Fusion approaches for spatiotemporal Face Expression Recognition (FER)

Table of Contents

Abstract

Facial expressions recognition is an important task of computer vision with many practical applications in mental health, surveillance, customer satisfaction,…etc. Using computer vision and convolutional neural networks, we can learn how to do the mapping from image frames to the expression class of emotions, like anger, sadness, joy,…etc. However, learning affective features from video stream is a challenging task that requires incorporation of not only the spatial fixed image frame features, but also the temporal features of the successive video frames. In this work, we tackle the problem of dynamic facial expression recognition from videos, using spatio-temporal approaches that fuses both spatial and temporal modalities. Basically we use the optical flow as the temporal features that aggregates frames over time. Moreover, we evaluate six different neural networks architectures that fuses both modalities, with different design choices in terms of the fusion operators and the type of fusion layers. In all our experiments, we rely on the ability of the convolutional neural networks to extract useful, local, translation invariant and abstract features from both spatial and temporal inputs. Finally, we evaluate our approaches on the BAUM dataset [3] based on careful and fair data split that accounts for the temporal correlation. We analyse the results using confusion matrices and visualize the learnt features over the spatial aspect using Grad-CAM technique [5].

1 Introduction

In this work, we tackle the problem of recognizing facial expression in videos. Our target is to map a group of video frames to a certain emotion class. The main goal is to incorporate both temporal and spatial aspects.

There are many approaches in literature to account for the temporal information in videos, like 3D CNN, Recurrent Nets and frame stacking (see related work in [1]). In this work we focus on a two-stream CNN approach as in [1], and extend that with different fusion operations and layers, to have a comparative study.

2 Background and Related Work

According to [1], we will follow the following architecture:

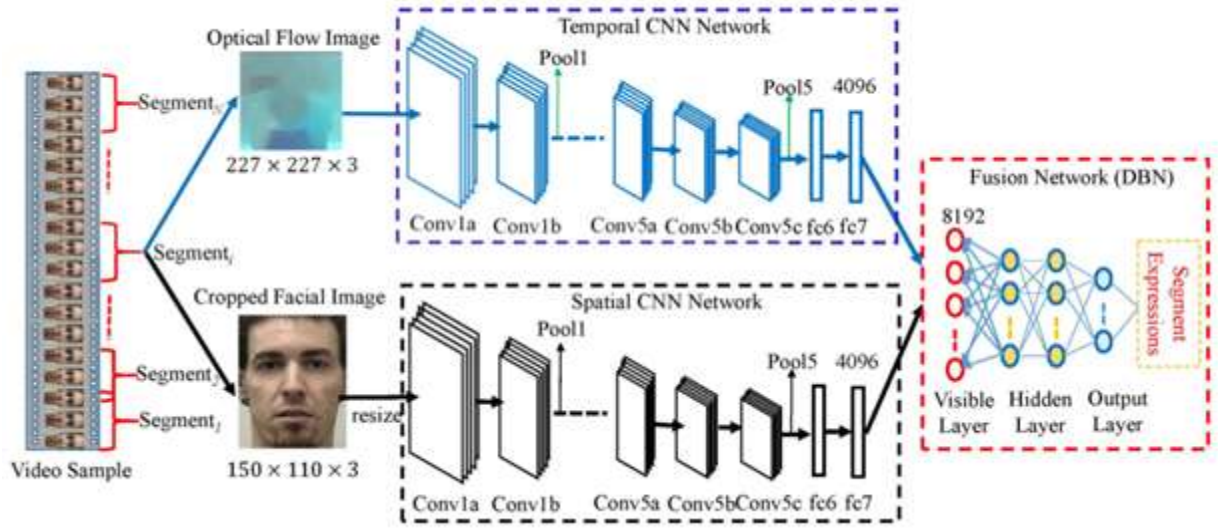


Figure 1 Spatio-temporal approach as in [1]

With the following modifications:
1- Replace DBN with normal Dense (Fully Connected) layers
2- Use normal softmax with categorical cross entropy loss as the classification layer

3 Baseline model

Following this approach, we developed two baseline models:

1. Spatial CNN model

Figure 2 Baseline spatial model

2. Spatiotemporal 2 stream CNNs model (Fusion)

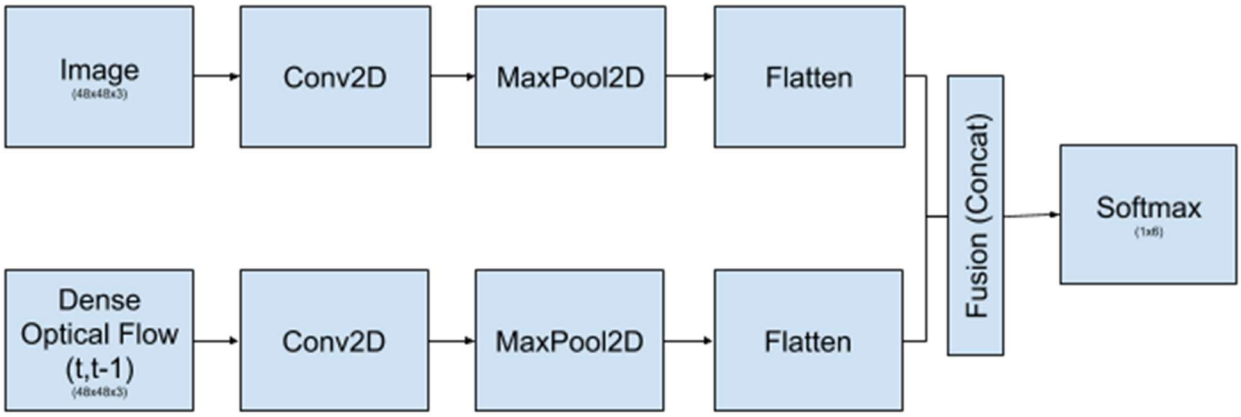


Figure 3 Baseline spatio-temporal model

In [1] VGG back ends were used. However, VGG is known for its large memory footprint. Following the same for the spatial model caused overfitting. Also, with the frame skipping approach (see in Data preparation section), the data is reduced by ⅓, which caused the model to under fit. Going to the fusion model, it was not possible to use 2 VGGs due to GPU memory constraints. For all those reasons, we resort to a smaller model, which matches the problem size.

The full layers details is shown below:

```
# Create Model
def create_model():
  input spatial = Input(shape=(48,48,3))
  x = layers.Conv2D(64, (5, 5), activation='relu')(input_spatial)
  x = layers.MaxPooling2D(pool_size=(5,5), strides=(2, 2))(x)
  x = layers.Conv2D(64, (3, 3), activation='relu')(x)
  x = layers.Conv2D(64, (3, 3), activation='relu')(x)
  x = layers.AveragePooling2D(pool_size=(3,3), strides=(2, 2))(x)
  x = layers.Conv2D(128, (3, 3), activation='relu')(x)
  x = layers.Conv2D(128, (3, 3), activation='relu')(x)
  x = layers.AveragePooling2D(pool size=(3,3), strides=(2, 2))(x)
  Out_spatial = layers.Flatten()(x)

  input_flow = Input(shape=(48,48,3))
  x = layers.Conv2D(64, (5, 5), activation='relu')(input flow)
  x = layers.MaxPooling2D(pool_size=(5,5), strides=(2, 2))(x)
  x = layers.Conv2D(64, (3, 3), activation='relu')(x)
  x = layers.Conv2D(64, (3, 3), activation='relu')(x)
  x = layers.AveragePooling2D(pool_size=(3,3), strides=(2, 2))(x)
  x = layers.Conv2D(128, (3, 3), activation='relu')(x)
  x = layers.Conv2D(128, (3, 3), activation='relu')(x)
  x = layers.AveragePooling2D(pool_size=(3,3), strides=(2, 2))(x)
  Out flow = layers.Flatten()(x)

  merged1=concatenate([Out_spatial, Out_flow])

  x = layers.Dense(1024, activation='relu')(merged1)
  x = layers.Dropout(0.4)(x)
  x = layers.Dense(1024, activation='relu',
               kernel regularizer=l2(0.01),
               activity regularizer=l1(0.01))(x)
  x = layers.Dropout(0.4)(x)
  Out = layers.Dense(6, activation='softmax')(x)
```

```
    model = models.Model(inputs=[input_spatial, input_flow], outputs=[Out])
    return model
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 48, 48, 3)]  0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 48, 48, 3)]  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 44, 44, 64)   4864        input_1[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 44, 44, 64)   4864        input_2[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 20, 20, 64)   0           conv2d[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 20, 20, 64)   0           conv2d_5[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 18, 18, 64)   36928       max_pooling2d[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 18, 18, 64)   36928       max_pooling2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 16, 16, 64)   36928       conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 16, 16, 64)   36928       conv2d_6[0][0]
__________________________________________________________________________________________________
average_pooling2d (AveragePooli (None, 7, 7, 64)     0           conv2d_2[0][0]
__________________________________________________________________________________________________
average_pooling2d_2 (AveragePoo (None, 7, 7, 64)     0           conv2d_7[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 5, 5, 128)    73856       average_pooling2d[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 5, 5, 128)    73856       average_pooling2d_2[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 3, 3, 128)    147584      conv2d_3[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 3, 3, 128)    147584      conv2d_8[0][0]
__________________________________________________________________________________________________
average_pooling2d_1 (AveragePoo (None, 1, 1, 128)    0           conv2d_4[0][0]
__________________________________________________________________________________________________
average_pooling2d_3 (AveragePoo (None, 1, 1, 128)    0           conv2d_9[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 128)          0           average_pooling2d_1[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 128)          0           average_pooling2d_3[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 256)          0           flatten[0][0]
                                                                 flatten_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 1024)         263168      concatenate[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 1024)         0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1024)         1049600     dropout[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 1024)         0           dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 6)            6150        dropout_1[0][0]
==================================================================================================
Total params: 1,919,238
Trainable params: 1,919,238
Non-trainable params: 0
```

