Computer Vision

Assignment 2 - Deep Learning

Parmenion Koutsogeorgos - i6328191
Ioannis Grepsios - i6331461

Contents

1 Introduction **1**

2 Dataset **2**
2.1 Train/validation/test sets 3
2.2 Data pre-processing 3

3 CNN model architectures **3**

4 Training **6**
4.1 Methodology 6
4.2 Results 7
4.2.1 5-fold cross-validation results 7
4.2.2 Training the chosen architecture 10

5 Testing **10**
5.1 Testing on real-life videos 11
5.2 Comparison with ResNet50 12

6 Visualizing the activations **15**

1 Introduction

In this assignment we use Convolutional Neural Networks (CNNs) for emotion recognition. The goal is to train a CNN-based model in order to classify facial expressions by the emotions they express. We try 3 different architectures which are compare using k-fold cross-validation ($k = 5$) on a subset of the dataset. We also compare the performance of the most promising architecture among the above with a pre-trained and fine-tuned ResNet50 architecture. Finally, we test our architecture on 3 real-life videos which show different emotions.

2 Dataset

For this assignment we use the FER2013 dataset[1]. It consists of 35,887 grayscale images, each with a resolution of 48x48 pixels. The dataset is labeled with one of seven basic emotions: anger, disgust, fear, happiness, sadness, surprise, or neutral. The images in the FER2013 dataset capture facial expressions of individuals from various demographics and age groups, with different facial features, lighting conditions, and background conditions.

We plot some example images from each label.

(a) Angry. (b) Disgust. (c) Fear.

(d) Happy. (e) Sad. (f) Surprise.

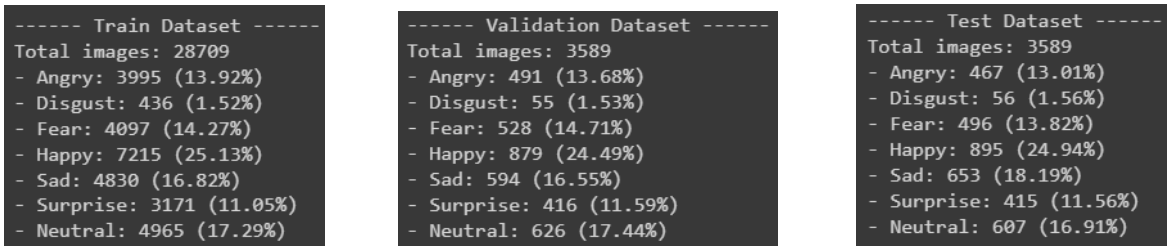
(g) Neutral.

Figure 1: Example images from each label.

[1]Source: `https://www.kaggle.com/datasets/ashishpatel26/facial-expression-recognitionferchallenge`

2.1 Train/validation/test sets

We load the dataset from a csv file containing the label, and pixel information of each picture. The dataset is already split into train ($\sim 80\%$), validation ($\sim 10\%$) and test ($\sim 10\%$) sets with the following distributions:



(a) Train. (b) Validation. (c) Test.

Figure 2: Dataset distributions.

We observe that the three datasets have very similar distributions. Moreover, the distribution of the labels is highly imbalanced, with a high number of examples from the classes "Happy" and "Neutral" and very low number of examples from the class "Disgust".

2.2 Data pre-processing

Several operations are performed to prepare the FER2013 dataset for training. Firstly, the grayscale values are rescaled to lie in the range $[0, 1]$ and repeated 3 times, resulting in a normalized 3-channel RGB representation. In order to combat overfitting and help the networks generalise better, the following random transformations are applied to each image at the point when it is fed into the network:

- random horizontal flip with probability 0.5;
- random adjustments to the brightness, contrast and saturation of the image, each by a factor uniformly sampled from the range $[0, 2]$;
- random adjustment of the sharpness of each image by a factor uniformly sampled from the range $[1, 6]$.

Finally, the images are normalized to have mean of

$$(0.485, 0.456, 0.406)$$

and standard deviation of

$$(0.229, 0.224, 0.225).$$

These numbers are chosen according to the documentation of the pre-trained ResNet50 model found in PyTorch.

3 CNN model architectures

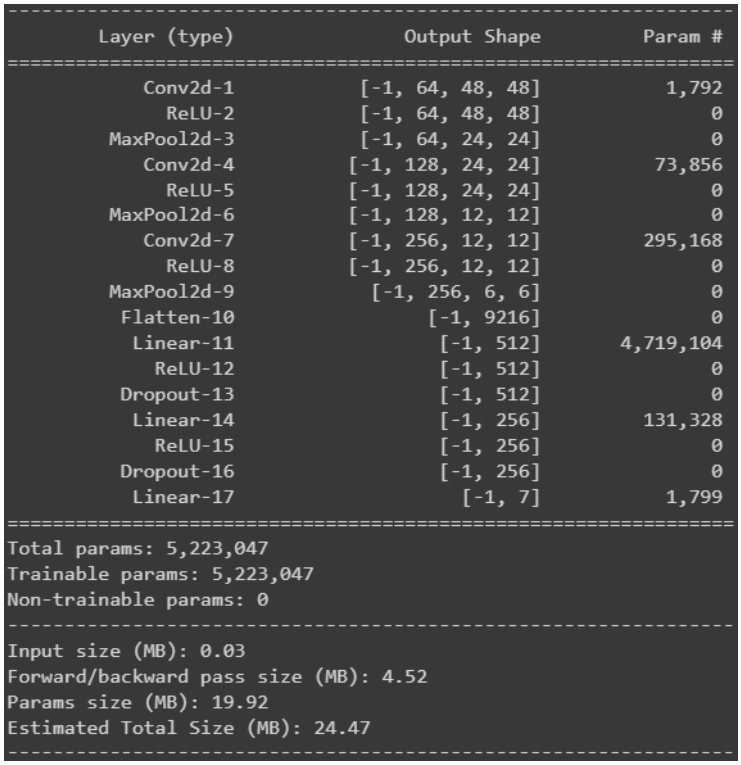
We define and test the following CNN models.

