# DEVELOPMENT OF IMAGE CLASSIFICATION APPROACHES BASED ON DIMENSIONALITY REDUCTION METHODS

by Viktor Avhust

A Capstone Project

Presented in Partial Fulfillment of the Requirements for the Degree

Master

American University Kyiv

2026

APPROVED BY:

Prof. Vasyl Semenov, Faculty Mentor

TABLE OF CONTENTS

# ABSTRACT

Principal Component Analysis (PCA) is a powerful tool for reducing the dimensionality of image data while retaining critical information. This capstone project explores the application of PCA in image recognition, focusing on its use in simplifying high-dimensional image datasets to improve recognition efficiency and performance. The study focuses on two primary applications: recognizing handwritten digits (HWD) and classifying galaxies using three different types of classification.

Custom software was developed to implement these image classification tasks, integrating dimensionality reduction techniques with classification algorithms to achieve high accuracy rates. The results demonstrate the effectiveness of these approaches in handling high-dimensional datasets, paving the way for using the method and implementing it in human detection software using HWD and tools to support astronomers in their research.

# INTRODUCTION

Image recognition is a critical area of research in machine learning and computer vision, with applications ranging from human (facial) recognition to object detection. However, dealing with high-dimensional image data presents significant challenges, especially high computational costs and difficulties in processing redundant information.

Dimensionality reduction techniques like PCA are essential for simplifying these datasets, improving efficiency and accuracy. PCA is a statistical method that identifies the directions of maximum variance in data, allowing us to represent complex image datasets in a reduced form without significant loss of information.

In this capstone project we do the following:

- Make overview of existing approaches to image recognition
- Apply PCA to the recognition of HWD (handwritten digits) and galaxies images
- Define proper ways preprocessing the data
- Define the optimal number of PCA components for various applications
- Investigate the influence of various hyperparameters onto the process of classification
- Set the directions for further research

# CHAPTER 1. EXISTING APPROACHES FOR IMAGE CLASSIFICATION IN ML

Deep learning, especially convolutional neural networks (CNNs), has become a dominant approach in image classification due to its ability to automatically learn hierarchical features from raw images [1–7].

Image classification has evolved significantly, with a variety of methods developed over the years. These methods range from traditional machine learning algorithms, which rely on handcrafted features, to more modern deep learning approaches that self-learn features automatically. Here's an overview of some of the key most popular methods used in image classification:

## 1.1 Traditional Machine Learning Methods

- **Support Vector Machines (SVM)**: SVMs are supervised learning models that can be used for image classification. They work by finding an optimal hyperplane that separates classes in a high-dimensional space. For image classification, SVMs often rely on pre-extracted features, such as Histogram of Oriented Gradients (HOG) or Scale-Invariant Feature Transform (SIFT) descriptors, to create a robust feature representation.
- **K-Nearest Neighbors (KNN)**: KNN classifies an image based on the majority label of its "k" nearest images in the feature space. Though simple, it can be effective for small-scale image datasets but struggles with large datasets due to its computational inefficiency.
- **Decision Trees and Random Forests**: Decision trees classify images by making a series of decisions based on feature values. Random forests, an ensemble of decision trees, improve accuracy and reduce overfitting by averaging the output from multiple trees. These methods, however, are limited: particularly by their reliance on handcrafted features.

### 1.2 Feature Extraction Techniques

- **Histogram of Oriented Gradients (HOG)**: This technique extracts gradient orientation features to represent the shape and structure within an image, making it useful in object detection.
- **Scale-Invariant Feature Transform (SIFT)**: SIFT is used to detect key points and descriptors that are invariant to scale and rotation, useful for matching similar objects in different images.
- **Bag of Visual Words (BoVW)**: BoVW uses key points (like SIFT) to create a "visual vocabulary." It then represents images as a histogram of the occurrence of these "visual words," enabling classification by conventional machine learning methods.

### 1.3 Deep Learning Methods

- **Convolutional Neural Networks (CNNs)**: CNNs revolutionized image classification by automating feature extraction. They consist of layers such as convolutional layers, pooling layers, and fully connected layers that learn spatial hierarchies and complex patterns from images. Each layer captures different levels of abstraction, from simple edges to complex shapes, making CNNs highly effective for image classification.
    - **Popular Architectures**: Some well-known CNN architectures include:
        - **AlexNet**: The first CNN to popularize deep learning for image classification, winning the ImageNet competition in 2012. It introduced innovations like ReLU activations and dropout for better performance.
        - **VGGNet**: Known for its simplicity and depth, VGGNet uses small, uniform filters to achieve high performance but is computationally demanding.
        - **Inception (GoogLeNet)**: Introduced by Google, Inception uses multi-scale processing and dimensionality reduction to increase efficiency and accuracy.

    - **ResNet**: Introduced residual connections to address the problem of vanishing gradients in very deep networks, allowing for training much deeper models.

- **Transfer Learning**: This technique leverages pre-trained models (e.g., ResNet, VGG) trained on large datasets and fine-tunes them for specific tasks. It's effective when labeled data is limited, as the model has already learned generalized features from a broader dataset.
- **Data Augmentation**: A technique used to artificially increase the size of the training dataset by applying transformations like rotation, flipping, and cropping. This helps prevent overfitting, allowing models to generalize better and be more "prepared" for new images.

### 1.4 Emerging and Specialized Methods

- **Capsule Networks**: Capsule networks are an alternative to CNNs, proposed to better capture spatial relationships between distinct features. They preserve hierarchical spatial information, which can be useful for applications requiring high spatial precision, such as medical imaging.
- **Attention Mechanisms**: Originally developed for natural language processing, attention mechanisms have been adapted for image classification. They allow models to focus on the most relevant parts of an image, which can improve accuracy by helping the network ignore irrelevant background details.
- **Vision Transformers (ViTs)**: Transformers, which revolutionized NLP, are being adapted to image classification with models like Vision Transformers. They split an image into patches and apply self-attention to capture global dependencies across patches, making them particularly effective for large image datasets.

### 1.5 Challenges and Techniques for Improvement

- **Class Imbalance**: Many image datasets have class imbalance, where certain classes are over-represented. Specific techniques like oversampling, undersampling, or using weighted loss functions help address this issue.

- **Model Interpretability**: Image classification models, especially deep learning models, are often black boxes. Methods like Grad-CAM and saliency maps provide insights by highlighting image regions that contributed most to a prediction.
- **Efficiency in Deployment**: Models designed for high accuracy often have high computational costs. Techniques like model pruning, quantization, and knowledge distillation are used to create smaller, more efficient models suitable for real-time applications.

In summary, each method has its strengths and trade-offs. While CNNs remain dominant, advances in capsule networks and vision transformers are expanding possibilities in image classification. By handcrafted combination of traditional methods with modern architectures and techniques, machine learning approach continues to improve image classification, enabling more accurate and efficient applications across industries.

# CHAPTER 2. EXISTING PROBLEMS AND CHALLENGES IN IMAGE CLASSIFICATION

Image classification has seen significant advancements, yet there are several ongoing challenges that limit its effectiveness, especially as models are applied to real-world situations. Here are some of the primary problems faced in the field:

## 2.1 Data Challenges

- **Data Scarcity**: High-quality, labeled datasets are essential for training accurate models, yet labeled data can be expensive and time-consuming to collect. In specialized fields like medical imaging, labeled datasets may be limited, which can hinder model performance.
- **Class Imbalance**: Many real-world datasets have imbalanced class distributions, where some classes are heavily represented while others are rare. Particularly in this work we rejected the use of imbalanced galaxy data. This can lead to bias in predictions, as models tend to favor the over-represented classes.
- **Data Privacy and Security**: Collecting image data can pose privacy concerns, particularly in sensitive areas such as healthcare and surveillance. This work uses open-source data only. Regulations like GDPR restrict the usage of personal data, making it challenging to gather diverse datasets.

