A Comparison between Vision Transformers and Convolutional Neural Networks based on the Classification of Archaeological Artefacts

Jonas Buchart Svenningsen [*1], Magnus Just Sund Carlsen [†1], and Mate Guber [‡1]

[1]Department of Computer Science, Aalborg University, Aalborg

January 2024

Abstract

This research is a comparative exploration between two Vision Transformers; one trained on our dataset, another pre-trained, and a Convolutional Neural Network. Leveraging our dataset comprising 80.000 coin images and 220.000 artifacts, this study involves hyperparameter tuning and model comparisons. The CNN showed promising results on the training data but failed to make a good generalization based on it. Our custom ViT-model showed small signs of improvement during training but ultimately learned nothing and was no better than 50/50 chance. In addition, the pre-trained Google/vit-base-patch16-224-in21k ViT showed robustness in identifying patterns in the data, and sported an impressive accuracy. Not a lot of research has been conducted in the combined field of computer vision and archaeology, and these findings offer insight as to which technology should be employed for classification in a museum context.

1 Introduction

In the area of artifact curation and preservation, the integration of technology has become important for museums dealing with archaeological treasures. One demanding challenge faced by museums is the identification and classification of archaeological artifacts, some partially eroded. This is a task that demands efficiency and precision. This research will embark on classifying coins with varying conditions. The focus of the research lies in a comparative exploration of two capable image classification models; namely Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) regarding binary classification to determine whether or not an artifact uncovered by either professionals or amateur archaeologists is a coin.

Within museums, the significance of accurate classification, lies in its potential to ease the cataloguing and preservation process. As we dig into the details and nuances of these two specific models, we get inspiration from influential works in computer vision and image classification. Particularly, the work of Krizhevsky et al. ("ImageNet Classification with Deep Convolutional Neural Networks," 2012)[1] that was a front runner for the widespread use of convolutional neural networks in image classification problems, with demonstrations of the model's impressive ability to detect features from raw pixel data. This leap in deep learning and image classification has since had an influence on various domains, but the specific application to archaeological artifacts, eroded or not, remains a nearly unexplored area.

In a manner similar to the evolution of CNNs, the development of Vision Transformers, as introduced by Dosovitskiy et al. ("An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," 2020)[2], has begun to reshape the image processing landscape. Remarkable success has been demonstrated by ViTs in different visual recognition problems, and has begun to challenge the supremacy of the traditional convolutional architectures. [2] However, again, the application thereof to the domain of archaeological coin detection has yet to be examined and compared against the ingrained CNNs.

In this paper, we present a thorough evaluation of the performance of both CNNs and ViTs in identifying coins, some partially

[*]jsve22@student.aau.dk
[†]mcarls22@student.aau.dk
[‡]mguber22@student.aau.dk

eroded, within archaeological imagery. The methodology involves training both models on a curated dataset from the DIME database, comprising varied coin images, thus replicating the real-world challenges faced by museums. Through experimentation and analysis, we aim to bring insights that will both improve the accuracy of coin-detection and spearhead the development of solutions tailored to museums dealing with archaeological artifacts and their wear and tear.

2 Background and Theory

2.1 The Dataset

The dataset used in this study is sourced from the DIME database [3], that is publicly available. It is a comprehensive collection of archaeological artifact images. This database is comprised of contributions from both professional archaeologists and general enthusiasts, thereby covering a diverse array of archaeological findings uncovered in various conditions and contexts.
Comprising a total of approximately 300.000 images, the dataset is split into two primary categories; coins and other artifacts. More specifically, the coin category contains around 80.000 images, while the 223.000 remaining images are of diverse artifacts, beyond coins. That means that this dataset showcases a broad variety of relics and artifacts.

The DIME database images are quite diverse in terms of image quality and contextual information. They span several epochs and civilizations, offering a varied representation of archaeological findings. The diversity in the dataset enables the model to come across a wide array of coin visual characteristics, helping build robustness. The images were obtained using a multitude of devices, spanning from professional cameras to smartphones, giving the images varying resolutions, light conditions, and perspectives. The variance reflects the real-world scenarios faced by museums, enhancing the dataset's relevance for training a model to classify these artifacts. However, this also means that, comparing this dataset to established datasets used in machine learning, like CIFAR10 [4], ImageNet21K [5] etc. our dataset is not as reliable. The artifact in question is not always the largest object in our photos, sometimes more than one object for classification is present in the photos, and sometimes the archaeologists who uncovered the artifacts simply put down a wrong classification when uploading the data to the DIME-database, yielding label noise in the dataset. This incorrect labelling can distort the learning process of the models and it can lead the models to learn patterns based on incorrect information, also referred to as *garbage in, garbage out.*
in figure 1 are some examples sampled from the dataset. Each type of questionable data we have come across will have one image representing the problem.

(a) "Other" labeled as "Coin"

(b) "Coin" labeled as "Other"

(c) Multiple "Coins" in one image

(d) Multiple "Others" in one image

Figure 1: Examples of noise in the dataset

When using data to train a model to identify general features amongst objects of the same class, instances like the ones seen in Figure 1a and 1b, are obviously not useful. Just a few of these instances can have an impact on the final model. While figure 1c and 1d do in fact represent objects from the correct class, having multiple, even an arbitrary amount, of these objects, can also have an impact on the model. The dataset is meant to show *one* instance of every class per image, as to enable the model to generalize as well as possible.

The research is focused on specifically coin images, however, a substantial amount of the dataset - 210.000 images - are of non-coin artifacts. This imbalance in the dataset class distribution can skew model training, leading to biased predictions favoring the majority class, and hinder the model's ability to recognize the minority class.

The utilization of this rather expansive and diverse dataset from the DIME database will serve as the foundation in the evaluation and comparison of Convolutional Neural Networks and Vision Transformers in this specialized

task of identifying partially eroded coins among many archaeological artifacts. This will provide a practical benchmark for model comparison.

2.1.1 Data cleaning

Different techniques used to scale down images to take up less space which was proposed in both [2] and [6]. But the idea is also to prevent the overfitting of deep learning models [7]. Manual selection of two methods consists of horizontal flipping and vertical flipping using the Pytorch library [8].

For small datasets this approach can help introduce diversity into the dataset and help prevent overfitting the model to the said dataset. The wanted result is a more robost model on unseen data. Figure 2 shows an example usage of the selected augmentation techniques used in the experiments i.e. horizontal flipping and vertical flipping. The approach is based on some randomness with the probability.

Let P_{vertical} be the probability of a vertical flip, and $P_{\text{horizontal}}$ be the probability of a horizontal flip.

$$\text{RandomVerticalFlip}(P_{vertical} = 0.5) = \begin{cases} \text{VerticalFlip} & \text{with probability } 0.5 \\ \text{Identity} & \text{with probability } 0.5 \end{cases}$$

$$\text{RandomHorizontalFlip}(P_{horizontal} = 0.5) = \begin{cases} \text{HorizontalFlip} & \text{with probability } 0.5 \\ \text{Identity} & \text{with probability } 0.5 \end{cases}$$

The chance when loading the images is $\frac{1}{2}$ of being augmented either horizontal and vertically for each image or $\frac{1}{4}$ for being augmented both ways. This way randomness on image augmentation is introduced to enhance robustness of the models.

