# Static Facial Expression Recognition Using Artificial Neural Networks

Fredrik Lastow

Research School of Computer Science, Australian National University, Australia
u7088255@anu.edu.au

**Abstract.** The *Static Facial Expression in the Wild* database [2] contains screenshots of facial expressions in movies, which mimic real world scenarios better than widely used lab generated data. Local phase quantization (LPQ) and pyramid of histogram of gradients (PHOG) features are passed into an *single hidden layered neural network* (SHLNN) for classification. A *convolutional neural network* (CNN) is also developed to try and contrast the confined nature of the SHLNN. The CNN will take images as inputs rather than the pre-calculated features. The network structures are empirically developed and tested with 10-fold cross validation. A performance measure of *accuracy*, *precision*, *recall* and *specificity* is used to evaluate the networks. The datasets are also analysed to apply preprocessing. For the SHLNN the techniques include normalising data, investigating outliers and keeping an even class distribution. Noisy training is also implemented to try and increase performance together with the preprocessing in SHLNN. The normalising proved to give an 4 p.p increase in accuracy, but the other techniques prove not to noticeably increase the performance. For the CNN preprocessing is applied using a pre-trained multi-task CNN (MTCNN) to crop the faces out of the raw SFEW images. The concluding facial classification accuracy is 23% for the SHLNN and 31% for the CNN compared to the baseline accuracy of 19% [2]. Using equilateral encoding and balancing the uneven datasets could have resulted in better performance. The FER systems do not achieve a sufficient performance to be usable in their current state, but the open nature of the CNN has potential to improve and become acceptable.

**Keywords:** Facial expression recognition · Static Facial Expression in the Wild · Artifical Neural Network · Preprocessing .

## 1 Introduction

The human face can express a large variety of emotions. There's is a natural connection between how a person is feeling and their facial expression. For most humans the ability to read these expressions and interpret them to understand a person's affective state, is effortless and part of our natural behaviour. However, translating this behaviour to a computer has proven to be a much harder task.

Today there are a lot of databases used to test *facial expression recognition* (FER). However, most of the extensively used facial expression databases consist of lab generated data. This is, data that has been generated in a controlled environment and thus do not portray real life conditions accurately. Two good examples are the databases *Multi-PIE* [5] and *JAFFE* [4]. To test FER in more realistic environments we will use the database *Static Facial Expressions in the Wild* (SFEW) [2]. SFEW contains 675 images labelled with the seven different facial expressions *angry*, *disgust*, *fear*, *happy*, *neutral*, *sad* and *surprise*. The origin of SFEW is another database called *Acted Facial Expressions in the Wild* (AFEW) [3]. AFEW contains clips from 37 different movies, all chosen to portray a large range of ages and realistic scenarios. The database was created by searching for keywords in movies' *Subtitles for the Deaf or Hard-of-Hearing (SDH)* and *Closed Caption (CC)*. These keywords are associated with an expression and could be 'laughs' or 'scared' for example. The scenes containing subtitles with these keywords where then annotated by a human with more information about the actor, character and their expressions in the scene. SFEW has been created by picking frames of the clips in AFEW. Even though facial expressions are better represented as a sequence of images, as it would capture facial motion, it's outside the scope of this paper. Here only static expressions are used with the images in SFEW compared to AFEW. SFEW still covers different resolutions, focus and head poses as well as realistic lighting and thus a good representation of real world situations. [2] It's worth noting that it is still just actors playing a role and not actual real world situations. However, it is still a better approximation than lab generated data and thus leads us to training a model with high generality

The FER systems that will be evaluated are two different *artificial neural network*s (ANN). The first one is a fully connected network with a single hidden layer and the second one is a *convolutional neural network* (CNN).

