# Fusion approaches for spatiotemporal Face Expression Recognition (FER)

Abstract ..... 3
1 Introduction ..... 4
2 Related work ..... 4
3 Spatio-temporal FER ..... 4
4 Baseline model ..... 5
5 Fusion methods ..... 9
5.1 Fusion operators ..... 9
5.2 Fusion layers ..... 10
6 Experimental setup ..... 17
6.1 Datasets ..... 17
6.2 Data preparation ..... 17
6.3 Training and model tuning ..... 18
6.4 Evaluation protocol ..... 19
7 Results ..... 19
7.1 Analysis and discussion ..... 20
7.2 Model attention maps ..... 22
7.3 Confusion matrix analysis ..... 22
8 Conclusion ..... 28
9 Future works ..... 28
10 References ..... 29
**Appendix** ..... 30
1. **Data preparation** ..... 30
2. **Training** ..... 30
3. **Inference** ..... 30

## 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 analyze 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 Related work

**Commented [AE1]:** Related work added
Paper approach described separately

Facial Expression Recognition (FER) approaches can be divided into dynamic and static. Static approaches are mainly based on single image features (usually RGB), see [6]-[9], where faces are first cropped, then a CNN is applied to extract useful features before given to a classifier, like SVM, MaxEnt,…etc. Such approaches obviously lack to capture the useful temporal features that can be obtained from the successive video frames. On the other hand, dynamic approaches [1] treat the problem as sequence of correlated events. The main challenge of dynamic approaches is how to efficiently encode the video sequence to capture the temporal aspect. In [10], 3D CNNs are used to encode the stacked video frames as 3D tensor.

# 3 Spatio-temporal FER

A recent approach that fuses both static and dynamic aspects is presented in [1]. This approach follows a two stream CNN fusion architecture. The spatial RGB stream is encoded via stacked CNN+MaxPool layers, followed by a flatten and Dense layer to provide a 4096 feature vector. The input to this stream is a 150x110x3 face cropped RGB image. The temporal stream is a Dense Optical Flow (DOF) image, encoded over 2 time steps RGB images. The DOF size is 227x227x3. The temporal CNN follows the same architecture of the spatial stream. The 2 stream vectors (both 4096 dimensions tensor) are then fused via a simple concatenation layer, followed by a DBN network, and finally an SVM classifier to provide the class labels over the six facial expressions.

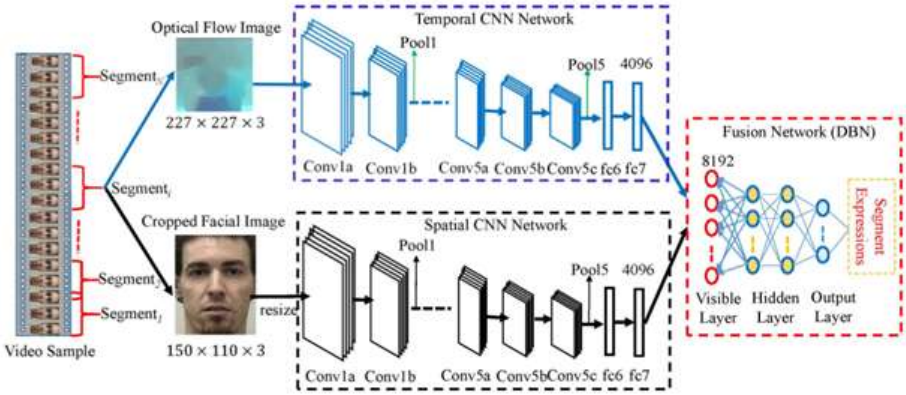



Figure 1 Spatio-temporal approach as in [1]

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

# 4 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)

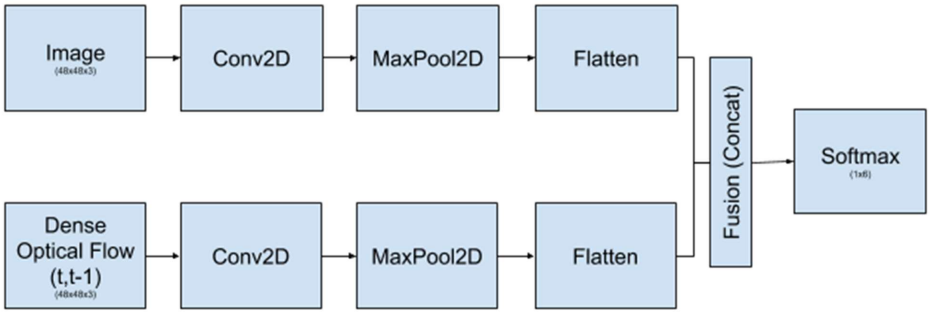


Figure 3 Baseline spatio-temporal model

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 are 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
```

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

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

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

# 5 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:

## 5.1 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.

5.2 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 denser vector than normal flatten

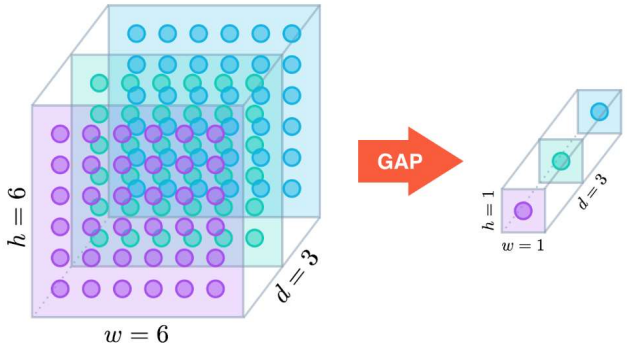



Figure 4 Global Average Pooling

The GAP layer transforms the dimensions by performing channel wise average as shown in Figure 4. Global Average Pooling has the following advantages over the fully connected layers:

- It has much less number of trainable parameters. It can be viewed as the fully connected layers, but with preset weights of the average. This makes the model less prone to overfitting.
