Benefits of Layered Software Architecture in Machine Learning Applications

Ármin Romhányi and Zoltán Vámossy
John von Neumann Faculty of Informatics, Obuda University, Bécsi Street, Budapest, Hungary

Keywords: Machine Learning, Layered Architecture, Software Engineering, Facial Authentication, Facial Landmark Points.

Abstract: The benefits of layering in software applications are well-known not only to authors and industry experts, but to software enthusiasts as well because the layering provides a testable and more error-proof framing for applications. Despite the benefits, however, the increasingly popular area of machine learning is yet to embrace the advantages of such a design. In the present paper, we aim to investigate if characteristic benefits of layered architecture can be applied to machine learning by designing and building a system that uses a layered machine learning approach. Then, the implemented system is compared to other already existing implementations in the literature targeting the field of facial recognition. Although we chose this field as our example for its literature being rich in both theoretical foundations and practical implementations, the principles and practices outlined by the present work are also applicable in a more general sense.

1 INTRODUCTION

Using a layered architecture in software applications has become a popular element of software design. Benefits such as a more future-proof design or the use of the "Law of Demeter", are not only listed by acknowledged works on software engineering (Sommerville, 2011; Fowler, 2012) but industry experts and enthusiasts alike also organize their work in this manner to produce more testable, understandable and robust software. This manner of design, however profound it may be in the industry, does not seem to influence the architectural design of machine learning applications. Theoretical works aiming to improve on already existing techniques (Wu et al., 2017; Rodriguez and Marcel, 2006; Wu and Ji, 2015) as well as practical software solutions of this kind (Aydin and Othman, 2017; Kumar and Saravanan, 2013; Weinstein et al., 2002) adopt a rather monolithic way of organizing machine solver algorithms into one single component. In these works, there is either no reference to the architecture whatsoever (since a new principle is at scrutiny not its immediate applications), or the designed system adopts a well-known architectural design principle (e.g., client-server) *and* one of the components is solely responsible for machine learning operations. This component is usually deployed to a performant server computer, but building on the increasing computational power available to mobile devices, there are applications where a handheld device assumes this role (Hazen et al., 2003; Weinstein et al., 2002).

In an attempt to connect this software engineering principle to machine learning, the present work aims to apply the concept of layering to a machine learning solution from an arbitrarily-chosen field: facial recognition. This is achieved by adapting well-established characteristics of layering to the problem in question, resulting in a software application in which multiple components working in tandem are responsible for the task of identifying persons based on their facial features. Since this work concentrates on principles rather than concrete applications, the software in design is intended to be a prototype only and as such, the building blocks from which it is composed of are purposefully made as simple as possible to determine the lower limits of such systems. For this reason, the component structure of the prototype is based on the simple client-server architecture and even its most complicated part - the server-side neural network solution - is confined to the bare minimum of its potential.

In the remainder of this article, the problem in question is briefly presented first accompanied by a literature review strictly restricted to the topics relevant to the present work. This is followed by establishing design principles for solving it using a client-server configuration. The components of the system

Romhányi, Á. and Vámossy, Z.
Benefits of Layered Software Architecture in Machine Learning Applications.
DOI: 10.5220/0010424500660072
In *Proceedings of the International Conference on Image Processing and Vision Engineering (IMPROVE 2021)*, pages 66-72
ISBN: 978-989-758-511-1

are described afterwards with special attention devoted to the purpose they play in solving the machine learning problem. The system is also evaluated as to what extent it adheres to the established principles and comparisons are made between the performance of already existing applications of similar purposes.

2 LITERATURE REVIEW

One of the most widely-studied element of machine vision is facial recognition, a field revolving around searching for visual cues of human faces on images without any assumption of its portrayed contents. This problem is solved using a toolbox similar to image clustering and shape identification as facial features proven to exhibit a degree of variance within the individual are at the center of recognition attempts. Searching for the shape of the eyes, mouth and the nose is most common (Manjunath et al., 1992), but studies exist on using the contour lines of the chin and the brows (Walker et al., 1998) and it is also possible to exploit the difference in the geometric depth of facial features (BenAbdelkader and Griffin, 2005). The literature commonly refers to these features under the umbrella term *facial landmark points* (Walker et al., 1998; Wu et al., 2017; Wu and Ji, 2015).