The *single hidden layer neural network* (SHLNN) will work as a simple classifier and be trained using resilient backpropagation (RPROP) as an optimiser and cross entropy as a loss function. RPROP requires fewer

iterations and learns faster than regular backpropagation and is therefore used. Cross entropy is a useful loss function for a multi-classification problem, which combines log softmax and negative log likelihood (NLL). The network will use the sigmoid activation function in the forward pass. It will be fed ten pre-calculated features extracted from each image of the SFEW dataset. The first five of these features are principal components of *local phase quantization (LPQ)*. LPQ is an extension of the widely used *local binary pattern (LBP)* and has proven to give better performance. LPQ looks at signs of the Fourier coefficients of each pixel to extract the phase information of that pixel in a local window. The following five features are principal components of *pyramid of histogram of gradients (PHOG)*. PHOG has been used in object recognition and shown prominent performance. It is an extension of *histogram of oriented gradients (HOG)* which count gradient orientations in a section of the image. [2]

The second network is a convolution neural network (CNN) and is in contrast with the first network, fed raw images. Here, we let the network itself figure out the feature extraction rather than calculating them ourselves. The network will be trained using backpropagation with an Adam optimizer and the negative loss likelihood (NLL) loss function. The Adam algorithm efficient and has low requirements of memory. [10] This loss function performs the same as the cross entropy for the SHLNN, since we use a log softmax on the output layer in the CNN. In the networks fully connected linear layer, it will use an ReLU activation function.

Both networks will be trained using supervised learning with the SFEW database. We will be developing the network structure as well as hyperparameters empirically and see what ultimately gives us the highest performance. For the CNN we follow classic networks as inspiration for building the structure. Here, LeNet [6] will be used as a basis. When evaluating the structures we will use a performance measure of *accuracy*, *precision*, *recall* and *specificity*. [2] We will also divide the dataset into a training and test set using 10-fold cross validation to ensure the results.

With two FER systems at hand we will try to improve our performance further by analysing the data more closely. For the SHLNN we will be looking at the distribution of feature values and make sure it's normalised. We will also add random noise during the training and perform a noisy training with a decreasing amplitude. [1] For the CNN we will crop the faces from the raw images to concentrate the network on the facial features. The distribution of training examples over the classes for both the networks will also be examined.

The nature of the (SHLNN) is very confined with pre-calculated features compared to the open and undefined CNN. Having these two networks in contrast with each other give an insight of how important this is when building a network or if it's not important at all.

## 2 Method

### 2.1 Hyperparameters and network structure

The first thing we will do is to decide the appropriate hyperparameters and structure for our networks. This is done empirically by testing different values and evaluating the result using the performance measure. With many parameters needing to be determined it is important to only make changes to one parameter at a time to understand how it affects the network.

#### 2.1.1 Single Hidden Layer Neural Network

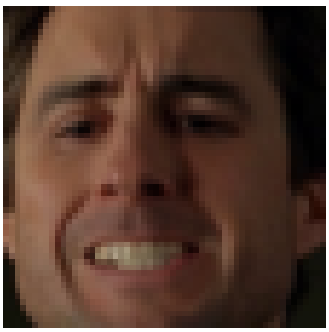
We will only use one hidden layer in the network which should give us sufficient performance given enough hidden neurons [9]. A good starting point for the amount of hidden neurons is between the number of outputs and inputs i.e. 7-10. While measuring the performance we slowly increase the number of hidden neurons and see if more neurons are beneficial.

With RPROP as an optimiser we will not be as reliant on an optimal learning rate as in Stochastic Gradient Descent (SGD), since we are updating each weight individually and only looking at the sign of the gradient. We will therefore not evaluate this hyperparameter and keep it at a standard 0.01.

#### 2.1.2 Convolutional Neural Network

When deciding the hyperparameters and structure of the CNN we follow standard conventions and take inspiration from classic networks. We start with a small filter size to extract small features of the images and then gradually increase the filter size to extract features with higher level representations deeper in the network. The same idea is applied to the number of filters. More filters widens our feature space and helps the network learn abstract representations. The number of channels to the first layer will be three, since there's three colour channels in the images. We start with a lower number, which will be three since we have three colour channels in the input images. We want to detect low level features at first and then increase the amount of filters gradually to make global complex shapes with the combination of the lower level representations. Therefore, we want to keep the number of filters high, while not overfitting. Since we generally don't care about information in the

(a)



(b)

Fig. 1: Images from the SFEW dataset labelled angry. The image (a) is the original raw image from the dataset, while (b) is the results of the preprocess cropping by the MTCNN of the same image.

borders of the image the padding is not evaluated and kept at 0. We also keep the stride as 1 since we reduce the network size fast enough with the pooling layers and we also don't want to skip any information.