## 2.2 Model Interpretability and Explainability

- **Black-Box Nature of Deep Learning**: Most deep learning models, particularly CNNs, operate as “black boxes,” meaning it is difficult to understand how they arrive at their decisions. So, this lack of interpretability is a barrier in fields requiring accountability, such as healthcare, where understanding the basis of a classification is crucial.
- **Need for Interpretability Tools**: Methods like Grad-CAM and saliency maps attempt to highlight the regions that contributed to a model’s prediction.

Considering that these tools are still evolving, and there is no universal solution for reliably interpreting complex models, we don't use it in this work.

### 2.3 Generalization to New Domains

- **Domain Shift and Distribution Changes**: Models trained on a specific dataset may struggle when applied to new data that differs slightly in distribution, such as images taken in different lighting conditions, from different angles, or with different devices. This limits the robustness and generalizability of models across varied environments.
- **Data Augmentation and Domain Adaptation**: Although techniques like data augmentation and domain adaptation can help models generalize, these techniques aren't always sufficient to fully address large domain shifts. Models trained on synthetic or simulated data often struggle in real-world applications.

### 2.4 Computational and Resource Constraints

- **High Computational Costs**: Training deep learning models on large image datasets requires extensive computational resources, which can be "killing" for smaller organizations or applications running on edge devices.
- **Latency and Real-Time Requirements**: Many applications require real-time or low-latency predictions, especially in areas like autonomous driving and surveillance. Standard deep learning models can be too slow for these tasks, especially when run on low-power devices.

### 2.5 Handling High Intra-Class Variability and Low Inter-Class Variability

- **Intra-Class Variability**: Objects within the same class can vary significantly in appearance (e.g., different breeds of dogs or variations in handwritten characters). Models need to learn general features that can handle these variations while maintaining accuracy.

- **Low Inter-Class Variability**: Conversely, objects from different classes can look very similar (e.g., different types of birds or different brands of vehicles). This makes it challenging for models to distinguish between similar classes accurately.

### 2.6 Adversarial Vulnerabilities

- **Susceptibility to Adversarial Attacks**: Image classification models, especially deep neural networks, can sometimes be vulnerable to adversarial attacks, where small, imperceptible changes to an image cause the model to misclassify it. This poses a security risk in applications like facial recognition and autonomous vehicles.
- **Need for Robustness**: Developing models that can resist adversarial manipulation is an active area of research, but no robust solution exists yet, especially for large-scale, real-world applications.

### 2.7 Ethical and Bias Concerns

- **Bias in Datasets**: If a model is trained on data that underrepresents certain groups or scenarios, it may produce biased results. For example, facial recognition systems have been found to perform poorly on minority groups due to lack of representation in training data.
- **Ethical Implications**: In some cases, misclassification can have serious consequences, especially in sensitive applications such as law enforcement and healthcare. Selecting galaxies classification was used for this work specifically to not address ethical concerns required for careful curation of datasets and potentially rethinking certain applications of image classification.

### 2.8 Complexity of Real-World Scenes

- **Multi-Object and Context Recognition**: Real-world images often contain multiple objects and complex backgrounds. While image classification

typically assumes a single dominant object, recognizing the primary object amidst background noise or multiple objects is challenging.

- **Context Awareness**: Models may struggle to understand context, which can lead to incorrect classifications. For instance, distinguishing a picture of a cat from a dog may require understanding not only features of the animal itself but also contextual clues in the image.

### 2.9 Scalability and Model Maintenance

- **Need for Continuous Learning**: Real-world scenarios may involve changing distributions or new categories over time. Current models struggle with "continual learning" without forgetting previously learned categories, which limits their scalability.
- **Model Updating and Fine-Tuning**: Updating models as new data becomes available can be resource-intensive. Additionally, there is a risk that updating a model on new data could lead to degraded performance on older data.

In summary, despite advancements, image classification models face numerous challenges, from data limitations and interpretability issues to ethical concerns and computational constraints. Researchers like us are working on new methods, including self-supervised learning, robust architectures, and efficient model compression techniques, to address these issues. However, overcoming these challenges will be essential for applying image classification effectively and responsibly across different domains in the future.

# CHAPTER 3. THE APPLICATION OF PCA TO IMAGE RECOGNITION

PCA is a mathematical technique used to transform data into a new coordinate system, where the axes represent the directions of maximum variance, known as principal components. By projecting data onto these new axes, PCA reduces the number of dimensions while retaining the most critical aspects of the original data. This transformation is achieved by computing the eigenvectors and eigenvalues of the covariance matrix of the data. The principal components are ordered by the amount of variance they capture, allowing us to select the most important components for further analysis. PCA is particularly useful for unsupervised learning tasks where the goal is to find patterns in high-dimensional data.

Images are often represented as high-dimensional matrices of pixel values, which can contain redundant or irrelevant information. PCA helps reduce the dimensionality of these image representations by capturing the essential features that contribute most to the variance in the dataset. By remaining only the top principal components, PCA significantly reduces the complexity of the image data, making it easier to process and analyze.

In image recognition tasks, PCA is used to extract the most informative features from images. These features correspond to the principal components that distinguish between different images. Focusing on these components allows PCA to improve the accuracy and efficiency of recognition algorithms, as they can concentrate on the most relevant information rather than processing redundant pixel data.

The application of PCA in image recognition offers several benefits. It reduces the computational complexity of recognition algorithms, making them faster and more efficient. Additionally, by removing noise and focusing on key features, PCA can improve the robustness of recognition systems, leading to better performance in real-world situations.

### 3.1 Basics of PCA

Let us consider how PCA detects and removes the redundancy in the data from the mathematical viewpoint.

Suppose our feature vectors $x_i$, $i = 1, ..., m$ (preliminary normalized to have zero mean and unit variance) are grouped row-by-row into matrix. The main steps of PCA are as follows.

Let us consider how PCA detects and removes this redundancy. Suppose our feature vectors $x_i$, $i = 1, ..., m$ (preliminary normalized to have zero mean and unit variance) are grouped row-by-row into matrix $\boldsymbol{X}$. The main steps of PCA are as follows.

1. **Covariance Matrix:** Compute the covariance matrix $C$ of the normalized data:

$$C = \frac{1}{m}\sum_{i=1}^{m} x_i x_i^T$$

2. **Eigenvalue Decomposition:** Perform eigenvalue decomposition on $C$ to find the eigenvectors and eigenvalues:

$$C = V\Lambda V^T$$

   Here, $V$ is the matrix of eigenvectors , and $\Lambda$ is the corresponding diagonal matrix of eigenvalues (sorted by their decreasing).

3. **Projection:** Select the top $k$ eigenvectors corresponding to the largest eigenvalues and project the original data onto this new $k$-dimensional subspace:

$$Z = XV_k$$

   where $\boldsymbol{X}$ is the centered data matrix, and $V_k$ contains the top $k$ eigenvectors.

So, the resulting matrix $Z$ has $m$ vectors with $k$ components (instead of original $n$ components).

### 3.2 Application of PCA to Recognition of Handwritten Digits

To demonstrate the use of PCA in image recognition, consider an example involving a dataset of handwritten digits. The following steps outline the implementation process:

- Load the image dataset and flatten each image into a vector.
- Standardize the data to have zero mean and unit variance.
- Apply PCA to reduce the dimensionality of the dataset by selecting the top principal components.
- Use the reduced feature set to train a classification algorithm (feedforward neural network) to recognize the images.

As the first step of the experiment, we consider recognition of digits from MNIST dataset [8].

The **MNIST dataset** (Modified National Institute of Standards and Technology) is a foundational benchmark dataset in machine learning, widely used specifically for handwritten digit recognition. It consists of **70,000 grayscale images** of digits ranging from 0 to 9, with each image represented as a **28×28 pixel matrix**, resulting in 784 features per sample when flattened. The pixel values range from 0 (black) to 255 (white), providing a consistent and straightforward representation of the digits. The dataset is divided into **60,000 training samples** and **10,000 testing samples**, allowing for robust evaluation of model performance.

