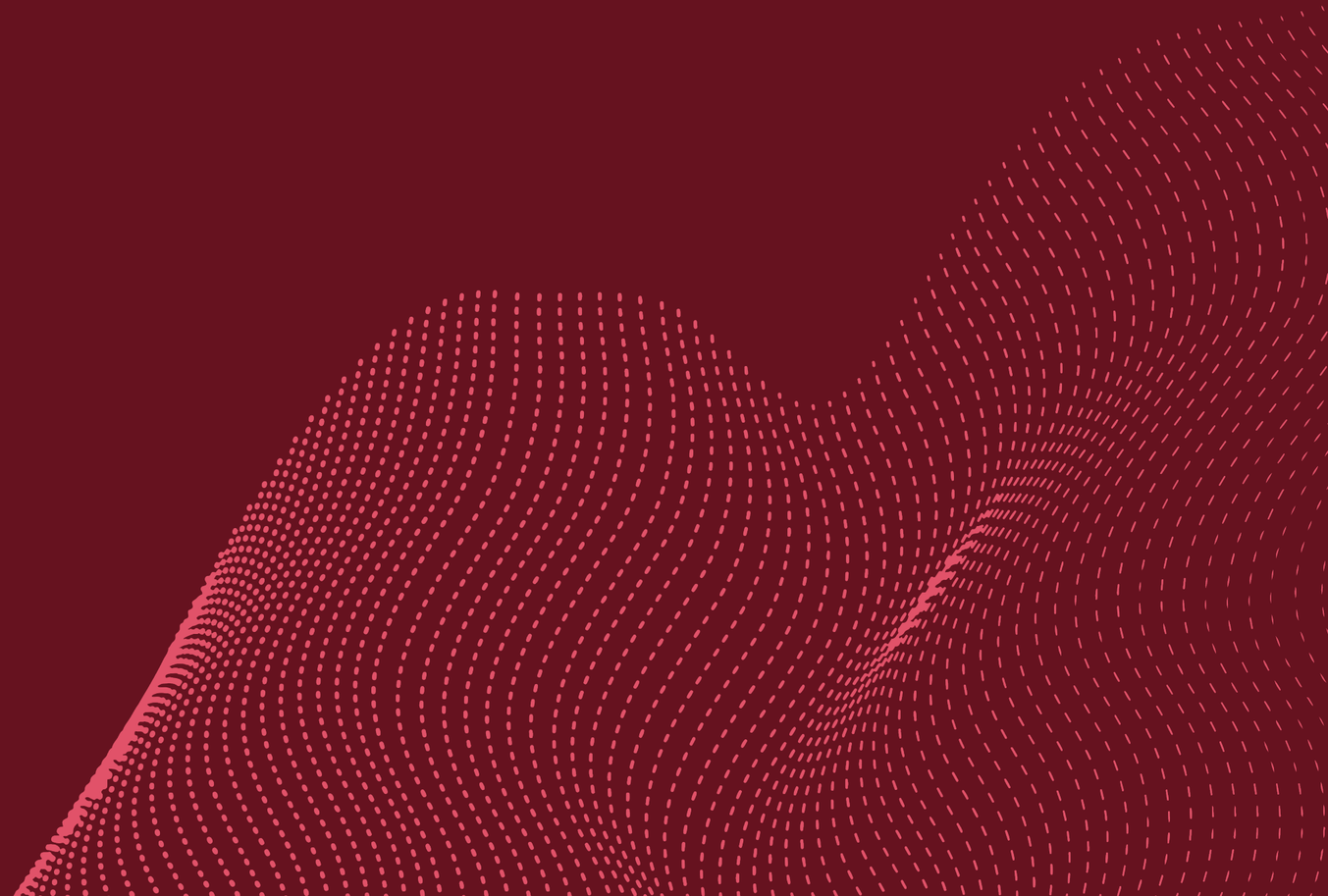
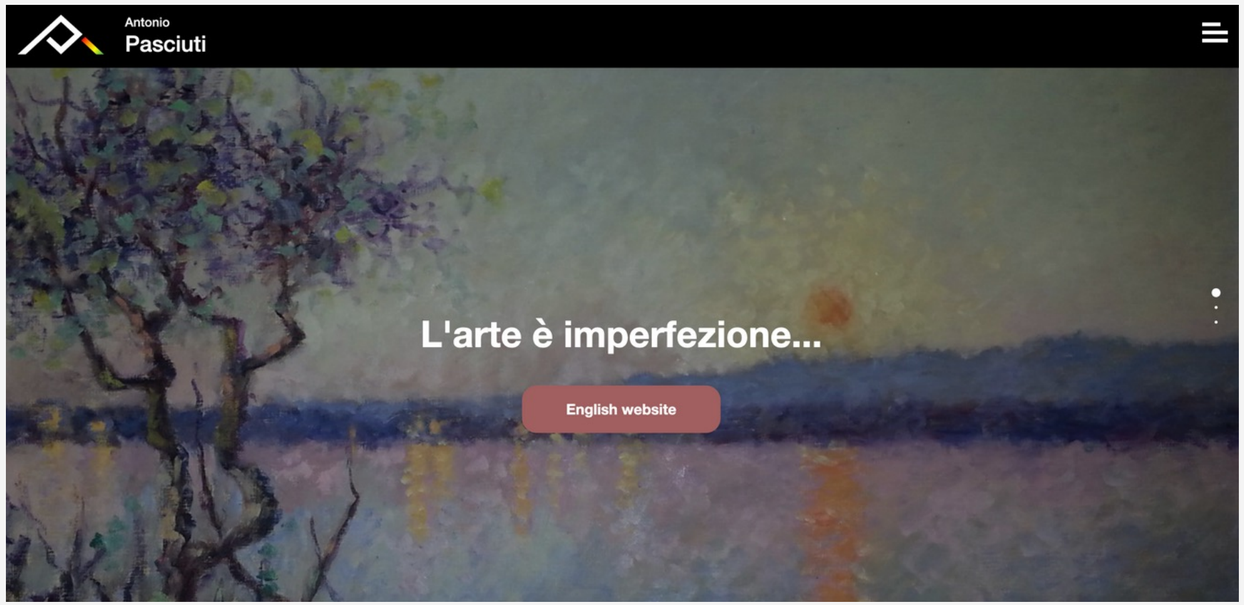
Artworks classification by using **Principal Component Analysis**

Mattia Tritto

MASTER DEGREE IN COMPUTER ENGINEERING



Project's overall context



Who I am

- I'm **Mattia Tritto**, a student enrolled in the Master's Degree program in Computer Engineering.

- I work at **Pasciuti Arte Srls**, a family-owned business specializing in selling artworks created by the artist **Antonio Pasciuti**, a famous Italian painter.

- My contribution to the business revolves around the field of innovation, process optimization and programming.

Project definition

- The objective of this project is to **differentiate between figurative and abstract artworks**.
- This is achieved by initially reducing the dimensionality of the data through **Principal Component Analysis** (PCA).
- Subsequently, a classifier is trained to predict the classification of artworks based on new input data.

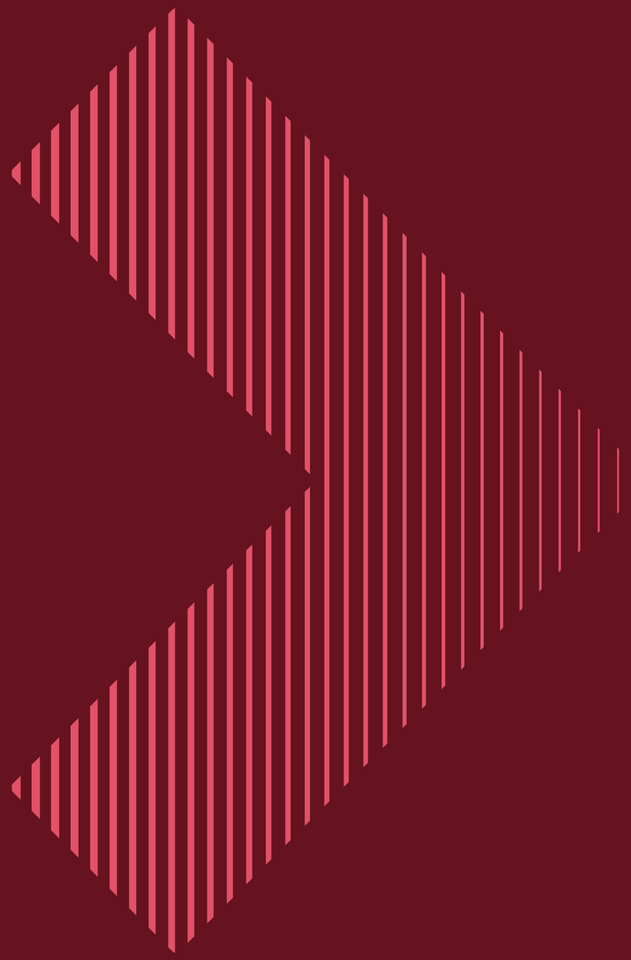
Example of a figurative artwork and an abstract one.

Dataset available

- For this project, I used a total of **120 artwork images**, evenly divided into 60 abstract and 60 figurative pieces.

- Out of these, **100 images are designated to train the classifier**, and the remaining **20 will serve to evaluate the classifier's performance** on new unseen data.

Example of a figurative artwork and an abstract one.

