Eye Gaze Estimation

Francesco Milano
fmilano@student.ethz.ch

Constantin Stamatiadis
stamatic@student.ethz.ch

ABSTRACT

The goal of this project was to develop a CNN architecture to perform the task of estimating eye gaze direction from single-eye images obtained by preprocessing samples from the MPIIGaze dataset. Starting from a custom architecture we tried out other famous architectures such as VGG16 and DenseNet, eventually turning back to a custom-type inspired by DenseNet. The best results were obtained with the latter, with a mean squared eye-gaze error of around 0.00667 on the validation set (score 0.00689 on the public test set on Kaggle).

1 INTRODUCTION AND RELATED WORK

The task of eye gaze estimation is usually tackled with either a *model-based* approach or an *appearance-based* approach. In the former a geometric model of the eye is used and gaze direction is estimated through the detected eye landmarks. Appearance-based approaches, instead, directly use eye images as input.

In general, in both cases, a preprocessing step is needed to extract first the face from the entire image and then subsequently the eye(s).

For this project we were given a set of eye images, so this preprocessing step was not needed. Samples from both a real (MPIIGaze, [8, 10]) and a synthetic (UnityEyes, [7]) dataset were provided for training purposes, but the testing set consisted only of images from MPIIGaze. In particular, according to the given specifications, the training set and the testing set contained images respectively from 10 and 5 participants. The resulting preprocessed samples (i.e., with only a single eye-image extracted) were of low resolution (60x36 pixels). Furthermore, some of the sample images were very hard to extract features from (e.g. eye closed or almost closed, bad illumination conditions for the real dataset, etc.). Therefore, we decided to first use an appearance-based model, as a front-end of which we developed a CNN architecture.

As a first attempt we used a hand-crafted architecture simply consisting of convolution and fully-connected layers. Following the approach of Zhang et al. [10], that used 13 layers inherited from a 16-layer VGG network, we then implemented a similar network, fine-tuning its fully-connected part. While doing so, we both tried to load the weights pre-trained on ImageNet - letting only the fully-connected part of the network train - and to let the entire network train. In both cases, however, we did not obtain an improvement over the first model.

Afterwards, in order to better capture the structure in the images we turned to a DenseNet-based approach, in which we did not load any pre-trained weights. Through hyperparameter tuning we were able to improve our performance.

Finally, motivated by the limited amount of samples in the dataset and by the high complexity of the net structure we implemented an easier architecture. Therefore, we took our first network as the basic structure and added forward connections between the layers to integrate the idea of DenseNet. This gave us the best result, that we used for our final Kaggle submission.

2 CNN ARCHITECTURES

In this section a more detailed description of the architectures used is given, along with the motivations that led us to choosing them.

2.1 Convolution + fully-connected architecture

Our initial model uses a simple structure that consists of 9 convolution layers followed by 5 fully-connected layers. The detailed structure is the following:

- 2 convolution layers with kernel size 7, 64 filters and stride 1, followed by a max-pooling layer with pool size 2, stride 2;
- 3 convolution layers with kernel size 5, 128 filters and stride 1, followed by a max-pooling layer with pool size 2, stride 2;
- 4 convolution layers with kernel size 3, 256 filters and stride 1, followed by a max-pooling layer with pool size 2, stride 2;
- 5 fully-connected layers with 512, 256, 128, 60 and 2 hidden units respectively.

We also concatenate the *head-pose* after the 128-unit fully-connected layer. The activation function used is ReLU for all layers.

The input images are resized to size 30x18 pixels and normalized to $[-1,\ 1]$ intensities, from an initial single channel grayscale image.

2.2 VGG16

Using an approach similar to that of [10], we implement the 16-layer VGG architecture on top of the previous fully-connected structure.

Since VGG works with 3-channel images, we preprocess the images by duplicating the grayscale image into the two remaining channels. As a first step, we also use the suggested *transfer-learning* technique and load the VGG weights pretrained on ImageNet. To facilitate this process, we integrate Keras in our code and use Tensorflow as backend. However, as shown in section 3.2, the results are worse than the ones produced by the initial architecture and do not even pass the easy baseline.

