KØBENHAVNS UNIVERSITET

Facial Recognition App in F# - A Functional Journey

Tim Mondorf CJK681

Project Outside of Course Scope (PUK)

1st year of BSc in Machine Learning and Data Science

Supervisor: Professor Jon Sporring

Datalogisk Institut. 10. august 2022

1 Abstract

A 5-nearest neighbor model was applied to the MNIST-data set with a 92 per cent accuracy. The data set was then reduced to its 10 principal components using Principal Component Analysis (PCA), after which the 5-nearest neighbor model was able to predict with almost no loss of accuracy, 87 per cent. For visualization purposes, a model of only two principal components was developed. It was shown that the observations clustered according to labels in the two-dimensional plot. A 5-nearest neighbor model was applied to the Pins-data set with face photos of celebrities and performed disappointingly with 24 per cent accuracy.

It was possible to perform this analysis by coding the relevant functions in F#, adhering strictly to the functional programming paradigm. This included functions coded by the student himself for scaling, cropping and recentering bitmap images. This meant no loops or mutable variables anywhere in the code, with only one exception in 900 lines of code.

A set of expected benefits from this functional approach were defined: a well-structured software that was easy to write and debug, clearer and simpler analysis of how to build an image recognition app, increased proximity to the underlying mathematics, better understanding of functional programming, good management of input data, an approach that could scale to a large data set, more routine for the student in programming in F#, better understanding of the alternative, imperative and object-oriented programming paradigm and experience of a prolific ecosystem of modules. As documented in a meticulous research diary covering a 6-month period, these benefits were to a large extent achieved in practice.

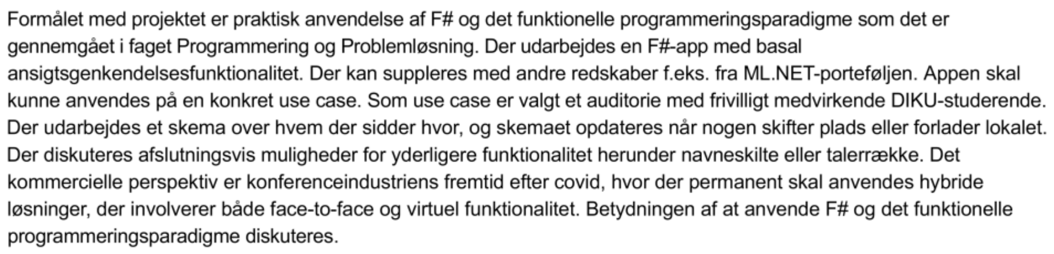
2 Preface

This project is a project outside of course scope as defined in Den fælles del af bachelor- og kandidatstudieordningerne for uddannelserne ved Det Natur- og Biovidenskabelige Fakultet Københavns Universitet [12].

The intended readership of the paper is teachers and other students on 1st year of Machine Learning and Data Science or Computer Science. This means that the discussion and examples should be approachable for readers with this level of knowledge and that only basic insight into facial recognition and its application is required. Throughout the paper, it has been attempted to illustrate the programme output by diagrams or examples of pictures.

I would like to thank my supervisor, professor Jon Sporring, for supervision and many good discussions as detailed in the Research Diary.

Indhold


Formålet med projektet er praktisk anvendelse af F# og det funktionelle programmeringsparadigme som det er gennemgået i faget Programmering og Problemløsning. Der udarbejdes en F#-app med basal ansigtsgenkendelsesfunktionalitet. Der kan suppleres med andre redskaber f.eks. fra ML.NET-porteføljen. Appen skal kunne anvendes på en konkret use case. Som use case er valgt et auditorie med frivilligt medvirkende DIKU-studerende. Der udarbejdes et skema over hvem der sidder hvor, og skemaet opdateres når nogen skifter plads eller forlader lokalet. Der diskuteres afslutningsvis muligheder for yderligere funktionalitet herunder navneskilte eller talerrække. Det kommercielle perspektiv er konferenceindustriens fremtid efter covid, hvor der permanent skal anvendes hybride løsninger, der involverer både face-to-face og virtuel funktionalitet. Betydningen af at anvende F# og det funktionelle programmeringsparadigme diskuteres.

Figur 1: Problem statement as per contract 31 January 2022

3 Project objective

Figure (1) shows the objective for this project as defined in the original project contract, elaborated before the project was initiated in February 2022. Over the course of the project, the project objective did evolve. It was decided to focus less on the app functionality and more on demonstrating functional programming in the specific case, defining expected benefits from functional programming and determining whether these benefits were achieved in practice. The reason for this evolution was primarily time to execute the project and the fact that even simple data sets such as MNIST and Pins lent themselves well to illustrate the chosen concepts, whereas a real-life test of a conference app would have involved issues such as test persons, GDPR-related consent and porting the programme code to a mobile app, all of which were highly relevant themes but outside of the first-year course scope.

3.1 Expected benefits from functional programming.

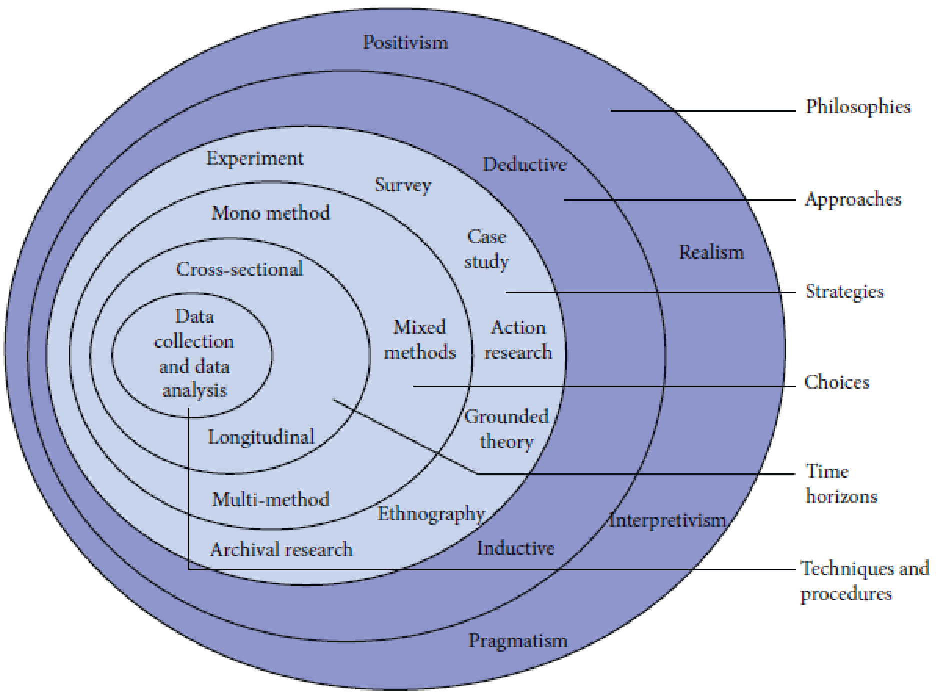
The project will examine whether developing an image-recognition app using functional programming will achieve the following benefits.

1. A well-structured software that is easy to write and debug
2. Clearer and simpler analysis of how to build an image recognition app
3. Increased proximity to the underlying mathematics
4. Better understanding of functional programming
5. Good management of input data
6. The app will be developed on small data set but the approach should scale to large data sets
7. More routine for the student in programming in F#
8. Better understanding of the alternative, imperative and object-oriented programming paradigm
9. Experience a prolific ecosystem of modules built for and by functional programmers

4 Methodology

The student has decided to define the scientific methodology using the Saunders Research Onion-approach [22] which is primarily used in social sciences, but which has also been applied to information technology for instance by Alturki [2]. In the research onion methodology, a research strategy for a project is defined by making successive choices, corresponding to peeling the layers of an onion from the outside in. One purpose of the research onion framework is to ensure that the appropriate data, which are defined in the core of the onion, are available for the strategy chosen. As a thesis supervisor at CBS, the student has had experience with the Research Onion.

In the first layer, pragmatism is adopted as the research philosophy. Where positivism is dependant on repeatable observations of a stable and constant reality, pragmatism deals with ideas or concepts that have practical consequences. It is deemed difficult to perform repeatable experiments of a programming paradigm either from a programmer or a user perspective. A less ambitious approach of



Figur 2: Research onion, developed by Saunders, as shown in Alturki 2021

studying the practical consequences for programmers, program output or users of different ideas in the form of programming paradigms is therefore chosen.

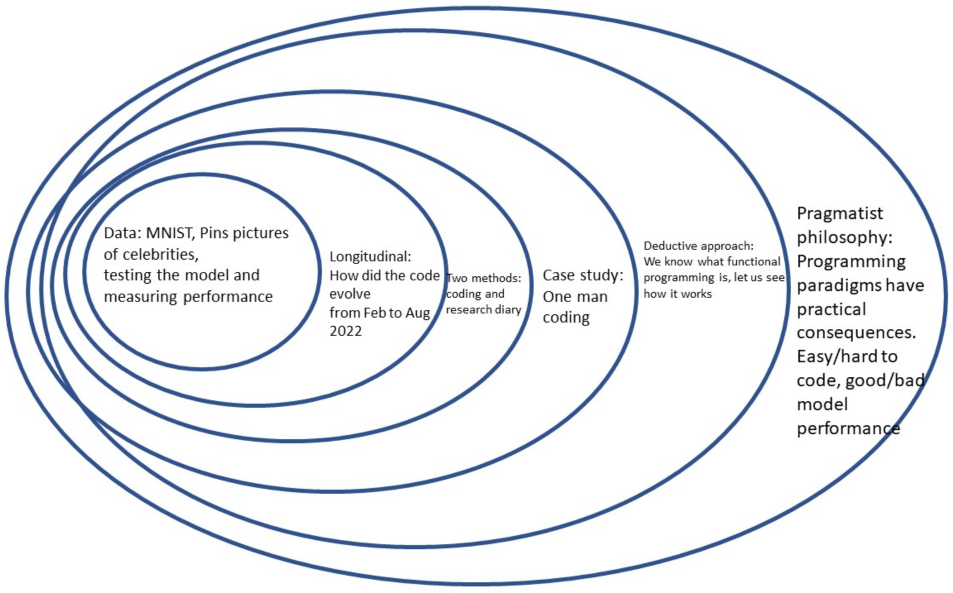
In the second layer, deduction is chosen as the research approach. Deduction moves from the general to the specific. The general idea here is what functional programming is and how it may have certain benefits when applied to image recognition. The specific is the experience of whether the benefits are realized in this particular case.

In the third layer, case study is chosen as a research strategy. The case is one of the student himself developing an app using functional programming. Where many case studies have the form of applying a theoretical framework to a case that already exists outside of the academic scope, this case is being performed as part of the project. Case studies have the benefit of allowing the understanding of a phenomenon more deeply in its context, but at the cost of being more difficult to generalize. Any general conclusions as to whether functional programming is well or less well suited for image recognition is specific to this student and this app and can not be considered generally valid. Case studies are however often very useful in a general context precisely because they provide deeper understanding and specific examples.

In the fourth layer, it is chosen to combine two methods. Firstly, programming code is developed in F# and applied to two data sets. The ease of programming is then discussed along with measurements of the model performance. Secondly, a research diary is maintained that records not just the final programme but how the different programme versions evolved and became closer to the functional paradigm. The research diary also records conversations with the supervisor. The research diary appears as part of this report, it was updated during the writing process and for reasons of data discipline has not been edited or proofread.

The choice of these two methods maps into the fifth layer where it is observed that the study has a longitudinal time-line, describing how the programming developed from February to August 2022 over various iterations. An alternative would have been a cross-section study, comparing two different apps or two different programmers.

The data collection consists of two data sets, first the MNIST-data set from Kaggle [1] which holds 42.000 different black-and-white images of hand-written digits. MNIST is often called 'The Hello World! of image recognition' because so many first-year students use it for simple models of image recognition. The second is the Pins-data set [9] with 17.534 facial photos of 105 celebrities. Since both data sets have clearly indicated labels, this is a process of supervised machine learning. The


Data: MNIST, Pins pictures of celebrities, testing the model and measuring performance
Longitudinal: How did the code evolve from Feb to Aug 2022
Two methods: coding and research diary
Case study: One man coding
Deductive approach: We know what functional programming is, let us see how it works
Pragmatist philosophy: Programming paradigms have practical consequences. Easy/hard to code, good/bad model performance

Figur 3: Research onion approach applied to this project, source: author, inspired by Saunders and Alturki

data collection also consists of the programming code in different iterations which is made available in a Github repository [16].

The research strategy as defined by the research onion is shown in Figure (3)

In sum, a research strategy has been defined that allows deep insight into how a first-year student applies functional programming to image recognition and whether by doing so he achieves certain predefined benefits. While care should be taken before generalizing the findings, it is also the expectation that the project will have a general relevance for the discussion of programming paradigms and more broadly of how to teach data science to a wider public.

5 What is functional programming?

The objective of this section is to define functional programming. Terminology and definitions can sometimes be mean different things in different contexts, and a literature search has therefore been performed in order to get as precise as possible an idea of what functional programming is. This again reflects the deductive approach, moving from a general idea to its manifestation in reality. The literature search has included sources such as Sporring [23], Boudreau [4], Loder [14] and Hu [10].

Most souces, such as Sporring [23], calls functional programming an example of the declarative programming paradigm or as Loder call functional a paradigm that is often, but not always, accompanied by a declarative style. In any event, the best place to start the definition would be to discuss the notion of a paradigm. A programming paradigm can be defined as an approach to programming. Boudreau [4] says that paradigms are a 'concept for the methodology of programming languages'. Wikipedia actually goes as far as to say that 'Many programming paradigms are as well known for the techniques they forbid as for those they enable' [26]. To pursue a kitchen metaphor, if the algorithm is the recipe and the programming language is the language in which the recipe is written, a paradigm would be a set of rules for how to behave in the kitchen, for instance whether the same utensils could be used for raw and processed foods. So whether functional is its own paradigm or a style within a paradigm, functional programming can be said to be a certain way in which programmers analyze, plan and execute their code.

Key in the definition of functional programming is the idea that programmes should evaluate functions and avoid state changes. This is mentioned by both Sporring [23], Hu [10] and Loder [14]. Loder [14] p. 3 ff devotes an entire section to the topic. Imperative programming is likened to a recipe for making bread that lists all ingredients as well as every step in the preparation, whereas functional programming is likened to a bread making appliance where all ingredients are input, then a specific bread-making programme is selected, but the user does not see the individual steps. Other characteristics that often appear in definitions of functional programming are the use of recursion [14], higher order functions meaning functions that call other functions, and lazy evaluation where functions and their arguments can be defined at first but not be evaluated until the result is required [10]. This is used in the handling of large input files. The F#-concept of sequence is an example of lazy evaluation. As expected from the above Wikipedia statement, many sources define functional programming by what it does not have: loops and destructive assignments into mutable data structures.

Pickering and Eason [20] provide a historical overview of the development of functional programming from IPL in 1955 over Lisp in 1958 to SML, Ocaml and APL. This student did try to programme in APL as an intern in Industrirådet in 1991 - 1993. The linear algebra-based approach lent itself well to considering macroeconomic data as matrices. For instance, a table of economic activity could be seen as a matrix with the various sectors (agriculture, manufacturing, services) as rows and each time period as an additional column of the matrix. The functional approach of APL served to perform macroeconomic forecasting on the matrices where an extra observation (column) was continually added and a new quarterly or monthly forecast was published. In spite of these benefits, APL was considered extremely difficult to use to the point where it was an actual knowledge monopoly in the organization. As part of the merger and creation of Dansk Industri (DI) in 1993, APL was replaced by spreadsheets which were much easier to use and could be used by a wider group of employees.

For the purpose of this project, and based on the above sources, the following definition of functional programming is established

- Use of functions
- Use of module calls or F# constant time-functions rather than loops
- Strong typing
- Use of lazy evaluation
- No mutable data structures
- No loops

The student has attempted to write his code in a functional style wherever possible and minimize the use of standard functions. Where standard functions such as the MathNet.Numerics are indeed used, the fact that this introduces a measure of object-orientation and limits the functional approach is discussed.

5.1 F# as a programming language.

From the choice of functional programming follows the choice of programming language. F# was taught in the first-year course of Programming and Problem Solving. Several sources such as Sporring [23], Pickering and Eason [20] and Syme [24] call F# a 'functional first'-programming language, meaning that it is designed to be functional but also supports imperative and object-oriented programming. Pickering [20] says that F# 'encourages' a functional syle with use of expressions that return a result rather than statements with side effects.

F# will be the programming language of choice for this project. F# and dotnet-based modules will be the first choice, and modules from the Python-based, 'object oriented'-ecosystem for data science and machine learning will only be the last resort.

5.2 A simple example of the choice between functional or object-oriented paradigms in in business applications.

Imagine that a Danish IT company wins a contract for an application that will keep track of vaccinations against Covid-19. Most companies would take an object-oriented approach, defining each citizen as an object with a number of attributes such as the number of times the person has been vaccinated and the vaccination dates. The object would then have methods, one method borger.vaccinate(dato) would augment the number of times the person was vaccinated. The object would be mutable, and the protection against errors would reside in a number of logical conditions, for instance that the person must not be vaccinated more than, e.g., twice and that there must 180 days between the first and the second vaccine. An object oriented approach would be centered around the citizen and being vaccinated would be a method attached to the object. Vaccination information would be object attributes.

Contrast this with a functional approach, where three custom record types are defined: citizenUnvaccinated, citizenOnceVaccinated and citizenTwiceVaccinated. Name, cpr and vaccination dates are attributes of the custom record types. Two functions are defined. vaccinateFirstTime takes as its input argument a citizenUnvaccinated and a date and outputs a citizenOnceVaccinated. vaccinateSecondTime takes as its input argument a citizenOnceVaccinated and - if the current date is 180 days or more since the first vaccination - outputs a new constant which is a citizenTwiceVaccinated with the name and cprNumber of the original citizenOnceVaccinated. If the current date is less than 180 days since the first vaccination date, a dummy record is output. Protection against error is here done via the strong typing, not via logical conditions. The vaccinateSecondTime cannot take as its input a person who is not vaccinated exactly once, and it can only output a person who is twice vaccinated. As a stateless function, it has no side-effects such as mistakenly increasing the vaccine count for an individual.

The choice between object oriented and functional will be a choice of identifying the parts of the business process that are the most critical. Developers and business owners will ask each other the question: 'What are the key success factors? Is it to keep track of the many people we are vaccinating, is there tremendous risk involved with vaccinating the wrong people or mismanaging individual vaccination dates? Or is the critical success factor to administer the vaccines, given that these need to be developed, tested, approved, produced, refrigerated and distributed in a never-before attempted roll-out?' If the latter is the case, a choice of functional programming will support a solution strategy that emphasizes support for the vaccine function (in its widest sense) and treats people receiving the vaccine as service recipients who are downstream of the most critical function. Discussions of programming paradigm will drive a discussion of business strategy.

Code examples of the two functions are shown in Appendix I.

5.3 Different ecosystems

The choice of paradigm, approach and programming language also leads to a choice of ecosystem. A number of resources exist supporting data science performed in F# available for instance on ML.Net, on the general dotnet platform and on independent sites such as https://fslab.org/. Accord.net and Accord.fsharp are examples of image analysis-modules from this ecosystem. All these resources have been reviewed as part of this project.

Today's dominant data science ecosystem centers around imperative programming with Python as a programming language and a number of frequently used modules such as pandas and numpy for analysis, cv2 for image recognition, and with MobileNet as one of the newest libraries for machine learning. For a new ecosystem to grow and become successful, not only must the underlying programming paradigm offer new benefits, but there must be a network effect involving a sufficient number of users and developers.

6 Why use functional programming for a facial recognition app?

Literature lists a large number of benefits that can be expected to follow from a choice of functional programming as approach and from F# as the programming language. This section attempts to produce a definition of these benefits that should be so precise that at then end of the project, it can be concluded whether these benefits were achieved or not.

For anyone who is trying to master a skill, and who performs it in a certain way, trying an alternative approach can provide new insights. Bjarne Stroustrup, the inventor of the $C++$ -language, in this interview with Lex Fridman [8] expressed how important it is that programmers do not limit themselves to one programming language, an argument that can also be applied to programming paradigms and approaches.

In 1990, John Hughes wrote the article 'Why Functional Programming Matters' and listed two benefits. Firstly, as software became more complex, structure would become even more important and this was where functional programming could provide 'well-structured software [that] was easy to write and to debug. Secondly, functional programming would create 'a collection of modules that can be reused to reduce future programming costs'. Today, the use of open-source modules is prevalent across all programming paradigms, but the emphasis that Hughes put on structure, ease of programming and ease of debugging are still among the main benefits of functional programming.

25 years later, Hughes reviewed progress in Hu, Hughes and Wang [10] and confirmed that one benefit of functional programming is that 'shorter and clearer code leads to improved development

productivity, higher quality and fewer bugs'. Between 1990 and 2015 the cost of the computer itself fell and the relative cost of the programmer and his time skyrocketed, justifying the emphasis on programmer productivity. Also Syme [24] stresses that F# is a modern choice for an enterprise environment with focus on productivity.

Pickering and Eason echo the list of benefits defined by the other sources and explicitly cite Hughes [11]. The elimination of the distinction between data and functions allows for a more natural solution to many problems and brings the written code closer to the underlying mathematics. In functional programming, the absence of mutable variables and side effects also reduces error potential.

Several sources also list benefits from the functional approach that are specific to data science. If 'data is the new oil', it makes sense to follow an approach that cannot dilute that data but which can only refine it into new constants, separated by strong typing. Lavers [13] used Kotlin to transform and recolor images, relying on strong typing to separate originals from manipulated images.

Ralf Herbrich, quoted in Mukherjee [17] describes how as an employee of Microsoft research around 2005, he witnessed the emergence of F# and saw in practice how it made machine learning code scripts much shorter. Herbrich sees F# as an important step in the development of machine learning.

Both Neumann [18], writing as early as 1994, Xu [28] and Wu [27] mention the chaining or piping of higher order functions that operate over simple data structures as a benefit for data science. When weights are updated in iterations of the model, they are not mutated but instead a new set of weights is output by the function. Functional programming also makes it easy to perform parallel processing, supporting extensive computing requirements. Xu describes the use of Clojure as a programming language and an extensive ecosystem of libraries based on Clojure.

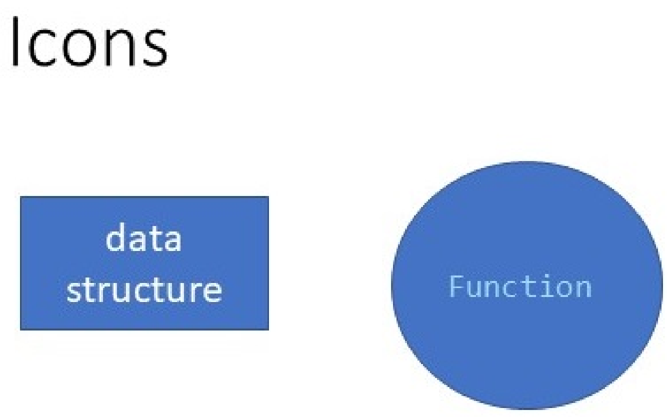
Lazy evaluation is mentioned by several, e.g., Xu [28] saying that it gives data scientists conceptual freedom by defining data structures that are in principle unlimited. Poole and Charleston [21], as early as 1993, found that when used for diagnostic imaging, the functional language Haskell met the requirements of being readable, mathematically based and efficiently executable, not least due to lazy evaluation. That said, the programme had to call external C modules to increase speed.

Gervàs [7] performs a comparative study of imperative, object-oriented programming done in Python and functional programming done in Haskell. Gervàs codes a random forest classifier and a convolutional neural networks classifier and compares the running time. The Python-based programme runs 1000 per cent faster for random forest and 50 per cent faster for convolutional neural networks and for both experiments requires two thirds less memory. Gervàs concludes that Haskell might still represent advantages in the form of strong typing, safety in the form of no side effects and safety when parallelizing code. Gervàs is a master student of a Spanish university and it is not clear if the experiment has been reproduced or peer-reviewed.

Mehmood [15] builds a neural network model in F# and states bluntly that '...functional programming [is] slow ...but gives a different perspective on any problem'. Mehmood actually shows that in practice, processing 3.000 epochs of his model took 3 seconds in F# but 168 seconds in Python on Google Colab. Mehmood uses the MathNet library, which will also be used for this project, which he notes provides the same functionality as (the python-based) numpy while being less straightforward.

Finally, a number of sources mention positve achievements from functional programming performed specifically within image recognition. Martin and Curtis [6] describe how a group of programming students were invited to use Haskell to develop the well-known Mandelbrot fractal images that can be developed through recursive programming. The study concludes that results in terms of student motivation were good not least due to the functional approach. For an illustration of the alternative, image processing performed in a C-based, object oriented approach, Bagdanov [3] describes user experience of the Horus module library for image processing. The point of the article is that the alternative to functional programming is not always easier, especially for entry-level users who found Horus quite challenging.

Having performed a literature survey, the set of expected benefits listed in the Project Objective above are defined. It is clear that these expected benefits are to some extent difficult to measure. There will be no controlled experiment where the same app is also coded in Python using the imperative paradigm and object-orientation. Objective 1 is probably measurable whereas objectives 2, 3 and 6 will depend on the subjective experience of the student. This informs the methodological choice of a pragmatist rather than a positivist philosophy. The approach is also similar to that taken by Martin and Curtis who emphasized how students came to understand the functional approach of seeing a picture as the input to or as the output from a function.



Figur 4: Explanation of symbols

7 Designing a facial recognition app through functional programming

7.1 Overview of the code

This section attempts to give an overview of the code and highlight the programming choices made, with the code itself being available as Appendix II as well as on Github.

As suggested by the supervisor, the code is expressed in flowcharts, where squares represent datastructures and circles represent functions as shown in Figure (4).

7.1.1 Writing programme output

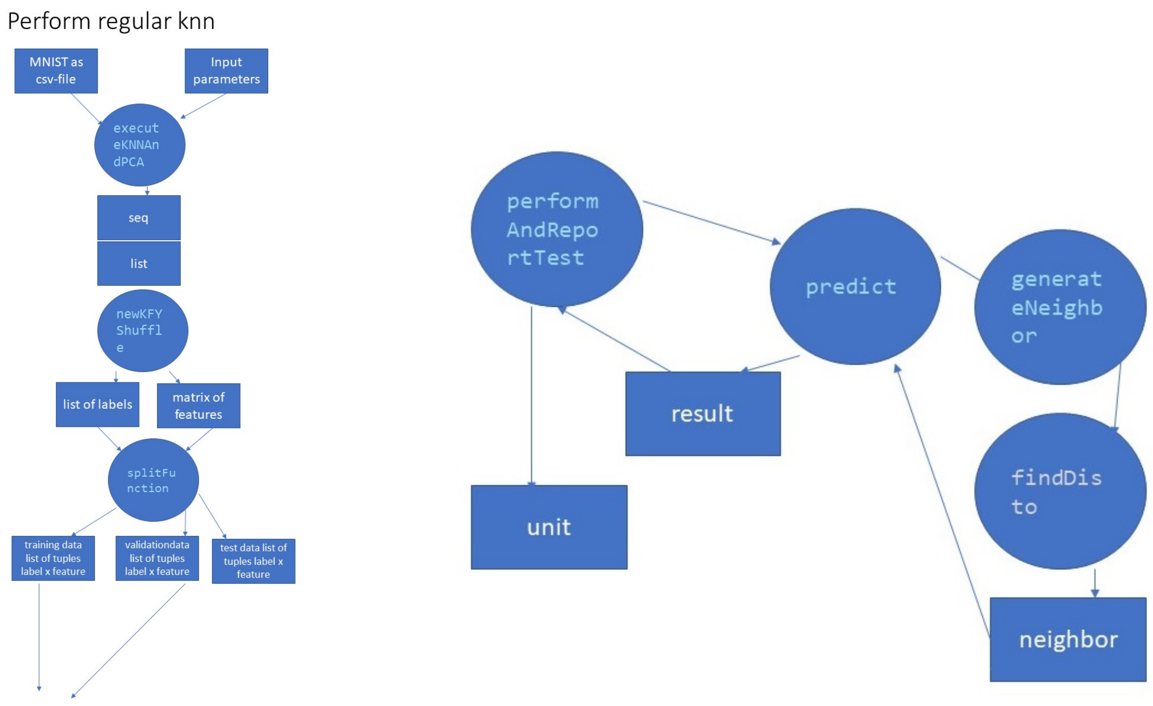
This is not shown in the flowcharts. A StreamWriter is defined that writes the programme output to a .txt-file. As input to the StreamWriter is defined a function that recursively reads a list of strings and combines them to one string, separated by spaces. This is an example of recursive programming.

7.1.2 Regular k-nearest neighbor on the MNIST-data set

Figure (5) shows the baseline model, a simple k-nearest neighbor (KNN) model performed on the MNIST-data set. The data set is read by the main function called performKNNAndPCA, along with the user defined input parameters such as how many neighbors to be used for prediction, here 5. The first iterations of the programme benefited from lazy evaluation, reading the first 5.000 elements from a sequence. For the final version, it was deemed more scientifically correct to extract 5.000 observations at random from the 42.000 observations in the full data set. This required shuffling which required transforming the sequence to a list. A list of 5.000 randomly selected observations is transformed into a list of strings representing the labels and a matrix, generated as an object by MathNet.Numerics.LinearAlgebra, where the pictures are rows and the features are columns. The list and the matrix are fed to the splitFunction, which randomly assigns them into train, validation and test data sets, the proportion of each again being set in the input parameters. A custom record type called pic is a tuple of a string label and a float list of features. Since the MNIST-bitmaps are (28x28), the feature lists are 748 elements long. Features (the Average pixel value of a given pixel) are conceptually integers between 0 and 255 but they are here defined as floats since F# can only uplift floats to the power of two, which is required for finding the Euclidean distance.

The training data set and the validation data set are two lists of pics. These two lists are passed to the higher order function performAndReportTest which calls the predict function. The predict function is also a higher order function, it reads a single observation from the validation data set, reads each observation from the training data set, passes them to the function generateNeighbor which calculates the Euclidean distance between the two in 748-dimensional space by calling the function findDisto, takes the label from the latter observation and creates a custom record type called neighbor which is the label and the distance.

The neighbor is passed back to the predict function which sorts all neigbhors according to distance, takes the 5 neighbors that have the lowest distance and finds the most recurring label among them



Figur 5: Flowchart for k-nearest neighbor algorithm

with ties being broken at random. For each observation in the validation data set, the predict function now has a result. result is a custom record type consisting of a tuple of actual and predicted value.

The list of results is passed back to performAndReportTest which determines whether results are correct (actual=predict) or not, and prints out the test results both generally and per label.

One example of functional programming is performAndReportTest generating the list of results using the List.init function

```
let (results:result list) =
    List.init (List.length testo)
    (fun i -> {actual=testo.[i].label; predicted=predict traino testo.[i] k})
```

testo is the list of test pics. for each test pic, there will be a result. List.init takes each list index and generates a result which is the tuple of the label of test image and the label that can be predicted from the test image features. In order to generate the prediction, the predict function is called using the entire training data set traino as its first argument and the single test observation as its second argument, with the required number of neighbors k (usually set to 5) as the final argument. This is a functional approach, passing results from one function to another.

The predict function takes the training data and the single test observation and initiates a list of neighbors in the following way

```
List.init (List.length trainPics) (fun i -> generateNeighbor pico trainPics.[i])
```

This generates a list that transforms each element in the training data set to a neighbor of the test observation in question, specifying its label and the distance to the test observation. That list is then piped into a sorting sequence that returns the most recurring label of the closed 5 training observations.

The generateNeighbor function calls the findDisto function to calculate the Euclidean distance. The two pics each have a list of floats representing the features which are processed by the List.map2-function. The features are called as attributes of the pic custom record type, and the resulting list is piped into first a list sum and the function that calculates the square root.

```
List.map2 (fun x y -> (x - y) ** 2.0) firstPic.features secondPic.features
|> List.sum
|> (fun x -> sqrt x)
```

The reporting function also calls List.init and List.filter on the list of results. For instance, to create a view of how the validation data set is distributed across labels, the following code is executed

```
let totalActualPerLab =
        List.init (List.length labels)
        (fun i -> List.filter (fun z -> z.actual = labels.[i]) results)
        |> List.map (fun listo -> List.length listo)
```

The code creates a list of lists, the first element of which is all the observations that have the first label, the second element is all the observations that have the second label et cetera, then pipes that list to a function that maps the List.length operator on that list of lists. The result is a list with for each label, the number of observations that have that label.

The following code finds the number of correct predictions per label

```
let correctByLab =
    List.init
    (List.length labels)
    (fun i -> List.filter (fun z -> z.actual = labels.[i] && z.predicted = labels.[i]) results)
    |> List.map (fun listo -> List.length listo)
```

List.init creates a list of lists which for each label has all the results where the actual value corresponds to that label and where actual is equal to predicted, i.e., the prediction was correct. This list is piped to a function that maps the List.length operator to the list of lists, determining how many such observations there are per label.

7.1.3 Principal Component Analysis (PCA) on the MNIST-data set

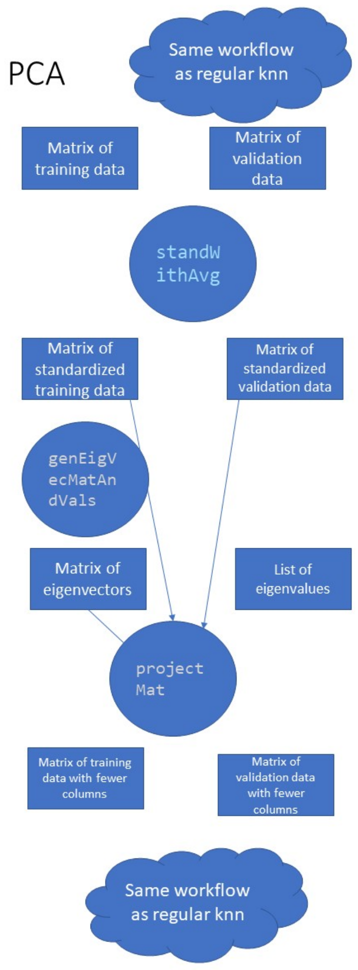
The PCA function is also performed inside the executeKNNAndPCA function, allowing it to use many of the same subfunctions and only take a detour from the workflow when specific PCA functionality is required.

