Online-Meetings: Real-Time Attention And Emotion Analysis

Dominik
Philipp Althaus
Amon Soares de Souza

Abstract We introduce a model to analyze attention and emotion of participants during an online meeting

1 Introduction

During the pandemic situation starting in 2020 many companies and universities started using applications to conduct meetings or lectures online. In case of in person team meetings or lectures the speaker and participants usually have a good intuition about the atmosphere as they can see each other and thus estimate the general mood. However, during an online-meeting this is not that easy and often impossible as videotelephony programs such as zoom only show a small number of participants on the screen. For this reason we introduce an application which is capable of analyzing attention and facial expression of users. It summarizes the emotions and attention scores for a meeting and visualizes them in order for everyone to get an accurate feeling about the general mood during an online-meeting.

Facial expressions are one of the most important components in human interaction and communication. The ability to recognize and correctly classify facial expressions is still an active field of research. Due to the varying conditions, e.g. different illumination, head pose or occlusion, many models are not able to achieve good results. With the advent of convolutional neural networks, deep learning architectures were able to achieve considerable results. We present a facial expression recognition (FER) model which is based on modern CNN architectures and was able to achieve an accuracy of 0.739. As many modern tools are not able to achieve better results we use this model to classify emotions and apply it to our app. Last but not least we also developed a tool that analyzes the attention of a person by measuring whether a person is looking into the camera, i.e. the screen and how open his eyes are.

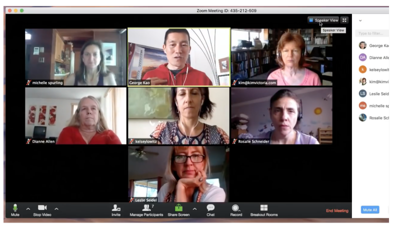


Figure 1: Exemplary screen during an online meeting, in this case zoom.

2 Screenshot Model

This would be the first step in our model. The process starts by taking screenshots of a webcam or video (if specified as input) and transforming those videos into a tensor representation, in order to be processable by our models. The initial thought was to use the package presented in PyImageGui. However, due to it's lack of speed and it's tendency to be better integrated into PILLOW, we discarded that Idea. The first class we originally designed was a screenshot class based on the module mss. It's speed was superior to that screenshot option integrated into the cv2 module, however it returned the tensors with the axes arranged in RGB instead of BRG, which is the used arrangement in cv2. So we settled on using the integrated cv2 methods, as we decided to use a model structure where the use of cv2 integrated functions was prevalent.

The screenshot model, which became the main function, uses a while loop to return frames as soon as they are processable, or in specific intervals, which will later come in handy to make the model more performant. Screenshots in intervals are also meaningful on an intuitive level, as attention swings do not occur in milliseconds.

3 Real-Time Attention Plotting

While emotional states are hard to interpret and keep track of while a meeting is happening, having a simple and user-friendly interface to track the attention of all meeting participants is useful, especially if you are the speaker. According to our thesis, while individual lack of attention might be due to personal circumstances, taking an attention score averaged over all listeners into account could give one valuable insight of how you perform as a speaker and presenter. Confronted with a low average, you could adjust your presentation style accordingly, to either slow down and try to explain some concepts more thoroughly, or to speed up a bit because you were going too much into detail for concepts already familiar to your listeners. For our attention score, we chose the standard arithmetic mean.

let $x_1, ..., x_n$ attention scores for listeners 1-n

$$\mu = \frac{1}{n}\Sigma_i^n x_i \tag{1}$$

The obvious choice here was to use the out-of-the-box power of matplotlib for data visualization. However, we noticed that real-time graph updates were rather slow and resource consuming. We decided to use PyQt5 and PyQTGraph to have fast real-time graphing functionality and also the liberty to design a desktop app according to our visions. Since you build an app from scratch, we had total freedom to design the app in a modular way and use the freedom of designing with CSS by using PyQT5's integrated QSS styling language. Our app consists of a single bar chart, which gives you a clue of the attention level of the participants momentarily, and also a graph, which will plot attention averages over time periods, and gives insight of how attention decreased or increased during the last minutes.