- According to [4], averaging makes the network more robust to spatial translations in the data.

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

**Commented [m2]:** Can you paraphrase this part, it matches exactly to the article text on the website https://adventuresinmachinelearning.com/global-average-pooling-convolutional-neural-networks/

**Commented [AE3R2]:** Yes you are right. I paraphrased it.

### 5.2.1 Baseline: Flatten + Concatenate

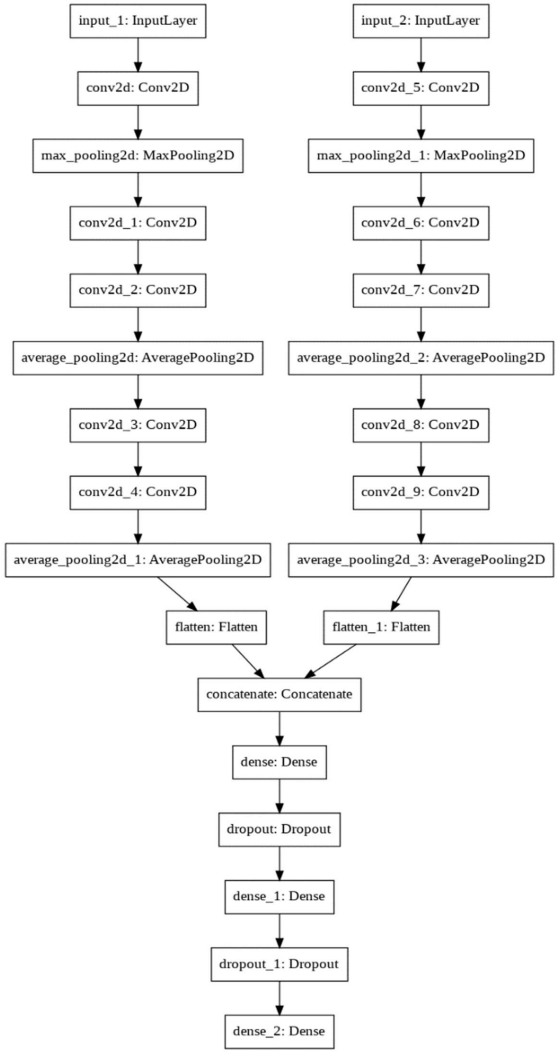



Figure 5 Two stream CNN: OF + RGB Fusion with 3 layers with concatenation on top of the 2 streams

### 5.2.2 Flatten + Add

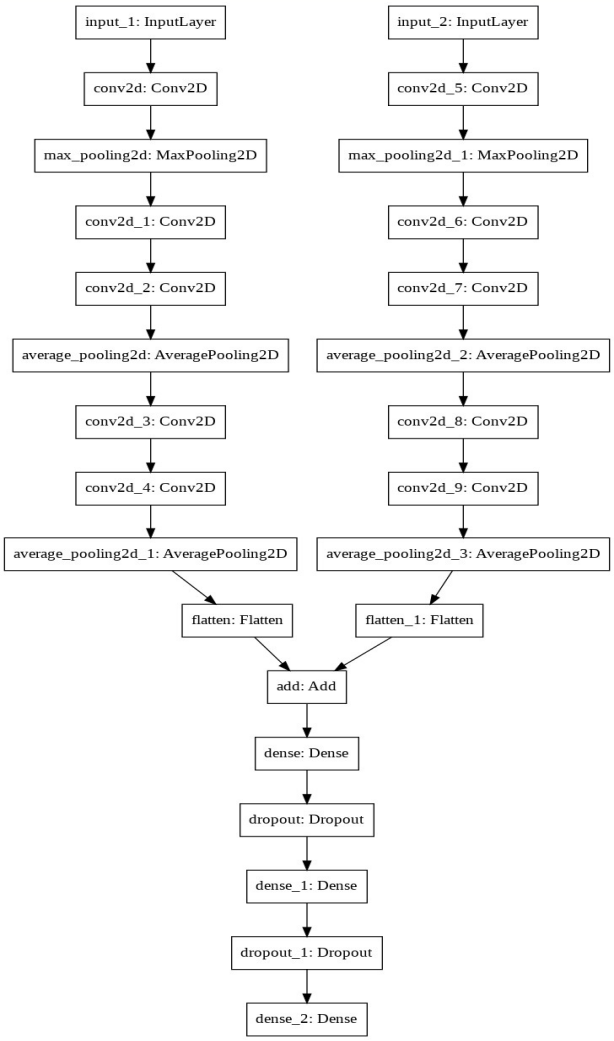



Figure 6 Two stream CNN: OF + RGB Fusion with 3 layers with sum on top of the 2 streams

### 5.2.3 Flatten + Average

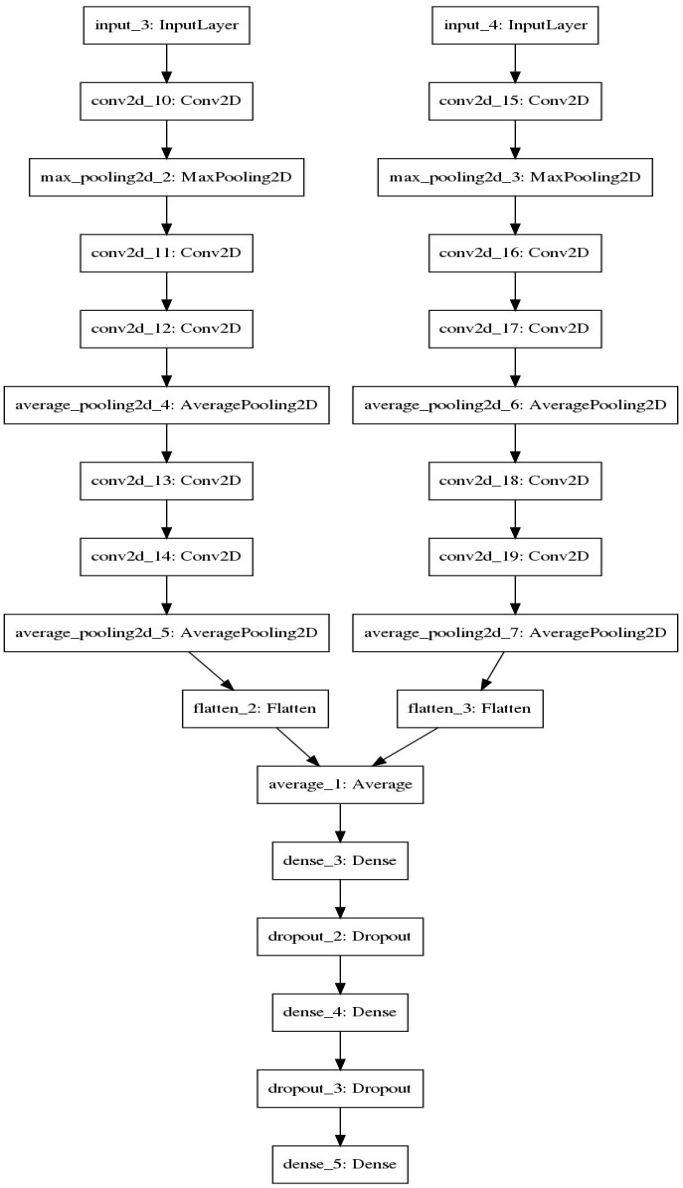



Figure 7 Two stream CNN: OF + RGB Fusion with 3 layers with average on top of the 2 streams

