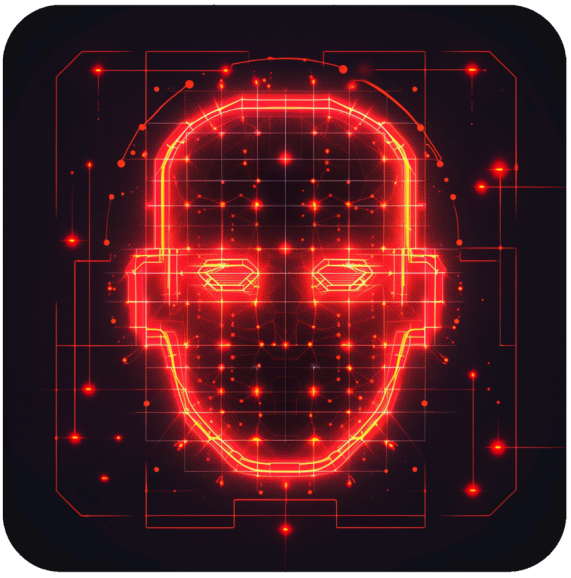
Léo LE CORRE
Hugo KINDEL
Rêzan OZCAN
Séverine BENIER

Promotion 2024

Artificial Intelligence, Deep Learning and Applications

Project: VaultProtect



Summary

Introduction

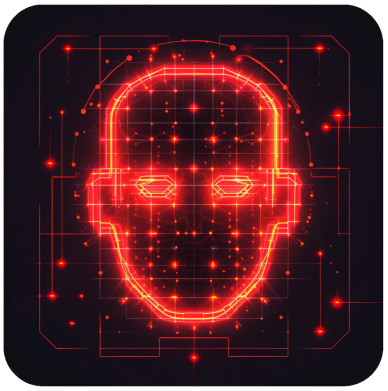
To conclude the "Artificial Intelligence, Deep Learning and Applications" module, we are asked to carry out a project. We are a group of 4 students and chose to do a project on the theme of **facial recognition** with a project we decided to call VaultProtect.

We thought it was an interesting topic as we are all from cybersecurity backgrounds. Facial recognition is used both in security measures (like double authentication with biometric solutions) or information security with a lot of safety to be taken into account when dealing with this data that is generally understood as personal data and thus require special care in, for example, GDPR.

Link to the GitHub repository containing the whole project (model and parts of the dataset included): https://github.com/hugokindel/vaultprotect

Our Project: VaultProtect

VaultProtect is supposed to be a "state-of-the-art" facial recognition system designed to provide our customers with secure access control thanks to cutting-edge artificial intelligence.



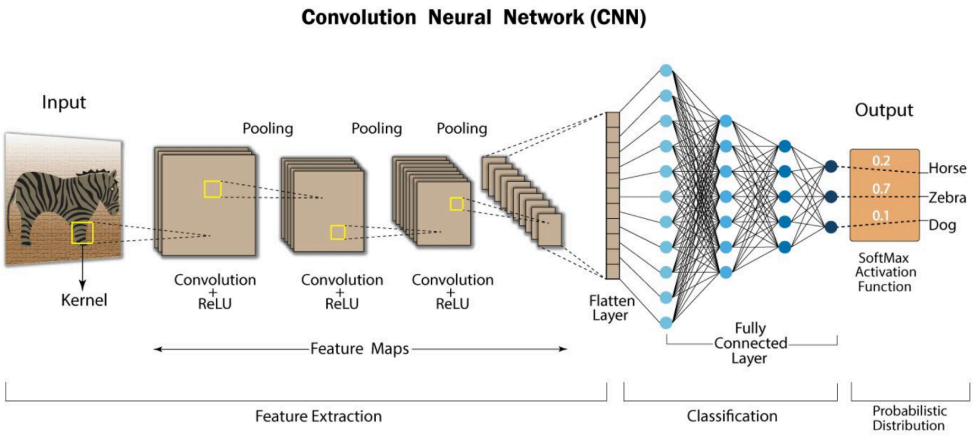
The core of this system relies on **Convolutional Neural Networks (CNNs)**, a class of deep neural networks particularly effective in image processing tasks.

One prominent architecture used in VaultProtect is **VGG-16**. Others have been tried like VGG-19, ResNet50 or ResNet152. But the results were either subpar or similar and so, will not be vastly discussed in this document, only indicated here for reference.

Of course, we know the final solution will not be secure nor functional enough to be used in a production environment. The idea is just to learn and try out model training and usage with this simple use case.