Although using the same tools, the area the present study choses to explore is not facial recognition, but the problem of *facial authentication*, a subcategory of the previous, in which the images under scrutiny are guaranteed to portray human faces and the task is to make distinction between certain individuals using these images only. The main component of this practice is to calculate a difference between the reference image of an individual and the image given as a task for evaluation. There is a wide range of techniques applied to solve these kinds of problems from formal methods such as principal component analysis to machine learning approaches including artificial neural networks, but at the end of the process, all of these methods commonly produce a probability score as to how likely it is that the image in question indeed portrays the same person as the image used as a basis of reference. To thoroughly investigate the problem, the present study implements two kinds of facial authentication: one in which the system needs to identify which person is found on the image out of 3 pre-defined individuals, and another that decides whether a given image portrays the same person as the one used as a reference of that person.

A core component of performing facial recognition analyses is inevitably linked to how comparisons are made between existing and new information and thus, how data is stored and accessed during these processes. Using labelled images was the most common practice for earlier methods, which was preceded by the application of abstraction layers generated from the raw images: these techniques aimed to extract certain sub-elements of importance (e.g., colors or prominent image segments) this way reducing the size of the data to store (Liu et al., 2007). The literature commonly refers to image searches using these methods under the umbrella term *Content Based Image Retrieval* (CBIR). One of the most common methods of CBIR is applying formal transformations to image segments (Liu et al., 2007). (Karnila et al., 2019), for example, uses the mechanism of *Discrete Cosine Transform* (DCT), one of the building blocks of the JPG image format (Miklos et al., 2020), to extract coefficients from JPG images, which are stored in a database afterwards, reducing the file size from 14 kB to around 3 kb, which is even preceded by their own methods called *Discrete Wavelet Transform* that achieved a storage size of 0.4 kB. During comparisons, histograms can be created which are subjected to mathematical operations once again (e.g., simple euclidean distance) to calculate a difference. Machine learning approaches usually replace these formal processes with prediction procedures, but due to the statistical nature of machine learning data, storage methods can be largely the same. To put data retrieval speeds into perspective, (Kumar and Saravanan, 2013) recorded measurements of multiple implementations using 500 images: an implementation using common methods in the literature achieved an average retrieval speed of 3 sec. (Weinstein et al., 2002) reports measurements of approximately 2 sec, while a special DCT-based implementation by the previous authors is reported to have a retrieval speed of 1 sec, on average.

3 DESIGN PRINCIPLES

Although there have been a number of nature-inspired methodological improvements in the field of Machine Learning (Chaczko et al., 2020), to establish the core concepts of a layered machine learning solution, we rather turned to tried and proven methods applied by the field of software engineering. We chose to base our designing principles on how layering is portrayed by Ian Sommerville as his books are among the most widely-recognized works on the subject. One of the core concepts of layering in his works is the complex idea that, although layered architecture allows for the creation of individual components (Sommerville, 2011) restricting changes to a minimum number of el-

ements, these parts also have to maintain a certain degree of dependence on each other. Specifically, each layer should only transfer data to layers directly below it - a concept commonly referred to as "Law of Demeter". These characteristic features, he argues, not only promote incremental development cycles (an idea widely adopted by software developer companies to increase customer satisfaction), but it also allows for the application of "exchangable" components, in which the exact implementation of a sub-process is not part of the system, but rather included as a dependency that can be changed without any effect to surrounding other components. This decreases "brittleness" in a system by allowing for the use more robust tools to cope with changing requirements, while software testing and validation are also enhanced as "test components" can be substituted with real ones to eliminate any effects other than those of the component under test.