Model 1. We first define a small (1.85 million parameters) CNN architecture with 2 convolutional layers, each with kernel size of 5, stride of 1 and padding of 1, followed by a 2-layer linear classifier. After each convolutional layer we apply a max pooling layer with kernel size of 2. The activation function used is ReLU. The model architecture summary is shown below.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 46, 46]           4,864
         MaxPool2d-2          [-1, 64, 23, 23]               0
              ReLU-3          [-1, 64, 23, 23]               0
            Conv2d-4         [-1, 128, 21, 21]         204,928
         MaxPool2d-5         [-1, 128, 10, 10]               0
              ReLU-6         [-1, 128, 10, 10]               0
           Flatten-7               [-1, 12800]               0
            Linear-8                 [-1, 128]       1,638,528
              ReLU-9                 [-1, 128]               0
           Linear-10                   [-1, 7]             903
================================================================
Total params: 1,849,223
Trainable params: 1,849,223
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.03
Forward/backward pass size (MB): 2.28
Params size (MB): 7.05
Estimated Total Size (MB): 9.36
----------------------------------------------------------------
```

Figure 3: Model 1 architecture summary.

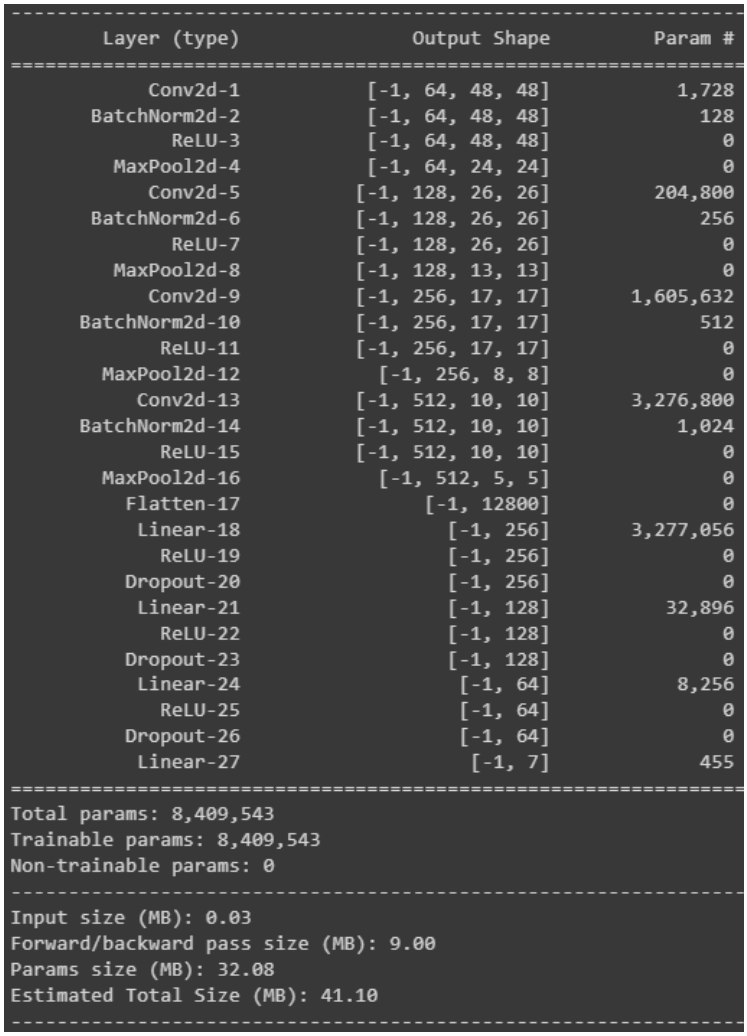
Model 2. We then define a bigger (5.22 million parameters) CNN architecture with 3 convolutional layers, each with kernel size of 3, stride of 1 and padding of 1, followed by a 3-layer linear classifier. After each convolutional layer we again apply a max pooling layer with kernel size of 2, and the activation function used is still ReLU. Between the 1st and 2nd linear layers, as well as the 2nd and 3rd linear layers we add a dropout layer with probability 0.5. The model architecture summary is shown below.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 48, 48]           1,792
              ReLU-2          [-1, 64, 48, 48]               0
         MaxPool2d-3          [-1, 64, 24, 24]               0
            Conv2d-4         [-1, 128, 24, 24]          73,856
              ReLU-5         [-1, 128, 24, 24]               0
         MaxPool2d-6         [-1, 128, 12, 12]               0
            Conv2d-7         [-1, 256, 12, 12]         295,168
              ReLU-8         [-1, 256, 12, 12]               0
         MaxPool2d-9           [-1, 256, 6, 6]               0
           Flatten-10                [-1, 9216]               0
           Linear-11                 [-1, 512]       4,719,104
             ReLU-12                 [-1, 512]               0
          Dropout-13                 [-1, 512]               0
           Linear-14                 [-1, 256]         131,328
             ReLU-15                 [-1, 256]               0
          Dropout-16                 [-1, 256]               0
           Linear-17                   [-1, 7]           1,799
================================================================
Total params: 5,223,047
Trainable params: 5,223,047
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.03
Forward/backward pass size (MB): 4.52
Params size (MB): 19.92
Estimated Total Size (MB): 24.47
----------------------------------------------------------------
```

Figure 4: Model 2 architecture summary.

Model 3. Finally we define an even bigger (8.4 million parameters) CNN architecture with 4 convolutional layers with kernel sizes 3, 5, 7 and 5 in order, each with stride of 1 and padding of 1. Right after each convolutional layer we apply batch normalization. The convolutional layers are followed by a 4-layer linear classifier, between the layers of which there are dropout layers, each with probability 0.5. The max pooling layers and the activation functions are the same as before. The model architecture summary is shown below.



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 48, 48]	1,728
BatchNorm2d-2	[-1, 64, 48, 48]	128
ReLU-3	[-1, 64, 48, 48]	0
MaxPool2d-4	[-1, 64, 24, 24]	0
Conv2d-5	[-1, 128, 26, 26]	204,800
BatchNorm2d-6	[-1, 128, 26, 26]	256
ReLU-7	[-1, 128, 26, 26]	0
MaxPool2d-8	[-1, 128, 13, 13]	0
Conv2d-9	[-1, 256, 17, 17]	1,605,632
BatchNorm2d-10	[-1, 256, 17, 17]	512
ReLU-11	[-1, 256, 17, 17]	0
MaxPool2d-12	[-1, 256, 8, 8]	0
Conv2d-13	[-1, 512, 10, 10]	3,276,800
BatchNorm2d-14	[-1, 512, 10, 10]	1,024
ReLU-15	[-1, 512, 10, 10]	0
MaxPool2d-16	[-1, 512, 5, 5]	0
Flatten-17	[-1, 12800]	0
Linear-18	[-1, 256]	3,277,056
ReLU-19	[-1, 256]	0
Dropout-20	[-1, 256]	0
Linear-21	[-1, 128]	32,896
ReLU-22	[-1, 128]	0
Dropout-23	[-1, 128]	0
Linear-24	[-1, 64]	8,256
ReLU-25	[-1, 64]	0
Dropout-26	[-1, 64]	0
Linear-27	[-1, 7]	455

Total params: 8,409,543
Trainable params: 8,409,543
Non-trainable params: 0

Input size (MB): 0.03
Forward/backward pass size (MB): 9.00
Params size (MB): 32.08
Estimated Total Size (MB): 41.10

Figure 5: Model 3 architecture summary.

4 Training

The above networks are defined using the PyTorch deep learning network. We use the PyTorch Lightning wrapper for training and testing the models. The code can be found in the `2_Deep_Learning.ipynb` notebook.

4.1 Methodology

During initial training attempts it was observed that the models quickly overfitted to the training data. Validation loss was observed to be initially decreasing and then rapidly increasing, often to values higher than the initial one. To counteract this, along with applying the random transformations explained in the data pre-processing section, we also use the following techniques:

- We start with a small learning rate of 10^{-5}.
- The weighted cross-entropy function is used for the calculation of the training loss, where the weight of each class is inversely proportional to its frequency within the dataset. It's worth noting that we also tried using weighted sampling of the images instead of the weighted cross-entropy loss, but it did not seem to provide any significantly better results.
- The AdamW optimizer is used during parameter optimization.
- A scheduler is used to reduce the learning rate by a factor of 0.1 whenever the validation loss stops improving for more than 10 epochs.

The training stops once the validation loss has not improved for over 30 epochs (early stopping), and the model with the lowest validation loss is stored as a checkpoint.

The performance of each model is evaluated using the k-fold cross validation technique ($k = 5$). In particular, the training dataset is split into 5 equally distributed subsets. Each model is trained 5 times over 60 epochs. In each experiment, the model is trained on 4 out of the 5 splits and evaluated on the remaining split, which is different each time. The metrics corresponding to each of the 5 checkpoints stored are averaged and reported.