Each image in the MNIST dataset is labeled with the corresponding digit, enabling its use in supervised learning tasks. The dataset's simplicity and clarity have made it a standard for testing algorithms in classification, especially for deep learning models like convolutional neural networks (CNNs). It has been employed to evaluate a wide range of machine learning techniques, from basic logistic regression and support vector machines to advanced neural network architectures.

Despite its age, MNIST remains a valuable tool for benchmarking due to its well-structured format and manageable size. It is often used to introduce concepts in computer vision and neural network design, as it provides an ideal balance of

accessibility and challenge. Moreover, its extensive use in literature makes it a reference point for comparing new algorithms and methodologies in image classification.

First, preprocessing is needed. An important step is to scale the data. Dividing to 255 helps:

- **Brings Values to a Standard Scale**: Scaling to [0, 1] ensures that all pixel values are of comparable magnitude and directly interpretable as intensities.
- **Improves Numerical Stability**: Operations like covariance computation in PCA, as well as weight updates in neural networks, become more stable because the input values are smaller.
- **Better Gradient Flow in Neural Networks**: Smaller input values prevent excessively large activations and gradients, reducing the likelihood of vanishing or exploding gradients.

Also, since PCA aims to identify directions of maximum variance in the dataset, for this process to work effectively, the data must be centered around the origin by subtracting the mean value calculated on the training dataset.

Using this dataset of grayscale images, each image was reshaped into a 784-dimensional vector to construct a high-dimensional data matrix.

PCA was applied to decompose the data into orthogonal components ranked by their variance contribution, allowing the selection of the most informative features.

First, we investigated how many PCA components are enough to retain a meaningful amount of information. Calculating the covariance matrix for the training dataset of 60000 images we can determine its eigenvalues. In this example the first 10 eigenvalues are:
[0.10047663, 0.07544487, 0.06140516, 0.05425807, 0.05031249, 0.04246363, 0.03311404, 0.02950288, 0.02729858, 0.02278041]. It means that they preserve correspondingly 10% , 7.5%, 6.1%, 5.4%, 5% , 4.2%, 3.3%, 3% , 2.7% and 2.3% of information about the original dataset.

To determine the appropriate number of components we can analyse the cumulative sum of its eigenvalues as presented in Figure 1:

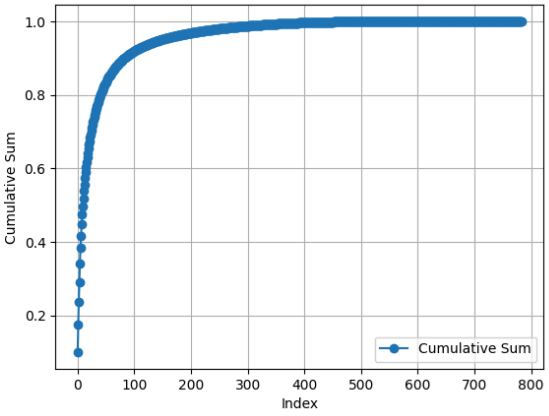


**Figure 1.** Cumulative Sum of Eigenvalues

Number of PCA components to preserve some percentages of image information is shown in Table 1.

**Table 1.**

| Amount of preserved info | Number of PIC |
|---|---|
| 0.99 | 322 |
| 0.95 | 148 |
| 0.9 | 83 |
| 0.8 | 41 |
| 0.75 | 32 |
| 0.7 | 24 |

As we see from the graph, 83 components are enough to preserve 90% of information. This substantial reduction in dimensionality—from 28*28=784 pixel features to just 83 components—highlighted PCA's effectiveness in compressing high-dimensional image data while retaining meaningful information.

Also let us pay attention to the view of **eigenimages** for MNIST digits obtained by PCA. These eigenimages correspond to the eigenvectors (principal components) obtained during **Principal Component Analysis (PCA)** by formulas . These eigenvectors represent the directions of maximum variance in the dataset and are reshaped back into the original image dimensions (e.g., 28×28 pixels for MNIST) for visualization.

The first 16 eigenimages have the following form:

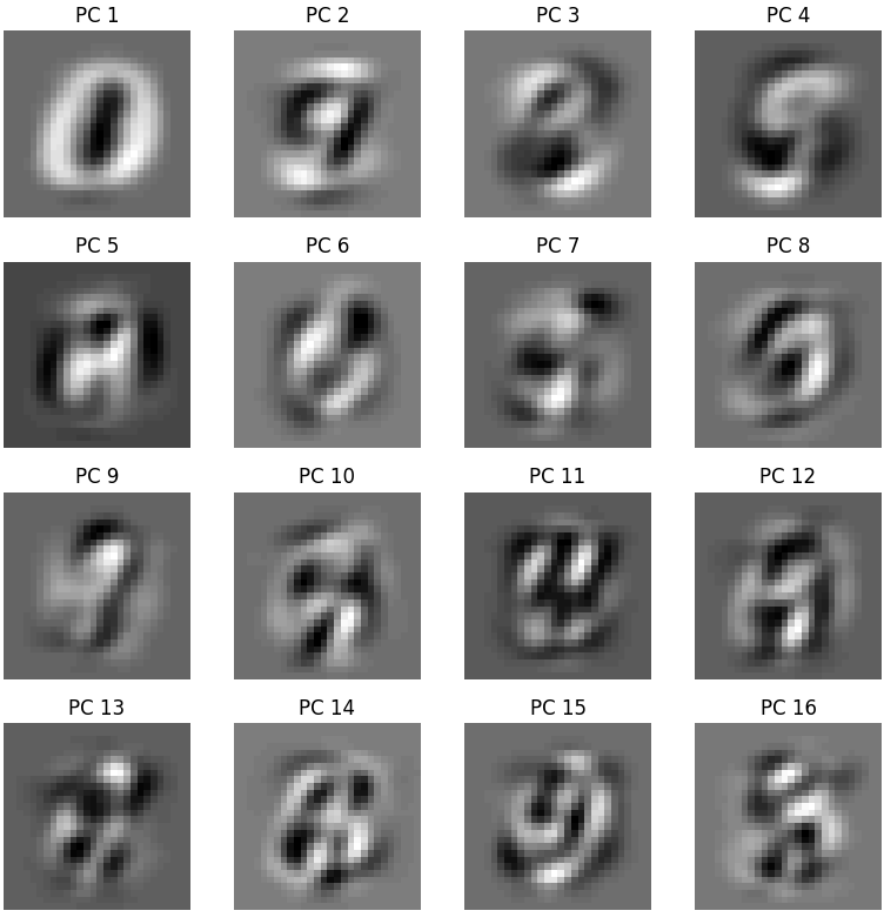


**Figure 2.**

These Eigenimages **(**eigenvectors) can be interpreted as "basis images" that capture the key patterns or variations in the dataset. For MNIST digits, these might look like "smudges" or "hints" of digit shapes because they describe the directions of variance in the data.

- Each eigenvector is associated with an eigenvalue that indicates the amount of variance it explains.
- The eigenimages with larger eigenvalues are more representative of the dataset.
- The first few eigenimages explain the bulk of the variance and contain the most prominent features.
- Later eigenimages (corresponding to smaller eigenvalues) capture finer details or noise.
- Moreover, the visualization of the first few principal components revealed their interpretability. The principal components corresponded to patterns resembling

generic features, such as edges, gradients, and high-level structure present in the images.