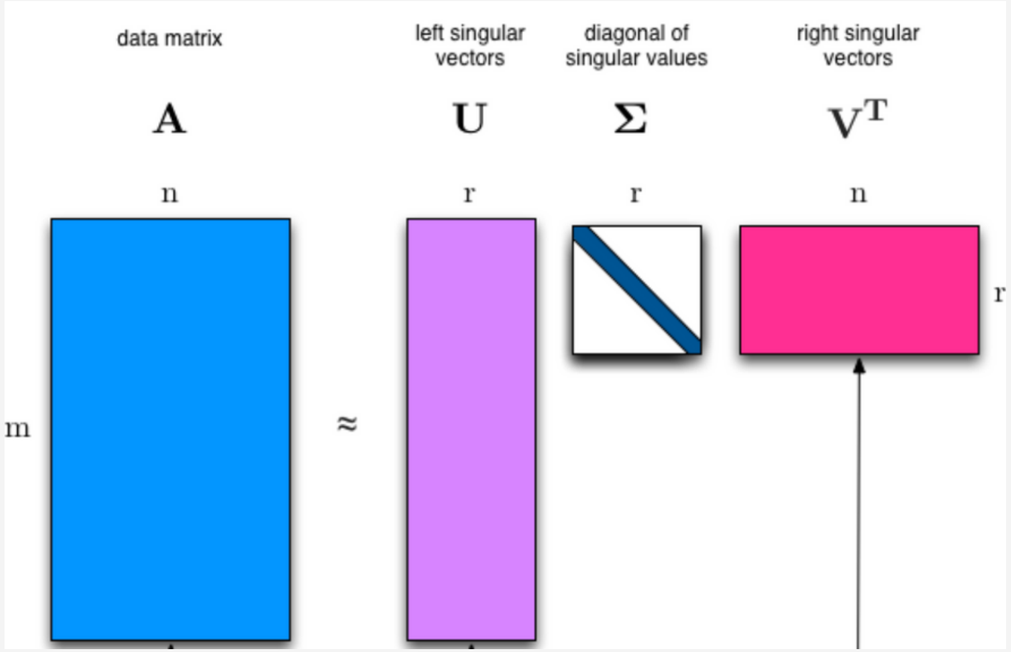


Topic covered during the course

- The upcoming slides provide a brief recap on the theoretical topics used in this project, all of which were fully covered during the course.

SVD

SVD stands for **Singular Value Decomposition**. It is a mathematical technique used in linear algebra and numerical analysis, and it has various applications in fields such as signal processing, data analysis, and machine learning.



Singular Value Decomposition

Given an arbitrary matrix $X \in \mathbb{R}^{nxm}$, X can be re-written as:

$$X = U \Sigma \mathrm{V}^{\mathrm{T}}$$

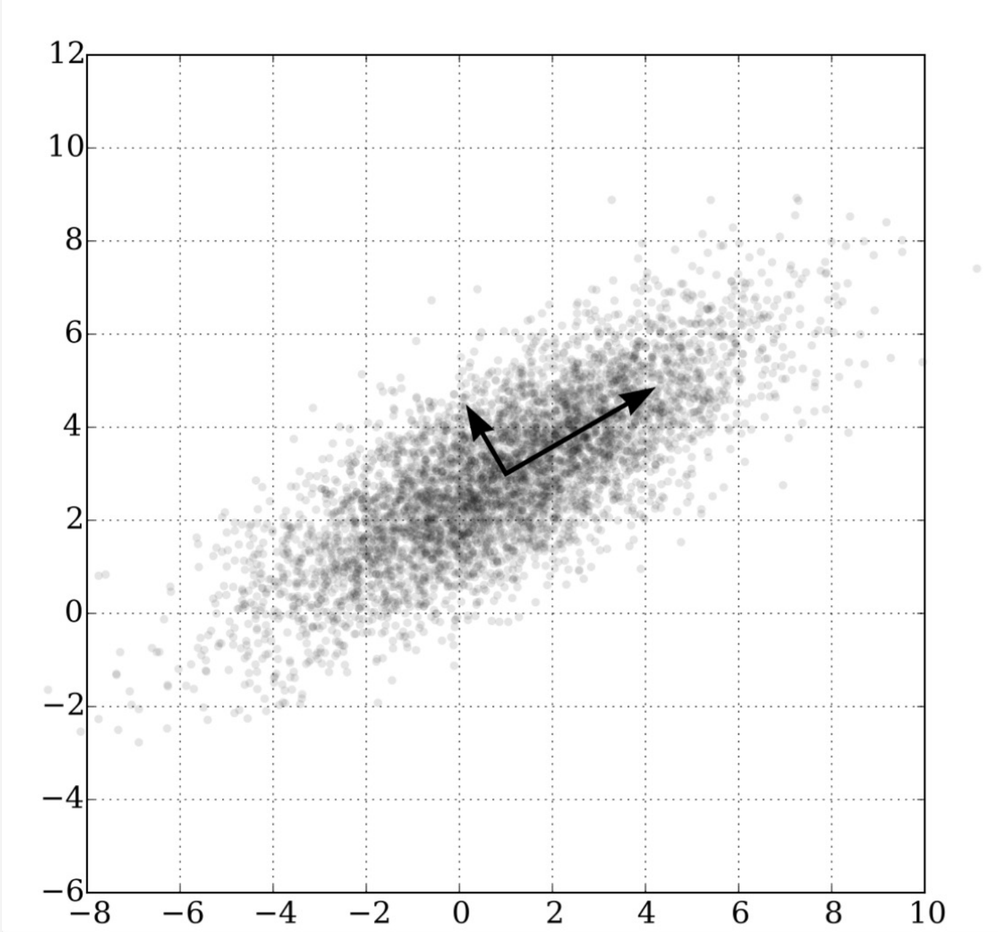
Where:

1) $U \in \mathbb{R}^{nxn}$, $V \in \mathbb{R}^{mxm}$ are **unitary matrices** such that $U^T U = UU^T = I_n$, and $V^T V = VV^T = I_m$. The columns of U and V are called **left singular vectors** and **right singular vectors** of X;
2) $\Sigma \in \mathbb{R}^{nxm}$ is a diagonal matrix. The entries of Σ are called **singular values** of X, and are uniquely determined. The number of non zero singular values is equal to the rank of X.

PCA

PCA stands for **Principal Component Analysis.** It is a statistical technique used for dimensionality reduction and feature extraction in data analysis and machine learning.

The main goal of PCA is to transform the original features of a dataset into a new set of variables, called **principal components**, which capture the **maximum variance in the data**.



How to compute PCA by using SVD

The principal components are new variables which are uncorrelated and ordered so that few retain most of the variation present in all of the original variables.

The first thing to do is to calculate the new dataset B by **removing the mean** of each column:

$$B = X - \bar{X}$$

The first Principal Component is the **eigenvector of the covariance matrix** $B^T B$ corresponding to the largest eigenvalue. Equivalent is to say that the first Principal Component is the **left singular vector of** B corresponding to its largest eigenvalue.

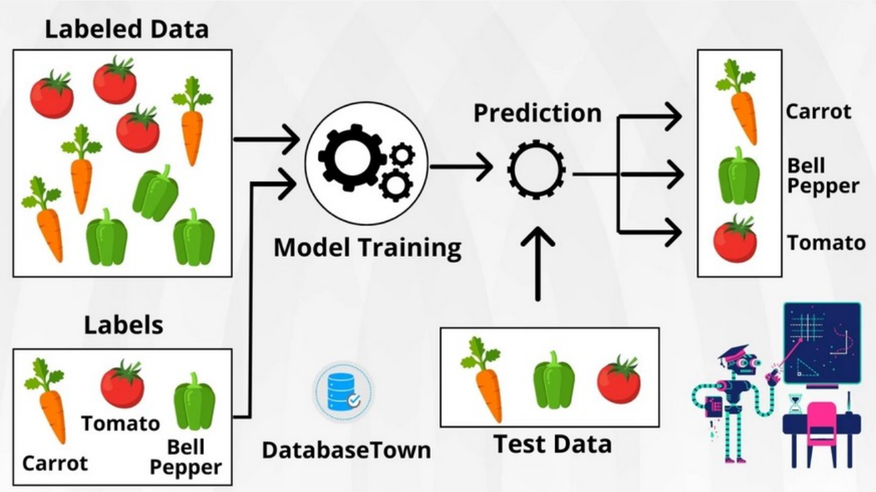
If $B = U \Sigma V^T$ then the PCs are given by the **columns of the Score Matrix** defined as:

$$T = U \Sigma$$

Supervised learning

Supervised learning is a type of machine learning paradigm where the algorithm is trained on a labeled dataset, which means that the input data is paired with corresponding output labels.

The goal of supervised learning is to **learn a mapping or relationship** between the input features and the target labels so that the algorithm can make predictions or classifications on new, unseen data.