The network structure will be based on a Conv-Pool-Conv-Pool structure inspired by LeNet. We will use max-pooling layers between the convolutional layers to reduce the input dimensionality while still keeping the most prominent and feature i.e. the max value. These max values can be seen as the feature that's activated for the current training example. We consecutively add layers to the network and compare the performance while making sure not to over fit. After each layer we compute the shape of the output to be able to connect it to the following layer. To reduce overfitting we add dropout layers between every pooling layer, as well as one in the fully connected, linear layer. The dropout layers will have a increasing probability deeper in the network.

### 2.2 Training and preprocessing

The training on both networks is performed using 10-fold cross validation. This entails dividing the dataset into 10 equal parts and then using one tenth as a validation set and the remaining as a training set. When the training has finished the next tenth is chosen and the process is repeated over the whole dataset. Thus, we will ultimately use all data to train and test, which lowers the bias towards the data. When training the network, the optimal number of epochs is most easily determined by starting with a as high number as possible without the total training time becoming to large. By taking note of the loss in each epoch we can see how our network learns and when it closes in on asymptotic values and additional training is unnecessary. This also helps us to only train as long as the test loss is reducing to prevent overfitting.

Another important aspect in optimal training for both netowrks is having an even amount of training examples for each class. Having uneven data for a class means that our network will know less about that underlying class compared to the rest and thus potentially perform worse overall. Therefore, we will also look at histograms of class occurrence.

#### 2.2.1 Single Hidden Layer Neural Network

When performing the noisy training on the SHLNN we will add a random noise from a normal distribution. The distribution's standard deviation will decrease over the training period and subsequently become zero. The noisy training is also alternated and only performed on every other 25 epochs. This sequence is started after the first 100 epochs of training.

To investigate the data, i.e. the calculated LPQ and PHOG features, we start by looking at the specific features and their distribution. We will calculate histograms of each feature to get a picture of the its mean and variance. For optimal learning we want our data in the same interval, i.e. normalised. Potential outliers could also be a factor in decreasing our ability to train well on the data. Outliers often hold important information of the underlying class and removing them would be counterproductive. We can locate the outliers by defining them as values outside three standard deviations from the mean. By locating all the outliers and changing their values to the mean of the feature values we can get an idea of their impact on the training process.

#### 2.2.2 Convolutional Neural Network

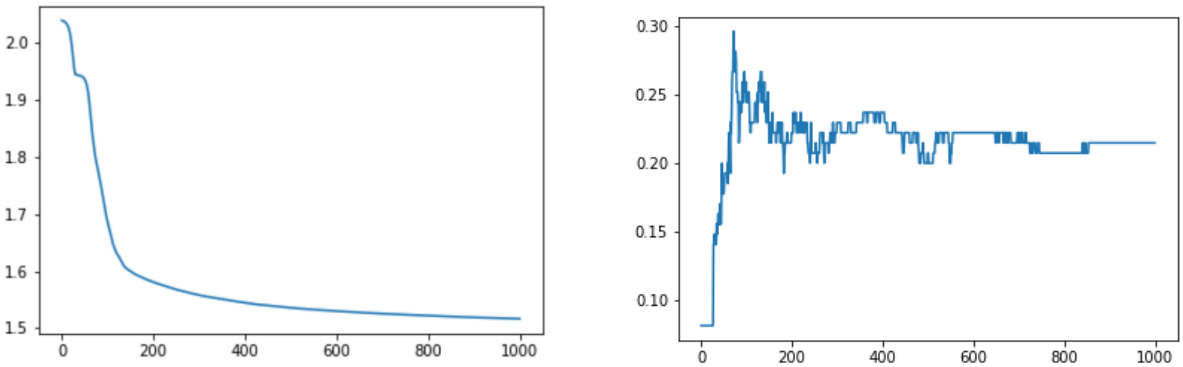
When feeding an image through a CNN, the separate convolutions in our filters pick up features. Therefore, when looking at images from SFEW the network will not only learn facial features, but also the surrounding environment. This will lead to a worse performance and a loss in generality since this is teaching our network that the background in the image is linked to the facial expression. This is obviously not true and not a desired trait. Therefore, the dataset will be preprocessed by cropping 56x56 images only containing the faces of SFEW.