By considering the plots of the logged metrics of each model during the 5 runs as well as the average checkpoint metrics, we decide on the best architecture for the task amongst the 3 we defined. We then train this architecture using the same setup as before, this time over a maximum of 400 epochs (the large number was chosen so that the training will have the chance to stop itself once validation loss stops improving). This time the model is trained over all training examples, and uses the original validation set for evaluation after each training epoch. Through this process we obtain our final checkpoint, which we then test and compare with a pre-trained and fine-tuned ResNet50 architecure.

4.2 Results

We now report on the results of the training methodology explained before.

4.2.1 5-fold cross-validation results

The average metrics achieved by the checkpoints logged during the 5-fold cross validation process are shown below.

Model	Training loss	Training accuracy	Validation loss	Validation accuracy
model 1	1.508	0.3381	1.46	0.3999
model 2	1.603	0.3066	1.51	0.3742
model 3	1.635	0.2022	1.533	0.295

Table 1: Average checkpoint metrics for each model over 5-fold cross validation.

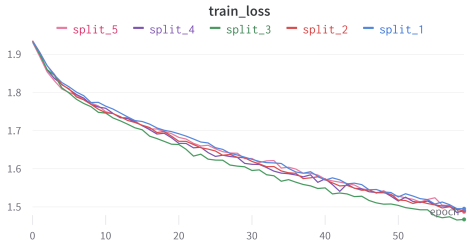
We also show the epoch at which each checkpoint was logged for each training experiment, as well as the validation loss achieved by that checkpoint.

	Model 1		Model 2		Model 3	
Split	epoch	val. loss	epoch	val. loss	epoch	val. loss
1	58	1.457	59	1.516	59	1.447
2	58	1.479	58	1.525	59	1.671
3	58	1.449	59	1.508	59	1.529
4	59	1.476	59	1.506	58	1.595
5	59	1.429	59	1.493	59	1.410

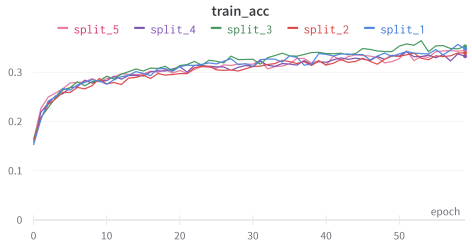
Table 2: Epoch at which each checkpoint was logged for each training experiment and validation loss achieved by that checkpoint.

We see that the smaller networks achieve better performance faster, however all networks seem to have potential to further improve.

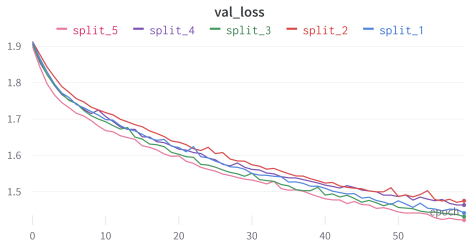
We plot the progress of the train loss, train accuracy, validation loss and validation accuracy over the 5 training experiments for model 1.



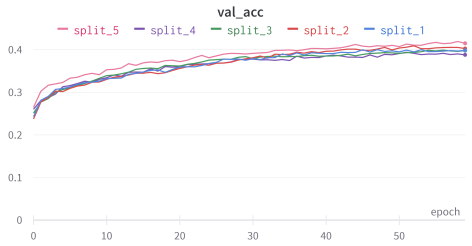
(a) Model 1 train loss.



(b) Model 1 train accuracy.



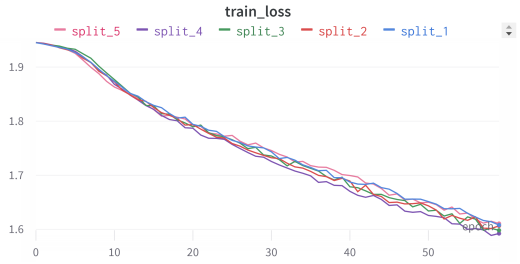
(c) Model 1 validation loss.



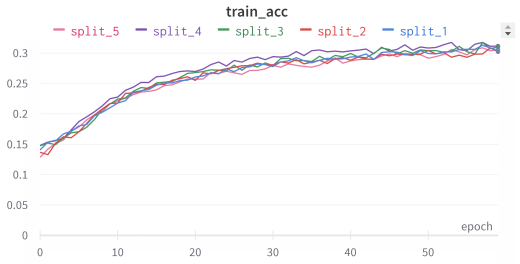
(d) Model 1 validation accuracy.

Figure 6: Model 1 performance during training over 5 splits.

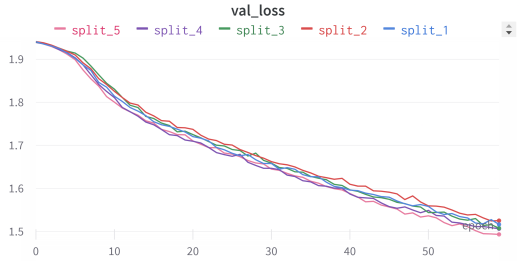
We now show the progress of the same metrics as above over the 5 training experiments for model 2.



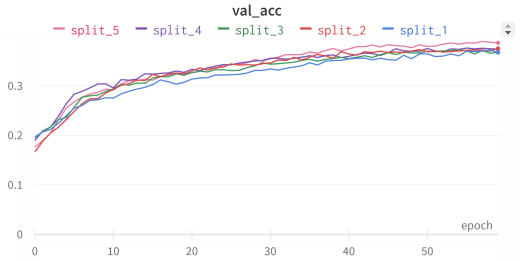
(a) Model 2 train loss.



(b) Model 2 train accuracy.



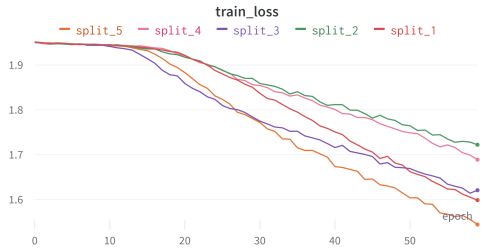
(c) Model 2 validation loss.



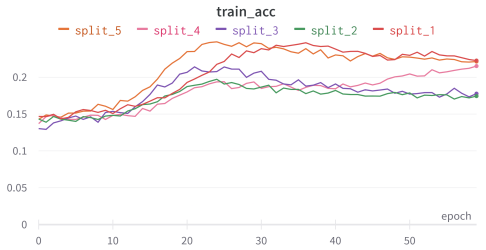
(d) Model 2 validation accuracy.

Figure 7: Model 2 performance during training over 5 splits.

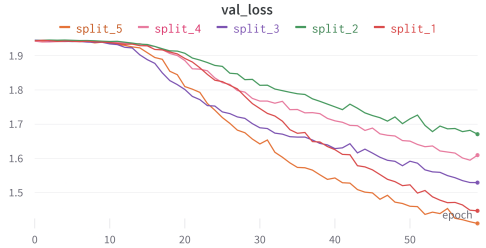
Similarly, we show the same metrics for model 3.



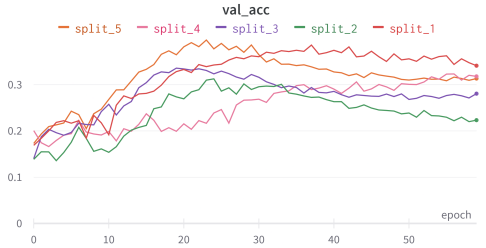
(a) Model 3 train loss.