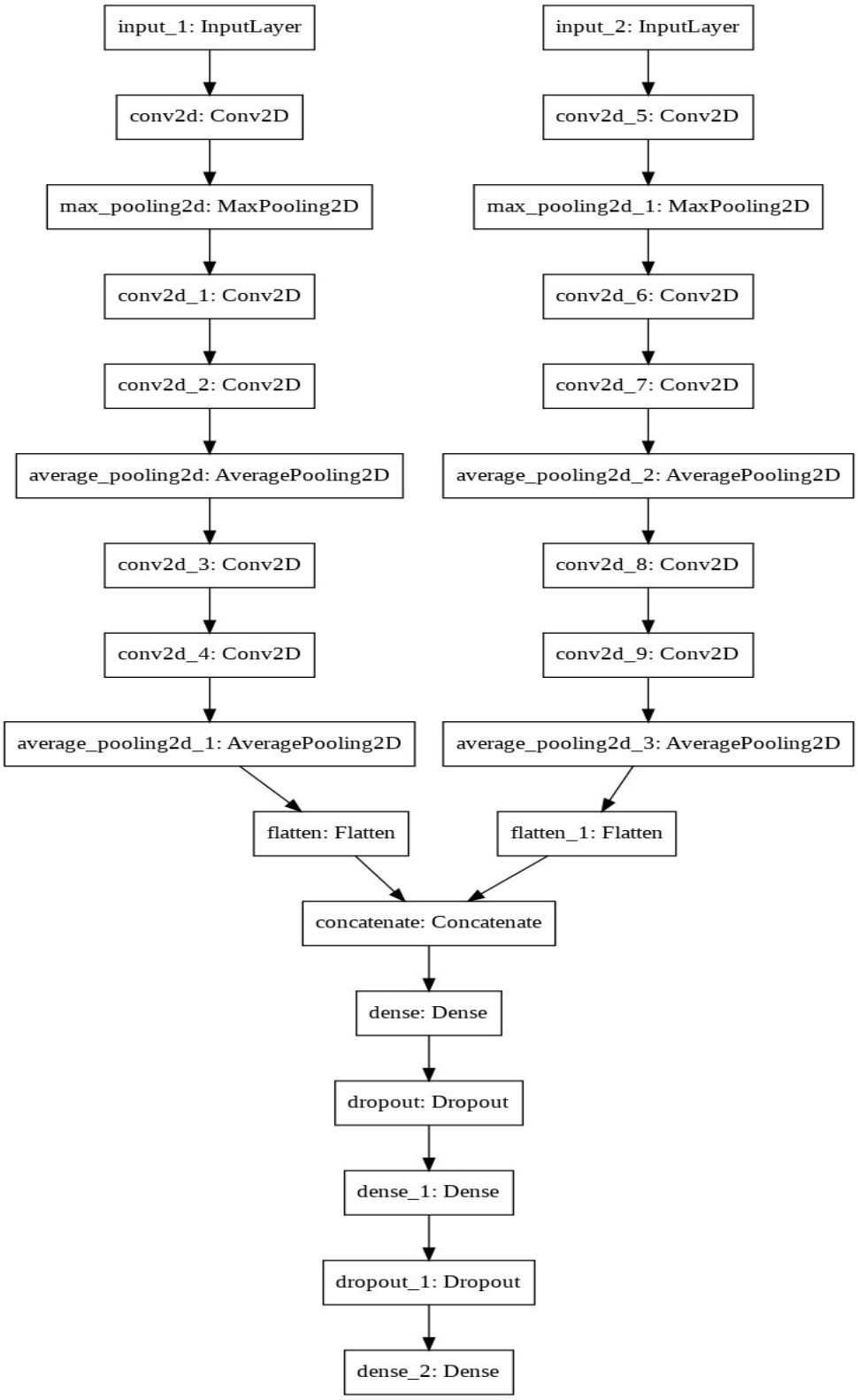


Figure 4 Baseline spatio-temporal model

We employ cross entropy loss for multi-class problems [1]

$$\min_{W,\theta} \sum_{i=1}^{N} H(\text{softmax}\,(W \cdot \Upsilon(a_i; \vartheta)), y_i)$$

$$H(\vartheta, y) = -\sum_{j=1}^{C} y_j \log(y_j)$$

4 Fusion methods

In [1], only one basic approach is tried, 2 CNN streams with concatenation layer as a merger.

In our experiments, we try two different axes:

4.1 Fusion operators

We want to see the effect of the following operators on the fused layers:

- Concat (same as in the paper): if the final 2 stream vectors are n-dim, the fused vector is 2n-dim
- Sum: the 2 stream vectors are added. If the final 2 stream vectors are n-dim, the fused vector is n-dim. This is equivalent to concat, but with non-learnable weights of 1s.
- Average: the 2 stream vectors are averaged. If the final 2 stream vectors are n-dim, the fused vector is n-dim.

4.2 Fusion layers

- Flatten (same as in the paper): The fused vector (n-dim) is fed into a Dense layer of output m-dim. The weight matrix here is nxm. The n-dim vector is simply the flattening of the final CNN feature maps, and their output channels taken row-wise.
- GlobalAveragePooling (GAP): Following [4], the final CNN feature maps are averaged across their output channels, resulting into a more compact and Dense vector than normal flatten

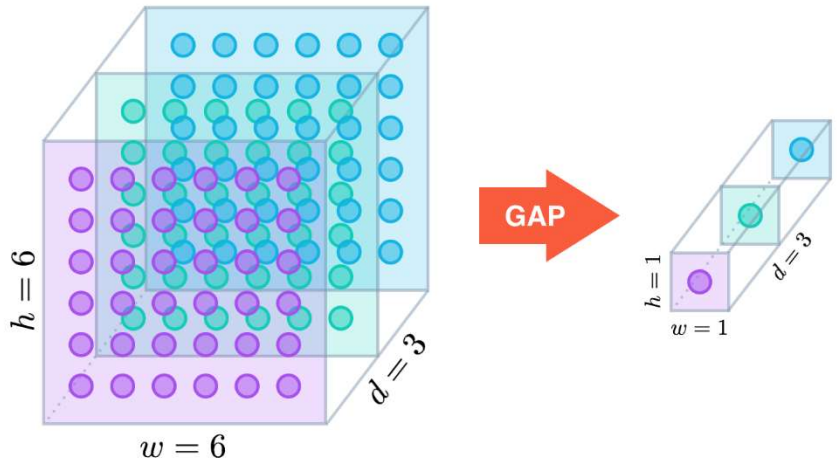


Figure 5 Global Average Pooling

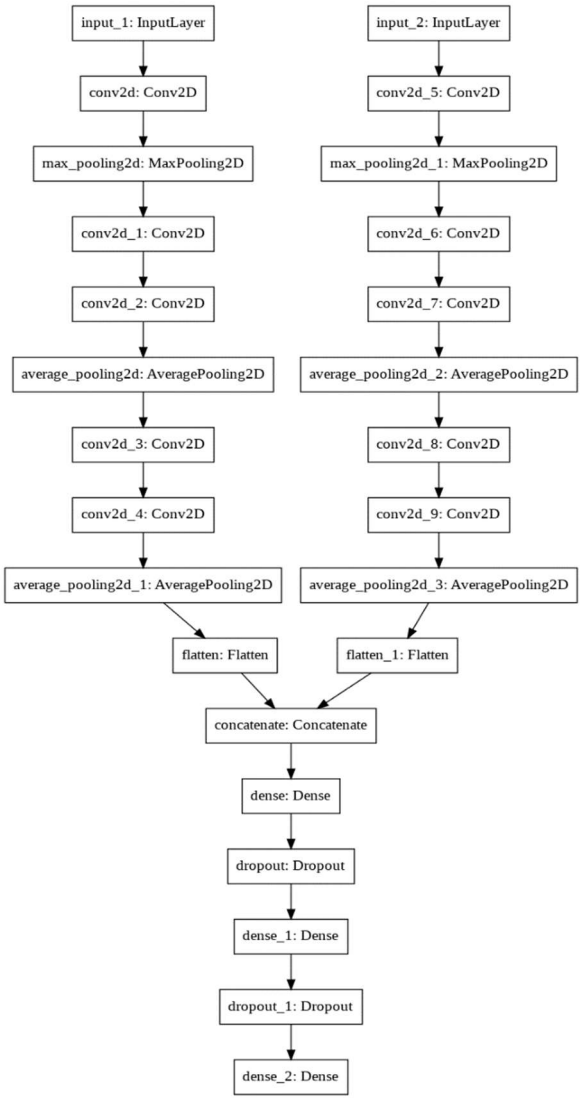
The GAP layer transforms the dimensions from (7, 7, 64) to (1, 1, 64) by performing the averaging across the 7 x 7 channel values. Global Average Pooling has the following advantages over the fully connected final layers paradigm:

- The removal of a large number of trainable parameters from the model. Fully connected or dense layers have lots of parameters. A 7 x 7 x 64 CNN output being flattened and fed into a 500 node dense layer yields 1.56 million weights, which need to be trained. Removing these layers speeds up the training of your model.
- The elimination of all these trainable parameters also reduces the tendency of over-fitting, which needs to be managed in fully connected layers by the use of dropout.
- The authors argue in the original paper that removing the fully connected classification layers forces the feature maps to be more closely related to the classification categories – so that each feature map becomes a kind of "category confidence map".
- Finally, the authors also argue that, due to the averaging operation over the feature maps, this makes the model more robust to spatial translations in the data. In other words, as long as the requisite feature is included / or activated in the feature map somewhere, it will still be "picked up" by the averaging operation.

