Implementation of an end-to-end Deep Learning pipeline for Facial Expression Recognition

Thi Huyen Cao

University of Applied Science Hamburg
Berliner Tor 5, 20099 Hamburg, Germany
thihuyen.cao@haw-hamburg.de

Abstract. This paper presents the follow-up project mentioned in [1]. It contains the implementation of the end-to-end Deep Learning pipeline for classifying 6 basic emotions on short videos. Concretely, it summarizes the implementation of each steps, the choices of technology, the training process as well as a report on the result and further problems.

Keywords: Facial Expression Recognition (FER) · Deep Learning (DL) · Face Detection · Face Tracking · Neural Network · LRCN

1 Introduction

This work delivers the step-by-step implementation of the pipeline described in [1]. It was designed to classify 6 basic emotions (anger, disgust, fear, happiness, sadness, surprise) on short videos that were recorded in laboratory under professional instructions. A detail about the potential architecture, technology, algorithm, libraries, methodology etc. were already discussed and explained in the previous work [1] and will be avoided here to prevent unnecessary duplicate. This work focuses mainly on the implementation itself and the author's decision on certain concepts, solutions along with the arguments. The rest of the paper is structured as follows. Next section, also the main section, contains the implementation of the pipeline including 1. an analysis on the dataset 2. detailed preprocessing steps and their results 3. the hardware setup 4. training process and 5. evaluation. A short conclusion is drawn in the last section.

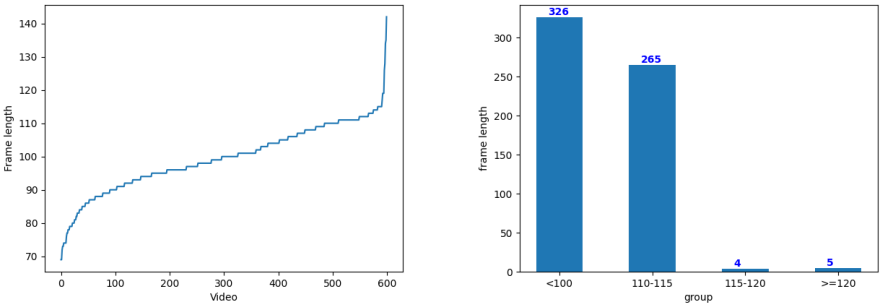
2 Implementation

2.1 Dataset

The original dataset BU-4DFE from university Binghamton contains in laboratory recorded videos from 101 objects with a variety of background. To gain an overview about the dataset, some statistics are created in advance.

- Out of 101 objects, there are 43 males and 57 females. For object F16, there is unfortunately a missing video of emotion happiness. For this experiment, a request of missing data is not done. Instead, object F16 is simply removed from the dataset in further steps.

- There are in total 100 objects x 6 emotions = 600 short videos.
- The video resolution (height x width) is 480x640, frame per second is 25.
- The shortest video is 69 frames ≈ 2.76 seconds long and the longest video 142 frames ≈ 5.68 seconds. The average length of all videos is 3.98 seconds. Below are 2 diagrams which visualize the statistic of frame length



(a) Frame length of each video

(b) Frame length in 4 groups

Fig. 1: Frame length statistic

Figure 1(a) shows the length of each video. The list of video is sorted by length in ascending order in advance. Figure 1(b) visualizes 4 categories: number of videos with less than 100 frames, between 110-115, 115-120 and more than 120 frames. It is clear to see that most videos are less than 115 frames long. This information plays a very important role at the decision of input dimension at training. Because certain type of networks e.g. convolutional neural network might need the same dimension for all input which requires a padding or truncation of the original data. A padding of too many zero frames or a truncation of too many informative frames might hurt the performance of the model ultimately.

2.2 Preprocessiong

This step prepares data for training. From the original data, face will be detected and tracked. It is then re-sized reasonably and saved for further steps.