Already in the above example, the picture features were populated into matrices, with labels in separate lists, and the splitFunction randomly assigned pictures into train, validation and test data sets. To standardize pictures, or as Chakraborty [5] puts it 'to subtract the average picture', the function findAvgs takes the average of each matrix column. It is the column averages of the training data set that are used to standardize both the training data set and the validation data set. Since the matrix is a MathNet.Numerics object, it has an attached method that can sum columns and output a vector, and the vector has a generic Map-function. This is an example of something that is object oriented rather than functional programming. It should be noted that it is absolutely possible to consider vectors and matrices as functions (for instance, a matrix can take one vector as input and output a function of that vector) and therefore, manipulating vectors and matrices should not in itself exclude a functional approach.

```
let findAvgs (mato:Matrix<float>) : (Vector<float>) =
    let resultVecto = mato.ColumnSums()
    resultVecto.Map(fun x -> x / (float mato.RowCount) )
```

The training data set is now a matrix that is standardized. It is input into the genEigVecMatAndVals function, which calculates the covariance matrix and uses MathNet.Numerics to calculate the eigenvectors and the eigenvalues. MathNet.Numerics generates a special object that contains these parameters. The eigenvalues are used to illustrate the eigenvalue spectrum (shown in the Research Diary) after which they are not processed further. It took a number of iterations to determine that MathNet.Numerics presents the eigenvectors in sorted ascending order according to eigenvalues, therefore the vectors are read into a list using the object method EnumerateColumns() |> List.ofSeq, after which the list is reversed and again read into matrix form.

The projectMat-function projects matrices onto the reduced dimensionality created by the eigenvectors and the number of principal components that is specified as a user input (most frequently, 10 is used). projectMat takes a standardized matrix, a matrix of eigenvectors and the numPC-parameter (10), takes the first 10 eigenvectors into a new matrix, transposes that matrix and multiplies it with the matrix that is to be projected. 10 eigenvectors in a 748-dimensional space will span a 10-dimensional subspace. The transposed eigenvector matrix will be (784x10). If there are 3.000 training images, the training data set will be a (3.000x748)-matrix. Multiplying the two together will create a matrix

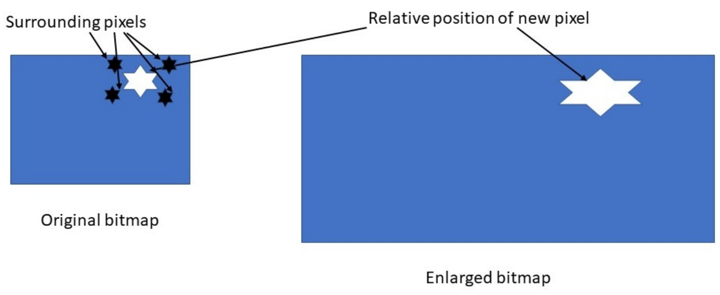

PCA
Same workflow as regular knn
Matrix of training data
Matrix of validation data
standWithAvg
Matrix of standardized training data
Matrix of standardized validation data
genEigVecMatAndVals
Matrix of eigenvectors
List of eigenvalues
projectMat
Matrix of training data with fewer columns
Matrix of validation data with fewer columns
Same workflow as regular knn

Figur 6: Flowchart for PCA

that is (3.000x10). The 3.000 training images have been projected onto a 10-dimensional subspace. projectMat is called again on the matrix of standardized validation images, if there are 1.000 validation images, the validation data set will now be a matrix that is (1.000x10). These two transformed matrices can then be passed onto the performReportAndTest-function in precisely the same way as for the regular knn-analysis. This again is a strength of functional programming that many of the same higher order functions can be called, PCA simply becomes an intermediate step that reduces the column size of the matrices involved, as shown in Figure (6).

7.1.4 KNN-analysis of the Pins-data set.

The analysis of the Pins data set was performed through a combination of reuse of the generic functions defined above and several specific function that became required due to the difference in input data. Where MNIST is already in numeric format in a csv-file, Pins existed in the form of 17.534 .jpg-files in different formats, showing faces. The .jpg-files are downloaded to the student's pc. The function createFilos reads all file path names and writes them to a txt-file called filos.txt, returning unit. A separate python programme, readFromOpenCV.py, then reads filos.txt and applies the OpenCV-module. OpenCV uses a Haar Cascade algorithm to identify specific sub-areas of a picture. In this case, the area of interest is a rectangle comprising the area from the eyes to the chin. The approach was based on Tiwari in Real Python [25]. For each picture, readFromOpenCV.py will then write a tuple with the file path and the coordinates of the bounding box identified. Each tuple is written as a line in a file called tuplefile.txt. tuplefile.txt is then specified as one of the input parameters to the main function called pins that will be described below. The filenames in tuplefile.txt are read into a



Figur 7: Calculation of new pixels in enlargement function

seq. Again, in the initial iterations, it was convenient to work with a seq, but for the final version, the seq was converted to a list. The list was then sorted so that out of the 17.534 files covering 105 celebrities (each celebrity is a label), it was reduced to 10 labels with 100 photos each. This was done randomly using the newKFYShuffle. This curated list of filenames and bounding box information is then read into the processTuple function. processTuple reads a line in tuplefile.txt, reads the label, finds the .jpg-file on the pc, generates a bitmap, calculates the center of the bounding box which will then presumably be the center of the face, calculates how close the center of the photo is to the border of the photo and collects all this information in the pinsPic custom record type. A pinsPic is similar to a pic defined above but holds substantial additional information. It is an example of how in the functional approach, a custom record type can be defined to organize a number of specific parameters.

```
type pinsPic =
    {
        bmpo : System.Drawing.Bitmap;
        label : string;
        features:float list;
        center: int * int;
        minHorDist: int;
        minVerDist: int;
    }
```

Unlike the pic, which only keeps numeric features, the pinsPic has the features read into a bitmap. It also has a center, which is the center of the face, calculated as the center of the bounding box. It then has minHorDist, the smallest horizontal distance between the center of the face and the edges, and minVerDist which is the smallest vertical distance from the center of the face and either top or bottom.

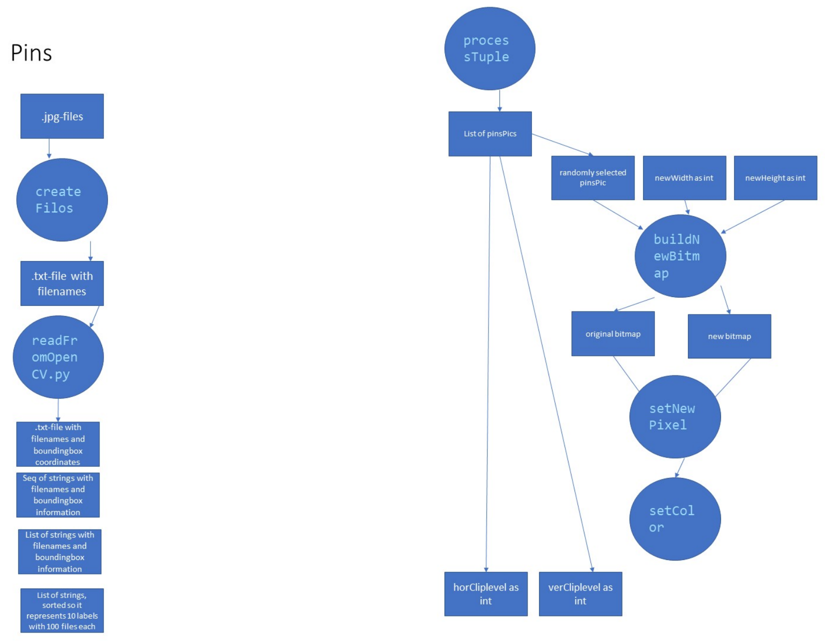
As a separate branch of the programme, a pinsPic is selected randomly and passed to the buildNewBitmap function, along with a newWidth-parameter that is 50 per cent larger than the existing width parameter, and a newHeight parameter which is 7 per cent larger than the existing height parameter. buildNewBitmap creates a new bitmap of the appropriate dimensions and initiates a unit list of length corresponding to the number of pixels. This function updates the color of each pixel in the new bitmap. This is done with the following code

```
List.init (newWidth * newHeight) (fun i -> setNewPixel oldBmp newBmp i) |> ignore
```

setNewPixel calls setColor once for each pixel in the new enlarged photo.

Each new pixel will have a relative position in the new bitmap (x percent along the width axis, y percent along the height axis). The function will then find this relative position in the original bitmap. The position will either correspond to an original pixel or to a point in between four corner pixels. The enlarge function reads the colors of these four corner pixels and calculates the colors of the new pixel as an average. This is a functional approach and one that yielded correct results.

The main branch of the programme now continues. The objective is to center all images around the center of the face for comparability. This centering will be achieved by cropping. Some images will have the face center close to either the horizontal or vertical border of the picture, in which case


Pins
.jpg-files
createFilos
.txt-file with filenames
readFromOpenCV.py
.txt-file with filenames and boundingbox coordinates
Seq of strings with filenames and boundingbox information
List of strings with filenames and boundingbox information
List of strings, sorted so it represents 10 labels with 100 files each
processTuple
List of pinsPics
randomly selected pinsPic
newWidth as int
newHeight as int
buildNewBitmap
original bitmap
new bitmap
setNewPixel
setColor
horCliplevel as int
verCliplevel as int

Figur 8: First and second part of Pins flow

cropping will produce a very small picture (if the face center is in the second pixel from the left border of the image, a cropping that centers the image around the face center will produce an image that is only 3 pixels broad). The programme therefore reads the minHorDist from all pinsPics and finds the 10th percentile which it calls horClipLevel, then does the same for the minVerDist and calls the result verClipLevel. It then calls the List.filter function to eliminate all pinsPics which have either a minHorDist that is below the horClipLevel, or a minVerDist which is below the verClipLevel, or both. This can reduce the data set by up to 20 per cent, but in several iterations of the function, parameters were relatively stable, eliminating around 13 per cent of all images and defining cliplevels that were around 50 pixels.

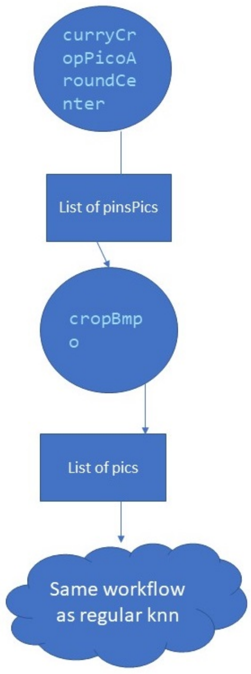
The horClipLevel and verClipLevel are curried into the function cropPicoAroundCenter. This function creates a new bitmap of the required dimensions, reads the center attribute from the pinsPic and then calls a function cropBmpo which performs the actual cropping. The cropping is done by starting at a given point within the original bitmap and then reading the pixels one by one into the new bitmap. If for instance the picture must be cropped so that the first 10 pixels in the original bitmap are discarded, cropBmpo simply starts 10 pixels into each line. Since curryCropPicoAroundCenter has used all the rich information that was in the pinsPic custom record types, it is enough for it to return a list of pics, i.e., a label and a list of features. The rest of the program flow is therefore similar to the knn-analysis above, the splitFunction reads the pics, splits them into train, valid and test and passes train and valid to performAndReportTest. This is shown in Figures (8) and (9)

7.2 Overview of the app output

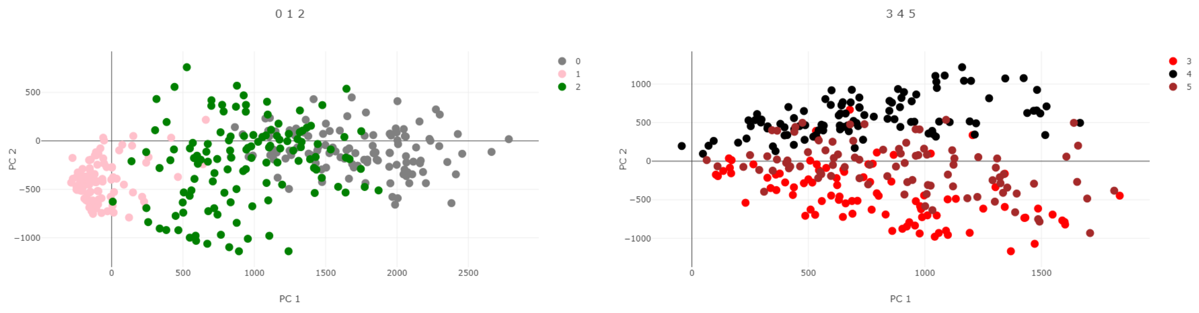
This section discusses the app output shown in Appendix III.

7.2.1 Output from analysis of MNIST-data set

It can be seen from the output that applying the KNN-model to the MNIST-data set results in predictions of which 92 per cent are correct. The output shows precision and recall according to label, for instance, among the 1.000 test observations, there were 97 sevens. Whenever the model predicted that a test observation was a seven, it was correct 92 pr cent of the time (precision). Of the 97 sevens, 85 per cent were predicted correctly (recall). It can be seen from the Research Diary that even the



Figur 9: Third part of Pins flow



Figur 10: 0,1 and 2 (left) and 3, 4 and 5 (right)

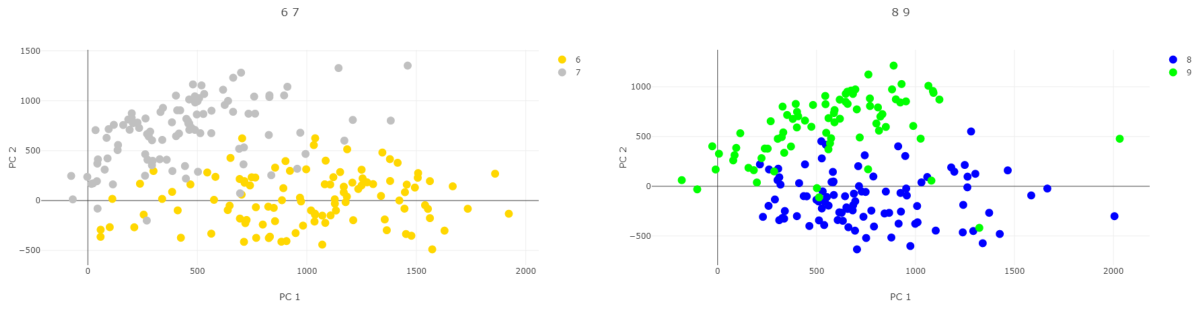
first versions of the knn-model, coded in a style closer to imperative programming, had the same high level of performance on MNIST. As for the model that first transforms to 10 principal components, then performs KNN on the transformed data set, it is correct an impressive 87 percent of the time, meaning almost no loss in functionality for a substantial reduction in data dimensionality from 784 to 10.

It can be seen in the Research Diary how the training data set was reduced to 2 principal components, after which the digits were shown color-coded in a two-dimensional plot. It can be seen visually that the digits cluster around their label values as shown in these figures. From the Research Diary:

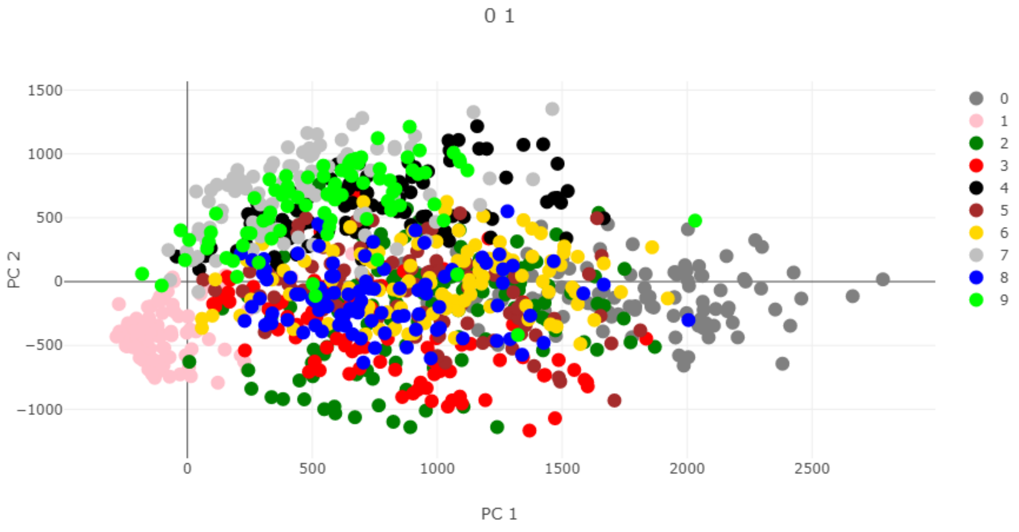
Figure 10 shows pictures with the labels 0, 1 and 2 plotted against each other in two-dimensional space, as well as the same plot for 3, 4 and 5. The 1s sit to the left in a pink cluster and the zeroes sit in an elongated cloud towards the right.

Figure (11) shows pictures with the label 6 and 7 as well as the same plot for 8 and 9. Again, the pictures seem clearly separated but more so along the second axis. A sanity check was performed at this point as to whether the axes had in fact been swapped somehow but this did not seem to be the case.

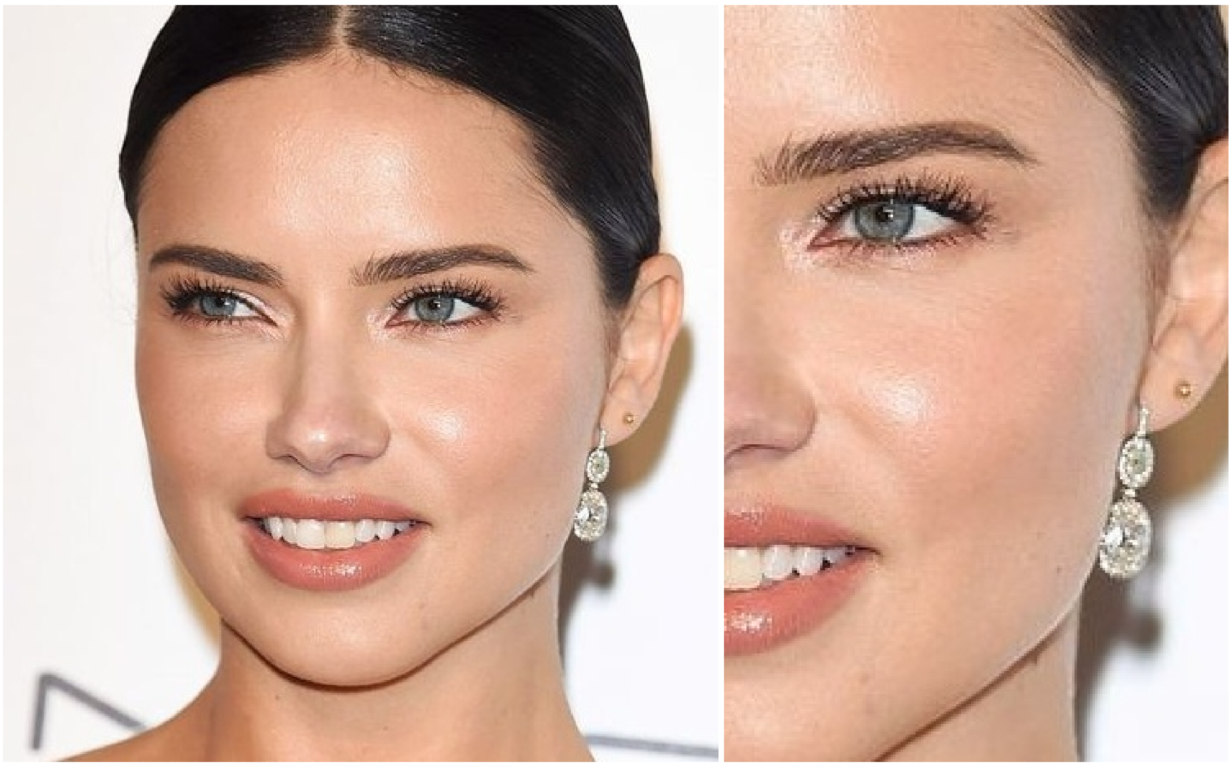
Finally, in Figure (12) all 10 labels were plotted against each other for completeness although this diagram is more difficult to interpret and does not show a perfect clustering.



Figur 11: 6 and 7 (left) and 8 and 9 (right)



Figur 12: All labels 0-9



Figur 13: Before and after cropping

7.2.2 Output from analysis of Pins-data set

In the analysis of the Pins-data set, first a picture is selected at random for testing of the cropping algorithm, as shown in Figure (13), also shown in the Research Diary. The cropping occurs at pixel level and the two pictures are shown in equal size due to the LaTeX-settings.

Then a picture is selected at random for testing of the the scaling algorithm buildNewBitmap. The original picture of the actress Emma Stone has dimensions (121x129) pixels and it is enlarged to (182x138). The result can be seen in Figure (14). For comparability, the two photos are coerced to the same width and height in LaTeX (otherwise, it would be LaTeX performing the enlargement instead of the app doing it) and the setting keepaspectratio has been selected. Therefore, the original photo actually appears larger, but it is clear that the algorithm has succesfully scaled the second picture, and more so in the horizontal than in the vertical dimension.

A sanity check is also performed of the function that crops pictures around the center, shown in Figure (61) in Appendix V. 4 pictures are selected randomly and shown in the format where they are cropped around the center. The function appears to perform the cropping correctly (all 4 pictures have the center of the face in the center of the frame). This would in itself improve performance since noses would be compared to noses, eyes to eyes and so on. Although the Pins data set is relatively controlled (facial photos of one person), it seems that a lot of information is lost by cropping to the boundaries. It is recalled that the boundaries are set so that 90 per cent of the data set would have the center of the face inside the boundaries, and only 10 percent would have the center outside and therefore be discarded. Furthermore, not all photos have the same scale, and therefore comparability will decrease in pixels that are far from the center. This was discussed with the supervisor, who referred to scientific work demonstrating that features such as hair played a substantial role in actual facial recognition by humans. Cropping that eliminated part of the face could therefore be expected to reduce performance in the KNN-model. The supervisor mentioned more advanced algorithms that through a patching-approach would identify first eyes and compare these, then identify ears and compare these across different photos before generating a consolidated prediction.

In the analysis of the Pins-data set, 1.000 observations are selected as described above, the 10th percentile of of horizontal distance to center of face is read as 50 pixels, the corresponding number in the horizontal dimension is read as 53 pixels, selecting only pictures that have larger distances than these numbers reduces the data set to 859 observations. These are then cropped to (50, 53)-

Figur 14: Original photo (121x129) (left) and enlarged photo (182x138) (right)

pixel images that are perfectly centered around the center of the face. The 859 observations are then split into 515 train observations, 172 validation observations and 172 test observations. A 5-nearest neighbor analysis is then performed without PCA. The overall accuracy of the model is a disappointing 24 per cent, the model performs best for the not very well-known actor Dominic Purcell who may be best known for a small part in Mission: Impossible 2 in 2000 (precision and recall both 33 per cent) and most poorly for actress Camila Mendes who voiced one episode of the Simpsons (precision 17 per cent recall 14 per cent). All this is after restricting the data set to only 10 labels. It is clear that even after the recentering of images, the knn-model does not perform acceptably on a complex data set such as Pins.

7.2.3 Running time

It should be noted that the optimization of running time is not an objective for this project. This is based on a conversation with the supervisor where it was noted that several other optimization steps had to be taken before actual experiments of running time could be performed. Running time still does factor into the project.

One limitation of KNN is the running time which is quadratic $O(n^2)$in the number of observations, as each picture has to be compared with all other pictures. Execution time furthermore rises linearly in the length of the feature lists, as each feature element in the test observation must be compared to the corresponding element of each element in the training set. This also made analysis of the Pins-data set slower than analysis of the MNIST-data set. MNIST pictures are monochromatic and with (28x28) pixels have 748 features. For comparison, Pins-pictures were cropped to roughly (50x50), for each pixel comparing three channels (RGB) which made their feature lists around 10.000 element long.

8 Learnings from the functional journey

To reflect the method choice of a longitudinal time-horizon, this section reviews the Research Diary and how the code and project evolved from February to August 2022. The Research Diary itself appears as Appendix IV.

8.1 From start to end

To illustrate how the functional approach developed throughout the project, it can be useful to look at the very first two versions of code.

- The very first version of the KNN-programme, coded in the week of 16 February 2022 and available on Github as https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/kNearNeighb.fsx relied heavily on the imperative paradigm. The total amount of data was read into a list, not benefiting from the lazy evaluation of sequence. The list was recursively processed from string into integer format. The data was then split into train and test data by defining these three as mutable lists. The programme contained a total of 5 loops. A while loop was defined under the boolean condition that it should continue untill the pointer had reached the end of the total data set. A single call to the randomization algorithm then decided whether the observation should go into train or test, then moved on to the next. Piping was not used, instead intermediate mutable variables held subtotals that were then added together in the final line of the function. The programme did correctly implement the KNN-algorithm and produced plausible output values. This illustrates the fact that F#, as also described in Section 5.1. does not explicitly forbid, e.g., destructive assignments or loops, so more work was required by the student in order to understand how F# was 'functional-first' and 'encouraged' a functional style. It became progressively clear that this encouragement happened by the functional style being easier to read. For instance piping meant that intermediate, mutable variables with names that were difficult to interpret were no longer necessary. A list could be processed by List.map which made it clearer that the list elements were the input arguments to the function in question.

- The very first version of the programme also allowed the input data to dictate the typing. MNIST was available in a csv-file where each line represented a picture. The line had 785 elements, the first of which was the label (a digit from 0 to 9) and the following 784 were a pixel value of the A-channel in a monochromatic bitmap of the digit. The programme treated such a line as a picture, simply read the first element as a label and read the other elements as the features. The supervisor commented that even though this was explicitly documented in the programme, it did not comply with an approach of strong typing. In response, the custom record type pic was create as a tuple of a string label and a float list of features, eliminating the potential for error.

- In the second instalment of the programme, coded in the week of 23 February 2022 and available on Github as https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/kNNImproved.fsx, 4 of the 5 loops were replaced by recursive statements. For instance, the Euclidean distance between two points was found by currying the coordinates of the first point into the distance function, then applying it recursively to the second point. Insertion sort was also performed recursively. The supervisor commented that while recursion was closer to the functional approach, it would be better to call to F#-modules such as List.sort.

- Another thing commented upon by the supervisor was that if types and constants are named in a certain way, this can sometimes indicate poor programming choices. At some point, a custom record type called pic2 was defined, because for some process step it became necessary to alter the custom record type pic which was generally used. The supervisor said that the '2' was a red flag and that the name should reflect the actual purpose. This became the pinsPic custom record type, reflecting that as documented above, the Pins-records had to be represented by many more attributes (centers, distances to center etc.) than the MNIST-records.

These two initial programmes can be contrasted to the final app which has only one loop, no mutable values, only one recursive statement and extensive application of piping, custom record types and module calls. It is also evident that the final code, while having a large number of lines, has mostly short statements that are easy to read and that are close to the underlying mathematical approach.

8.2 How did model performance evolve throughout the project.

The initial simple KNN-model coded in February performed almost as well as the final model shown in this project, with improvements coming mostly from adding more observations in the final version.

The initial PCA-model had very low performance, and it has required substantial work to achieve the performance shown in this project. When the KNN-model was applied to the Pins-data set, performance was poor. This was addressed by centering the images and by coercing how many different labels were in the data set and how many images were available per label, but performance remained poor. The supervisor suggested running a neural network, which lead to finding the work of Haaris Mehmood [15] that provided neural network functionality coded in F# and available on Github. Two blogs by Mehmood presented an interesting discussion of how for instance in the neural network, a strong typing is ensured by defining tuples consisting of each layer and its activation function. The layers then functionally pass their output from one to another. For time management purposes, this avenue was not pursued further.

As a sanity check, it was examined whether the model would perform better on the Pins-data set if this were transformed from polychromatic to monochromatic. The argument for why this would improve performance started with the fact that MNIST was monochromatic and the model performed well on MNIST. Theoretical reflections began by considering a binary model that predicted only whether the image represents one specific label or not ('is this Harrison Ford or not?) with the zero hypothesis being 'This is not Harrison Ford'. In such a model, removing colors would increase the number of Type I-errors. The model would more frequently predict that people who had the same facial features as Harrison Ford, but different eye color, were in fact Harrison Ford. The transformation would however reduce Type II-errors. The pictures that were in fact of Harrison Ford but which had wrongly been predicted as 'NOT' due to e.g. differences in lighting or Ford having dyed his hair for a role would now more often be predicted correctly. The net effect cannot be predicted theoretically, and this is even more the case for a model with 10 different labels. The experiment was discussed with the supervisor in the last meeting on the 10th of August 2022. In coding terms, the transformation could be done quite easily because the function bmpoToFeatures was originally coded taking either 'alpha' or 'RGB' as an argument. 'alpha' meant that the model read the alpha-channel of each bitmap, creating a list of features that had one average number for each pixel. As shown in the programme output in Appendix III, the transformation deteriorated model performance further. Overall accuracy was now down to 11 per cent and the model for instance could no longer recognize one single out of 30 images of the above-mentioned Camila Mendes.

8.3 Example of functional approach: The scaling function

The scaling function took considerable time and effort, but once again reflected the differences between object-oriented and functional approach. The objective was to develop a scaling function that could enlarge from any format to any format, reflecting that the Pins-data set contained pictures in many different formats.

Two approaches were attempted

- The first approach was based on altering the existing bitmap object. If the bitmap had to scale from a width of 100 pixels to a width of 110 pixels, the challenge was simply to generate 10 new pixels and place them at the appropriate places among the existing pixels and calculate their covers as averages. While this function technically created a new bitmap object, in reality it updated an existing bitmap object. The object-oriented approach was never implemented correctly, the Research Diary contains a number of examples.
- The second approach, which is described in Section 7.1.4. above treated the enlarged picture as a function of the existing bitmap and yielded correct results.

8.4 Two ecosystems

PCA was one example where the student searched widely for a standard F#-module, raised a question within the official F#-discord group https://discord.com/channels/196693847965696000/386375157918466069 and reviewed material available on the FSlab.org server. Several external modules exist such as Deedle, FSharp.Stats, FSharp.Data and MathNet.Numerics, but it was difficult to determine how much traction they had. The user base did not seem to converge on a few, globally accepted approaches. The student also struggled because in the dotnet-universe, while many relevant functions did exist such as ML.NET, the documentation often referred to C#-resources. The student eventually settled on the MathNet.Numerics-module which is object-oriented but which does have a usable online manual in https://numerics.mathdotnet.com/api/MathNet.Numerics.LinearAlgebra.Factorization/Evd601.htm. A workable and truly functional resource was not found.

It was finally discussed with the supervisor how the object-oriented and the functional paradigms can also be seen as two distinct answers to two distinct questions. For an entry level data science student, who is impatient to see real output on benchmark data sets, object-oriented paradigm based on Python can seem hard to beat. This is a major reason for the traction of the Python-based ecosystem in Machine Learning and Data Science. However, the low entry point for Python also allows many bad habits to exist and leads to the creation of a large body of code that, while executing correctly, is not an optimized or clear representation of the algorithm and data in question.

The position of Python can be compared to the role of English as a global language. The English language has attained this role for a number of reasons, including its low entry point. The large proportion of non-native speakers in turn drives many versions of English that are not always consistent with the formal grammar. Already in 1946, George Orwell [19] commented on how English was evolving.'It [The English language] becomes ugly and inaccurate because our thoughts are foolish, but the slovenliness of our language makes it easier for us to have foolish thoughts. The point is that the process is reversible. Modern English, especially written English, is full of bad habits which spread by imitation and which can be avoided if one is willing to take the necessary trouble. If one gets rid of these habits one can think more clearly'.

The functional programming paradigm is the response to the question of how to maintain structure in a world with an ever increasing body of suboptimally written code. Syme [24] and others have stressed the role of functional programming in an enterprise environment with focus on the productivity of a stable of experienced programmers, but as documented in this project, functional programming has even wider applications.

9 Conclusion: were the expected benefits from functional programming achieved?

Having followed the above deductive method and in Section 6 having defined a set of expected benefits from functional programming, it will now be concluded whether these expected benefits were in fact achieved.

9.1 A well-structured software that is easy to write and debug (Hughes)

Splitting the code into various subfunctions made the 900-line fsx-file relatively well structured. The code was difficult to write, but once written it ran well and - although the student did not collect quantitative data of this - there were relatively few crashes or examples of the code not compiling. The time was spent searching for programming solutions rather than finding errors in code that had already been written and executed.

9.2 Clearer and simpler analysis of how to build an image recognition app.

For any entry level programmer, a certain amount of time is spent seeing the trees rather than the forest, and the overall picture of the app did not materialize until the performance data was available. However, the functional approach did allow a good structure, splitting out different functionalities and making clear how tools such as PCA or cropping were to be applied in the overall workflow of the app.

9.3 Increased proximity to the underlying mathematics

This was clearly achieved for instance in the calculation of Euclidean distances or the treatment of matrices.

9.4 Better understanding of functional programming

This was certainly achieved as demonstrated in sections 5 and 6 above.

9.5 Good management of input data

This was certainly achieved, managing data as custom record types called pic or pinsPic or consolidated into matrices.

9.6 The app will be developed on small data set but the approach should scale to large data sets

This was not achieved, the student's pc could only process an input of 5.000 MNIST-observations and 1.000 Pins-observations in little under one hour.

9.7 More routine for the student in programming in F#

Overall, the benefits described by Martin and Curtis [6] in the form of better student engagement were clearly achieved.

9.8 Better understanding of the alternative, imperative and object-oriented programming paradigm

This was achieved.