Accordingly, we end up with 6 combinations to test the best fusion strategy. Details are shown below:

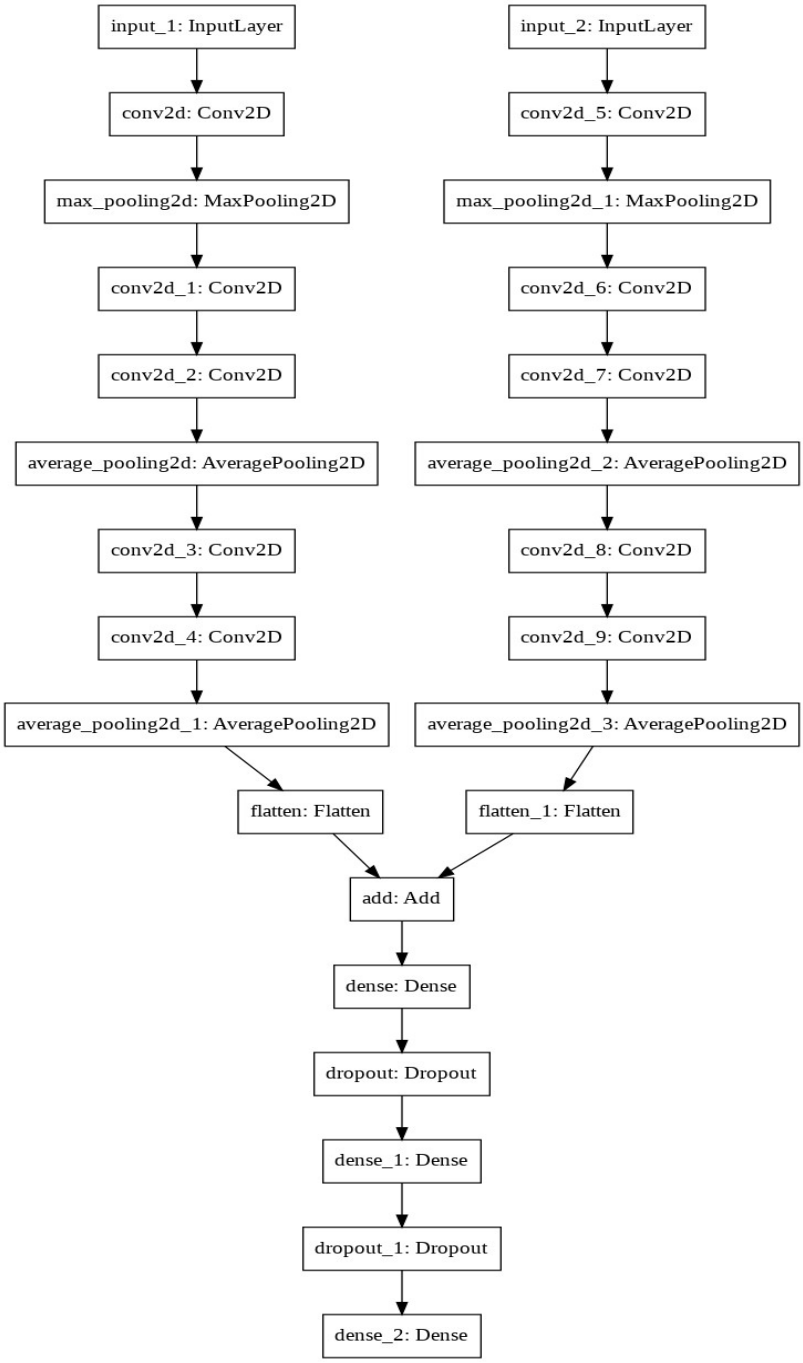
4.2.1 Baseline: Flatten + Concatenate

Two stream CNN: OF + RGB Fusion with 3 layers with **concatenation** on top of the 2 streams:



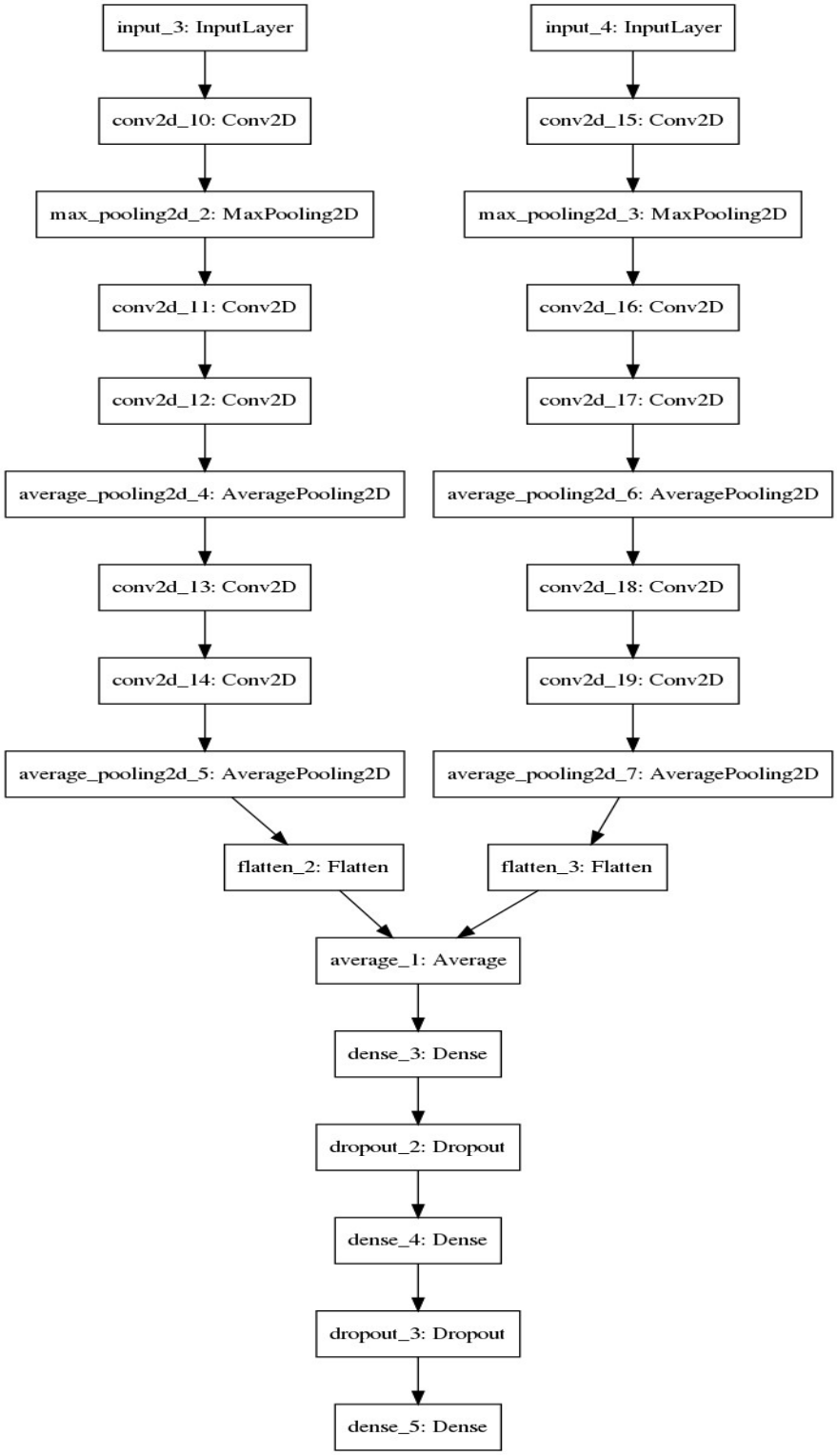
4.2.2 Flatten + Add

Two stream CNN: OF + RGB Fusion with 3 layers with **sum** on top of the 2 streams