In the final version of the VGG16-based architecture, we let the entire network be trained, but with no significant improvements over the previous results.

2.3 DenseNet

Due to the non-optimal performance of VGG, we turned to other widely-used architectures, looking in particular at DenseNet ([2]).

The main idea behind DenseNet is to introduce direct feed-forward connections from each layer to every other layer. Compared to other architectures such as ResNet ([1]) outputs are concatenated instead of being summed. For our purposes, this brings the advantage of reducing the number of parameters, therefore yielding an easier-to-train architecture, which is preferred for limited-size datasets as the one we are given. Also, the feed-forward connections between layers at different depths introduce correlations between feature maps of different level of abstraction. Therefore, this helps extracting features that are more informative of the entire dataset

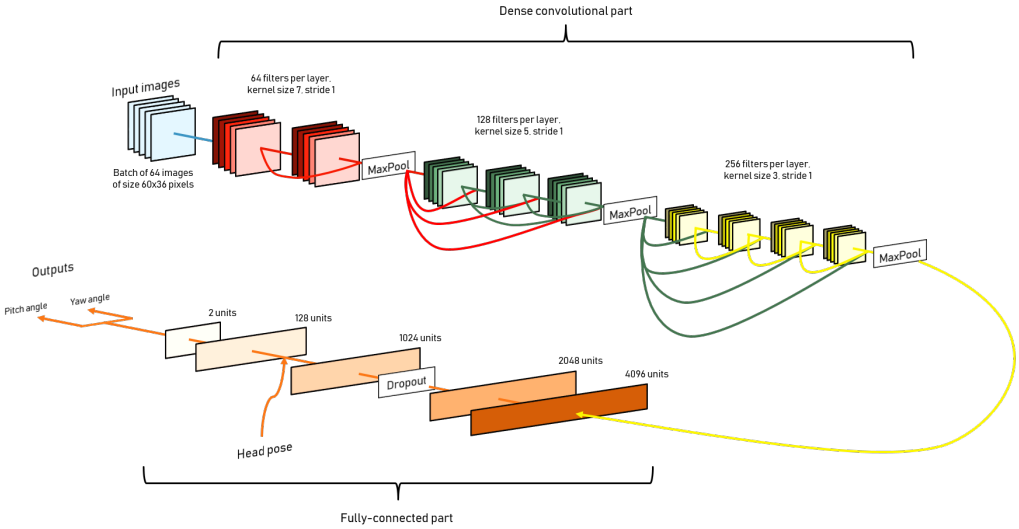

Figure 1: Scheme of the final architecture. Connections between lines indicate concatenation. ReLU is used as activation function for each layer, batch normalization is performed after each concatenation operation. Maxpool layers have pool size 2 and stride 2. Dropout rate is 0.2.

rather than single-image features. This is particularly suited for the low-resolution images that we are given.

The full architecture that we use consists of DenseNet121 on top of 3 fully-connected layers (2048, 512 and 2 units) with ReLU activation function. Head-pose information is concatenated after the 512-unit fully-connected layer. As in the VGG-based architecture, grayscale images need to duplicated on two more channels. To handle changes in illumination and in general different image qualities, we equalize the histogram of each sample and normalize the intensities to $[0, 1]$. In particular, this latter change produces a significant performance improvement. As a further means of regularization we insert a dropout layer with dropout rate of 0.2 between the first two fully-connected layers.

2.4 Final architecture

Following the intuition that a previous, rather simple architecture, still achieved an acceptable performance, we tried to test whether the complexity of our task was such that a shallower net could be preferred to deep and complex nets such as DenseNet.

As a final architecture we therefore take the same as the initial one and concatenate at the start of each convolution layer the outputs from all the previous layers. Of course, this only works with layers preceding a max-pooling layer, because these have the same dimension. This way, each of the interconnected layers can be thought of as resembling a *dense block* of a dense net.

The same normalization/preprocessing techniques as DenseNet are used, as well as dropout regularization. *Batch normalization* is also introduced after each concatenation.