(b) Model 3 train accuracy.



(c) Model 3 validation loss.



(d) Model 3 validation accuracy.

Figure 8: Model 3 performance during training over 5 splits.

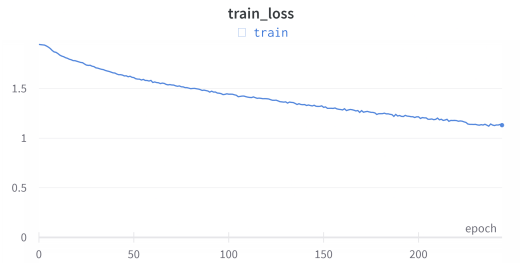
We observe that for most runs and for all 3 models, validation loss seems to be consistently less than training loss and similarly validation accuracy is higher than training accuracy, which is most likely explained by the various random augmentation techniques we have applied on the data, as well as the weight decay applied by the AdamW optimizer.

The plots for model 1 and model 2 are similar, showing good and relatively stable progress in the training and validation loss. On the other hand, model 3 seems to behave more unpredictably during training, and despite the addition of droupout layers and batch normalization techniques we can see that the validation accuracy is even decreasing after a certain point in some runs.

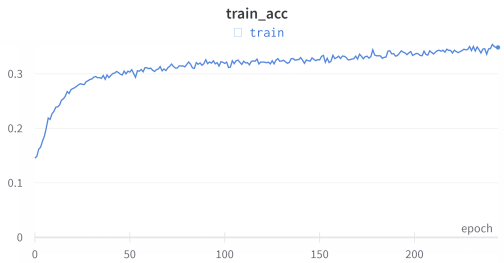
Based on the above information we choose to train model 2, since it is larger than model 1 and its training logs are more stable than those of model 3.

4.2.2 Training the chosen architecture

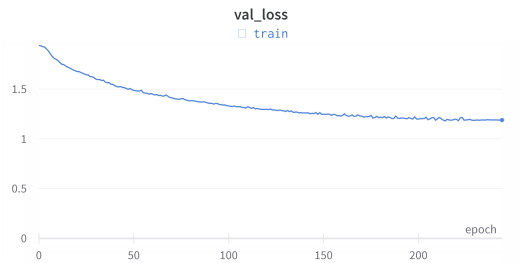
We train model 2 on the entire training dataset over a maximum of 400 epochs. The metrics logged during training can be seen below.



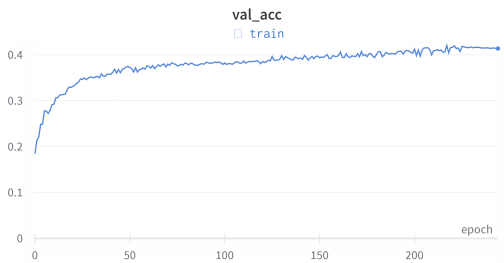
(a) Model 2 train loss.



(b) Model 2 train accuracy.



(c) Model 2 validation loss.



(d) Model 2 validation accuracy.

Figure 9: Model 2 performance during training.

The checkpoint was obtained on epoch 214 with a validation loss of 1.180.

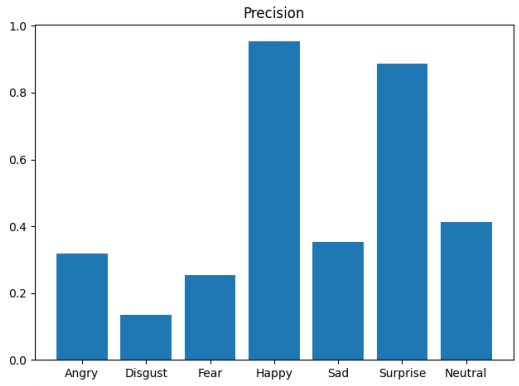
5 Testing

We now test our chosen architecture on the original test dataset. The metrics we get are the following:

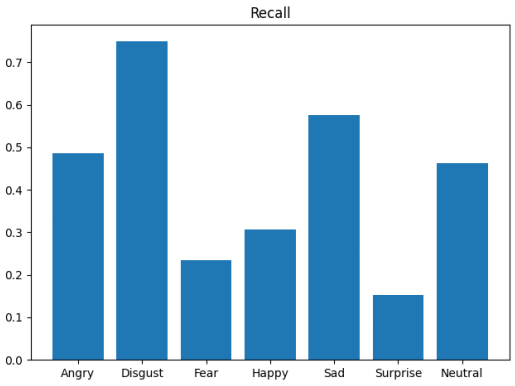
Test loss	Test accuracy
1.224	0.3781

Table 3: Test metrics for model 2 checkpoint.

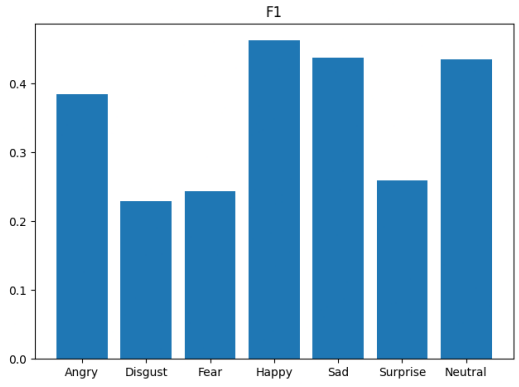
We log the precision, recall and F1 score per class, and also show the resulting confusion matrix.



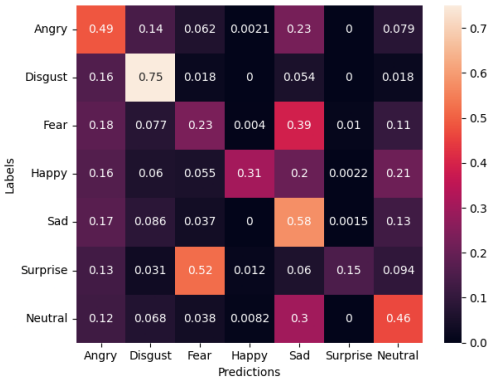
(a) Precision per class.



(b) Recall per class.



(c) F1 score per class.



(d) Confusion matrix.

Figure 10: Precision, recall, F1 score and confusion matrix for the logged model 2 checkpoint evaluated on the original test set.

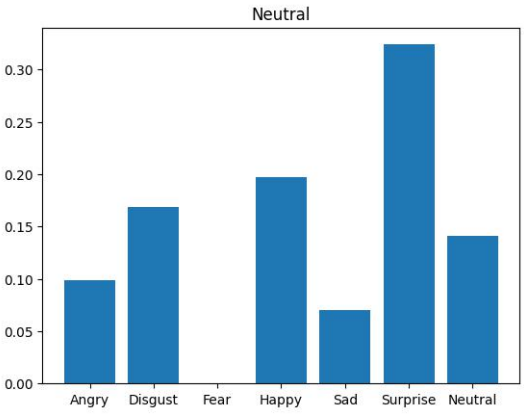
5.1 Testing on real-life videos

We test the model on 3 videos captured by a web-cam, each depicting a different emotion.