2.2 Convolutional Neural Network

Convolutional Neural Network (CNN, or ConvNet) is a deep-learning architecture that takes inspiration from biology by imitating the way natural visual perception manifests in living beings, as highlighted by Hubel & Wiesel. [9] Since its inception in 1990, the CNN has gone through many changes and improvements which propelled it to be a popular tool and research subject in recent years. [10]

O'Shea and Nash describe CNNs as being comprised of neurons that self-optimise through learning. [11] Their description is built on comparing CNNs to ANNs (Artificial Neural Networks). From this perspective, the neurons in the CNN can be observed to work in the same way as in ANNs, with each neuron receiving an input then performing some operation. Even though the input is an image in this scenario, the entirety of the network still produces a "single perceptive score function (the weight)". [11] O'Shea and Nash also make note of the fact that CNNs are mainly used for pattern recognition in images, which allows for the adoption of image-specific features into the architecture, which in turn enhance their image-related functionality and reduces setup complexity. [11]

Figure 3 shows a simple CNN architecture example, which highlights the main building blocks of a generic CNN. In this section we aim to provide an overview of the basic components of a CNN in an effort to illustrate their importance and functionality.

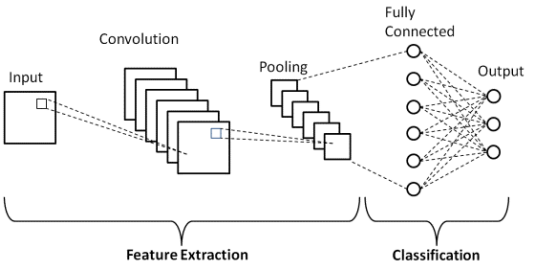


Figure 3: A simple Convolutional Neural Network Architecture.

Examining figure 3, we can observe that it consists of three types of layers, more precisely convolutional, pooling, and fully-connected layers. The purpose of the convolutional layer is to learn feature representations from the input. Such a layer is composed of multiple kernels that produce various feature maps. A receptive field is a group of neighbouring neurons, which are connected to the individual neurons of the feature map in the proceeding layer. [10] By convolving the input using a learned kernel and applying an activation function a new feature map can be obtained. The feature value at location (i, j) in the kth feature map of the lth layer, $z^l_{i,j,k}$ can be calculated:

$$z^l_{i,j,k} = {\mathbf{w}^l_k}^T \mathbf{x}^l_{i,j} + b^l_k \tag{1}$$

where $\mathbf{w}^l_k$ and b^l_k are the weight vector and bias term of the kth filter of the lth layer respectively, and $\mathbf{x}^l_{i,j}$ is the input patch with a center at location (i, j) of the lth layer. [10] Gu et al. note that the kernel $\mathbf{w}^l_k$ generating the shared feature map $\mathbf{z}^l_{:,:,k}$ is a weight sharing mechanism

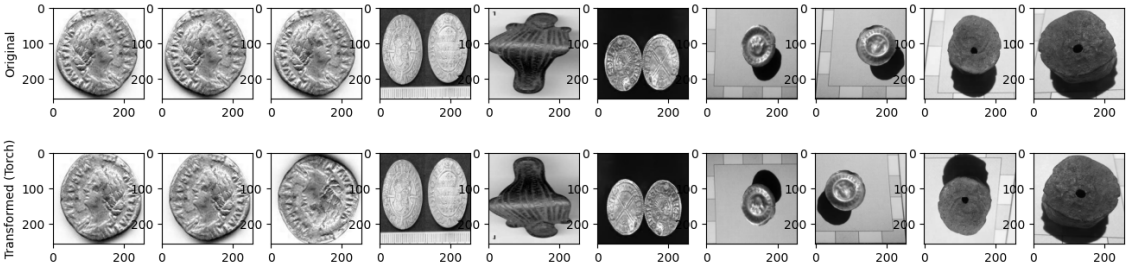


Figure 2: Example of Image Augmentation. Flipped horizontally or vertically.

that results in reduced model complexity and easier training.

Another component of CNNs is the activation function, which introduces nonlinearities allowing multi-layer networks to detect nonlinear functions. Sigmoid, tanh and ReLU are some of the most common activation functions used with CNNs. [10] With a$(\cdot)$ denoting the non-linear activation function, the activation value $a^l_{i,j,k}$ of convolutional feature $z^l_{i,j,k}$ can be calculated as:

$$a^l_{i,j,k} = \mathrm{a}(z^l_{i,j,k}) \tag{2}$$

Next up is the pooling layer, which is usually situated between two convolutional layers. Such a layer reduces the resolution of the feature maps in order to achieve shift-invariance. Every feature map in a pooling layer is directly connected to a corresponding feature map in the convolutional layer preceding it. Let pool$(\cdot)$ denote the pooling function, then for each feature map $\mathbf{a}^l_{:,:,k}$, we compute the corresponding pooled representation, $y^l_{i,j,k}$ as:

$$y^l_{i,j,k} = \mathrm{pool}(a^l_{m,n,k}), \forall (m,n) \in R_{ij} \tag{3}$$

where R_{ij} is a local neighborhood around location (i,j). Usually two types of pooling operations are differentiated, average pooling and max pooling. [10] Since the kernels in the first convolutional layer detect low-level features (edges, curves), and the kernels in higher convolutional layers encode more and more abstract features, we can stack alternating convolutional and pooling layers in an effort to extract gradually higher-level feature representations. [10]

The third type of layer to mention is the fully connected (FC) layer. Such a layer performs high-level reasoning by way of generating global semantic information. This is achieved by considering every neuron of the previous layer and connecting them to every single neuron of the current layer. According to Gu et al. [10] there can be several FC layers in a CNN or none at all, if replacement techniques are implemented such as 1x1 convolution or global average pooling. For traditional CNN models, the FC layer is an essential component and is widely used to this day. [12] However, it has been observed that FC increases the number of parameters in the model and may also contribute to overfitting, as such, recent CNN models (such as GoogLeNet and ResNet) have elected to replace the last FC layer with a global average pooling layer. [12]

The last layer in a CNN is the output layer. In order to solve classification tasks, most commonly the softmax operator is used, however other methods can also be combined with CNN features in an effort to adapt to different classification problems. [10] Gu et al. propose that obtaining the optimal parameters for a specific task means minimizing an appropriate loss function defined on said task. The loss of a CNN, L can be calculated:

$$L = \frac{1}{N} \sum_{n=1}^{N} l(\theta; \mathbf{y}^{(n)}, \mathbf{o}^{(n)}) \tag{4}$$

where θ denotes every parameter of a CNN, N is the number of desired input-output relations $\{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}); n \in [1, \ldots, N]\}$, with $\mathbf{x}^{(n)}$ being the n-th input data and $\mathbf{y}^{(n)}$ being its corresponding label and $\mathbf{o}^{(n)}$ being the output of the CNN. Along these lines Gu et al. [10] argue, that the training process of a CNN is a problem of global optimization, and by minimizing the loss function the best fitting parameters can be found. They mention stochastic gradient descent being a common solution for the optimization of CNNs. [10]

2.3 The Vision Transformer

The Vision Transformer (ViT) holds significance in the field of image classification. Originally designed as a sequence transduction model for Natural Language Processing [13], Google introduced a new architecture for image classification in their paper "An Image is Worth 16x16

Words" [2]. The ViT in figure 4 divides the input image into fixed-size non-overlapping patches, and each patch is then linearly embedded into a high-dimensional vector space (The dimensional space is a chosen fixed constant applied to each image patch).
To preserve the spatial relationships between image patches, positional encoding are added to the patch embedding. Different steps are taken before the actually encoder is applied:

- $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$: Input image tensor, where H is height, W is width, and C is the number of channels. In our case, $C = 3$ for RGB images, and we are using 256×256 images, thus $H, W = 256$. $\mathbf{X}$ represents the input image.
- $\mathbf{P} \in \mathbb{R}^{N \times D}$: Patch embeddings, obtained by dividing the image into non-overlapping patches are then linearly embedded denoted $\mathbf{P}$. D represents the embedded dimension for each image patch N, and D is a hyperparameter set by the user (The dimensional space). D represents the spatial information of each image patch $IP_{i,j}$, i.e. the information to store the relative information between every image patch.
- $\mathbf{E} \in \mathbb{R}^{N \times D}$: Positional embedding, preserving spatial relationships between patches. $\mathbf{E}$ corresponds to a positional embedding matrix with size $\mathbb{R}^{N \times D}$. These matrices are initialized as learnable parameters in our model. By this, zero matrices are employed as the initial values, allowing the ViT model to learn the appropriate positional information (i.e. positional information between each linear projected image patch $\mathbf{P}_i$) during training. $\mathbf{E}$ represents the matrix which is added with the embedded image patch $\mathbf{P}$.
- $\mathbf{EP} \in \mathbb{R}^{N \times D}$: Performing element wise addition of $\mathbf{P}$ and $\mathbf{E}$ along the D dimension, and then fed to the encoder part of the ViT model. $\mathbf{EP}$ is the result matrix.

This way the model can capture information about relative positions of different patches in the original image.
Compared to the traditional Transformer model presented in the paper "Attention Is All You Need" [14], the ViT only incorporates the encoder and neglects the decoder as a design choice, since the image input of the model is fully available and no need to decode a sequence to guess the next image patch.
Instead the decoder is replaced with a MLP (Multi-Layer Perceptron) Head network that outputs the prediction.

2.3.1 Class token

A class token is added to the patch embedding before encoding takes place. It's a learnable parameter added by prepending **Class** to the **EP** patch embedding matrix, as suggested in [2]. The class token **Class** is instantiated as a zero matrix $\mathbf{0} \in \mathbb{R}^{N \times D}$, and is learned throughout training.

2.4 The encoder

The encoder is the heart of the Vision Transformer and consists of multiple processing steps for the input feature. Referring to figure [4] to the right represents the encoder body. It consists of multiple normalization layers, using layer-normalization [15], a multi-head attention layer and a multi layer perception layer (MLP). The (+) symbol represents the concatenation operation. This body can be duplicated as represented by Lx. Layer normalization is suggested in "An Image is Worth 16x16 Words" [2], and used to stabilize the output of the MLP i.e. the layer normalization is applied to the output neurons of the MLP.

2.4.1 Encoder MLP

The MLP consists of a two layer Feed Forward Network with a single ReLU activation function. Layer Normalization $LN(z)$ is then applied to the output.

$$
\begin{aligned}
z_1 &= x \cdot W_1 + b_1 \\
a &= \mathrm{ReLU}(z_1) \\
z_2 &= a \cdot W_2 + b_2 \\
y &= \mathrm{LN}(z_2)
\end{aligned}
$$

x represents the input feature i.e. the normalized output of the multi-head attention layer. $\mathbf{W}_i$ represents weight matrices and $\mathbf{b}_i$ is bias matrices. Lastly, Layer normalization is applied to z_2. Before being processed by another encoder layer L_n, the results from the output neurons are concatenated.

2.4.2 Self-Attention Mechanism

The core of the ViT model as seen in figure 4 is the use of Multi-head attention. This allows the model to capture complex relationships between different image patches. It extends the self-attention mechanism used by the original transformer model [14].

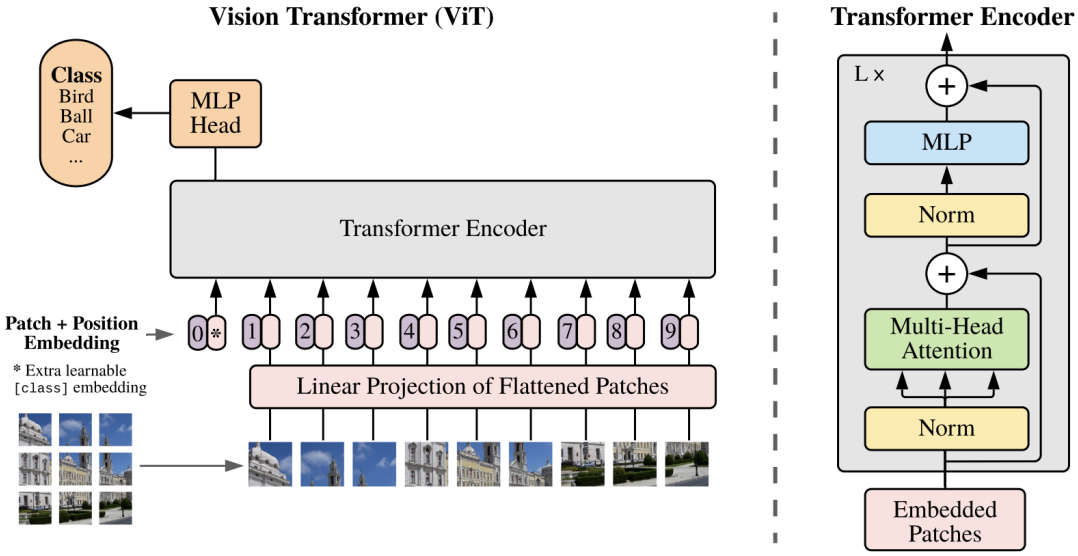


Figure 4: The Vision Transformer architecture from the original setup [2].

For $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times D}$,

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D}}\right)\mathbf{V}$$

The attention mechanism computes attention scores using the dot product of queries $\mathbf{Q}$ and keys $\mathbf{K}$, scaled by $\sqrt{D}$, followed by a softmax operation. The softmax function is used in the attention mechanism to convert the scaled dot product of queries $\mathbf{Q}$ and keys $\mathbf{K}$ into a set of attention scores. The result is added to the $\mathbf{V}$ matrix. The $\sqrt{D}$ is used to prevent high values of the dot product between $\mathbf{Q}$ and $\mathbf{K}$ and thereby stabilize the training process and prevent high gradients.

2.4.3 Multi-Head Self Attention

The self attention with multiple heads applies the Self-attention mechanism in parallel on the same image, and concatenates the results which is made to a dot product to a weight matrix $\mathbf{W_O}$. The input matrix $\mathbf{EP}$ is linearly projected to obtain query $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$ matrices for each attention head.
This means each input element $IP_{i,j}$ (image patch) is transformed three times to create the $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ matrices.

$$\text{MultiHead}(\mathbf{EP}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W_O}$$

The $head_i$ represent each individual self-attention head which is applied independently, and $\mathbf{W_O}$ represents the learnable weight matrix.

2.5 Classification Head

The classification head is the last part of the model which makes the prediction of the images. It consists of two linear layers with a single tanh activation function and then a sigmoid for normalization purposes.

$$\begin{aligned} z_1 &= x \cdot W_1 + b_1 \\ a &= \tanh(z_1) \\ z_2 &= a \cdot W_2 + b_2 \\ \hat{y} &= \sigma(z_2) \end{aligned}$$

The linear layers are represented by z_i which takes an input feature x, a W_k weight matrix and a bias b_n. The outut from the z_1 linear layer is the input for the $\tanh(z_1)$ activation function. The last linear layer z_2 is the final output which is the input to the sigmoid represented as $\sigma(z_2)$ and give the prediction $\hat{y}$.