Face detection An example of the original data is shown in figure 2. It is quite easy to detect face in such setup since the background is monochrome. A naive way of detecting face is based on the different colors among background and content, which might be sufficient for this dataset although hair and neck might be redundantly captured. However, the author decided to go with a classical

machine learning face detector so that the model can be directly tested with data from other contributions later on if desired. Haar Cascade face detector by Paul Viola and Michael Jones [2] was chosen under careful consideration. The first and main argument is that Haar Cascade detector operates in general faster than modern Deep Learning detectors such as Multi-task Convolutional Neural Network (MTCNN). It has also a good accuracy, especially at detecting frontal faces. Deep Learning face detectors on the other hand have very good performance even in case of occlusion and non-frontal. Since this is a preprocessing step of a lot of data, inference speed is a decisive factor as long as performance is kept at an accepted level. Also detected faces in certain angle or with occlusion might not be interesting for FER since it is very difficult for model to determine the emotion after all.

There are available pre-trained Haar Cascade face detectors with parameters stored in xml files, which can be downloaded from [3] and imported easily in opencv. haarcascade_frontalface_default.xml was used for this step.

```
# load pre-trained detector
detector = cv2.CascadeClassifier ('Path to pre-trained face
    detector')
# detecting face
faces = detector.detectMultiScale(frame,
                                  scaleFactor=1.1,
                                  minNeighbors=10,
                                  minSize=(30, 30),
                                  flags=cv2.CASCADE_SCALE_IMAGE)
if len(faces) != 1:
  # log error and exit
```

A tolerance of detect_frame_allowed = 10 frames was given, means trying to detect face in the first 10 frames. However, thank to the center and frontal position of face, the detecting process was quite fast, stable and accurate. Figure 2 is an example. While expressing the emotion "surprise", an open mouth is

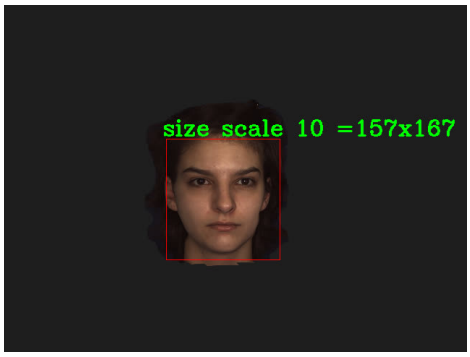


Fig. 2: Detected face with dimension of width x height = 157x167, extended pixel = 10

commonly seen. Unfortunately, the detector sometimes failed to detect it or determined the bounding box inaccurately (cut through half of the mouth) which caused the tracking process later on a lot of difficulties. As that, 10 pixel is extended in the height dimension.

Face tracking Opencv has a lot of implemented trackers, each with its own advantages and disadvantages. A detailed comparison among them was done in the previous project [1]. For this particular dataset, most trackers should meet the requirements and should perform well enough. Despite that, the author decided to run a test on 4 objects, corresponding to 24 videos and chose the tracker based on the method "trial and error". The result was summarized in table 1.

Tracker	Failed samples	Time
CSRT	0	214s
KCF	0	167s
MedianFlow	0	66s

Table 1: Result of MedianFlow, KCF and CSRT tracker

All tracker performed as expected very well. For all tested trackers, no failed example was recorded. As a consequence, MedianFlow was chosen because of its fastest inference time. Average size of detected and tracked face is from 150x150 $\rightarrow$ 190x190, mostly in the range from 155 $\rightarrow$ 165. Therefore, faces were resized to 160x160 and written to video temporally with lossless compression for further process. Lossless compression ensures the quality of the original video to be kept as much as possible.

```
# defined codec
fourcc = cv2.VideoWriter_fourcc(*'HFYU')
# init output writer
out = cv2.VideoWriter('Path to save video',
                      fourcc,
                      fps,
                      (width, height),
                      color_mode)
```

Figure 3 is an example of detected and re-sized face which is used for training.

Data was then split into train, validation and test set carefully so that each set contains enough data for a reasonable result. Concretely, the ratio of train/validation/test is 60%/20%/20% corresponding to 60/20/20 persons and 360/120/120 videos. Even though the dataset is quite small, the validation and test set need to be big enough in comparison to the total data to present the model quality at the end. The model will have to be trained to learn from a

Fig. 3: Detected and resized face

group of people and be able to give good prediction on completely new people. This will be more useful in real life applications even though it might be more challenging in comparison to a setup where it is allowed to learn and test from/on the same group of people.