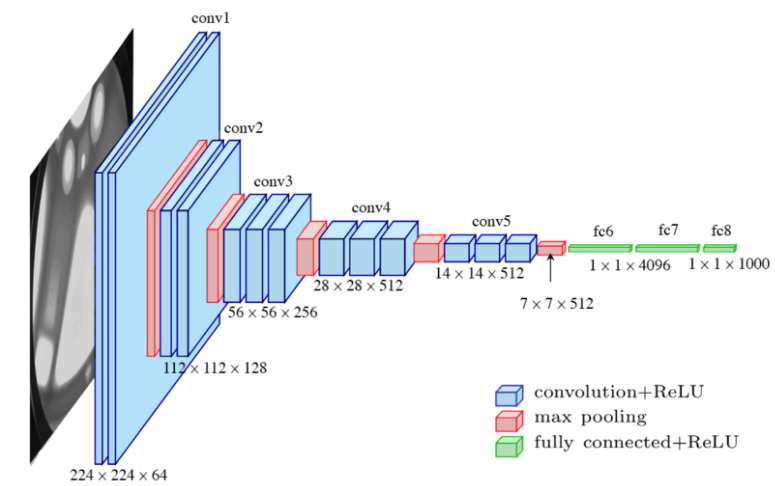
CNNs in Artificial Intelligence

Convolutional Neural Networks (CNNs) are a class of deep neural networks specifically designed for tasks involving images and visual data. They excel in learning hierarchical features and patterns through convolutional layers, which apply filters to input data to detect spatial hierarchies of features. This makes CNNs particularly suitable for image recognition, object detection, and facial recognition tasks for ProtectVault.



VGG-16: Our Base Model

VGG-16, short for Visual Geometry Group 16, is a convolutional neural network architecture developed by the Visual Geometry Group at the University of Oxford. It gained prominence for its simplicity and effectiveness in image classification tasks. The architecture consists of 16 weight layers, including 13 convolutional layers and 3 fully connected layers. The use of small receptive fields (3x3) for convolution, along with max-pooling layers, allows VGG-16 to capture intricate patterns in images and achieve impressive accuracy.



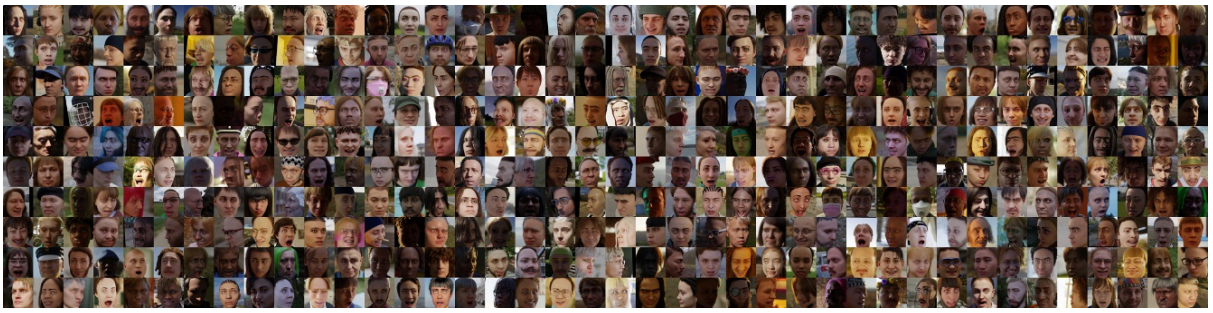
Datasets: Training with LFW

VaultProtect was first tried with the Labeled Faces in the Wild (LFW) dataset using the VGG-16 model. This dataset, with over 13,000 labeled facial images, serves as a benchmark. VGG-16 learns facial features by processing batches of LFW images, adjusting its weights to minimize prediction errors. This ensures VaultProtect generalizes well to diverse faces and real-world scenarios, making it a reliable and secure solution for access control. In reality, it didn't achieve as good as we would have liked, and the results weren't the most promising. Maybe it was due to the quality of the pictures or the fact that the data is mostly imbalanced due to the differentiating number of pictures per face contained. Some, like George W. Bush, can have more than 500 pictures while others less than 50 or even only 1 picture.