2.6 Hybrid Architecture

In the work by Dosovitskiy et al. [2], a hybrid architecture combining elements of a Convolutional Neural Network (CNN) with the Vision Transformer (ViT) is introduced. This hybrid approach aims to leverage the strengths of both architectures to achieve effective image understanding.

The key components of the hybrid architecture are as follows:

1. Image Patching with Convolutional Layer:

The input image is initially processed using a convolutional layer. This layer extracts features from the image, treating each patch as a feature map. This step helps capture local information and spatial hierarchies inherent in images.

2. Linear Projection of Feature Maps:
Each patch, now represented as a feature map, undergoes linear projection to transform the spatial information into a high-dimensional vector space. This linear projection is akin to the process employed in the traditional Vision Transformer patch embedding.

3. Similar Performance to Patch Embedding:
The hybrid architecture yields comparable results to the standard patch embedding approach. This suggests that incorporating convolutional layers in the initial stages of processing does not compromise the model's ability to capture relevant visual information.

The hybrid architecture addresses the question of whether the inherent spatial hierarchies captured by convolutional layers can be effectively integrated with the self-attention mechanism of a ViT. This approach opens avenues for exploring the synergy between convolutional and transformer-based architectures for image analysis tasks.

It's worth noting that the success of the hybrid architecture underscores the flexibility and adaptability of ViT models, allowing researchers to explore innovative combinations to enhance performance on various tasks.

2.7 Binary Cross Entropy

We have chosen the Binary Cross Entropy [16] as our loss function due to its fit for binary class classification tasks.

$$\ell(x, y) = L = \begin{bmatrix} l_1 \\ \vdots \\ l_N \end{bmatrix}^T,$$

$$l_n = -w_n \left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n) \right]$$

y_n represents the target label, i.e., the truth value the model tries to predict for the n-th example. Similarly, x_n represents the predicted value or output generated by the model for the n-th example.

3 Experiment

In this section we outline how the fine-tuning of the hyperparameters for each model was conducted, which configuration was selected for each model; along with a breakdown of the architecture for each model. The design of the experiments will be described, and finally, the results of the experiment and the comparison between the models are presented.

3.1 Hyperparameter tuning using grid search

In pursuit of the best possible performance of both our CNN and our ViT, hyperparameter tuning was conducted using the grid search technique [17]. This technique is systematic, such that is explores pre-defined hyperparameter combinations within a specified range in order to get an idea of which configuration has the optimal performance for model training.

In the two configurations, we see the search space of the grid search, where the numbers marked in bold font is the value that was selected for each of the hyperparameters. The uniform and loguniform are values sampled between the first two values in the parenthesis, and the last number, in bold, is the selected value.

Configuration for the Vision Transformer grid search:

lr:	loguniform($10^{-4}, 10^{-1}, \mathbf{0.001}$),
batch_size:	randint($\mathbf{4}, 8$),
weight_decay:	loguniform($10^{-3}, 1, \mathbf{0.004}$),
nEpochs:	grid_search($\mathbf{10}, 20$),
d_model:	grid_search($256, \mathbf{512}$),
n_heads:	grid_search($1, 2, \mathbf{4}$),
dim_feedforward:	grid_search($\mathbf{256}, 512$),
dropout:	uniform($0.1, 0.5, \mathbf{0.35}$),
n_layers:	grid_search($\mathbf{1}, 2, 4$),
patch_size:	grid_search($\mathbf{8}, 16$),
n_hidden:	grid_search($\mathbf{1}, 2, 4$),
activation:	grid_search("gelu", "**relu**"),
threshold:	uniform($0.1, 0.9, \mathbf{0.5}$)

Configuration for the Convolutional Neural

Network grid search:

threshold:	uniform$(0.1, 0.9, \mathbf{0.45})$,
nEpochs:	grid_search$(10, 15, \mathbf{20})$,
batch_size:	randint$(8, 32, \mathbf{23})$,
out_channels_1	grid_search$(10, \mathbf{20}, 30)$,
out_channels_2	grid_search$(\mathbf{10}, 20, 30)$,
padding_conv:	grid_search$(\mathbf{0}, 1, 2)$,
stride_conv:	grid_search$(\mathbf{1}, 2, 3)$,
stride_pool:	grid_search$(\mathbf{3}, 4, 5])$,
kernelSize_conv:	grid_search$(3, 4, \mathbf{5})$,
kernelSize_pool:	grid_search$(2, 3, \mathbf{4})$,
lr:	loguniform$(10^{-4}, 10^{-1}, \mathbf{0.003})$,
ffn_out_features:	grid_search$(300, \mathbf{500}, 600)$,
weight_decay:	loguniform$(10^{-3}, 1, \mathbf{0.003})$

3.1.1 Methodology for finding the optimal model

For conducting the grid search, a hyperparameter tuning library, called Ray [18], was used. This library runs the searches in parallel, leaving not much computer power for each search. This forced us to drastically lower the size of the dataset on which the search was conducted; namely 100 images from each class for each search, repeated *nEpochs* times, taken from the search space. All training has been done with a 70%/15%/15% split for training-, validation-, and test datasets.

- **Grid Definition:** The predefined grid of hyperparameters specifies a certain range for each parameter. Every *grid_search* parameter, e.g., *nEpochs*, means that every value in the list will be exhaustively tested along with every combination of possible values from other *grid_search* parameters, while every *unform* or *loguniform* parameter specifies a range from which each value is equally likely to be sampled. This combined forms a multidimensional grid that is searched for potential configuratrions
- **Model Training and Evaluation:** For each hyperparameter combination in the grid, a model is trained using the training set and validated on the validation set. For each configuration, we are calculating the metrics as seen in 3.1 to ensure proper model validation.
- **Selection of Optimal Hyperparameters:** The grid search determines the hyperparameter combination that produces the best output, measured on the *accuracy* performance metric. This configuration is then a representation of the model with the best performance that could be made under the specific hyperparameter search space. This model is selected for further evaluation and comparison.

In this study, grid search is the methodology used for fine tuning the hyperparameters for each of our models. The optimal configurations, measured on the highest on accuracy achieved, will form the basis for subsequent evaluation and comparison in the pursuit of the most effective model when in comes to the specialized task of identifying coins within archaeological artifacts.

3.2 Selected models

Here, an overview of the three different models up for comparison, each along with the output of the model summary, showcasing, precisely, the complexity of each model, is given. For each model we showcase the following: The amount of trainable parameters, total mult-adds; roughly the amount of memory it takes to train the model and conduct multiplication and addition operations, the input-size in megabytes; how much memory is required for holding the input tensor, forward/backward pass size in megabytes; roughly how much memory is required to hold intermediate activations and tensors in the forward pass and gradients for the parameters in the backward pass, the size of the parameters in megabytes; roughly how much memory is needed to hold the weights and biases of the model parameters, and the estimated total size of the model in megabytes; roughly the memory it takes to hold the entire model.

3.2.1 CNN

Taking the CNN configuration that achieved the highest accuracy score during the grid search resulted in the architecture summarized in Appendix figure A1. The input shape of the model is [23,3,256,256] where 23 is the size of the batches, 3 is the color channels, and 256 x 256 describes the resolution of the images. The overall output has the shape [23,1], where 23 is, again, the batch size, and 1 is the final classification value. The architecture consists of six layers in total, two sets of convolutional-pooling layer pairs followed by two fully connected layers.

The first convolutional layer (conv1) uses (5,5) kernels with a stride of 1 and valid padding. The hyperparameter tuning has determined that the number of channels this layer outputs should be 20, each channel representing an extracted feature. Thus, the output of this layer takes the shape [23,20,252,252], where 252 x 252 is the size of the feature maps as a result of the convolution.

