NOVA IMS Information Management School

Master's Degree in Data Science 23/24

Group 6

Deep Learning Project:

Eye Tracking

Collaborators:

- Peter Falterbaum (20230956)
- Jannik Himmelsbach (20230550)
- Luis Penteado (20230441)
- Alex de Figueiredo (r20201607)
- Fabian Collao (20230503)

Introduction

Eye tracking technology has revolutionized various fields, including human-computer interaction, psychology, and healthcare. The ability to accurately track eye movements has numerous applications, such as understanding human behavior, diagnosing neurological disorders, and enhancing user experience. This project aims to develop an accurate, and efficient eye tracking system using Convolutional Neural Networks (CNNs) over a pre-trained model.

Our system aims to address a significant real-world problem: the lack of accessible and affordable eye tracking technology. By developing a CNN-based eye tracking system, we can provide a cost-effective, and widely applicable solution for various applications, including:

- Virtual Reality: accurate eye tracking can enhance immersion by rendering high-quality visuals only in the user's field of view, thereby optimizing computational resources
- Marketing and advertising: understanding consumer behavior and preferences
- Education: developing personalized learning systems and adaptive educational tools

By developing an accurate and efficient eye tracking system using CNNs, we can unlock new possibilities for various industries and improve the lives of individuals worldwide.

Scope

The scope of this project is to design, develop, and evaluate a CNN-based eye tracking system that can accurately detect and track eye movements in real-time. We will explore the use of pre-trained models, specifically VGG16 and ResNET, and fine-tune them for our specific task, leveraging the dataset we created and annotated. The goal is to track the user's eyes, enabling real-world applications. To achieve this, we will focus on developing a fast and lightweight model, with low complexity, to ensure seamless performance.

Libraries Used (Python)

In this Python-based project, several 3rd party libraries came into play during implementation. An overview of those utilized is provided below:

- labelme: Annotation of images with ground truth labels
- opencv-python: Image processing tasks
- numpy: Handling of image data in form of arrays
- json: Parsing and handling the annotated data generated using labelme
- albumentations: Data augmentation
- tensorflow: Data loading, model development, model training and evaluation

Proposed Method

Data Collection

The initial step in developing the pursued eye tracking system involved data collection, forming the foundation for training the models. Each team member contributed a collection of images featuring diverse head poses, angles, backgrounds, lightning conditions and eye states (open/closed). After consolidating and standardizing these images into a common format, the dataset underwent random shuffling and partitioning into separate subsets for training (70%), validation (15%), and testing (15%). Subsequently, each image was annotated with coordinates representing key points on the eyes.
To increase the sample size, various data augmentation techniques, including random horizontal and vertical flipping, random cropping, brightness and color value adjustments, as well as random gamma value alterations, were applied to all original images.
Contrary to best practices, data augmentation was also applied to the validation and test sets. This approach was necessary due to the limited volume of original data available, ensuring that there was a sufficient amount of data to effectively evaluate the model's performance.

During the development of this eye tracking system, three distinct datasets were compiled. The initial dataset was assembled to generate first outcomes. A second dataset was then created, derived from the first but with several modifications intended to enhance model performance. These modifications included increasing the volume of augmented data and excluding images that had poor lighting conditions or low-quality representations of the eye area. The third dataset was an extension of the second one, characterized by the addition of numerous images of the same person, just to the training data. This approach was aimed at facilitating the model's training process while avoiding the risk of overfitting.

Data Preparation & Preprocessing

Before training the model, the data is prepared through a series of steps. Initially, training, validation and testing images are loaded into tensorflow datasets, ensuring a consistent ordering. Each image is then read and decoded from JPEG format, followed by resizing to a uniform dimension of 250x250 pixels. Subsequently, the pixel values are normalized by dividing each image by 255 to scale them within the range [0, 1]. Similarly, labels are loaded from JSON files, extracting the key points data from each file. The labels and their corresponding images are then combined into a single dataset, facilitating joint processing during training. Finally, the datasets are shuffled to randomize the order of samples, batched into groups of 128 for efficient processing, and prefetched to minimize data loading latency during model training.

Model Training

After preparing and processing the data, the project advanced to the model training phase, focusing on two well-known neural network architectures: ResNet and VGG. The models were chosen for their proven effectiveness in image analysis tasks, and were specifically adapted to meet the specific needs of eye tracking. Input were the different preprocessed training and

validation data sets. The output is a 4-element array, representing the coordinates of the points that define the eye location. For the training process an Adam optimizer was utilized.