- train set: 23 male + 37 female (M21-M43, F22-F48)
- validation set: 10 male + 10 female (M01-M10, F01-F10)
- test set: 10 male + 10 female (M11-M20, F11-F21)

Unfortunately, data can not be written to hdf5 with h5py, which is an optimal format for storing training data. To the best of author's knowledge, h5py has not supported storing multidimensional data with variable length (video with different length) yet. Therefore, data will be stored in hard disk under original folder structure /person-id/video-id. Be aware of the small training set, data augmentation will be exploited at some points.

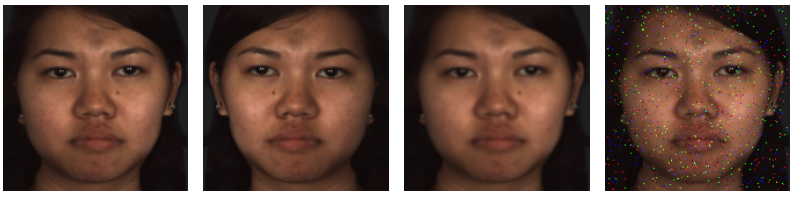
Data augmentation In short, data augmentation is a very useful technique to help generate more data from available data. Multiple transformation methods can be used to increase the size of train set as long as the essential information needed for the task is kept. As a result, the following methods were considered in order to remain all the movement of eyes and mouth.

- flipping horizontally
- using Guassian blur (different blur effects)
- adding salt and pepper noise (different percentage of noise)
- equalizing histogram
- changing contrast (higher and lower)
- changing brightness (higher and lower)
- using grayscale

Most methods are well implemented within opencv and can be used easily within some lines of code.

```
# example of transformation methods
flipped_horizontally = cv2.flip(image, 1)
guassian_blur = cv2.GaussianBlur(image, (3, 3), 0)
brightness_higher = cv2.convertScaleAbs(image,
                                        alpha=1,
                                        beta=50)
brightness_lower = cv2.convertScaleAbs(image,
                                       alpha=1,
                                       beta=20)
contrast_higher = cv2.convertScaleAbs(image,
                                      alpha=2,
                                      beta=0)
contrast_lower = cv2.convertScaleAbs(image,
                                     alpha=1.7,
                                     beta=50)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
histogram_equalization = cv2.equalizeHist(image)
```

Keras officially support loading and augmenting data on-the-fly or in other word online data augmentation for image data with class ImageDataGenerator. For video data, it is required to implement an own custom loader or to use an available implemented plugin. Due to the quite small size of the dataset and the available memory of renderfarm from CSTI [4] where the model was trained, the author decided to load and augment all data right before training manually. Figure 4 visualizes an example of some augmented frames.



(a) original (b) flip horizontally (c) guassian blur (d) salt and pepper

(e) brightness (f) contrast (g) equalize histogram (h) grayscale

Fig. 4: Augmented example

2.3 Hardware setup

The training was done in the renderfarm provided by CSTI [4]. Following infrastructure was determined in advance.

- OS: CentOS Linux release 7.7.1908 (Core)
- GPU: 4 NVIDIA Quadro P6000 each with ≈ 24 GiB
- CPU: Intel Core Processor (Broadwell, no TSX) 2.3 GHz 18 cores 18 threads
- RAM: ≈ 163 GiB

2.4 Training

The training process was containerized in 2 docker containers: one for training and one for tracking result. The result was forwarding to local machine for the purpose of monitoring.

```
# start tracking container in background
docker run -d --runtime=nvidia -it --rm
        -v /workspace/:/app/
        -p 6006:6006
        tensorflow/tensorflow:2.3.2-gpu
        tensorboard --logdir /app/logs --bind_all
# start training container
docker run --runtime=nvidia -it --rm
        -v /workspace/:/app
        --env-file /workspace/env
        --shm-size=512m
        fer
# forwarding to local machine
ssh -L 6006:127.0.0.1:6006 username@host-ip-addrss
# monitoring processes (mem usage etc.)
watch nvidia-smi
docker stats
```

Choice of architecture The choice of neural network architecture is further discussed in [1]. It strongly depends on the task itself. Basically, Facial Expression Recognition requires a model which is able to understand both the visual information in each frame as well as the temporal information across all frames. Cascade Network is shown to be able to perform well in most research papers, which is a combination of different types of neural networks in order to take advantage of all their pros. Table 5 shows again most possible choices and their characteristics in term of data size requirement, information learn ability, frame length requirement, performance and computational efficiency.