The next layer is the first pooling layer (pool1) in the model. It performs max pooling using a (4,4) kernel and a stride of 3. This operation reduces the size of the feature maps, which can be seen on the shape of the output [23,2,83,83].

The following is the second convolutional layer (conv2), with a (3,3) kernel, a stride of 2, and once again valid padding. This layer produces 10 feature maps. Due to the stride value being higher, the reduction in size is more significant, which the output shape [23,10,41,41] illustrates well.

Next is the second pooling layer (pool2), which also performs max pooling using a (4,4) kernel and a stride of 3. The feature map size reduction results in the output shape [23,10,13,13].

The last two layers are two fully connected layers (fc1 and fc2). Fc1 is constructed by flattening each feature map into a one dimensional vector, with size $(10 * (13 * 13) = 1690$, which acts as its input. The output vector is derived from the configuration obtained during hyperparameter tuning, namely a size of 500. Therefore, the input vector for fc2 has size of 500, whereas its output is size 1.

ReLU is used as the activation function on the output of conv1, conv2, and fc1. During training, an optimizer called Adam is used, with the learning rate set at 0.003 as per the results of the hyperparameter tuning process. As seen in Appendix figure A1, the total number of parameters in this model is a relatively low amount at 849,331, all of which are trainable. On the other hand, the total number of multiply-accumulate operations is relatively high at 2.31 billion, which suggests a moderate computational requirement. The model achieves a high computational workload despite a relatively small number of parameters. This can suggest the efficient use of parameters due to the layer configuration and the overall architectural design. However, it is worth noting that the input size for the model also affects the total number of mult-adds, with higher sizes resulting in larger computational demands.

3.2.2 Vision Transformer From Scratch

The ViT that we built from scratch breaks away from the conventional architectures based on convolutions by instead using self-attention mechanisms to capture global features and relations between patches of an image. Constructed from scratch, the ViT architecture holds attention mechanisms, patch embeddings, positional encodings and feed-forward networks, where most of the complexity lies in the attention mechanisms.

In Appendix Figure A2, we get an architecture overview of our ViT model. We see the input shape being [4,3,256,256] which means we have a batch size of 4, 3 color channels and images of size 256 x 256 pixels. Next we see the *patchEmbeddings*, responsible for converting the input image into patches and projecting them into a high-dimensional space, and with a patch size of 8, we get an output of 1024, since we can fit 1024 patches of size 8 x 8 into one 256 x 256 image, and then the *positionalEmbeddings* introduces positional information for these patches. After the patches have been linked with position, the *transformerencoder* and subsequent *transformerencoderlayer* take as input the 1024 patches + the 1 class token. In the transformerencoder layers, the multi-head self-attention mechanism lies. The *LayerNorm* supplies layer normalization to the final output of the transofrmer encoder, and the last step of the model is the *ClassificationHead*, that is responsible for classification.

3.2.3 Pre-trained Google/vit-base-patch16-224-in21k ViT

This pre-trained model is well known in the evolution of models trained for image classification. It is pre-trained on an enormous dataset that includes 21.843 classes and more than 14 million images, *ImageNet21k*,[19] and it was first introduced in the paper *An image is worth 16x16 words: Transformers for image recognition at scale.*

[2] By pre-training this model, it learns an inner representation of images, and this can be used to make feature extractions that can be utilized in downstream tasks. With our labelled binary dataset, we can then further train this model to incorporate our dataset.

The architecture of this model, seen in Appendix Figure A3 is very similar to the architecture seen in Appendix Figure A2, with an exception of the *Dropout* [20], that randomly selects neurons in the network that is then not used during training, in an attempt to avoid overfitting.

3.3 Experiment Design

The comparative study between the optimized CNN and ViT, and the pre-trained ViT, is structured to evaluate their performance across multiple metrics. The experiment is a systematic comparison between the best-performing model of the respective architectures. The comparison between the three models will be made tangible through a suite of metrics to assess the efficiency. These are listed below:

- Recall: Evaluate the models' capability to classify specifically the coins among the whole dataset. Recall measures the amount of true positive identifications. In this case, how many actual coins were predicted as being coins.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- Accuracy: Determine overall correctness in coin detection. Accuracy quantifies the ratio of coin images and non-coin images predicted correctly by the model

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Population}}$$

- Precision: Like accuracy, precision determines correctness in coin detection, but only coin detection. Precision is a measure of how often the model is correct in predicting the target class (coin)

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- F1-score: This is a balanced measure between two of the above metrics; precision and recall. The F1-score is relevant in the case of unbalanced datasets, as it accounts for the trade-off between precision and recall.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Average Confusion Matrix: An average confusion matrix aggregates the true positives, true negatives, false positives, and false negatives across multiple runs or subsets of data.

$$\begin{bmatrix} \text{True Negative} & \text{False Positive} \\ \text{False Negative} & \text{True Positive} \end{bmatrix}$$

  In our case, true positives are coins predicted to be coins, false positives are others predicted to be coins, true negatives are others predicted to be others, and false negatives are coins predicted to be others.

- Area Under ROC-curve (AUC-ROC): This value, between 0 and 1, illustrates the models' capability to differentiate between coin and non-coin images across various thresholds

- Time to Train: Simply reflecting on computational efficiency; the time it takes to train each model, brings insight into the resource requirement and scalability of the models

- Model Complexity: This considers the inner workings of the models, such as the number of layers and parameters.

After establishing the best-performing versions of the CNN and the ViT, based on the grid search, the models are subjected to evaluation using the aforementioned metrics, and the comparative analysis is made with a side-by-side assessment, comparing their performances across these dimensions. The focus is not only on predictive accuracy but also on complexity and computational efficiency. Focusing on both aspects of the models ensure a broad understanding of the strengths and weaknesses in both model architectures, for this specialised task. The information gained from this analysis will make it easier to make a decision in the deployment of these models in a real-world museum setting. This will make it clear, based on performance and computational requirements, which is more suitable for applications in artifact classification.

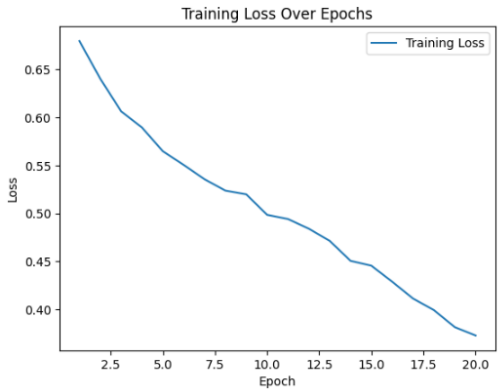
3.4 Results

This section describes the results of our comparative analysis between the Convolutional Neural Network (CNN) and Vision Transformers (ViTs) in detecting eroded coins within archaeological imagery. It highlights each model's performance metrics, computational efficiency, and complexity. These insights steer us toward identifying the most effective models for museum coin classification. Each model has been trained on 5.000 images from each class, with a training-, validation-, and test set split of 70%,15%,15%

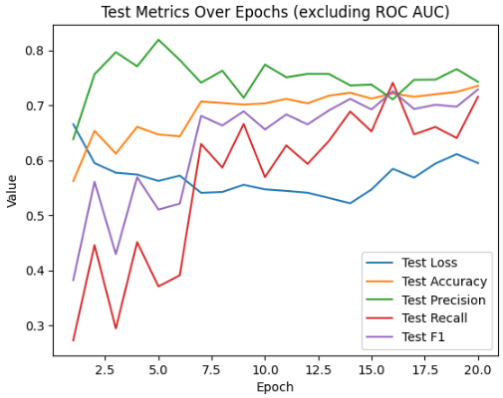
3.4.1 CNN

This first section showcases the results from the training and testing of our CNN, which took on average 21 seconds per epoch to train, validate, and test.