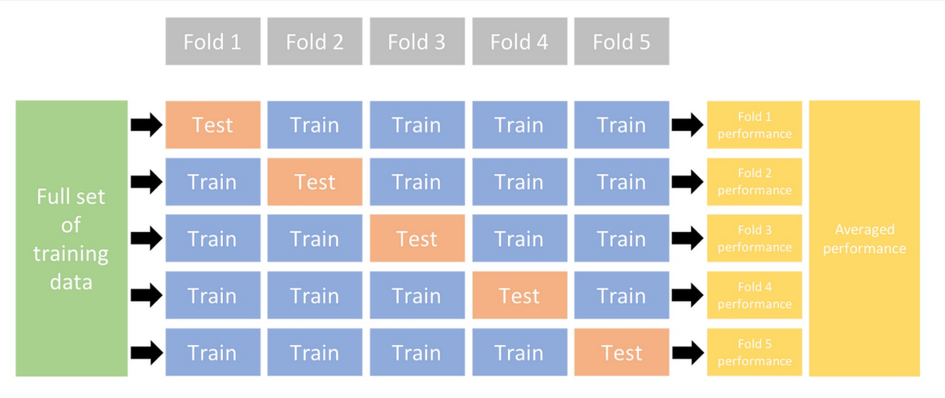


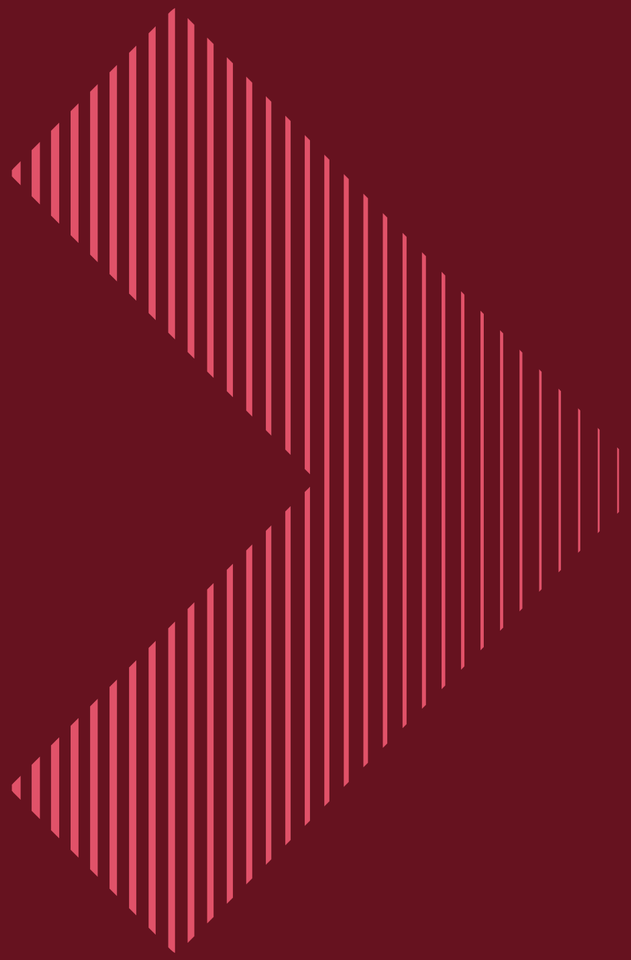
Cross validation

Cross validation is a resampling procedure used to evaluate ML models on a limited data sample.

The idea is **to use different portions of the dataset to train and test** a model on different iterations. This approach involves randomly dividing the set of observations into **k folds**.

The first fold is treated as a validation set and the model is trained on the remaining k-1 folds.





Outline of the project

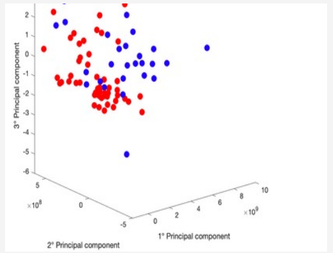
- The upcoming slides explain the methodology employed in constructing the model, utilizing the theory mentioned in the preceding slides as key reference points.

Workflow followed

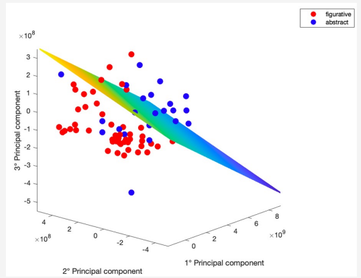
Crop and padding images with Photoshop

Preprocessing data

PCA and data reduction



Train a binary classifier



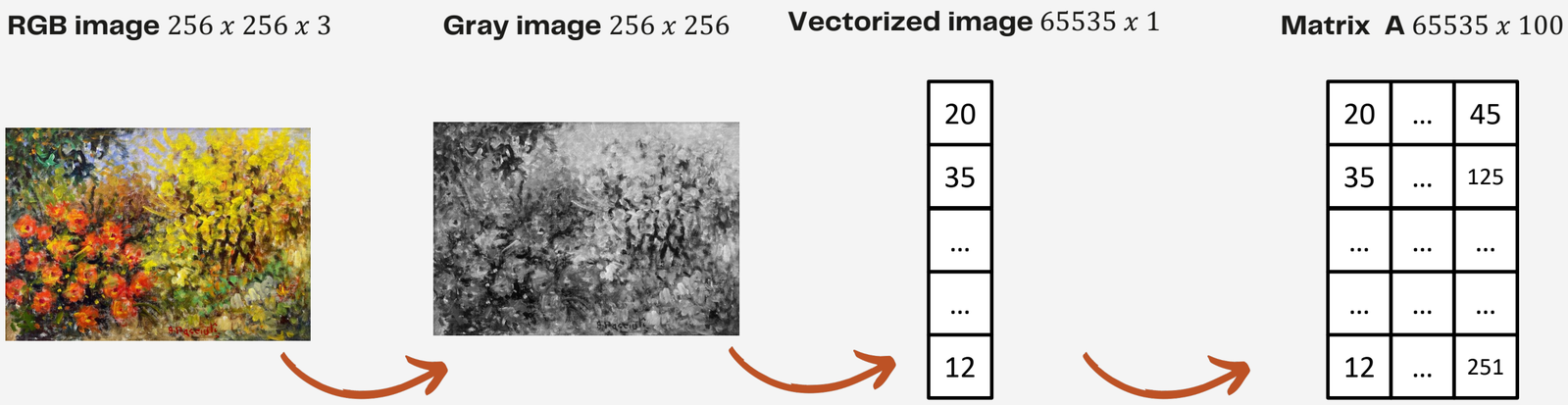
Crop and padding images with Photoshop

The initial step involved transforming all images into square dimensions of **256px x 256px**. Given that the artworks varied in shapes, rectangular images were filled with **white paddings** to ensure uniformity in the dimensions of each artwork.

I opted to utilize Photoshop for this task due to my familiarity with the software, and it facilitated the efficient batch processing of all images in a single operation.

Data preprocessing

Let A be the starting matrix of dimensions $m\ x\ n$ on which we want to apply PCA. In our specific case, A is a matrix of dimensions $65535\ x\ 100$. Every image is transformed from a matrix of $256\ x\ 256$ into a column vector of dimensions $65535\ x\ 1$. Then, each column vector is stacked horizontally to form our matrix A.



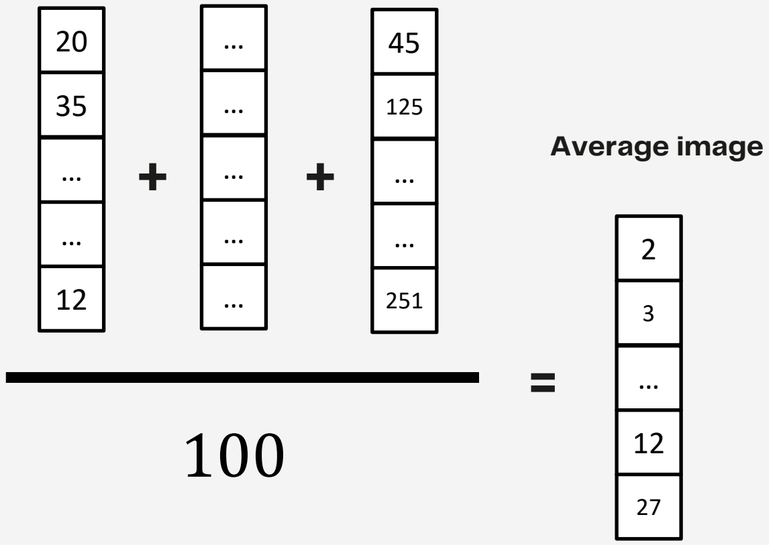
Average image calculation

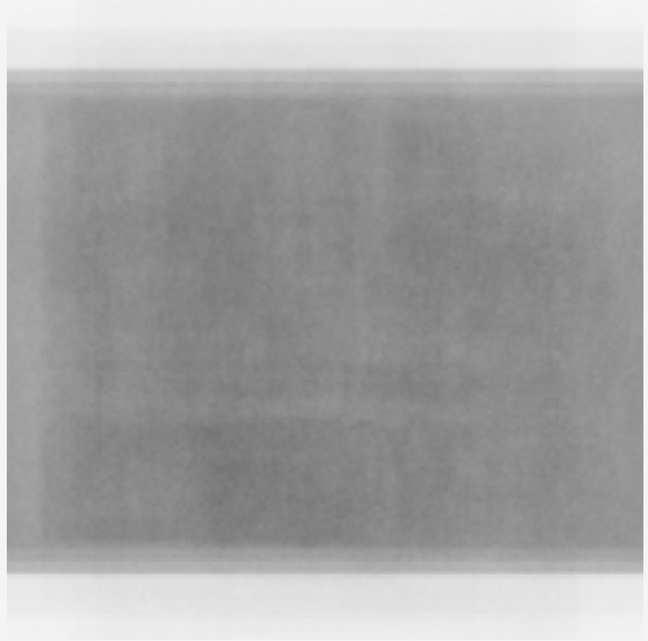
The **average image** (so a column vector $65535\ x\ 1$) is computed as the sum of all columns of A divided by the number of columns (the number of images used). The matrix B is composed of each column of A subtracted by the average image.