9.9 Experience a prolific ecosystem of modules built for and by functional programmers

This is one complexity of an F#, functional programming-based approach. The mere network-effect of the current investments in the Python-based data science ecosystem in the form of for instance Pytorch or MobileNet seem to outweigh what is available in dotnet - with C# representing the majority of dotnet-resources. As an example, for this project it was attempted to use the F#-based Accord image processing library, but the bridge to python ended up being more workable.

The expected benefits from using functional programming were thus to a large extent achieved in practice.

10 Appendix I. Code for the simple example of the choice between functional or object-oriented paradigms in business applications.

```
###Simpel vaccination app - object-oriented approach
class Borger():

    def __init__(self, name, cpr_number):
        self.name = name
        self.cpr_number = cpr_number
        self.times_vaxed = 0
        self.dates_vaxed = []

    def perform_vax(self, dato):
        self.times_vaxed += 1
        self.dates_vaxed.append(dato)

    def vaccinate(self, dato):
        if self.times_vaxed > 2:
            print('Already twice vaccinated')
        if self.times_vaxed == 1:
            if dato - self.dates_vaxed[0] < 180:
                print('Already once vaccinated within the last 6 months')
            else:
                self.perform_vax(dato)
        if self.times_vaxed == 0:
            self.perform_vax(dato)

    def report_status(self):
        print('Citizen '+ self.name + ' with cpr ' + self.cpr_number + ' has been vaccinated ' + (str(self.times_vaxed)) + ' times')

///Simpel vaccination app - functional approach
///Simpel vaccination app - functional approach
type citizenUnvaccinated =
    {
        name:string;
        cprNumber:string;
    }

type citizenOnceVaccinated =
    {
        name:string;
        cprNumber:string;
        firstvacdate:int;
    }

type citizenTwiceVaccinated =
    {
        name:string;
        cprNumber:string;
        firstvacdate:int;
        secondvacdate:int;
    }

///<summary>Vaccinates an unvaccinated person</summary>
///<param name="dato">The date</param>
///<param name="cito">A citizenUnvaccinated</param>
///<returns>A citizenOnceVaccinated</returns>
let vaccinateFirstTime (dato:int) (cito:citizenUnvaccinated) : citizenOnceVaccinated =
    {name=cito.name;
    cprNumber=cito.cprNumber;
    firstvacdate=dato}

///<summary>Vaccinates a once-vaccinated person</summary>
///<param name="dato">The date</param>
///<param name="cito">A citizenOnceVaccinated</param>
///<returns>A citizenTwiceVaccinated</returns>
let vaccinateSecondTime (dato:int) (cito:citizenOnceVaccinated) : citizenTwiceVaccinated =
    if dato - cito.firstvacdate >= 180 then
        {name=cito.name;
        cprNumber=cito.cprNumber;
        firstvacdate=cito.firstvacdate;
        secondvacdate = dato}
```

```
else
    {name="error";
    cprNumber="000000-0000";
    firstvacdate=0;
    secondvacdate=0;
    }
```

11 Appendix II. App code (also available on Github)

11.1 Fsharp app app.fsx

This app is also available on Github https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/app.fsx

```
///Contents
///1. Welcome statement to user
///2. Opening modules and setting current directory
///3. Defining custom record types
///4. Functions that are common to both the MNIST- and PINS-analysis
///5. Functions that are specific to the MNIST-analysis
///6. Functions that are specific to the PINS-analysis
///7. Setting input parameters
///8. Executing programme
///9. Farewell-statement to user

///1. Welcome statement
printfn "Starting to execute Facial Recognition App in F#"
printfn "%s" (System.DateTime.Now.ToString())
printfn "Writing output to timsOutput.txt"

///2. Opening modules and setting current directory
System.IO.Directory.SetCurrentDirectory("c:\\users\\timmo\\onedrive\\diku\\new_puk1\\F10082022")
///The System.Drawing-library must now be called via nuget. In case there is an error message saying that
///dotnet cannot find the relevant dll-file, the user must
///Go to this path and manually delete the dll-file
///"C:\Users\timmo\.nuget\packages\system.drawing.common\6.0.0\runtimes\win\lib\net6.0\System.Drawing.Common.dll"
///manually manually copy the following file
```

```
///FROM "C:\Program Files\dotnet\sdk\6.0.101\System.Drawing.Common.dll"
///TO "C:\Users\timmo\.nuget\packages\system.drawing.common\6.0.0\runtimes\win\lib\net6.0\"
#r "nuget:System.Drawing.Common"
open System.Drawing
///opening MathNet.Numerics via nuget
#r "nuget:MathNet.Numerics.FSharp, 4.15.0"
open MathNet.Numerics.LinearAlgebra
///opening Xplot.Plotly
#r "nuget:XPlot.Plotly"
open XPlot.Plotly
let timsOutput = System.IO.File.CreateText "timsOutput.txt"
let rando = System.Random()

///3. Defining custom record types
type pic =
    {
        label : string;
        features : float list
    }

type neighbor =
    {
        label : string;
        distance : float
    }

type result =
    {
        actual : string;
        predicted : string
    }

type inputparameters =
    {
        numObs:int;
        validPct:float;
        testPct:float;
        k:int;
        firstNumPC:int;
        fileName:string;
    }

type pinsPic =
    {
        bmpo : System.Drawing.Bitmap;
        label : string;
        features:float list;
        center: int * int;
        minHorDist: int;
        minVerDist: int;
    }

///4. Functions that are common to both the MNIST- and PINS-analysis

///<summary>Combines a list of strings to one string</summary>
///<param name="listo">A list of strings</param>
///<returns>A string</returns>
let rec recursWrite (listo:string list) : string =
    match listo with
    |[] -> ""
    |h::[] -> h
    |h::t -> h + " " + recursWrite t

///<summary>Writes to a txt-file</summary>
///<param name="wo">A streamWriter</param>
///<param name="listo">A list of strings</param>
///<returns>Unit</returns>
let timWrite (wo:System.IO.StreamWriter) (listo:string list) : unit =
    printfn "Printing to output"
    let stringo = recursWrite listo
    printfn "%s" stringo
    wo.WriteLine stringo
```

```
///<summary>Writes to a txt-file</summary>
///<param name="listo">A list of strings</param>
///<returns>Unit</returns>
let cWT = timWrite timsOutput

///<summary>Performs the Knuth-Fisher-Yates Shuffle</summary>
///<param name="n">An integer</param>
///<returns>A shuffled list of indexes</returns>
let newKFYShuffle (n:int) : (int list) =
    printfn "Shuffling - expect some process time as %i elements are shuffled" n
    let arrayo = [|0..(n-1)|]
    let mutable (tmp:int) = 0
    let mutable (nextIndex:int) = 0
    for i = 0 to (n-1) do
        tmp <- arrayo.[i]
        nextIndex <- rando.Next(n-1)
        arrayo.[i] <- arrayo.[nextIndex]
        arrayo.[nextIndex] <- tmp
    printfn "Shuffling completed"
    List.ofArray arrayo

///<summary>Takes a random sample of a given length from a list</summary>
///<param name="listo">The list to be sampled - can be string, int or float list</param>
///<param name="k">The size of the sample</param>
///<returns>A list</returns>
let shuffleAndTake (k:int) listo =
    listo
    |> List.length
    |> newKFYShuffle
    |> List.map (fun i -> listo.[i])
    |> List.take k

///<summary>Generates an integer of observations to be selected for train/valid/test</summary>
///<param name="pct">The percentage of all observations</param>
///<param name="n">The number of all observations</param>
```

```
///<returns>An integer number</returns>
let findInt (pct:float) (n:int) =
    n
    |> float
    |> (fun x -> x * pct)
    |> (fun x -> x + 0.5)
    |> int

///<summary>Finds the Euclidean distance between two pics</summary>
///<param name="firstPic">The first picture</param>
///<param name="secondPic">The second picture</param>
///<returns>A float</returns>
let findDisto (firstPic:pic) (secondPic:pic) : float =
    List.map2 (fun x y -> (x - y) ** 2.0) firstPic.features secondPic.features
    |> List.sum
    |> (fun x -> sqrt x)

///<summary>Generates a neighbor, i.e. for a given picture the distance to another picture and the label of that picture</summary>
///<param name="firstPic">The original picture</param>
///<param name="secondPic">The neighbor picture</param>
///<returns>A neighbor</returns>
let generateNeighbor (firstPic:pic) (secondPic:pic) : neighbor =
    {
        label = secondPic.label;
        distance = findDisto firstPic secondPic
        }

///<summary>For a given test picture and a training set of neighbors, predicts the label of the test picture</summary>
///<param name="neigbors">The training set of neighbors</param>
///<param name="pico">The test picture</param>
///<returns>A predicted label</returns>
let predict (trainPics:pic list) (pico:pic) (k:int) : string =
    let labelsWithFreq =
        List.init (List.length trainPics) (fun i -> generateNeighbor pico trainPics.[i])
        |> List.sortBy (fun x -> x.distance)
        |> List.take k
        |> List.map (fun x -> x.label)
        |> List.countBy id
        |> List.sortByDescending (fun x -> snd x)
    let highestFreq =
        labelsWithFreq
        |> List.head
        |> (fun x -> snd x)
    let mostFrequentLabels =
        List.filter (fun x -> (snd x) = highestFreq) labelsWithFreq
    let winningIndex = rando.Next(List.length mostFrequentLabels)
    fst labelsWithFreq.[winningIndex]

///<summary>Performs a test and prints out the results</summary>
///<param name="traino">A training set of pics</param>
///<param name="testo">A list of test pictures</param>
///<returns>Unit - results are printed to standard output</returns>
let performAndReportTest (traino:pic list) (testo:pic list) (k:int) : (unit list) =
    let (results:result list) = List.init (List.length testo) (fun i -> {actual=testo.[i].label; predicted=predict traino testo.[i] k})
    let labels =
        traino
        |> List.map (fun x -> x.label)
        |> List.distinct
        |> List.sort
    let corrects =
        results
        |> List.filter (fun x -> x.actual = x.predicted)
        |> List.map (fun x -> x.actual)
    let totalCorrect =
        List.length corrects
    cWT ["Total number of results"; (List.length results |> string)]
    cWT ["Total number of correct"; (string totalCorrect)]
    let correctPct =
        totalCorrect
        |> float
        |> (fun x -> x / (float (List.length results) ) )
    cWT ["Percentage correct"; (100.0 * correctPct|> string)]
    let totalActualPerLab =
        List.init (List.length labels)  (fun i -> List.filter (fun z -> z.actual = labels.[i]) results)
        |> List.map (fun listo -> List.length listo)
    let totalPredByLab =
        List.init (List.length labels) (fun i -> List.filter (fun z -> z.predicted = labels.[i]) results)
        |> List.map (fun listo -> List.length listo)
    let correctByLab =
        List.init
            (List.length labels)
            (fun i -> List.filter (fun z -> z.actual = labels.[i] && z.predicted = labels.[i]) results)
        |> List.map (fun listo -> List.length listo)
    let precision = List.map2 (fun x y -> (float x) / (float y) ) correctByLab totalPredByLab
    let recall = List.map2 (fun x y -> (float x) / (float y) ) correctByLab totalActualPerLab
    List.init (List.length labels)
        (fun i ->
        cWT ["Label"; (string labels.[i]); "number of obs"; (string totalActualPerLab.[i]);
        " precision"; (string precision.[i]); "recall"; (string recall.[i])])

///5. Functions that are specific to the MNIST-analysis. This includes functions that draw on MathNet.Numerics to perform PCA

///<summary>Generates a float list from a list of strings</summary>
///<param name="listo">A list of strings</param>
///<returns>A list of floats</returns>
let makeFloatList (listo:string list) : (float list) =
    List.map (float) listo

///<summary>Reads from a file and generates a list of float lists</summary>
///<param name="fileName">Name of the file</param>
///<param name="numObs">Number of observations to be read</param>
///<returns>Data in the form of a list of float lists</returns>
let generateDataList (fileName:string) (numObs:int) : (float list list) =
    System.IO.File.ReadLines fileName
    |> List.ofSeq
    |> List.map (fun x -> List.ofArray (x.Split ',') )
    |> List.filter (fun x -> not (List.contains "label" x) )
    |> List.map makeFloatList
    |> shuffleAndTake numObs

///<summary>Builds a list of pics from a list of labels and a matrix of features</summary>
///<param name="labels">A list of labels</param>
///<param name="features">A matrix of floats</param>
///<returns>A list of pics</returns>
let buildPictureList (labels:string list) (features:Matrix<float>) : (pic list) =
    let numPic = features.RowCount
    List.init numPic (fun i -> {label=labels.[i]; features=features.Row(i)|> List.ofSeq })

///<summary>Calculates column averages on a matrix</summary>
///<param name="mato">The matrix</summary>
///<returns>A vector of averages</returns>
let findAvgs (mato:Matrix<float>) : (Vector<float>) =
    let resultVecto = mato.ColumnSums()
    resultVecto.Map(fun x -> x / (float mato.RowCount) )

///<summary>Standardizes a matrix by subtracting column averages</summary>
///<param name="mato">The matrix</param>
///<param name="avgos">The column averages - not necessarily from the matrix itself</param>
///<returns>A matrix</returns>
let standWithAvg (mato:Matrix<float>) (avgos:Vector<float>) : (Matrix<float>) =
    let stanMato =
        List.init (mato.ColumnCount) (fun i -> mato.Column(i).Map(fun x -> x - avgos.At(i)))
        |> List.map (List.ofSeq)
        |> matrix
    (stanMato.Transpose())

///<summary>Splits a list of float lists into a list of strings representing data labels and a matrix representing features</summary>
///<param name="listo">A list of float lists</param>
///<returns>A tuple with a list of strings and a matrix</returns>
let splitMatrix (listo:float list list) : (string List * Matrix<float>) =
    let mato = matrix listo
    let labels =
        mato.Column(0)
        |> List.ofSeq
        |> List.map int
        |> List.map string
    let featMato =
        mato.RemoveColumn(0)
    labels, featMato

///<summary>Splits the data set into train, valid and test and into labels and features for each of these categories</summary>
///<param name="dataList">The entire data set</param>
///<param name="validPct">Percentage set aside for validation</param>
///<param name="testPct">Percentage set aside for test</param>
///<returns>Train labels, train data set, valid labels, valid data set, test labels, test data set</returns>
let splitFunction
    (dataList:float list list) (validPct:float) (testPct:float) :
    (string list * Matrix<float> * string list * Matrix<float> * string list * Matrix<float>) =
        let numObs = List.length dataList
        let indexes = newKFYShuffle numObs
        let numValid = findInt validPct numObs
        let numTest = findInt testPct numObs
        let numTrain = numObs - numValid - numTest
        let trainIndexes = indexes.[0..(numTrain-1)]
        let validIndexes = indexes.[numTrain..(numTrain+numValid-1)]
        let testIndexes = indexes.[(numTrain+numValid)..]
        let trainLabels, trainMat =
            List.init numTrain (fun i -> dataList.[trainIndexes.[i]])
            |> splitMatrix
        let validLabels, validMat =
            List.init numValid (fun i -> dataList.[validIndexes.[i]])
            |> splitMatrix
        let testLabels, testMat =
            List.init numTest (fun i -> dataList.[testIndexes.[i]])
            |> splitMatrix
        cWT
            ["Number of observations:"; (string numObs); " train: "; (
            string numTrain); " valid: "; (string numValid); " test: "; (string numTest)]
        trainLabels, trainMat, validLabels, validMat, testLabels, testMat
```

```
///<summary>Generates an specified number of eigenvectors from a given covariance matrix</summary>
///<param name="mato">The training data with pictures in rows and full number of features as columns</param>
///<param name="numPC">The number of principal components</param>
///<returns>A matrix with ALL eigenvectors as columns WITH THE LARGEST EIGENVECTOR IN THE FIRST COLUMN</returns>
let genEigVecMatAndVals (mato:Matrix<float>) : (Matrix<float>)*(float list) =
    let matoTransp = mato.Transpose()
    let covs = matoTransp.Multiply(mato)
    let divisor = covs.ColumnCount - 1 |> float
    let covsDiv = covs.Map(fun x -> x / divisor)
    cWT ["Number of rows of covariance matrix"; (string covsDiv.RowCount)]
    cWT ["Number of columns of covariance matrix:"; (string covsDiv.ColumnCount)]
    let covsSvd = covsDiv.Svd()
    cWT ["Rank of covariance matrix"; (string covsSvd.Rank)]
    let covsEvd = covsDiv.Evd()
    let listo = (covsEvd.EigenVectors.EnumerateColumns() ) |> Seq.map (fun x -> List.ofSeq x) |> List.ofSeq
    let revListo = List.rev listo
    let evMat = matrix revListo
    let eigVals = List.init (covsDiv.RowCount) (fun i -> covsEvd.EigenValues[i].Real)
    evMat.Transpose(), eigVals

///<summary>Projects a matrix of features onto a matrix of eigenvectors</summary>
///<param name="mato">The matrix of features</param>
///<param name="eigVecMat">The matrix of eigenvectors</param>
///<param name="firstNumPC">The number of principal components selected for this analysis</param>
///<returns>A new matrix of reduced dimensionality</returns>
let projectMat (mato:Matrix<float>) (eigVecMat:Matrix<float>) (firstNumPC:int) : Matrix<float> =
    let fEVM =
        Seq.take firstNumPC (eigVecMat.EnumerateColumns() )
        |> Seq.map (fun x -> List.ofSeq x)
        |> List.ofSeq
        |> matrix
    let fEVMt = fEVM.Transpose()
    cWT
        ["Multiplying a matrix of"; (string mato.RowCount); "rows and"; (string mato.ColumnCount);
        "columns with eigenvectormatrix with"; (string fEVMt.RowCount); "rows and";
        (string fEVMt.ColumnCount); "columns"]
    let outputMatrix = mato.Multiply(fEVMt)
    cWT
        ["returning a matrix of projected images with"; (string outputMatrix.RowCount); "rows and";
        (string outputMatrix.ColumnCount); "columns"]
    cWT [" "]
    outputMatrix

///<summary>Performs a sanity check of a matrix</summary>
///<param name="mato">A matrix</param>
///<returns>Unit</returns>
let reportOnMatrix (mato:Matrix<float>) : unit =
    let matoSvd = mato.Svd()
    cWT ["matrix has rows:"; (string mato.RowCount); "columns:"; (string mato.ColumnCount); "rank:"; (string matoSvd.Rank)]
    if mato.ForAll(fun x -> x >= 0.0 && x < 256.0) then
        cWT ["all matrix values are between 0 and 256"]
    else
        cWT ["matrix has values outside of range 0-255"]

///<summary>Assigns colors according to label</summary>
///<param name="labelo">The label as a string</param>
///<returns>A color</returns>
let selectColorFromLabel (labelo:string) : System.Drawing.Color =
    if labelo = "0" then System.Drawing.Color.Gray
    elif labelo = "1" then System.Drawing.Color.Pink
    elif labelo = "2" then System.Drawing.Color.Green
    elif labelo = "3" then System.Drawing.Color.Red
    elif labelo = "4" then System.Drawing.Color.Black
    elif labelo = "5" then System.Drawing.Color.Brown
    elif labelo = "6" then System.Drawing.Color.Gold
    elif labelo = "7" then System.Drawing.Color.Silver
    elif labelo = "8" then System.Drawing.Color.Blue
    elif labelo = "9" then System.Drawing.Color.Lime
    else System.Drawing.Color.Maroon

///<summary>Executes the main programme</summary>
///<param name="inputo">Input parameters</param>
```

```
///<returns>Unit list</returns>
let executeKNNAndPCA (inputo:inputparameters) =
    cWT ["Performing k-nearest neighbor analysis on MNIST"]
    cWT [(System.DateTime.Now.ToString())]
    let dataList = generateDataList inputo.fileName inputo.numObs
    let trainLabels, trainMat, validLabels, validMat, testLabels, testMat = splitFunction dataList inputo.validPct inputo.testPct
    cWT ["reporting on trainMat"]
    reportOnMatrix trainMat
    let globalAvgs = findAvgs trainMat
    let trainMatStan = standWithAvg trainMat globalAvgs
    let validMatStan = standWithAvg validMat globalAvgs
    let (totalEigVecMat:Matrix<float>), (totalEigVals:float list) = genEigVecMatAndVals trainMatStan
    cWT
        ["Now performing comparisons of ordinary KNN and PCA for the following number of principal components";
        (string inputo.firstNumPC)]
    let trainProjectMat = projectMat trainMatStan totalEigVecMat inputo.firstNumPC
    let validProjectMat = projectMat validMatStan totalEigVecMat inputo.firstNumPC
    cWT ["trainProjectMat has"; (string trainProjectMat.RowCount); "rows and"; (string trainProjectMat.ColumnCount); "columns"]
    cWT ["validProjectMat has"; (string validProjectMat.RowCount); "rows and"; (string validProjectMat.ColumnCount); "columns"]
    let trainPicReg = buildPictureList trainLabels trainMat
    let trainPicPCA = buildPictureList trainLabels trainProjectMat
    let validPicReg = buildPictureList validLabels validMat
    let validPicPCA = buildPictureList validLabels validProjectMat
    cWT [" "]
    cWT ["Performing regular k-nearest neighbor validation of model on untransformed data sets with 784 features"]
    performAndReportTest trainPicReg validPicReg inputo.k |> ignore
    cWT [" "]
    cWT ["Performing PCA validation of model on transformed data sets with"; (string inputo.firstNumPC); "features"]
    performAndReportTest trainPicPCA validPicPCA inputo.k |> ignore
    cWT [" "]

///<summary>Performs a special variant of the PCA with only two principal components that can then be graphically illustrated</summary>
///<param name="inputo">Input parameters</param>
///<returns>Unit</returns>
let twoDims (inputo:inputparameters) =
    cWT ["Now generating visual illustration of the data in two-dimensional space"]
cWT [System.DateTime.Now.ToString()]
let dataList = generateDataList inputo.fileName inputo.numObs
let trainLabels, trainMat, validLabels, validMat, testLabels, testMat = splitFunction dataList inputo.validPct inputo.testPct
let globalAvgs = findAvgs trainMat
    let trainMatStan = standWithAvg trainMat globalAvgs
    let validMatStan = standWithAvg validMat globalAvgs
    cWT ["The eigenvector matrix is generated"]
    let (totalEigVecMat:Matrix<float>), (totalEigVals:float list) = genEigVecMatAndVals trainMatStan
    let projectedMat = projectMat validMat totalEigVecMat 2
    let labelMatrixTupleList = List.init (List.length validLabels) (fun i -> (validLabels.[i], projectedMat.Row(i) ))
    let generateOneChartOfLabel (labelo:string) =
    let xSeries, ySeries =
        labelMatrixTupleList
        |> List.filter (fun x -> fst x = labelo)
        |> List.unzip
    |> snd
    |> List.map (fun x -> x.[0], x.[1])
    |> List.unzip
    let coloro = selectColorFromLabel labelo
    let myPlot = new Scatter()
    myPlot.mode <- "markers"
    myPlot.x <- xSeries
    myPlot.y <- ySeries
    myPlot.marker <- Marker(color=coloro, size=12)
    myPlot.text <- labelo
    myPlot.name <- labelo
    myPlot
    let generateCombinedPlotOfLabels (labeloList:string list) =
    cWT ["Generating plot and publishing to browser"]
    let chartList = List.map generateOneChartOfLabel labeloList
    let buildTitle (labeloListo:string list) : string =
    if List.length labeloListo = 3 then
        labeloListo.[0]+ " " + labeloListo.[1] + " " + labeloListo.[2]
    else
        labeloListo.[0] + " " + labeloListo.[1]
    let titlo = buildTitle labeloList
```

```
        let timLayout = Layout(title=titlo, xaxis=Xaxis(title="PC 1"), yaxis=Yaxis(title="PC 2") )
        chartList |> Chart.Plot |> Chart.WithLayout timLayout |> Chart.Show
    List.map generateCombinedPlotOfLabels [["0"; "1"; "2"]; ["3"; "4"; "5"]; ["6"; "7"]; ["8";"9"]] |> ignore
    List.map generateCombinedPlotOfLabels [["0"; "1"; "2"; "3"; "4"; "5"; "6"; "7"; "8"; "9"]] |> ignore

///6. Functions that are specific to the analysis of the PINS-data set

///<summary>Cleans a string before label and bounding box values can be read</summary>
///<param name="stringo">The string to be cleaned</param
///<returns>A string that can be used as a label or as bounding box values</returns>
let cleanString (stringo:string) : string =
    let cleanChar (charo:char) =
        if charo = '"' || charo=''' || charo = '[' || charo = ']' || charo = '(' || charo = ')' then
            ""
        else
            string charo
    String.collect cleanChar stringo

///<summary>Transforms a bitmap to a float list of features</summary>
///<param name="bmpo">The bitmap</param>
///<param name="alphaOrRGB">Should the function read a list of alpha-values of length n or a list of RGB-values of length 3*n</param>
///<returns>A list of feature floats</returns>
let bmpoToFeatures (bmpo:Bitmap) (alphaOrRGB:string) : (float list) =
    let coordinates = List.init (bmpo.Width*bmpo.Height) (fun i -> (i % bmpo.Width) , (i / bmpo.Width) )
    if alphaOrRGB = "alpha" then
        List.map (fun coord -> bmpo.GetPixel(coord)) coordinates
        |> List.map (fun pixo -> float pixo.A)
    elif alphaOrRGB = "RGB" then
        List.map (fun coord -> bmpo.GetPixel(coord)) coordinates
        |> List.collect (fun pixo -> [float pixo.B; float pixo.G; float pixo.R])
    else
        [-900.0;-900.0]

///<summary>Finds label from a string representing a tuple with entire filepath and bounding box-information</summary>
///<param name="stringo">
///<returns>A string with the label</returns>
```

```
let findLabelFromString (stringo:string) : string =
    stringo
    |> (fun x -> x.Split("\\"))
    |> (fun x -> x.[(Array.length x)-3])
    |> (fun x -> x.[5..])
    |> String.map (fun charo -> if charo = ' ' then '_' else charo)

///<summary>Transforms a tuple with the file name of a jpg and the values for a boundingBox to a pinsPic</summary>
///<param name="stringo">The string with the tuple</returns>
///<returns>A pinsPic</returns>
let processTuple (stringo:string)  : pinsPic =
    let arrayo = stringo.Split(',')
    let bmpoString =
        arrayo.[0]
        |> cleanString
    let bmpoo = new Bitmap(bmpoString)
    let featos = bmpoToFeatures bmpoo "RGB"
    let labelo = findLabelFromString arrayo.[0]
    let rectString =
        arrayo.[1]
        |> cleanString
        |> (fun x -> x.Split(' '))
        |> Array.filter (fun x -> not (x="") )
        |> Array.map (fun x -> int x)
    let centero = rectString.[0] + rectString.[2] / 2, rectString.[1] + rectString.[3] / 2
    let minHorDisto = List.min [bmpoo.Width - (fst centero); (fst centero)]
    let minVerDisto = List.min [bmpoo.Height - (snd centero); (snd centero)]
    {bmpo=bmpoo; label=labelo; features=featos; center=centero; minHorDist=minHorDisto; minVerDist=minVerDisto}

///<summary>Crops a bitmap to a subsect defined by points in the bitmap</summary>
///<param name="oldBmpo">The bitmap to be cropped</param>
///<param name="startX">Start of new cropped image on x-axis</param>
///<param name="endX">End of new cropped image on x-axis</param>
///<param name="startY">Start of new cropped image on y-axis - note bitmap counts from top to bottom</param>
///<param name="endY">End of new cropped image on y-axis</param>
///<returns>A cropped bitmap</returns>
```

```
let cropBmpo (oldBmpo:Bitmap) (startX:int) (endX:int) (startY:int) (endY:int) : Bitmap =
    let newBmpo = new Bitmap (endX-startX,endY-startY)
    let numPixos = (endX-startX)*(endY-startY)
    let findNewCoordinatesFromI (i:int): (int*int) =
        (i%(endX-startX),i/(endX-startX) )
    let findOldPixelFromI (i:int) =
        let oldCoords =
            findNewCoordinatesFromI i
            |> (fun k -> (fst k) + startX, (snd k) + startY)
        if (fst oldCoords) <= 0 then
            cWT ["fst oldcoords is less than or equal to zero"; (fst oldCoords |> string); (snd oldCoords |> string)]
        if (fst oldCoords) > oldBmpo.Width then
            cWT ["fst oldcoords is higher than Height"; (fst oldCoords |> string) ; (snd oldCoords |> string) ]
        if (snd oldCoords) <= 0 then
            cWT ["snd oldcoords is less than or equal to zero"; (fst oldCoords |> string); (snd oldCoords |> string) ]
        if (snd oldCoords) > oldBmpo.Height then
            cWT ["snd oldcoords is higher than or equal to zero"; (fst oldCoords |> string); (snd oldCoords |> string) ]
        oldBmpo.GetPixel(fst oldCoords, snd oldCoords)
    List.init
        numPixos
        (fun i -> newBmpo.SetPixel(findNewCoordinatesFromI i |> fst, findNewCoordinatesFromI i |> snd, findOldPixelFromI i) )
        |> ignore
    newBmpo

///<summary>Splits the pins-data set into train, valid and test</summary>
///<param name="picList">A list of pictures</param>
///<param name="validPct">The proportion to be allcoated for validation</param>
///<param name="testPct">The proportion to be allocated for testing</param>
///<returns>A tuple of three pic lists</returns>
let pinsSplitFunction (picList:pic list) (validPct:float) (testPct:float) : ( (pic list) * (pic list) * (pic list) ) =
    let numObs = List.length picList
    let indexes = newKFYShuffle numObs
    let numValid = findInt validPct numObs
    let numTest = findInt testPct numObs
    let numTrain = numObs - numValid - numTest
    let trainIndexes = indexes.[0..(numTrain-1)]
    let validIndexes = indexes.[numTrain..(numTrain+numValid-1)]
    let testIndexes = indexes.[(numTrain+numValid)..]
    let trainList = List.map (fun i -> picList.[i]) trainIndexes
    let validList = List.map (fun i -> picList.[i]) validIndexes
    let testList = List.map (fun i -> picList.[i]) testIndexes
    cWT ["Number of observations"; (string numObs); "train"; (string numTrain); "valid"; (string numValid); "test"; (string numTest)]
    trainList, validList, testList

///<summary>Crops a picture to a given dimension, putting the center of the face in the middle of the new picture</summary>
///<param name="horClipLevel">The distance from the center to the left and to the right side</param>
///<param name="verClipLevel">The distance from the center to the top and to the bottom</param>
///<param name="pico">The picture to be centered and cropped</param>
///<returns>A simple picture, maintaining label and features and discarding all the other information from the pinsPic</returns>
let cropPicoAroundCenter (horClipLevel:int) (verClipLevel:int) (pico:pinsPic) : pic =
        let startXo =
            pico.center
            |> fst
            |> (fun x -> x - horClipLevel)
        let endXo =
            pico.center
            |> fst
            |> (fun x -> x + horClipLevel)
        let startYo =
            pico.center
            |> snd
            |> (fun x -> x - verClipLevel)
        let endYo =
            pico.center
            |> snd
            |> (fun x -> x + verClipLevel)
        let newBmpo = cropBmpo pico.bmpo startXo endXo startYo endYo
        let featos = bmpoToFeatures newBmpo "RGB"
        {label= pico.label; features = featos}

///<summary>Sets a color of a pixel in a new, enlarged photo as the weighted average of the colors
/// of the four pixels that were closest in the original photo</summary>
```

```
///<param name="ulc">Color of the upper-left corner</param>
///<param name="urc">Color of the upper-right corner</param>
///<param name="llc">Color of the lower-left corner</param>
///<param name="lrc">Color of the lower-right corner</param>
///<param name="hf">The proportion of the distance between two pixels in the original bitmap</param>
///<param name="vf">The proportion of the distance between two pixels in the original bitmap</param>
///<returns>A color</returns>
let setColor (ulc:Color) (urc:Color) (llc:Color) (lrc:Color) (hf:float) (vf:float) : Color =
    let r =
        (float ulc.R) * hf + (float urc.R) * (1.0-hf) + (float llc.R) * vf + (float lrc.R) * (1.0-vf)
        |> (fun x -> x / 2.0)
        |> (fun x -> x + 0.5)
        |> int
    let g =
        (float ulc.G) * hf + (float urc.G) * (1.0-hf) + (float llc.G) * vf + (float lrc.G) * (1.0-vf)
        |> (fun x -> x / 2.0)
        |> (fun x -> x + 0.5)
        |> int
    let b =
        (float ulc.B) * hf + (float urc.B) * (1.0-hf) + (float llc.B) * vf + (float lrc.B) * (1.0-vf)
        |> (fun x -> x / 2.0)
        |> (fun x -> x + 0.5)
        |> int
    Color.FromArgb(0,r,g,b)

///<summary>Sets the color of a pixel in a new Bitmap as a function of an original Bitmap</summary>
///<param name="oldBmp">The original Bitmap</param>
///<param name="newBmp">The new Bitmap</param>
///<param name="i">The index of the particular pixel in the new Bitmap</param>
///<returns>Unit</returns>
let setNewPixel (oldBmp:Bitmap) (newBmp:Bitmap) (i:int) : unit =
    let wido =
        if i < (newBmp.Width*newBmp.Height) then
            i % newBmp.Width
        else
            newBmp.Width - 1
    let heigho =
        if i < (newBmp.Width * newBmp.Height) then
            i / newBmp.Width
        else
            newBmp.Height - 1
    let leftNeighbor =
        float wido
        |> (fun x -> x / (newBmp.Width - 1 |> float) )
        |> (fun x -> x * (oldBmp.Width - 1 |> float) )
        |> int
    let rightNeighbor =
        if wido < newBmp.Width - 1 then
            leftNeighbor + 1
        else
            leftNeighbor
    let upperNeighbor =
        float heigho
        |> (fun x -> x / (newBmp.Height - 1 |> float) )
        |> (fun x -> x * (oldBmp.Height - 1 |> float) )
        |> int
    let lowerNeighbor =
        if heigho < newBmp.Height - 1 then
            upperNeighbor + 1
        else
            upperNeighbor
    let horizontalFracto = 0.5
    let verticalFracto = 0.5
    let upperLeftCorner = oldBmp.GetPixel(leftNeighbor, upperNeighbor)
    let upperRightCorner = oldBmp.GetPixel(rightNeighbor, upperNeighbor)
    let lowerLeftCorner = oldBmp.GetPixel(leftNeighbor, lowerNeighbor)
    let lowerRightCorner = oldBmp.GetPixel(rightNeighbor, lowerNeighbor)
    let nextColor = setColor upperLeftCorner upperRightCorner lowerLeftCorner lowerRightCorner horizontalFracto verticalFracto
    newBmp.SetPixel(wido, heigho, nextColor)

///<summary>Builds a new Bitmap as a function of an existing Bitmap, enlarging to new dimensions</summary>
///<param name="oldBmp">The existing Bitmap</param>
```

```
///<param name="newWidth">The width of the new Bitmap</param>
///<param name="newHeight">The height of the new Bitmap</param>
///<returns>A Bitmap</returns>
let buildNewBitmap (oldBmp:Bitmap) (newWidth:int) (newHeight:int) : Bitmap =
    let newBmp = new Bitmap(newWidth, newHeight)
    List.init (newWidth * newHeight) (fun i -> setNewPixel oldBmp newBmp i) |> ignore
    newBmp

///<summary>Reads all filenames in a directory</summary>
///<param name="diro">Name of directory</param>
///<returns>A list of filenames</returns>
let getFileNamesFromDir (diro:string) : (string list) =
    System.IO.Directory.GetFiles(diro)
    |> List.ofSeq

///<summary>Reads filenames and writes them to a txt-file</summary>
///<returns>Unit</returns>
let createFilos : unit =
    cWT ["Reading filenames of the Pins data set and writing them to timFilos.txt"]
    let listOfDirectories =
        System.IO.Directory.GetDirectories("c:\\users\\timmo\\onedrive\\diku\\new_puk1\\pins")
        |> List.ofSeq
    let data set =
        Seq.collect getFileNamesFromDir listOfDirectories
        |> List.ofSeq
    let timFilos = System.IO.File.CreateText "timFilos.txt"
    let timFilosWrite (wo:System.IO.StreamWriter) (stringo:string) : unit =
        wo.WriteLine stringo
    let curryTimFilosWrite = timFilosWrite timFilos
    List.map curryTimFilosWrite data set |> ignore
    timFilos.Close()

///<summary>Extracts from a pinsPic the minimal horizontal distance from the center to either side
///and the minimal vertical distance from the center to either top or bottom</summary>
///<param name="pico">A pinsPic</param>
///<returns>A tuple of integers</returns>
```

```
let takeMinima (pico:pinsPic) : (int * int) =
    pico.minHorDist, pico.minVerDist

///<summary>Main function to execute the PINS-analysis</summary>
///<param name="inputo">Input parameters</param>
///<returns>Unit</returns>
let pins(inputo:inputparameters) =
    cWT ["Now analyzing the Pins data set "]
    cWT ["https://www.kaggle.com/data sets/hereisburak/pins-face-recognition"]
    cWT [System.DateTime.Now.ToString()]
    cWT ["First, all the filenames of the jpg-files in the Pins data set are read into a txt-file called filos.txt."]
    cWT ["This is done in the separate function called createFilos"]
    createFilos |> ignore
    cWT ["Secondly, a Python-programme named readFromOpenCV.py reads the files names and runs the OpenCV-module."]
    cWT ["This calculates the bounding boxes for each picture"]
    cWT
        ["The bounding box is a rectangle from the eyes to the chin area for each picture. readFromOpenCV.py writes\
        a new txt-file called tuple_file.txt."]
    cWT ["tuple_file.txt has for each picture a tuple with first the filename and secondly the coordinates of the bounding box."]
    cWT ["The below programme reads this information into the F#-programme and processes it further."]
    let tuples =
        System.IO.File.ReadLines inputo.fileName
        |> List.ofSeq
    cWT
        ["Now, the list of filenames is analyzed to find 10 random labels each of which must have no less than \
        100 pictures in the data set"]
    let top10Labels =
        tuples
        |> List.map findLabelFromString
        |> List.countBy id
        |> List.filter (fun x -> snd x >= 100)
        |> shuffleAndTake 10
        |> List.map (fun x -> fst x)
    cWT ["The list of filenames is reduced to only to 10 labels and 100 images per label"]
    let top10TupleList =
        List.init 10 (fun i -> List.filter (fun stringo -> (findLabelFromString stringo) = top10Labels.[i]) tuples |> shuffleAndTake 100)
    |> List.collect (fun x -> x)
printfn ""
let picoList = List.map processTuple top10TupleList
cWT ["As explained in the report, it is eventually decided not to enlarge the photos to obtain comparable sizes."]
cWT
    ["In order to demonstrate this capability, a photo is selected at random and enlarged by 50 per cent\
     in the horizontal dimension"]
cWT ["and by 7 per cent in the vertical dimension"]
cWT ["The original photo is saved as originalbmp.jpg and the enlarged photo is saved as enlarged.jpg"]
let randIndPico = rando.Next(List.length picoList)
let originalBmp = picoList.[randIndPico].bmpo
originalBmp.Save("originalbmp.jpg", System.Drawing.Imaging.ImageFormat.Jpeg)
let newWidth =
    originalBmp.Width
    |> float
    |> (fun x -> x * 1.5)
    |> (fun x -> x + 0.5)
    |> int
let newHeight =
    originalBmp.Height
    |> float
    |> (fun x -> x * 1.07)
    |> (fun x -> x + 0.5)
    |> int
cWT ["Enlarging picture of"; picoList.[randIndPico].label; "with an existing width of"; (string originalBmp.Width)]
cWT ["and and existing height of "; (string originalBmp.Height)]
cWT ["to a new width of "; (string newWidth); "and a new height of"; (string newHeight)]
let enlargedBmp = buildNewBitmap originalBmp newWidth newHeight
enlargedBmp.Save("enlargedbmp.jpg", System.Drawing.Imaging.ImageFormat.Jpeg)
cWT ["Finding the 10th percentile for horizontal and vertical distance between center and border"]
let minimaList = List.map takeMinima picoList
let horMinima =
    List.unzip minimaList
    |> fst
    |> List.sort
let verMinima =
        List.unzip minimaList
        |> snd
        |> List.sort
    cWT ["Lowest horizontal distance is"; (string horMinima.[0])]
    cWT ["Lowest vertical distance is"; (string verMinima.[0])]
    let horClipLevel = horMinima[List.length horMinima |> float |> (fun x -> x * 0.1) |> (fun x -> x + 1.0) |> int]
    let verClipLevel = verMinima[List.length verMinima |> float |> (fun x -> x * 0.1) |> (fun x -> x + 1.0) |> int]
    cWT ["10th percentile of horizontal distance is"; (string horClipLevel)]
    cWT ["10th percentile of vertical distance is"; (string verClipLevel)]
    let selectPico = List.filter (fun x -> x.minHorDist > horClipLevel && x.minVerDist > verClipLevel) picoList
    let curryCropPicoAroundCenter = cropPicoAroundCenter horClipLevel verClipLevel
    let cropPico = List.map curryCropPicoAroundCenter selectPico
    cWT ["We have this number of obs"; (List.length cropPico |> string)]
    let trainList, validList, testList = pinsSplitFunction cropPico inputo.validPct inputo.testPct
    cWT [" "]
    performAndReportTest trainList validList inputo.k |> ignore
    cWT [" "]

///<summary>Crops pinsPics around their center and returns pinsPics</summary>
///<param name="horClipLevel">The 10th percentile of horizontal distance between face center and border in the data set</param>
///<param name="verClipLevel">The same as horClipLevel but vertical</param>
///<param name="pico">The pinsPic to be cropped</param>
///<returns>A pinsPic which has been cropped around its center</returns>
let cropPicoAroundCenterMono (horClipLevel:int) (verClipLevel:int) (pico:pinsPic) : pinsPic =
        let startXo =
            pico.center
            |> fst
            |> (fun x -> x - horClipLevel)
        let endXo =
            pico.center
            |> fst
            |> (fun x -> x + horClipLevel)
        let startYo =
            pico.center
            |> snd
            |> (fun x -> x - verClipLevel)
        let endYo =
            pico.center
            |> snd
            |> (fun x -> x + verClipLevel)
        let newBmpo = cropBmpo pico.bmpo startXo endXo startYo endYo
        let featos = bmpoToFeatures newBmpo "RGB"
        {bmpo=newBmpo;
        label = pico.label;
        features = featos;
        center = pico.center;
        minHorDist = pico.minHorDist;
        minVerDist = pico.minVerDist;
        }

///<summary>Function for sanity check and reduction to monochromatic of Pins-data set</summary>
///<param name="inputo">Input parameters</param>
///<returns>Unit</returns>
let pinsMono(inputo:inputparameters) =
    cWT ["Now analyzing the Pins data set -sanity check and reduction to monochromatic scale "]
    cWT [System.DateTime.Now.ToString()]
    createFilos |> ignore
    let tuples =
        System.IO.File.ReadLines inputo.fileName
        |> List.ofSeq
    cWT
        ["Now, the list of filenames is analyzed to find 10 random labels each of which must have no less than \
        120 pictures in the data set"]
    let top10Labels =
        tuples
        |> List.map findLabelFromString
        |> List.countBy id
        |> List.filter (fun x -> snd x >= 120)
        |> shuffleAndTake 10
        |> List.map (fun x -> fst x)
    cWT ["The list of filenames is reduced to only to 10 labels and 120 images per label"]
    let top10TupleList =
    List.init 10 (fun i -> List.filter (fun stringo -> (findLabelFromString stringo) = top10Labels.[i]) tuples |> shuffleAndTake 120)
    |> List.collect (fun x -> x)
printfn ""
let picoList = List.map processTuple top10TupleList
cWT ["Finding the 10th percentile for horizontal and vertical distance between center and border"]
let minimaList = List.map takeMinima picoList
let horMinima =
    List.unzip minimaList
    |> fst
    |> List.sort
let verMinima =
    List.unzip minimaList
    |> snd
    |> List.sort
cWT ["Lowest horizontal distance is"; (string horMinima.[0])]
cWT ["Lowest vertical distance is"; (string verMinima.[0])]
let horClipLevel = horMinima[List.length horMinima |> float |> (fun x -> x * 0.1) |> (fun x -> x + 1.0) |> int]
let verClipLevel = verMinima[List.length verMinima |> float |> (fun x -> x * 0.1) |> (fun x -> x + 1.0) |> int]
cWT ["10th percentile of horizontal distance is"; (string horClipLevel)]
cWT ["10th percentile of vertical distance is"; (string verClipLevel)]
let selectPico = List.filter (fun x -> x.minHorDist > horClipLevel && x.minVerDist > verClipLevel) picoList
let curryCropPicoAroundCenterMono = cropPicoAroundCenterMono horClipLevel verClipLevel
let cropPico = List.map curryCropPicoAroundCenterMono selectPico
cWT ["Now randomly selecting 4 pictures cropped around  the center"]
let selectedFour =
    List.init 4 (fun i -> rando.Next(List.length cropPico))
    |> List.map (fun i -> cropPico.[i])
    |> List.map (fun x -> x.bmpo.Save(x.label+"cropC.jpg", System.Drawing.Imaging.ImageFormat.Jpeg))
cWT ["Now transforming features of pinsPics to monochromatic features"]
let pinsPicToMono (pico:pinsPic) : pic =
    {label=pico.label;
    features = bmpoToFeatures pico.bmpo "alpha"}
let monos = List.map pinsPicToMono cropPico
cWT ["We have this number of obs"; (List.length monos |> string)]
let trainList, validList, testList = pinsSplitFunction monos inputo.validPct inputo.testPct
cWT [" "]
```

```
    performAndReportTest trainList validList inputo.k |> ignore
    cWT [" "]

///7. Setting input parameters

let (mnistInputos:inputparameters) =
    {
    numObs = 5000;
    validPct = 0.2;
    testPct = 0.2;
    k = 5;
    firstNumPC = 10;
    fileName = "c://users//timmo//onedrive//diku//new_puk1//mnist//train.csv";
    }

///The number of observations used for the Pins-analysis is now set in the selection mechanism in the Pins-function above
let (pinsInputos:inputparameters) =
    {
    numObs = -7000; /// number of observations are now fixed by the 10x100-sorting in the pins-function
    validPct = 0.2;
    testPct = 0.2;
    k = 5;
    firstNumPC = 10;
    fileName = "c:\\users\\timmo\\onedrive\\diku\\new_puk1\\Master\\tuple_file.txt";
    }

///8. Executing programme
executeKNNAndPCA mnistInputos
twoDims mnistInputos
pins pinsInputos
pinsMono pinsInputos
timsOutput.Close()

///9. Farewell-statement to user
printfn "Execution of app succesfully concluded"
printfn "%s" (System.DateTime.Now.ToString())
```