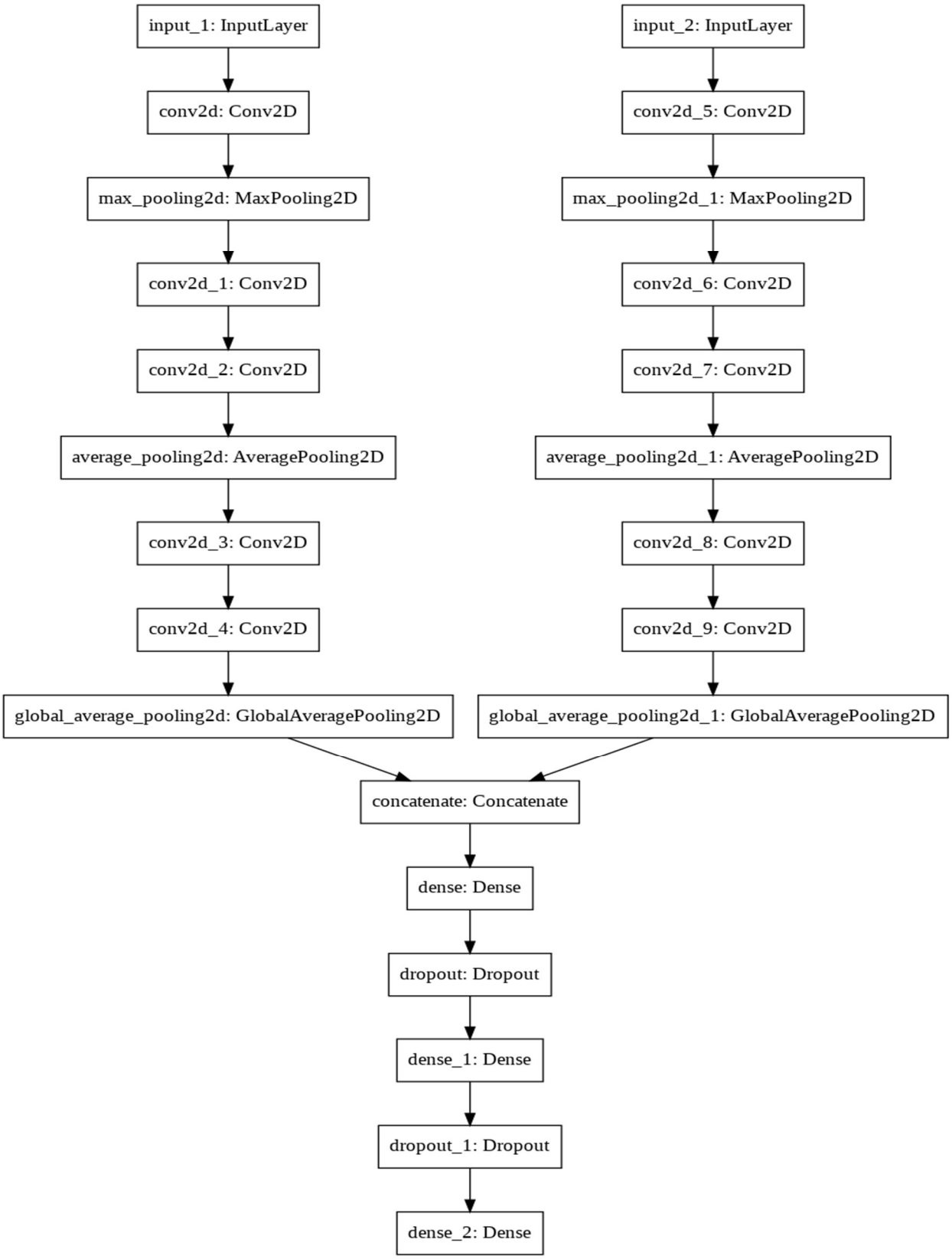
4.2.3 Flatten + Average

Two stream CNN: OF + RGB Fusion with 3 layers with **average** on top of the 2 streams



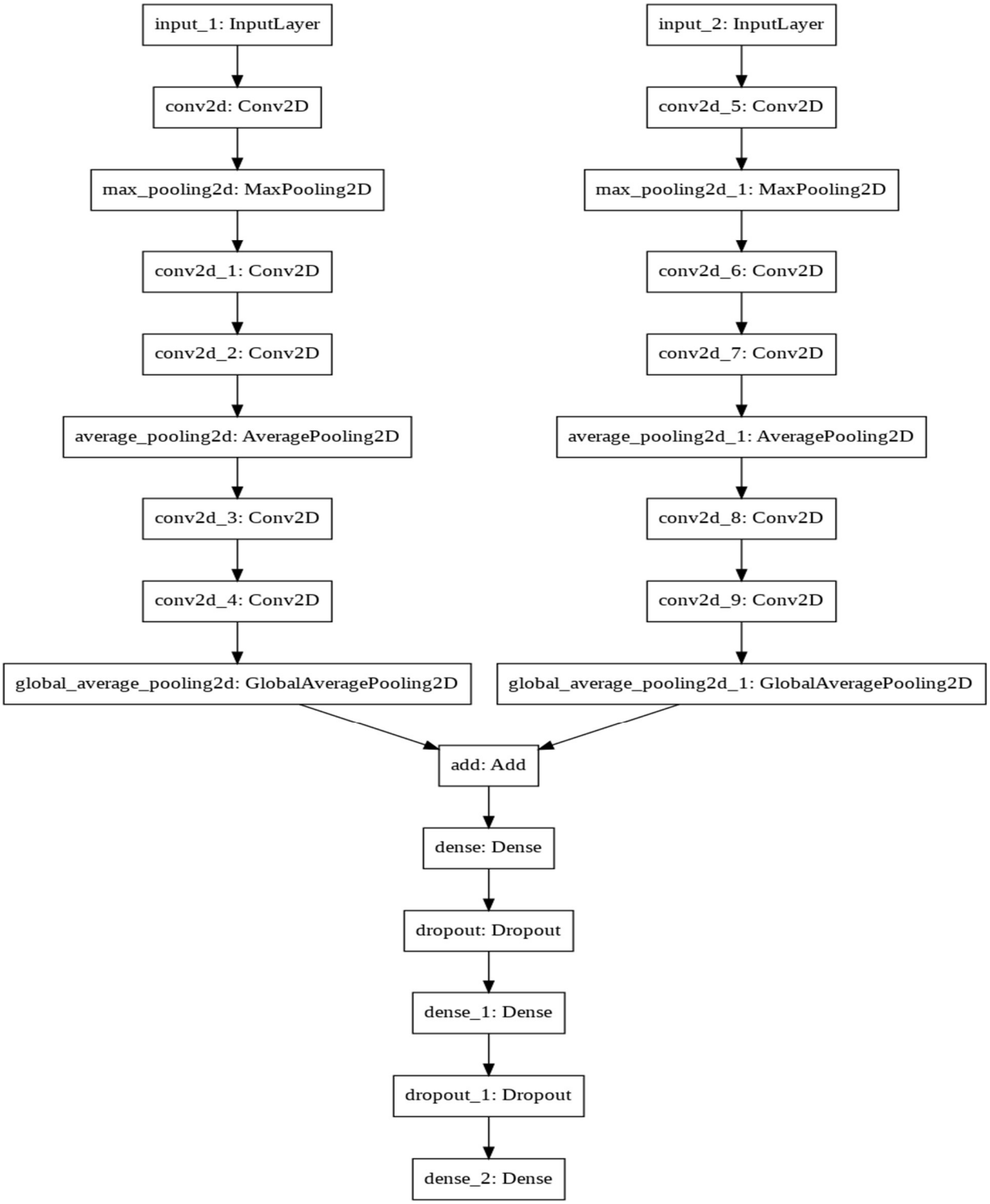
4.2.4 GAP + Concatenate

Same as Flatten + Concatenate, but using GlobalAveragePooling2D instead of Dense layer.