Why do we have to center our data? For two main reasons:

- When values are close to zero, the machine tends to be more precise in floating-point calculations;
- The PCA algorithm is much easier in terms of calculations when we center data.

Average image calculation





This is how it looks like

When stretching the eye, we note in the average image only abstract patterns. Figurative elements are perceived as a form of noise in the data.

Do we have to normalize our data?

It's not an absolute rule. The **presence of information might be encoded within the variance** of our data. This aspect is treated as one of the hyperparameters in our model. Following hyperparameter optimization, we can determine whether it proves to be a suitable choice or not.

Anyway, the standardization procedure for matrix B follows the same methodology seen as averaging. We calculate the **standard deviation for each column** and subsequently divide each column by its corresponding standard deviation.

Data dimensionality reduction using PCA

On matrix B is applied the **PCA algorithm** by using the **SVD decomposition**. The matrix B can be decomposed as follows:

$$B = U\,S\,V^T$$

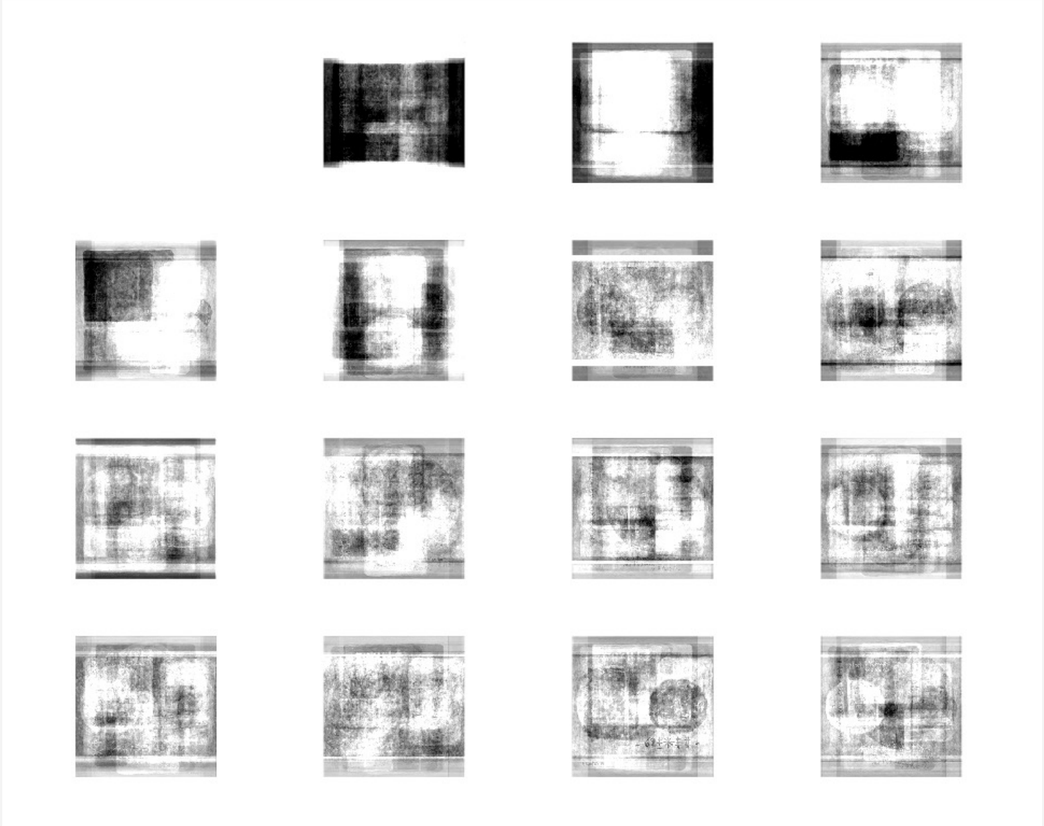
How do we obtain the principal components of B? The **principal components are the columns of the $Score$ matrix**, defined as:

$$Score = U\,S$$

Now the $Score$ matrix has 100 columns, and so 100 PCs. How do we choose the best number of Principal Components to use?

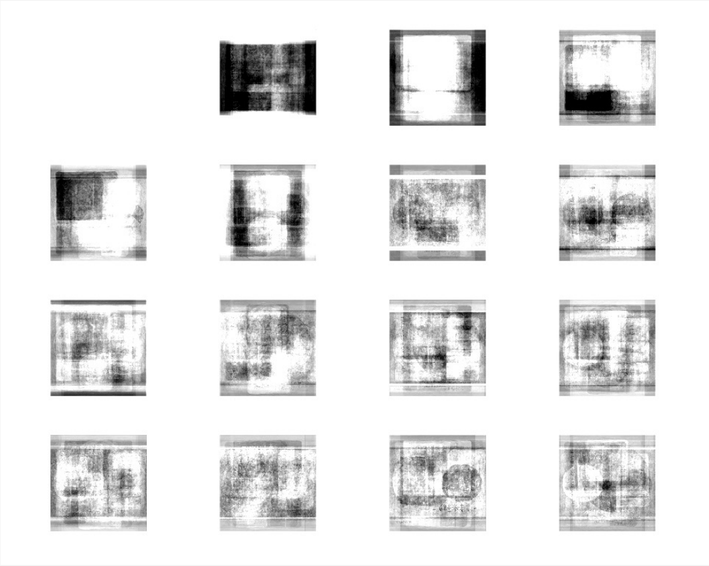
Principal Components, a visual representation

If we display the first 16 Principal Components (the first 16 columns of the $Score$ matrix), we can recognize some of the **abstract patterns** that are present in our data.

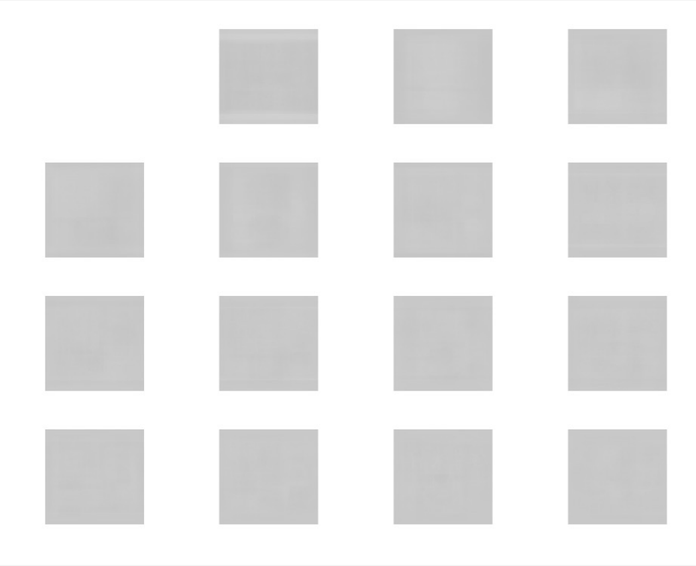


Principal Components, a visual representation

An intriguing observation is that when data is **normalized**, patterns are not really discernible. This suggests that normalizing the data might **obscure valuable information econded in the variance**, emphasizing the importance of considering whether or not.



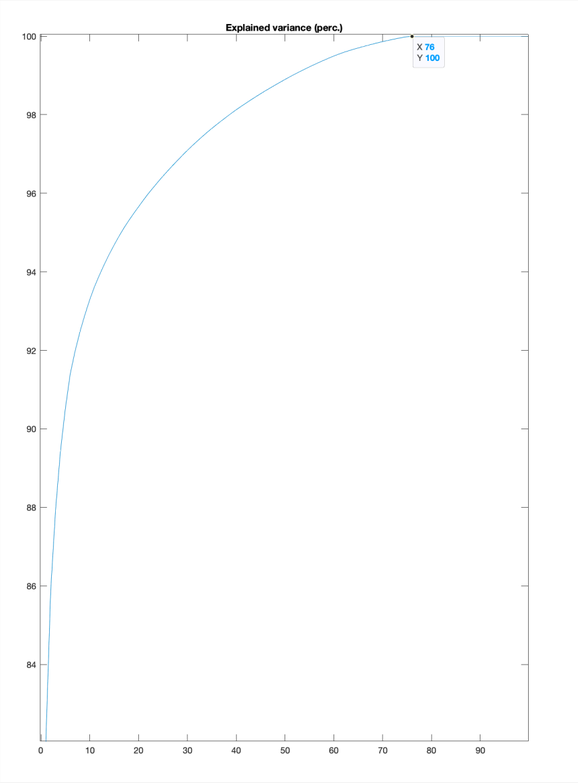
Without normalization



With normalization

A rough idea of how many PCs to use

One way is to calculate the **Explained Variance** for every number of Principal Component used to project new data. In our case, the explained variance for k principal components used is defined as:



$$EV_k = \frac{\sum_{i=1}^{k} S_{ii}^2}{\sum S_{ii}^2} * 100$$

Note that starting from the 76st principal component, we achieve 100% explained variance.

This is attributed to the fact that our images are padded with white pixels, effectively zeroing the variance.

Data projection in the new subspace

After having defined how many Principal Components to use, now we can project our dataset into the new subspace.