### 5.2.4 GAP + Concatenate

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

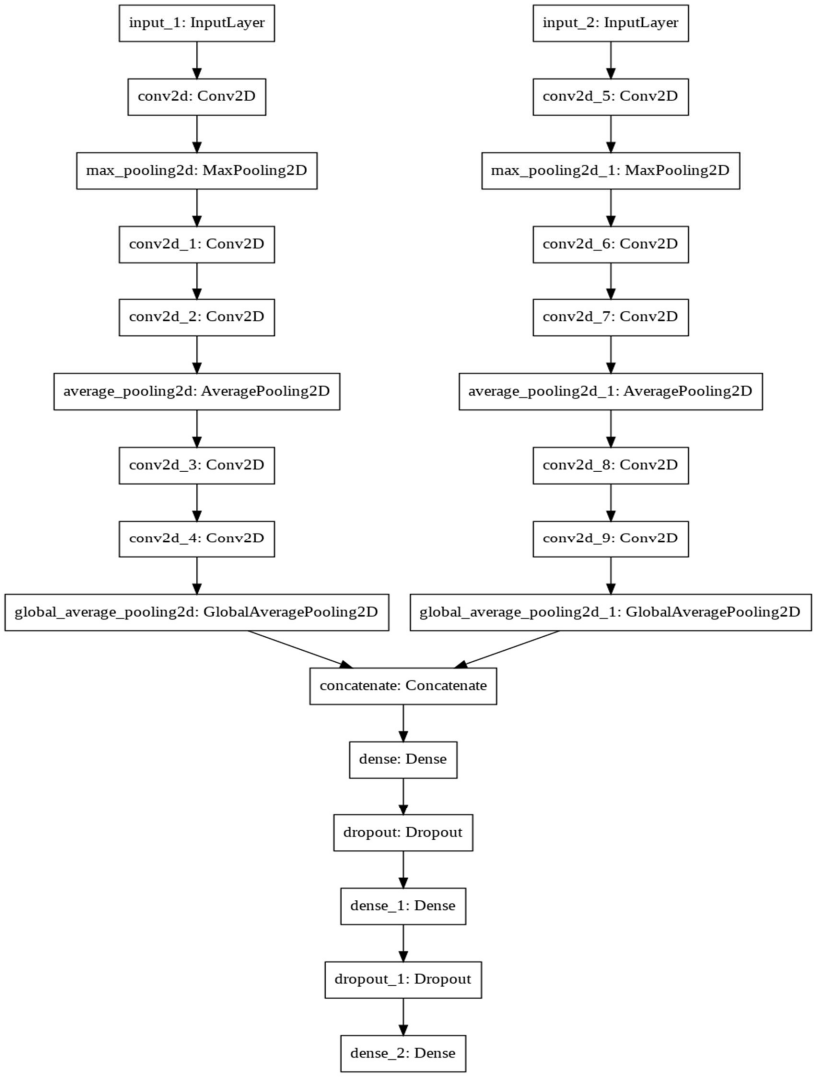



Figure 8 GAP + Concatenate

### 5.2.5 GAP + Sum

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

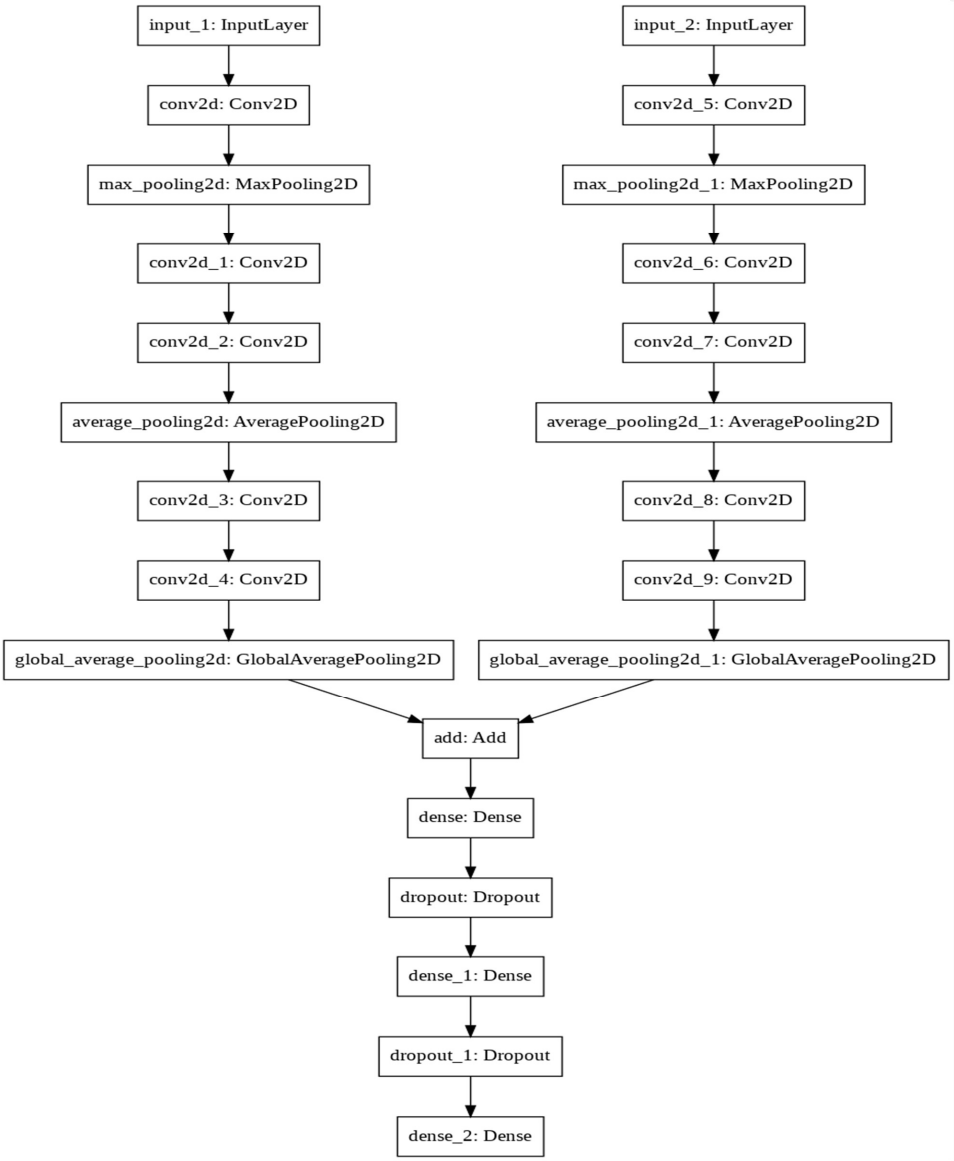



Figure 9 GAP + Sum

### 5.2.6 GAP + Average

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

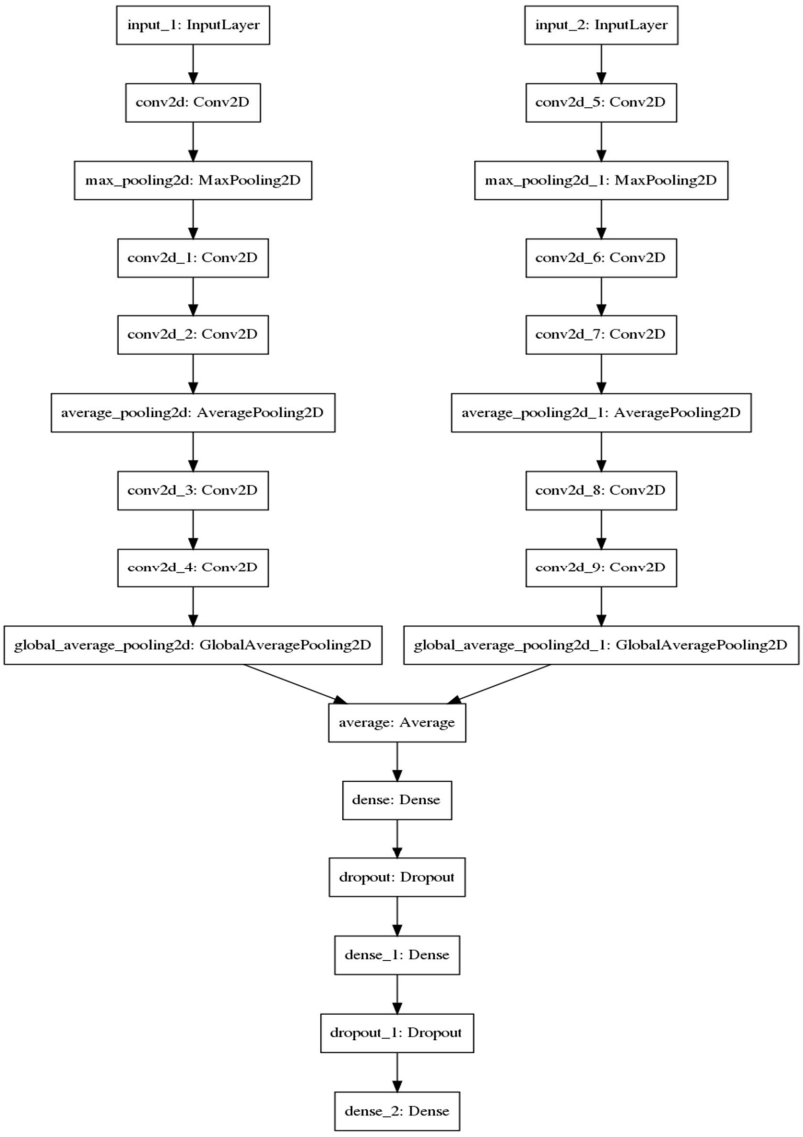



Figure 10 GAP + Average

# 6 Experimental setup

## 6.1 Datasets

We used The BAUM 1a dataset [3] in our baseline experiments. At 30 fps, we got 120K frames. Operating on these 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.

## 6.2 Data preparation

Loop over frames in each video to:

1. Crop the face using haarcascade face detector [12].
2. Calculate flow between each two consecutive frames
3. Save the cropped image and the flow image.

Figure 11 Sample faces detected from BAUM 1s dataset

### 6.2.1 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.

### 6.2.2 Dense Optical Flow

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