It is evident in Figure 5a that the loss value decreases over a relatively low number of epochs, signifying that the model is learning and improving on its ability to make predictions. We



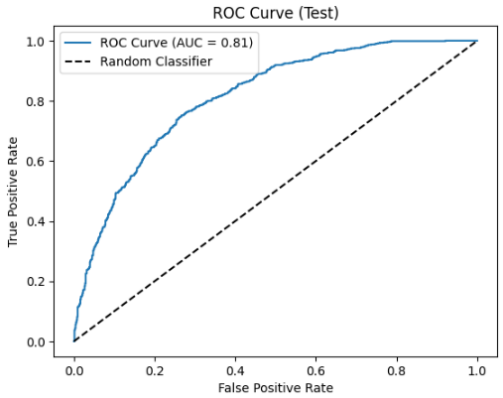
(a) Training loss for CNN



(b) Test metrics for CNN

Average Confusion Matrix on Test Set			
TARGET / OUTPUT	Other	Coin	SUM
Other	613.20 40.87%	141.80 9.40%	754 81.30% 18.70%
Coin	325 21.67%	421 28.07%	746 56.43% 43.57%
SUM	938 65.35% 34.65%	562 74.91% 25.09%	1034 / 1500 68.93% 31.07%

(c) Test confusion matrix for CNN



(d) ROC for CNN

Figure 5: CNN metrics

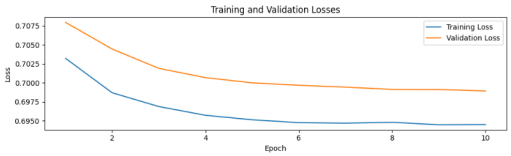
can see that the model is minimizing the difference between the predicted output and the actual labels in the training data. The reduction in loss value from 0.7 to 0.4 might seem small, but it signifies that the model is gradually improving its prediction over epochs, and the reduction seems consistent. Sometimes, a continuous decrease in training loss may indicate over-fitting, where the model becomes too specialized in the training data and performs poorly on new, unseen data. We see the training loss decreasing while the test loss remains relatively steady, as seen in Figure 5b, indicating over-fitting. The fact that the test loss value is steady at around 0.65 to 0.55 suggests that the performance of the model, on unseen data, does not see significant improvement over 20 epochs. This could indicate that the model has not generalized well beyond the training data.

Taking a look at figure 5d, we see that the model possesses quite good performance in predictions, and it suggests that the model's ability to differentiate the two classes is moderate. An AUC of 0.81 means that the model is much better than random chance (AUC 0.5) but it does not distinguish between the classes exceptionally well. In the context of classifying coins in a museum setting, getting some wrong answers is not as significant as it might be in some other domains. It does, however, suggest that there is room for improvement.

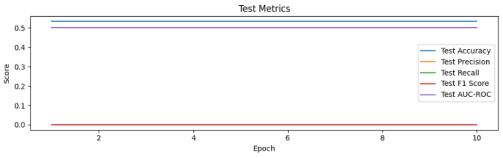
In Figure 5c, we can see that the model has a slight tendency to predict one label over the other, namely that the images are not coins. This indicates that the model has not been able to generalize exactly what the images containing coins look like. However, 1.034 out of 1.500 are true positives or true negatives, meaning that the model has been able to generalize to some extent, beyond the training data.

3.4.2 ViT

Below are the diagrams capturing the performance of the ViT we made from scratch, which took on average 59 seconds per epoch to train, validate, and test



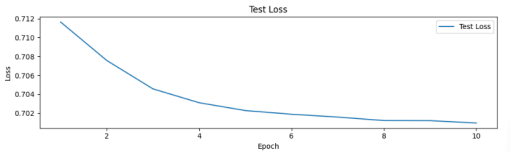
(a) Training and validation loss for ViT



(b) Test metrics for ViT

Average Confusion Matrix on Test Set			
TARGET / OUTPUT	Other	Coin	SUM
Other	769 51.27%	0 0.00%	769 100.00% 0.00%
Coin	731 48.73%	0 0.00%	731 0.00% 100.00%
SUM	1500 51.27% 48.73%	0 NaN% NaN%	769 / 1500 51.27% 48.73%

(c) Test confusion matrix for ViT



(d) Test loss for ViT

Figure 6: ViT metrics

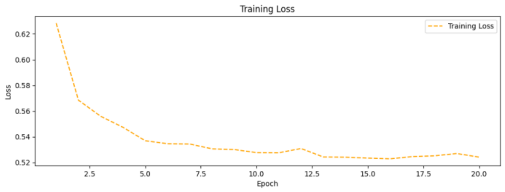
Even though, as in Figure 5a, we can see the loss value decrease over epochs in training in Figure 6a, we do not see the model learning anything. Looking back at Appendix Figure A2 we notice that the model has more than 3.000.000 parameters, and given the limited computer power and limited dataset, it seems that the model complexity and the training efforts are not in harmony. The models seems to be overly complex for the task and the available resources. The model exhibits behaviour where the loss slightly decreases but it still fails to change its predictions. This might come from the lack of data, and that the model still focuses on noise or irrelevant patterns in the data, resulting in a bad generalization. To address this issue in the future, we would need to host the model on a machine that has the power to perform the training with every image we have available, and run for a considerable amount of epochs and if that is not enough we can be fairly certain that this type of model requires a substantial amount of data to be trained.

Looking at the remaining figures, 6c, 6d, and 6b, it becomes more evident that the model has, in fact, not learned anything or been able to make a generalization based on the training. We still see a slight decrease in the test loss, but we see that every single metric measured during testing is completely flat and we have an AUC at 0.5, which is effectively the same as a random classifier.

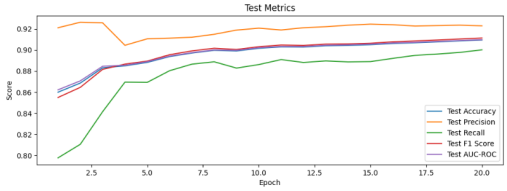
3.4.3 Pre-trained ViT

This last result section showcases the performance of the pre-trained Google/vit-base-patch16-224-in21k model during training and testing, which took on average 220 seconds per epoch to train, validate, and test.