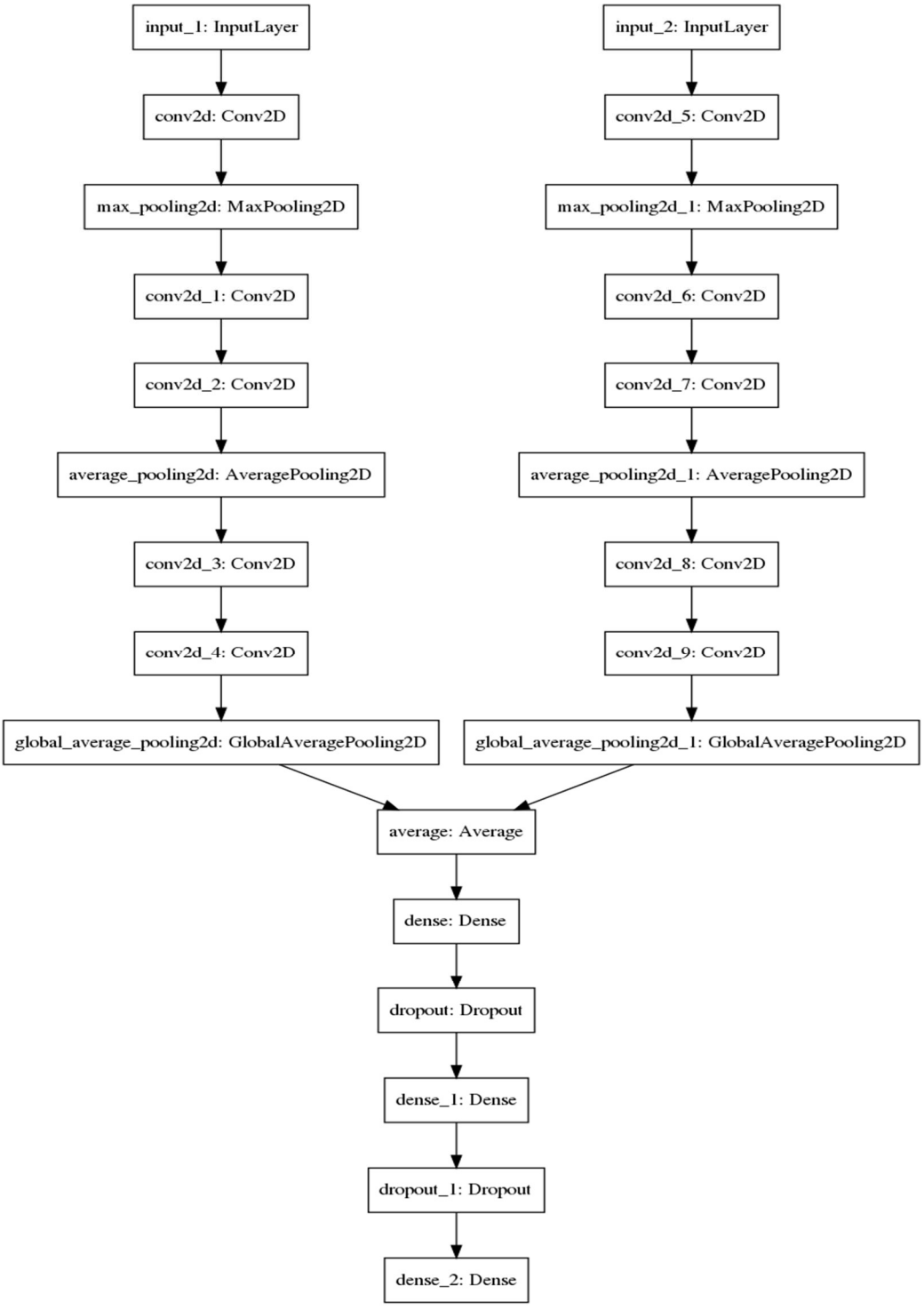
4.2.5 GAP + Sum

Same as Flatten + Sum, but using GlobalAveragePooling2D instead of Dense layer.


input_1: InputLayer
conv2d: Conv2D
max_pooling2d: MaxPooling2D
conv2d_1: Conv2D
conv2d_2: Conv2D
average_pooling2d: AveragePooling2D
conv2d_3: Conv2D
conv2d_4: Conv2D
global_average_pooling2d: GlobalAveragePooling2D
input_2: InputLayer
conv2d_5: Conv2D
max_pooling2d_1: MaxPooling2D
conv2d_6: Conv2D
conv2d_7: Conv2D
average_pooling2d_1: AveragePooling2D
conv2d_8: Conv2D
conv2d_9: Conv2D
global_average_pooling2d_1: GlobalAveragePooling2D
add: Add
dense: Dense
dropout: Dropout
dense_1: Dense
dropout_1: Dropout
dense_2: Dense

4.2.6 GAP + Average

Same as Flatten + Average, but using GlobalAveragePooling2D instead of Dense layer.



5 Experimental setup

5.1 Datasets

We used The BAUM 1a dataset [3] in our baseline experiments. At 30 fps, we got 120K frames. Operating on this amount of frames made the training very slow. Also, the model over-fits, because all the frames are very correlated.

Inspired by the approach in section III.A of [1] we divided the data in segments of 16 frames. For each segment we have 15 Dense Optical Flows (DOF) and 16 spatial images. We summarize each segment by the 1st DOF and image, and do frame skipping. This improves the training time and results. We end up having 60K frames.

Note:
We tried to get all of the datasets listed here. We only got replies from BAUM creators.

5.2 Data preparation

Loop over frames in each video to:

1. Crop the face using haarcascade face detector [7].
2. Calculate flow between each two consecutive frames
3. Save the cropped image and the flow image.

Figure 6 Sample faces detected from BAUM 1s dataset

5.2.1 Spatial images Flow

To match the problem size, we first crop the images to only the faces. We used cascaded face classifiers (following Viola Jones approach). The cropped images are then resized to 48x48x3 patches. This enables slim model size and reasonable training times.

5.2.2 Dense Optical Flow

We follow the same approach in [1] (section III. 1) to obtain the Dense Optical Flow (DOF):

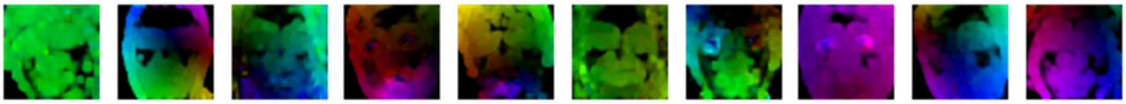


Figure 7 Sample DOF from BAUM 1s dataset

5.3 Training and model tuning

We used some tricks to regularize and smooth the model training reaching best accuracy:

5.3.1 Optimizer

We used RMSprop optimizer, with the learning rate adjustments below.

5.3.2 Reduce Learning Rate on Plateau

We used this callback in Keras to gradually reduce the learning rate when the val_acc is not improving for 2 epochs (patience). We reduce the lr with factor=2

5.3.3 Cyclic Learning Rate

We used this callback in Keras [6] to restart learning when the lr is highly reduced that the train and vall accuracies hit a dead end (none is learning). We follow the approach in [2]

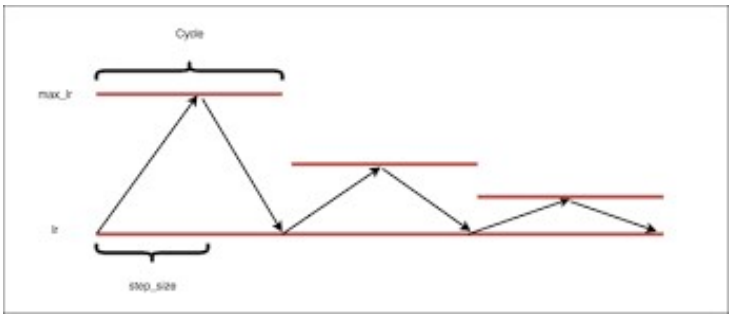


Figure 8 Traingular 2 Cyclical Learnig Rate adjustment

5.3.4 Model checkpoints

We save the best val_acc model of all the epochs.

5.3.5 Regularization

We use Dropout with probability 0.1 on Dense layers (not on Conv2D), and l2 kernel/activity regularizes, with 0.001 factor.

5.4 Evaluation protocol

Next, we tried with a mixed data of BAUM 1a and BAUM 1s [3], with balanced train and unseen test data as follows:

- Keep separate unseen test subjects in test than in train.
- Keep balanced expression classes in train and test.
- Keep whole videos as train or test, and not shuffle at the frame level, in order to avoid correlation at high fps between train and test data, which lead to information leak.

We are not sure this is the same protocol followed in the paper, since it is not mentioned. However, this is the standard approach anyways. In other words, our split could be harder than in the paper.

Since the data is already enough, we do not need to use cross validation. In addition, the train and val accuracy are stable over different runs.

We used accuracy as an evaluation metric as used in the paper. However, we do recommend using other metrics that takes into consideration class imbalance, since we noticed this issue with BAUM data. This is also reflected in the paper evaluation in the BAUM 1s confusion matrix Figure 5.

Finally, we consider only the six classes in the paper not the whole 13 classes in BAUM 1s.

6 Results

The accuracies and model sizes are shown in *Table 1*, for all fusion models:

Experiment	Details	Model size (# trainable params)	Test accuracy (%)
Baseline 1: Flatten + Concatenate	2 stream CNN: OF + RGB Fusion with 3 layers with **concatenation** on top of the 2 streams	3,961,350	57.7
Flatten + Sum	2 stream CNN: OF + RGB Fusion with 3 layers with **sum** on top of the 2 streams	3,699,206	59.77
Flatten + Average	2 stream CNN: OF + RGB Fusion with 3 layers with **average** on top of the 2 streams	3,699,206	57.71
GAP + Concatenate	Same as Flatten + Concatenate, but using GlobalAveragePooling2D instead of Dense layer.	3,961,350	59.8
GAP + Sum	Same as Flatten + Sum, but using GlobalAveragePooling2D instead of Dense layer.	3,699,206	56.48
GAP + Average	Same as Flatten + Average, but using GlobalAveragePooling2D instead of Dense layer.	3,699,206	57.66