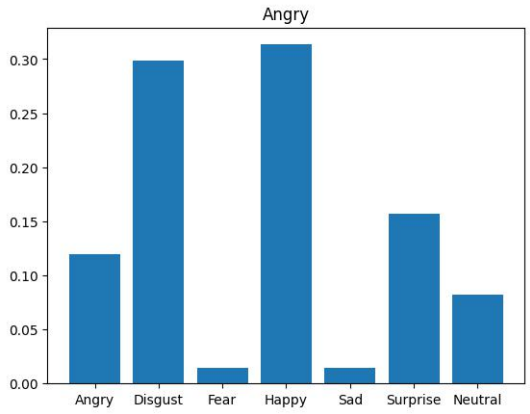
Video	Label	Number of frames
1	Neutral	71
2	Angry	134
3	Happy	105

Table 4: Video information.

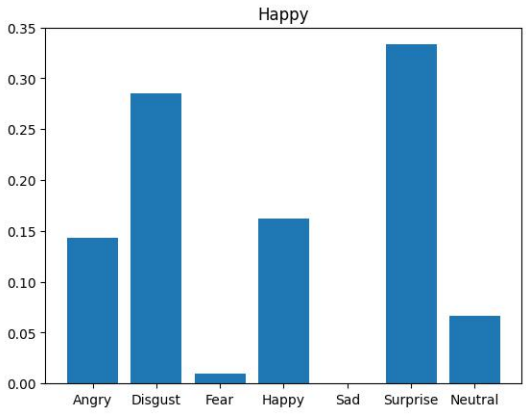
We plot the distribution of the predictions made by the model 2 checkpoint for each video.



(a) Predictions per frame for "Neutral" video.



(b) Predictions per frame for "Angry" video.



(c) Predictions per frame for "Happy" video.

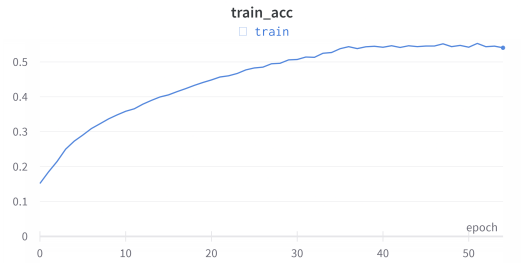
Figure 11: Distribution of predictions made by model 2 for each frame of each video.

5.2 Comparison with ResNet50

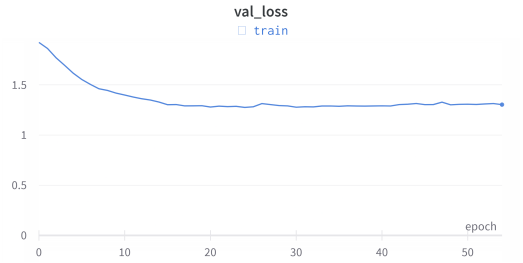
We load the pre-trained ResNet50 model from PyTorch using the default weights. We add a linear classifier layer on top of its own in order to transform the output into the 7 classes required for this emotion classification task. We fine-tune the model by training it in the exact same way as the model 2

checkpoint. We plot the metrics that were logged during training.

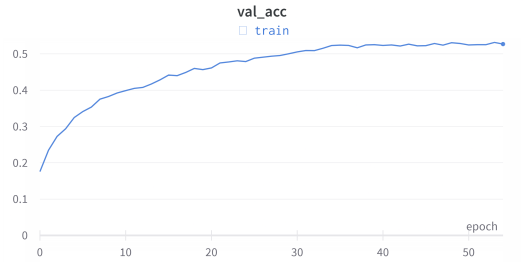
(a) ResNet50 train loss.



(b) ResNet50 train accuracy.



(c) ResNet50 validation loss.



(d) ResNet50 validation accuracy.

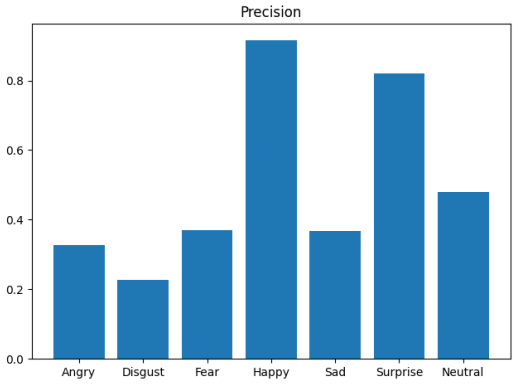
Figure 12: ResNet50 performance on the validation set during training.

The ResNet50 checkpoint was obtained on epoch 24 with a validation loss of 1.275. We evaluate the checkpoint on the original test dataset.

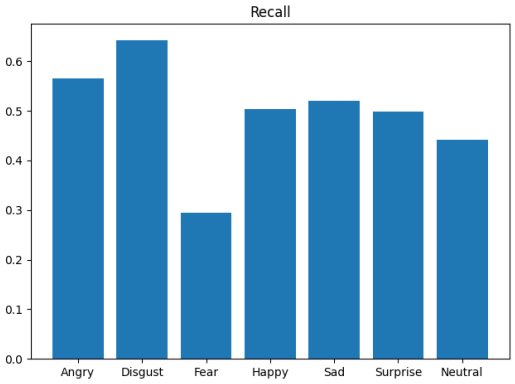
Test loss	Test accuracy
1.255	0.4745

Table 5: Test metrics for ResNet50 checkpoint.

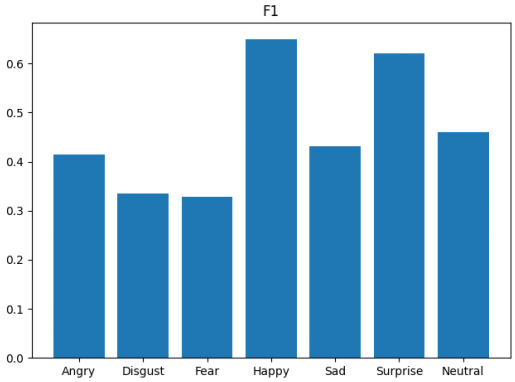
As before, we plot the precision, recall and F1 score per class, as well as the confusion matrix.



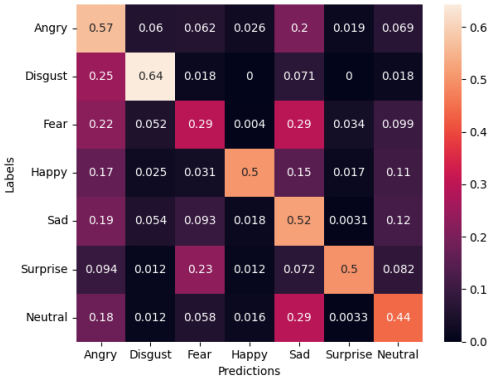
(a) Precision per class.



(b) Recall per class.



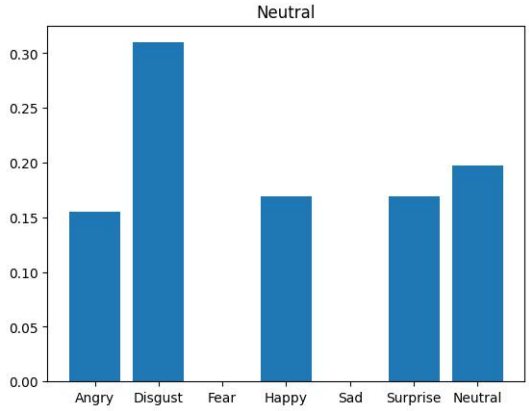
(c) F1 score per class.