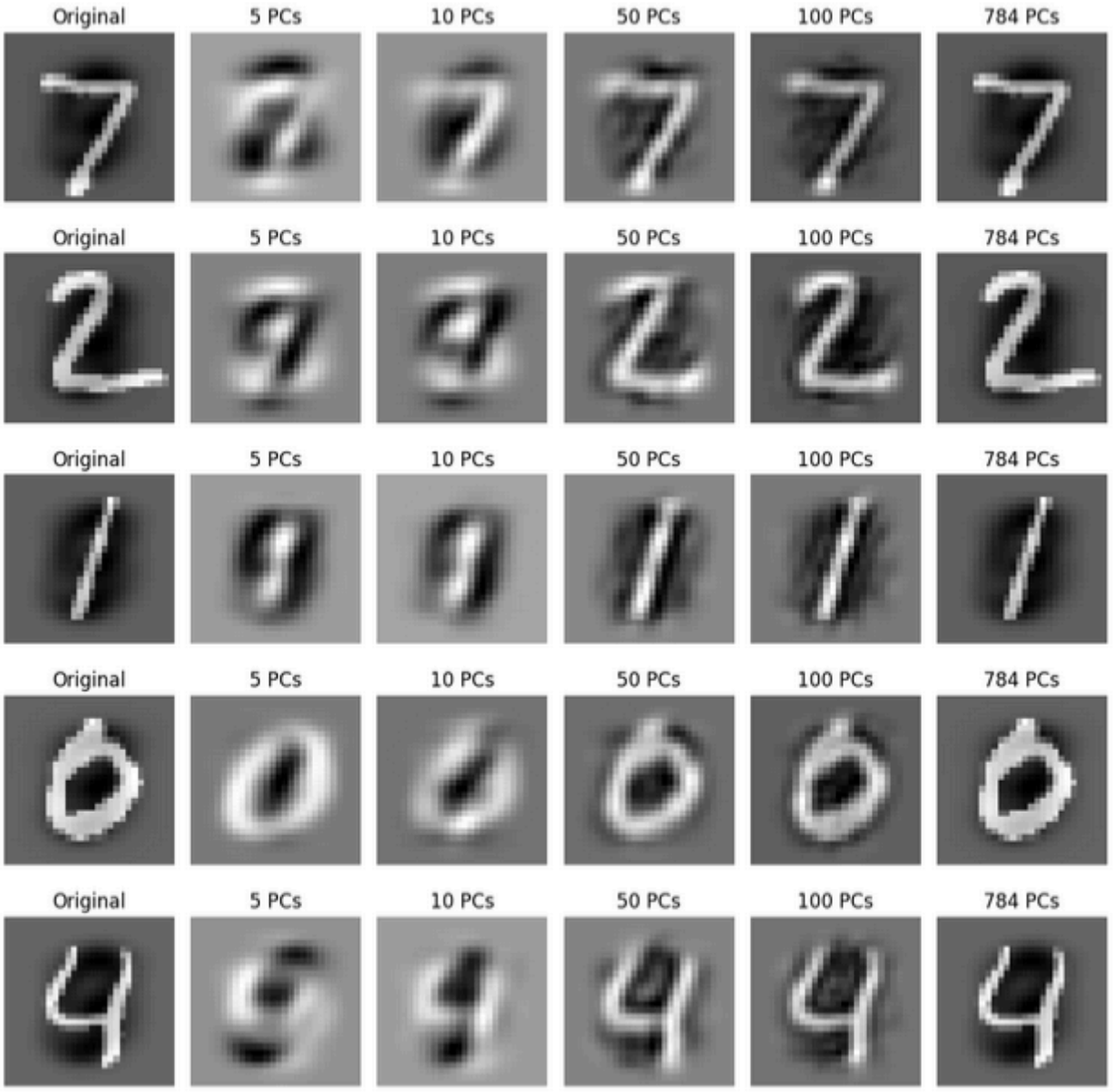


**Figure 3.**

The reconstructed images from these principal components demonstrated that PCA effectively captured the core structure of the data. While some fine-grained details were lost, the dominant patterns and overall visual representation of the images were preserved (for sufficiently high numbers of components). This aligns with the goal of PCA in focusing on global features critical for recognition tasks while discarding noise and less informative details.

Subsequently, the reduced-dimensional data was used for classification tasks.

We used 60000 of standard MNIST training images to train Feedforward Neural Network. Then another 10000 images were used for testing. The FNN of the following structure was used for the classification:

```
network = models.Sequential([
layers.Input(shape=(n_components,)),
layers.Dense(256, activation='relu'),
layers.Dropout(0.25),
layers.Dense(128, activation='relu'),
layers.Dense(64, activation='relu'),
layers.Dense(10, activation='softmax')
])
```

**Figure 5.**

This neural network is a **fully connected feedforward neural network** designed to perform classification tasks. It takes a reduced representation of the data (after applying PCA) as input and predicts the class of the input image.

### 3.3 Layers of Fully Connected Feedforward Neural Network

The network has the following components:

**Input Layer**

- **Input Shape**: (n_components,)
    - This layer accepts a vector of size n_components, which corresponds to the number of principal components retained after PCA.
    - Each input represents a flattened version of an image (e.g., MNIST digits reduced to a lower-dimensional space using PCA).

**First Hidden Layer**

- **Layer Type**: Dense (fully connected)
- **Number of Neurons**: 256
- **Activation Function**: ReLU (Rectified Linear Unit)

**Purpose**:

- This layer learns 128 features from the input data.
- ReLU introduces non-linearity, which allows the network to model complex relationships in the data.
- Each neuron computes a weighted sum of the inputs, adds a bias, and applies the ReLU activation function.
- ReLU helps avoid issues like vanishing gradients, making training faster and more effective.

**Second Hidden Layer**

- **Layer Type**: Dense (fully connected)
- **Number of Neurons**: 128
- **Activation Function**: ReLU

**Purpose**:

- This layer refines the features learned by the first hidden layer and condenses the information into 64 dimensions.
- Similar to the first hidden layer, it applies the ReLU activation function to introduce non-linearity.

**Third Hidden Layer**

- **Layer Type**: Dense (fully connected)
- **Number of Neurons**: 64
- **Activation Function**: ReLU

**Purpose**:

- This layer refines the features learned by the first hidden layer and condenses the information into 64 dimensions.

- Similar to the first hidden layer, it applies the ReLU activation function to introduce non-linearity.

**Output Layer**

- **Layer Type**: Dense (fully connected)
- **Number of Neurons**: 10
- **Activation Function**: Softmax

**Purpose**:

- This layer outputs a vector of size 10, corresponding to the 10 possible classes in the MNIST dataset (digits 0–9).
- The **softmax activation function** ensures that the output values represent probabilities and sum to 1:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{10} e^{z_j}}$$

- The class with the highest probability is selected as the predicted class.

Here's a concise explanation of what we did:

- **Compiled the Model**:
  - Defined the optimization algorithm (**Adam**) to adjust weights during training.
  - Specified the loss function (**categorical crossentropy**) to measure the difference between predicted and actual labels for multi-class classification.
  - Added an evaluation metric (**accuracy**) to track model performance during training.
- **Trained the Model**:
  - Used the training data (`x_train` and `y_train`) to adjust weights over **18 epochs**.

- Set a **batch size** of 128 to process the data in smaller chunks for efficient learning.
- Included validation data (`x_test` and `y_test`) to monitor the model's performance on unseen data after each epoch.