Long-term Recurrent Convolutional Network (LRCN) was chosen as the starting point under careful consideration. LRCN is a Cascade Network which contains CNN layers as front-end and Long Short Term Memory (LSTM) layers

Network type		data	spatial	temporal	frame length	accuracy	efficiency
Frame aggregation		low	good	no	depends	fair	high
Expression intensity		fair	good	low	fixed	fair	varies
Spatio-temporal network	RNN	low	low	good	variable	low	fair
	C3D	high	good	fair	fixed	low	fair
	$\mathcal{FLT}$	fair	fair	fair	fixed	low	high
	$\mathcal{CN}$	high	good	good	variable	good	fair
	$\mathcal{NE}$	low	good	good	fixed	good	low

Fig. 5: Comparison of different types of methods for dynamic image sequences in terms of data size requirement, representability of spatial and temporal information, requirement on frame length, performance, and computational efficiency. FLT = Facial Landmark Trajectory; CN = Cascaded Network; NE = Network Ensemble. [5]

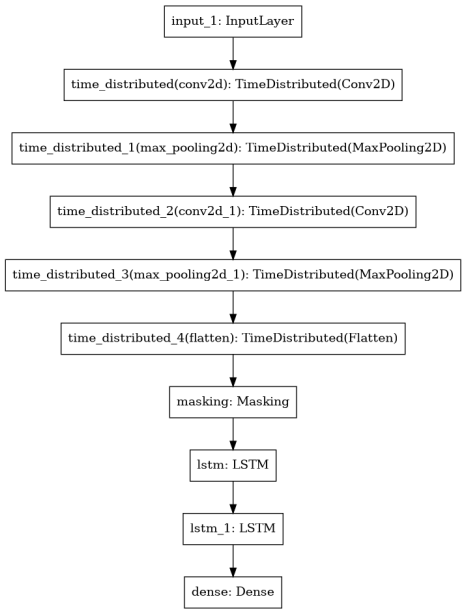
as backend. As that, it accumulates all the advantages of 1. convolutional network at learning features in space and 2. recurrent network at learning features in time. A further read about those networks and why are they able to learn such features can be found in previous work [1]. Despite all the advantages of LRCN or Cascade Network, a drawback which needs to be kept in mind is its requirement of big dataset to perform well. Based on the author's own experience, the given dataset might be sufficient for training a reasonable deep network.

Training strategy All models were built with Keras Functional API to have more control over the training process in comparison to Sequential API. Keras is basically the high level interface on top of Tensorflow. A more read can be found in the official page of the libraries as well as in the author's previous work [1].

After experimenting with some scenarios, the author came to the decision of network architecture and some base parameters.

- Based on the frame statistic recorded from the dataset analysis in section 2.1 and the requirement of same input dimension of CNN, the length of all videos is re-sized to 115 frames, meaning longer videos were truncated and shorter videos were zero-padded. Because firstly most videos are within this frame range and secondly a brutal truncation or a redundant padding both result in a very bad network performance.
- Experiment with input data in color and gray scale pointed out that there was no considerate difference. As the data with 3 color channels is three times bigger and as a result takes longer to train and consumes additionally more resources, the author decided to use gray scale for further process.
- As already discussed in section 2.2, the train, validation and test set should be split reasonably to ensure the quality of trained models. Therefore, cross-validation is purposely avoided because it might mix the dataset randomly and cause less reliable results. In addition, cross-validation requires also a lot more computing power which is a trade off to be considered.

– Experiment with different learning rates showed that an extremely small learning rate is needed for the model to learn. Also an important notice is that the learning rate should be changed proportionally with the batch size.
– The base parameters are listed as follows:
  • Input dimension (frame length x height x width x number of color channel): 115x160x160x1
  • Learning rate: 0.00001
  • Optimizer: Adam
  • Batch size: 16
  • Epochs: 100
  • Shuffle data during training: True
  • Callbacks: EarlyStopping to stop the training process if validation accuracy increases 30 epochs continuously; ModelCheckpoint to save best model based on validation accuracy.