Building on Sommerville's ideas, the core concepts of our design principles are also going to revolve around component individualism and exchangeability. In this manner, we can conclude that any machine learning implementation using a layered architecture has to apply components that can function on their own allowing for the use of exchangeable implementations in them. This, in turn, also guarantees that changes regarding the exact problem-solving in that component cannot affect other components in any way. This is only possible if certain layers near the "extremities" of these components have a common way of passing information from one layer to another functioning as interfaces. We can achieve this by carefully choosing compatible input/output layers on conjoined poles of components, in other words, the output of the *n*. component should be the input of the *n+1*. component. In this case, the process of solving the problem as a whole can be described as consecutive acts of data transformation and at each stage (or component) the initial data is changed just enough to be acceptable by the next stage in line. This can also accommodate the idea of "Law of Demeter" implied by Sommerville: unless these processes are equivalent transformations, i.e., the output of a layer could also be its input, there can be only one direction of information exchange originating at the "uppermost" layer closest to the original input and a backward-oriented flow of information is not possible. In brief, therefore, we can conclude that a machine learning solution using a layered architecture:

- has to operate with individually usable components
- these components has to have compatible layers in their adjoined edges

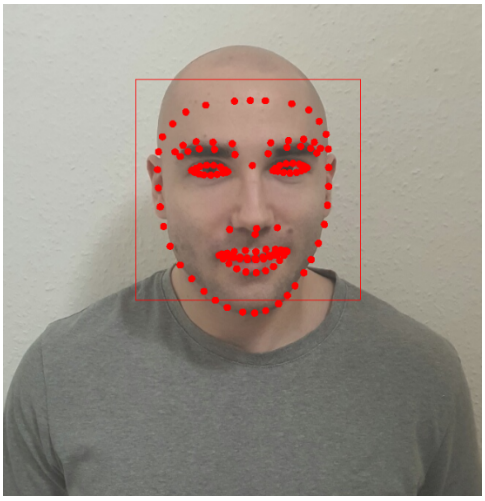

Figure 1: The image of one of the authors and the landmark points on his face as captured by the client application.

- the machine learning solution as a whole is realized as a series of acts of data transformation
- these consecutive transformations are operated on a flow of data that is expected to have a predefined, non-changeable direction

4 IMPLEMENTATION DETAILS

Conforming to both the principles outlined above and those of client-server architecture, we divided the problem into two separate sub-problems and assigned each of them to one of the components. In this manner, the client side of the software is responsible for extracting information regarding facial features in the form of landmark point coordinates that are transferred to the server for analysis in JSON format.

4.1 The Client

To be able to better compare our solution to existing methods, we implemented an Android mobile application in which the Google Firebase API is responsible for feature extraction. The workflow is intentionally kept very rudimentary and it only consists of capturing an image within the application that is followed by an automatic analysis right away. At this point, the user is given visual feedback on the result of the analysis and s/he can decide to send the image to the server or capture another image. An example can be seen in Figure 1.

4.2 The Server

The server component is implemented using a Javascript-based server-side framework because it allows for an easy cooperation with any machine learning library provided that the machine learning (ML) architecture is convertible to the format used by one of the most popular machine learning libraries *Tensorflow* by way of using one of its derivative projects, *TensorflowJS*. Since executing predictions with the designed machine learning models is delegated to TensorflowJS, this component is only responsible for handling web requests and controlling which authentication method out of the two are taken into consideration during predictions.

Although technically not part of the server layer, a brief mention must be made of the database layer here as well. The system is completely database-agnostic, thus we decided to use Microsoft SQL to store data as this is one of the most frequent database configurations at the time of writing. The application has one database table with a number of columns representing data collected during the user registration process. From the perspective of the problem, the only relevant piece of information is that the list of landmark points extracted from the user is also stored here as a basis of comparison for predictions. This data is stored as a comma-delimited consecutive line of string in which coordinates are placed next to each other.