Model Architecture

The project initially explored using the ResNet152V2, a pre-trained deep convolutional neural network. For the specific regression needs in eye tracking, the top layer was removed (include_top = False) to customize the network to handle inputs of 250x250 pixels across three color channels. The network included several convolutional layers with varying filter sizes and strides. However, due to better performance metrics of the VGG based model, the ResNet model was not chosen as the final model.

The project's final model was based on the pretrained VGG16 model, likely a variant of the VGG network, which is known for its simplicity and effectiveness in image recognition tasks. The model benefits from learned features such as edges, textures, and gradients, which are useful for higher-level feature extraction specific to eye detection.
In order to avoid customization capability for the specific need of the regression problem, it was included without its top layer (include_top = False). The input shape was defined according to our labeled pictures dataset with a shape 250x250 pixels by three channels for the colors, resulting in (250, 250, 3).
The architecture also incorporates convolutional layers combined with max pooling layers. Starting with four Conv2D with 96 filters of size 7x7, 128 filters of 5x5 size, 256 filters of 3x3 size and 512 filters with 2x2 size, all of them with ReLU (rectified linear unit) activation. They were used to give the model more capacity to learn complex features. Max pooling was also added between some of the convolutional layers to reduce the spatial dimensions, decreasing the computational complexity.
A Flatten layer is used to transform the 3D feature maps into 1D feature vectors, preparing the data for the fully connected dense layers. Aiming to capture the non-linear relationships necessary for predicting precise output values for eye detection, three dense layers (512, 128 and 32 units) with ReLU activation function were added to reduce dimensionality.
Dropout layers following some of the dense layers are crucial in preventing overfitting by randomly setting a fraction of input units to 0 during training, which enhances the generalization of the model. The final dense layer with 4 units and 'linear' activation function produces the regression outputs, typically representing normalized coordinates concerning the eye's position and size in the image.

Hyperparameter Tuning

To further optimize the model's performance, a random search was conducted on the VGG16-based model. This involved experimenting with various settings for learning rates, dropout rates and unfreezing some of the layers to allow the model to learn more effectively. This approach enabled us to achieve the best results in our experiments.

The summary of the final model architecture can be found in the *Annex* [1].

Evaluation

We assessed the performance of our model using two primary metrics: root mean squared error (RMSE) and visual inspection. The RMSE provided a quantitative measure of the model's accuracy in predicting eye coordinates, while visual inspection involved overlaying predictions on input images to evaluate the model's ability to detect eye locations. Due to the variability in hardware across our team, we did not consider training and inference times as reliable performance indicators. Although Mean Average Precision (MAP) and Intersection over Union (IoU) could have offered additional insights, resource constraints limited their inclusion.

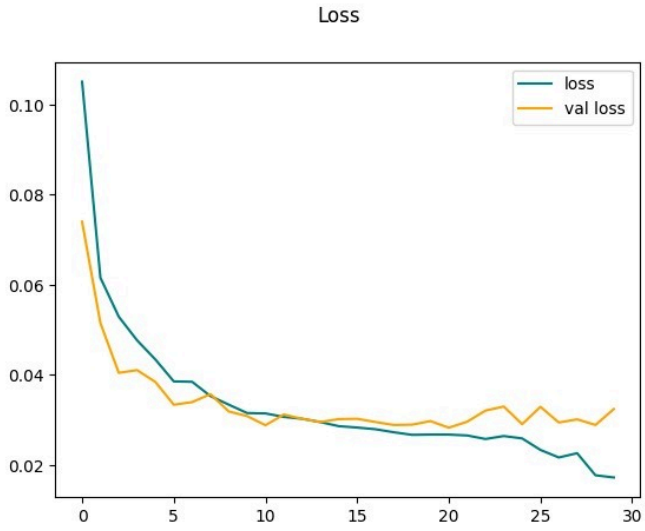
Despite these limitations, the RMSE and visual inspection provided valuable insights into the model's performance. Future work should incorporate MAP and IoU metrics to provide a more comprehensive assessment of the model's object detection and localization capabilities.