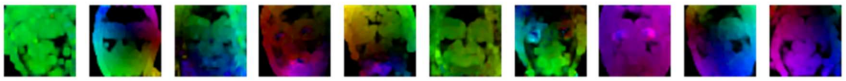


Figure 12 Sample DOF from BAUM 1s dataset

## 6.3 Training and model tuning

The following are some methods used to regularize and smooth the model training reaching best accuracy:

### 6.3.1 Optimizer

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

### 6.3.2 Reduce Learning Rate(LR) on Plateau

We used ReduceLROnPlateu callback in Keras [11] to gradually reduce the learning rate when the validation accuracy is not improving for 2 epochs (patience). We reduce the learning rate with factor=2

**Commented [m4]:** Didn't understand "this"

**Commented [AE5R4]:** Added explicitly the call back name

### 6.3.3 Cyclic Learning Rate

A common symptom of quick overfitting is when the learning rate is too high. To overcome this, we usually reduce the learning rate gradually, which could hit very small learning rate that the network is hardly learning anything. Another approach is to start with small learning rate, and increase it gradually, which might suffer the same effect of overfitting. To combine both approaches cyclical learning rate schedules can be used (see [2]). The main idea is to periodically restart the learning rate between two extremes to capture the advantages of both worlds.

**Commented [m6]:** Didn't understand the sentence below.

**Commented [AE7R6]:** Added description of cyclical learning rate, with reference to the paper

We used CyclicLearningRate callback in Keras [11] to restart learning when the LR is highly reduced that the train and validation accuracies hit a dead end (none is learning). We follow the approach in [2].

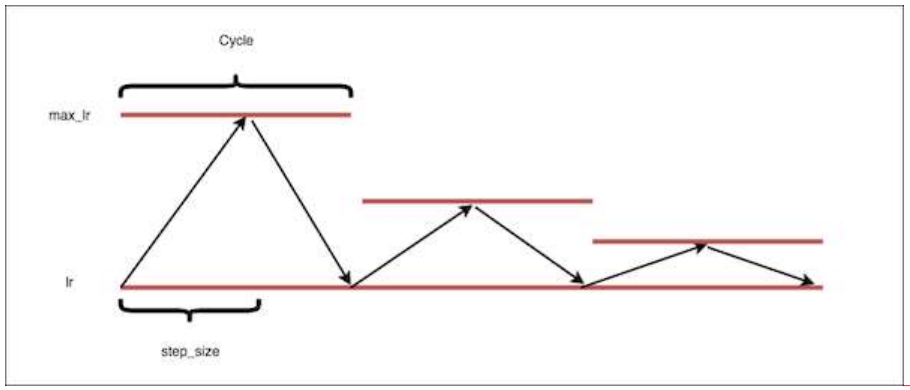



Figure 13 Triangular 2 Cyclical Learning Rate adjustment

**Commented [AE8]:** Figure made clearer

### 6.3.4 Model checkpoints

We save the best validation accuracy model of all the epochs.