(a) Network architecture

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 115, 160, 160, 1)	0
time_distributed (TimeDistri	(None, 115, 158, 158, 32)	320
time_distributed_1 (TimeDist	(None, 115, 79, 79, 32)	0
time_distributed_2 (TimeDist	(None, 115, 77, 77, 32)	9248
time_distributed_3 (TimeDist	(None, 115, 38, 38, 32)	0
time_distributed_4 (TimeDist	(None, 115, 46208)	0
masking (Masking)	(None, 115, 46208)	0
lstm (LSTM)	(None, 115, 128)	23724544
lstm_1 (LSTM)	(None, 128)	131584
dense (Dense)	(None, 6)	774

Total params: 23,866,470
Trainable params: 23,866,470
Non-trainable params: 0

(b) Parameters summary

Fig. 6: Network architecture and parameters summary

– The baseline network contains 2 CONV 2D layers, each with 32 filters, 3x3 filter kernel, relu as activation function and followed by a MAX POOLING layer with 2x2 filter kernel. The extracted features from the CNN layers are then fit to the backend. In order to pass the representation vector of each frame as an input for a timestep in LSTM, those layers need to be wrapped in TimeDistributed layers. The backend contains 2 LSTM layers, each with 128

hidden units. As output layer, a Dense layer with activation function softmax of 6 classes was used. Activation function softmax outputs the probability of each class and is the best practice for a multi-class classification. Figure 6 above summarizes the model architecture and number of parameters .

As further mentioned in the previous work [1], there is no silver bullet when it comes to train a neural network. The key lies on the understanding of network performance during the training process, the training techniques and their effects as well as the personal experience of the trainer. To understand how the model performs at the current point, one needs to get familiar with the concept of bias and variance, underfitting and overfitting. Some common training techniques nowadays are:

- Using more data with data augmentation or collecting more data
- L1, L2 and dropout
- Normalization and batchnormalization
- Transfer learning

The training strategy of this work is strongly based of the recommendation of Prof. Andrew Ng from his courses at Coursera and Stanford University. In short, if the network shows a very high bias, considering training longer, using bigger network etc. On the other hand, if the bias seems to be fine, but the variance is high, consider using more data and regularization techniques to improve network generalization and avoid unwanted learn-by-heart effect.

Hyperparameters tuning The difficult part of training a neural network lies on the process of tuning hyperparameters which are number of layers, number of neurons in each layer, type of layer, initialization methods, dropout, l1, l2, size of kernel, optimizer, learning rate and so on. The training process is without doubt an empirical process and very resource-consuming in term of time, computing power and human effort. In the following, the author would like to present some of the steps which were executed and the lesson learnt.

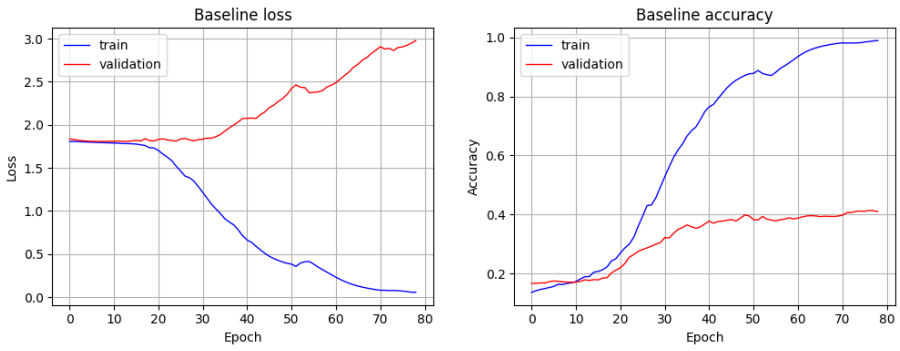


Fig. 7: Baseline accuracy and loss

0. Baseline: As shown in figure 7 above, the baseline indicated a very big gap of overfitting, almost 60% which was actually no surprise for such a small amount of train data. The train loss decreased continuously that suggested the sufficient capacity of learning of the chosen network. Train accuracy achieved 100% around 60 epochs. Validation accuracy increased slowly but then stayed stable around 40% after 30 epochs. Despite that, validation loss tended to increase which seemed questionable at first. One answer for this phenomena is the characteristic of the softmax activation function at output layer. For example sample s is predicted correctly in the early epochs with the probability of 0.9. Later on, the network is getting worse and predicts sample s with a lower probability say 0.55. This happens to some of the samples and causes the loss to increase even though the accuracy seems to be the same. Ultimately, the network showed a good learning capacity but a bad ability of generalization. In the next steps, some methods are tested to fight again overfitting.

