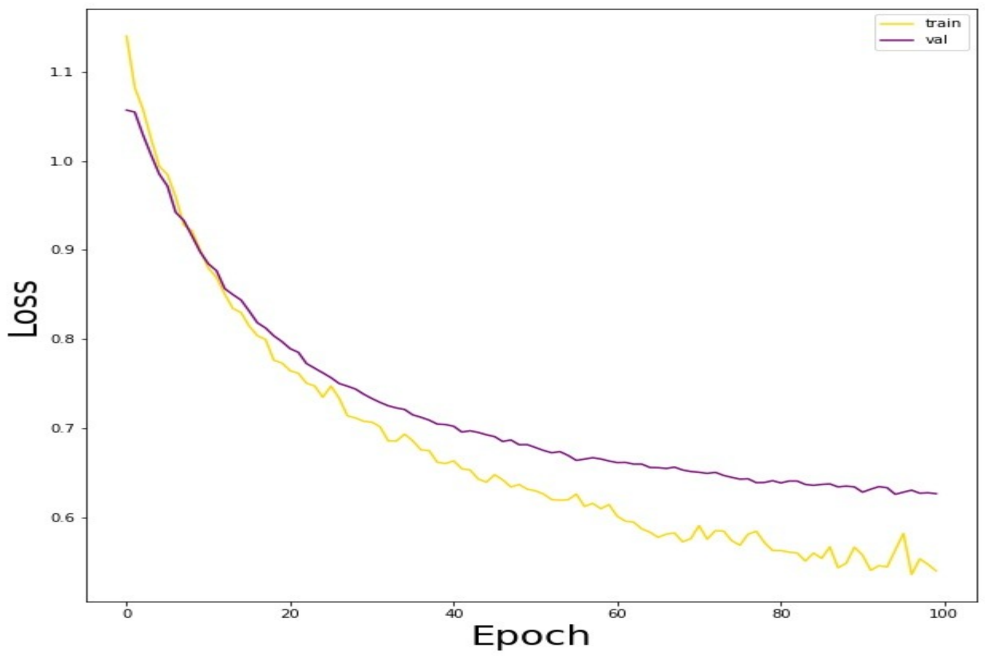
We used two techniques on the dimension-reduced data set - KNN and SVM, with the Sci-Kit Learn Machine Learning framework.

Both of the techniques proved not so robust but yet solid ability to learn when KNN using 10 neighbors gave an accuracy of 66% and SVM with Polynomial Kernel gave an accuracy of 70%.

More Thoughts - Deeper Neural Network

The relative success of the KNN and SVM models assured that the feature extraction is done well by the backbone and led to examining the technique of a new deep neural network on the vectors data set. We've designed a neural network with 4 fully connected layers with batch normalization and dropout. The results of the training were pretty surprising - the accuracy of the network was 66%, but the loss was extremely lower - 0.03 (in contrast to 0.8 with the Resnet-based model). This means that the new model may be less accurate, but when it makes a wrong decision it is less confident.

Model Learning Process of Loss for Train & Val



Wrapper - User Interface

As part of our work, we built a simple, easy-to-use interactive web application that allows the user to upload an image of a coin, send it to the Server, and retrieve its prediction of which ruler-era the coin is related to.
Both Server and Client lie in a Dockerfile that enables each of them to run on an isolated, out-of-the-box runtime, which can be run on any interested party computer.
Docker environment with Kubernetes also allows the usage of multiple containers, that minimize any necessary downtime of the application.