The results of training by the 18 epochs are shown on the Figure 6 below:

```
Epoch 1/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.8013 - loss: 0.6662 - val_accuracy: 0.9572 - val_loss: 0.1375
Epoch 2/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9495 - loss: 0.1669 - val_accuracy: 0.9694 - val_loss: 0.0997
Epoch 3/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9620 - loss: 0.1234 - val_accuracy: 0.9728 - val_loss: 0.0861
Epoch 4/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9690 - loss: 0.1020 - val_accuracy: 0.9764 - val_loss: 0.0743
Epoch 5/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9706 - loss: 0.0894 - val_accuracy: 0.9745 - val_loss: 0.0774
Epoch 6/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9748 - loss: 0.0824 - val_accuracy: 0.9800 - val_loss: 0.0644
Epoch 7/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9767 - loss: 0.0740 - val_accuracy: 0.9790 - val_loss: 0.0668
Epoch 8/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9775 - loss: 0.0676 - val_accuracy: 0.9797 - val_loss: 0.0603
Epoch 9/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9795 - loss: 0.0632 - val_accuracy: 0.9821 - val_loss: 0.0596
Epoch 10/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9804 - loss: 0.0596 - val_accuracy: 0.9815 - val_loss: 0.0595
Epoch 11/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9819 - loss: 0.0535 - val_accuracy: 0.9820 - val_loss: 0.0611
Epoch 12/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9839 - loss: 0.0507 - val_accuracy: 0.9809 - val_loss: 0.0639
Epoch 13/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9845 - loss: 0.0472 - val_accuracy: 0.9809 - val_loss: 0.0639
Epoch 14/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9847 - loss: 0.0452 - val_accuracy: 0.9813 - val_loss: 0.0625
Epoch 15/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9856 - loss: 0.0427 - val_accuracy: 0.9830 - val_loss: 0.0573
Epoch 16/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9866 - loss: 0.0416 - val_accuracy: 0.9818 - val_loss: 0.0603
Epoch 17/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9862 - loss: 0.0423 - val_accuracy: 0.9831 - val_loss: 0.0589
Epoch 18/18
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9862 - loss: 0.0399 - val_accuracy: 0.9845 - val_loss: 0.0565
Час виконання: 43.98 секунд
```

**Figure 6.**

The results showed a model trained on the PCA-reduced data (with 32 components) achieved classification accuracy **98%**. This is comparable with accuracy obtained by applying CNN to non-compressed images (99%).

Confusion Matrix for 32 Components (and layers.Dropout(0.25)) presented in Figure 7:

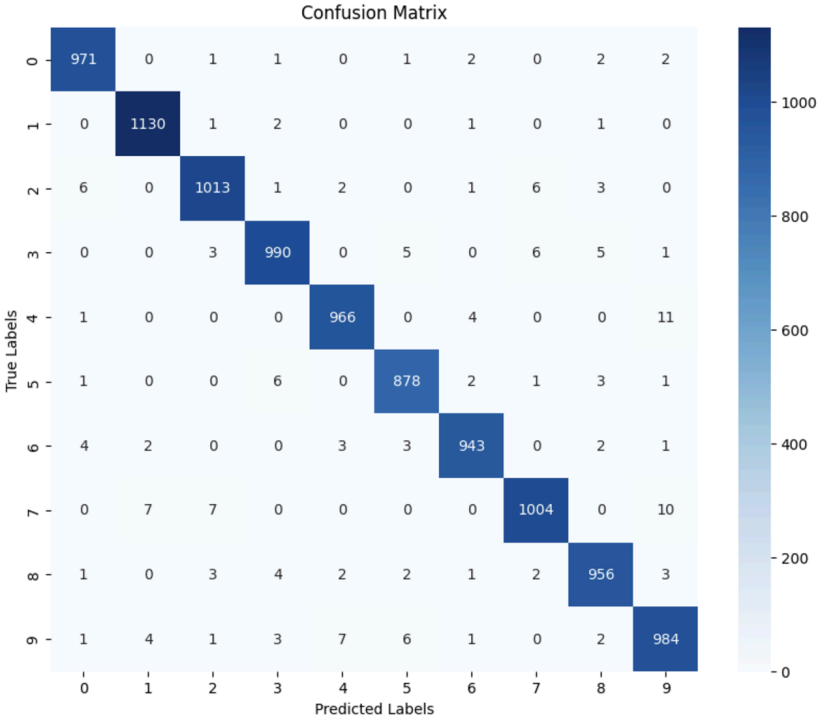


**Figure 7.**

**Table 3.** Accuracy based of number of PCA components

| Number of PCA components | Validation Accuracy |
|---:|---:|
| 2 | 0.48 |
| 4 | 0.66 |
| 8 | 0.92 |
| 16 | 0.97 |
| 32 | 0.98 |
| 50 | 0.99 |
| 64 | 0.992 |
| 128 | 0.992 |
| 256 | 0.992 |

### 3.4 Batch Size Influence on Accuracy

**Table 2.** Different batch size influence on accuracy (for 10 epochs)

| Batch Size | Validation Accuracy | Seconds to train |
|---|---|---|
| 8 | 0.9766 | 329 seconds |
| 128 | 0.9829 | 42 seconds |
| 1024 | 0.9746 | 17 seconds |
| 2048 | 0.9722 | 17 seconds |
| 10000 | 0.9329 | 12 seconds |
| 60000 | 0.6835 | 14 seconds |

Smaller batch sizes have been shown to enhance the performance of machine learning models, particularly in terms of generalization and convergence. This phenomenon can be attributed to several key factors:

**Stochastic Gradient Descent (SGD) Effect:**

- **Noise Injection:** Smaller batches introduce noise into the gradient updates, enabling the model to escape suboptimal local minima and explore a wider range of solutions.
- **Enhanced Generalization:** This stochasticity promotes better generalization by preventing the model from overfitting to the training data.

**Generalization Challenges with Large Batches:**

- **Precise but Limited Gradients:** Larger batches yield more precise gradient estimates but lack the variability necessary for diverse exploration.
- **Sharp Minima:** This can lead to convergence to sharp minima, which often exhibit poor generalization performance.

**Frequency of Gradient Updates:**

- **Accelerated Learning:** Smaller batches allow for more frequent weight updates, accelerating the learning process.
- **Slower Convergence:** Larger batches, with fewer updates per epoch, can hinder convergence.

**Regularization Effect:**

- **Implicit Regularization:** The inherent noise in smaller batches acts as a form of implicit regularization, mitigating overfitting.
- **Reduced Regularization:** Larger batches reduce this noise, potentially increasing the risk of overfitting.

**Computational Considerations:**

- **Memory Efficiency:** Smaller batches require less memory, making them suitable for training on large datasets or with limited hardware resources.
- **Precision Trade-offs:** Large batches may force compromises in floating-point precision or introduce instability in the optimization process.

**Trade-offs of Large Batches:** While large batches offer computational efficiency and stable gradient estimates, they can compromise generalization performance and convergence speed.

**General Recommendations:**

- **Prioritize Smaller Batches:** For optimal generalization, smaller batch sizes (e.g., 32 or 64) are often preferred.
- **Warmup Strategy:** Consider using larger batch sizes initially to accelerate convergence, followed by smaller batches for fine-tuning.
- **Adjust Learning Rate:** The learning rate should be adjusted in proportion to the batch size to maintain optimal convergence.

By carefully considering these factors and employing appropriate techniques, we can effectively leverage batch size optimization to improve the performance of their machine learning models.

### 3.5 Misclassification analysis

In Figure 8 examples of images that were misclassified are presented:

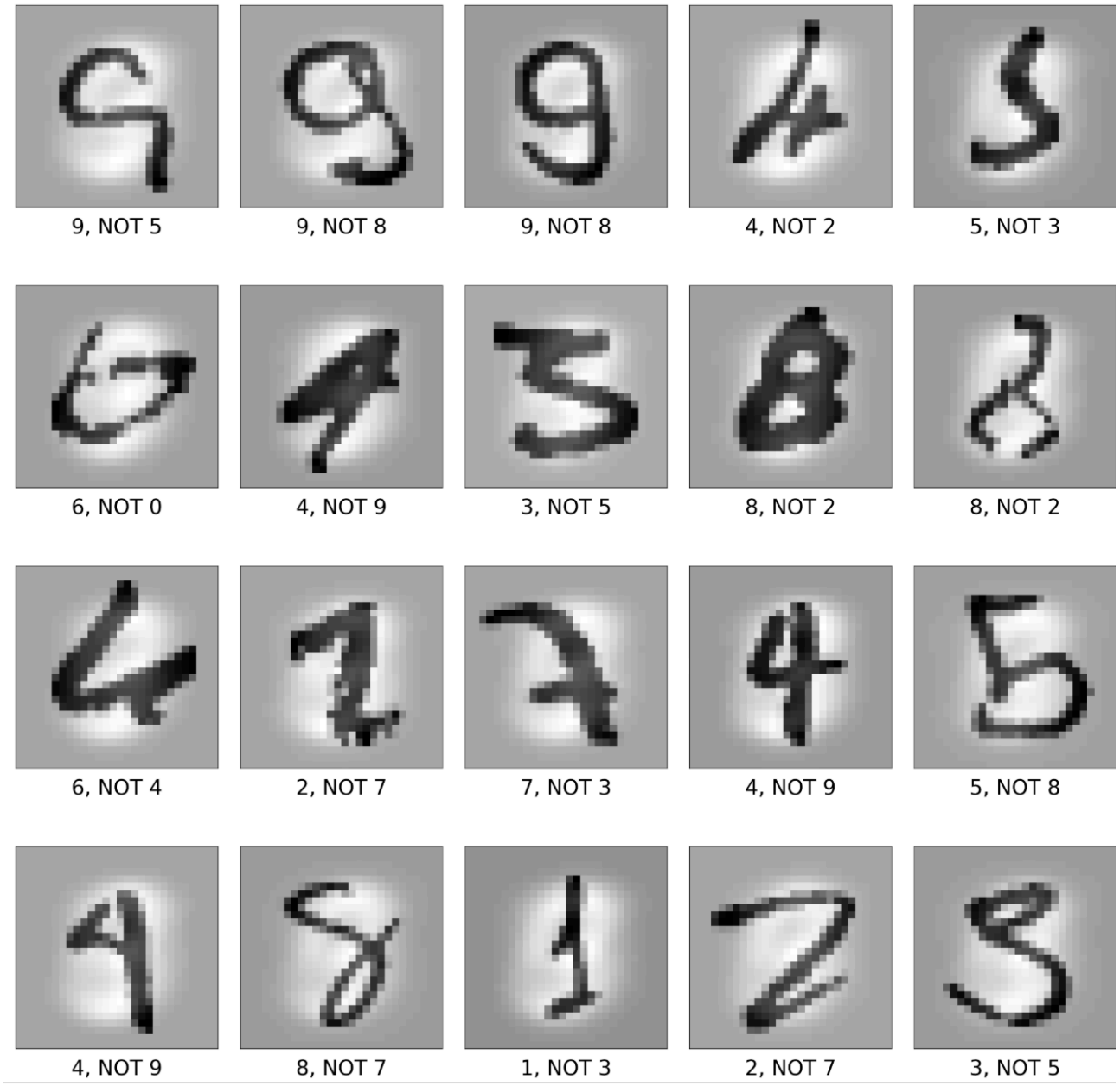


**Figure 8.**

Analyzing possible reasons we can consider such:

1. The width of lines matters
2. Context matters as it might be easier to identify if we have more instances of the same digit written by the same person. We could learn from the same digits written more legibly.
3. Some digits are really difficult to identify even for humans, especially if there are no labels underneath like in Figure 9:

**Figure 9.**

# CHAPTER 4. APPLICATION OF PCA TO RECOGNITION OF GALAXIES' IMAGES

For the experimental investigation we took 91481 color images of galaxies in png format from the Galaxy Zoo Decals database [9] for which the classification results by volunteer scientists were available. Each image is a vector with 424×424×3 coordinates.

Examples of images of different types of galaxies are shown in Figure below:

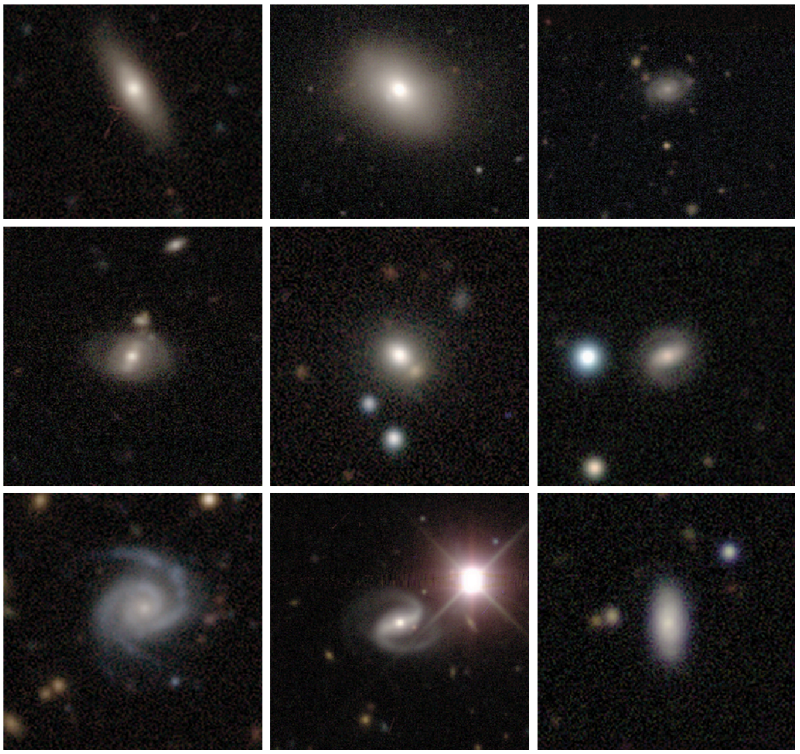

**Figure 9:** Example of images from Galaxy Zoo Decals database

In order to estimate the adequate number of information components that can be used to represent the image data using manifold learning, we applied the PCA method (which allows us to conveniently estimate information loss) to images of all types of galaxies. The result is shown in Figure 2. It shows that 99% of the information is retained with 32 components.

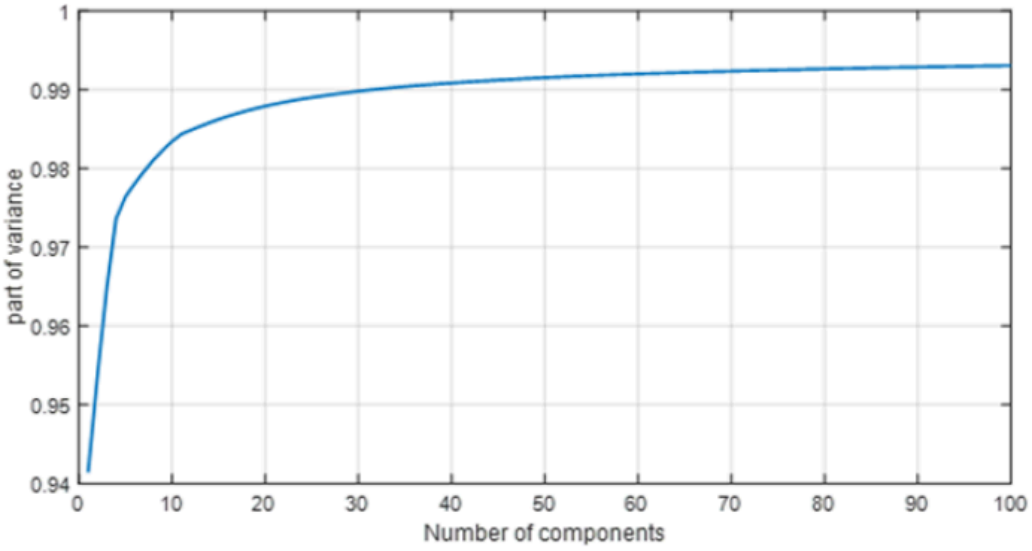


**Figure 10:** Dependence of the amount of preserved information on the number of information components.

The first 10 eigenimages are shown in Figure below:

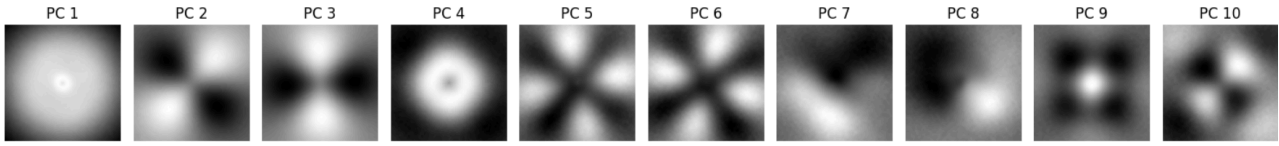


**Figure 11.**

Next we consider an example for the classification of galaxies onto “round”, “in between” and “cigar” types features (see Figure 12). A total of 8596 galaxies were used in this experiment (2690 ”round”, 4455 ”in between”, 1451 ”cigar”).