First, we define $ScoreReduced$ matrix as only **the first k columns of the matrix** $Score$:

$$ScoreReduced = Score(:, k)$$

The image n in our matrix B (the $n - th$ column) can be transformed in the new subspace as follows:

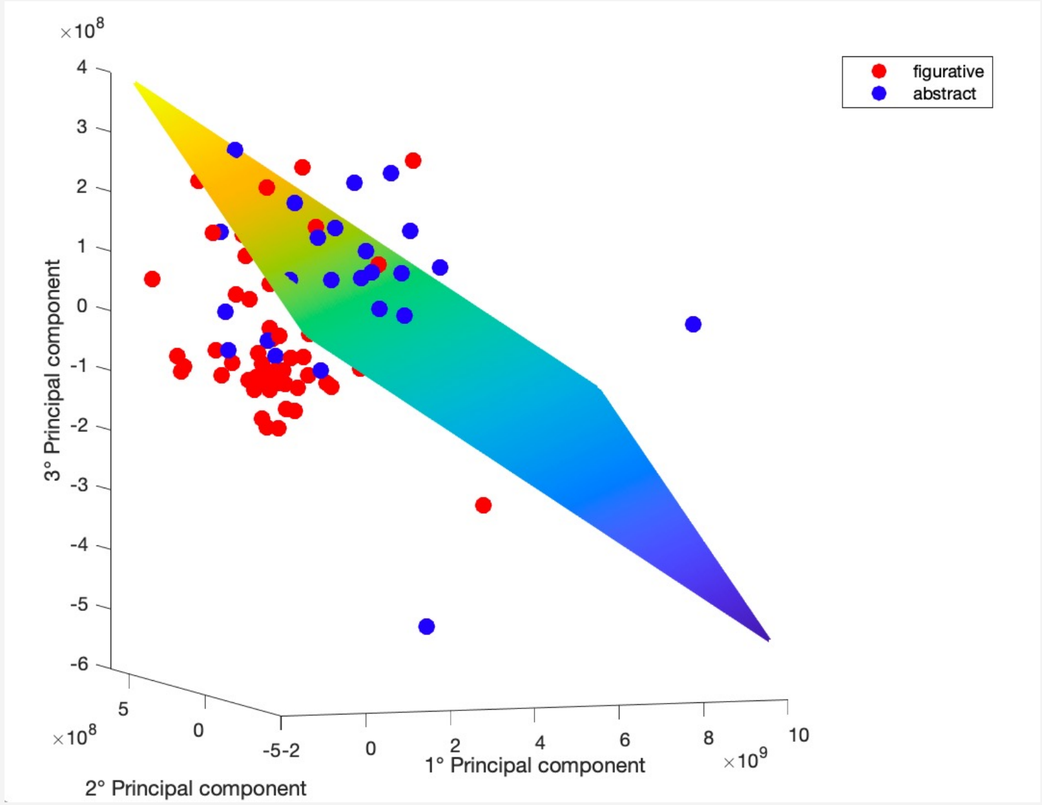
$$newImage_n = B^T(:, n) * ScoreReduced$$

Logistic regressor

Since my data is labeled, we can employ a **supervised algorithm**. In this instance, I' ve opted for a simple **logistic regressor** to assess how effectively PCA has linearly separated the data.

Each row in our dataset now represents an artwork, and each column the new principal component extracted. The last column represents the label (0 if figurative, 1 if abstract).

The logistic regressor finds the **equation of an hyperplane** that separate the best the figurative artworks from the abstract ones.



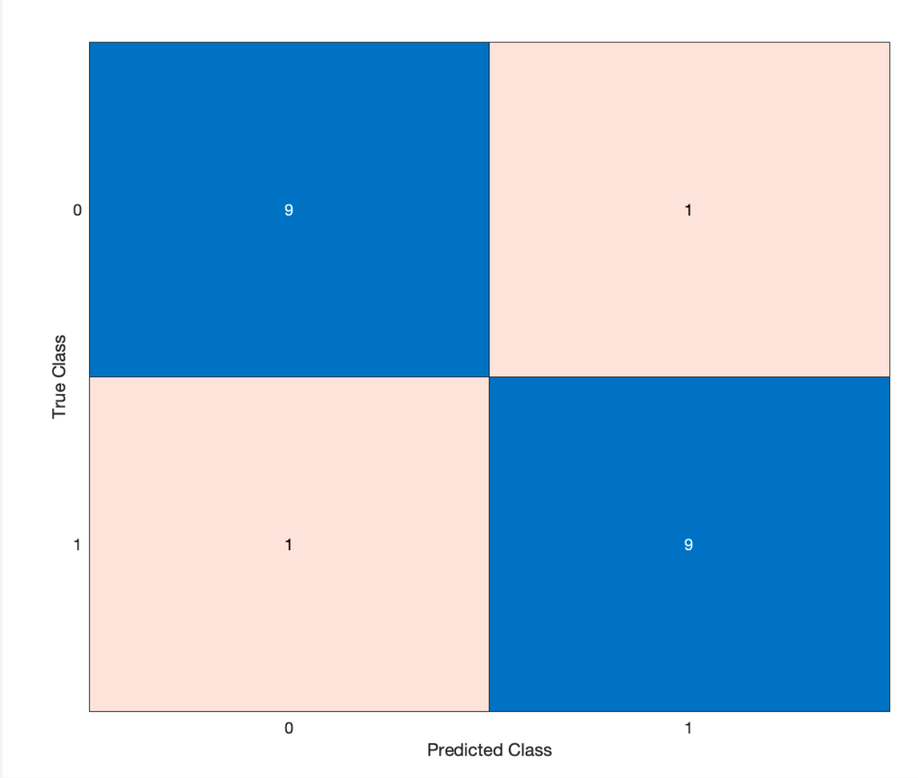
Logistic regressor evaluation

I decided to evaluate the performance of our regressor with the **accuracy metric**. The accuracy is a general classficiation metric used, and it is calculated as follows:

$$accuracy = \frac{True\ Positives + True\ Negatives}{Total\ number\ of\ predictions}$$

Accuracy gives us an idea of **how many times the predictor has labeled correctly our samples** with respect to all predictions has made.

In general, we can have an idea of how our model behaves by looking at his **confusion matrix.**

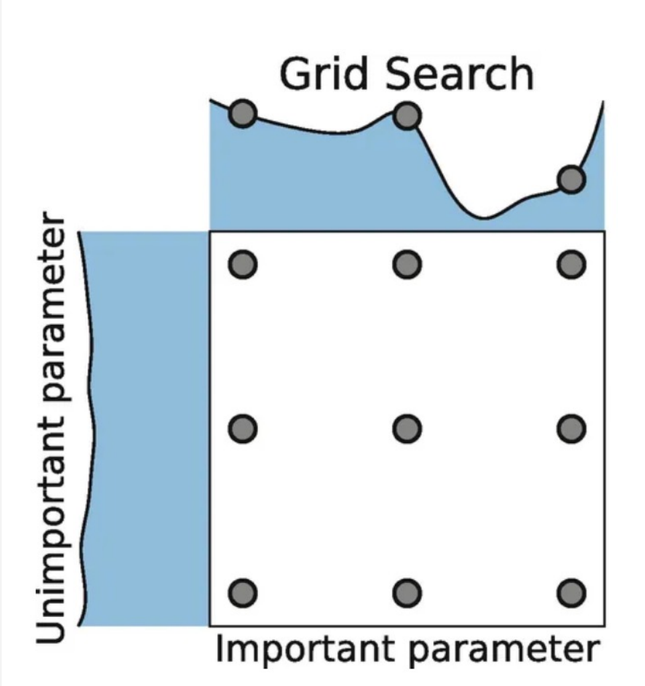


K-fold cross validation

In the process of evaluating our model, I utilized a **cross validation** technique known as **k-fold cross validation**. K-fold cross-validation involves partitioning the dataset into k equally-sized folds. The model is trained and evaluated k times, each time using a different fold as the test set and the remaining k-1 folds as the training set. This process helps ensure robustness in assessing the model's performance, as it covers different subsets of the data for training and testing.

The accuracy test score, in this context, is calculated as the **mean accuracy across all k esperiments**. This approach provides a more comprehensive and reliable estimation of the model's performance by considering its consistency over multiple subsets of the data. It helps mitigate potential biases that may arise from a specific split of the data, offering a more robust evaluation of the model's generalization ability.