1. Data augmentation: An increase of data using data augmentation is the first try to reduce the overfitting gap. Data was augmented with the methods mentioned in section 2.2. A rise of 7 times the original train data improved the performance clearly. Train accuracy run up faster than the baseline, see figure 8 (left). The validation accuracy was generally 10% better which is a proof of a better generalization. Even though the validation accuracy reached and stayed around 50%, there was still a very high overfitting gap which needed to be further reduced. A more extreme increase of data to 11 times the original data showed no further improvement except a slightly faster increase of train accuracy as shown in figure 8 (right). It could be a sign that more real data need to be collected or other augmentation methods need be considered which should be kept in mind. However, sacrificing the train speed and memory for such a small effect was not effective at this point. As that, the author decided to increase the data only 7 times in the further process.

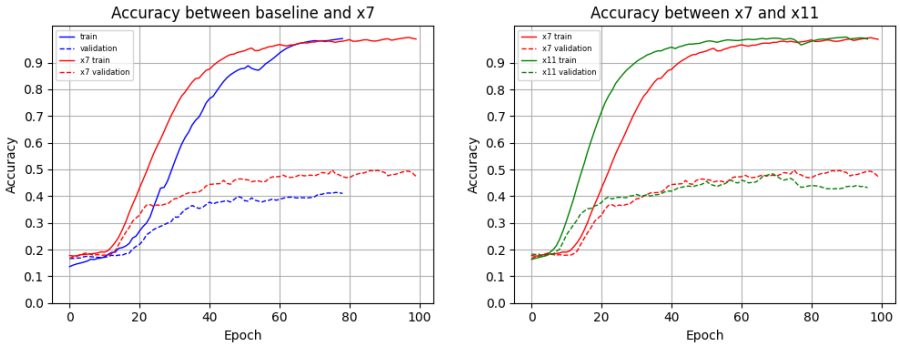


Fig. 8: Data augmentation effect

2. L1, L2: L1, L2 regularization are classic techniques which assign a penalty on weights to keep it smaller and smaller. By forcing some weights close to zero, it deactivates certain neurons which results a smaller and more qualitative network and correspondingly avoid overfitting. A higher λ, also known as regularization parameter, means more regularization. By observing with L2, a $\lambda \geq 0.01$ will prevent the network from learning at all. Some experiments were setup with a smaller λ also resulted no improvement (see figure 9). There could be multiple reasons why such regularization did not work out. But before diving further into the reasons, the author chose to experiment with another well-known technique called dropout.

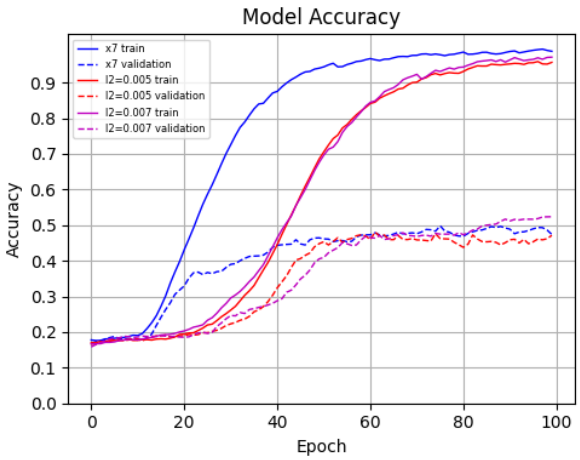


Fig. 9: L2 regularization effect

3. Dropout: Dropout is one of the most powerful regularization techniques nowadays. Dropout was experimented with different amounts from [0.0, 1.0]. 0 means drop all, 1 means keep all neurons and a value in between is the best fit to look for. Especially for LSTM layers, 2 types of dropout were tested: input dropout and recurrent dropout. As the name suggests, input dropout is applied on the input/output in each timestep while recurrent dropout on the recurrent connections from timestep to timestep. Different scenarios were investigated: dropout on only CNN layers, on only LSTM layers and on all layers. The network reacted very sensitive on dropout at CNN layers. No matter which amount of dropout is applied on CNN layers, the model performed very poorly. The train accuracy increased very slowly and the best accuracy reached only 30% (see figure 10). Dropout on LSTM layers gained generally good impact. A naive dropout on recurrent layers helped increase the train accuracy much faster. Nonetheless, the validation accuracy showed very small improvement of about

4% and some high peaks over 55%. The same effect was recorded on recurrent dropout.