This was done using a pre-trained multi-task CNN (MTCNN) network for facial recognition called *facenet*. [8] The MTCNN works in three stages, the first one using a shallow CNN to find potential regions in the image for the face . The second stage filters out wrongful regions with a more complex CNN, followed by the third stage, which again removes unwanted regions, but with a more powerful CNN. Thus, we have a bounding box for a face with a proven high accuracy. [7] We will use this to create a new version of the SFEW database consisting of only faces. This transformation can be seen in figure 1. The new dataset consisting only of faces is then fed into the CNN.

## 3 Results and Discussion

### 3.1 Single Hidden Layer Network

Starting at 8 hidden neurons and gradually increasing there was no connection found between higher performance and a higher amount of hidden neurons. With more hidden neurons the training accuracy increased to 99% at 50 hidden neurons. However, the test accuracy had no noticeable improvement and thus we have overfit the training data and need a more generalised model. We will therefore stay at a total of 8 hidden neurons. The learning rate is kept a a standard 0.01 and with 1000 epochs we get the result shown in Figure 2.




(a) (b)

Fig. 2: The training loss (a) and test accuracy (b) over 500 epochs shown on the x-axis. After the optimal 400 epochs we reach an 23% test accuracy.

We can see that after around 800 epochs our training loss is starting to reach the asymptotic value and is already quite low at 400 epochs. The test accuracy has a consistent value after 300 epochs, but decrease slightly after 400. In future evaluations we will use 400 epochs as the optimal. The chosen optimal value of each hyperparameter and the final network structure can be seen in table 1.

| Hyperparameter: | Input neurons | Hidden neurons | Output neurons | Epochs | Learning rate |
|---|---|---|---|---|---|
| Optimal value: | 10 | 8 | 7 | 400 | 0.001 |

Table 1: The networks optimal hyperparameters and structure.

With the network fully determined we can calculate a explicit performance measure which is shown in Table 2. The average test accuracy in the cross validation can be established as 23% and can be seen in Figure 3b. This is above the baseline classification accuracy of 19% [2]. The performance measure also looks similar to the baseline results with a higher specificity on all classes and a generally higher recall as well except for *disgust* (class 2). The reason for this we will get to shortly, but overall our performance is sufficient compared to the baseline.

| Emotions | Angry | Disgust | Fear | Happy | Netural | Sad | Surprise |
|---|---|---|---|---|---|---|---|
| Precision | 0.16 | 0.12 | 0.36 | 0.27 | 0.25 | 0.26 | 0.11 |
| Recall | 0.14 | 0.097 | 0.40 | 0.43 | 0.17 | 0.30 | 0.13 |
| Specificity | 0.91 | 0.96 | 0.76 | 0.76 | 0.89 | 0.80 | 0.93 |
| Overall accuracy: | 23% | | | | | | |

Table 2: The SHLNN's performance measure averaging for each of the seven classes over the 10-fold cross validation. The measures are *precision*, *recall* and *specificity*.