Find the best hyperparameters



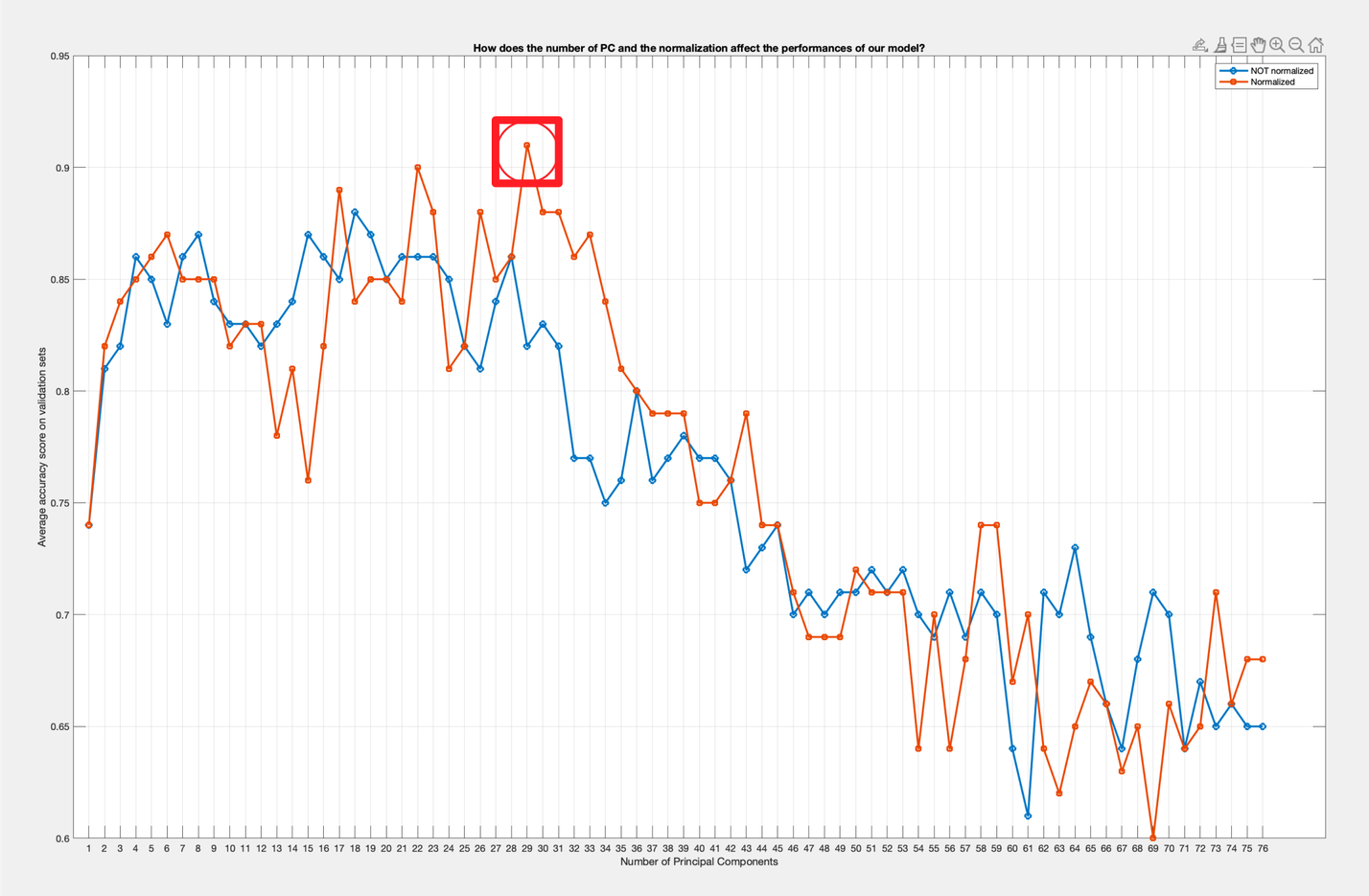
How to find the best configuration

- To determine the optimal hyperparameters, I selected predefined values and conducted a **grid search**, choosing the set that yields the highest accuracy.
- This task is quite time-consuming, as it took 3 hours to search for the best combination among 152 different hyperparameter sets.
- The grid search focused on optimizing two key parameters: whether **normalization should be applied and** determining the optimal **number of principal components** to use.

Hyperparameters grid search: results

In this graph, we can see how the accuracy goes with respect to the number of principal components used.

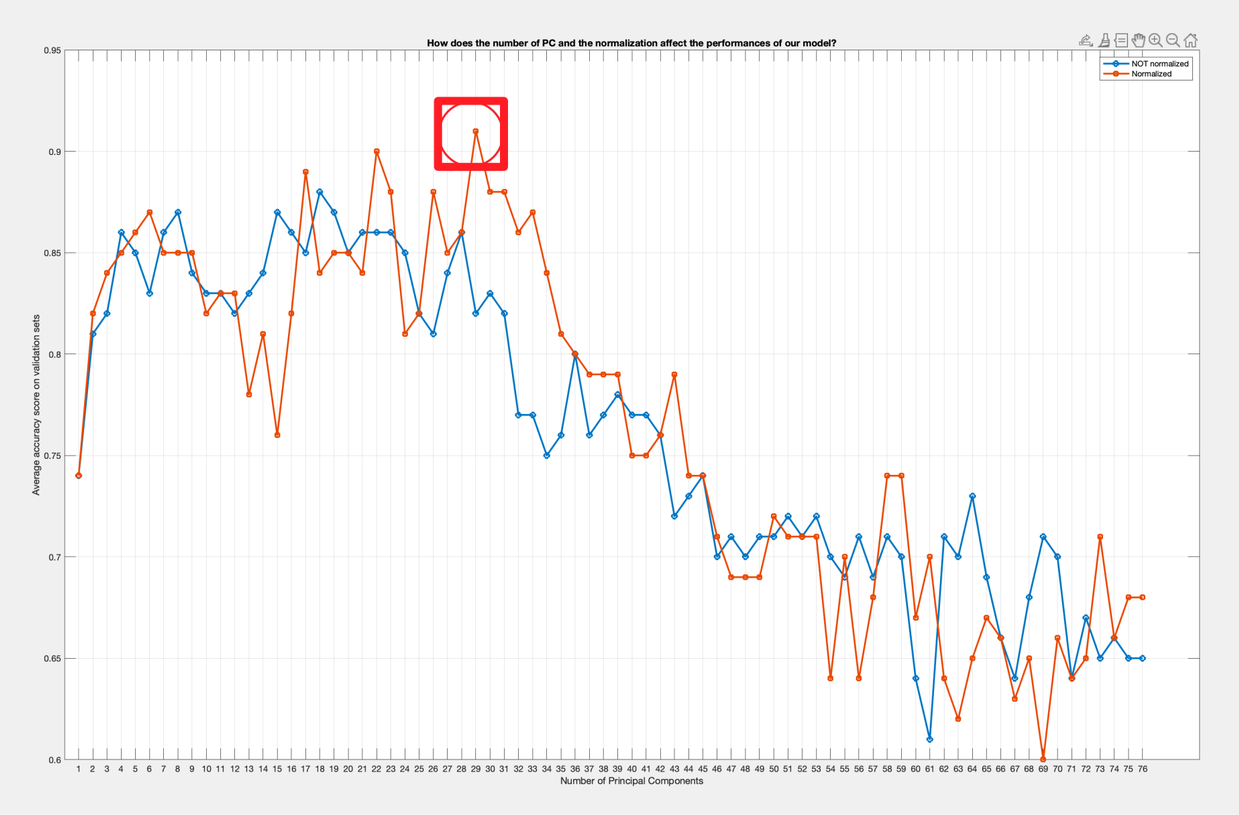
The blue line is without normalization, while the orange line is with normalization.



Hyperparameters grid search: results

According to the graph, I have selected the following hyperparameters, opting for those that resulted in the **highest average accuracy on validation sets:**

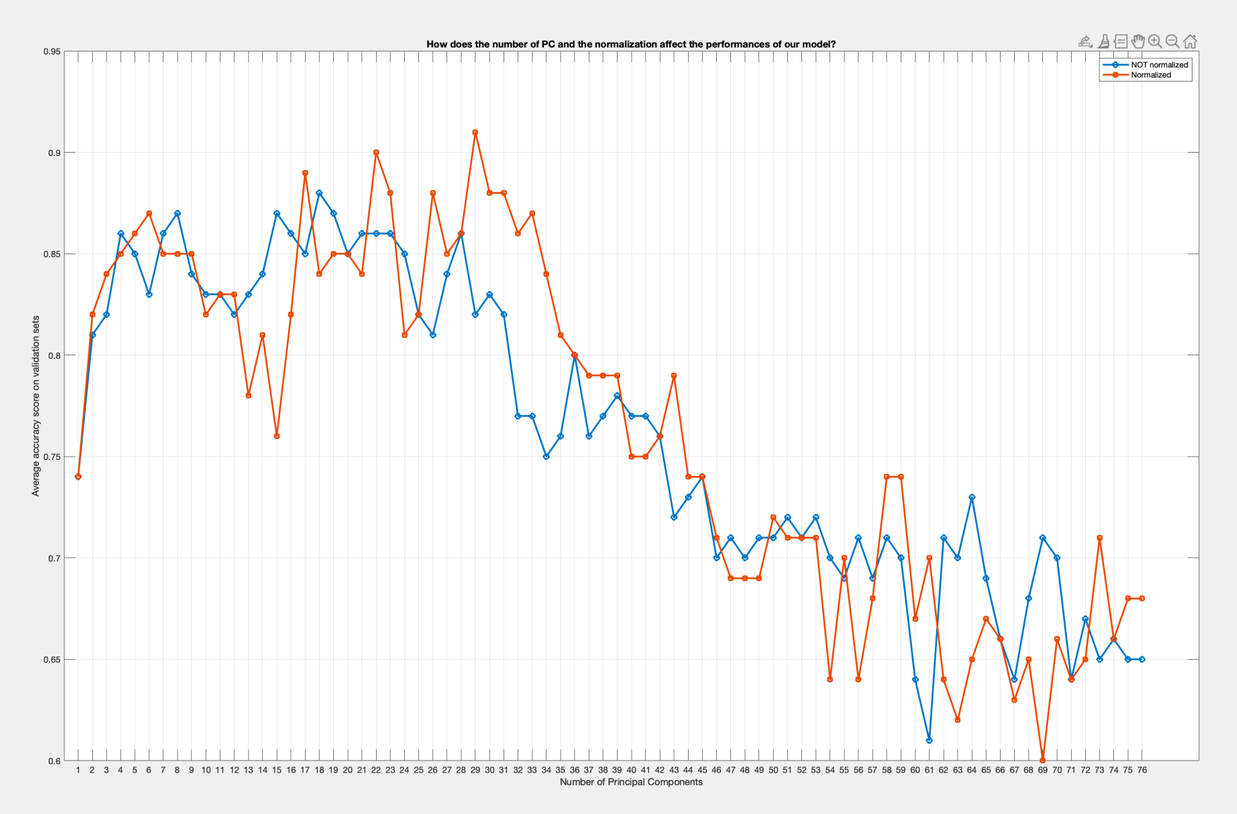
- Normalization: Yes
- Principal Components: 29



Hyperparameters grid search: the curse of dimensionality

Note that after the optimal number of principal components, the **model's performance tends to decline**. This serves as a practical demonstration of the **curse of dimensionality** in Machine Learning.