In figure 7a we see a decrease in loss from 0.62 to 0.52 over 20 epochs, indicating that the model is learning and improving its performance during training and it is evident that the model is converging towards a better solution and is gradually minimizing errors in prediction on the training data. Comparing this with what we see in figure 7a, where there is a decrease in loss from 0.58 to 0.54, we see an indication of some level of generalization. This reduction in loss, albeit smaller than the decrease on the training set, suggests that the model seems to be learning patterns that extend beyond the training data, and it implies that there is some ability to generalize on unseen examples. The comparatively smaller reduction in loss from training to test can also suggest over-fitting to some degree. The model might be fitting training data in good fashion but struggles to generalize optimally on the test dataset.
The test metrics gathered from the pre-trained ViT, as seen in figure 7b, with generally high val-



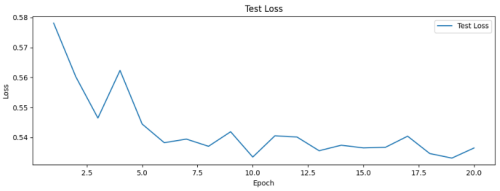
(a) Training loss for pre-trained ViT



(b) Test metrics for pre-trained ViT

Average Confusion Matrix on Test Set			
TARGET / OUTPUT	Other	Coin	SUM
Other	658.20 43.87%	71.80 4.73%	729 90.26% 9.74%
Coin	82.50 5.47%	689.50 45.93%	771 89.36% 10.64%
SUM	740 88.92% 11.08%	760 90.66% 9.34%	1347 / 1500 89.80% 10.20%

(c) Test confusion matrix for pre-trained ViT



(d) Test loss for pre-trained ViT

Figure 7: pre-trained ViT metrics

ues, signifies a rather robust model; which makes sense, seeing as it has been pre-trained with millions of images and thousands of classes. The high accuracy, as opposed to the ViT we made from scratch, means that this model has the ability to identify objects from both of our classes, and is not limited to predicting one class over another for the majority of the predictions. The values for precision and recall indicates that the model has a good ability to identify coins, and the high AUC value indicates that the model is able to distinguish the coins from other artifacts with high accuracy. Overall, these metrics collectively suggest positive progress in the model's performance across multiple dimensions.
Finally, taking a look at figure 7c, we can see that the cells representing true positives (TP) and true negatives (TN) have far larger values than the cells representing false positives (FP) and false negatives (FN). This, again, means that the model seems to have a good generalization of the data, and performs well on identifying both the positive instance; coins, and the negative instance; non-coins.

3.4.4 Summary

The main takeaway from these results is the stark difference in performance between the ViT that we made from scratch, and the more complex ViT, pretrained on ImageNet21k. The ViT made from scratch learned nothing, while the pre-trained model performed really well. One thing to notice about the pretrained model is the slight contrast in the relatively high loss on the test set (around 0.5), and the high accuracy on the test set (nearing 0.9). This can be caused by different things, like the threshold being too high or too low, where far from the threshold, the model has correct predictions, but at one side, close to the threshold, most predictions might be wrong. In order to definitively say what is causing this, we would have to conduct deeper analysis on exactly when the model makes correct and incorrect predictions. Worth mentioning is the performance of the CNN, which sports a good AUC ROC value, and a noticeable decrease in the loss value on training, despite the small amount of epochs and limited data.

4 Concluding Remarks

The results of our experiments line up with our expectations based on our pre-existing knowledge of the domain and preliminary research results. The biggest unexpected result is the magnitude of difference between the performance of the pre-trained ViT and our other two models, and the circumstances surrounding the process of hyperparameter tuning.

Metrics	CNN	Coin ViT	Pre-trained ViT
Recall	0.71	0.00	0.92
Accuracy	0.73	0.51	0.90
Precision	0.74	0.00	0.89
F1	0.72	0.00	0.91
Correct predictions	68.93%	51.27%	89.80%
AUC-ROC	0.81	0.50	0.90
Time to train	420 s	1190 s	4409 s
Number of params.	0.85 mil	3.26 mil	85.80 mil
Mult-adds	2.31 bil	778 mil	2.01 bil

Table 1: A summary of the experiment results after 20 epochs

Table 1 illustrates a summary of the key metrics collected during the experiments. As we can see, the custom ViT created by our group exhibits a rather peculiar behaviour pointing to the model not learning throughout the training process. This results in the model essentially classifying every image as 'not a coin', which ultimately renders the recall, precision, and F1 scores zero. While the CNN model achieves respectable scores after the 20 epoch training, the pre-trained ViT runs laps around it with roughly a 0.20 increase in most metrics. An important note to make is that the CNN took significantly less time to train at just 420 seconds, while the 20 epoch fine-tuning of the pre-trained ViT on our dataset took around 4409 seconds. It is worth mentioning, however, that getting the CNN to the training process required a significant time and effort in the form of conducting the hyperparameter search, while the pre-trained ViT can be fine-tuned right out of the box. Therefore, it can be concluded that even though the actual training time was insignificant for the CNN, utilizing the pre-trained ViT for the task of classifying coins is the desirable approach.

An unexpected difficulty was posed by conducting the hyperparameter tuning for the two models built by us. While training the models did not pose a challenge in terms of time and computational power requirements, that certainly wasn't the case with the tuning process. This can partly be assigned to the search method that was used, grid search, since introducing new parameters grows the search space exponentially. Ray tune, the tuning framework we used, severely limited the amount of images we could use for training. The results of this research could be enhanced by dedicating time to refactor the tuning framework in a manner that allows for a significantly larger pool of images to be used for training. Furthermore, other search methods could be researched and considered for use, such as random search, Bayesian optimization, and evolutionary algorithms.

Additionally, we have learned that the quality of the data we have worked with, extracted from the DIME database, contains a relatively large amount of inaccuracies. These manifested in the form of images being assigned incorrect labels, multiple artifacts being present in one image, foreign objects present in the images (eg. a human hand holding the object in question), and the subject of the images taking up only an insignificant part of the whole image. All these characteristics make it difficult for the model to learn consistently, warranting an increased effort directed at data cleaning for future projects that aim to utilize data from the DIME database. This was explained by the Queen of Denmark in her speech on New Year's Eve, 2023: "Den nye teknologi er jo netop "kunstig". Den tænker ikke selv. Den bliver fodret med det, mennesker allerede har skabt." [21]

A significant limitation encountered during this project was the lack of sufficient computational power and storage space. This limitation reared its head most prominently during the hyperparameter tuning process, significantly limiting the amount of input images we could utilize. Streamlining this process with the use of additional computing power would lead to a more robust process, which would allow for more values to be tested as well as increased confidence in the optimal configuration selected.

Reflecting upon the development process of the CNN and the ViT that was made from scratch, out of the two, the CNN was by far the simplest to implement and understand. With this in mind, and the fact that the CNN was much more receptive to training with our data, the CNN would be the obvious choice in a real-world-application where we would need to make and train our own model from scratch.

The results learnt from this paper can be expanded upon in the future by focusing on training on a larger set of data with the help of more powerful and plentiful computational resources.

The objects of classification can also be widened to include a variety of different artifacts. An interesting avenue to explore is the reconstruction of artifacts based on patterns learned through images of incomplete artifacts.

While writing ones own implementation of a CNN or ViT is certainly possible, for a task like classifying coin images in an archaeological database this approach does not pay off. Utilizing a pre-trained model does not only yield better performance, it requires significantly less investment leaving more resources for fine-tuning the model and clearing the data.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (visited on 01/09/2024).

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021. arXiv: 2010.11929 [cs.CV]. (visited on 11/05/2023).

[3] "Dime metal detektor fund." (2022), [Online]. Available: https://www.metaldetektorfund.dk/ny/ (visited on 11/25/2023).

[4] V. N. Alex Krizhevsky and G. Hinton, *Cifar-10*, https://www.cs.toronto.edu/ kriz/cifar.html. (visited on 01/07/2024).

[5] T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor, "Imagenet-21k pretraining for the masses," *arXiv preprint arXiv:2104.10972*, 2021. (visited on 01/09/2024).

[6] K. O'Shea and R. Nash, *An introduction to convolutional neural networks*, 2015. arXiv: 1511.08458 [cs.NE].

[7] S. Yang, W. Xiao, M. Zhang, S. Guo, J. Zhao, and F. Shen, *Image data augmentation for deep learning: A survey*, 2023. arXiv: 2204.08610 [cs.CV].