11.2 Python-based script readFromOpenCV.py

This code is available on https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/read_from_openCV.py

```python
import os
import random
import cv2
import numpy as nd

os.chdir("c:\\users\\timmo\\onedrive\\diku\\new_puk1\\F21052022")
filos = open("filos.txt","r")
filos_string = filos.read()
filos_list = filos_string.split('\n')
filos_list = filos_list[:-1]
filos_list = filos_list[:100]
rand_ind = random.randint(0,len(filos_list))

print("first element in file is " + filos_list[0])
print("last element in file is" + filos_list[-1])
print("random element from filos_list is image number " + str(rand_ind))
print(filos_list[rand_ind])

##Source: https://realpython.com/face-recognition-with-python/

def find_rec(image_name):
    casc_path = "c:\\users\\timmo\\onedrive\\diku\\new_puk1\\faceDetectCV2\\haarcascade_frontalface_default.xml"
    faceCascade = cv2.CascadeClassifier(casc_path)
    imgo = cv2.imread(image_name)
    imgo_gray = cv2.cvtColor(imgo, cv2.COLOR_BGR2GRAY)
    faco = faceCascade.detectMultiScale(
        imgo_gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
        flags = cv2.CASCADE_SCALE_IMAGE
    )
    return faco
```

```
faco_list = []
for image_name in filos_list:
    ##print("generating bounding box for this image "+image_name)
    faco_list.append(find_rec(image_name))

print("type of first element in faco_list is")
faco_1 = faco_list[0]
print(type(faco_list[0]))
for i in range(4):
    print(faco_1[0][i])

print("First element in faco_list is")
print(faco_list[0])
print("Last element in faco_list is")
print(faco_list[-1])
print("The random element in faco_list is")
print(faco_list[rand_ind])

def testo(N):
    for i in range(N):
        rand_ind = random.randint(0,len(filos_list))
        next_image = cv2.imread(filos_list[rand_ind])
        next_bounding_box = faco_list[rand_ind]
        first_coordinate = next_bounding_box[0][0]
        second_coordinate = next_bounding_box[0][1]
        width = next_bounding_box[0][2]
        height = next_bounding_box[0][3]
        cv2.rectangle(next_image,(first_coordinate, second_coordinate), (first_coordinate+width, second_coordinate+height), (0,255,0), 2)
        cv2.imshow("hallo", next_image)
        cv2.waitKey(0)
        file_name = "testing_rectangle_"+str(rand_ind)+".jpg"
        cv2.imwrite(file_name, next_image)

####Nu er funktionaliteten testet på et mindre delmængde. Nu udvikles et program, der kan køre på hele listen,
###og som skal aflevere en tuple bestående af *.jpg-filnavn og rektangel-koordinater
```

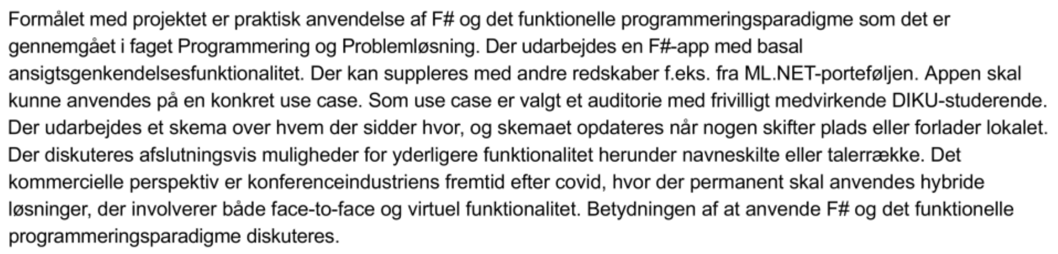
```
filos = open("filos.txt","r")
filos_string = filos.read()
filos_list = filos_string.split('\n')
tuple_list = []
could_process = 0
could_not_process = 0
tuple_file = open("tuple_file.txt","w")
for pic in filos_list[:100]:
    rec = find_rec(pic)
    try:
        rec_string = str(rec[0][0])+" "+str(rec[0][1])+" "+str(rec[0][2])+" "+str(rec[0][3])
        print("Picture " + pic + " has this rectangle "+rec_string)
        tuplo = (pic, rec_string)
        tuple_file.write(str(tuplo)+"\n")
        print(type(rec))
        tuple_list.append((pic,rec_string) )
        could_process += 1
    except:
        print("could not process this rectangle")
        print(type(rec))
        print(str(rec))
        could_not_process += 1

print("tuple_list has this length "+str(len(tuple_list)))
print("was able to process this number of files "+str(could_process))
print("Was unable to process this number of files "+str(could_not_process))

def testo2(N):
    for i in range(N):
        rand_ind = random.randint(0,len(tuple_list))
        print(tuple_list[rand_ind])

testo2(3)
```

12 Appendix III. Programme output

This is the output generated by executing the app.fsx-programme.

```
Reading filenames of the Pins data set and writing them to timFilos.txt
Performing k-nearest neighbor analysis on MNIST
10/08/2022 15.45.16
Number of observations: 5000  train:  3000  valid:  1000  test:  1000
reporting on trainMat
matrix has rows: 3000 columns: 784 rank: 628
all matrix values are between 0 and 256
Number of rows of covariance matrix 784
Number of columns of covariance matrix: 784
Rank of covariance matrix 628
Now performing comparisons of ordinary KNN and PCA for the following number of principal components 10
Multiplying a matrix of 3000 rows and 784 columns with eigenvectormatrix with 784 rows and 10 columns
returning a matrix of projected images with 3000 rows and 10 columns

Multiplying a matrix of 1000 rows and 784 columns with eigenvectormatrix with 784 rows and 10 columns
returning a matrix of projected images with 1000 rows and 10 columns

trainProjectMat has 3000 rows and 10 columns
validProjectMat has 1000 rows and 10 columns

Performing regular k-nearest neighbor validation of model on untransformed data sets with 784 features
Total number of results 1000
Total number of correct 915
Percentage correct 91.5
Label 0 number of obs 100  precision 0.9603960396039604 recall 0.97
Label 1 number of obs 102  precision 0.864406779661017 recall 1
Label 2 number of obs 114  precision 0.9809523809523809 recall 0.9035087719298246
Label 3 number of obs 94  precision 0.9081632653061225 recall 0.9468085106382979
Label 4 number of obs 107  precision 0.9038461538461539 recall 0.8785046728971962
Label 5 number of obs 90  precision 0.9101123595505618 recall 0.9
Label 6 number of obs 100  precision 0.9259259259259259 recall 1
Label 7 number of obs 97  precision 0.9213483146067416 recall 0.845360824742268
Label 8 number of obs 107  precision 0.9886363636363636 recall 0.8130841121495327
Label 9 number of obs 89  precision 0.8 recall 0.898876404494382

Performing PCA validation of model on transformed data sets with 10 features
Total number of results 1000
Total number of correct 870
Percentage correct 87
Label 0 number of obs 100  precision 0.9207920792079208 recall 0.93
Label 1 number of obs 102  precision 0.9026548672566371 recall 1
Label 2 number of obs 114  precision 0.9541284403669725 recall 0.9122807017543859
Label 3 number of obs 94  precision 0.8809523809523809 recall 0.7872340425531915
Label 4 number of obs 107  precision 0.7913043478260869 recall 0.8504672897196262
Label 5 number of obs 90  precision 0.8478260869565217 recall 0.8666666666666667
Label 6 number of obs 100  precision 0.9065420560747663 recall 0.97
Label 7 number of obs 97  precision 0.9010989010989011 recall 0.845360824742268
Label 8 number of obs 107  precision 0.8365384615384616 recall 0.8130841121495327
Label 9 number of obs 89  precision 0.7380952380952381 recall 0.6966292134831461

Now generating visual illustration of the data in two-dimensional space
10/08/2022 15.51.40
Number of observations: 5000  train:  3000  valid:  1000  test:  1000
The eigenvector matrix is generated
Number of rows of covariance matrix 784
Number of columns of covariance matrix: 784
Rank of covariance matrix 628
Multiplying a matrix of 1000 rows and 784 columns with eigenvectormatrix with 784 rows and 2 columns
returning a matrix of projected images with 1000 rows and 2 columns

Generating plot and publishing to browser
Generating plot and publishing to browser
Generating plot and publishing to browser
Generating plot and publishing to browser
Generating plot and publishing to browser
Now analyzing the Pins data set
https://www.kaggle.com/data sets/hereisburak/pins-face-recognition
10/08/2022 15.53.13
First, all the filenames of the jpg-files in the Pins data set are read into a txt-file called filos.txt.
```

```
This is done in the separate function called createFilos
Secondly, a Python-programme named readFromOpenCV.py reads the files names and runs the OpenCV-module.
This calculates the bounding boxes for each picture
The bounding box is a rectangle from the eyes to the chin area for each picture. readFromOpenCV.py writes\
        a new txt-file called tuple_file.txt.
tuple_file.txt has for each picture a tuple with first the filename and secondly the coordinates of the bounding box.
The below programme reads this information into the F#-programme and processes it further.
Now, the list of filenames is analyzed to find 10 random labels each of which must have no less than 100 pictures in the data set
The list of filenames is reduced to only to 10 labels and 100 images per label
As explained in the report, it is eventually decided not to enlarge the photos to obtain comparable sizes.
In order to demonstrate this capability, a photo is selected at random and enlarged by 50 per centin the horizontal dimension
and by 7 per cent in the vertical dimension
The original photo is saved as originalbmp.jpg and the enlarged photo is saved as enlarged.jpg
Enlarging picture of camila_mendes with an existing width of 102
and and existing height of  108
to a new width of  153 and a new height of 116
Finding the 10th percentile for horizontal and vertical distance between center and border
Lowest horizontal distance is 42
Lowest vertical distance is 27
10th percentile of horizontal distance is 50
10th percentile of vertical distance is 53
We have this number of obs 859
Number of observations 859 train 515 valid 172 test 172

Total number of results 172
Total number of correct 42
Percentage correct 24.418604651162788
Label Adriana_Lima number of obs 24  precision 0.17647058823529413 recall 0.25
Label Dominic_Purcell number of obs 12  precision 0.3333333333333333 recall 0.3333333333333333
Label Eliza_Taylor number of obs 16  precision 0.29411764705882354 recall 0.3125
Label Emma_Watson number of obs 13  precision 0.21428571428571427 recall 0.23076923076923078
Label Gwyneth_Paltrow number of obs 21  precision 0.2727272727272727 recall 0.14285714285714285
Label Jason_Momoa number of obs 18  precision 0.25 recall 0.16666666666666666
Label Katherine_Langford number of obs 21  precision 0.26666666666666666 recall 0.38095238095238093
Label Miley_Cyrus number of obs 22  precision 0.3157894736842105 recall 0.2727272727272727
Label camila_mendes number of obs 14  precision 0.16666666666666666 recall 0.14285714285714285
Label gal_gadot number of obs 11  precision 0.18181818181818182 recall 0.18181818181818182

Now analyzing the Pins data set -sanity check and reduction to monochromatic scale
10/08/2022 16.17.43
Now, the list of filenames is analyzed to find 10 random labels each of which must have no less than 120 pictures in the data set
The list of filenames is reduced to only to 10 labels and 120 images per label
Finding the 10th percentile for horizontal and vertical distance between center and border
Lowest horizontal distance is 40
Lowest vertical distance is 33
10th percentile of horizontal distance is 49
10th percentile of vertical distance is 52
Now randomly selecting 4 pictures cropped around  the center
Now transforming features of pinsPics to monochromatic features
We have this number of obs 1030
Number of observations 1030 train 618 valid 206 test 206

Total number of results 206
Total number of correct 22
Percentage correct 10.679611650485436
Label Brie_Larson number of obs 22  precision NaN recall 0
Label Eliza_Taylor number of obs 26  precision NaN recall 0
Label Emilia_Clarke number of obs 15  precision NaN recall 0
Label Jessica_Barden number of obs 18  precision NaN recall 0
Label Maisie_Williams number of obs 19  precision NaN recall 0
Label Rami_Malek number of obs 20  precision NaN recall 0
Label Robert_Downey_Jr number of obs 16  precision NaN recall 0
Label alycia_dabnem_carey number of obs 18  precision NaN recall 0
Label camila_mendes number of obs 30  precision NaN recall 0
Label ellen_page number of obs 22  precision 0.10679611650485436 recall 1
```


Formålet med projektet er praktisk anvendelse af F# og det funktionelle programmeringsparadigme som det er gennemgået i faget Programmering og Problemløsning. Der udarbejdes en F#-app med basal ansigtsgenkendelsesfunktionalitet. Der kan suppleres med andre redskaber f.eks. fra ML.NET-porteføljen. Appen skal kunne anvendes på en konkret use case. Som use case er valgt et auditorie med frivilligt medvirkende DIKU-studerende. Der udarbejdes et skema over hvem der sidder hvor, og skemaet opdateres når nogen skifter plads eller forlader lokalet. Der diskuteres afslutningsvis muligheder for yderligere funktionalitet herunder navneskilte eller talerrække. Det kommercielle perspektiv er konferenceindustriens fremtid efter covid, hvor der permanent skal anvendes hybride løsninger, der involverer både face-to-face og virtuel funktionalitet. Betydningen af at anvende F# og det funktionelle programmeringsparadigme diskuteres.

Figur 15: Objective

13 Appendix IV. Research Diary

The objective of this project is shown in Figure 15 below

In order to analyze whether and how F# and the functional programming paradigm is well suited for facial recognition, it has been suggested to record how the project and the programming of the app have evolved over time, in particular how the programming becomes more functional to respond to the challenges of developing the app. The programming approaches that were attempted reflect hypotheses of how the best way of approaching the project. Therefore, a research journal will be kept. This research journal will be chronological and will later be supplemented by more analytical discussions.

13.1 Begining of project

The project began when I followed the course 'Programmering og Problemløsning' in the first year of the BSc programme Machine Learning and Data Science at DIKU. Having a background in Python, I found that F# and functional programming had a high threshold for entry for new programmers and that several of the principles were not intuitive. I met with Professor Jon Sporring on 15 November 2021 and we discussed this, Jon Sporring made the point that functional programming was in fact the best for a number of data science purposes involving very large data sets. Python and imperative programming might have a lower entry point but students might also pick up a number of bad habits in the form of programming approaches that would work for small but not for large data sets.

At the end of the semester, I again reached out to Jon Sporring suggesting an extra-curricular project in F#. The original idea was to implement a catalogue of standard algorithms in F#. Jon Sporring instead suggested work of more direct relevance to data science, in the form of either data visualization, facial recognition or other forms of machine learning.

By February 2021, we had agreed the above objective and started a weekly cadence of supervision meetings and programming work. The various stages of programming were documented in this research diary, in a Github repository and via correspondance in a Slack channel.

13.2 Week 1 - 2 February 2022

dotnet 6 was selected rather than the mono environment. It presented advantages in the form of easy access to installation of new modules via the integration with nuget.

Ronneberger (2015) was reviewed and it was decided to start with literature that took a more fundamental approach to the issue.

It was decided to start with basic functionality to analyze the MNIST data set of handwritten digits. The MNIST data set is available on Kaggle in the form of 42.000 labelled data points.

13.3 Week 2 - 9 February 2022

The 'k nearest neighbor' technique for image recognition was discussed.

13.4 Week 3 - 16 February 2022

The first version of a programme was presented with the title kNearNeighb.fsx, it is available on `https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/kNearNeighb.fsx`.

The programme was able to run without error and arrive at plausible values for precision and recall. The programme followed the documentation standard.

The programme was in F# but largely reflected an imperative approach.
The programme first read the entire Kaggle data set from a csv-file, then selected a given number of observations for actual analysis. Due to time limitations, the programme was not able to process all 42.000 data points. The structure of the MNIST data set was maintained in the programme with data points in the form of integer lists. The first element of the list was the label and the following elements were the features.

The programme contained 5 loops. One loop read the index and randomly assigned a designation of 'train' or 'test' to each data point. One loop then assigned the data points into the appropriate data subsets based on this designation. One loop read all elements of the train data set and calculated the distance to a given test observation. One loop read the test observations and called the loop that calculated the distances. One loop read all predictions for test observations and augmented the score if the prediction was correct. One loop read the correct scores and assigned into two lists called respectively 'correct by prediction' that served to calculate precision and 'correct by actual' that served to calculate recall.

The programme contained one recursive function that read the comma separated strings and converted them to integer lists.

The programme ran in interactive mode and did not take advantage of the compile functionality in dotnet 6.

The advantages of the programme was manageability and familiarity for the programmer. It was discussed with the supervisor that more could be done to reflect the functional paradigm. The primary objection by the supervisor was that the programme first read a large number of data points from file then selected a much smaller number (500 vs 42.000). It was suggested to rely more on the F# Sequence data structure that only carries out instructions on individual collection elements when that element is required by the programme. It was also discussed to rely more on module calls.

13.5 Week 4 - 23 February 2022

The programme kNNImproved.fsx was presented `https://github.com/TimMondorf/PUK-Image-Recognition-FSharp` `blob/main/kNNImproved.fsx`. The programme contained one loop statement that printed the results per category.

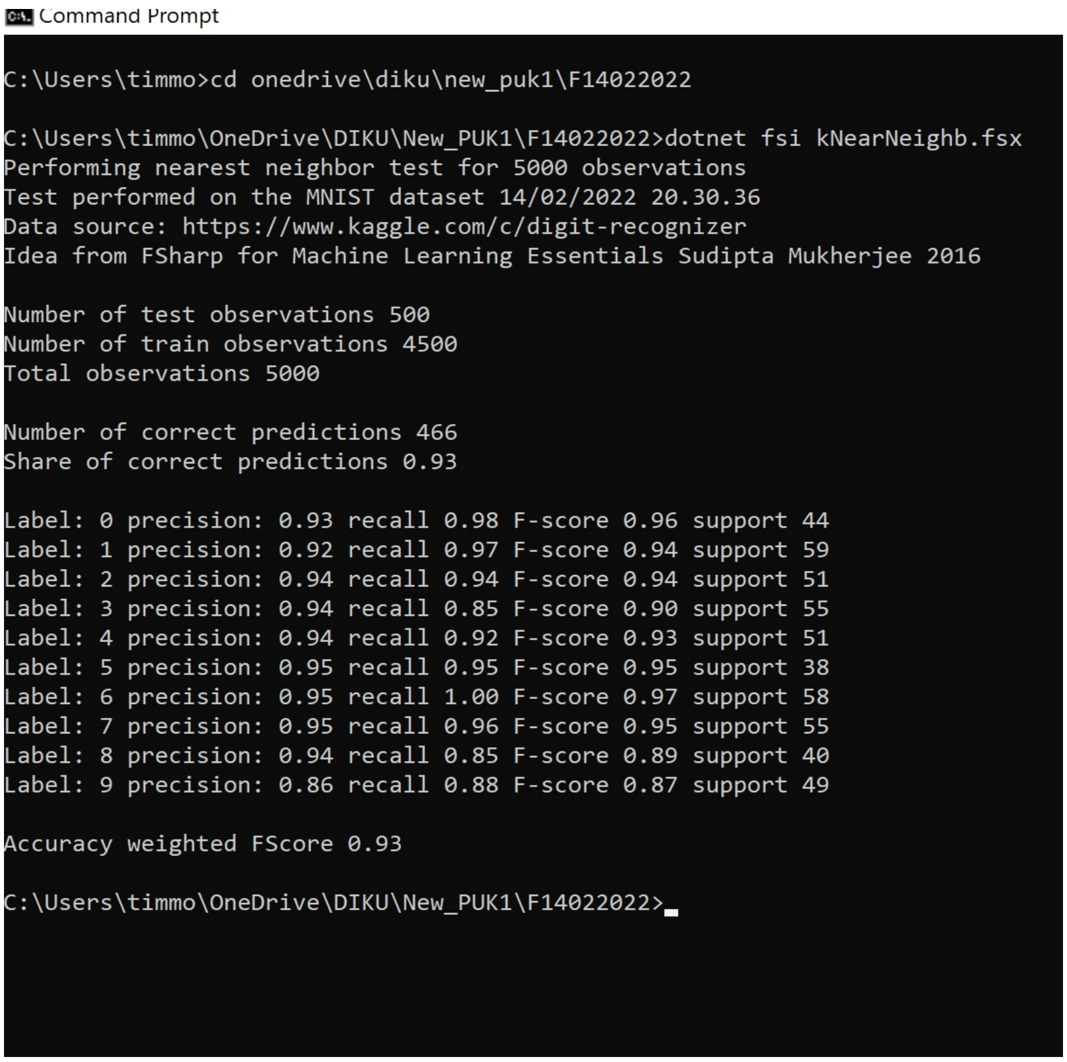
The programme contained 4 recursive functions. One function read the comma-separated strings and converted them to integer lists. One function calculated the distances from each element in the training set to a given element in the test set. One function performed insertion sort of a list in order to rank the train observations according to their distance from a given test observation. One function read test observations and returned a tuple with the actual and predicted value.

The supervisor suggested to rely even more on module calls for instance for the purpose of sorting lists.

The results of the programme are shown in Figure 16
It was verified that the output was plausible, i.e. precision and recall values between 0 and 1 and closer to 1 with a larger number of observations. Results for the subsequent versions of the programme are also available. While the model was under construction, focus was on programming with a sanity check of results.