Final Results

This report presents the outcomes of a deep learning project aimed at automating eye tracking and labeling within images. We trained a convolutional neural network (CNN) using a dataset that used a generalized set of images from our 5 group members that included images of the 5 of us, different backgrounds and images with and without glasses. Eye tracking is crucial in many fields such as facial recognition, VR/AR and market studies but manually labeling eye positions is time-consuming. So, we developed a model using Keras and TensorFlow to automatically detect eyes. These tools, Tensorflow and Keras served as flexible and scalable tools to build the models. Using TensorFlow, we were able to implement neural network architectures and execute computations efficiently. With its library of pre-built functions and modules helping to streamline the development process. Meanwhile Keras made it easy to create neural networks without worrying about technical details. Its interface allowed us to try out ideas quickly. Together with TensorFlow, these tools really aided us in building our models.

As for the final models we considered and tested multiple different datasets and models, and manually inspected our results. Our third dataset as mentioned above returned us the best results being more generalized, larger and more consistent for the model to learn from. In tuning our final model we also took advantage of a random search to identify the most effective and optimized hyperparameter ranges across multiple iterations, this made a considerable improvement to our results compared to our initial results.

The VGG-based model achieved the best accuracy and performance after 30 epochs of training. However, we observed that the results converged after the first 20 epochs, indicating no significant improvement in subsequent epochs.



The final results show an achieved training loss of 0.018 and validation loss of 0.03 implying that the model generalizes effectively to unseen data, as both losses are relatively low, indicating minimized prediction errors on both the training and validation datasets.
However, the slight overfitting towards the end, with the validation loss slightly higher than the training loss, suggests that the model might be memorizing the training data too closely. This could imply capturing noise or specific patterns present only in the training data, potentially limiting its performance on new, unseen data. Images with the practical labeled results can be seen in the *annex [2]*.

Further Steps

While our proposed eye tracking system could be improved, there are several avenues for future exploration and improvement.

Firstly, it would be interesting to further refine our model by increasing the size and diversity of our training dataset, as well as experimenting with different architectures and hyperparameters to improve accuracy and robustness. This could involve collecting more data, exploring alternative pre-trained models, and fine-tuning the model's hyperparameters.

A more ambitious direction for future work is to leverage our eye tracking system as a foundation for exploring attention mechanisms in humans. Inspired by recent research in automatic visual attention detection, we envision using our model as a starting point for developing a more comprehensive understanding of human attention. Specifically, we propose to develop methods to automatically detect and annotate what people are visually paying attention to in videos, specifically the areas of interest (AOIs) that attract their gaze. By automating this process, we can overcome the tedious manual annotation process that has traditionally hindered research in this area, and unlock new insights into human attention and behavior.

Bibliography

Krafka, K., Khosla, A., Kellnhofer, P., Kannan, H., Bhandarkar, S., Matusik, W., & Torralba, A. (2016). Eye Tracking for Everyone. arXiv preprint arXiv:1606.05814.

Rakhmatulin, I. (n.d.). Dataset for eye-tracking tasks. [Dataset]. South Ural State University, Department of Power Plants Networks and Systems.

Maryim Omran and Ebtesam N. AlShemmary (2020) An eye Recognition System Using Deep Convolutional Neural Network.

Ahmad, N., Yadav, K.S., Kirupakaran, A.M. *et al.* Design and development of an integrated approach towards detection and tracking of eye using deep learning. *Multimed Tools Appl* (2023). https://doi.org/10.1007/s11042-023-17433-z

Annexes

[1] Summary of the final model architecture.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 conv2d (Conv2D)             (None, 7, 7, 96)          2408544

 conv2d_1 (Conv2D)           (None, 7, 7, 128)         307328

 max_pooling2d (MaxPooling2  (None, 3, 3, 128)         0
 D)

 conv2d_2 (Conv2D)           (None, 3, 3, 256)         295168

 max_pooling2d_1 (MaxPoolin  (None, 1, 1, 256)         0
 g2D)

 conv2d_3 (Conv2D)           (None, 1, 1, 512)         524800

 max_pooling2d_2 (MaxPoolin  (None, 1, 1, 512)         0
 g2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 512)               262656

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 128)               65664

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 32)                4128

 dense_3 (Dense)             (None, 4)                 132

=================================================================
Total params: 18583108 (70.89 MB)
Trainable params: 6228228 (23.76 MB)
Non-trainable params: 12354880 (47.13 MB)
_________________________________________________________________
```

[2] Practical labeled final results