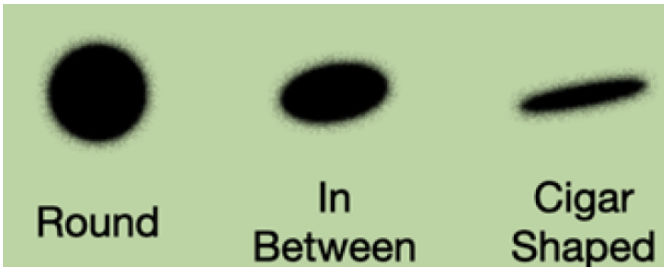


**Figure 12:** Explanation for “round”, “in between”, “cigar” galaxies.

The galaxies for which not less than 20 scientists voted and one decision received not less than 80% of votes were selected. 50% of the images were used for training and 50% for independent testing. The PCA method was used to reduce the

dimensionality of the images. Then the 3-layer FNN was used for the classification. The structure of this network is shown on Figure 13 below.

```
# Define the model
network = models.Sequential()
network.add(layers.Input(shape=(n_components,)))         # Input layer with shape matching the flattened image
network.add(layers.Dense(256, activation='relu'))  # First hidden layer with 256 units and ReLU activation
network.add(layers.Dropout(0.3))
network.add(layers.Dense(128, activation='relu'))   # Second hidden layer with 128 units and ReLU activation
network.add(layers.Dropout(0.2))
network.add(layers.Dense(64, activation='relu'))   # Second hidden layer with 64 units and ReLU activation
network.add(layers.Dense(3, activation='softmax')) # Output layer with 3 units (one for each class)
```

**Figure 13:** Structure of FFNN used to classify images of the galaxies.

Using only 64 PCA components we achieved classification accuracy of 95%. The same result was obtained by CNN multilayer Convolutional Neural Network (CNN) with 9 layers and 2861083 parameters which is specifically designed for image recognition.

**Table 4.** Accuracies for other number of PCA components

| Number of PCA | Accuracy |
|---|---|
| 2 | 0.78 |
| 8 | 0.94 |
| 16 | 0.95 |
| 64 | 0.95 |
| 256 | 0.9463 |
| 512 | 0.94 |
| 1024 | 0.93 |

In conclusion, the application of PCA in this study demonstrated its capability to effectively reduce dimensionality in image recognition tasks, maintaining high accuracy while improving computational efficiency. The results confirmed that PCA provides a powerful framework for extracting informative features from

high-dimensional data, making it a valuable preprocessing step in image recognition pipelines.

In this analysis, three types of galaxy classifications are used: **Shape**, **Disk Edge Orientation**, and **Featured Characteristics**.

### 4.1 Shape Classification

Galaxies can be classified by their shape into three main categories as mentioned above:

- **Round**: These galaxies typically have a spherical or ellipsoidal shape, often referred to as elliptical galaxies. Their appearance suggests a smooth, featureless structure without defined spiral arms or intricate details.
- **In Between**: These galaxies exhibit characteristics that fall between elliptical and spiral galaxies. They might show some structure but are not as clearly defined as the typical round or cigar shapes.
- **Cigar**: These galaxies are elongated and resemble the shape of a cigar, usually representing galaxies with an elliptical or lenticular form but with a more pronounced elongation.

**Accuracy**: The classification of galaxies by shape achieved a high validation accuracy of **0.96**, indicating the model's strong performance in distinguishing between round, in-between, and cigar-shaped galaxies.

### 4.2 Disk Edge Orientation Classification

This classification focuses on whether the galaxy is oriented such that its disk is seen edge-on or face-on. The two classes in this category are:

- **Disk Edge On: Yes**: This refers to galaxies where the disk is oriented in such a way that we are observing it from the edge, giving it a thin, elongated appearance.

- **Disk Edge On: No**: This refers to galaxies that are viewed face-on or at an angle where the disk is not seen edge-on.

**Accuracy**: The recognition of disk edge orientation achieved a validation accuracy of **0.89**, indicating that the model performs well but could benefit from further improvement, especially in cases with ambiguous or tilted orientations.

### 4.3 Featured Classification

Galaxies can also be classified based on their surface features, such as their smoothness or the presence of distinct features like spiral arms, star formation regions, or bulges.

- **Featured Galaxy** (labeled 0): These galaxies exhibit distinct features, such as spiral arms or irregularities in their structure, which give them a more intricate and well-defined appearance.
- **Smooth Galaxy** (labeled 1): These galaxies have a more homogeneous structure, with fewer prominent features or irregularities. They tend to appear more uniform and are often classified as elliptical galaxies.

**Accuracy**: The classification of featured vs. smooth galaxies achieved a high validation accuracy of **0.96**, demonstrating the model's effectiveness in distinguishing between these two types based on their surface characteristics.

### 4.4 Summary of Results:

- **Galaxy Type Recognition (Shape)**: Accuracy = **0.96**
- **Galaxy Type Recognition (Disk Edge)**: Accuracy = **0.89**
- **Galaxy Type Recognition (Featured)**: Accuracy = **0.96**

These results indicate that the classification model is highly effective for shape and feature-based recognition tasks but slightly less accurate in detecting the orientation of galaxies' disks. This suggests that further fine-tuning or additional data may improve the disk edge classification model.

# CHAPTER 5. SOFTWARE SOLUTION

## 5.1 Architecture

The proposed software architecture is composed of two distinct yet tightly integrated components to ensure modularity, scalability, and efficiency:

1. **SvelteKit Application**: This serves as both the frontend and web backend, typically deployed using Node.js. It is designed to be cloud-ready, supporting seamless scalability. The application provides a responsive user interface and incorporates backend logic for request validation and system interaction.
2. **FastAPI Service**: A Python-based microservice that handles computationally intensive tasks, including image preprocessing, dimensionality reduction, and classification. This component serves as the core engine for both neural network (NN) training and inference.

The architecture leverages containerization via Docker, with orchestration provided by CapRover. The containers communicate over a secure private network, ensuring efficient data exchange and straightforward deployment across diverse environments.

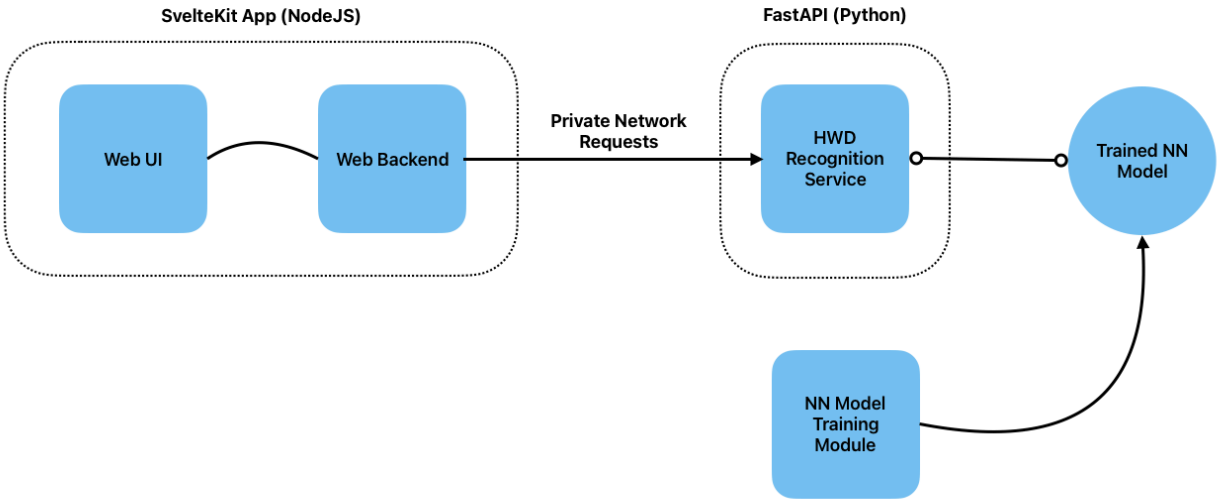


**Figure 14.**

This design reflects a robust and modular architecture, well-suited for scalable deployment and integration into real-world applications [10].