4.3 The Machine Learning Units

The ML units were implemented using the Tensorflow ML library and the Keras API in Python. Two kinds of neural network configurations were implemented for the prototype: one for choosing a person from a pre-established pool of 3 persons (I) and another one that compares facial data of a person to data stored in the database (II). Dispite that I uses only 1 person as input, while II also needs the reference given, the two networks are using the same information from the client to produce a prediction. In this fashion, the input layer of I consists of 262 neurons (1 for each coordinate in the 131 point landmark map), while II expects an input size of 524. The output of I is a probability score for the occurrence of each person in the pool, the prediction in II results in a binary answer of whether the two data stream given as parameters represent the same person or not.

As previously stated, the system was intentionally confined to a minimum level of complexity and this is reflected in the shallow nature of the implemented network architectures. Although the solutions were tested with a great variability in configurable hyper-parameters such as the activation and loss functions, optimizers or weight initializers, their core architectures are simple. Apart from the input and output layers, I has two, while II has three fully-connected hidden layers only. It does not come as a surprise, therefore, that performance-wise the implemented networks are falling behind the established standards of an accuracy score well above 90%: on an average, I achieved an accuracy score of 75-80%, while II reached as high as 85-90%. Given that the key factor was that both of these networks are operating on the same inputs, this was deemed sufficient for the purpose of the prototype system. The details of the implemented architectures are given in Table 1.

Table 1: Hyper-parameters for the implemented neural networks.

Hyper parameter	Hyper parameter Value	
Name	I	II
Input layer size	262	524
Output layer size	3	2
Batch size	500	32
Maximum epochs	1500	1000
Weight initiliazer	LeCun uniform	LeCun uniform
Activation function	SELU	ReLU
Loss function	SCC	SCC
Optimizer	Adamax	Adam

5 EVALUATION

In this section, we present our preliminary findings and measurements in support of our claims that using a distributed approach for machine learning problems combines the advantages of layered software systems with the benefits inherent to components of the original design. Therefore, first we list two aspects of practical facial recognition, in which using landmark points - of any architectural design - can produce superior results to other common forms used in the literature, then we present measurements to support that the implemented system indeed possesses the characteristics for these aspects and finally, we show that from a software design point of view, implementing a layered facial recognition system is even more beneficial to a “monolithic” approach.

5.1 Data Storage

The present system uses the landmark extraction feature of the Android client to reduce the size of the data that needs to be stored for an analysis. Given how little information has to reside in the database, our first assumption was that this method of storage outperforms other methods in economical terms. To test this hypothesis, we conducted measurements using MSSQL for a comparison to common database-related methods and we also made estimations as to how much storage space would be required using some formal methods based on (Karnila et al., 2019).

Table 2: Preliminary measurements on storage space.

Implementation	Storage (in B)
Original JPG	5120
MSSQL image	4537
Base64 Encoded	12104
131 point landmark	5120
dlib 68 pont landmark	960
DCT (estimated)	2778
DWT (estimated)	487

The original .jpg image used a storage space of 5120 B on disk and we loaded it to the database using two formats: MSSQL *image* format intended for image storage and Base64 format - a possible, but not recommended method. We also loaded one instance of a 131 point landmark data of the original image made with our software as well as a 68 point one made with the popular image processing library *dlib*. Using a simple mathematical analogy, we also calculated what result the formal methods referenced above would achieve using our image. The results are summarized in Table 2.

Our own Firebase API 131 point approach did not achieve a data extraction rate as efficient as most of the more common methods due to the small size of the original image, but the dlib implementation managed to outperform the most common DCT approach. This could even be improved taking into consideration that dlib is experimenting with a 5 landmark point approach that would largely excel in this area. The usability of this method in facial recognition, however, is unclear at this point. It is also worth mentioning that all the landmark approaches produce an output of constant size, irrespective of the size of the original image, whereas other solutions grow in size with the original file. Therefore, we can conclude that despite that the prototype underperformed multiple methods, landmark extraction is a powerful alternative to common methods in the literature. Mention must also be made of how storage size is also a relevant factor in internet communication as lower bandwidths also consume a lower amount of resources.