The curse of dimensionality refers to the challenges and limitations that arise as the number of features or dimensions in the data increases.



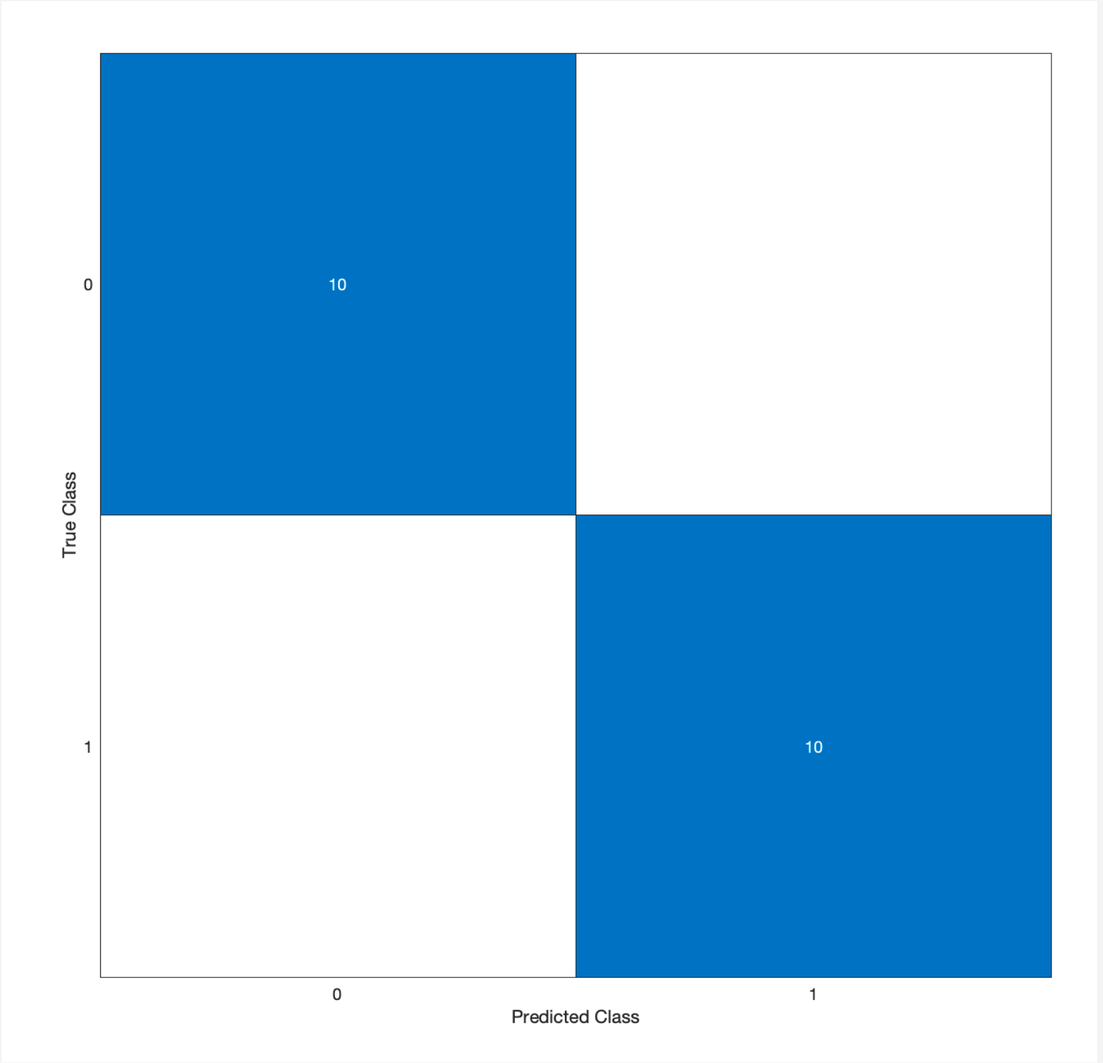
Results

The best model has an accuracy of 100%* on new unseen data.

- The current result may seem trivial due to the small testing data size (only 20 samples), which is **insufficient** for a comprehensive model evaluation.

- However, this suggests that the model can be really robust, and increasing the dataset size could lead to substantial success.

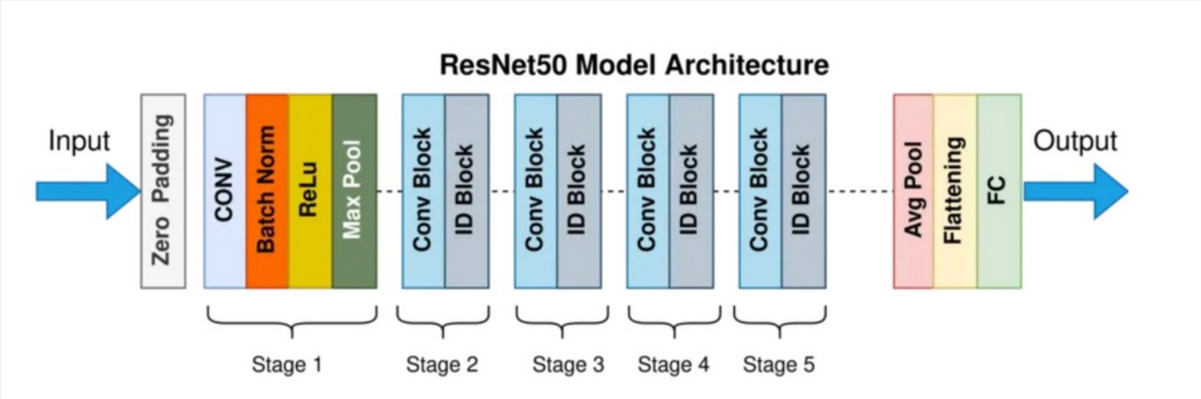
- On the right we have the confusion matrix that is referred to the test set used.



How about a NN?

In literature, the best way to classify images is by using convolutional NNs. In this case, I've used the Neural Network ResNet-50.

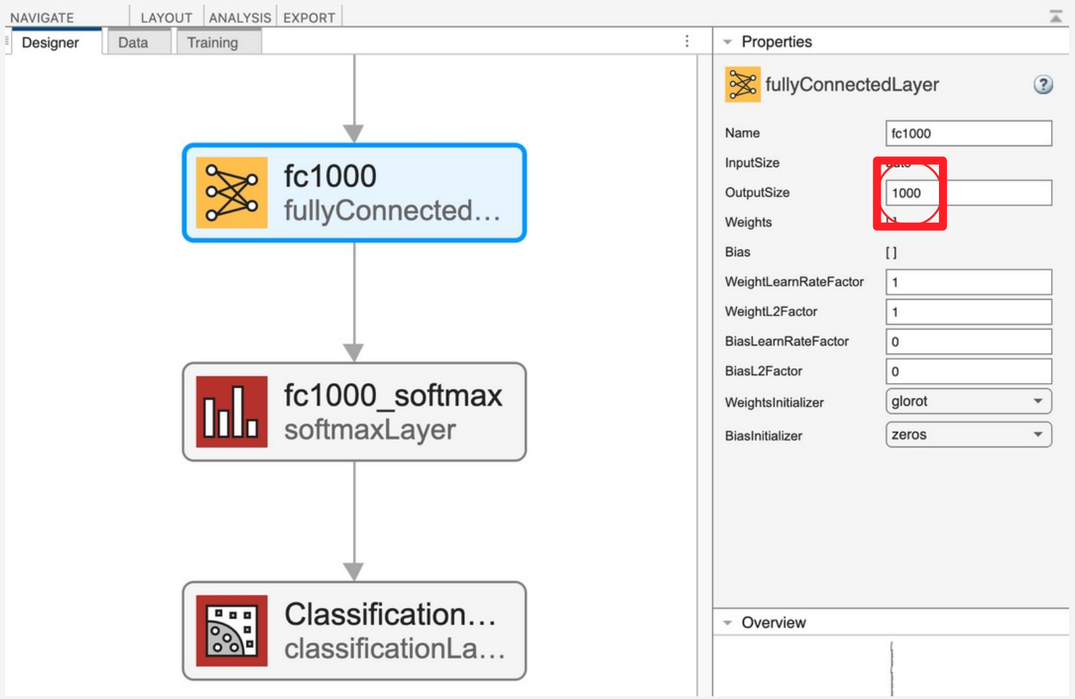
ResNet-50



- ResNet-50 is a **convolutional Neural Network** that is 50 layers deep.
- We can load a pretrained version of the neural network trained on more than a million images from the ImageNet database.
- Using Transfer Learning, we can fine-tune the pretrained version using our data.

ResNet-50 architecture modifications

I utilized the **Deep Network Designer toolbox** in MATLAB to load the default version of ResNet-50. To align with our specific problem constraints, I made modifications solely to the output layers.