13.6 Week 5 - 2 March 2022

The programme kNN_01032022.fsx was presented `https://github.com/TimMondorf/PUK-Image-Recognition-FSharp` `blob/main/kNN_01032022.fsx`. The programme read an integer which was the required number of observations, then defined a sequence of that length consisting of data points from the MNIST data



```
C:\Users\timmo>cd onedrive\diku\new_puk1\F14022022

C:\Users\timmo\OneDrive\DIKU\New_PUK1\F14022022>dotnet fsi kNearNeighb.fsx
Performing nearest neighbor test for 5000 observations
Test performed on the MNIST dataset 14/02/2022 20.30.36
Data source: https://www.kaggle.com/c/digit-recognizer
Idea from FSharp for Machine Learning Essentials Sudipta Mukherjee 2016

Number of test observations 500
Number of train observations 4500
Total observations 5000

Number of correct predictions 466
Share of correct predictions 0.93

Label: 0 precision: 0.93 recall 0.98 F-score 0.96 support 44
Label: 1 precision: 0.92 recall 0.97 F-score 0.94 support 59
Label: 2 precision: 0.94 recall 0.94 F-score 0.94 support 51
Label: 3 precision: 0.94 recall 0.85 F-score 0.90 support 55
Label: 4 precision: 0.94 recall 0.92 F-score 0.93 support 51
Label: 5 precision: 0.95 recall 0.95 F-score 0.95 support 38
Label: 6 precision: 0.95 recall 1.00 F-score 0.97 support 58
Label: 7 precision: 0.95 recall 0.96 F-score 0.95 support 55
Label: 8 precision: 0.94 recall 0.85 F-score 0.89 support 40
Label: 9 precision: 0.86 recall 0.88 F-score 0.87 support 49

Accuracy weighted FScore 0.93

C:\Users\timmo\OneDrive\DIKU\New_PUK1\F14022022>
```

Figur 16: Results for the programme version kNNImproved set.

The programme contained no recursive statements since these had been replaced by module calls. The reading of comma-separated strings and conversion to integer lists was performed by repeated use of the List.map and the List.filter commands. The calculation of distances was handled by repeated use of List.map functions, and eventually the List.map2 function was used to process a list of first-axis coordinates against a list of second-axis coordinates. The List.sortByDescending function was used instead of insertion sort. The List.init function was used to generate a list of tuples of results, the first element being the actual value and the second element being the predicted value. The List.init-function took as its argument the index, read the relevant test observation and calculated the distances in order to generate a prediction. It therefore returned a list of prediction which was the same length as the list of test observations.

The programme introduced a number of custom record types. A picture was a record type that had a label in the form of a string and features in the form of a separate integer list. It was therefore clearer that the label played a separate role and was not just one integer among many. A neighbor was a custom record type that had a label and a distance to a given test observation. A result was a record type that had one string for actual value and one string for predicted value.

The programme contained two loops. One loop performed the random assignment of data points into train and test. One loop printed precision and recall values for each category of labels.

To avoid the first loop, the supervisor suggested applying the Knuth-Fisher-Yates randomization algorithm to 'shuffle' the data set https://stackoverflow.com/questions/17455909/how-to-generate-a-specific 17457044#17457044.The supervisor suggested using an appropriate string or data structure for printing to avoid the second loop. He finally suggested increased reliance on piping rather than successive assignment into generic intermediate variables.

13.7 Weed 6 - 9 March 2022

The programme kNN_09032022.fsx was presented https://github.com/TimMondorf/PUK-Image-Recognition-FSharp blob/main/kNN_09032022.fsx.

The programme contained no recursive statements and only one loop since a loop is prescribed in the Knuth-Fisher-Yates shuffling algorithm.

The functions were now rewritten using the piping operator rather than parentheses and intermediate variables.

The list of comma-separated strings was shuffled according to the Knuth-Fisher-Yates algorithm. The Knuth-Fisher-Yates algorithm reads an array and for each index picks a random number number in the range of array indexes, then swaps the element at the index with the element at the index that has been randomly generated. The underlying random integer generator was the F# System.Random().

A new custom record type was defined called pic2 which was a picture with an additional attribute of 'train/valid/test'. Once this attribute was assigned using the shuffle described above, the train and valid data subsets were created using the List.filter module call.

A new custom record type was defined called scoresPerCategory containing a label, a number of observations and values for precision and recall. The print loop was now replaced by a logical statement that verified that the relevant values were available for all label categories. If this condition was met, a list of scoresPerCategory was created with scores for each label. Then a custom function was applied to the list that for each record in the list read all relevant attributes and printed them. That way, all test scores per category were printed successively. This was an example of a List.map call that returns unit rather than a value. If the logical condition was not met, a dummy function that also returned unit was called. This was necessary to meet the F# principle that all branches of a logical statement must return the same type.

In the meeting, it was also discussed how to run the programme in compile mode. It is required to specify 'dotnet new console language F#' since F# is not the default, then the name of the fsx-file must be populated in the file with the suffix .fsproj. It is possible to separate the programme into an .fs and an .fsx-file, both file names must then be populated with the .fsx-file as the latter entry. Separation into .fs and .fsx could not be combined with calling other modules with the command

```
#r "nuget:..."
```

It was suggested that as a next step, dimensional complexity should be reduced using the Principal Component Analysis (PCA) which is well suited for image recognition.

Discussion
Model performance in terms of predictive power remained relatively stable over the course of these model modifications. The large improvements came from reading more data points.

Through these successive modifications, the programme had come much closer to functional paradigm. The initial programme had 173 lines and the final programme 230, but the initial programme had long involved statements whereas the final programme has more lines but containing shorter, simpler statements.

The issue of running time was discussed with the supervisor, who commented that without optimization of the programmes it was difficult to conclude.

What has been achieved on this journey toward functional programming can therefore not be reduced to a discussion of execution time or model performance. Rather, a programming approach has been implemented where functions are first-class citizens, where data is being pushed into immutable types that are defined by their purpose in the analysis. Greater clarity and simplicity will follow from this approach.

13.8 Week 7 - 16 March 2022

A first attempt at a programme that performed PCA was presented. Substantial work had gone into identifying the appropriate external module to perform PCA in F#. I did consult the official F# Discord group `https://discord.com/channels/196693847965696000/386375157918466069` as well as review material available on the FSlab.org server. Several external modules exist such as Deedle, FSharp.Stats, FSharp.Data and MathNet.Numerics. The dotnet ecosystem for F# will be discussed more in detail later in the report.

The programmes `https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/pCA_16032022.fs` and `https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/pCA_16032022.fsx` were presented.

The idea was to maintain as much as possible of the data processing functionality that had already been developed and then add specific functionality that could perform PCA.

The programme calculated the covariance-matrix by converting the feature list to floats, then defining a function that could be called by List.map2 on two float lists to calculate the covariance. A list of lists of covariances was calculated by nesting two calls to List.init. It was discussed with the supervisor that this approach did not take advantage of the symmetry of any covariance-matrix but calculated all covariances twice. An alternative approach would have required a mutable data structure and a logical loop that for each cell would first verify if that particular covariance had been calculated by checking the mirrored cell across the diagonal, and then either copying that value or calculating a value. Such an approach would conflict with several principles of functional programming such as immutable constants and loops.

The programme contained functionality that measured execution time to allow for comparison between dotnet 6 interactive and compiled mode and the supervisor commented that execution time should always be measured externally from the programme.

It was concluded that the appropriate modul for PCA in F# was MathNet.Numerics but that it should be kept in mind that the module delivered its results in object form and that for instance eigenvalues should then be read as attributes of these objects.

13.9 Week 8 - 23 March 2022

A familiarization with the MathNet.Numerics-module was conducted and discussed again in the supervisor meeting on 23 March. A matrix was created and the eigenvalues were read from the object.

13.10 Week 9 - 30 March 2022

The programme https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/pCA25032022.fsx performs a Principal Component Analysis (PCA).

Consideration went into whether the features should be standardized first, i.e. for each image the average and standard deviation of the numeric values should be calculated and for each value, the average should be subtracted and the value should be divided by the standard deviation. This would ensure that all pictures, represented as a list of floats, would have the average of zero and the variance of 1.

This technique is applied to models where the different features have different types, for instance a financial model can have interest rates in percentage points and GDP in USD. The variation in one feature may therefore appear more important than others. This is less of a concern since here all features have the same type, the color in a specific pixel on a scale from 0 to 255. Still, it was decided to perform this standardization. In programming terms, the standardization was quite easy to perform by simply replacing the custom record type pic2 with a new custom record type pic3 which in addition to the attributes of pic2 also had normalized features. The normalization was performed by a separate function which was called with the List.init function and processed the original features. The rest of the programme did not need to be updated.

A custom record type eigValVectors was created which was a vector of eigenvalues and a list of eigenvectors. A function genEigenValuesAndVectors transformed the covariance matrix from a list of lists of floats into a MathNet.Numerics matrix and called the .Evd() operator that creates an object of eigenvalues and eigenvectors. Eigenvalues were then read with the addition that only the Real component was relevant, and eigenvectors were read with the .Column(i) command. This manipulation required experience with MathNet.Numerics. The result was returned as an eigValVectors record.

It was noted that as predicted by theory, eigenvalues vary in magnitude and when the list of eigenvalues is sorted according to value, values tend to drop off sharplys. The below output from the programme pca_25032022.fsx shows the eigenvalues of covariance matrix for a randomly selected sub data set of 300 observations that have been standardized. The largest value is 57, the second largest is zero with 5 zeros after the decimal point, the next 17 eigenvalues are zero with 7 zeros after the decimal point.
It was discovered that actually MathNet.Numerics returns eigenvalues in sorted order with the maximal element first and that eigenVectors are returned accordingly, so it was not necessary to define an separate maximum priority queue in the programme.

It was decided to arbitrarily select the three most important principal components. Each principal component is a line in 784-dimensional space. The train and test observations are projected onto each principal component as defined in Henrik Laurberg Petersen Lineær Algebra i Datalogi Definition 4.6 p. 205. This means that their distances can now be calculated in 3 unidimensional spaces. The function projectPointOntoLineAndFindDistance returns a new float list of 784 elements that can be interpreted as a point along the line. Since all project points lie along the same line, their position can now be reduced to the distance from origo, so one float number instead of 784, and their distances can be calculated as the deltas of this number. This distance can be interpreted as the distance that must be travelled from origo along the principal component before the projection of the point has been reached.

A new custom record type is created called pic4 which is similar to pic3 but with 3 additional attributes which are the distances from origo along the 3 principal components. A function makePic4 curries the 3 principal components into 3 respective versions of projectPointOntoLineAndFindDistance and applies these 3 functions to calculate the distances. A function makePic4List can be applied at a data set level to transform from pic3 to pic4. In this function, the 3 principal components are transformed from MathNet.Numeric vectors (which are actually seqs) into lists with the List.ofSeq-command. The programme thus wrapped the objects from MathNet.Numerics in sequences and lists to be manipulated in a transparent way in the programme.
The generateNeighbor-function was now simpler. Where in previous versions of the programme, distances between two points was calculated as the Euclidean distance in 784-dimensional space, it now suffices to subtract distances along the first pca from each other, distances from the second pca from each other and distances from the third pca from each other. These 3 distances are squared, added together and the square root is taken. This new distance can be interpreted as the distance between the points in a 3-dimensional space made up of the 3 principal components. Simplification from 784 dimensions to 3 dimensions has therefore been achieved.
It now becomes possible to reapply the functionality from previous versions of the programme. The 5 nearest neighbors are identified in the same way and test results and test scores are generated and reported in the same way.
In the initial iterations of the programme, model performance became so low that the test score function had to be rewritten. As a use case, if the list of correct results does not include one single "9"then the list of correct per category will simply be one element shorter. Therefore, the scoring functions had to not just sort the correct by category but actively count the frequence of each category even when it was zero.

The output of the programme is shown below

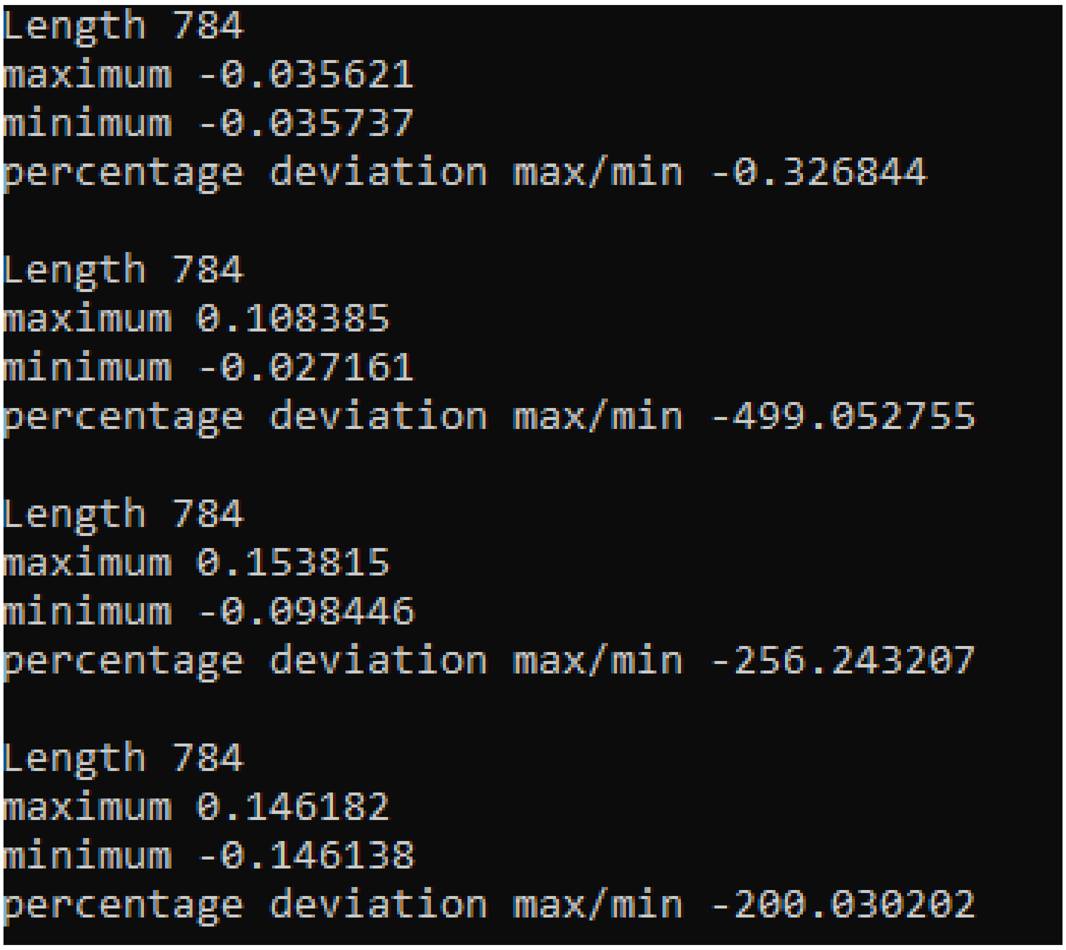
```
C:\Users\timmo\OneDrive\DIKU\New_PUK1\F25032022>dotnet fsi pCA25032022.fsx
Test performed on the MNIST data set 25/03/2022 09.58.46
Data source: https://www.kaggle.com/c/digit-recognizer
Idea from FSharp for Machine Learning Essentials Sudipta Mukherjee 2016

Total number of observations in data set 500

Total number of observations in data set 500
- of which used for training 300
- of which used for validation 100
- of which set aside for final testing (not used currently) 100

Test performed on the MNIST data set 25/03/2022 09.58.47
Data source: https://www.kaggle.com/c/digit-recognizer
Idea from FSharp for Machine Learning Essentials Sudipta Mukherjee 2016

- of which used for training 300
- of which used for validation 100
- of which set aside for final testing (not used currently) 100

all data elements have same length
This is the number of rows of the cov-matrix 784
This is the number of cols of the cov-matrix 784
Verifying that not all values of cov-matrix are zero

44687.78
These are the eigenvalues [56.99975826; -3.710888507e-05; 7.260099922e-07; 5.515936583e-07;
 4.046737843e-07; 3.763601606e-07; 3.166140151e-07; 2.309797031e-07;
 2.113883228e-07; 2.024679417e-07; 1.818260851e-07; 1.670183075e-07;
 1.461462031e-07; 1.391204946e-07; 1.341924604e-07; 1.253783098e-07;
 1.148028058e-07; 1.097190312e-07; 1.012362636e-07; 9.25979657e-08;
 9.182788005e-08; 8.328610967e-08; 8.173088182e-08; 7.933886636e-08;
 7.527765396e-08; 7.22632425e-08; 6.474464415e-08; 5.976315682e-08;
 6.252747349e-08; 5.690436881e-08; 5.527643183e-08; 5.307265533e-08;
 4.943884511e-08; 4.792138658e-08; 4.850805267e-08; 4.424928494e-08;
 4.343219766e-08; 4.119873269e-08; 3.964946373e-08; 3.796459126e-08;
 3.772708582e-08; 3.564969886e-08; 3.483025909e-08; 3.318738892e-08;
 3.135069201e-08; 3.077616772e-08; 2.928118146e-08; 2.866737454e-08;
 2.813926819e-08; 2.749433108e-08; 2.620319115e-08; 2.550994494e-08;
 2.413252753e-08; 2.3674489e-08; 2.255781203e-08; 2.202888263e-08;
 2.164240256e-08; 2.083455357e-08; 2.017470361e-08; 1.900520931e-08;
 1.862114838e-08; 1.868759811e-08; 1.815392578e-08; 1.76644689e-08;
 1.687621231e-08; 1.650349827e-08; 1.592365778e-08; 1.577342589e-08;
 1.506132509e-08; 1.470797084e-08; 1.447833365e-08; 1.413995673e-08;
 1.38383998e-08; 1.33041584e-08; 1.301042404e-08; 1.248861922e-08;
 1.203838457e-08; 1.197027639e-08; 1.168027119e-08; 1.162896815e-08;
 1.14160802e-08; 1.09718843e-08; 1.047067608e-08; 1.021828945e-08;
 9.912618824e-09; 9.607572288e-09; 9.438745462e-09; 9.195542686e-09;
 9.093753407e-09; 9.001258521e-09; 8.634536666e-09; 8.831585755e-09;
 8.25099084e-09; 8.106348364e-09; 8.013152528e-09; 7.885836313e-09;
 7.720015533e-09; 7.226490535e-09; 7.528348701e-09; 7.008472999e-09; ...]

This is the first principal component seq [-0.03573747934; -0.03573747934; -0.03573747934; -0.03573

Total proportion of correct among observations set aside for validation 0.33

Categories ["0"; "1"; "2"; "3"; "4"; "5"; "6"; "7"; "8"; "9"]

Number of categories 10
number of actualByCat 10
number of predictedByCat 10
number of correctByCat 10
Category 0 number of obs 9 precision 0.55 recall 0.67
Category 1 number of obs 13 precision 0.92 recall 0.85
Category 2 number of obs 8 precision 0.08 recall 0.12
Category 3 number of obs 11 precision 0.12 recall 0.09
Category 4 number of obs 12 precision 0.50 recall 0.42
Category 5 number of obs 6 precision 0.12 recall 0.17
Category 6 number of obs 10 precision 0.20 recall 0.20
Category 7 number of obs 11 precision 0.25 recall 0.27
Category 8 number of obs 7 precision 0.00 recall 0.00
Category 9 number of obs 13 precision 0.20 recall 0.23
```

Several comments should be made at this stage.

Firstly, accuracy of this first model iteration is not good, overall 33 per cent and much lower for some subcategories. This could either be related to an outright error in the programming or to the fact that 3 principal components are chosen and applied with equal weight while actually the first principal component accounts for a much larger proportion of the variation than the following two. Secondly, it should be noted that the eigenvector of the first principal component seems to lie exactly on the diagonal of the 784-dimensional space which requires further interpretation.
Therefore, a second iteration was conducted that omitted the second and third principal component and only project points onto the eigenvector of the first principal component. This could be done easily by modifying the generateNeighbor function that calculates distances. Instead of improving model performance, overall accuracy dropped to 0.19.

As a final attempt, the number of observations was increased from 500 to 5000 which meant 3000 rather than 300 observations in the training data set. This larger set had an accuracy of 0.32 overall. The two elements that should be further examined, the largest eigenvalue being orders of magnitude larger than the second largest and the corresponding eigenvector lying on the diagonal, remained present.

As an improvement it was considered to actually drop the first principal component for the po-



```
Length 784
maximum -0.035621
minimum -0.035737
percentage deviation max/min -0.326844

Length 784
maximum 0.108385
minimum -0.027161
percentage deviation max/min -499.052755

Length 784
maximum 0.153815
minimum -0.098446
percentage deviation max/min -256.243207

Length 784
maximum 0.146182
minimum -0.146138
percentage deviation max/min -200.030202
```

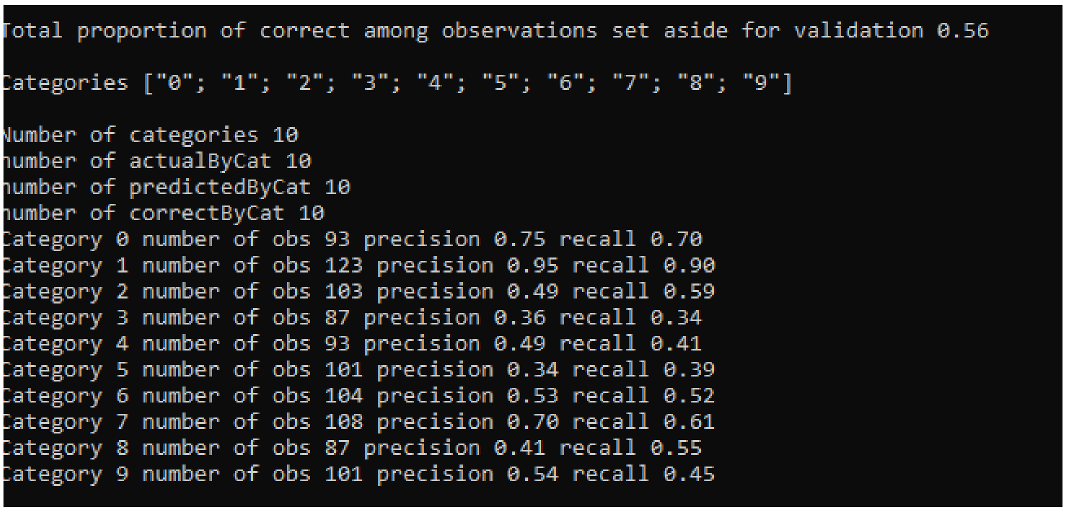
Figur 17: Analysis of principal components

ssibility that it represented some form of anomaly and to perform the analysis on a set of principal components beginning with the second highest eigenvalue. This lead to considerations of how to design the programme so that the number of principal components could be managed in a discretionary way rather than operating with a fixed number of principal components that was arbitrarily set to three.

The programme version `https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/pCA_29032022.fsx` implemented these changes. A new custom record pic5 was created which was similar to pic4 but which did not have the distance along the principal components as separate attributes but rather in a list of float which could then have any given length. The function genEigenvaluesAndVectors was updated so that it took advantage of the fact that MathNet.Numerics provides eigenvalues and eigenvectors as seqs. The eigenvectors would now be seqs that would be called in the appropriate number. The number of principal components was now defined as a parameter in the beginning of the programme. A consideration for a later day was that the number of principal components could be determined endogeneously by the programme as a function of the proportion of the total variation that the user would want to cover.

The numbers were also analyzed further. The first three eigenvectors all seemed to be almost on the diagonal, but actually there were small variations. It had been discussed with the supervisor that points generally tend to be close to each other in high-dimensional space. Also, given that the eigenvectors were 784 elements long, it was difficult to conclude from a visual inspection. Therefore, the numbers shown in Figure (17)

This deeper analysis demonstrated that even though the eigenvectors seemed close to the diagonal and close to each other when looking at the first elements on the screen, they were quite different from each other and from the diagonal when examining their maximal and minimal elements.



```
Total proportion of correct among observations set aside for validation 0.56

Categories ["0"; "1"; "2"; "3"; "4"; "5"; "6"; "7"; "8"; "9"]

Number of categories 10
number of actualByCat 10
number of predictedByCat 10
number of correctByCat 10
Category 0 number of obs 93 precision 0.75 recall 0.70
Category 1 number of obs 123 precision 0.95 recall 0.90
Category 2 number of obs 103 precision 0.49 recall 0.59
Category 3 number of obs 87 precision 0.36 recall 0.34
Category 4 number of obs 93 precision 0.49 recall 0.41
Category 5 number of obs 101 precision 0.34 recall 0.39
Category 6 number of obs 104 precision 0.53 recall 0.52
Category 7 number of obs 108 precision 0.70 recall 0.61
Category 8 number of obs 87 precision 0.41 recall 0.55
Category 9 number of obs 101 precision 0.54 recall 0.45
```

Figur 18: Model performance for number of PC=10 and sample size=5000

It was now attempted to increase the number of principal components to 10 to see how that affected model performance. Applied to a data set of 500 observations, this only improved performance marginally, but combined with an increase in the sample size to 5000 (of which 3000 in the training set), model performance now increased to 0.56 per cent as shown in Figure 18 Increasing the number of principal components from 10 to 20 only increased accuracy marginally to 0.59.
In order to illustrate the difference in eigenvalues, a Scree-diagram was considered. Such a diagram would however be dominated by the first principal component. A table is shown instead, interestingly the second largest eigenvalue is negative and numerically much larger than the following eigenvalues.

```
Eigenvalues of PCs
56.9997448882
-0.0000377184
0.0000008077
0.0000005354
0.0000004592
0.0000003731
0.0000003056
0.0000002791
0.0000002340
0.0000002226
0.0000001962
0.0000001668
0.0000001754
0.0000001422
0.0000001456
0.0000001297
0.0000001205
0.0000001174
0.0000001076
0.0000000965
```

Even though the first eigenvalue is orders of magnitude larger, model performance is markable improved by adding additional principal components. This challenges the previous assumption that MathNet.Numerics already presents eigenvalues and eigenvectors in sorted order and would indicate that they are presented in sorted order according to numerical size. One possible development of the programme could be to pull all 784 records of eigenvalues/eigenvectors and sort them inside the programme.

As a conclusion on the project so far, it can be noted that

- PCA has been implemented in functional programming, there is only one loop which occurs in the sorting algorithm, and there are no mutable variables.
- functional programming made a number of steps easier, for instance the transition from operating with a fixed number of 3 principal components to making the number of principal components a parameter that the user could select
- numbers for eigenvalues and eigenvectors are difficult to interpret
- the model clearly has explanatory power with an overall accuracy of 0.56, but this is still lower than the k-nearest neighbor model that analyzed all data without reduction in dimensionality and easily obtained accuracy scores over 0.9 as shown in Figure 16.

A final programming observation is that it was attempted to restructure the .fsx-file so that a first part defined all custom record types and a second part defined all function. This threw an error message, apparently because the compiler does not recognize custom record types if they are defined too far from the function in which they are to be applied. This in particular applied to the many different custom record types that defined pictures such as pic1,pic3 and pic5. Therefore, custom record types were defined closer to the functions that wold be calling them.

In the meeting with the supervisor, a number of improvements were suggested. Numbers should not be used to separate constants, functions or record types that were somehow developments of each other ("pic1", "pic3", "pic5"), instead the name should represent the actual difference. Therefore, names such as "pic", "standardPic"and "pCPic"were implemented. When triaging pictures into train, valid and test data sets, the attributes should not be strings, instead a discriminated union was implemented where the attribute could only take on the values of Train, Valid or Test. This seemed to resolve the above mentioned issue of not being able to define custom record types far from the functions where they were going to be called.

The previous anomaly with one eigenvalue being orders of magnitude larger than any other eigenvalue was resolved by detecting that the normalization of observations had not been done correctly in that the average had not been subtracted correctly, something that would affect the first principal component but not any other principal component.

13.11 Week 10 - 6 April 2022

A new version of the programme was published to github `https://github.com/TimMondorf/PUK-Image-Recognition-` `blob/main/pCA_05042022.fsx`. The programme does currently not produce correct output.

The following updates have been performed

- Custom record types have been arranged alphabetically
- A reverse topological sort of functions has been implemented. The main function executePCA is now the root of this topological sort. Functions that are called by executePCA are the first layer, functions that are called by these functions are the second layer and other functions are in the third layer. The objective is to make debugging more logical and related to specific functions.
- A number of functions that perform sanity check of the output of other functions such as covariance matrix and eigenvectors have been implemented. They print key summary variables such as max and min to standard output to make analysis simpler and more intuitive.
- Standard deviation of several features is zero so the standardization of features has been updated to simply preserve the original value of the feature (often zero) if the standard deviation is zero.

13.12 Week 11 and 12 - 20 April

The model version relative to this section is available on `https://github.com/TimMondorf/PUK-Image-Recognition-F` `blob/main/pca18042022.fsx`

I decided to actually try to implement a version of the programme that relied more fully on MathNet.Numerics.LinearAlgebra, where matrices were the primary data structure. The reason for this was that in order to obtain the eigenvectors, I would anyhow need to insert my data into a matrix form, so it would simplify the programme to do this consistently. I was still determined to follow the functional paradigm whenever I could, but for this iteration the main theme was processing data in the form of large matrices. From 1991 to 1993, I programmed in APL as an intern in Industrirådet and therefore had the experience of manipulating data sets in the form of matrices.

The programme splits the input into train, valid and test and then builds a matrix of the train part of the data set. The first column of this matrix represents the labels, so they are read into a separate vector and dropped fromt he column. MathNet.Numerics has a function <EnumerateColumns> that extracts all columns of a matrix into a seq, and this function is applied to read each column and standardize it by subtracting the average of the entire column from each element and then dividing the element with the standard deviation unless the standard deviation is zero, in which case only the subtraction is performed (this will apply to the many columns that have only zeroes, such as the feature for the upper left corner of a scanned picture of a hand-written digit which normally is zero). The processed columns are read back into a matrix that is transposed. This matrix is now 784x784, the rows represent the pictures and the columns the features.

Processing entire matrices made the calculation of the covariance matrix simpler. Since the average of each column (feature) in the standardized matrix was now zero, the deviation from the average was simply the element itself. The covariance matrix was therefore calculated applying the matrix of covariance estimators

$$Q = \frac{1}{1-n} \sum (x_i - \hat{x})(x_j - \hat{x}) \tag{1}$$

which for $\hat{x} = 0$ reduced to

$$Q = \frac{1}{1-n} \sum x_i x_j \tag{2}$$

Which could be calculated using MathNet.Numerics.LinearAlgebra built-in functions for transposing the standardized feature matrix, multiplying it with itself and dividing by the number of rows minus one.

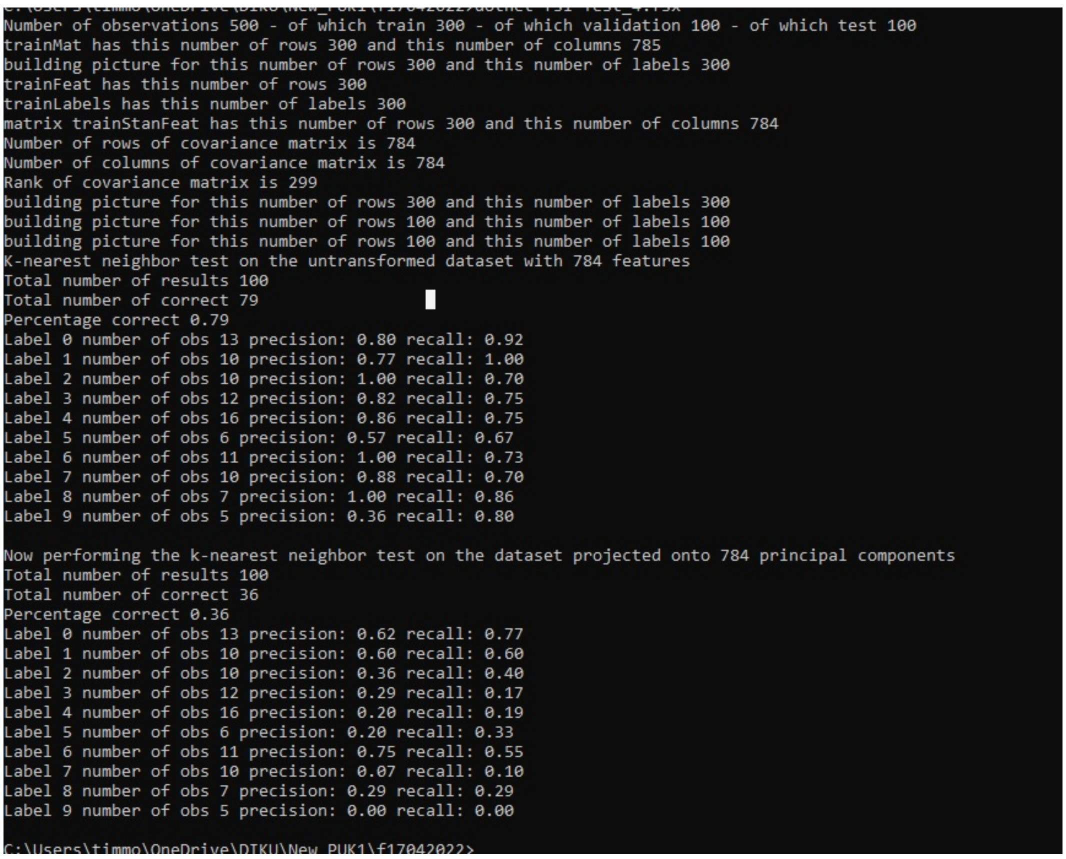
Previous iterations had had a large number of sanity check functions, these were required since the data set was large with many null observations and therefore could not be visually inspected. These sanity check functions were replaced by simply calculating the rank of the covariance matrix to ensure that the programme was processing a non-trivial data set.

Once the covariance matrix had been calculated, the 10 eigenvectors corresponding to the 10 largest eigenvalues were extracted, and the 784 features were collapsed into 10. Only at this stage was a custom record type for picture generated. Neighbors and distances were then calculated in the same way as in previous versions of the programme. Since these functions were generic, they could be applied equally to the 10-dimensional features of the PCA and to the 784-dimensional features of the original data set, which made it possible to compare the original analysis to the PCA.