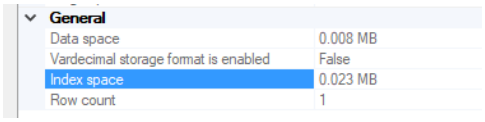

Figure 2: Storage space for 1 record in the database.

5.2 Data Retrieval

Using the referenced speed data as a basis of reference, we also conducted measurements both on inference speed and database storage. Since end-to-end testing results are not available at this point, we measured the corresponding components one by one summing the results afterwards. Inference speeds were measured using the Tensorflow library test framework that loads the testing chunks of the data and performs predicitons on each of them, simulating the worse possible scenario when the data in question is only found at the end. For both I and II, the whole process took approximately 1 s to run to completion with I conducting predictions on a sample size of 1215, while in the case of II, 8221 chunks were used.

Database retrieval speeds are no doubt uniquely specific to each implementation, so at this stage, we only measured the speed of our own MSSQL infrastructure to provide a basis of comparison for other methods. A Microsoft benchmarking software publically available on Github[1] was used to conduct the measurements. This utility software can generate a data chunk of any size after which data retrival processes are launched and measured for a preconfigured time. Using the storage space required to store 1 record of our data (Figure 2), we estimated 1000 records to be approximately of 24 MB in size. The measurements were running for 120 s, and as a result, an average retrieval speed of 453 ms was recorded. The preliminary results of retrieval speeds are summarized in Figure 3.

5.3 Benefits of Layering in the Prototype

At this point, the prototype is outperformed in certain areas, but it successfully implements one of

[1] https://github.com/microsoft/diskspd

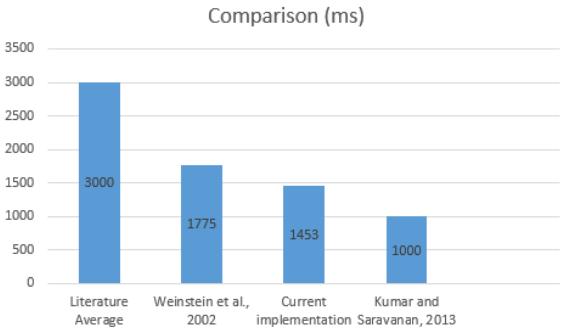

Figure 3: Priliminary measurements on retrieval speeds.

the main ideas related to a layered architecture: it uses exchangeable components and the implementation of these components are completely invisible to the other component due to network communication. This is also supported by how the two implementations of facial authentication are able to use the very same data sent by the Android client despite their completely different approach to the problem. With having similar architectures for different approaches, these solutions resemble two separate implementations of the same interface rather than different ML problem solvers.

This also means that unlike conventional approaches, in which the whole of the system has to be reconfigured for functional changes, functionality in either side of the system is arbitrarily modifiable. Although this is shadowed by how uncommon the 131 landmark point implementation is making input size changes necessary for other implementations, both ends of the system can be changed with little effort needed on the other half. Adding additional functionality to the server side, facial landmark-based lie detection (Upadhyay and Roy,) or emotion analysis (Day, 2016) for instance, does not require any changes on client side. In fact, the client is completely oblivious to any kind of changes on server side as long as the common way of communication is upheld, contrary to current approaches in the literature that require retraining in the entirety of the process. In this manner, improvements can also be made on both sides of the system, an implementation using 64 or even 5 landmark points can be used on the client, while more complex ML architectures can be integrated into the server. In the general sense, this not only means that a complete retraining of the architecture is unnecessary, but certain sub-elements can also be added incrementally to it, a great advantage over “ordinary” approaches in terms of productivity.

6 CONCLUSIONS

We investigated the idea of extending layering to machine learning solutions by first establishing the fundamental principles of such a design and then, using a prototype system, we compared it to techniques and implementations focusing on facial recognition.