4 Attention Model

For our attention model we tried many things. Firstly, we tried to use cv2-integrated models and discard facial landmark structures introduced by the dlib model. This was tried because we thought of the landmark construction as being too costly to use in a while loop with all the complexity of classifying convolutional neural nets and real-time plotting layered on top. However, the models integrated, like 'haarcascade-frontalface-default', and it's corresponding eye model, were too inaccurate to use, especially with the outlook of having several faces on an image, with differing in image and lighting quality. So we integrated the dlib model for landmark construction.

The attention score uses two mechanisms to determine one's attention. The first is the eye aspect ratio (short EAR) the other is a mechanism which determines the direction a person is looking at by looking at the location of the pupil in relation to the eye landmarks.

The EAR is using spatial measures to calculate the distance of the upper and lower eye landmarks.

let $p_1, ..., p_n$ eye landmarks for one eye

$$EAR = \frac{\| p2 - p6 \| + \| p3 - p5 \|}{2 \| p1 - p4 \|} \tag{2}$$

This is used as a measure for tiredness, as nearly-closed eyelids are usually an indicator for tiredness, whereas widely opened eyes indicate that the participant is paying attention. It turns out, however, that the eye aspect ratio is not enough, since a participant could be wide awake, paying attention, but paying attention to something else. Especially if the person is looking up, the eyelids from the camera's perspective are even further apart than if the person was looking at the computer screen, resulting in a high EAR score.

That is where the mechanism comes into play that keeps track of the direction of a participants gaze. The GazeTracker module was originally taken from the repository of Antoine Lame*, and thoroughly adjusted in order to enable the detection of multiple faces on the analyzed screen, and avoid the loading of a second model, which would result in the same model being loaded for every face. The mechanism calculates a vertical pupil-eyelid ratio and a horizontal pupil-eyelid ratio to determine where a participant's gaze is looking at. It uses the following formulas:

let $VR(x)$ be vertical ratio of an eye
let $HR(x)$ be horizontal ratio of an eye
let $MP(x, y)$ be the middle point between two points

$$MP(x, y) = \frac{x + y}{2} \tag{3}$$

MP will be used to calculate the centroid of the isolated pupil, as well as the center point of the eye.

let $L_H(x)$ be the horizontal location of the pupil, where x is an eye

let $L_V(x)$ be the vertical location of the pupil, where x is an eye
let $DIST_H(x)$ be the horizontal ratio of the pupil location and the eye centroid
let $DIST_V(x)$ be the vertical ratio of the pupil location and the eye centroid
let $p_1, ..., p_n$ eye landmarks for eye x

$$DIST_H(x) = \frac{L_H(x)}{2MP(p_1, .., p_n) - 10} \tag{4}$$

$$DIST_V(x) = \frac{L_V(x)}{2MP(p_1, .., p_n) - 10} \tag{5}$$

let x, y be left, right eye respectively

$$HR(x, y) = \frac{DIST_H(x) + DIST_H(y)}{2} \tag{6}$$

$$VR(x, y) = \frac{DIST_V(x) + DIST_V(y)}{2} \tag{7}$$

The ratios are then used to determine the gaze. The result will be a scalar $x = [0, 1]$. We then use heuristic thresholds to determine when the algorithm classifies a gaze to go up, be centered or left/right. For the horizontal ratio, if that ratio turned out to be below 0.35, the participant looks left. If it's 0.35 ¡ x ¡ 0.75, the ratio will be classified as centered. For the vertical model it's similar, with the threshold being 0.35 ¡ x ¡ 0.75, it is centered, below that, the participant is looking up, above that, the participant is looking down.

5 Emotion Classification

The emotion derived from the facial expression will also be classified. Even though this could theoretically be done by any classification algorithm. many machine learning algorithms like Support Vector Machines or k-nearest-neighbor classifiers will fail due to the complexity of image classification. Images usually consist of a number of pixels divided into 3 color channels. For a high-definition image this results in over 6 million inputs per image. Even for a very low resolution image, e.g. 48×48 there are about 7,000 inputs. Deep learning algorithms are generally able to deal with such inputs. An architecture that has been proven well in the field of image classification are so called Convolutional Neural Networks (CNN). Due to a special type of layer, the convolutional layer, CNNs are able to exploit local structure in data which is especially relevant for images.