### 3.2 Convolutional Neural Network

With three layers for the LeNet inspired structure the best performance was found. Both two and four layers?? Different learning rates were tested, but the optimal was found to be 0.001. With a learning rate smaller than 0.001 the network's loss barely decreased and with values larger than 0.001 the loss fluctuated indecisively. Starting with 4 filters and then doubling in each consecutive convolutional layer didn't give a good performance measure. Therefore, the number of filters were increased to first 8 and then finally 12 in the first layer, which showed satisfactory results.

| Layer | Number of Filters | Filter size | Stride | Padding | Shape |
|---|---|---|---|---|---|
| Input Image | - | - | - | - | 56x56x3 |
| Conv2d | 12 | 3 | 1 | 0 | 54x54x12 |
| MaxPool | - | 2 | 2 | 0 | 27x27x12 |
| Dropout(p=0.1) | - | - | - | - | 27x27x12 |
| Conv2d | 24 | 5 | 1 | 0 | 23x23x24 |
| MaxPool | - | 2 | 2 | 0 | 11x11x24 |
| Dropout(p=0.15) | - | - | - | - | 11x11x24 |
| Conv2d | 48 | 5 | 1 | 0 | 6x6x48 |
| MaxPool | - | 2 | 2 | 0 | 3x3x48 |
| Dropout(p=0.25) | - | - | - | - | 3x3x48 |
| Linear | - | - | - | - | 768x1 |
| Dropout(p=0.5) | - | - | - | - | 768x1 |
| Linear | - | - | - | - | 120x1 |
| Log Softmax | - | - | - | - | 7x1 |

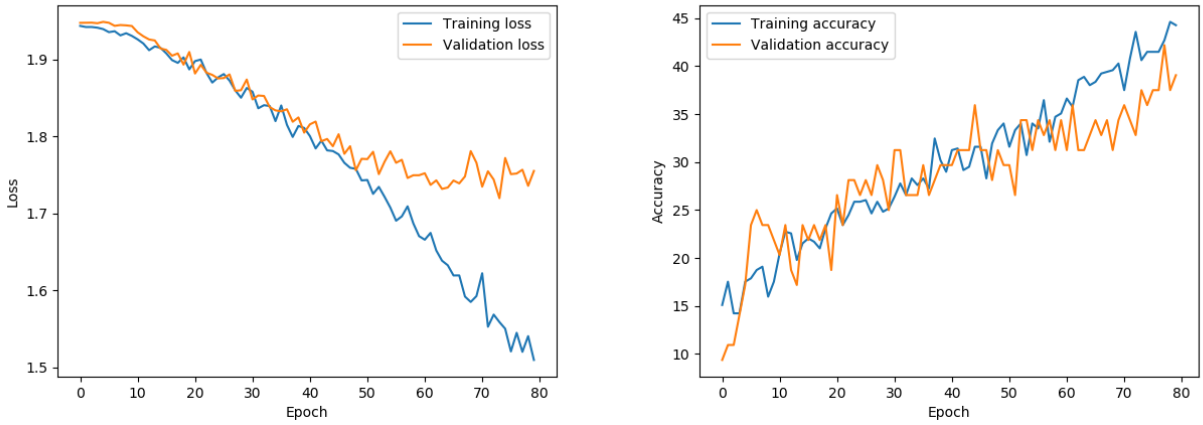
Table 3: The CNN's network structure with every layer's shape and settings. The structure is following a Conv-Pool-Conv-Pool structure inspired by LeNet.

The filter sizes of 3-5-5 was the most prominent. Starting with higher and also ending with higher filter sizes than 3-5-5 made the network unable to find good representations in the feature mapping. In table 3 the full developed network structure can be seen with every layer's shape and settings.

| Emotions | Angry | Disgust | Fear | Happy | Netural | Sad | Surprise |
|---|---|---|---|---|---|---|---|
| Precision | 0.32 | 0.21 | 0.40 | 0.44 | 0.24 | 0.20 | 0.31 |
| Precision | 0.32 | 0.21 | 0.40 | 0.44 | 0.24 | 0.20 | 0.31 |
| Recall | 0.39 | 0.19 | 0.32 | 0.44 | 0.23 | 0.20 | 0.31 |
| Specificity | 0.82 | 0.94 | 0.85 | 0.77 | 0.89 | 0.90 | 0.85 |
| Overall accuracy: | 31% | | | | | | |

Table 4: The CNN's performance measure averaging for each of the seven classes over the 10-fold cross validation. The measures are *precision*, *recall* and *specificity*.