Datasets: Training with DigiFace-1M

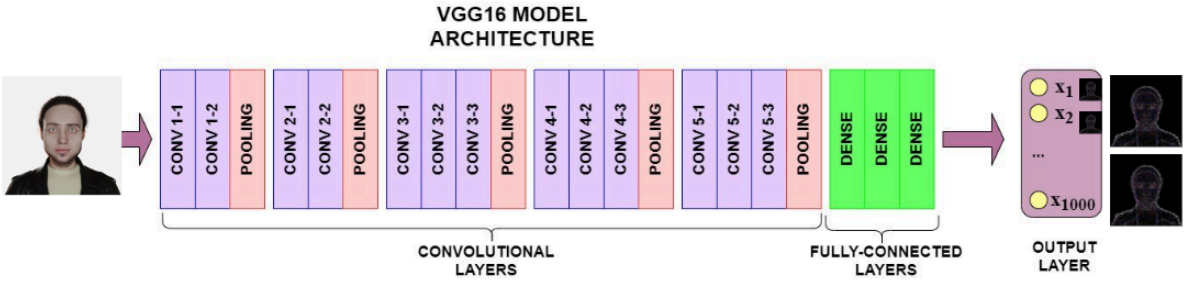
VaultProtect achieves high accuracy through training with the DigiFace-1M from Microsoft, using the same VGG-16 base model. This dataset, with over 720,000 labeled facial images, was used in the same way we previously did with LFW. Although we only used a small subset of this, around 50,000 images in total, due the processing power and time

that would be needed to drive training from too many images and the fact that the training would probably need a way more deep model for complex details.



Architecture of Our Proposed Solution

Our architecture therefore aims to use the VGG-16 model and train it with a few datasets containing approximately 10,000 facial photos each. This training phase will subsequently make it possible to prepare the validation phase and determine the precision of the model. Below is a vision of our VGG 16 architecture:

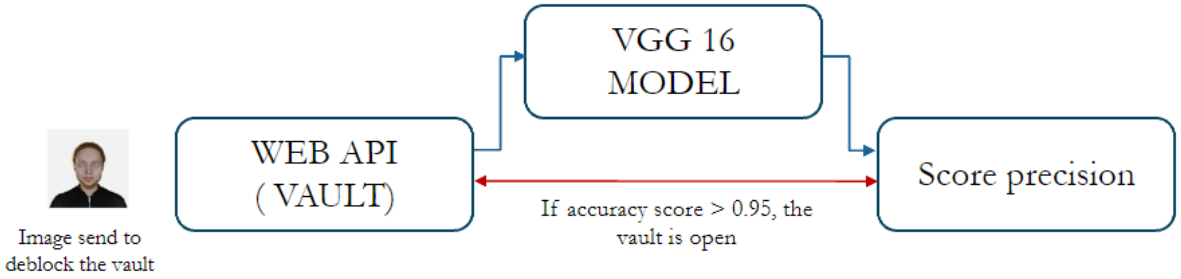


During model training on the VGG 16 model, an RGB image upscaled to size 224 x 224 from the dataset is inserted into the first convolution layer. For all convolution layers, the convolution kernel is of size 3×3 which is the smallest capture dimension, the goal being to filter the image and only keep discriminating information such as atypical geometric shapes.

These convolution layers are accompanied by Pooling layers, each of size 2×2, to reduce the size of the filters during training. So we can see one of the results on the right side after the process.

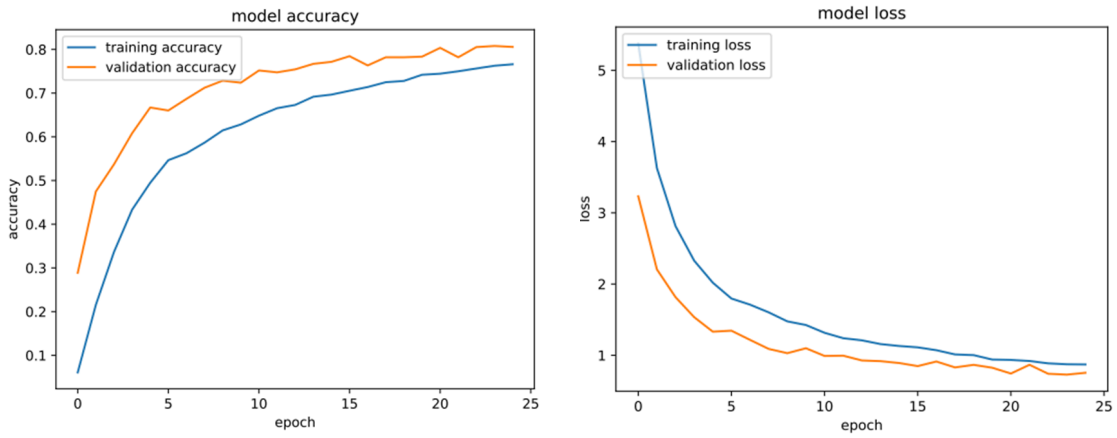
We can therefore see below a succinct explanation of our project, we therefore have an API allowing us to verify the facial identity thanks to an image which is inserted if the face corresponds then the vault is open otherwise it remains blocked, for this the The API receives in INPUT an image of the model which will give us a precision score according to

the difference vector which will allow us to verify the identity of the person requesting access to the vault, if the precision score is deemed sufficient then the chest will be opened.