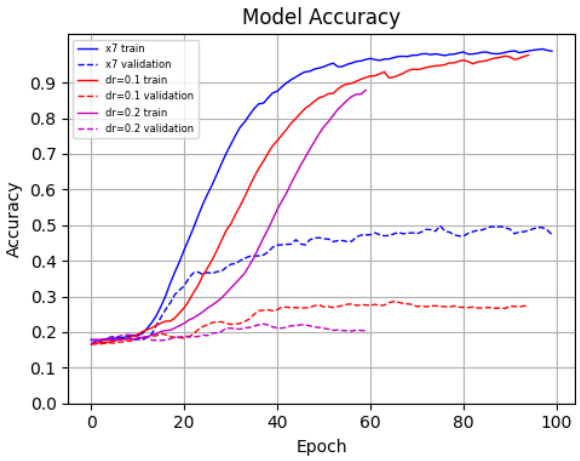


Fig. 10: Dropout regularization effect

2.5 Evaluation

The classification among 6 classes has a random metric of 16.67%. Even the baseline achieved better result (40%) than a random metric. Best validation accuracy recorded was 59%. However, the training, especially with dropout resulted in general quite unstable validation accuracy course. A clear improvement was hard to determine. A clear boost was found while applying data augmentation to increase the train set 7 times, the overfitting gap reduced 10% correspondingly.

Nevertheless, after applying different regularization methods, the network still showed a very big overfitting which is a signal of poor generalization ability. One reason, as discussed in the previous work, could be the small amount of data. By splitting into 3 sets, the amount of data for training reduced even more. Despite the fact that data augmentation did help against overfitting, the model ultimately need to see enough good data to generalize. Some analysis also showed that neutral faces were found at the beginning and at the end of many videos which could also mess up the learning. Another reason could be the high complexity of this task. The model had to learn data from some people and give prediction on other people. Belows are some possible solutions to give a try in future works:

- Collecting more qualitative data or using different augmentation methods
- Handling neutral faces at the beginning and at the end of video
- Using transfer learning such as VGGFACE [6]

3 Conclusion

This paper summarizes the step-by-step implementation of the pipeline introduced in [1] to classify 6 basic emotions from short laboratory recorded videos. A small LRCN with 2 CNN layers as front-end and 2 LSTM layers as backend was chosen, achieved at best an accuracy of 59% on validation set. Through the training process, the author found it extensively difficult to avoid overfitting. The author also acknowledged the effectiveness of data augmentation at fighting against overfitting. Other methods such as dropout, l1, l2 etc. turned out to have very little effect on this task. The author suggests a further training using some solutions mentioned above such as handling neutral faces, using transfer learning and so on which unfortunately exceeds the project timeline. As that, there is no detailed error analysis on the mislabeled samples because of the little value it brings at this point.

References

1. Author, Thi Huyen Cao : Article title. End-to-end Deep Learning pipeline for FacialExpression Recognition `https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2021-proj/cao.pdf`. Last accessed 20 Jul 2021
2. Authors, Paul Viola and Michael Jones: Article title. Rapid Object Detection using a Boosted Cascade of Simple Features (2001)
3. OpenCV Github Page, `https://github.com/opencv/opencv/tree/master/data/haarcascades`. Last accessed 20 Jul 2021
4. Homepage Creative Space for Technical Innovation (CSTI), `https://csti.haw-hamburg.de/`. Last accessed 20 Jul 2021
5. Authors, Shan Li and Weihong Deng : Article title. Deep Facial Expression Recognition: A Survey (2018) `http://arxiv.org/abs/1804.08348`
6. Authors Omkar M. Parkhi and Andrea Vedaldi and Andrew Zisserman : Article title. Deep Face Recognition