Unfortunately, the poor performance of the PCA persisted. As a sanity check, the PCA was performed with the parameter for number of Principal Components set to the total number, 784. In this case, the two models should be identical, but as shown in Figure 19, the performance of the PCA-model remained significantly below that of the simple k-nearest neighbor model. It was clear that an implementation error persisted.

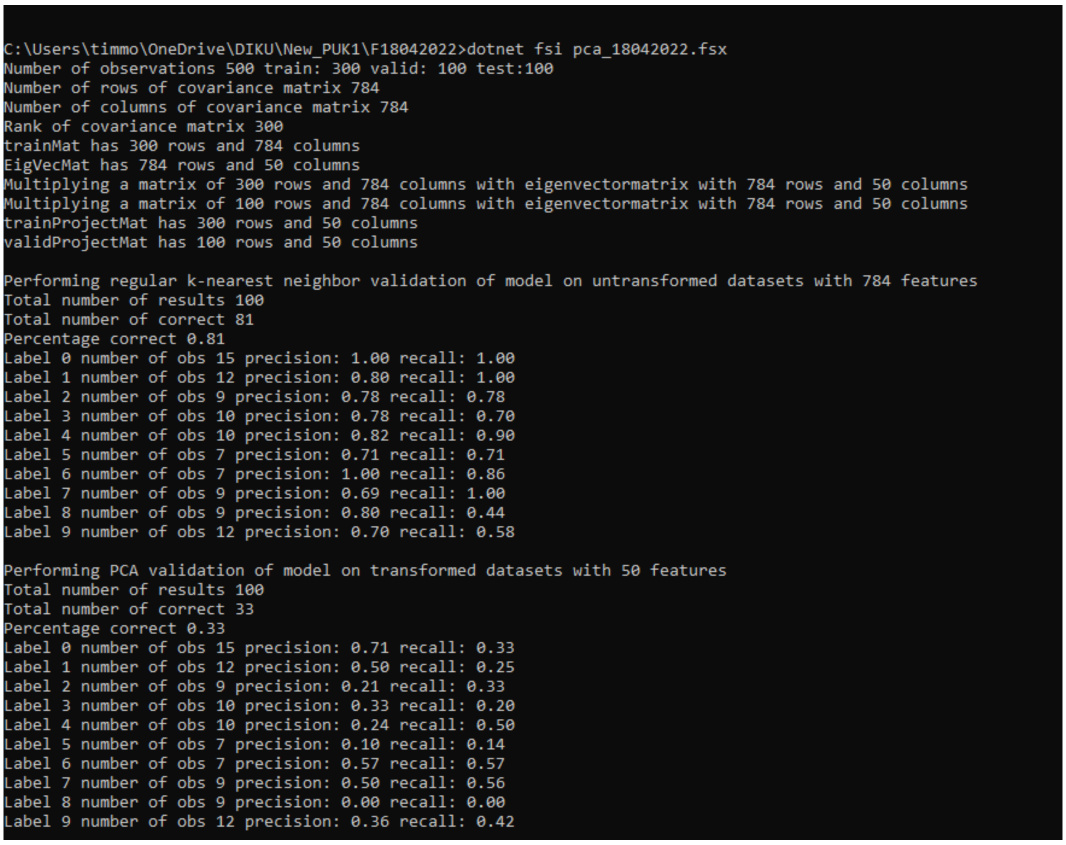
In the subsequent iteration, the method of projecting vectors onto principal components was revisited. Instead of projecting these vector by vector, it was decided to simply multiply the principal component matrix with the feature matrix. Since principal components are vectors in 784-dimensional space, 10 principal components will constitute a matrix of 784x10 which multiplied onto a matrix of pictures with as many rows as there are pictures and 784 columns will constitue a matrix of as many rows as there are pictures and as many columns as there are principal components, in this case 10. The basis for doing this was this source on OpenGenus `https://iq.opengenus.org/algorithm-principal-component-analysis-pca/?msclkid=733ec715c07711eca8de783d37471ccd`. Remarkably, this lead to restoration of good performance in the limit case where the number of principal components was equal to the full number of features, but when the number of principal components was reduced to 10 or 50, model performance deteriorated rapidly. Increasing the number of observations from 500 to 5000 did not address this problem. This can be seen from the following two model simulations, one with number of principal components set to 10 and one with principal components set to 784, shown in Figure 20 and Figure 21

In light of these results, three next steps were decided. Firstly, to re-run the model without standardization. This could be justified since the pixel values are already limited within the 0-255 range for all features. Secondly, to perform a sanity check of whether MathNet.Numerics actually



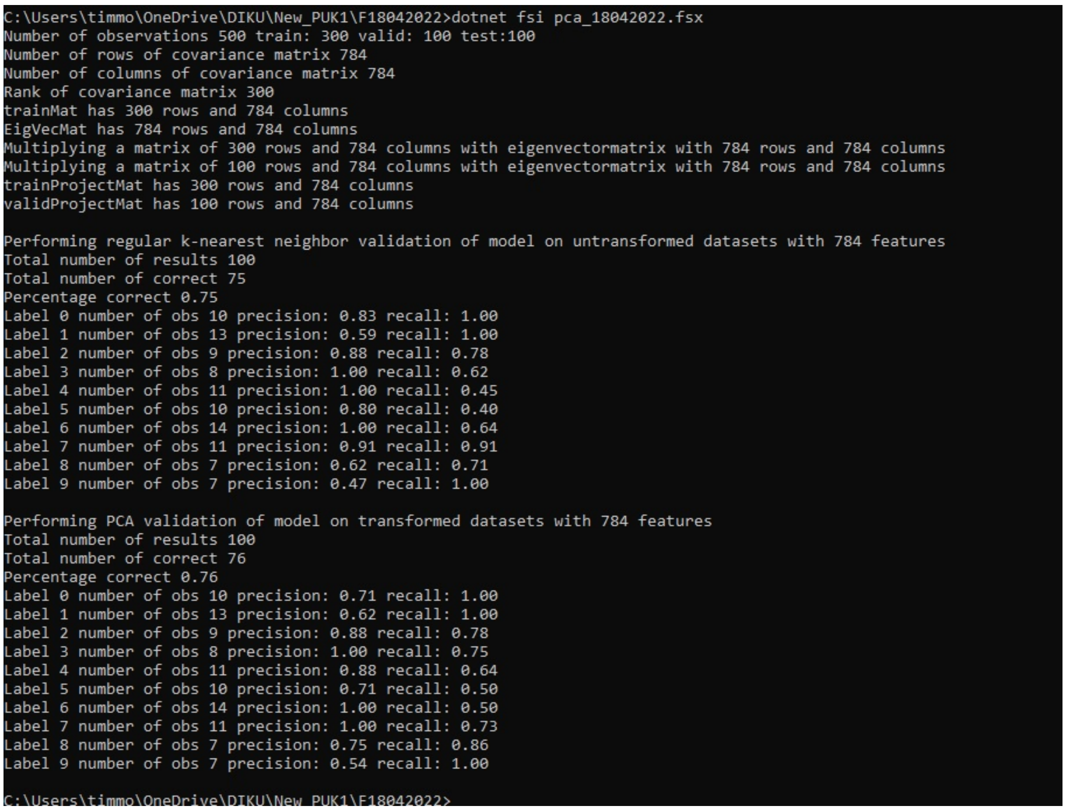
```
C:\Users\timmo\OneDrive\DIKU\New_PUK1\f17042022>dotnet fsi test_4.fsx
Number of observations 500 - of which train 300 - of which validation 100 - of which test 100
trainMat has this number of rows 300 and this number of columns 785
building picture for this number of rows 300 and this number of labels 300
trainFeat has this number of rows 300
trainLabels has this number of labels 300
matrix trainStanFeat has this number of rows 300 and this number of columns 784
Number of rows of covariance matrix is 784
Number of columns of covariance matrix is 784
Rank of covariance matrix is 299
building picture for this number of rows 300 and this number of labels 300
building picture for this number of rows 100 and this number of labels 100
building picture for this number of rows 100 and this number of labels 100
K-nearest neighbor test on the untransformed dataset with 784 features
Total number of results 100
Total number of correct 79
Percentage correct 0.79
Label 0 number of obs 13 precision: 0.80 recall: 0.92
Label 1 number of obs 10 precision: 0.77 recall: 1.00
Label 2 number of obs 10 precision: 1.00 recall: 0.70
Label 3 number of obs 12 precision: 0.82 recall: 0.75
Label 4 number of obs 16 precision: 0.86 recall: 0.75
Label 5 number of obs 6 precision: 0.57 recall: 0.67
Label 6 number of obs 11 precision: 1.00 recall: 0.73
Label 7 number of obs 10 precision: 0.88 recall: 0.70
Label 8 number of obs 7 precision: 1.00 recall: 0.86
Label 9 number of obs 5 precision: 0.36 recall: 0.80

Now performing the k-nearest neighbor test on the dataset projected onto 784 principal components
Total number of results 100
Total number of correct 36
Percentage correct 0.36
Label 0 number of obs 13 precision: 0.62 recall: 0.77
Label 1 number of obs 10 precision: 0.60 recall: 0.60
Label 2 number of obs 10 precision: 0.36 recall: 0.40
Label 3 number of obs 12 precision: 0.29 recall: 0.17
Label 4 number of obs 16 precision: 0.20 recall: 0.19
Label 5 number of obs 6 precision: 0.20 recall: 0.33
Label 6 number of obs 11 precision: 0.75 recall: 0.55
Label 7 number of obs 10 precision: 0.07 recall: 0.10
Label 8 number of obs 7 precision: 0.29 recall: 0.29
Label 9 number of obs 5 precision: 0.00 recall: 0.00

C:\Users\timmo\OneDrive\DIKU\New_PUK1\f17042022>
```

Figur 19: Results of PCA with number of principal components equal to number of features



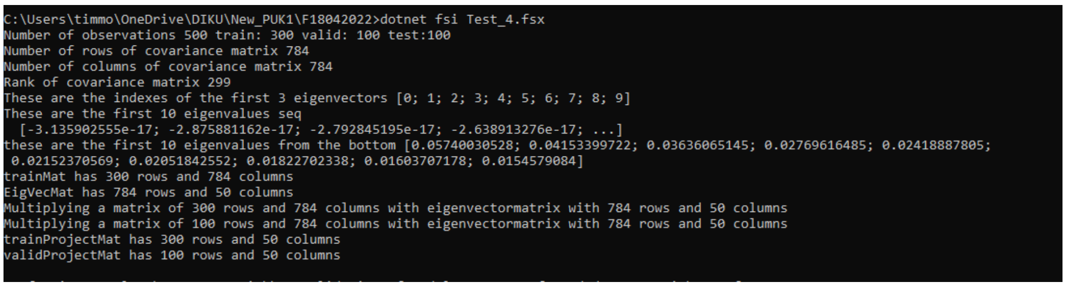
```
C:\Users\timmo\OneDrive\DIKU\New_PUK1\F18042022>dotnet fsi pca_18042022.fsx
Number of observations 500 train: 300 valid: 100 test:100
Number of rows of covariance matrix 784
Number of columns of covariance matrix 784
Rank of covariance matrix 300
trainMat has 300 rows and 784 columns
EigVecMat has 784 rows and 50 columns
Multiplying a matrix of 300 rows and 784 columns with eigenvectormatrix with 784 rows and 50 columns
Multiplying a matrix of 100 rows and 784 columns with eigenvectormatrix with 784 rows and 50 columns
trainProjectMat has 300 rows and 50 columns
validProjectMat has 100 rows and 50 columns

Performing regular k-nearest neighbor validation of model on untransformed datasets with 784 features
Total number of results 100
Total number of correct 81
Percentage correct 0.81
Label 0 number of obs 15 precision: 1.00 recall: 1.00
Label 1 number of obs 12 precision: 0.80 recall: 1.00
Label 2 number of obs 9 precision: 0.78 recall: 0.78
Label 3 number of obs 10 precision: 0.78 recall: 0.70
Label 4 number of obs 10 precision: 0.82 recall: 0.90
Label 5 number of obs 7 precision: 0.71 recall: 0.71
Label 6 number of obs 7 precision: 1.00 recall: 0.86
Label 7 number of obs 9 precision: 0.69 recall: 1.00
Label 8 number of obs 9 precision: 0.80 recall: 0.44
Label 9 number of obs 12 precision: 0.70 recall: 0.58

Performing PCA validation of model on transformed datasets with 50 features
Total number of results 100
Total number of correct 33
Percentage correct 0.33
Label 0 number of obs 15 precision: 0.71 recall: 0.33
Label 1 number of obs 12 precision: 0.50 recall: 0.25
Label 2 number of obs 9 precision: 0.21 recall: 0.33
Label 3 number of obs 10 precision: 0.33 recall: 0.20
Label 4 number of obs 10 precision: 0.24 recall: 0.50
Label 5 number of obs 7 precision: 0.10 recall: 0.14
Label 6 number of obs 7 precision: 0.57 recall: 0.57
Label 7 number of obs 9 precision: 0.50 recall: 0.56
Label 8 number of obs 9 precision: 0.00 recall: 0.00
Label 9 number of obs 12 precision: 0.36 recall: 0.42
```

Figur 20: Simulation with 50 principal components



```
C:\Users\timmo\OneDrive\DIKU\New_PUK1\F18042022>dotnet fsi pca_18042022.fsx
Number of observations 500 train: 300 valid: 100 test:100
Number of rows of covariance matrix 784
Number of columns of covariance matrix 784
Rank of covariance matrix 300
trainMat has 300 rows and 784 columns
EigVecMat has 784 rows and 784 columns
Multiplying a matrix of 300 rows and 784 columns with eigenvectormatrix with 784 rows and 784 columns
Multiplying a matrix of 100 rows and 784 columns with eigenvectormatrix with 784 rows and 784 columns
trainProjectMat has 300 rows and 784 columns
validProjectMat has 100 rows and 784 columns

Performing regular k-nearest neighbor validation of model on untransformed datasets with 784 features
Total number of results 100
Total number of correct 75
Percentage correct 0.75
Label 0 number of obs 10 precision: 0.83 recall: 1.00
Label 1 number of obs 13 precision: 0.59 recall: 1.00
Label 2 number of obs 9 precision: 0.88 recall: 0.78
Label 3 number of obs 8 precision: 1.00 recall: 0.62
Label 4 number of obs 11 precision: 1.00 recall: 0.45
Label 5 number of obs 10 precision: 0.80 recall: 0.40
Label 6 number of obs 14 precision: 1.00 recall: 0.64
Label 7 number of obs 11 precision: 0.91 recall: 0.91
Label 8 number of obs 7 precision: 0.62 recall: 0.71
Label 9 number of obs 7 precision: 0.47 recall: 1.00

Performing PCA validation of model on transformed datasets with 784 features
Total number of results 100
Total number of correct 76
Percentage correct 0.76
Label 0 number of obs 10 precision: 0.71 recall: 1.00
Label 1 number of obs 13 precision: 0.62 recall: 1.00
Label 2 number of obs 9 precision: 0.88 recall: 0.78
Label 3 number of obs 8 precision: 1.00 recall: 0.75
Label 4 number of obs 11 precision: 0.88 recall: 0.64
Label 5 number of obs 10 precision: 0.71 recall: 0.50
Label 6 number of obs 14 precision: 1.00 recall: 0.50
Label 7 number of obs 11 precision: 1.00 recall: 0.73
Label 8 number of obs 7 precision: 0.75 recall: 0.86
Label 9 number of obs 7 precision: 0.54 recall: 1.00

C:\Users\timmo\OneDrive\DIKU\New_PUK1\F18042022>
```

Figur 21: Simulation with 784 principal components



```
C:\Users\timmo\OneDrive\DIKU\New_PUK1\F18042022>dotnet fsi Test_4.fsx
Number of observations 500 train: 300 valid: 100 test:100
Number of rows of covariance matrix 784
Number of columns of covariance matrix 784
Rank of covariance matrix 299
These are the indexes of the first 3 eigenvectors [0; 1; 2; 3; 4; 5; 6; 7; 8; 9]
These are the first 10 eigenvalues seq
  [-3.135902555e-17; -2.875881162e-17; -2.792845195e-17; -2.638913276e-17; ...]
these are the first 10 eigenvalues from the bottom [0.05740030528; 0.04153399722; 0.03636065145; 0.02769616485; 0.02418887805;
 0.02152370569; 0.02051842552; 0.01822702338; 0.01603707178; 0.0154579084]
trainMat has 300 rows and 784 columns
EigVecMat has 784 rows and 50 columns
Multiplying a matrix of 300 rows and 784 columns with eigenvectormatrix with 784 rows and 50 columns
Multiplying a matrix of 100 rows and 784 columns with eigenvectormatrix with 784 rows and 50 columns
trainProjectMat has 300 rows and 50 columns
validProjectMat has 100 rows and 50 columns
```

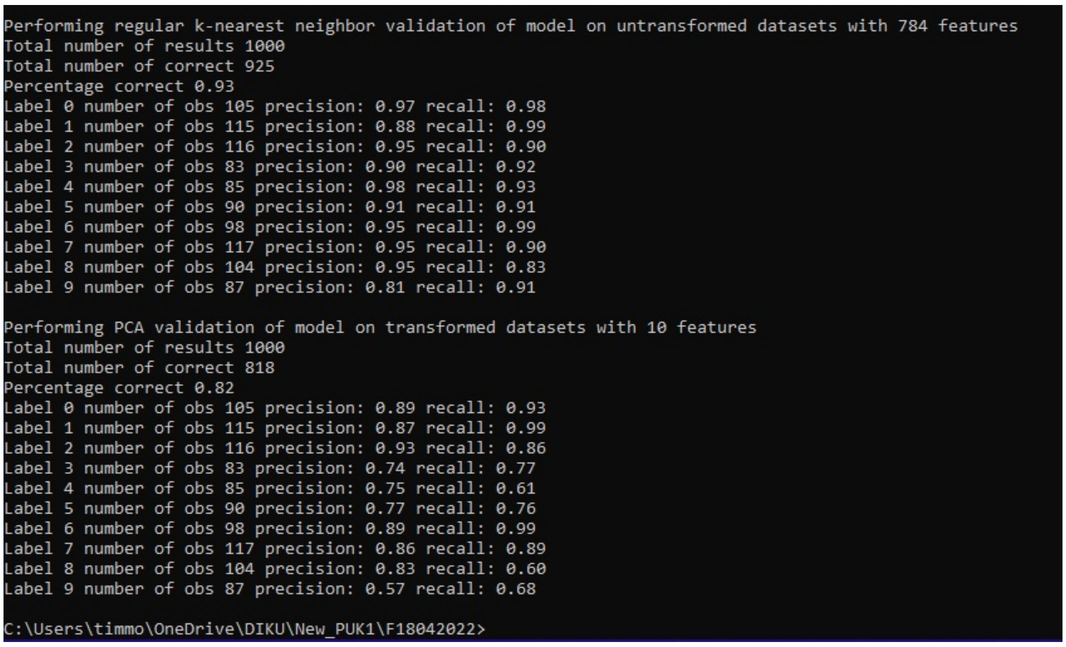
Figur 22: Eigenvector indexes and eigenvalues from Evd-decomposition of covariancematrix

does provide the eigenvalues in appropriately sorted order. If model performance is poor when only a subset of eigenvectors is applied, this could be because the module does not correctly select the eigenvectors with the highest eigenvalues the way the PCA-method prescribes. Thirdly, the analysis could be illustrated by graphically comparing actual pictures and pictures projected onto principal components.

Firstly, the standardization was removed from the algorithm and it was concluded that this did not improve performance, to the contrary.

Secondly, in order to verify that eigenvectors were indeed returned from the matrix object in sorted order, the function EnumerateColumnsIndexed was applied. This function returns columns in a tuple with their index. At the same time, the sequence of eigenvalues was analyzed, studying the first 10 and the last 10 of the 784 eigenvalues. The analysis revealed that while MathNet.Numerics does sort the eigenvectors according to the eigenvalues, it does so ascendingly and not descendingly. This appears in Figure 22

With the model thus improved, the results came closer to the expected. For 5000 observations, the regular k-nearest neighbor model had an accuracy of 93 per cent while the PCA, reducing dimensionality to 10 principal components, had an accuracy of 82 per cent, as shown in Figure 23



```
Performing regular k-nearest neighbor validation of model on untransformed datasets with 784 features
Total number of results 1000
Total number of correct 925
Percentage correct 0.93
Label 0 number of obs 105 precision: 0.97 recall: 0.98
Label 1 number of obs 115 precision: 0.88 recall: 0.99
Label 2 number of obs 116 precision: 0.95 recall: 0.90
Label 3 number of obs 83 precision: 0.90 recall: 0.92
Label 4 number of obs 85 precision: 0.98 recall: 0.93
Label 5 number of obs 90 precision: 0.91 recall: 0.91
Label 6 number of obs 98 precision: 0.95 recall: 0.99
Label 7 number of obs 117 precision: 0.95 recall: 0.90
Label 8 number of obs 104 precision: 0.95 recall: 0.83
Label 9 number of obs 87 precision: 0.81 recall: 0.91

Performing PCA validation of model on transformed datasets with 10 features
Total number of results 1000
Total number of correct 818
Percentage correct 0.82
Label 0 number of obs 105 precision: 0.89 recall: 0.93
Label 1 number of obs 115 precision: 0.87 recall: 0.99
Label 2 number of obs 116 precision: 0.93 recall: 0.86
Label 3 number of obs 83 precision: 0.74 recall: 0.77
Label 4 number of obs 85 precision: 0.75 recall: 0.61
Label 5 number of obs 90 precision: 0.77 recall: 0.76
Label 6 number of obs 98 precision: 0.89 recall: 0.99
Label 7 number of obs 117 precision: 0.86 recall: 0.89
Label 8 number of obs 104 precision: 0.83 recall: 0.60
Label 9 number of obs 87 precision: 0.57 recall: 0.68

C:\Users\timmo\OneDrive\DIKU\New_PUK1\F18042022>
```

Figur 23: Results of final simulation with both KNN and PCA performing above 80 per cent accuracy

13.13 Week 13 - 25 April

It was concluded that with simple k-nearest neighbor and PCA, the logical engine for the app was now in place and the project could move on to the next phase which would be graphical comparisons between untransformed images and images transformed by PCA. The supervisor suggested to analyze images in .tiff-format since this is a richer format than .jpg or .png.

The Fsharp-library System.Drawing is being transitioned from a system library to more of an elective library to be invoked with the nuget-command and therefore System.Drawing did not work as expected in dotnet 6. It was first attempted to run programme versions in Mono but after internet research and a link suggested by the supervisor, it was decided to manually copy the System.Drawing.Common.dll-file from

```
///FROM "C:\Program Files\dotnet\sdk\6.0.101\System.Drawing.Common.dll"
///TO "C:\Users\timmo\.nuget\packages\system.drawing.common\6.0.0\runtimes\win\lib\net6.
```

which obviously is a very manual solution.

The strategy for comparing untransformed and transformed images was considered. With a full picture at (28x28=784) dimensions, dimensionality could be reduced as shown in Figure 24

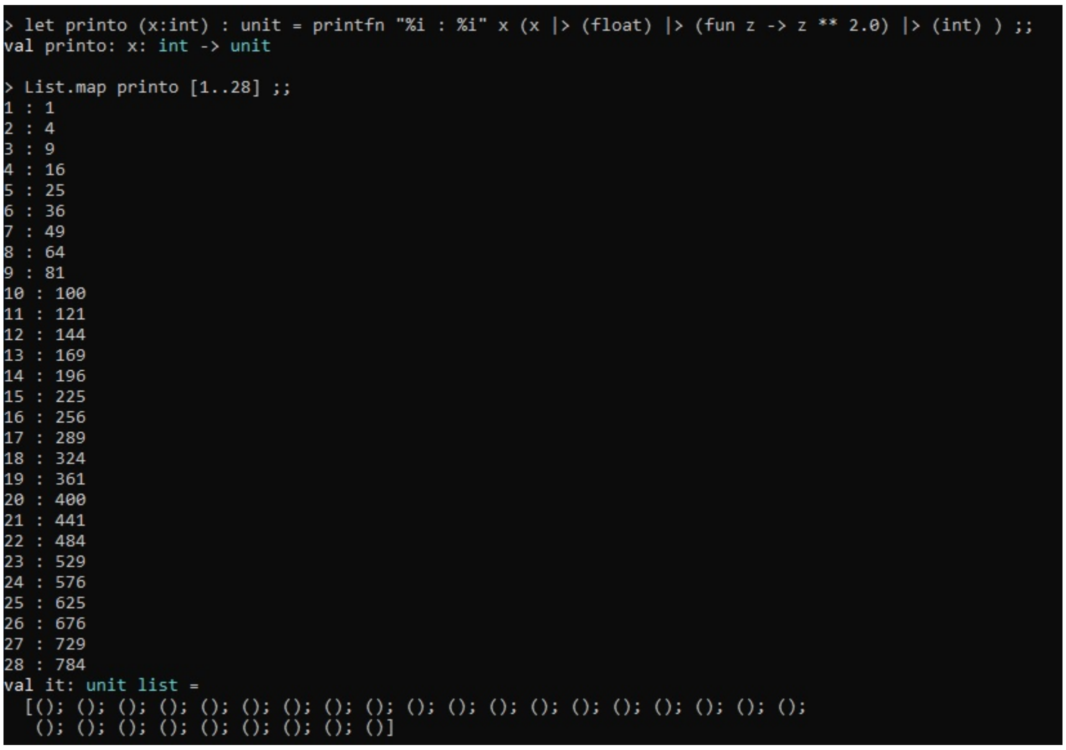
Based on the above, it was decided to analyze pictures for a number of principal components in the range of (5, 10, 15, 20, 25, 28), with 28 being identical to the untransformed picture. 3 pictures in the training data set would be selected at random and compared.

The programme then had to be rewritten to perform the relevant iterations. The functional structure rendered this rewrite relative easy for two reasons. Firstly, he function genEigVecMat already took numPC, the number of principal components, as an argument. The generated matrix of eigenvectors then was multiplied with the matrix of standardized observations and there were no further explicit references to the number of principal components. Secondly, the untransformed and transformed training data sets already existed as two separate matrices that were transformed to two separate picture lists.

The programme was now expanded with a function generateAndSaveBmps, that read a picture file, confirmed that the number of features were a quadratic number, created a corresponding square bitmap, populated the pixels in this bitmap and saved a .tiff-file to disc. The name of this file reflected the number of dimensions and the label. If the first picture in the untransformed data set had the label 8, it would get the filename

```
"pic_1_PC784_label_is_8.tiff"
```

The user could then visually compare the image to the label. This nomenclature was of course quite cumbersome and manual, so a note was made to consider more functional approaches such as custom



```
> let printo (x:int) : unit = printfn "%i : %i" x (x |> (float) |> (fun z -> z ** 2.0) |> (int) ) ;;
val printo: x: int -> unit

> List.map printo [1..28] ;;
1 : 1
2 : 4
3 : 9
4 : 16
5 : 25
6 : 36
7 : 49
8 : 64
9 : 81
10 : 100
11 : 121
12 : 144
13 : 169
14 : 196
15 : 225
16 : 256
17 : 289
18 : 324
19 : 361
20 : 400
21 : 441
22 : 484
23 : 529
24 : 576
25 : 625
26 : 676
27 : 729
28 : 784
val it: unit list =
  [(); (); (); (); (); (); (); (); (); (); (); (); (); (); (); (); (); (); ();
   (); (); (); (); (); (); (); (); ()]
```

Figur 24: Possible reductions of dimensionality

record types. For the time being, focus was on being able to compare pictures visually in a LaTex document.

generateAndSaveBmps was called by a function convertDataListToTiffs that operated on entire picture lists.

It became necessary to make the programme more general. Before, the eigenvectormatrix had been generated and immediately reduced to the predefined number of principal components. Now, the programme had to make the entire eigenvectormatrix available as a global object so that iterations could be performed supporting different dimensionality. The solution again was to make the programme even more modular and granular.

The tiff-files were generated functionally. First, a smaller number (here 3) of images were selected at random from the rows of the matrix of training data and defined as a matrix (in the example a 3x784 matrix). Their labels were populated into a separate list.

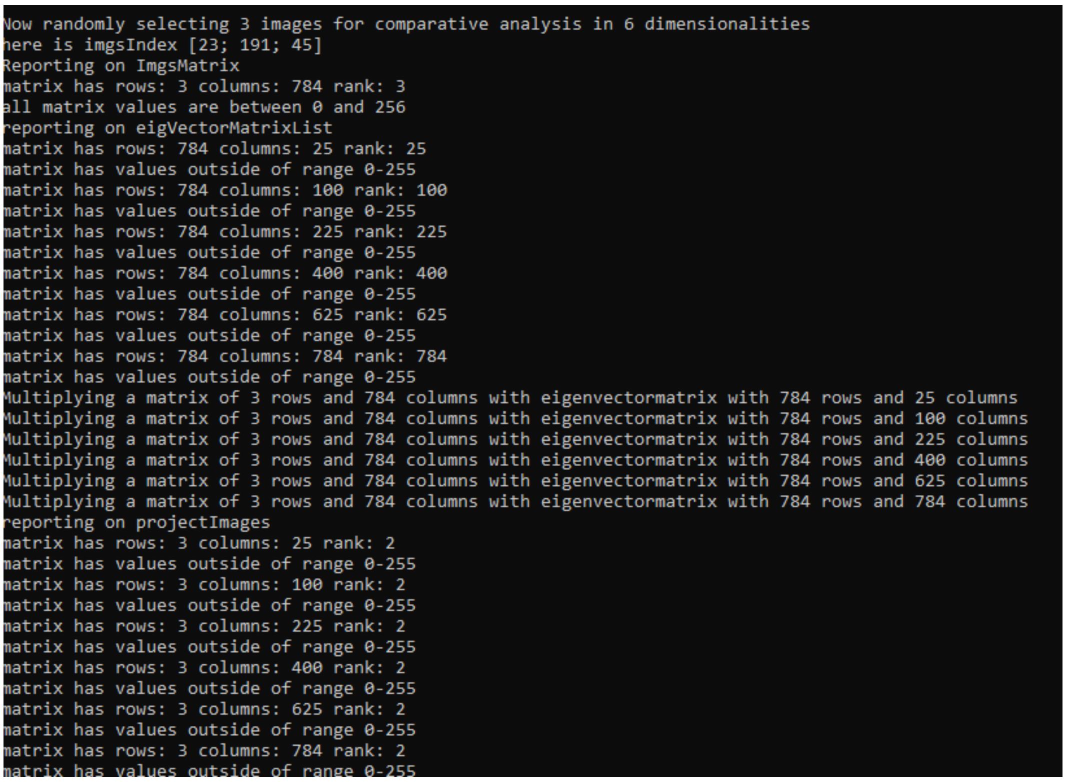
In order to simulate the analysis in (5,10,15,20,25,28) dimensions, a list of eigenvectors were generated using List.init. The first element of this list was the first (5*5=25) eigenvectors with the largest eigenvalues, the second element was the first (10*10=100) eigenvectors up to the last element with all (28*28=784) eigenvectors. A subsequent list was generated with all elements in transposed matrix form.

A new list was generated that held the matrix of selected elements, repeated 6 times, to match the above 6 different matrices.

The programme had already defined a projectMat-function that projects full-dimension pictures onto reduced-dimension eigenvectors, and this programme was now simply applied using the List.map function. The end result was a list of 6 matrices, the first being the 3 images projected onto 25 dimensions, the second being the 3 images projected onto 100 dimensions, and the final being the 3 images projected onto the full 784 dimensions.

Using the List.map-function, a function was now applied that read each row of the matrix, interpreted the numeric values as pixel data, populated the pixel values into a bitmap and saved this bitmap in tiff-format to the folder, with the label and dimension information in the name.

A sanity check function was defined that read all the matrices, reported their dimensions and rank and verified that all values were in the range between 0 and 255, meaning that the values could be interpreted as pixel values. The forall-method related to matrices in MathNet.Numerics meant that this could be done easily.



```
Now randomly selecting 3 images for comparative analysis in 6 dimensionalities
here is imgsIndex [23; 191; 45]
Reporting on ImgsMatrix
matrix has rows: 3 columns: 784 rank: 3
all matrix values are between 0 and 256
reporting on eigVectorMatrixList
matrix has rows: 784 columns: 25 rank: 25
matrix has values outside of range 0-255
matrix has rows: 784 columns: 100 rank: 100
matrix has values outside of range 0-255
matrix has rows: 784 columns: 225 rank: 225
matrix has values outside of range 0-255
matrix has rows: 784 columns: 400 rank: 400
matrix has values outside of range 0-255
matrix has rows: 784 columns: 625 rank: 625
matrix has values outside of range 0-255
matrix has rows: 784 columns: 784 rank: 784
matrix has values outside of range 0-255
Multiplying a matrix of 3 rows and 784 columns with eigenvectormatrix with 784 rows and 25 columns
Multiplying a matrix of 3 rows and 784 columns with eigenvectormatrix with 784 rows and 100 columns
Multiplying a matrix of 3 rows and 784 columns with eigenvectormatrix with 784 rows and 225 columns
Multiplying a matrix of 3 rows and 784 columns with eigenvectormatrix with 784 rows and 400 columns
Multiplying a matrix of 3 rows and 784 columns with eigenvectormatrix with 784 rows and 625 columns
Multiplying a matrix of 3 rows and 784 columns with eigenvectormatrix with 784 rows and 784 columns
reporting on projectImages
matrix has rows: 3 columns: 25 rank: 2
matrix has values outside of range 0-255
matrix has rows: 3 columns: 100 rank: 2
matrix has values outside of range 0-255
matrix has rows: 3 columns: 225 rank: 2
matrix has values outside of range 0-255
matrix has rows: 3 columns: 400 rank: 2
matrix has values outside of range 0-255
matrix has rows: 3 columns: 625 rank: 2
matrix has values outside of range 0-255
matrix has rows: 3 columns: 784 rank: 2
matrix has values outside of range 0-255
```

Figur 25: Results of sanity check

Performing this sanity check indicated that while the 3 selected images were of the appropriate values, all the calculated eigenvector matrices and the matrices of projected images had inappropriate values. The results of the sanity check are shown in Figure 25

In the meeting with the supervisor, the picture in Figure 26 was presented, it was supposed to be a picture of the digit four that had been compressed then decompressed.