To prefer broadness over depth we increase the number of hidden units per layer of the fully-connected part. In particular, 4096, 2048, 1024, 128 and 2 units are used, and concatenation with the head-pose information is done before the last fully-connected layer.

A scheme of the architecture is shown in fig. 1.

3 RESULTS

We now provide a more in depth comparison of the results that we obtained with the architectures described above, with some references to the hyperparameters used.

3.1 Convolution + fully-connected architecture

The initial network reaches a mean squared eye-gaze error of around 0.00831 on the validation set (score 0.00868 on Kaggle). We use the default learning rate taken from the skeleton code (1×10^{-3}) together with the other default values (i.e., batch size 64, Adam with the default values set by Tensorflow, 20 epochs). Trials with an increased number of epochs do not yield any better results. This can be most likely explained by the fact that we do not change the learning rate in any way throughout the training process.

3.2 VGG16

Using a more complex architecture based on VGG16 does not yield an improvement. As mentioned above, we use transfer-learning, to exploit weights pre-trained on ImageNet. However, this approach does not manage to provide good results, as the MSE is in the order of magnitude of 10^{-2} on the validation set. The score does not improve if the hyperparameters are changed and neither does when changing learning rate or when training over different epochs lengths. As a further attempt we modify the fully-connected part: we vary both width and depth, but without obtaining any significant improvements on the validation set.

We therefore attempt to train the entire network with our dataset, i.e., without importing pre-trained weights. This yields slightly better results, which are close to the score required to beat the easy baseline. The latter is almost perfectly matched by one of our submissions, that gives a a validation error of 0.00905 by training over 32 epochs, with batch size 64 and learning rate of 1×10^{-5}.

Also in this case changes to the hyperparameters do not provide better results. For both trials we use Adam as optimizer with the parameters specified by [10].

3.3 DenseNet

DenseNet allows us to further improve our results. We try both DenseNet121 and DenseNet169: the former provides better results, which confirms our intuition that a shallower net is better suited for this task. The first positive result we obtain is 0.00735 on the validation set (0.00796 on Kaggle), by training the network for 5000 steps with batch size 128, with an initial learning rate of 1×10^{-4} which is halved every 500 iterations. Reducing the batch size to 64 and increasing the number of steps to 700 yields a slight improvement, with a validation set error of 0.00711 (0.00762 on Kaggle). Limiting the learning rate to not decay below 1×10^{-7} also slightly improves the validation error, which reaches 0.00687 (0.00733 on Kaggle). Since reducing the batch size from 128 to 64 yielded an advancement, we also try to further reduce it to 32. This however does not result in any gain in performance, as the validation error is slightly above the one of our best result.

3.4 Final architecture

Our own neural network with feedforward connections between layers in DenseNet style is the one that provides the best results. The difference with simply using DenseNet121 is not very big, but we prefer this architecture because it is shallower. The network is trained with the same parameters as the DenseNet-based one that produced the best results. This yields a validation error of 0.00727 (0.00711 on Kaggle). Changing the learning rate every 1000 steps rather than every 500 reduces the validation error to 0.00635 (0.00693 on Kaggle). However, especially the second result is characterized by big fluctuations, whereas the first one had a smoother convergence. For our final submission, we try to tune the learning rate in such a way that the convergence is smooth. After several trials we end up using an initial learning rate of 2.5×10^{-4}, multiplied by a factor of 0.42 every 700 steps for a total of 8500 training steps. This approach yields a validation error of 0.00667 (0.00689 on Kaggle), which is slightly larger than our previous one, but with a smoother convergence, as can be seen in fig. 2.

4 OTHER APPROACHES TRIED AND FURTHER WORK

Apart from the architectures described above, we also reasoned about other possible approaches and started implementing them. However, these attempts did not lead to fully-working networks or Kaggle-submittable results.

We mainly focused on two alternative approaches:

- a *model-based* approach;
- an architecture including an *encoder-decoder* structure.

These two attempts are described below.