### 6.3.5 Regularization

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

## 6.4 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.

The paper (S.Zhang, et.al, 2019 [1]) doesn't mention if the above protocol has been implemented. 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 20). Finally, we consider only the six classes in the paper not the whole 13 classes in BAUM 1s.

**Commented [m9]:** Citation

**Commented [m10]:** Does 'this' means class imbalance?

# 7 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<br>Fusion with 3 layers with **concatenation** on top of the 2 streams | 3,961,350 | 57.7 |
| **Flatten + Sum** | 2 stream CNN: OF + RGB<br>Fusion with 3 layers with **sum** on top of the 2 streams | 3,699,206 | 59.77 |
| **Flatten + Average** | 2 stream CNN: OF + RGB<br>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

## 7.1 Analysis and discussion

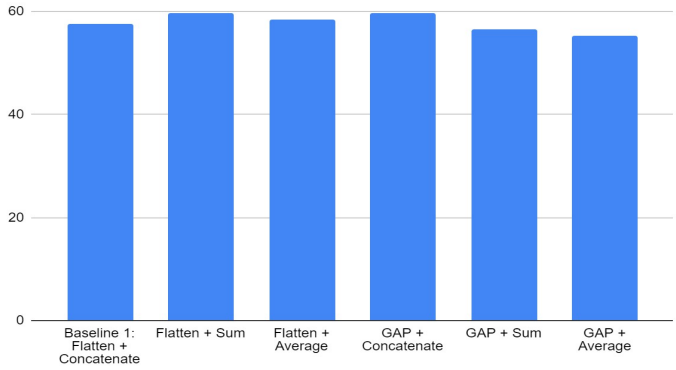
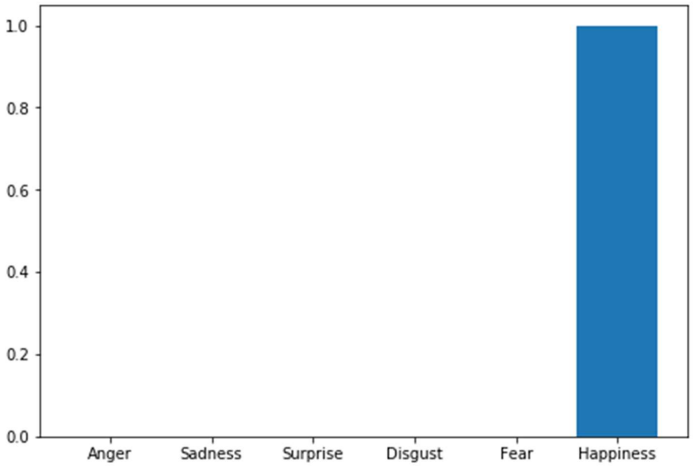



Figure 14 Models results bar chart comparison

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 method used in [1], 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, 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 15 Sample results for the baseline model

As we can see in Figure 16, the train and test curves are very well regulated and smoothed. No sign of overfitting, thanks to careful regularization.

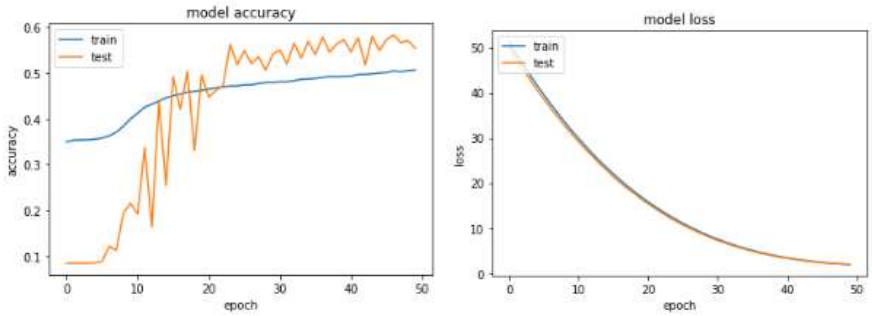



Figure 16 Loss and accuracy learning curves

## 7.2 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 17*, the model do pay attention to the important parts of the face to generate the expression:

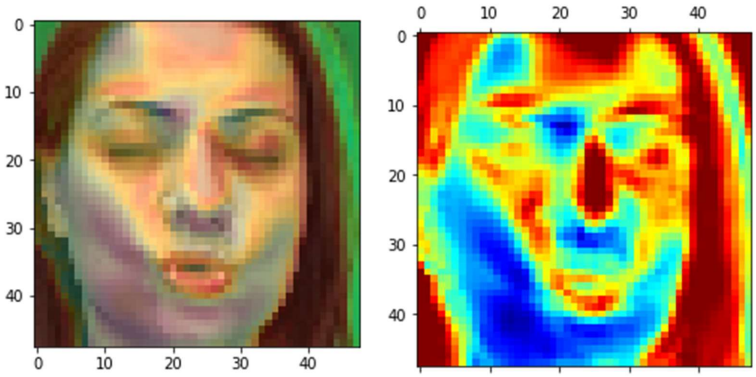