Even though our system was intentionally designed to target the lower end of its potential, we successfully demonstrated that facial recognition problems can be solved using a layered approach. We also demonstrated that even our simplistic system is capable of connecting the benefits of layering to the already existing benefits of current approaches. We achieved this by building two interchangeable Neural Nets on the server that both use the same landmark point approach to facial recognition as an input. We can conclude that even though the current prototype implementation has much to improve at this stage, preliminary findings suggest that the benefits inherent to the original systems can be integrated into a layered approach with the addition of time-proven success factors, which contemporary solutions are unable to provide.

Although producing promising preliminary results, the concept of layered approach to machine learning problems requires a great deal of further research before real-world applications can come in focus. For example, one of the most challenging aspects of software design is how the level of complexity rises with each piece of software component added to a system. Comprised of already intricate inner mechanisms, stacking machine learning solutions might prove too complex to economically scale. Another question coming to mind is how reusable can machine learning subcomponents really be when the system that uses them requires a certain “common denominator”, an interface, such as the use of facial landmark points, to work. Such questions need to be answered before attempts at large-scale application can start.

ACKNOWLEDGMENT

The authors would like to thank both the GPGPU Programming Research Group of Óbuda University and the Hungarian National Talent Program (NTP-HHTDK-20) for their valuable support.

REFERENCES

Aydin, I. and Othman, N. A. (2017). A new iot combined face detection of people by using computer vision for security application. In *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, pages 1–6. IEEE.

BenAbdelkader, C. and Griffin, P. A. (2005). Comparing and combining depth and texture cues for face recognition. *Image and Vision Computing*, 23(3):339–352.

Chaczko, Z., Klempous, R., Rozenblit, J., Adegbija, T., Chiu, C., Kluwak, K., and Smutnicki, C. (2020). Biomimetic middleware design principles for iot infrastructures. *Acta Polytechnica Hungarica*, 17(5).

Day, M. (2016). Exploiting facial landmarks for emotion recognition in the wild. *arXiv preprint arXiv:1603.09129*.

Fowler, M. (2012). *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch*. Addison-Wesley.

Hazen, T. J., Weinstein, E., and Park, A. (2003). Towards robust person recognition on handheld devices using face and speaker identification technologies. In *Proceedings of the 5th international conference on Multimodal interfaces*, pages 289–292.

Karnila, S., Irianto, S., and Kurniawan, R. (2019). Face recognition using content based image retrieval for intelligent security. *International Journal of Advanced Engineering Research and Science*, 6(1).

Kumar, A. R. and Saravanan, D. (2013). Content based image retrieval using color histogram. *International journal of computer science and information technologies*, 4(2):242–245.

Liu, Y., Zhang, D., Lu, G., and Ma, W.-Y. (2007). A survey of content-based image retrieval with high-level semantics. *Pattern recognition*, 40(1):262–282.

Manjunath, B., Chellappa, R., and von der Malsburg, C. (1992). A feature based approach to face recognition. In *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 373–378. IEEE.

Miklos, P., Zeljen, T., and Tatjana, L.-T. (2020). Analysis and improvement of jpeg compression performance using custom quantization and block boundary classifications. *Acta Polytechnica Hungarica*, 17(6):171–191.

Rodriguez, Y. and Marcel, S. (2006). Face authentication using adapted local binary pattern histograms. In *European Conference on Computer Vision*, pages 321–332. Springer.

Sommerville, I. (2011). Software engineering 9th edition. *ISBN-10*, 137035152:18.

Upadhyay, T. and Roy, D. Lie detection based on human facial gestures.

Walker, K. N., Cootes, T. F., and Taylor, C. J. (1998). Locating salient object features. In *BMVC*, volume 98, pages 557–566. Citeseer.

Weinstein, E., Ho, P., Heisele, B., Poggio, T., Steele, K., and Agarwal, A. (2002). Handheld face identification technology in a pervasive computing environment. In *Short Paper Proceedings, Pervasive 2002*, pages 48–54.

Wu, Y., Hassner, T., Kim, K., Medioni, G., and Natarajan, P. (2017). Facial landmark detection with tweaked convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):3067–3074.

Wu, Y. and Ji, Q. (2015). Robust facial landmark detection under significant head poses and occlusion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3658–3666.