Emotion	Samples
Angry	4,953
Disgust	547
Fear	5,121
Happy	8,989
Sad	6,077
Surprise	4,002
Neutral	6,198

Table 1: Emotion distribution of the FER2013 dataset.

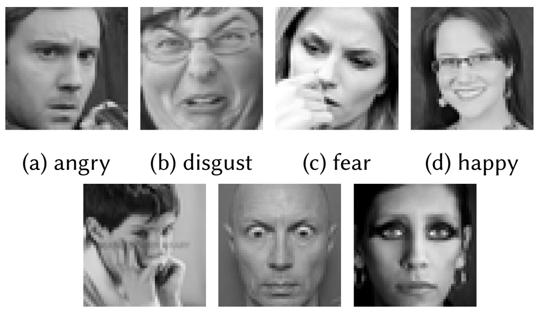

(a) angry (b) disgust (c) fear (d) happy
(e) sad (f) surprise (g) neutral

Figure 2: Exemplary images from all emotions present in the FER2013 dataset.

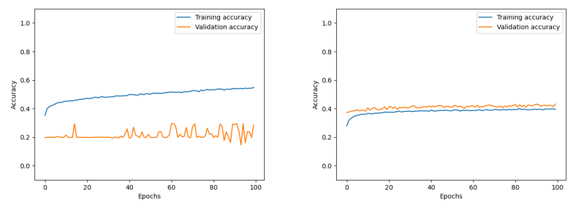
For our case we reviewed several CNN architectures which were originally developed for the ImageNet challenge (Russakovsky et al., 2015). These models were trained on several million images in order to classify images into more than 1,000 classes.

5.1 Training data

Dataset As training data the well-known FER2013 dataset, which is publicly available on kaggle, was used. It consists of 35,887 images normalized to 48×48 pixels in grayscale. Example images for each class can be seen in Figure 2. However, the dataset is not balanced as the classes are not uniformly distributed. The distribution of the seven emotions present in this dataset is summarized in Table 1.

It can be seen that disgust is especially underrepresented and happy has by far the highest number of samples. However, in our case the two emotions Disgust and Fear were removed as they are not relevant. In the case of online-meetings it is very unlikely to be disgusted or afraid and in the unlikely case there is probably no causality between the emotion and the online-meeeting.

Data augmentation As the FER2013 dataset is still rather small we augmented the dataset with real-time augmentation methods. The used techniques were horizontal mirroring, rotations of ±10 degrees, images



(a) ResNet50 (b) VGG16

Figure 3: Training and validation accuracy for the pre-trained models.

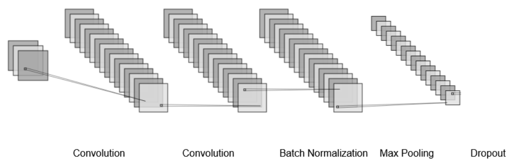


Figure 4: Convolution block structure. The first two convolutional layers are followed by batch normalization and max pooling. In the ende a dropout is applied.

zooms of ±10% as well as ±10% horizontal and vertical shifting.

5.2 Models

As already mentioned the used models were adapted from well-known CNN architectures. For training we used the categorical cross-entropy loss for all evaluated models.

ResNet50 ResNet50 (He et al., 2016) is a 50 layer residual network. As the original output layer was designed for a $1,000$ class classification problem we replaced the output layer with two fully connected layers with $4,096$ and $1,024$ neurons respectively and a fully connected layer with 5 output neurons and softmax activation for the final classification. After each of these fully connected layers we employed a dropout of 0.5 to reduce the risk of overfitting. We also used the pre-trained weights from the ImageNet challenge as the model already learned to detect various features in images. The original network was frozen, i.e. the weights will not be changed by training. The entire model was trained with stochastic gradient descent with a learning rate of 0.01 and a batch size of 128. After 100 epochs of training the training accuracy raised from 0.35 to 0.55. However, the validation accuracy stayed between 0.2 and 0.3. This behaviour implicates the model just remembered the training dataset and did not generalize the learned features which is a strong indicator for overfitting. The training and validation accuracy are also depicted in Figure 3 (a). It can be seen that the training accuracy is increasing while the validation accuracy does not increase it is more or less constant with some fluctuation which usually means the model overfits.