Figure 17 Attention Grad-CAM feature maps for Disgust classes. As can be seen, the model captures important class-specific facial features

## 7.3 Confusion matrix analysis

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

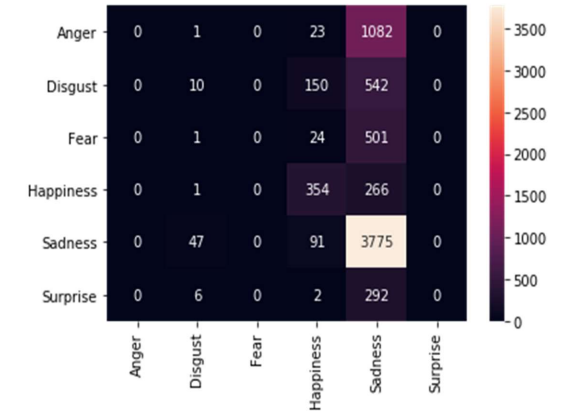



Figure 18 Baseline: Flatten + Concatenate confusion matrix

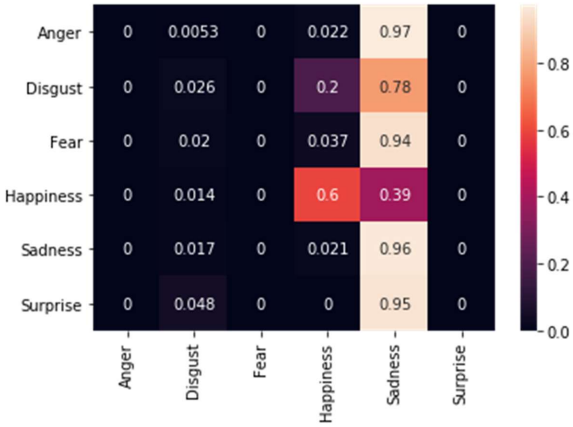



Figure 19 Flatten + Add confusion matrix

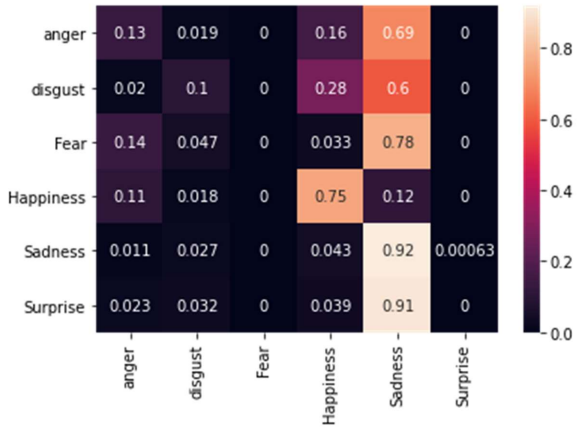



Figure 20 Flatten + Sum confusion matrix

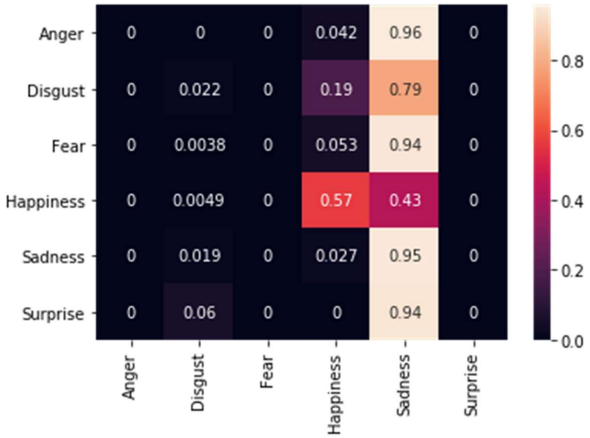



Figure 21 GAP + Concatenation confusion matrix

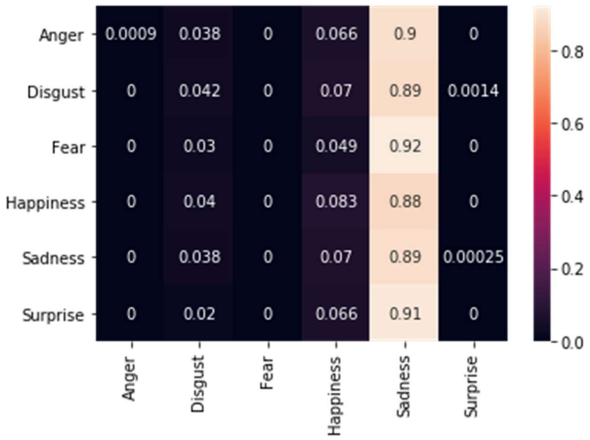



Figure 22 GAP + Sum confusion matrix

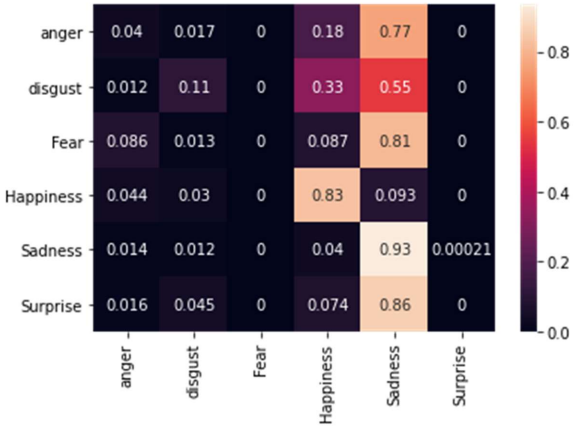



Figure 23 GAP + Average confusion matrix

We noticed an issue of class imbalance; some classes are more dominant as shown in *Figure 24*. The same is noticed in the confusion matrices above, and in the original paper in [1]

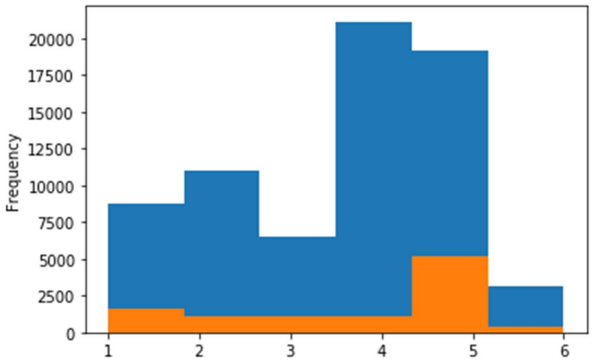



Figure 24 Classes distribution histogram. Blue: original BAUM 1s data, Orange: Down-sampled BAUM 1s data

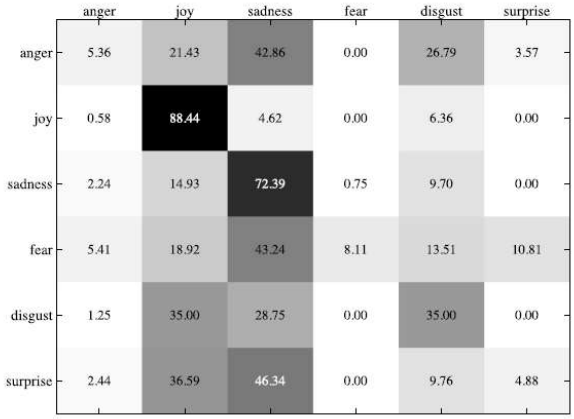



**FIGURE 5. Confusion matrix of recognition results with DBNs on the BAUM-1s dataset.**

Figure 25 Confusion matrix from shows the same sign of imbalanced confusions towards the dominant sadness class [1]

**Commented [m11]:** Citation of figure

**Commented [AE12R11]:** Ref added

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.

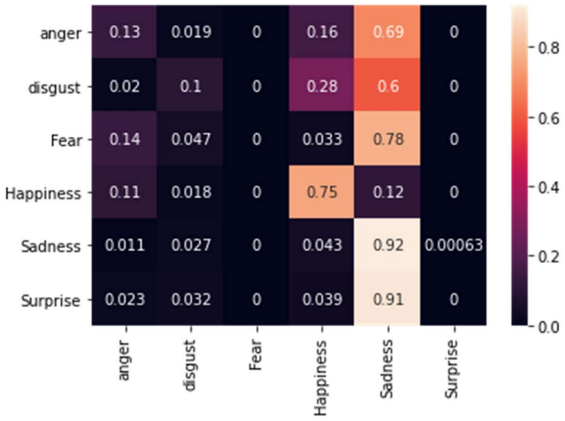
Most confusions are:

- **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.

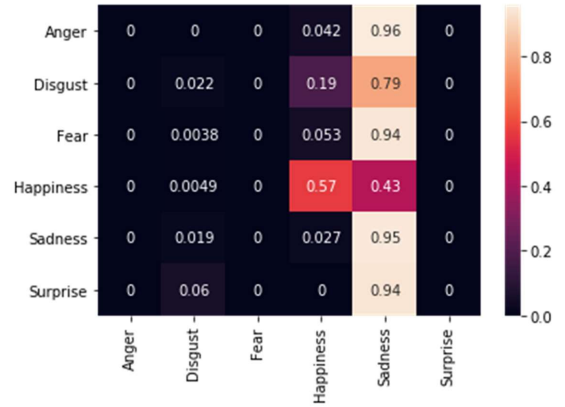
- **Category 2:** Anger/Disgust/Fear to sadness: it makes sense considering the learnt feature maps. It can also be due to subtle mislabels. This is less severe confusions since the expressions are close anyways.