4.1 Model-based approach

Following the reasoning presented in [4], we thought that extracting relevant features of the eye, such as the shape, and feeding these features, instead of the whole image, to the CNN structure could result in an improvement in performance. However, several difficulties arose when trying to practically implement this approach:

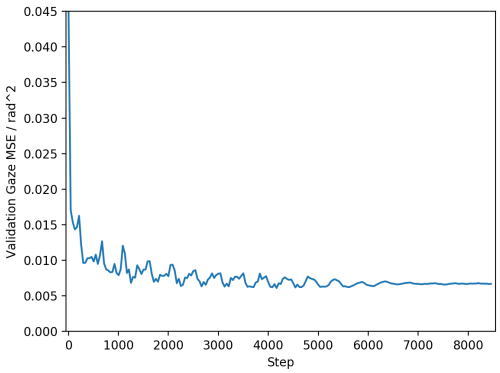

Figure 2: Gaze mean squared error on the validation set for the final architecture. An adaptive learning rate was used, with an initial value of 2.5×10^{-4} and a decay factor of 0.42 applied every 700 steps.

- For good accuracy training needed to be performed on a synthetic dataset, which required a very long training time, while the time for VM training was limited, so we preferred to use it for experiments that we were more confident with;
- Some details of the model implementation were too complex to ensure that a good result would have been reached in reasonable time;
- Extracting features from eyes (e.g., eyeball, eye shape) required to write a classification model from scratch and train it on a dataset. We tried to investigate other previous attempts on the task but were only able to find means to extract eyes from the image of a face, rather than extracting eye features from images of an eye.

4.2 Structure with encoder-decoder

As previously mentioned the image resolution represents a big limitation. To cope with it we reasoned about the possibility of extracting the most relevant features from images, in a way that the learned model could be as less sensitive as possible to the small variations in the general eye appearance from one image to another. This led us to considering an encoder-decoder structure as a possible improvement, to be inserted as the front-end of the CNN. In particular, the idea is to let an encoder-decoder architecture train on the dataset, so as to minimize the loss measure between the input and the output image, and in a subsequent step take the output from the encoder as the input to the CNN. It is reasonable to assume that the features learned at the output of the encoder are the ones that characterize the main differences between the eye images.

Unfortunately, we came up with this idea late and could not make our implementation of it work.

5 CONCLUSIONS

In this project we explored different neural network architectures for eye gaze estimation, ranging from well known ones, such as VGG16 and DenseNet, to shallower custom-built networks. The latter proved to be better for the given task. Moreover, we also explored what could possibly be done to obtain further improvements.

REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[2] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[3] Kyle Krafka, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra Bhandarkar, Wojciech Matusik, and Antonio Torralba. 2016. Eye tracking for everyone. *arXiv preprint arXiv:1606.05814* (2016).

[4] Seonwook Park, Xucong Zhang, Andreas Bulling, and Otmar Hilliges. 2018. Learning to Find Eye Region Landmarks for Remote Gaze Estimation in Unconstrained Settings. In *ACM Symposium on Eye Tracking Research and Applications (ETRA) (ETRA '18)*. ACM, New York, NY, USA.

[5] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russell Webb. 2016. Learning from Simulated and Unsupervised Images through Adversarial Training. *CoRR* abs/1612.07828 (2016). http://dblp.uni-trier.de/db/journals/corr/corr1612.html#ShrivastavaPTSW16

[6] K. Simonyan and A. Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014).

[7] Erroll Wood, Tadas Baltrusaitis, Louis-Philippe Morency, Peter Robinson, and Andreas Bulling. 2016. Learning an appearance-based gaze estimator from one million synthesised images. In *Proc. of the 9th ACM International Symposium on Eye Tracking Research & Applications (ETRA 2016)*. 131–138. https://doi.org/10.1145/2857491.2857492

[8] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. 2015. Appearance-based gaze estimation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4511–4520.

[9] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. 2017. It's Written All Over Your Face: Full-Face Appearance-Based Gaze Estimation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2017-05-18). 2299–2308. https://doi.org/10.1109/CVPRW.2017.284

[10] Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. 2018. MPIIGaze: Real-World Dataset and Deep Appearance-Based Gaze Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2018). https://doi.org/10.1109/TPAMI.2017.2778103