The supervisor suggested to normalize the training data set by only subtracting the column average and not dividing by the standard deviation. The supervisor also commented that even though validation and test observations should also be standardized, they should be so subtracting the relevant column average from the training data set rather than a separate column average from the validation/test data set. This is because validation and test observations should be seen as separate and external to the model.

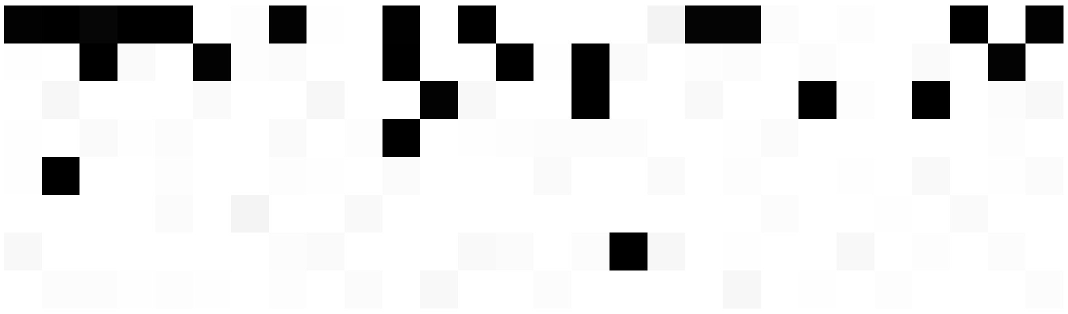
The code was then reviewed, including the use of the MathNet.Numerics matrix objects. It was suggested to have higher consistency so that functions that took matrices as arguments should also return matrices. The naming of the matrices should be made more consistent for instance through composite names reflecting the categories of rows and columns. Special care should be taken when documenting the transposition of matrices.

With the supervisor, the linear algebra behind compression and decompression was reviewed. When a 784-pixel picture was compressed to 50 principal components, the picture would still be in 784 pixels since it would be multiplied with an appropriate matrix with 1 in the first 50 cells in the diagonal and 0 in all other cells.

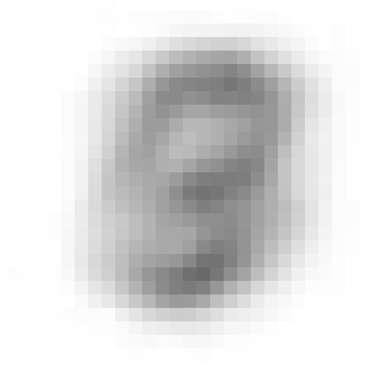
13.14 Week 14 - 25 April to 3 May

In the subsequent iteration of the programme, first the new form of standardization was implemented, something that could be done easily using the ColumnSum-method and Map-method defined on matrix objects in MathNet.Numerics. This made it possible to create a vector of column averages. This vector was now defined as a global variable since both train, valid and test data sets would be standardized using the same averages. It was verified that this new form of standardization was not detrimental to model performance and in fact, this performance improved.

Literature search was performed for references on how image compression and decompression could be visualized. A blog by Deewakar Chakraborty was selected `https://www.section.io/engineering-education/`



Figur 26: First attempt at compression and decompression of an image from the MNIST-data set that supposedly has the label 4


Figur 27: The average digit

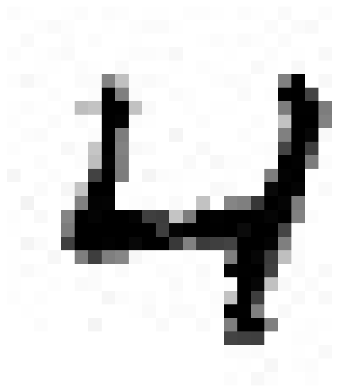

Figur 28: Original version

image-compression-using-pca/. The examples in the blog are based on the Python numpy-library and it was therefore possible to follow the logic of the blog while building own code in Fsharp. Chakraborty works on a Kaggle data set of images.

The first step for Chakraborty is to interpreted the column averages as "the average face (digit)". The average values of each pixel are mapped into a 784-pixel format which is less meaningful for digits than for faces but shown in Figure 27 for reference.

Chakraborty also shows the visual effect of standardization, so Figure 28 and 30 allow to compare the 17th picture in the training set, selected at random, in both its original and standardized form. The label is 4 and this is clearly visible in both versions of the picture. Standardization has been performed as described above.

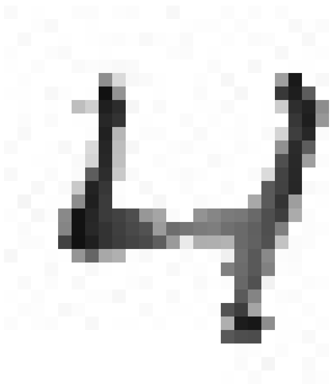
Principal Component analysis is often illustrated by reducing dimensionality of pictures and then projecting the reduced pictures back to full dimensionality, comparing the original picture to the result of the compression and subsequent decompression.

In model terms, compression/decompression is done in the following way, assuming for illustration that it is decided to reduce dimensionality to 10 principal components: From a training set, the covariance matrix and its eigenvectors are calculated. The 10 eigenvectors with the 10 largest eigenvalues are selected. They now form a (784x10) matrix, where each eigenvector is a column. The picture to be compressed forms a (1x784)-matrix. The picture is projected onto a lower-dimensional universe by multiplying the picture with the eigenvector matrix. The result is a (1x10)-matrix representing the picture in compressed form.

$$\begin{bmatrix} pixel_1 & ... & pixel_{784} \end{bmatrix} \begin{bmatrix} ev_{(1,1)} & ... & ev_{(10,1)} \\ ... & & \\ ev_{(1,748)} & ... & ev_{(10,748)} \end{bmatrix} = \begin{bmatrix} compressed_1 & ... & compressed_{10} \end{bmatrix}$$

This compressed form of the picture is now used for k-nearest neighbor or other forms of prediction. The reduced dimensionality makes computation easier and may also require less storage space.

In order to decompress the picture into 784-dimensional space so that it can be shown visually, the



Figur 29: Standardized version of the same picture

Figur 30: Original and standardized version of picture 17 with label 4

compressed picture is now multiplied to the right side of the eigenvector matrix. Since the eigenvector matrix has 10 columns and 784 rows, the result will be a (784x1)-matrix that can be transposed and interpreted as a (1x784) picture.

$$\begin{bmatrix} ev_{(1,1)} & \dots & ev_{(10,1)} \\ \dots & & \\ ev_{(1,748)} & \dots & ev_{(10,748)} \end{bmatrix} \begin{bmatrix} compressed_1 \\ \dots \\ compressed_{10} \end{bmatrix} = \begin{bmatrix} recompressed_1 \\ \dots \\ recompressed_{748} \end{bmatrix}$$

Any information not stored in the first 10 principal components of the compressed picture has been lost. Literature is full of examples of how compression/decompression manages to maintain the most important features. This applies in particular to pictures in color in (64x64). The convergence is less clear for the black-and-white pictures in the MNIST-data set.

In programming terms, the compression/decompression was conducted in the following way: A separate function named Chakraborty was defined, reading the same input parameters. The function calculates the eigenvector matrix and maintains it as a global variable since it will be read several times. The function randomly selects a picture from the validation training set and defines it as a (1x748) matrix. The function projectMat now reads the entire eigenvector matrix, the picture, the selected number of principal components and then compresses the picture, then returns it as a (1xnumPC)-matrix where numPC is the number of principal components. A new function called projectBack reads the compressed picture, the eigenvector matrix and the selected number of principal components and performs the inverse function. The main function now takes as an input argument a vector of dimensions for which the compression/decompression are to be performed. For this experiment, the list of (5,10,15,20,25,28) was selected, with 28 being the full dimensionality. A List.init function then calls the project and projectBack function taking each time and element from the list of dimensionalities as argument and saves the compressed/decompressed picture for that level of dimensionality. The pictures are saved in .tiff-format, manually saved in .jpg format and imported into Latex where they are shown below. The chosen programming approach was purely functional and well suited for this experimental purpose but probably did not display the benefits of PCA.

Figure 31 shows the picture from the 99 row of the validation training set, selected at random, with the label 4, having been compressed to 5 dimensions and then recompressed. The 4 is barely discernible.

Figure 32 shows the same picture compressed/decompressed to 10 dimensions.

Figure 33 shows the same picture compressed/decompressed to 15 dimensions.

Figure 34 shows the same picture compressed/decompressed to 15 dimensions.

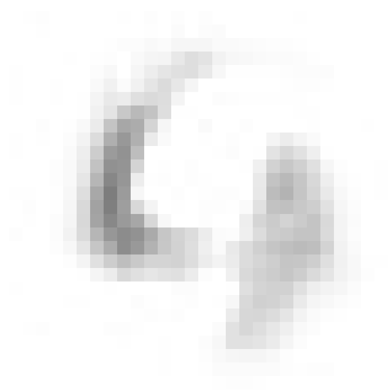
Figure 35 shows the same picture compressed/decompressed to 15 dimensions.

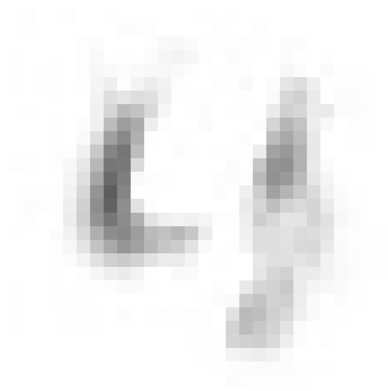
Figure 36 shows the same picture compressed/decompressed to 15 dimensions. This would be the recompression of the original picture to the full dimensionality.

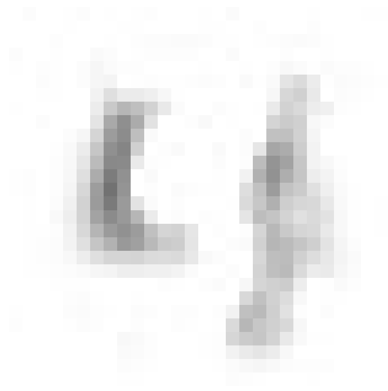
It is clear that the examples shown here are less salient that what is portrayed in literature.

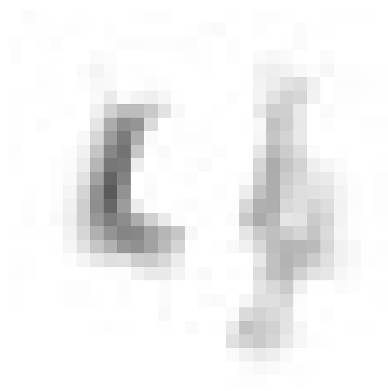
13.15 Week 15 - 3 May to 9 May

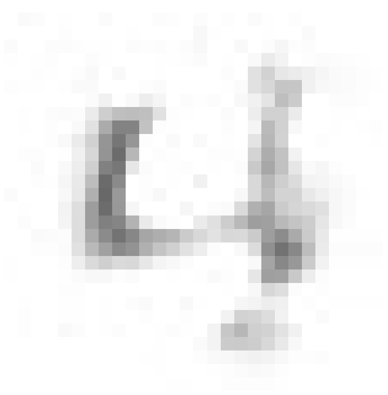
In the meeting with the supervisor the images were discussed. The following actions were suggested


Figur 31: The digit 4 - 5 dimensions compressed/decompressed

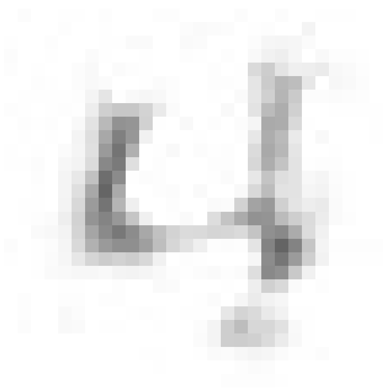

Figur 32: The digit 4 - 10 dimensions compressed/decompressed


Figur 33: The digit 4 - 15 dimensions compressed/decompressed


Figur 34: The digit 4 - 20 dimensions compressed/decompressed



Figur 35: The digit 4 - 25 dimensions compressed/decompressed



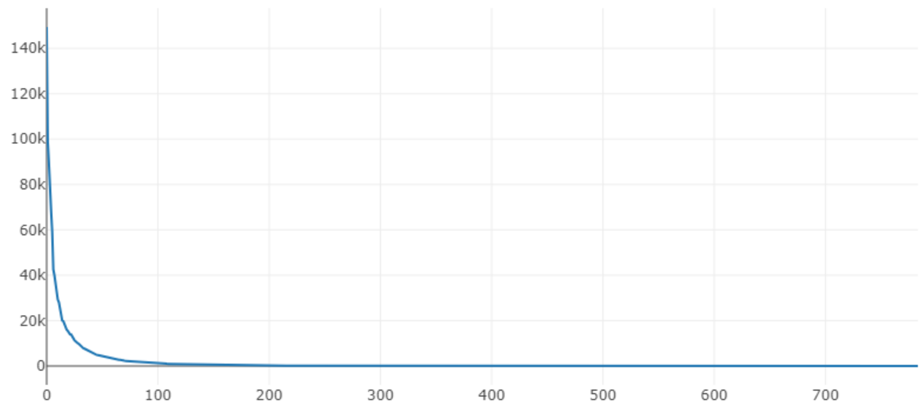
Figur 36: The digit 4 - 28 dimensions compressed/decompressed

- a sanity check of the code
- The visual representation of the standardized picture should be different. In the example shown above, negative values are simply transformed to the value zero which shows as blank. This risked only showing half the variation. Instead, it should be borne in mind that different pixels have different averages. The picture could be restandardized differently as shown below.
- showing the spectrum of eigenvalues. The idea behind PCA is that distribution of eigenvalues is sharply skewed to the right so that a small number of eigenvectors will comprise a large proportion of the total variation. If this is not the case for the data set in question, PCA will not be a good approach.
- compare the image uncompressed using 784 dimensions to the original pictu da re.
- reducing dimensionality to 2, sorting images according to their label and showing the two-dimensional representations. If reduction to 2 principal dimensions is a good model then the colored dots would be grouped in clouds that could easily be separated. If the clouds for various digits overlap then this reduction is too simple.

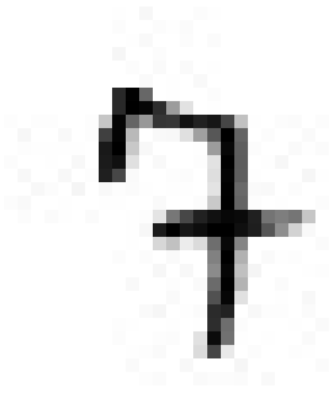
Processing of color images as opposed to the black-white images of the MNIST-data set was discussed. The alpha-channel should be dropped and a (3*4096=12288) vector should be created for computing purposes then retransformed back to 3x4096-format.

First, the sanity check of the code was performed.

It was discovered that the programme had originally considered the 784-dimensions as a 28x28-grid, and therefore the dimensions had been enumerated (5,10,15,25,20,25,28) with 28 representing the full 784 dimensions. Unfortunately, the programme did not elevate these numbers to the power of two which meant that the number 28 meant 28 dimensions not 784-dimensions. This error was corrected and the pictures came closer to what would be expected.



Figur 37: Distribution of eigenvalues



Figur 38: Picture 61 label 7 original version

First, the distribution of the eigenvalues is charted using XPlot.Plotly shown in Figure 37. It can be seen that eigenvalues do drop quite significantly and that PCA therefore should be a relevant model.

Then in Figure 38the picture and in Figure 39its standardized version are considered. It is the 61st picture in the validation set that has the label 7.

Figure 40 shows the compressed-then-uncompressed picture with $5^2 = 25$ dimensions

Figure 41 shows the compressed-then-uncompressed picture with $10^2 = 100$ dimensions

Figure 42 shows the compressed-then-uncompressed picture with $15^2 = 225$ dimensions

Figure 43 shows the compressed-then-uncompressed picture with $20^2 = 225$ dimensions

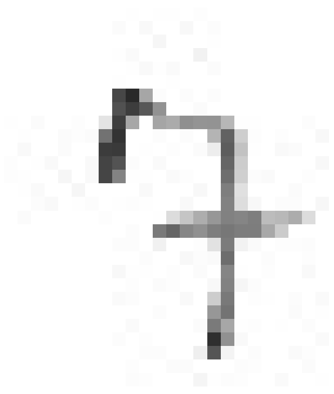
Figure 44 shows the compressed-then-uncompressed picture with $20^2 = 225$ dimensions

And finally Figure 45 shows the compressed-then-uncompressed picture with $28^2 = 784$ dimensions

From a visual inspection, it would appear that the compressed-then-uncompressed picture is not exactly similar to the original version but that there now is a clear improvement as the dimensionality increases.

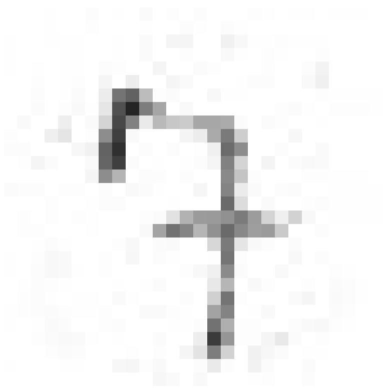
The images were now compressed to two-dimensions and plotted in different colors for different labels. If the observations are distributed in neat clusters according to labels then the PCA-model can be said to be appropriate because it is able to divide the observations in lower dimensions.

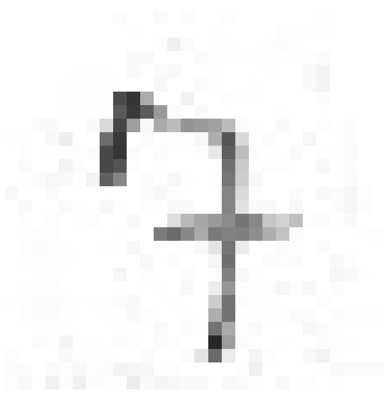
It was decided that the most stringent approach would be to calculate the eigenvectors based on the training data set and then transform the separate validation data set to a two-dimensional space using the first two eigenvectors. The model would be a good model if the validation data could be separated clearly.


Figur 39: Picture 61 label 7 standardized version

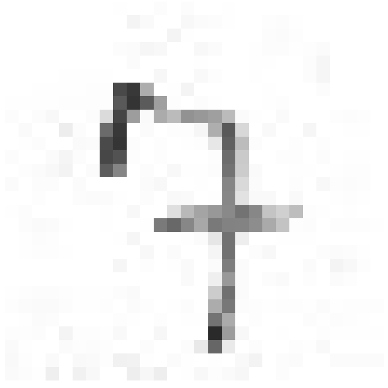
Figur 40: Picture 61 label 7 25 dimensions compress-then-uncompress

Figur 41: Picture 61 label 7 100 dimensions compress-then-uncompress

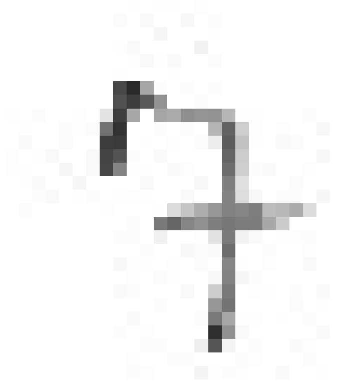

Figur 42: Picture 61 label 7 225 dimensions compress-then-uncompress



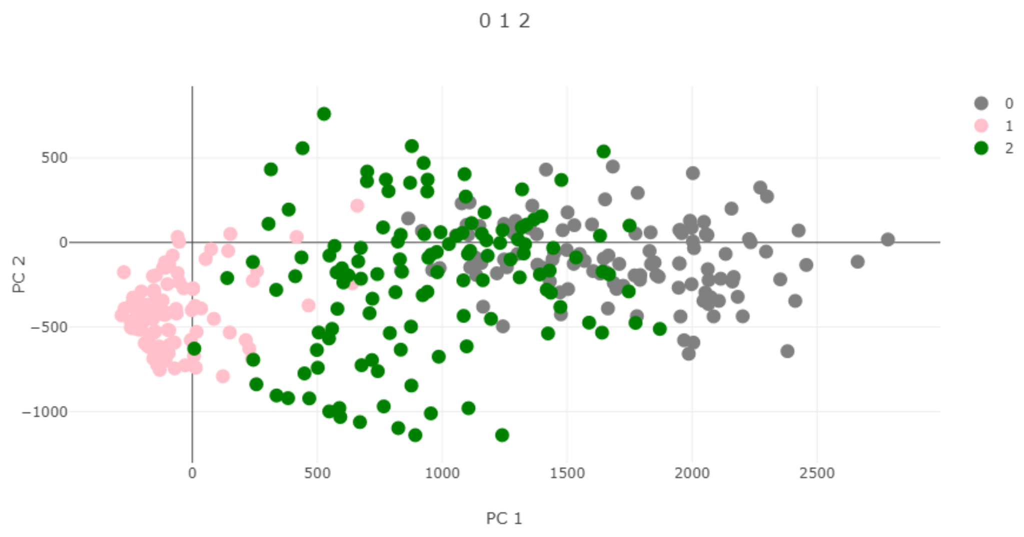
Figur 43: Picture 61 label 7 400 dimensions compress-then-uncompress



Figur 44: Picture 61 label 7 625 dimensions compress-then-uncompress



Figur 45: Picture 61 label 7 784 dimensions compress-then-uncompress



Figur 46: 0,1 and 2

5000 observations were read with 3000 in the training data set and 1000 in the validation data set. The module XPlot.Plotly was used for plotting. XPlot is avaiable for F# and takes an objectoriented approach where plots are objects with data, headlines, formats etc as object attributes.

In programming terms, the new plots were generated in a fully functional approach drawing on the functions already defined. A new master function called twoDims was defined. twoDims would use the existing projectMat function with the numPC-parameter set to 2 and would project the validation data into two-dimensional space. A list of tuples was created with the label in the first element and the 784 picture features in the second element. A function generateOneChartOfLabel would take a label value as an argument, select all pictures matching that label, generate an XPlot.Plotly scatterplot where each picture was one dot in a two-dimensional space with the first coordinate being the first coordinate of the picture projected into two-dimensional space and the second coordinate being the second coordinate of the picture. The function generateOneChartOfLabel was called by the function generateCombinedPlotOfLabels that could read as input a list of labels, generate the scatterplot for each label and plot the scatterplots together with different colors for different labels. The two functions mentioned were created as local functions within twoDims since they needed to read the tuple list as a global variable. generateCombinedPlotOfLabels could be used first to create 4 separate plots plotting respective (0,1,2), (3,4,5), (6,7) and (8,9) against each other, and then the same function could be used to generate one diagram of all 10 labels together.

Figure 46 shows pictures with the labels 0, 1 and 2 plotted against each other in two-dimensional space. The 1s sit to the left in a pink cluster and the zeroes sit in an elongated cloud towards the right.

Figure 47 shows pictures with the label 3,4 and 5. Again, some structure can be seen in the picture with the 3s in red towards the bottom and the 4s in black towards the top. Interestingly, the observations seem to be separated along the second axis but less so along the first axis.

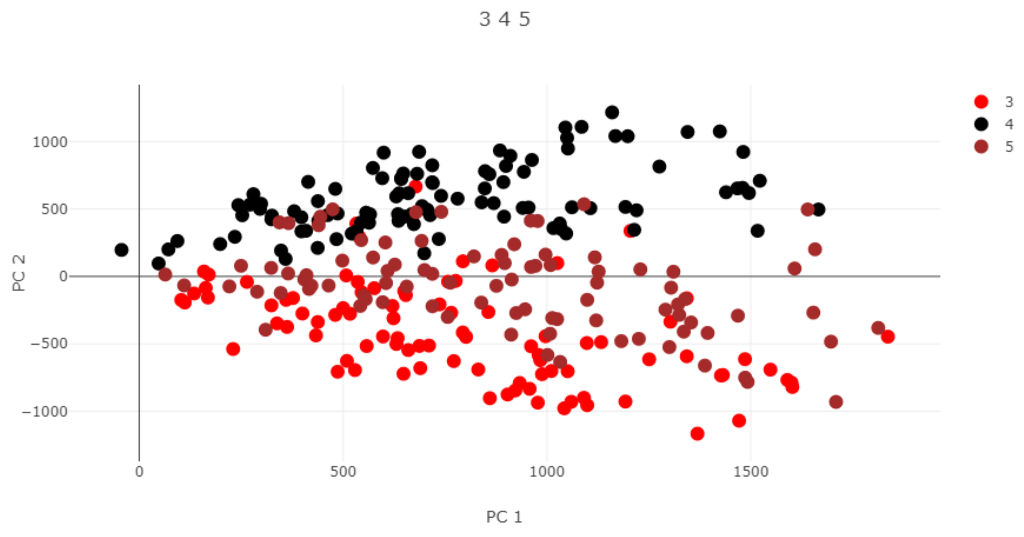
Figure 48 shows pictures with the label 6 and 7. Again, the pictures seem clearly separated but more so along the second axis. A sanity check was performed at this point as to whether the axes had in fact been swapped somehow but this did not seem to be the case.

Figure 49 shows pictures with the label 8 and 9. Again there seems to be a clear separation along the second axis. Again, caution must be observed as to these partial and visual inspections.

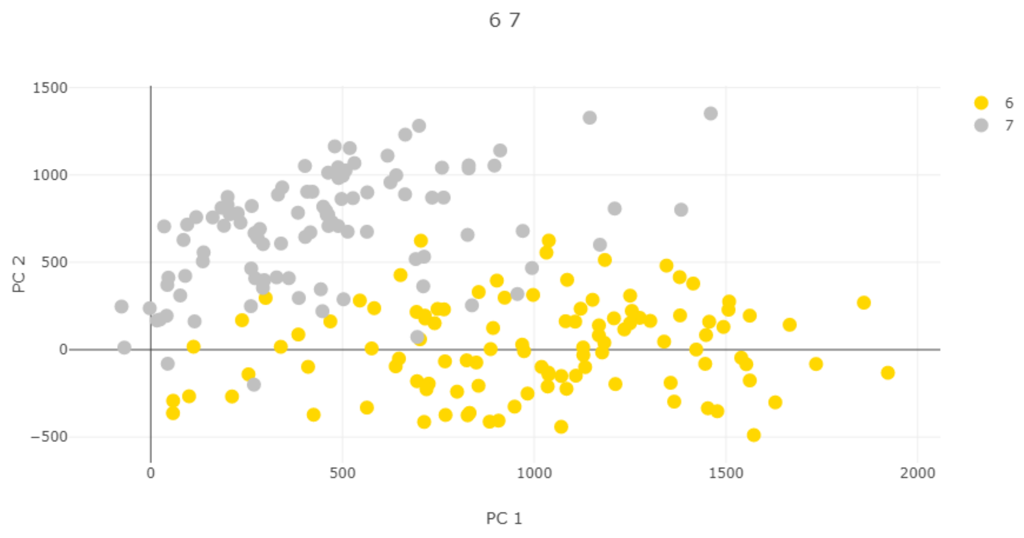
Finally, in Figure 50all 10 labels were plotted against each other.

13.16 Week 16 - 9 May to 19 May

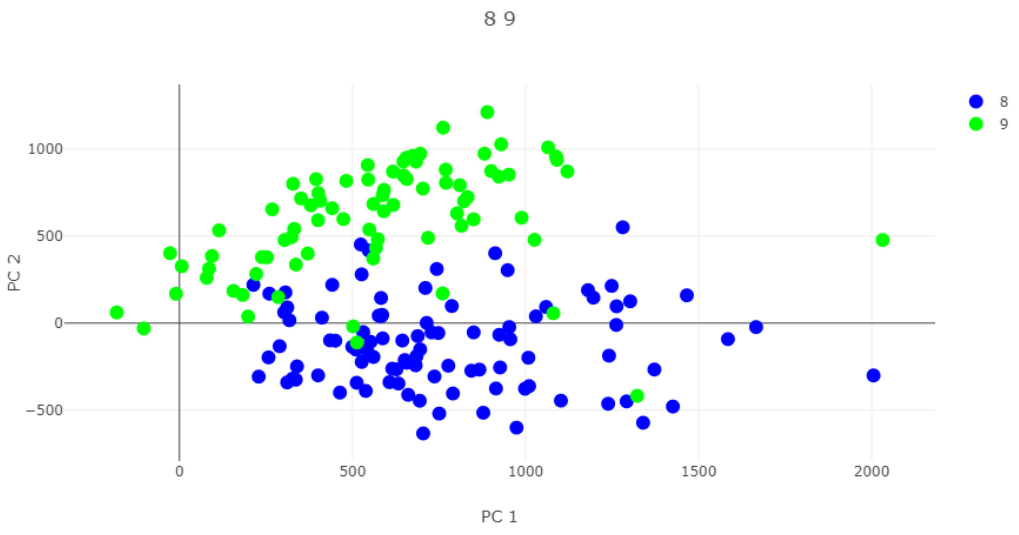
It was decided to use the data set on Kaggle `https://www.kaggle.com/datasets/hereisburak/pins-face-recognition`. In the meeting with the supervisor on the 19th of May, the following items were discussed



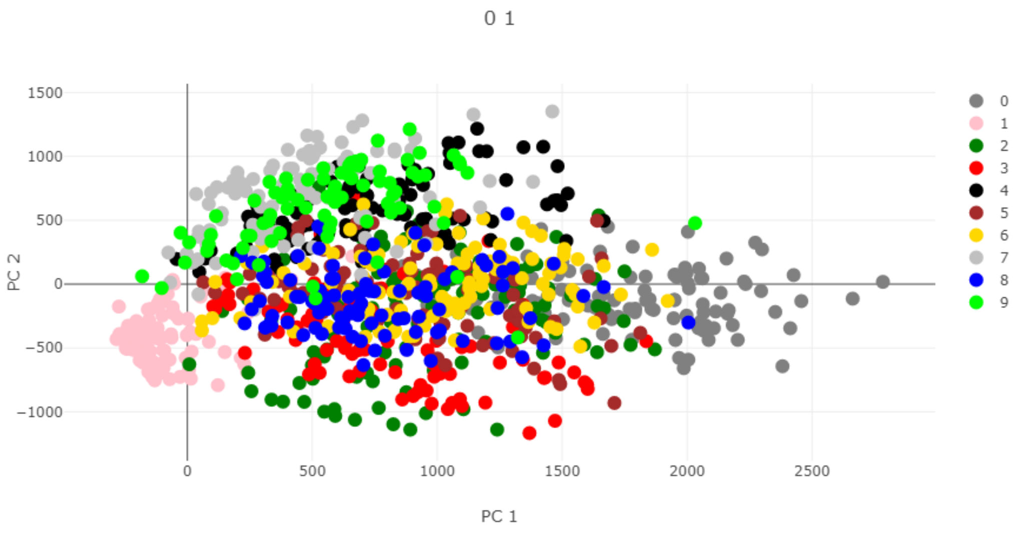
Figur 47: 3,4 and 5



Figur 48: 6 and 7



Figur 49: 8 and 9



Figur 50: All labels 0-9

- 17.000 billeder af 102 berømtheder fra Pinterest
- https://www.kaggle.com/datasets/hereisburak/pins-face-recognition
- Farvebilleder i *.jpg – hvordan transformeres?

  (Hvis første pixel er A=120,B=110,G=120,R=130, er de første tre features så [110,120,130 ...]) Solomon og Breckon p. 7 understøtter dette - men gælder det også når features skal anvendes til estimation?

  Hvordan standardiseres på tværs af forskellige bmp.Width og bmp.Height?

  Hvordan centreres billeder så øjnene sidder i midten?
- Hvert billed har nu ca. 33.000 features – betyder det at PCA fremover skal afvikles i cloud? (MathNet.Numerics går ned)?
- Afvikling af almindelig k-nearest neighbor med 33.000 features, 2.401 observationer.

  Gennemsnitlig accuracy 0.09 hvilket er lavt – men med 102 forskellige ansigter faktisk 9 gange bedre end baseline
- Idé til videreudvikling: Tilbage i februar var tanken at lave en ansigtsgenkendelsesapp til brug ved forelæsninger og konferencer.

  Use case: fire ansigter ved siden af hinanden

  Syntetiske billeder genereres

  Hvis billede 1 forudsiges til at være person A fjernes person A fra træningssættet til forudsigelsen af billede 2 og så fremdeles.

  Vil forøge running time men forbedre præcision i demografier med mange personer, der ligner hinanden (disse udvælges i Pins).

  Testes i forhold til generel model.

It was discussed that the Pins data set was dense in that it had between 100 and 200 images for each of the 102 celebrities in the data set.

The supervisor commented that the model accuracy of 9 per cent showed that a pure k-nearest-neighbor approach had its limitation. It was clear that scaling, cropping and centering of the images was required for a usable model.

A cropping function had been implemented in functional programming https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/scaleAndCrop29052022.fsx. The function cropBmpo read a bitmap as well as the coordinates within the bitmap to be used for cropping, then called two subfuntions findNewCoordinates and findOldPixels in order to position the relevant old pixels into the new coordinates in the new, smaller bitmap. List.init was used to execute this as many times as there were pixels in the new bitmap, and List.init produced a virtual list of units. The actual updating of the pixels was done within the List.init-statement. The new bitmap was created as an object called newBmpo and updating was done with the object method newBmpo.SetPixel(). It was concluded by visual inspection that the function cropped bitmaps correctly as shown in Figures 51 and 52

The student raised the issue that with this number of features the model had become difficult to operate in MathNet.Numerics and asked whether calculations should be transitioned to the cloud. The supervisor suggested reducing the matrix to its column space which would reduce the number of required transactions.

13.17 Week 17 - 19 May to 23 May

In order to center all pictures so that appropriate features were compared, the intention was first to use the Accord-library in F#, but it seemed to read a backlevel version of the System.Drawing.Common.dll-file and overwrite the current version that was imported via nuget. It was considered to use the Mobilenet library which is a current, python-based library. As a temporary workaround, it was decided to use an old version of opencv which the student has experience working with. A static approach was imposed where a python-scipt https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/read_from_openCV.py read d all the jpg-files from the Pins-data set, then calculated the bounding boxes using the opencv-module, then created a tuple with the file name of the jpg and the

Figur 51: Before cropping

Figur 52: After cropping

bounding box values and then wrote all tuples into a .txt-file. It is clear that this static approach will only serve to calibrate the model and a better approach must be found for the final app.