[8] *Transforming and augmenting images*, https://pytorch.org/vision/main/transforms.html. (visited on 12/28/2023).

[9] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, no. 1, pp. 215–243, 1968. DOI: https://doi.org/10.1113/jphysiol.1968.sp008455. eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1968.sp008455. [Online]. Available: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1968.sp008455.

[10] J. Gu, Z. Wang, J. Kuen, *et al.*, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.10.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320317304120.

[11] K. O'Shea and R. Nash, *An introduction to convolutional neural networks*, 2015. arXiv: 1511.08458 [cs.NE]. (visited on 01/04/2024).

[12] C.-L. Zhang, J.-H. Luo, X.-S. Wei, and J. Wu, "In defense of fully connected layers in visual representation transfer," in *Pacific Rim Conference on Multimedia*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:1138809.

[13] A. Graves, *Sequence transduction with recurrent neural networks*, 2012. arXiv: 1211.3711 [cs.NE].

[14] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. (visited on 11/05/2023).

[15] P. Contributors. "Torch.nn.layernorm." Accessed on: January 11, 2024. (2024), [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html.

[16] "Bceloss." (), [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html (visited on 01/05/2024).

[17] P. Liashchynskyi and P. Liashchynskyi, *Grid search, random search, genetic algorithm: A big comparison for nas*, 2019. arXiv: 1912.06059 [cs.LG].

[18] R. Project. "Ray tune documentation." Accessed: January 12, 2024. (2022), [Online]. Available: https://docs.ray.io/en/latest/tune/index.html.

[19] HuggingFace, *Google/vit-base-patch16-224-in21k*, https://huggingface.co/google/vit-base-patch16-224-in21k. (visited on 12/27/2023).

[20] S. Weidman, *Deep Learning from Scratch: Building with Python from First Principles*. O'Reilly Media, 2022, ISBN: 978-1492041412.

[21] Q. M. I. of Denmark, *Hendes majestæt dronningens nytårstale 2023*, 2023. [Online]. Available: https://www.kongehuset.dk/nyheder/laes-h-m-dronningens-nytaarstale-2023 (visited on 01/11/2024).

A Figures

```
==========================================================================================================================================================
Layer (type:depth-idx)                   Input Shape               Param #                   Output Shape              Mult-Adds                 Trainable
==========================================================================================================================================================
CoinClassifierFirst                      [23, 3, 256, 256]         --                        [23, 1]                   --                        True
├─Conv2d: 1-1                            [23, 3, 256, 256]         1,520                     [23, 20, 252, 252]        2,220,099,840             True
│    └─weight                                                      ├─1,500
│    └─bias                                                        └─20
├─MaxPool2d: 1-2                         [23, 20, 252, 252]        --                        [23, 20, 83, 83]          --                        --
├─Conv2d: 1-3                            [23, 20, 83, 83]          1,810                     [23, 10, 41, 41]          69,980,030                True
│    └─weight                                                      ├─1,800
│    └─bias                                                        └─10
├─MaxPool2d: 1-4                         [23, 10, 41, 41]          --                        [23, 10, 13, 13]          --                        --
├─Linear: 1-5                            [23, 1690]                845,500                   [23, 500]                 19,446,500                True
│    └─weight                                                      ├─845,000
│    └─bias                                                        └─500
├─Linear: 1-6                            [23, 500]                 501                       [23, 1]                   11,523                    True
│    └─weight                                                      ├─500
│    └─bias                                                        └─1
==========================================================================================================================================================
Total params: 849,331
Trainable params: 849,331
Non-trainable params: 0
Total mult-adds (G): 2.31
==========================================================================================================================================================
Input size (MB): 18.09
Forward/backward pass size (MB): 236.88
Params size (MB): 3.40
Estimated Total Size (MB): 258.37
==========================================================================================================================================================
```

Figure A1: CNN model summary

```
==========================================================================================================================================================
==============================================
Layer (type:depth-idx)                             Input Shape               Param #                   Output Shape
Mult-Adds                 Trainable
==========================================================================================================================================================
==============================================
VisionTransformer                                  [4, 3, 256, 256]          1,316,096                 [4, 1]
--                        True
├─PatchEmbeddings: 1-1                             [4, 3, 256, 256]          --                        [1024, 4, 512]
--                        True
│    └─Conv2d: 2-1                                 [4, 3, 256, 256]          98,816                    [4, 512, 32, 32]
404,750,336               True
├─PositionalEmbedding: 1-2                         [1025, 4, 512]            524,800                   [1025, 4, 512]
--                        True
├─TransformerEncoder: 1-3                          [1025, 4, 512]            --                        [1025, 4, 512]
--                        True
│    └─ModuleList: 2-2                             --                        --                        --
--                        True
│    │    └─TransformerEncoderLayer: 3-1           [1025, 4, 512]            1,315,584                 [1025, 4, 512]
271,584,000               True
│    └─LayerNorm: 2-3                              [1025, 4, 512]            1,024                     [1025, 4, 512]
1,049,600                 True
├─LayerNorm: 1-4                                   [4, 512]                  1,024                     [4, 512]
4,096                     True
├─ClassificationHead: 1-5                          [4, 512]                  --                        [4, 1]
--                        True
│    └─Linear: 2-4                                 [4, 512]                  513                       [4, 1]
2,052                     True
│    └─Tanh: 2-5                                   [4, 1]                    --                        [4, 1]
--                        --
│    └─Linear: 2-6                                 [4, 1]                    2                         [4, 1]
8                         True
│    └─Sigmoid: 2-7                                [4, 1]                    --                        [4, 1]
--                        --
==========================================================================================================================================================
==============================================
Total params: 3,257,859
Trainable params: 3,257,859
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 677.39
==========================================================================================================================================================
==============================================
Input size (MB): 3.15
Forward/backward pass size (MB): 109.16
Params size (MB): 3.56
Estimated Total Size (MB): 115.87
==========================================================================================================================================================
```

Figure A2: ViT model summary

```
==========================================================================================================================================
==========================================================
Layer (type:depth-idx)                                  Input Shape               Param #                   Output Shape
Mult-Adds                 Trainable
==========================================================================================================================================
==========================================================
ViTForImageClassification                               [10, 3, 224, 224]         --                        [10, 2]
--                        True
├─ViTModel: 1-1                                         [10, 3, 224, 224]         --                        [10, 197, 768]
--                        True
|    └─ViTEmbeddings: 2-1                               [10, 3, 224, 224]         152,064                   [10, 197, 768]
--                        True
|    |    └─ViTPatchEmbeddings: 3-1                     [10, 3, 224, 224]         590,592                   [10, 196, 768]
1,157,560,320             True
|    |    └─Dropout: 3-2                                [10, 197, 768]            --                        [10, 197, 768]
--                        --
|    └─ViTEncoder: 2-2                                  [10, 197, 768]            --                        [10, 197, 768]
--                        True
|    |    └─ModuleList: 3-3                             --                        85,054,464                --
--                        True
|    └─LayerNorm: 2-3                                   [10, 197, 768]            1,536                     [10, 197, 768]
15,360                    True
├─Linear: 1-2                                           [10, 768]                 1,538                     [10, 2]
15,380                    True
==========================================================================================================================================
==========================================================
Total params: 85,800,194
Trainable params: 85,800,194
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 2.01
==========================================================================================================================================
==========================================================
Input size (MB): 6.02
Forward/backward pass size (MB): 1621.83
Params size (MB): 342.59
Estimated Total Size (MB): 1970.45
==========================================================================================================================================
==========================================================
```

Figure A3: Pre-trained ViT model summary