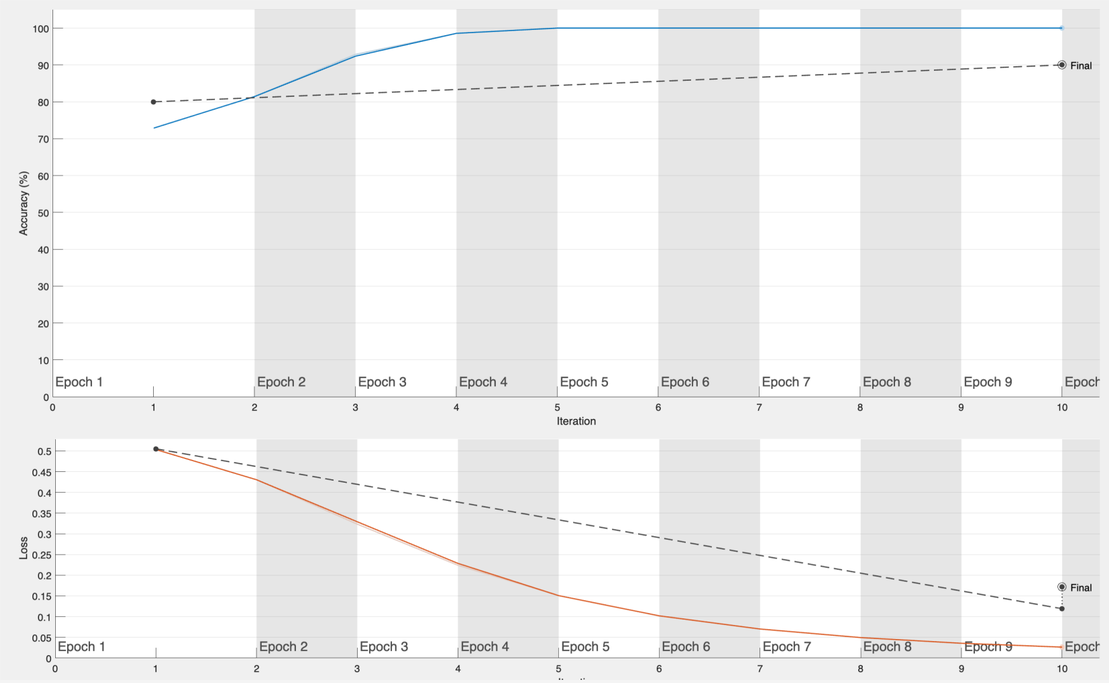


ResNet-50 fine tuning

The fine tuning was performed easily in the GUI provided by MATLAB.

The first graph is the **average accuracy on validation sets** for each epoch.

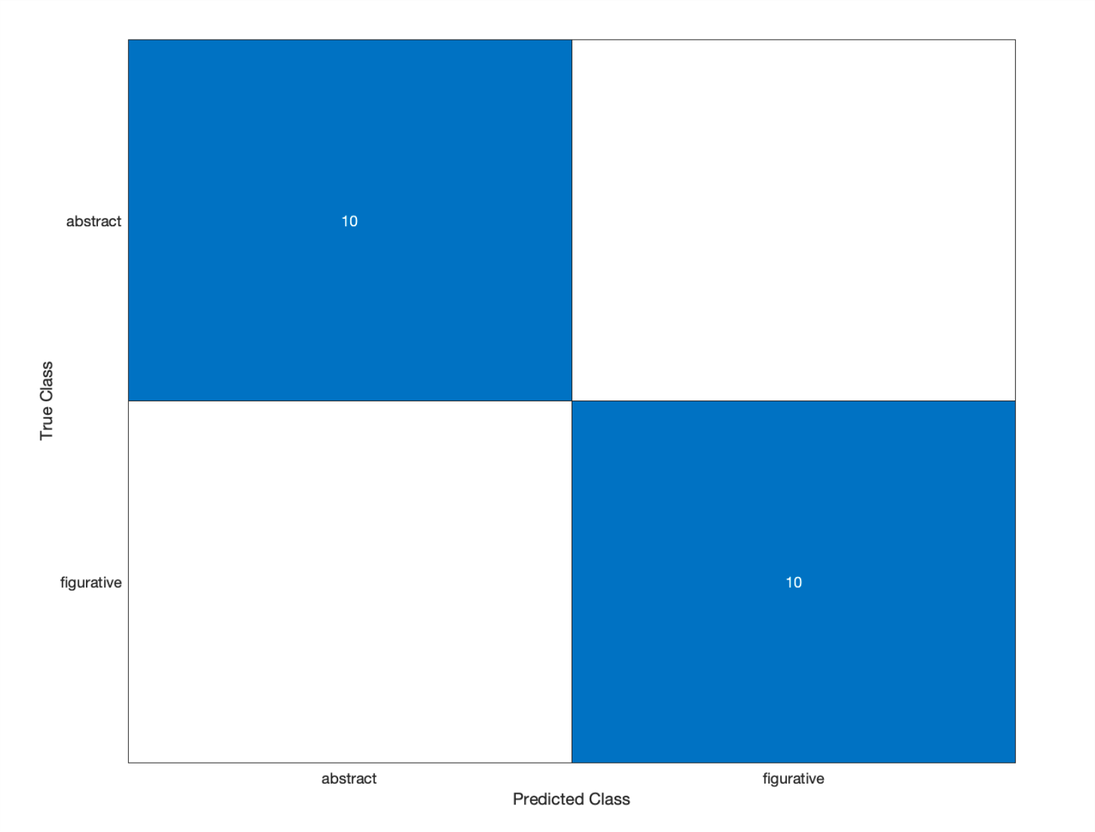
The second graph is the value of the **loss function** for each epoch.



Results

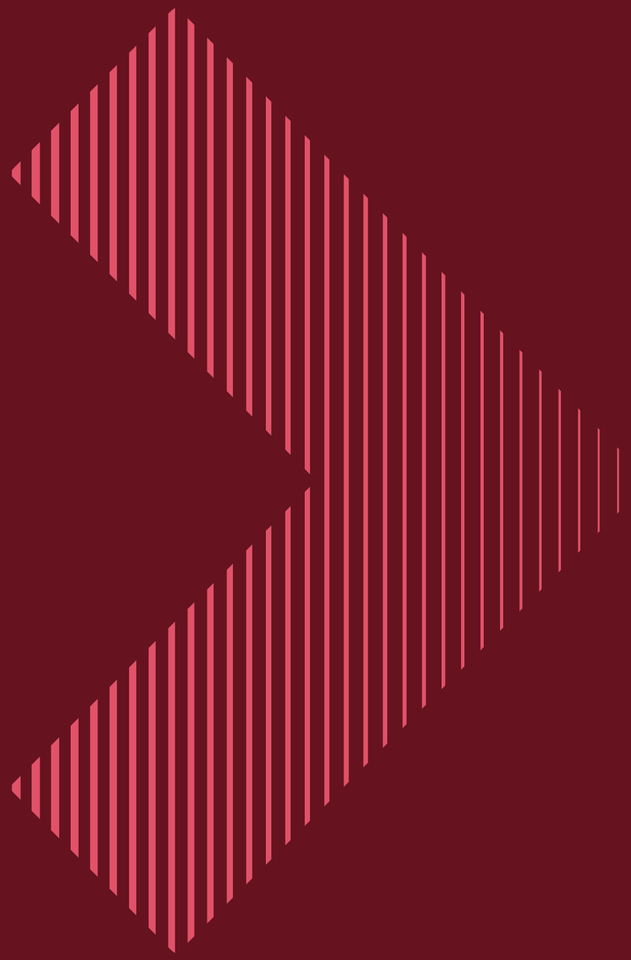
The Neural Network has an accuracy of 100%* on new unseen data.

Also in this case, the current result may seem trivial due to the small testing data of only 20 samples which is insufficient for a comprehensive model evaluation.



The winner is...

According to the current data, **there is not a clear winner**. However, once the artworks archive will be completed, it will be intriguing to determine who emerges as winner.



How do we use the model?

Once the optimal hyperparameters for our ML model are determined, what specific objects should be stored in order to make new predictions?

Use case of this model

This model proves highly beneficial for previous clients who have purchased artworks in the past and seek information such as the **range of years in which a particular painting was created**.

The concept involves the old client **uploading** a photo of their artwork, which is then subjected to a **PCA** dimensionality reduction function. Then, the model calculates whether **resulting data point lies above or below the hyperplane** that distinguishes abstract works from figurative ones.

What kind of information I need to store in order to make new predictions?

In order to make new predictions, I need to store:

1) The **average image vector**;
2) The **score matrix (only the first 29 columns)**;
3) The **coefficients of the hyperplane**.

To begin, the initial step involves **vectorizing the matrix** of the $newImage$. Subsequently, **subtract the average vector** and divide the result by its standard deviation. Following this, apply the transformation using the **score matrix**. Finally, insert the transformed data into the **hyperplane formula** to decide whether the artwork resides above or below the hyperplane.

What kind of information I need to store in order to make new predictions with the NN?

To generate new predictions using the Neural Network, we can export the network structure in **PyTorch**.

This exported structure can then be employed within a Python environment, linked with the website's backend. Additionally, it is imperative to include all the **weights** that the NN has learned during the fine-tuning stage in order to perform the **forward propagation**.

Bibliography

- MATLAB documentation, PCA
- MATLAB documentation, Deep Network Designer
- ResNet-50 paper