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.

# 8 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 made with fairer train/test split and careful evaluation protocol that considers even subject, expression and video splits. Also, 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 the learnt features are, which shows good sign of generalization.

# 9 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

# 10 References

Commented [m13]: Also add reference paper of "Harr cascade classifier".


[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] B. Martinez, M. F. Valstar, B. Jiang, and M. Pantic, ``Automatic analysis of facial actions: A survey," IEEE Trans. Affective Comput., to be published. doi: 10.1109/TAFFC.2017.2731763.

[7] X. Zhao and S. Zhang, ``A review on facial expression recognition: Feature extraction and classi_cation," IETE Tech. Rev., vol. 33, no. 5, pp. 505_517, 2016.

[8] C. A. Corneanu, M. O. Simón, J. F. Cohn, and S. E. Guerrero, ``Survey on RGB, 3D, hermal, and multimodal approaches for facial expression recognition: History, trends, and ffect-related applications," IEEE Trans. Pattern Anal. Mach. Intell., vol. 38, no. 8, pp. 548_1568, Aug. 2016.

[9] E. Sariyanidi, H. Gunes, and A. Cavallaro, ``Automatic analysis of facial affect: A survey of registration, representation, and recognition," IEEETrans. Pattern Anal. Mach. Intell., vol. 37, no. 6, pp. 1113_1133, Jun. 2015.

[10] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, ``Learning spatiotemporal features with 3D convolutional networks," in Proc. IEEE Int. Conf. omput. Vis. (ICCV), Santiago, Chile, Dec. 2015, pp. 4489_4497.

[11] https://keras.io/

[12] 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