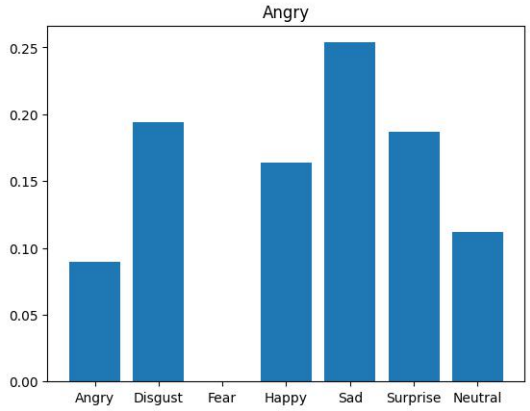
(d) Confusion matrix.

Figure 13: Precision, recall, F1 score and confusion matrix for the logged ResNet50 checkpoint evaluated on the original test set.

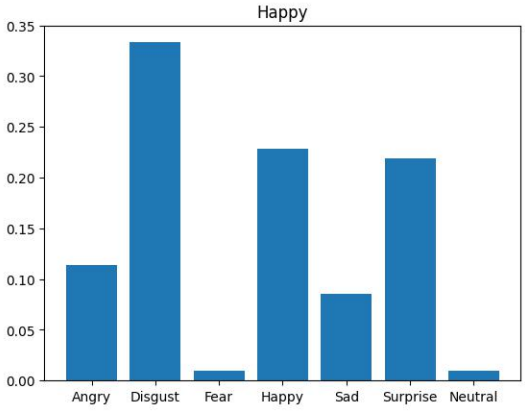
Finally, we also test the ResNet50 checkpoint on the 3 videos from before.



(a) Predictions per frame for "Neutral" video.



(b) Predictions per frame for "Angry" video.



(c) Predictions per frame for "Happy" video.

Figure 14: Distribution of predictions made by ResNet50 for each frame of each video.

6 Visualizing the activations

Visualizing the activations of our CNN provides valuable insights into how the network processes different emotions. We observed that certain channels in the layer exhibited specific preferences. Some channels seemed to focus on capturing the outline of the face, highlighting the overall shape and structure. Other channels appeared to pay attention to the facial characteristics, emphasizing the eyes, nose, and mouth. Additionally, some channels seemed to be sensitive to the background elements surrounding the face.

Here is the visualization of the activations of different images, one for each emotion (angry, disgust, fear, happy, sad, surprise, neutral).

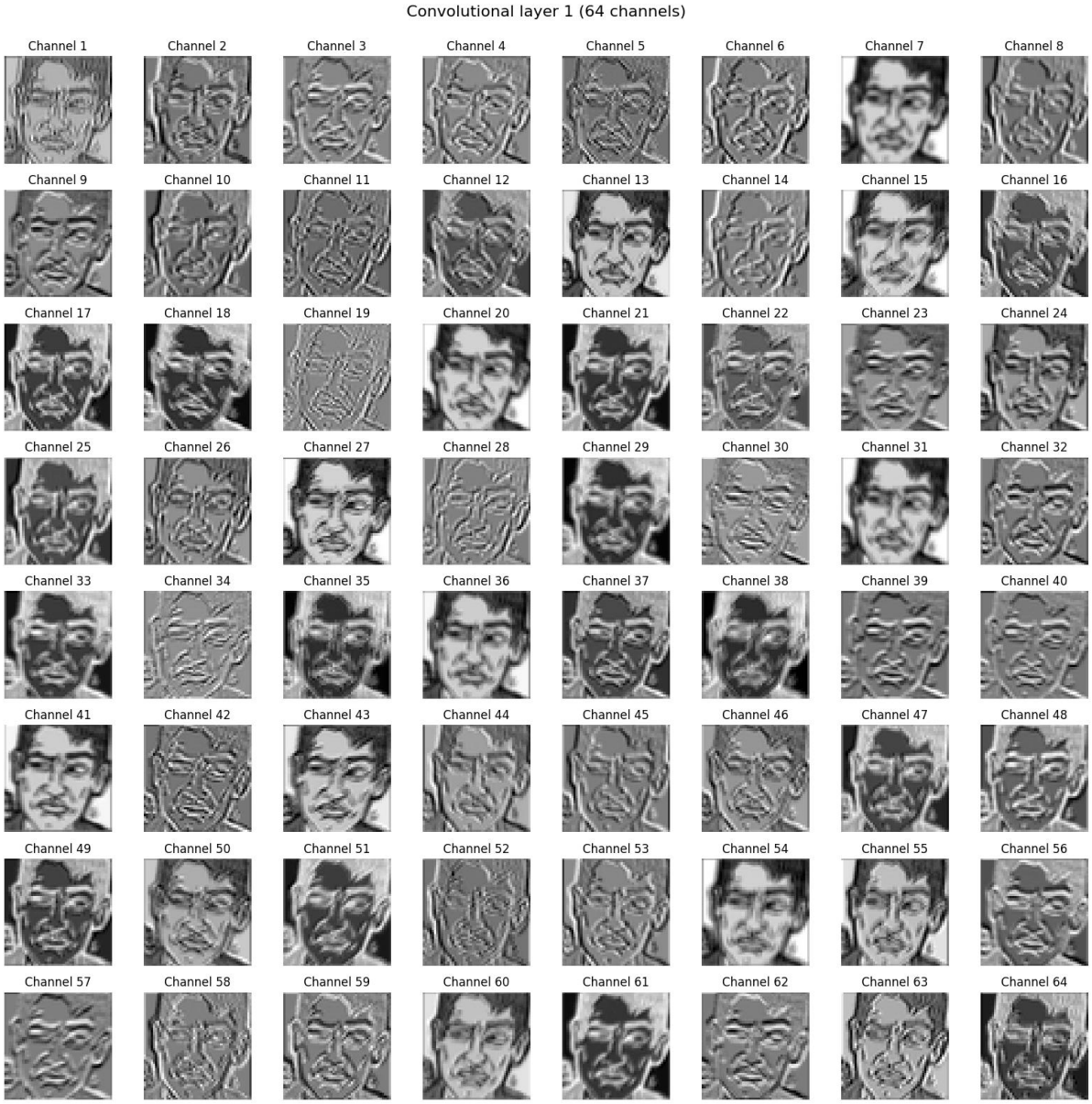


Figure 15: Activations of layer 1 for an image labeled as “Angry”.