The structure was then trained and tested with 10-fold cross-validation over 80 epochs. We can see the results of one fold in figure 3. After around 60 epochs the validation loss remains constant, while the training loss decreases steadily and it's evident that we're overfitting. Thus, 60 epochs is chosen for optimal performance which fits well on the other folds as well. Given this structure and hyperparameters a overall accuracy of 31% was achieved and the full performance measure can be seen in table 4.

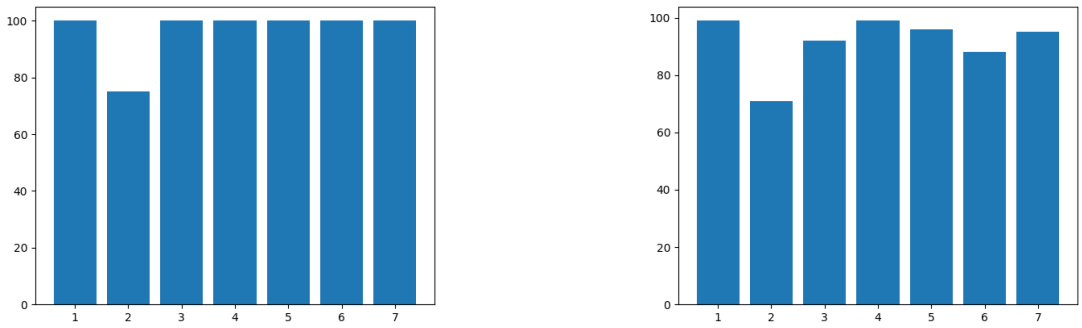



(a) (b)

Fig. 3: The training and test loss (a) and training and test accuracy (b) over 80 epochs. After the optimal 60 epochs we reach an 31% test accuracy.

### 3.3 Dataset Imbalance

The class occurrence in the dataset is shown as a histogram in Figure 4. We notice that for the dataset used for the SHLNN only have 75 training examples for class 2 compared to 100 for the other classes. This will make it more difficult for our network to learn the properties of class two, which we can see in Table 2 with a low recall and precision relative to the other classes. If we were to remove 25 training examples from the other classes we would see a rise in recall and precision for class 2 of about 0.1, but a decrease for the other classes. This is not a viable option since we want to use all the data we have and this would delete 18%. In the dataset with the cropped SFEW images, we see an even bigger imbalance. The cropping itself failed on a number of images and thus reducing the amount of training examples. With this imbalance it's hard for our network to train evenly and we'll get a worse performance. When looking closer at the dataset used for the CNN it is evident that a facial expression has multiple training examples from the same scene. That is, multiple frames are taken from the same scene portraying the same emotion. This could be a problem since the network might be biased towards that actor or face for a given emotion.



(a) (b)

Fig. 4: Histogram of the number of training examples for each class (1-7) in the dataset. The distribution of the dataset used for the SHLNN is shown in (a) and the one used for the CNN is shown in (b). Note, that (b) shows the distribution after the face-cropping by the MTCNN, otherwise they would look identical.

### 3.4 Preprocessing Techniques

When training the SHLNN a noticeable variance in the performance was noticed over different training iterations using the same parameters. The accuracy fluctuates from 22-24%, which makes it difficult to make correct

decisions about the network and evaluate data preprocessing. If the changes made in the data during preprocessing or in the network structure does not impact the performance in the same scale as this variance, it won't be noticeable. A reason for this behaviour could be that we don't have any constraints of the class distribution in our sets during the cross validation. Since we're only shuffling the data, it could mean that a large number of training examples from the same class could end up in one fold and thus impair our performance. The same issue is relevant in the cross validation of the CNN as well.

The feature value distribution of the data are in a interval from $-0.5$ to 1. Over this interval the corresponding class is also spread out for each feature. That is, for all the training examples of a feature, the corresponding class of the training examples are evenly distributed over that features value range. One class is not prominently linked to lower feature values. All this makes the data seem almost normalised already, but nevertheless we confine the feature values in the range 0 to 1. This does improves the performance of our network from 19% to 23%.