Table 1 Fusion models sizes and results

6.1 Analysis and discussion

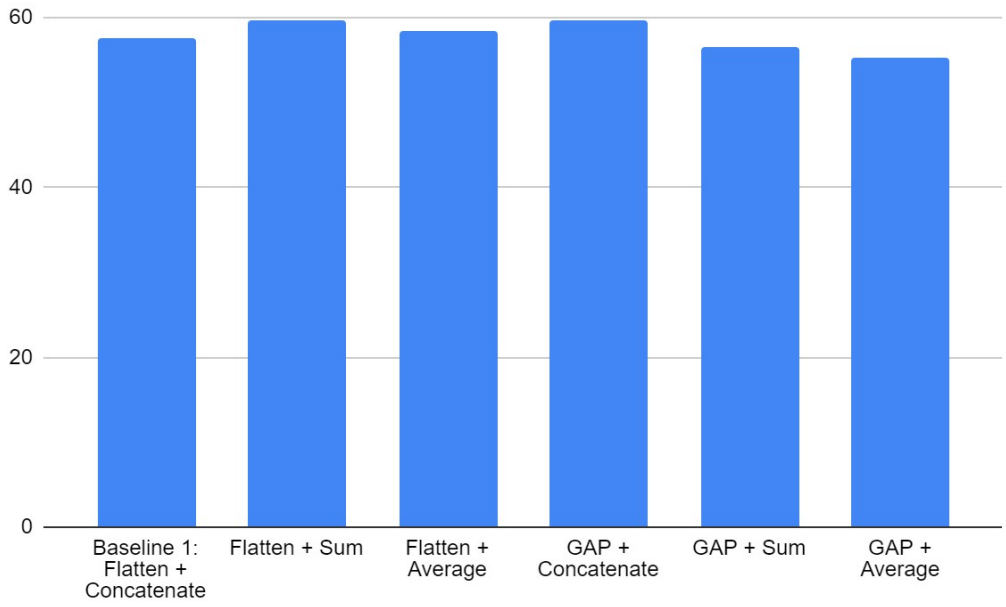


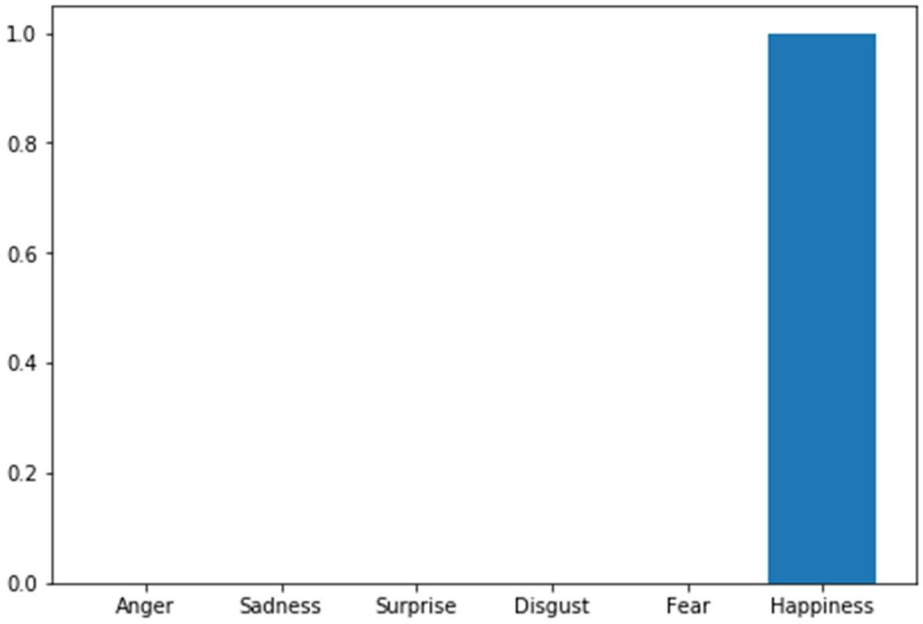
Figure 9 Models results bar chart comparison

In general, the models' performances are close to each other. Overall, careful model tuning and regularization, architecture choices and fusion choices, lead to 2-5% improvement over the paper, keeping in mind the aggressive and fair splits we did (described in data preparation above).

The Flatten+Sum and GAP+concat are the best performers. On average, we notice that GAP is performing better regarding the type of confusions (discussed below). Sum and average operations seem to give better performance on average. The conclusion here is that averaging operation helps smoothing the confusion, especially Category 1, either through explicit Average fusion operator or GAP.

Also, regarding model size, it is clear that the average and sum operators provide smaller models as appears in the 3rd column of the table above.

The output of the model is a probability distribution over the six classes as shown below




```
Video Name: S011_022 Label: Happiness
Frame: 0/142
Frame: 1/142
True: Happiness, Out: Happiness, Prob.: [1.8327164e-07 2.8490936e-08 2.1631139e-07 3.0177148e-08 3.1333666e-06
 9.9999642e-01]
```

```
Frame: 2/142
True: Happiness, Out: Happiness, Prob.: [5.4791337e-07 4.2510671e-08 5.2046607e-07 3.8287727e-08 6.4437481e-06
 9.9999237e-01]
```

```
Frame: 3/142
True: Happiness, Out: Happiness, Prob.: [1.78112145e-06 4.08525324e-07 1.63253890e-06 3.06193755e-07
 1.25788165e-05 9.99983311e-01]
```

Figure 10 Sample results for the baseline model

As we can see in Figure *11*, the train and test curves are very well regulated and smoothed. No sign of overfitting, thanks to careful regularization.

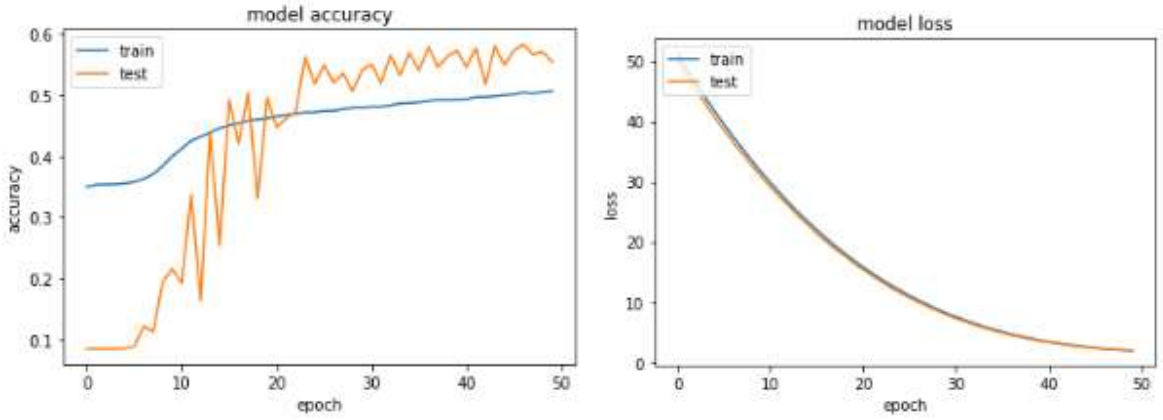


Figure 11 Loss and accuracy learning curves

6.2 Model attention maps

To make sure the model learnt the good features, we generated the Class Activation Maps, using Grad-CAM method [5]. As can be seen in *Figure 12*, the model do pay attention to the important parts of the face to generate the expression:

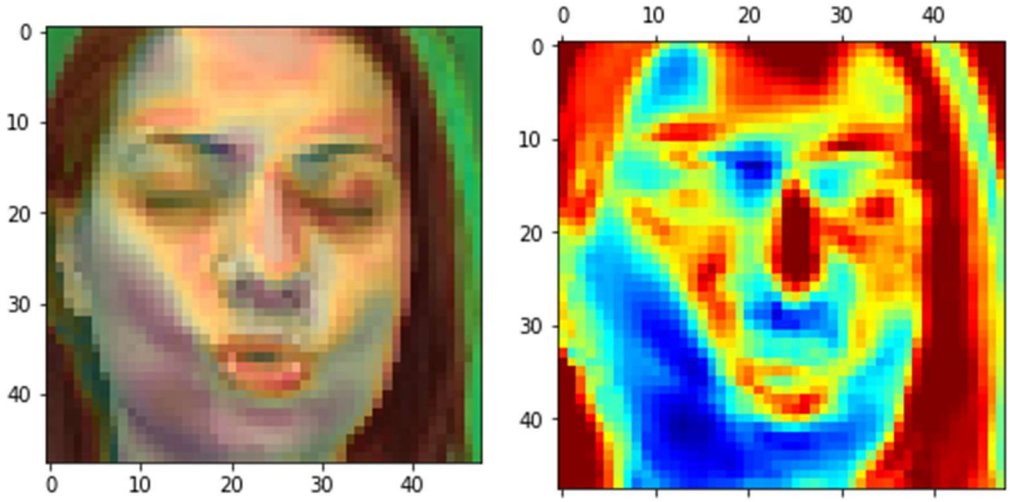


Figure 12 Attention Grad-CAM feature maps for Disgust classe. As can be seen, the model captures important class-specific facial features

6.3 Confusion matrix analysis

In the figures below, the normalized confusion matrix for each model is shown:

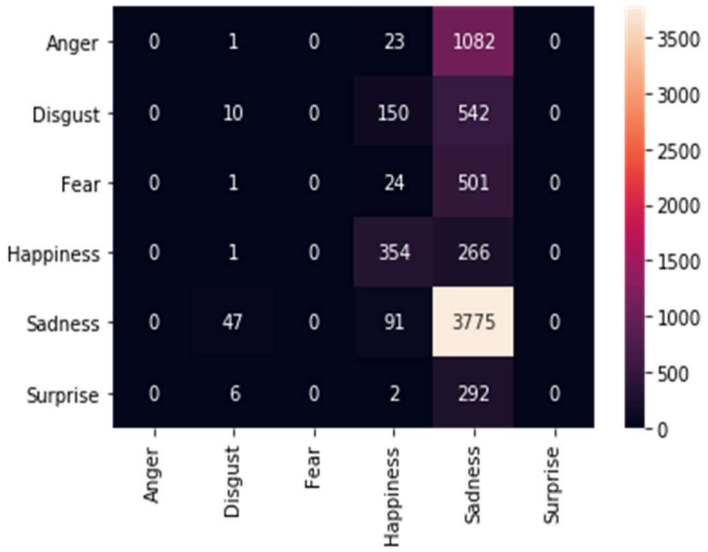


Figure 13 Baseline: Flatten + Concatenate confusion matrix

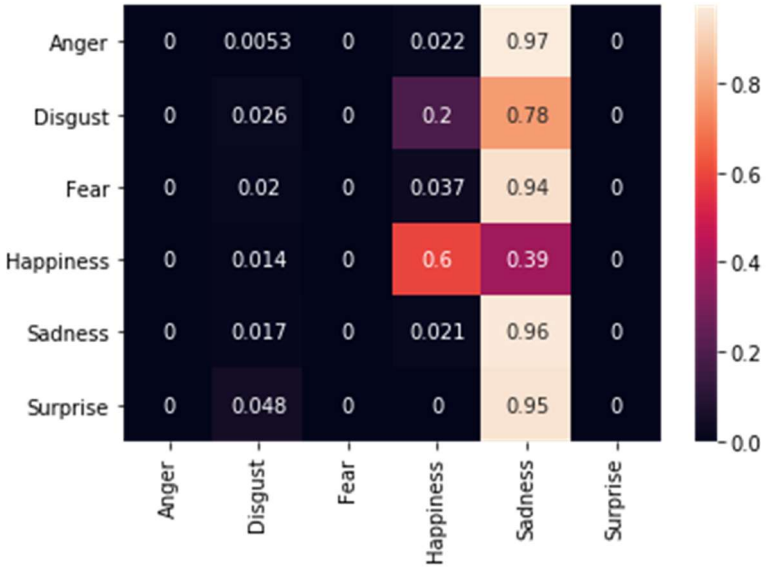


Figure 14 Flatten + Add confusion matrix

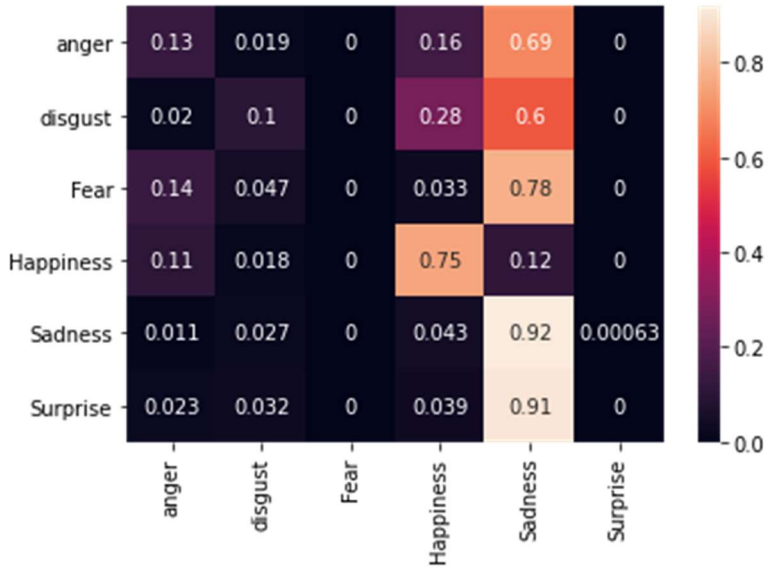


Figure 15 Flatten + Sum confusion matrix

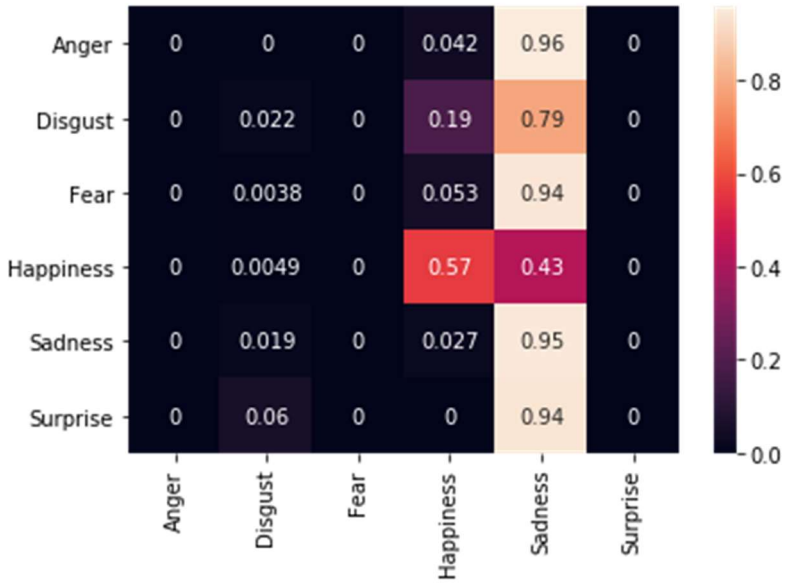


Figure 16 GAP + Concatenation confusion matrix

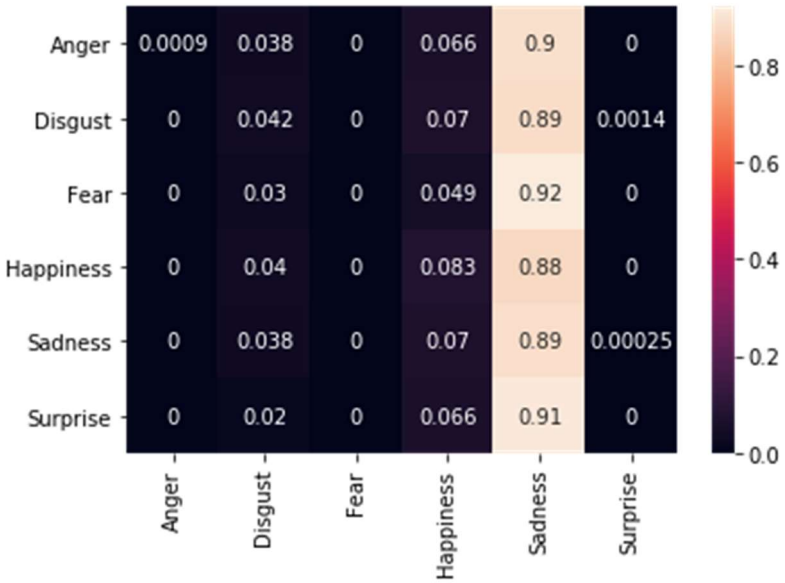


Figure 17 GAP + Sum confusion matrix

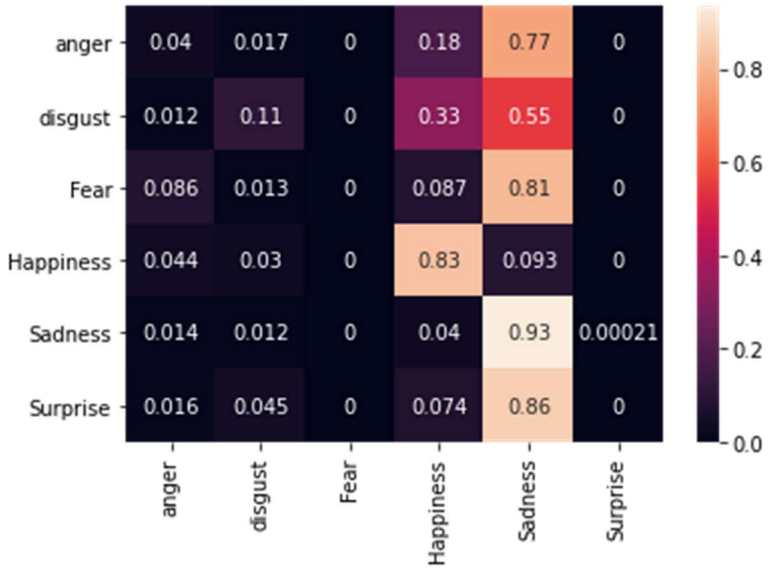


Figure 18 GAP + Average confusion matrix

We notice an issue of class imbalance; some classes are more dominant as shown in *Figure 19*. The same is noticed in the confusion matrices above, and in the original paper in [1]

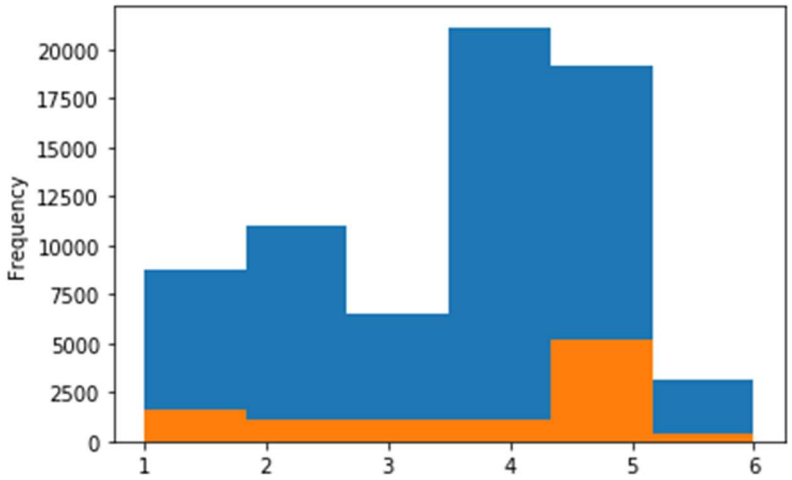


Figure 19 Classes distribution histogram. Blue: original BAUM 1s data, Orange: Down-sampled BAUM 1s data

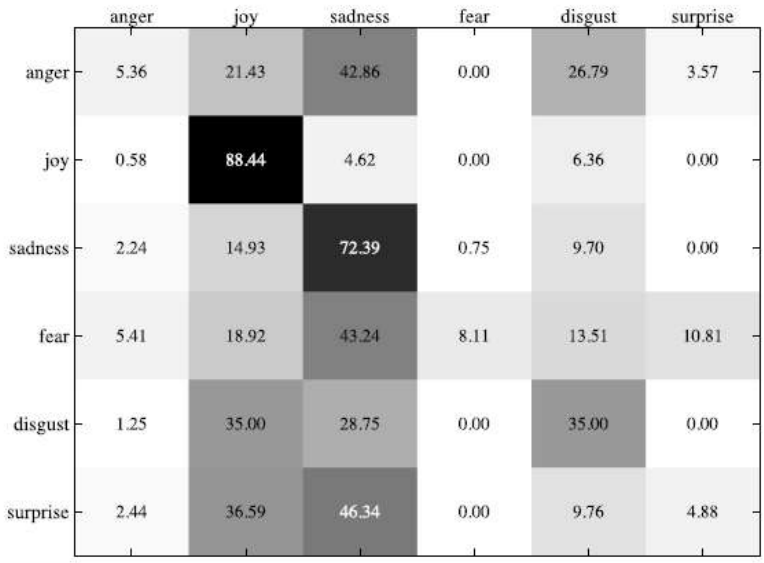


FIGURE 5. Confusion matrix of recognition results with DBNs on the BAUM-1s dataset.