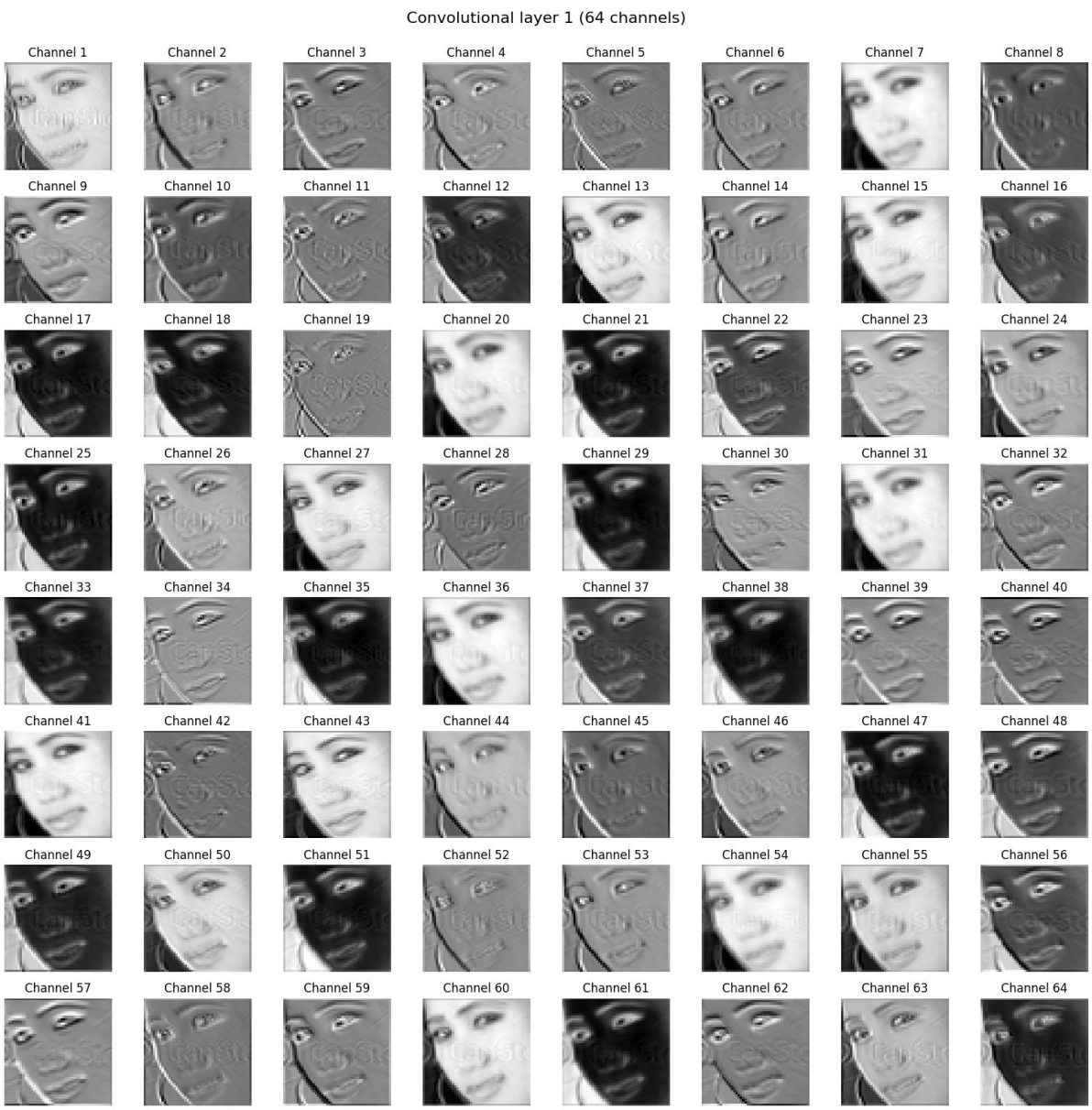


Figure 16: Activations of layer 1 for an image labeled as “Disgust”.

Figure 17: Activations of layer 1 for an image labeled as “Fear”.

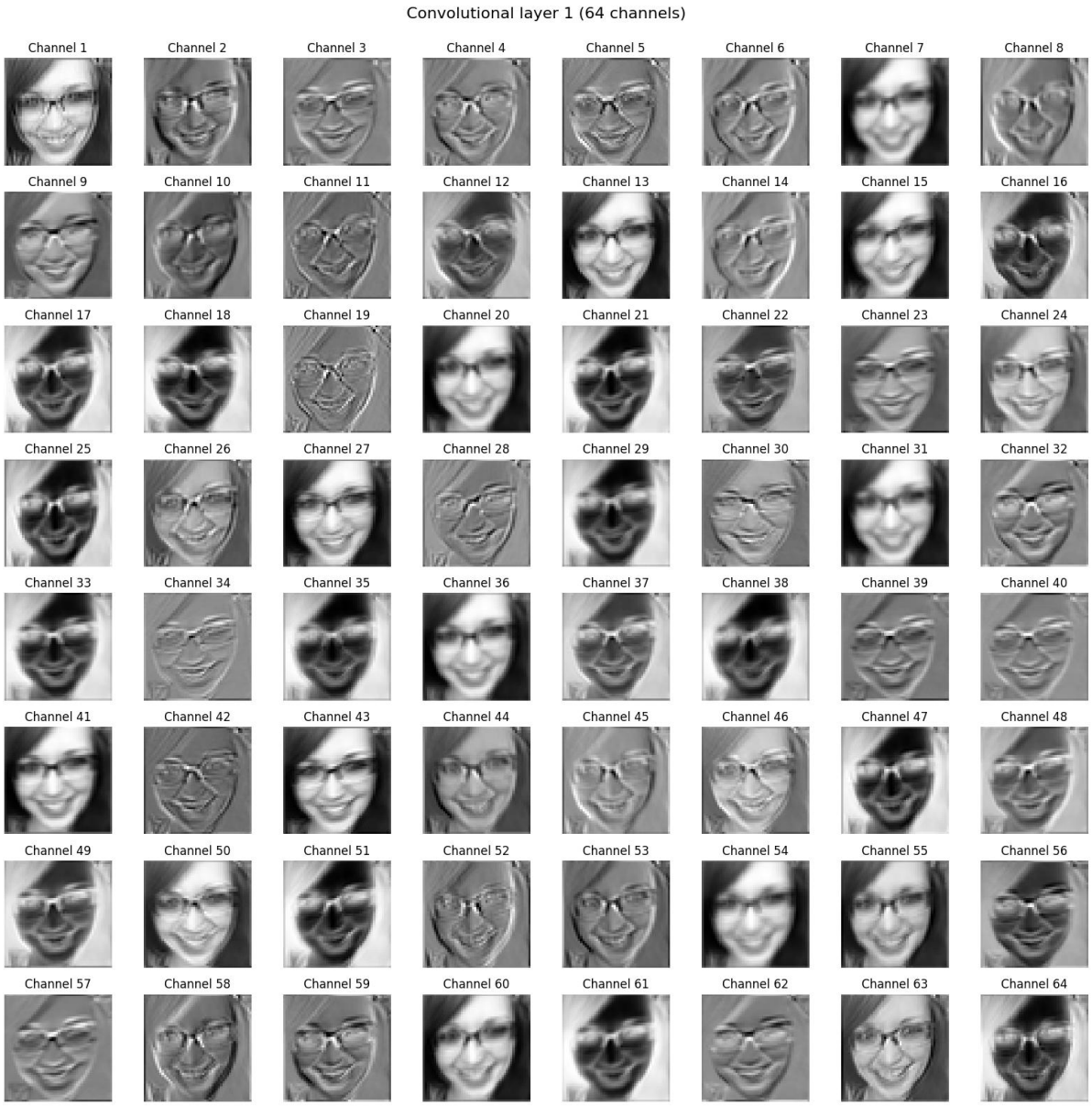


Figure 18: Activations of layer 1 for an image labeled as “Happy”.