VGG16 The VGG16 net (Simonyan and Zisserman, 2014) was also pre-trained on the ImageNet dataset. With only 16 layers the network structure itself is smaller than ResNet50 however it is more complex and has much more parameters. Again we froze the layers from the original net and added a customized classification layer consisting of two dense layers of size $4,096$ and $1,024$ respectively. To prevent the model from overfitting we added a 50% dropout after each of these dense layers. This model achieved a training and validation accuracy of approximately 0.4. However, the model starved after less than 40 epochs of training, i.e. the training as well as the validation accuracy did not increase anymore and stayed more or less constant. This can also be seen in Figure 3 (b). We trained the model for 200 more epochs but the validation accuracy did not increase.

Our model As both pre-trained networks did not achieve good results we decided to build a new architecture from scratch. As larger neural nets are generally prone to overfitting on small datasets we kept the number of parameters relatively small compared to e.g. ResNet50. We also used dropout to further reduce the risk of overfitting. The overall architecture is similar to the VGG16 net, i.e. we also used blocks of two convolutional layers followed by a max pooling layer.

Convolutional Blocks The main component of the architecture are the 4 convolutional blocks. Each block consists of two convolutional layers with the same number of filters and a 3×3 stride. They are followed by a batch normalization and a 2×2 max pooling layer. After that a dropout of 0.5 is applied to reduce the risk of overfitting. The overall block structure is also depicted in Figure 4.

Classification block After the convolutional blocks we employ 4 dense layers followed by a classification layer with softmax activation for final prediction. The number of neurons in these dense layers is the same as the number of filters in the convolutional blocks.

For the final model we used the following number of filters per block respectively: 256, 128, 64 and 32. The overall architecture has less than 1.5 million parameters making it a lot smaller compared to the other two evaluated models. The final architecture of this CNN is also depicted in Figure 5.

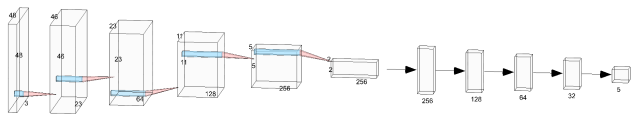


Figure 5: Overall architecture of our model.

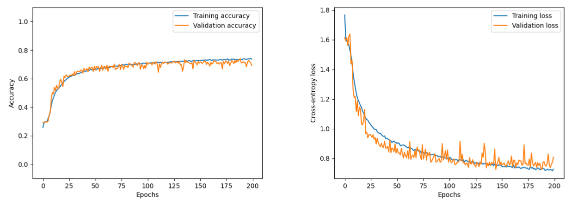
	Large	Normal	Small
1st conv. block	64	32	16
2nd conv. block	128	64	32
3rd conv. block	256	128	64
4th conv. block	512	256	128
1st dense layer	512	256	128
2nd dense layer	256	128	64
3rd dense layer	128	64	32
4th dense layer	64	32	16

Table 2: Number of filters and neurons for all evaluated versions of our model.

We also used other parameters to see if this would change the accuracy of the model. By doubling the number of filters per block we were able to raise the accuracy to 0.743, however, this model has 4 times more parameters than ours with an increase in accuracy of only 0.04 which is why we stayed with the smaller version. We also tried a smaller version of our model by halving the number of filters per block. The resulting model had only about 370,000 parameters but still achieved an accuracy of 0.697. The respective parameters for each of these models is summarized in Table 2.

5.3 Results

The results of the evaluated models are summarized in Table 3. TODO confusion matrix



(a) Accuracy (b) Loss

Figure 6: Accuracy and loss during 200 epochs of training for the final model we used.

Model	Parameters	Accuracy
ResNet50	36,180,869	0.296
VGG16	21,016,389	0.432
Our model (large)	5,906,757	0.743
Our model	1,479,845	0.739
Our model (small)	371,541	0.697

Table 3: Number of parameters and validation accuracy for the evaluated models.

References

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.

Simonyan, Karen and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.