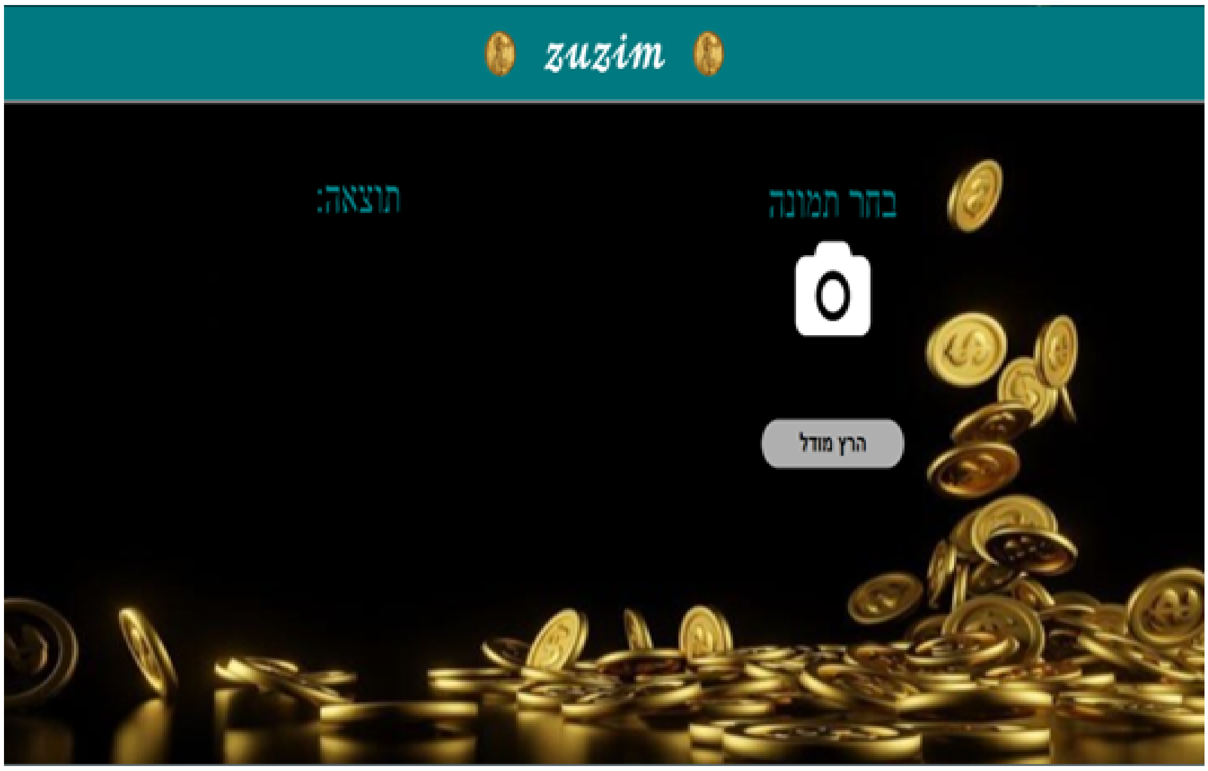
Server

Written in Python with Flask library. Listens to any request made by clients over HTTP communication. This party connects to the model and pre-processes the image before enquiring for prediction. The Server also allows injecting new models into its runtime. (Full and deep explanation about how to run the docked server lie in the Servers readme file).

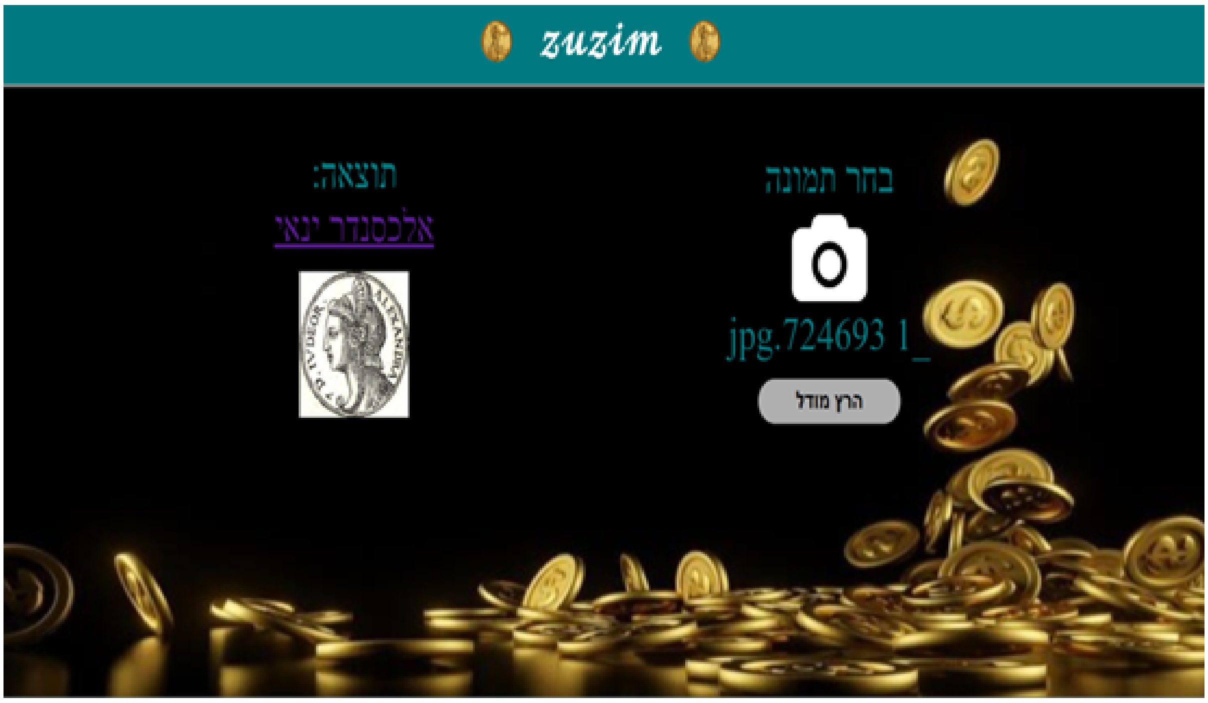
Client

Written in Vanilla JavaScript with HTML and CSS. enables easy and comfortable usage of our system. The user is asked to upload an image of a coin by browsing his local file system and selecting the targeted image. After uploading the image to the client app, the page enables a “Run Model” (“הרץ מודל”) button that sends the image to the server and retrieves the result of the prediction after just a few seconds. An example of use in next page.

1. Landing page, choose an image and submit button



2. Result from the server after the model's prediction.


zuzim
תוצאה:
אלכסנדר ינאי
בחר תמונה
jpg.724693 1_
הרץ מודל