Description of the ML/DL Training and Inference Pipeline

To show the model performances obtained during and after training, we need to understand each parameter that the following curves will use: The "model accuracy" schema illustrates the accuracy of the model per iteration (epochs). Also, there are two different curves, one for the training accuracy and the second for validating the model training allowing the algorithm to correct this one in case of overfitting or underfitting. Similarly to the "model accuracy", the "model loss" schema is complementary to the first one but instead of showing the accuracy the main goal is to illustrate the loss is constantly diminishing.



As explained previously the VGG-16 model is trained on a sample of 10, 000 images. For these schemas, the program ran for 45 minutes with only 25 epochs used to show the fast capacity of VGG-16 model's learning.

However, for the final version It'd been trained on 500 epochs using multiple training rounds with various parts of the DigiFace-1M dataset and the program execution took 15 hours to obtain a test accuracy as high as 90%, but not more.

Furthermore, to prepare the VGG-16 model we used our personal computer's GPU instead of Google Collab's GPU: it was mostly trained on an NVIDIA GeForce RTX 4080 and an Apple M1 Pro.

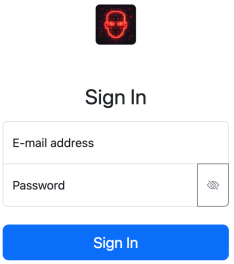
Afterwards, focusing on the content, the two different curves on each schema are close to each other which means the training method is optimized to avoid overfitting and underfitting. The result could be better if we had more computing power and used the entirety of the many images in the project dataset and if the model parameters were optimized with higher values like the number of images propagated into the neuronal network (batch_size) and the total iteration applied (epochs).

The Web Application and API

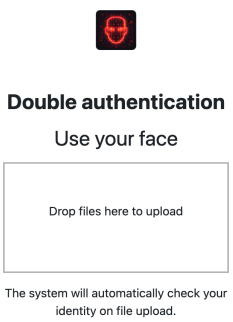
The model training application has been made in Python using Tensorflow and Keras as a machine learning framework. Due to these constraints, the web application has also been developed in Python to simplify the overall architecture and avoid tech debt. As such, we have chosen Python Flask as our go-to framework for the web application (for both frontend and backend with routing between the two).

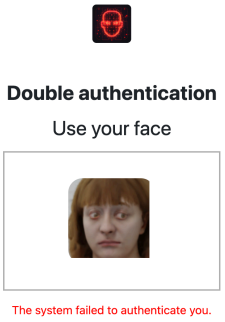
We have combined the two in a single Python application that can be used for both training the model and showcasing it using the web application. It's made to be a simple to run sample program, which means we decided to not include any database or "real world" development environment related to a web application. Due to these decisions, it's really easy to run and tweak the program by only installing the necessary packages (as included in the "requirements.txt" file). It means that there is no real user data and no data validation, the web application is only meant to showcase that the model can work as a face recognition solution (even though it's not secure enough to be used in a real scenario).

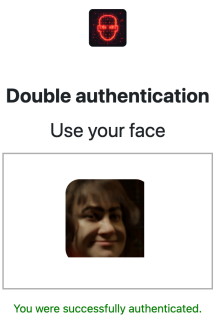
The index of the web application shows a simple sign-in sample page that can take any data as an input:



Then, you are redirected to the double authentication sample page that will take an image of someone's face as input and verify that it matches with the sample user id (4044):







After dropping in the image to verify, the frontend will send a request to the API (/api/v1/authenticate_face) that will return a JSON response indicating if the image matches with the sample user id (4044) label known to the model.

Conclusion

In conclusion, VaultProtect leverages the power of VGG-16, a robust CNN architecture, to achieve high-performance facial recognition. The training process with the LFW dataset enhances the model's ability to generalize and accurately identify faces, making VaultProtect a reliable and secure solution for access control.

The combination of advanced neural network architecture and a diverse training dataset positions VaultProtect as a nice proof of concept facial recognition system in the field of artificial intelligence.

While a lot of things can be improved :

- The system cannot be defined as secure, and there are way better face recognition algorithms in the wild, using machine learning as a part of their detection systems but not its entirety.
- More computing power and more time could permit a better model, especially combined with more data and maybe more layers to handle complex or edge cases.
- The web application and API is not an enterprise grade application. It's only made to be used as a sample and works well for this use case, but could be improved to make a real application.

We are still proud of the results we obtained with our model and the overall project. It works and shows promising results, even better than our expectations.