### 5.2 Data Modeling

The software employs a robust data model to manage the flow of information throughout the system, focusing on the following key elements:

- **Input Data**: Raw image files in widely used formats such as PNG or JPEG, uploaded by users or retrieved from datasets.
- **Preprocessed Data**: Images transformed and reduced in dimensionality using Principal Component Analysis (PCA), retaining only the most critical features.
- **Model Training Data**: Feature vectors extracted from preprocessed images, annotated with labels for supervised learning tasks.
- **Recognition Results**: Output in JSON format, containing classification predictions alongside confidence scores and other metadata.

### 5.3 C4 Representation

If represented through the C4 model, the system can be outlined as follows:

- **Context**: A web-based platform designed for image recognition, facilitating interaction between end-users and the backend services.
- **Container**: Two primary containers—SvelteKit for the frontend and FastAPI for the backend—communicating securely within a private network.
- **Component**: Modularized frontend and backend functionalities, including user validation, dimensionality reduction logic, and classification algorithms.
- **Code**: Highly modular Python scripts for NN implementation and feature extraction, alongside reusable components in SvelteKit for user-facing functionality.

### 5.4 Deployment Views

The deployment model is tailored for flexibility and cloud compatibility:

- **Frontend Deployment**: The SvelteKit application is containerized, ensuring seamless integration with CapRover for deployment. This component serves as the user-facing interface and backend logic handler. Drawing of digits allows testing the model right in the browser.
- **Backend Deployment**: The FastAPI service is likewise containerized, allowing isolated execution of recognition workflows and scalable performance.
- **Orchestration**: Both containers are managed through CapRover, simplifying the deployment process and ensuring secure inter-component communication.
- **Cloud Readiness**: The Docker-based design supports deployment on any major cloud provider, including AWS, Azure, and GCP, with minimal configuration.

### 5.5 Chosen Algorithms and Neural Network

**Dimensionality Reduction:**

- **Principal Component Analysis (PCA)**: Used to reduce the dimensionality of image datasets while preserving their most significant features. This technique

improves computational efficiency and enhances the performance of downstream classification tasks.

**Neural Network:**

- **Custom Neural Network**: Designed for specific tasks such as handwritten digit recognition and 3 types of galaxy classification. These models are trained on datasets optimized using PCA, reducing noise and redundancy.
- Different neural networks are designed to address specific tasks. For galaxy classification, the network architecture includes additional layers to handle the increased complexity and subtle variations within the dataset, ensuring accurate and nuanced classification.

### 5.6 Innovative Parts

The software introduces several innovative elements:

1. **Advanced Dimensionality Reduction**: Seamless integration of PCA into both the preprocessing pipeline and training workflow, enhancing both computational efficiency and classification accuracy.
2. **Misuse Prevention Mechanism**: A validation layer within the SvelteKit backend identifies if requests originate from legitimate human users or bots, safeguarding the FastAPI service from exploitation.
3. **Real-Time Interaction Capabilities**: The modular architecture supports the transition from HTTP to WebSocket communication, enabling live recognition in future iterations.

### 5.7 Technical Stack Rationale

- **SvelteKit**: Selected being modern, reactive framework and its capability to provide server-side rendering (SSR), which improves performance and compatibility.

- **FastAPI**: Offers a high-performance backend with native support for asynchronous operations, making it an ideal choice for AI and machine learning tasks.
- **Python**: Widely regarded as the standard for NN development, leveraging powerful libraries such as TensorFlow, PyTorch, and scikit-learn.
- **Docker + CapRover**: Ensures portability, scalability, and streamlined deployment processes across diverse server and cloud environments.

### 5.8 Deployment Documentation

1. **Prerequisites**:
    - Ensure Docker is installed on the server.
    - Configure and run CapRover.
    - Prepare image datasets for deployment.
2. **Deployment Steps**:
    - Build Docker images for both the SvelteKit and FastAPI components.
    - Push the images to a container registry if deploying remotely.
    - Configure applications in CapRover:
        - Deploy the SvelteKit container, setting up relevant environment variables.
        - Deploy the FastAPI container, linking it to the private network for secure communication.
    - Use CapRover's built-in tools to set up domain routing and enable SSL.
3. **Verification**:
    - Access the frontend to verify proper deployment and functionality. [11]
    - Test backend APIs to ensure correct operation of image recognition workflows.
    - Perform end-to-end testing to validate seamless interaction between components.

# CONCLUSION

PCA plays a crucial role in simplifying the complexity of image data for recognition tasks. By reducing dimensionality, PCA makes image recognition more efficient, improves computational performance, and helps algorithms focus on the most relevant features.

Developed an approach to recognize images using PCA, achieved a significant reduction in dimensionality while maintaining the level of accuracy. Even 32 PCA components are sufficient to get 98% accuracy of image classification Achieved 25 times less parameters (32 instead of 784) in the neural network and reducing the time for training the network (~100 times comparing to CNN)

Developed a software product for HWD classification and galaxy classification by 3 different criterias. High recognition accuracy is achieved (~95%).

# REFERENCES


1. Jolliffe, I. T. (2002). Principal Component Analysis. Springer Series in Statistics.
2. Turk, M., & Pentland, A. (1991). Eigenfaces for Recognition. Journal of Cognitive Neuroscience.
3. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
4. Deep Learning with Python, François Chollet, 2017.
5. Slyusar, V., Protsenko, M., Chernukha, A., Kovalov, P., Borodych, P., Shevchenko's., Chernikov, O., Vazhynskyi, S., Bogatov, O., & Khrustalev, K. (2021). Improvement of the model of object recognition in aero photographs using deep convolutional neural networks. Eastern-European Journal of Enterprise Technologies, 5(2 (113), 6–21. https://doi.org/10.15587/1729-4061.2021.243094
6. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin (2017). Attention Is All You Need, https://doi.org/10.48550/arXiv.1706.03762
7. R. Ramachandran, G. Ravichandran and A. Raveendran, "Evaluation of Dimensionality Reduction Techniques for Big data," 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2020, pp. 226-231
8. Yann LeCun (1998). THE MNIST DATABASE of handwritten digits https://yann.lecun.com/exdb/mnist/
9. Galaxy Zoo Decals database, https://zenodo.org/records/4573248
10. Capstone Project Source Code, https://github.com/avhust/sveltecap
11. Capstone Project Live, https://capstone.avhust.com

# FUTURE DIRECTIONS

This project opens several paths for further exploration and application. One significant direction is the development of user-friendly tools that enable scientists to efficiently process and analyze their datasets (not only galaxies images). By incorporating dimensionality reduction techniques such as PCA into intuitive software interfaces, researchers can manage high-dimensional data more effectively, unlocking new insights across various scientific domains.

Another promising application lies in bot detection systems. Leveraging the developed human detection (with HWD) software, future implementations could enhance security frameworks by verifying human presence as proof against automated bots. This approach holds potential for online authentication systems, CAPTCHA replacements, and fraud prevention measures.

Additionally, the software developed during this project can serve as a foundational platform for training new machine learning models. By integrating PCA into the preprocessing pipeline, the training process can be optimized, reducing computational costs and improving model generalization. This capability is particularly valuable for large-scale datasets, ensuring scalable and efficient solutions for future image classification challenges.