We also explore the impact of outliers in our dataset. For the first feature, since the values are so confined, we change the definition to half a standard, rather than three. When carried out over all the feature distributions the performance measure is not affected in any detectable way. Thus the outliers can in fact be determined not being a problem, but rather a needed description of the underlying class.

Noisy training did not make any improvement in our performance either. Trying different initial standard deviations and decrease rates did not show any noticeable improvements.

## 4 Conclusion and Future Work

Our FER system only constructed of a SHLNN managed to reach above the baseline on SFEW. It reached a classification accuracy of 23%. The performance was improved by feature value normalisation, but not the other preprocessing techniques nor nosiy training. We had our hands tied by starting in the last stage of a FER system by just building a classifier for the calculated features. This gives us little room to preprocess the data since we can not change the data encoding, nor the features themselves. Thus, a different FER constructed of a CNN was evaluated which reached a classification accuracy of 31%. The improvement of performance between the SHLNN and CNN is clear. It's evident that having the freedom to fine tune and change larger part of an FER system, rather than being confined by pre-calculated features, results in better performance. The classification accuracy, despite improving and passing the baseline with a margin, shows that the tested FER systems are not viable. They do not achieve a sufficient performance to be usable, but the open nature of a CNN has potential to improve whole FER system. Nevertheless, SFEW provides useful data of close to real world scenarios and a good tool for further development in the field of FER.

Future work should contain balancing the uneven amount of class-wise training examples. This could be done by generating more data for the deficient classes or by introducing a weighted loss function over the classes, balancing out the effect of an uneven dataset. The implemented cross validation did not take into account to keep an even class distribution in the training and test sets either. By setting up equilateral encoding of our features it could solve this problem. All this would possibly improve the overall performance of the networks by having a more even and stable training process.

Regarding the CNN one could try different classic network structures as VGG or AlexNet to see if they would work better in an FER system. Increasing the resolution in the cropped faces could also prove to be useful and consequently enable a deeper network if the amount of data could supply it.

## References


1. R. A. Bustos & T. D. Gedeon (1995). *Decrypting Neural Network Data: A GIS Case Study.* Artificial Neural Nets and Genetic Algorithms, pp. 231-234.
2. A. Dhall, R. Goecke, S. Lucey & T. D. Gedeon (2011, November). *Static facial expressions in tough conditions: Data, evaluation protocol and benchmark.* 1st IEEE International Workshop on Benchmarking Facial Image Analysis Technologies BeFIT, ICCV2011.
3. A. Dhall, R. Goecke, S. Lucey & T. D. Gedeon (2011, September). *Acted Facial Expressions in the Wild Database.* ANU Computer Science Technical Report Series.
4. M. J. Lyons, S. Akamatsu, M. Kamachi & J. Gyoba (1998). *Coding facial expressions with gabor wavelets.* Proceedings of the IEEE International Conference on Automatic Face Ges- ture Recognition and Workshops, FG'98.
5. R. Gross, I. Matthews, J. Cohn, T. Kanade & S. Baker (2008). *Multi-PIE.* Proceedings of the Eighth IEEE International Conference on Automatic Face and Gesture Recognition, FG'2008, pp. 1–8.
6. Y. LeCun, L. Bottou, Y. Bengio & P. Haffner (1998). *Gradient-Based Learning Applied to Document Recognition.* Proceedings of the IEEE November, 1998.
7. K. Zhang, Z. Zhang, Z. Li & Y. Qiao (2016). *Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks.* IEEE Signal Processing Letters (SPL), vol. 23, no. 10, pp. 1499-1503, 2016
8. T. Elser. *Pretrained Pytorch face detection (MTCNN) and recognition (InceptionResnet) models.* Link: https://github.com/timesler/facenet-pytorchreferences
9. H. Kurt, M. Stinchombe & H. White (1989). *Multilayer Feedforward Networks are Universal Approximators.* Neural Networks, Vol. 2, pp. 359-366.
10. D. P. Kingma & J. L. Ba (2015). *Adam: A Method for Stochastic Optimzation.* ICLR 2015.