In the meeting with the supervisor on the 23rd of May, the student stated that he intended to discard the 20 per cent smallest images, then crop all other images to a joint format based on the bounding boxes. The supervisor commented that a better approach would be a combination of scaling and cropping. He suggested upscaling the smaller pictures rather than downscaling the larger pictures. The supervisor commented that interpolation was difficult to avoid.

It must be noted that the Pins-data set contains many different picture sizes, so a systematic approach must be taken.

13.18 Week 18 - 23 May to 31 May

Several considerations went into the creation of a logical and efficient way of scaling the pictures. One option was using built-in methods (openCV for instance has one) but these would have the same dependency on creating a real-time integration from F# to python. The student determined that coding a scaling algorithm was an appropriate test case for the use of the functional paradigm for image recognition.

It was then considered whether a scaling algorithm should be general (scale from any size to any size) or whether it should be limited to a set of pre-defined formats. It was decided that general was the best use case.

Since a functional approach was followed, it was decided not to define the upscaled image as an object and then imperatively populate the pixels of the object with either original pixels or interpolated pixels following an appropriate logical condition.

A recursive approach was then considered, starting by the last pixel of the original object and then adding an interpolated new pixel through a recursive process. This would be the most classic functional approach, but it lent itself better to scaling between predefined formats e.g. *double the number of pixels by adding one new pixel for each existing pixel.*

If a truly general approach were to be followed, it should also support "odd"cases such as *extend a 39 element list by adding 57 new elements.* It was concluded that this was best done by a function that could read the element index not just the element.

Therefore, as shown in https://github.com/TimMondorf/PUK-Image-Recognition-FSharp/blob/main/scaleAndCrop29052022.fsx, an approach based on List.init was selected. Elements of lists were transformed from floats to custom record types called elemento who had the following values index being the index, thisElem being the value of the element itself, nextElem the value of the following element to be used for interpolation, howManyFollowers being the number of new interpolated elements to be inserted before the next original element, and followers being these new elements. A function genElemento read the elements and defined elemento records. The function calcFollowers calculated the number of followers. By definition, the last element in the list could have none, otherwise the number of followers was defined by spreading the required number of additional elements evenly in all intervals, beginning from the end of the list. In case the number of new elements was not a multiplum of the number of intervals, the remaining elements would be placed again beginning from the right. Once the number of followers were known for each element, the followers themselves were calculated by the interPol function that interpolated between thisElem and nextElem. The difference rather than the relative difference was applied since the color values are bounded between 0 and 255. The fact that all list elements did not have the same number of followers was catered for by applying the List.collect-function that returned one long list of original elements with new follower elements in the appropriate intervals.

The function scalePico would take a data set picture as an argument, read the features as one long list and generate a list of lists with as many rows as there are rows of pixels. Each list in the list would have the same length as the pixel width of the bitmap, multiplied by 3 since one pixel represents three float values for RBG - Red, Blue and Green. scalePico would also take as arguments the requested new width and length of the bitmap and call the scaleRows function. scaleRows would triage the list into three sublists of reds, blues and greens, so that the green in each interpolated pixel was interpolation of the green before and after. scalePico would then transform features to elemento and call the interpol function.

scaleRows returned a long list of all features representing a bitmap that had been scaled in the horizontal direction but not yet in the vertical direction. These feature values were then organized into a new list of as many elements as the height of the original bitmap. Each element was a list of floats,

Figur 53: Original photo for before scaling

as many as the width of the scaled bitmap. This list of lists was then transposed. By transposing, red values were immediately next to red values, blue next to blue and green next to green. It was therefore not necessary to triage the values through the scaleRows function. Instead, the interPol function was curried with the requested new height of the bitmap and then applied directly to the transposed list of lists. Subsequently, the list of lists was transposed back and List.collect was again applied to even it out to one single list that now represented feature values of the scaled bitmap.

The values of the bounding box were adjusted correspondingly and were read into a new custom record type pinsPic representing label, bitmap, features and bounding box.

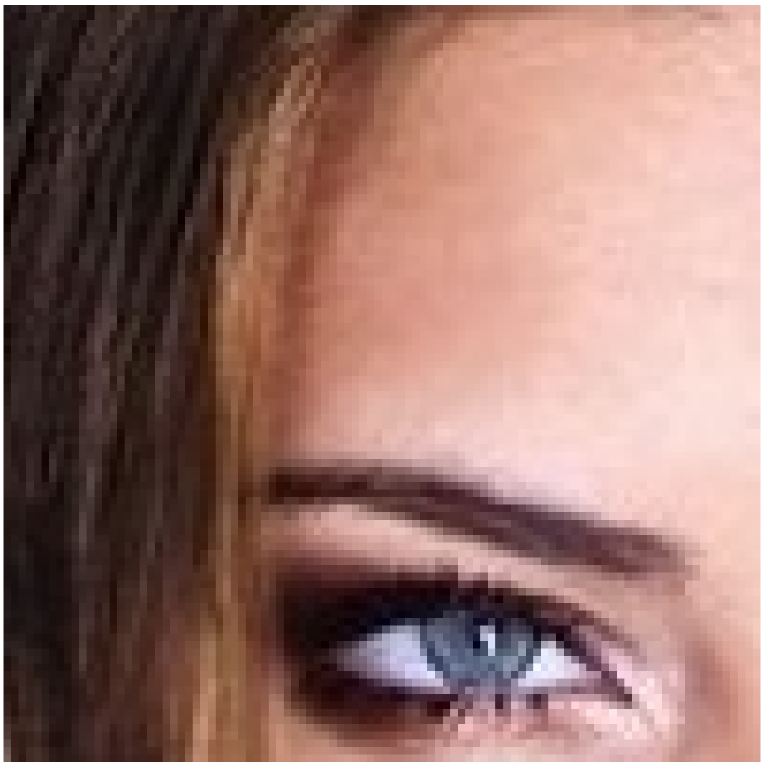
To test the function, a random picture was selected from the Pins data set, a smaller portion of 100x100 was cropped and this cropped picture was then scaled to 120x120. The original can be seen in Figure 53, the cropped version in Figure 54 and the scaled version in Figure 55.

It is notable that the scaling entails a distortion of the colours and this will be investigated further.

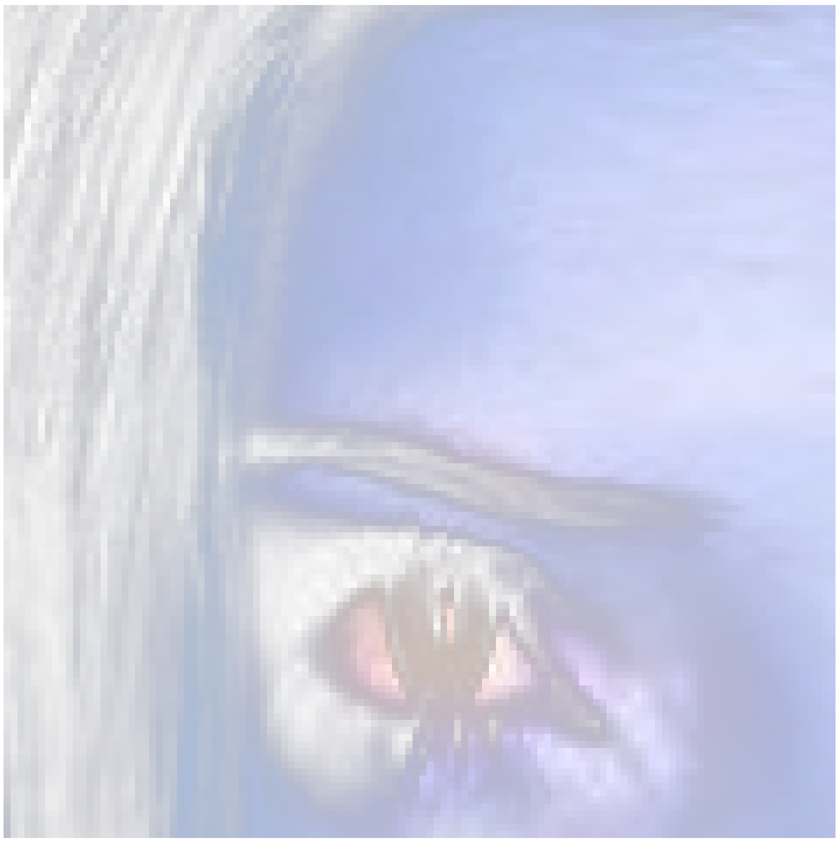
13.19 Week 19 - 31 May to 22 June

In the meeting with the supervisor on 31 May, a strategy for root cause analysis of the blue images was discussed. It was suggested

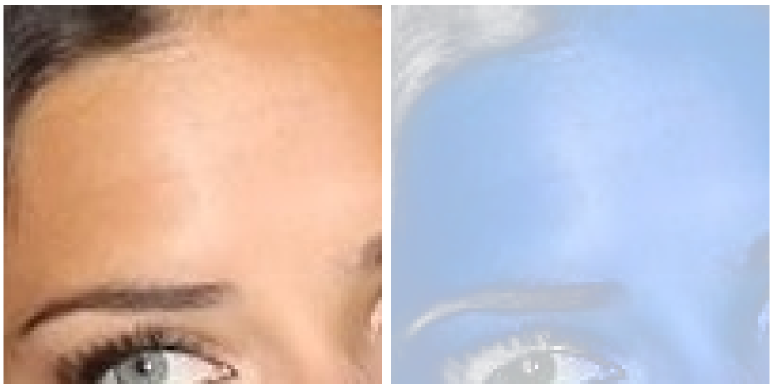
- create a 2x2-bitmap and scale it to 3x3
- scale a bitmap to its original size, this should preserve the colors
- create an image where all colors are set to 255, then subtract a scaled picture from this picture. This would reveal whether the color scheme had been inverted somehow.
- verify whether the color channels including the alpha channel were set correctly in the SetPixel-function.
- verify whether the function operates on pixels or on the feature vector. It would be preferable that it operated on a pixel level
- test the function on the MNIST-data set which is monochromatic. Scaling the hand-written digits should preserve the original format



Figur 54: Photo after cropping to 100x100before scaling



Figur 55: Photo after cropping to 100x100 and scaling to 110x110



Figur 56: Original picture (left) and picture resized from 100x100 to 100x100 (right)

The conceptual approach to scaling was also discussed. The initial approach of the student had been to preserve the original pixels, calculate the numer of additional pixels required for the scaled format and then inserted the new pixels into the intervals between the originals.

The supervisor suggested an alternative approach of generating the new bitmap by calculating only new pixels, the values of which would correspond to their relative proximity of the original pixels. It was discussed that this would be a more functional approach, generating the new bitmap as a function of the old bitmap. It would also provide a smoother image. It was also discussed that the scale function could operate directly on pixels rather than on the feature float list, and that in any event the custom record type pinsPic did not have to carry both the bitmap and the feature float list.

The root cause analysis on the original programme was carried out. First it was verified that the alpha channel is in fact the first argument of the F# System.Drawing namespace `https://docs.microsoft.com/en-us/dotnet/api/system.drawing?view=net-6.0`.

Then a picture was cropped to 100x100 then resized to 100x100, which is shown in Figure 56. It was clear that the distortion of the color scheme persisted.

Then a 2x2 bitmap is defined and all 4 pixels are given the color values (0,1,1,1). Interestingly, as can be seen in Figure 57, the interpolation process conserves the RBG-colors but changes the alpha channel from 0 to 1.

As a next step, the expected results of the interpolation function on small bitmap with non-trivial values are mapped out in an excel spreadsheet as shown in Figure 58. The colors of the bitmap will grow grow along both axes so the 5 new pixels in the middle row/column will be the result of interpolation, shown for simplicity for one color and then in the ARGB-format.

Scaling the bitmap in the programme yields the expected result as regards the colors R, G and B. The alpha channel appears to be updated to always be the average of the other colors, as shown in Figure 59

13.20 Week 22 - 22 June to 29 June

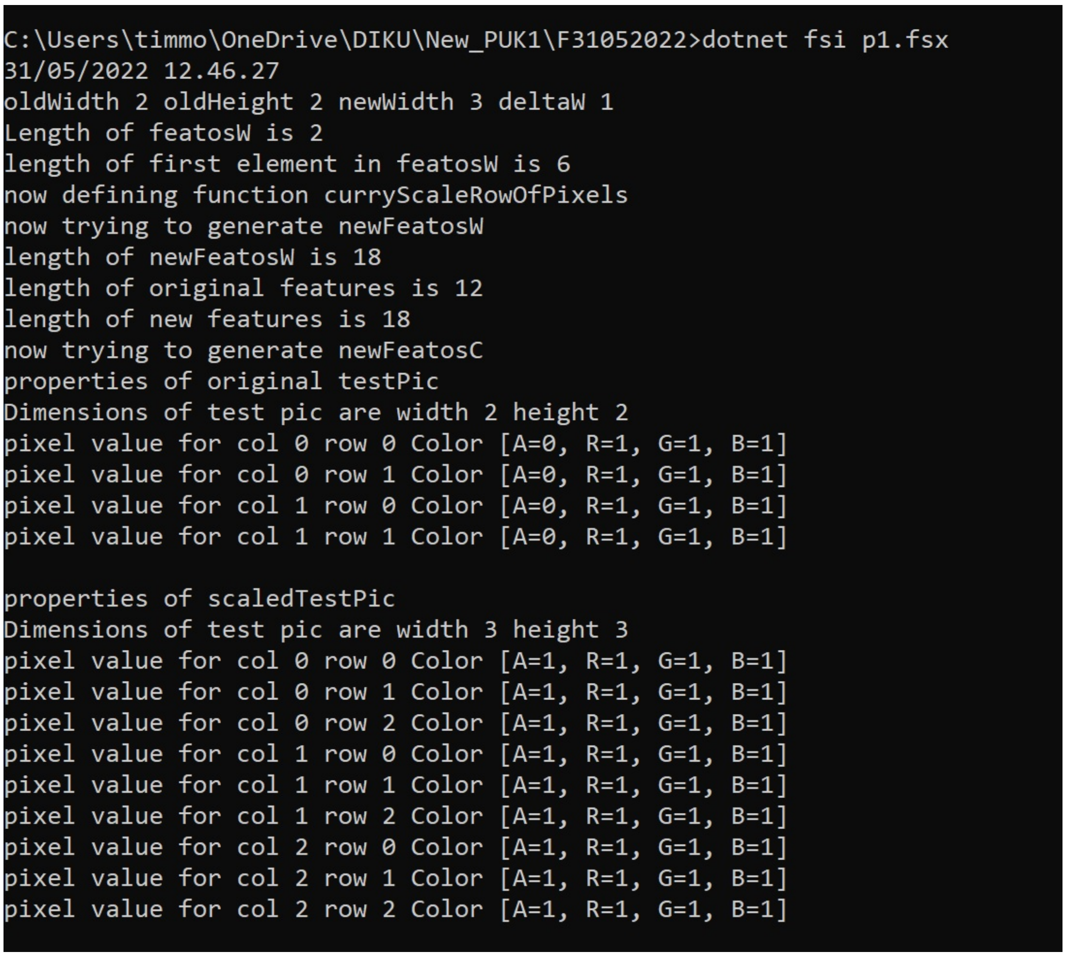
In the meeting with the supervisor on 22 June, it was discussed how piping from one function to another was an integral part of functional programming. F# inferred typing but verified these at compile time.

It was discussed that images in particular had a higher order neighbor relationship that was difficult to analyze. The student mentioned an example where Google Photos had predicted a commonality between a selfie taken in front of a tower in Tivoli and one taken in front of a boardwalk, simply because both photos had a vertical structure in the same part of the picture, as seen in Figure (60). The supervisor was not sure whether this was the an expression of higher order neighbor relationship between the two pictures.

The supervisor noted that k-nearest neighbor and PCA were approaches that had been available for a number of years and that the project should end with a benchmarking of model performance against a neural network-model in python. It was discussed that in addition to providing a benchmark, neural networks might not be so dependent on having to center to pictures for comparability.

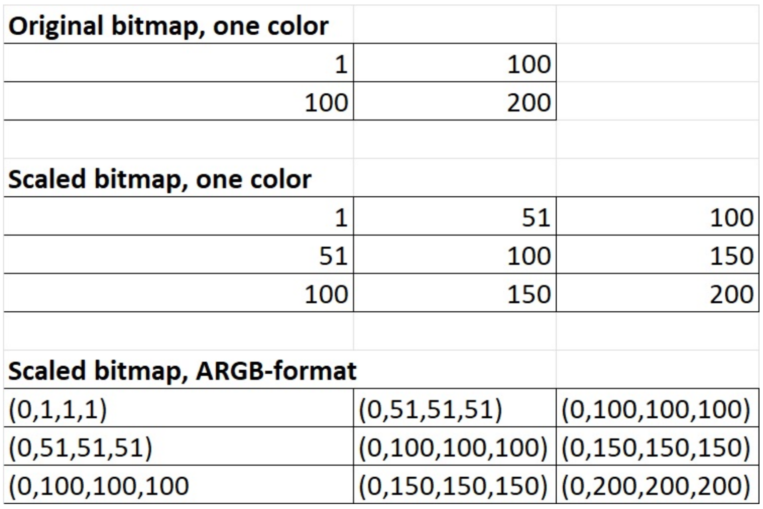
List.map and List.init were higher order functions in the sense of the functional paradigm.

Functional programming imposed a mathematical thinking of how functions mapped from the input space to the output space.



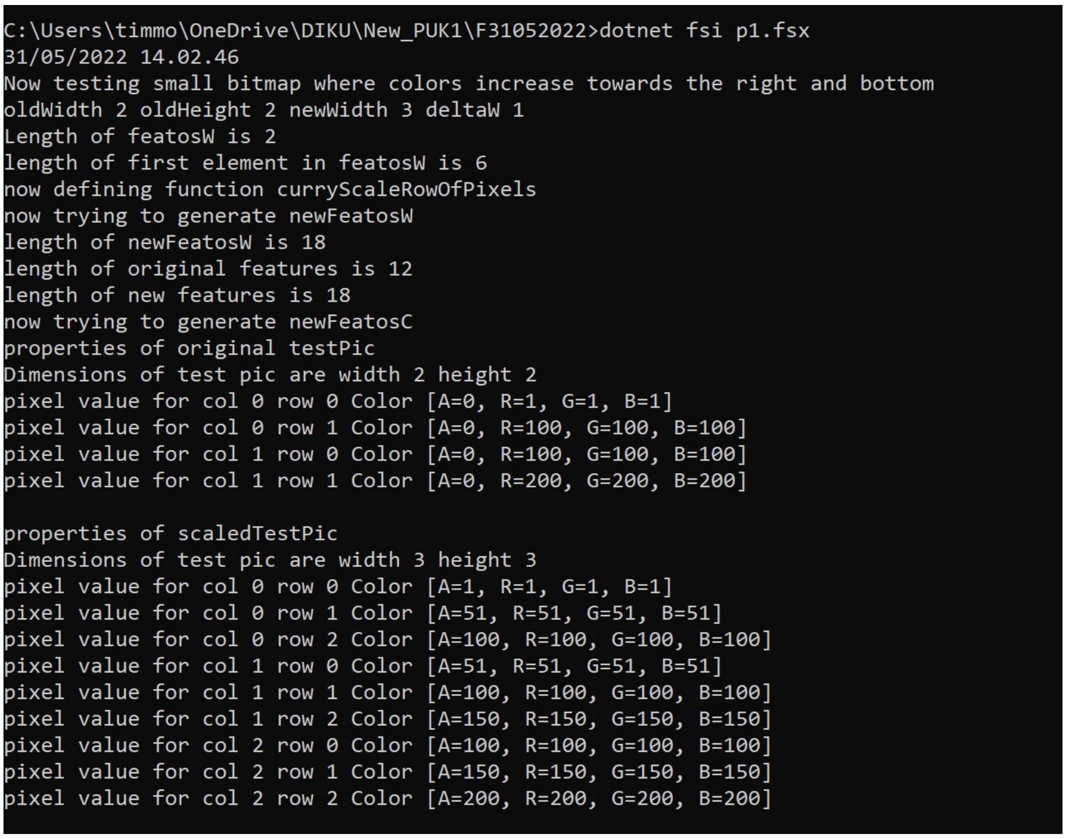
```
C:\Users\timmo\OneDrive\DIKU\New_PUK1\F31052022>dotnet fsi p1.fsx
31/05/2022 12.46.27
oldWidth 2 oldHeight 2 newWidth 3 deltaW 1
Length of featosW is 2
length of first element in featosW is 6
now defining function curryScaleRowOfPixels
now trying to generate newFeatosW
length of newFeatosW is 18
length of original features is 12
length of new features is 18
now trying to generate newFeatosC
properties of original testPic
Dimensions of test pic are width 2 height 2
pixel value for col 0 row 0 Color [A=0, R=1, G=1, B=1]
pixel value for col 0 row 1 Color [A=0, R=1, G=1, B=1]
pixel value for col 1 row 0 Color [A=0, R=1, G=1, B=1]
pixel value for col 1 row 1 Color [A=0, R=1, G=1, B=1]

properties of scaledTestPic
Dimensions of test pic are width 3 height 3
pixel value for col 0 row 0 Color [A=1, R=1, G=1, B=1]
pixel value for col 0 row 1 Color [A=1, R=1, G=1, B=1]
pixel value for col 0 row 2 Color [A=1, R=1, G=1, B=1]
pixel value for col 1 row 0 Color [A=1, R=1, G=1, B=1]
pixel value for col 1 row 1 Color [A=1, R=1, G=1, B=1]
pixel value for col 1 row 2 Color [A=1, R=1, G=1, B=1]
pixel value for col 2 row 0 Color [A=1, R=1, G=1, B=1]
pixel value for col 2 row 1 Color [A=1, R=1, G=1, B=1]
pixel value for col 2 row 2 Color [A=1, R=1, G=1, B=1]
```

Figur 57: Scaling a 2x2-bitmap with the same color in every pixel



Original bitmap, one color		
1	100	
100	200	
Scaled bitmap, one color		
1	51	100
51	100	150
100	150	200
Scaled bitmap, ARGB-format		
(0,1,1,1)	(0,51,51,51)	(0,100,100,100)
(0,51,51,51)	(0,100,100,100)	(0,150,150,150)
(0,100,100,100	(0,150,150,150)	(0,200,200,200)

Figur 58: The expected effects of interpolation mapped out in excel



```
C:\Users\timmo\OneDrive\DIKU\New_PUK1\F31052022>dotnet fsi p1.fsx
31/05/2022 14.02.46
Now testing small bitmap where colors increase towards the right and bottom
oldWidth 2 oldHeight 2 newWidth 3 deltaW 1
Length of featosW is 2
length of first element in featosW is 6
now defining function curryScaleRowOfPixels
now trying to generate newFeatosW
length of newFeatosW is 18
length of original features is 12
length of new features is 18
now trying to generate newFeatosC
properties of original testPic
Dimensions of test pic are width 2 height 2
pixel value for col 0 row 0 Color [A=0, R=1, G=1, B=1]
pixel value for col 0 row 1 Color [A=0, R=100, G=100, B=100]
pixel value for col 1 row 0 Color [A=0, R=100, G=100, B=100]
pixel value for col 1 row 1 Color [A=0, R=200, G=200, B=200]

properties of scaledTestPic
Dimensions of test pic are width 3 height 3
pixel value for col 0 row 0 Color [A=1, R=1, G=1, B=1]
pixel value for col 0 row 1 Color [A=51, R=51, G=51, B=51]
pixel value for col 0 row 2 Color [A=100, R=100, G=100, B=100]
pixel value for col 1 row 0 Color [A=51, R=51, G=51, B=51]
pixel value for col 1 row 1 Color [A=100, R=100, G=100, B=100]
pixel value for col 1 row 2 Color [A=150, R=150, G=150, B=150]
pixel value for col 2 row 0 Color [A=100, R=100, G=100, B=100]
pixel value for col 2 row 1 Color [A=150, R=150, G=150, B=150]
pixel value for col 2 row 2 Color [A=200, R=200, G=200, B=200]
```

Figur 59: Scaling a 2x2-bitmap with non-trivial colors

13.21 Week 23 - Meeting 29 June

The student reviewed the suggestion of benchmarking the project against a neural network-model in python tensorflow or pytorch. His view was that this was not the best fit for the project's substantial focus on F# and functional programming. Through further internet search, the student found an F#-programme that performed neural networks while clearly documenting the advantages from functional programming such as typing. The programme, published on [15] with the code available on Github, did not seem mature, as the student became the first Github-follower. Nevertheless, it was deemed the best approach to seek to conclude the programme with an application of this F#-code, used for benchmarking the previous models.

The student then picked up the work that had been initiated in Week 19. Rather than pursue the root cause analysis, the student decided to build the new scaling function that had been discussed where additional pixels were not inserted among the existing pixels, but rather a vector of entirely new pixels was created.

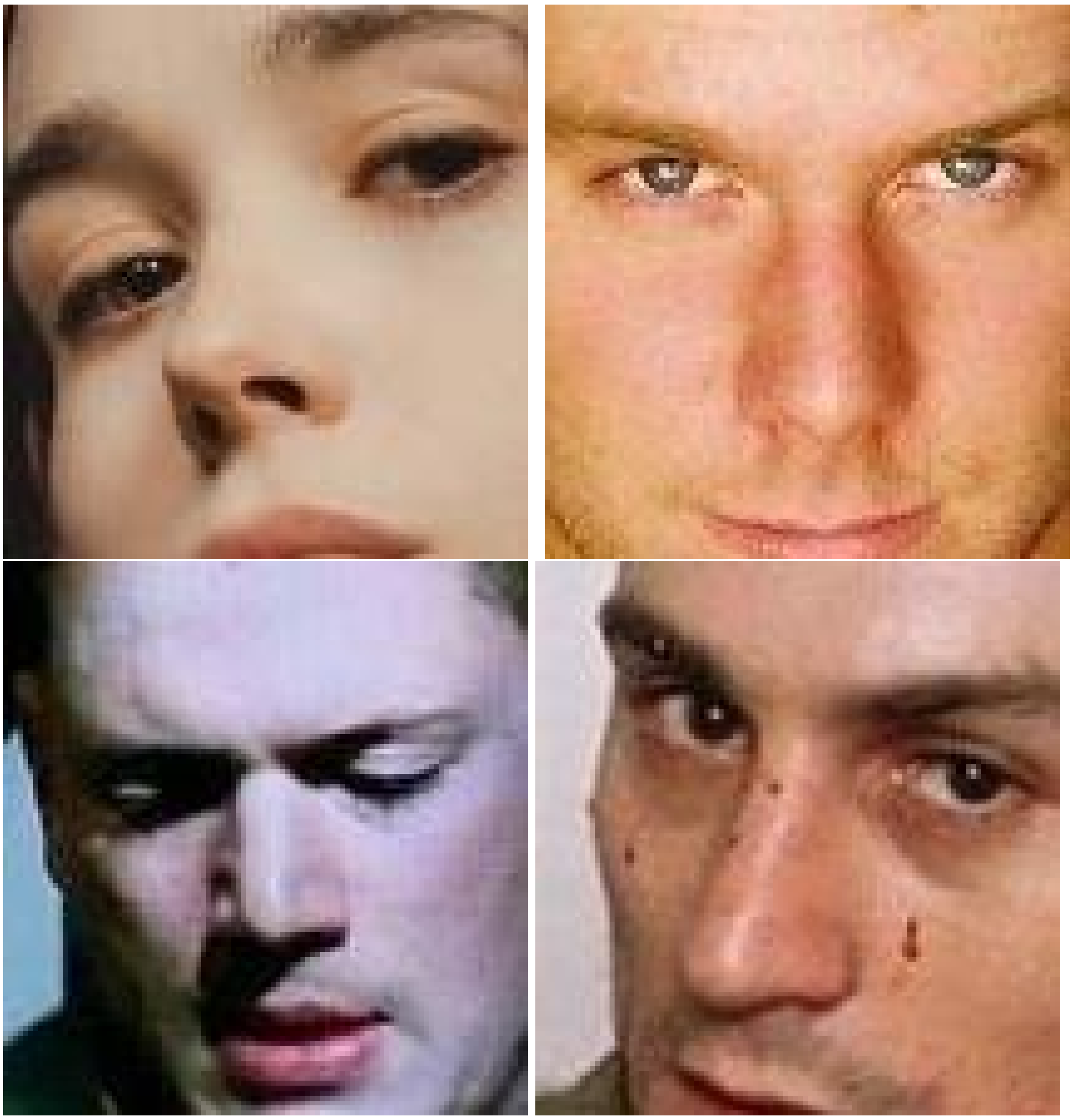
13.22 Week 24 - Meeting 4 August

After the summer vacation, the student decided to refocus the project on the knn- and PCA-analyses performed up to this point, with a view to meeting the established timeline. Focus at the later stage of the project was on saving bitmap-files that had been numerically generated by the F#-code. Whereas the programme was able to read original .jpg-files into bitmaps and to reproduce these using the bitmap.Save("filename.jpg")-function, this did not work with bitmaps that had been originated from inside the programme, for instance a (2x2)-bitmap with the pixel-values (1,2,3,4) could not be saved in a form so that it was readable by Latex.

14 Appendix V. Examples of pictures cropped around centre

Randomly selected pictures are shown in Figure (61).

Figur 60: Two photos that Google Photos selected as having commonality



Figur 61: Examples of photos centered on the center of the face and cropped according to defined boundaries

15 Literature

Litteratur

[1] Mnist. Digit Recognizer Learn computer vision fundamentals with the famous MNIST data, https://www.kaggle.com/c/digit-recognizer.

[2] R. Alturki. Research onion for smart iot-enabled mobile applications, https://downloads.hindawi.com/journals/sp/2021/4270998.pdf. *Hindawi Scientific Programming*, 2021.

[3] A. Bagdanov. A functional approach to software design in image processing research environments, http://www.cvc.uab.es/~bagdanov/thesis/thesis_08.pdf. Computer Vision Center.

[4] E. Boudreau. What is a programming paradigm?, https://towardsdatascience.com/what-is-a-programming-paradigm-1259362673c2. Towards Data Science, October 2020.

[5] D. Chakraborty. Image compression using principal component analysis (pca), https://www.section.io/engineering-education/image-compression-using-pca/. EngEd Community, September 2021.

[6] C. M. . S. Curtis. Educational pearl fractal image compression. *Journal of Functional Programming*, (23 (6)):629–657, December 2013.

[7] M. F. de Alarcón Gervás. Functional programming paradigm for machine learning algorithms in data mining, https://repositorio.uam.es/bitstream/handle/10486/688868/fernandez_de_alarcon_gervas_miguel_tfg.pdf. Escuela Politecnica, Universidad Autonoma de Madrid, June 2019.

[8] L. Fridman. Bjarne stroustrup lex fridman podcast 48, https://www.youtube.com//watch?v=uTxRF5ag27A. Youtube, November 2019.

[9] Hereisburak. Pins face recognition, https://www.kaggle.com/datasets/hereisburak/pins-face-recognition. Kaggle.

[10] Z. Hu. How functional programming mattered (with hughes and wang). *National Science Review*, (2):349–370, 2015.

[11] J. Hughes. Why functional programming matters. *Research Topics in Functional Programming*, pages 17–42, 1990.

[12] KUScience. Den fælles del af bachelor- og kandidatstudieordningerne for uddannelserne ved det natur- og biovidenskabelige fakultet københavns universitet, https://science.ku.dk/studerende/studieordninger/faelles_sto/faelles-del.pdf, September 2021.

[13] T. Lavers. Image processing is the perfect playground for functional programming, https://towardsdatascience.com/functional-programming-for-deep-learning-bc7b80e347e9. Medium, May 2017.

[14] W. Loder. *Erlang and Elixir for Imperative Programmers*. Apress, 2016.

[15] H. Mehmood. Building neural networks in f — part 1, https://towardsdatascience.com/building-neural-networks-in-f-part-1-a2832ae972e6 and part 2 https://towardsdatascience.com/building-neural-networks-in-f-part-2-training-evaluation-5e3a68889da6. Towards Data Science, May 2018.

[16] T. Mondorf. Timmondorf-puk-image-recognition-fsharp, https://github.com//TimMondorf//PUK-Image-Recognition-FSharp. Github, August 2022.

[17] S. Mukherjee. *F# for Machine Learning Essentials*. Packt Publishing, 1 edition, 2016.

[18] G. Neumann. Programming languages in artificial intelligence, https://www.dfki.de/~neumann/publications/new-ps/ai-pgr.pdf. German Research Center for Artificial Intelligence (LT–Lab, DFKI), 1994.

[19] G. Orwell. *Politics and the English Language*. Pamphlet, 1946.

[20] R. Pickering. *Beginning F# 4.0.* Apress, 2 edition, 2016.

[21] I. Poole. U sing the functional programming language haskell to specify image analysis systems. *Directions in Safety-Critical Systems*, 1993.

[22] M. Saunders. *Research Methods for Business Students.* Prentice-Hall, 2009.

[23] J. Sporring. *Learning to Program with F#.* Department of Computer Science, Copenhagen University, draft edition, 2021.

[24] D. Syme. *Expert F# 4.0.* Apress, 4 edition, 2015.

[25] S. Tiwari. Face recognition with python, in under 25 lines of code, https://realpython.com/face-recognition-with-python/. Real Python.

[26] Wikipedia. Programming paradigm, https://en.wikipedia.org/wiki/Programming_paradigm.

[27] J. Wu. The beauty of functional languages in deep learning—clojure and haskell, https://www.welcometothejungle.com/en/articles/btc-deep-learning-clojure-haskell. Welcome to the Jungle, September 2019.

[28] J. Xu. Functional programming for deep learning, https://towardsdatascience.com/functional-programming-for-deep-learning-bc7b80e347e9. Towards Data Science, June 2017.