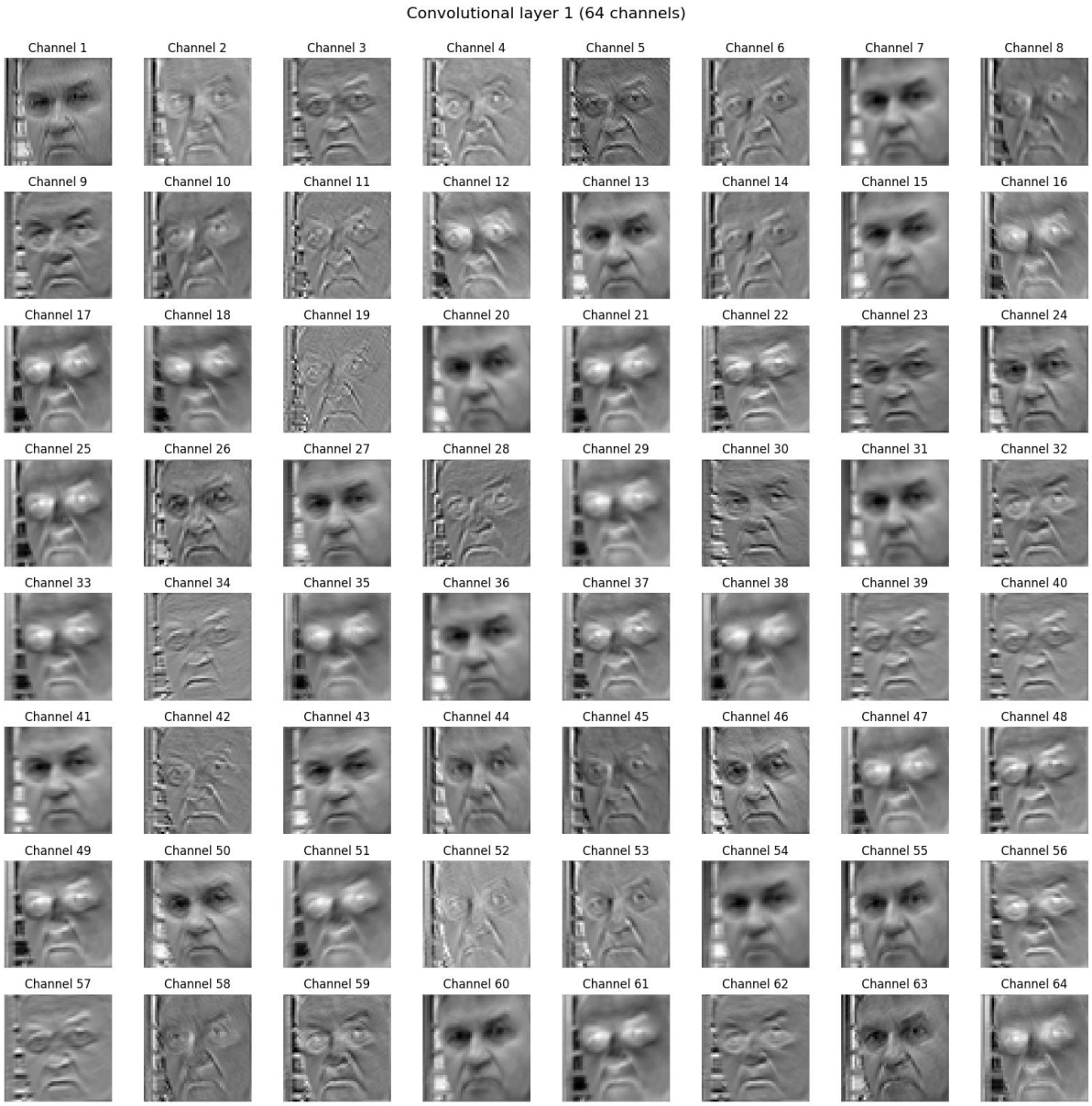


Figure 19: Activations of layer 1 for an image labeled as “Sad”.

Figure 20: Activations of layer 1 for an image labeled as “Surprise”.

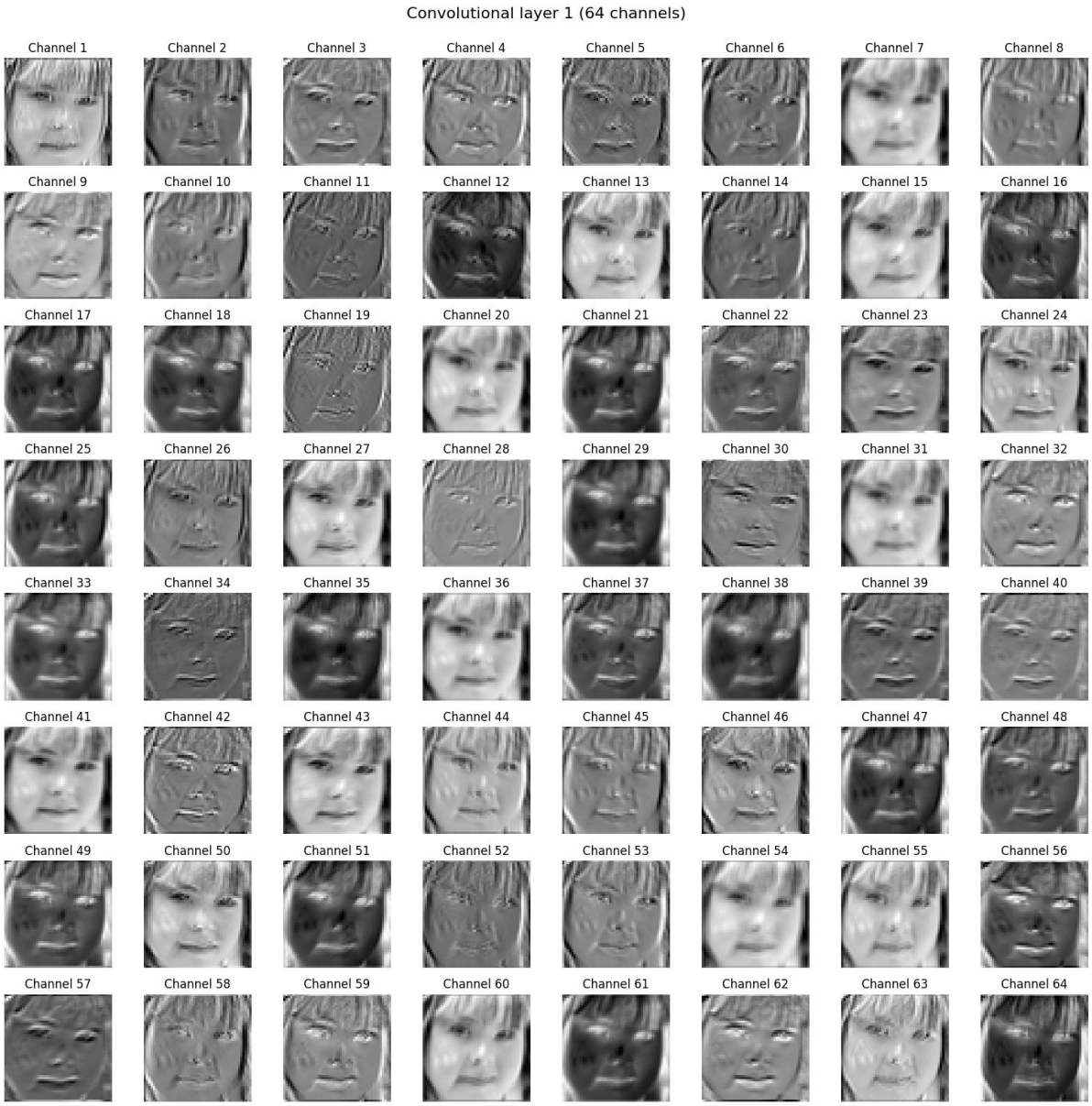


Figure 21: Activations of layer 1 for an image labeled as “Neutral”.