Figure 20 Confusion matrix from shows the same sign of imbalanced confusions towards the dominant sadness class

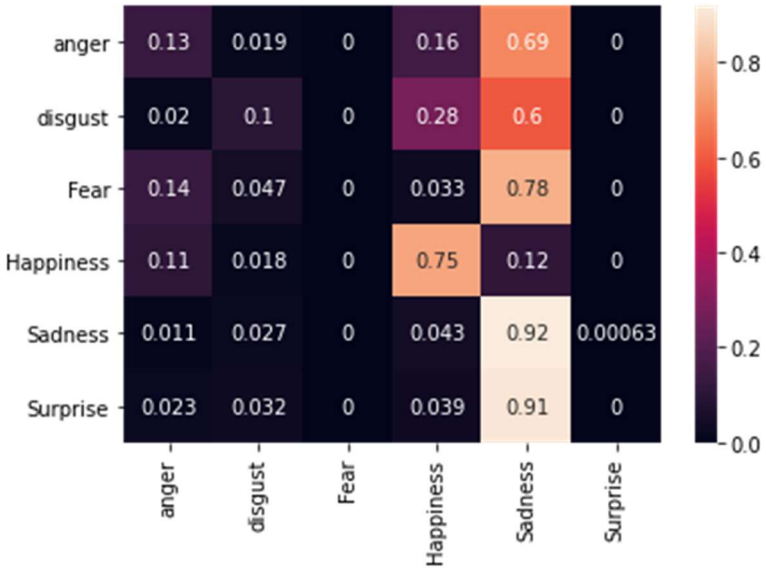
We notice dominance of the sadness class (4), which affects the final accuracy and make the classifier biased. This makes accuracy not the good metric. We need to use precision, recall or F1 (F1 Beta), that will reflect the issue better. You may notice that, some models give high accuracy, while their confusion matrix analysis shows high bias to Sadness class.

Most confusions are:

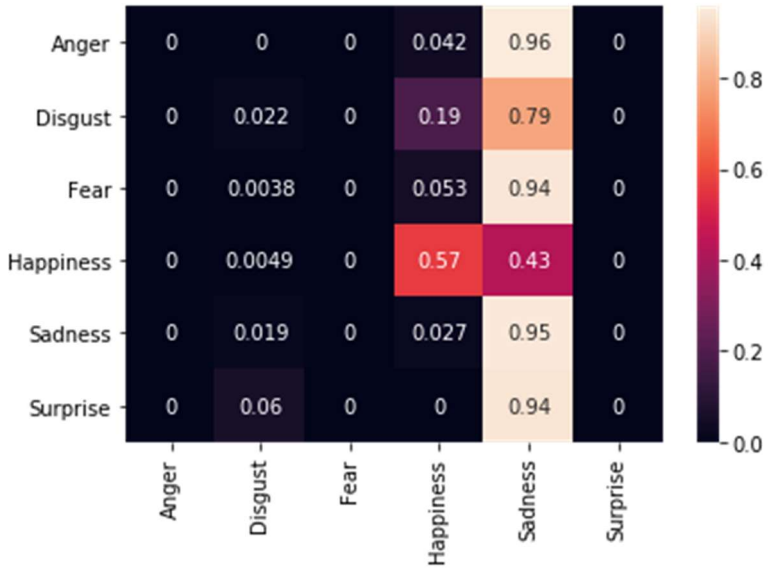
- **Category 1:** Happiness/Fear and sadness: probably due to mislabels. But to confirm, we check other models. This is the most severe confusion and not justified.
- **Category 2:** Anger/Disgust/Fear to sadness: it makes sense in light of the learnt feature maps. It can also be due to subtle mislabels. This is less severe confusions since the expressions are close anyways.

We can see in better fusion techniques models below, the heatmaps of confusion are slightly better distributed. For example, we start to see less severe confusion from **Category 1,** especially Happiness - Sadness confusions are more disentangled. The conclusion here is that averaging operation helps smoothing the confusion, especially Category 1, either through explicit Average fusion operator or GAP.

Flatten + Average



GAP + Concatenate



Possible treatments of this issues:

- Downsampling (orange smoothed histogram above): we tried this, but it highly reduces the training data, resulting in bad performance
- Upsampling (SMOTE)
- Outlier/anomaly detection
- One vs all classification of minor/major classes followed by minor classes distinction.
- Cost-sensitive loss: class weighting in keras

7 Conclusion

In this work, we extend the baseline FER spatiotemporal approach in S. Zhang et al.: Learning Affective Video Features for FER via Hybrid Deep Learning, with careful fusion techniques study. We study the effect of fusion operation and fusion layer type, and their combinations. Moreover, we optimize the used model and carefully regularize it, using Dropout, L2 Regularization and Cyclical learning rate tuning. We also optimize the architecture and model input. We achieve 2-5% overall accuracy improvement over the baseline approach, with the suggested improvements. The achieved results are achieved with fairer train/test split and careful evaluation protocol that considers even subject, expression and video splits. In addition, we extend our work to BAUM 1a dataset, which was not tested in the paper.

Moreover, we closely analyze the confusion matrix of BAUM dataset, which reflects a high imbalance toward sadness class. Some of our fusion techniques already improved this confusion, specially the most severe or unjustified confusions. The conclusion here is that averaging operation helps smoothing the confusion, especially Category 1, either through explicit Average fusion operator or GAP. We also tried downsizing the data to the lowest classes, but this reduced the training data and gave bad results. In the future work, we discuss some suggested treatments.

Finally, we visualize the learnt feature maps, using Grad-CAM method, to see how reasonable are the learnt features, which shows good sign of generalization.

8 Future works

Below are some possible extensions that are expected to further improve over the method in the paper (ordered in priority):

1. **Treat the class imbalance:**
    a. Downsampling: we tried this, but no good results
    b. Upsampling: using SMOTE methods for examples, and/or data augmentation
    c. One class vs. all (outlier detection pipeline)
    d. Class weighting
2. **Frame stacking:** stack different frames as CNN channels instead of Optical Flow. Could also be the 3rd stream. This can be done using the TimeDistributed layer in Keras.
3. **Transfer learning from other tasks:** train on similar task (ex: face recognition, activity recognition,...etc) and transfer the encoder weights to FER
    1. Face recognition on bigger datasets
    2. Static FER
4. **Fine tune different backends:** use pre-trained nets on normal image classification (say image net), and fine tune (after warm-up) to FER. Possible backends are:
    1. VGG16/19
    2. ResNet50
5. **3D CNN:** stack different frames as video frames instead of Optical Flow and use 3D Convolutions. Could also be the 3rd stream.
6. **Recurrent NN:** Use the spatial frames as inputs to spatial CNN encoder, and plug recurrent decoders like LSTM or ConvLSTM on top to track the temporal features, followed by the softmax classification layer. Possible recurrent decoders are:
    1. LSTM
    2. ConvLSTM

9 References

[1] Zhang, S., Pan, X., Cui, Y., Zhao, X. and Liu, L., 2019. Learning affective video features for facial expression recognition via hybrid deep learning. *IEEE Access*, *7*, pp.32297-32304.

[2] Smith, L.N., 2017, March. Cyclical learning rates for training neural networks. In 2017 IEEE Winter Conference on Applications of Computer Vision (WACV) (pp. 464-472). IEEE.

[3] S. Zhalehpour, O. Onder, Z. Akhtar, and C. E. Erdem, "BAUM-1: A spontaneous audio-visual face database of affective and mental states," IEEE Trans. Affective Comput., vol. 8, no. 3, pp. 300_313, Jul./Sep. 2016.

[4] Lin, M., Chen, Q. and Yan, S., 2013. Network in network. arXiv preprint arXiv:1312.4400.

[5] Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. and Batra, D., 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE international conference on computer vision (pp. 618-626).

[6] https://keras.io/

[7] https://docs.opencv.org/3.4/d7/d8b/tutorial_py_face_detection.html

Appendix

How to run

1. Data preparation

- Download and unzip BAUM 1 dataset in a home_dir
- Download and unzip face_detectors in home_dir
- In FER_data_preparation.ipynb set baum_dir to your BAUM directory
- Run FER_data_preparation.ipynb. This will generate csv file (data.csv) and two folders: imgs_spatial and imgs_flows as described in the data preparation section

2. Training

- Set baum_dir in FER_fusion.ipynb to the data directory in the previous step
- Run FER_fusion.ipynb. This will save a model models/Model_fusion.h5

3. Inference

- Set baum_dir in FER_BAUM Baseline.ipynb
- Set model_path = os.path.join(baum_dir, 'models', 'Model_fusion.h5')
- Run FER_BAUM Baseline.ipynb