Examining the effect of hyperparameters on the training of a residual network for emotion recognition

Abe Winters
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
a.winters-1@student.utwente.nl

ABSTRACT

Setting hyperparameters is a complex and hard task when training a neural network. Neural networks performing badly can be the effect of a sub-optimal hyperparameters setup before training. This study aims to examine the effects of hyperparameters on the training of a residual network, while using a small dataset. This network is a SE-ResNet-50 that is being used to recognize emotions from facial expressions. Different methods, both constant values as well as schedules, are compared based on the validation loss and validation accuracy.
For interesting outcomes, an examination is done to see whether these methods transfer well to a similar dataset in the same context of emotion recognition in facial expressions.
The study shows that low batch sizes contribute to a good performance, but tend to result in an unstable network. A balance must be found between a high accuracy and a stable training. Learning rate schedules that use small step sizes outperform those which make larger adjustments. Momentum is a valuable addition, but too high of a momentum shows signs of overfitting in the loss graph. A cyclical momentum causes the training to become unstable, this can be reduced by using higher batch sizes.

Keywords

deep learning, emotion recognition, hyperparameters, residual network

1. INTRODUCTION

Within the field of machine learning, the aim is to get a high accuracy and having a network that is able to generalize well. Tuning hyperparameters can be difficult and requires a lot of trial and error, since there are a lot of different techniques and theories. The aim of this report is to provide insights in finding optimal hyperparameters for a residual network with a small dataset, and use them to get a balance between overfitting and underfitting.

This study makes use of a residual network with Squeeze and Excitation (SE) blocks which aims to recognize emotions from images and videos of faces. This network is used in a broader application of multimodal learning, where the results of the analysis of video, speech and text are combined to make a prediction on the corresponding emotion. This field of emotion recognition is an emerging research field, combining artificial intelligence and psychology. It marks the context in which this research is applied.

Recognizing emotion from faces is a fundamental human trait. Both facial expressions as well as verbal expressions play a part in this. This ability is developed as a child and its accuracy increases with age [3]. Facial emotion recognition matures around the age of 11 and vocal emotion recognition develops further during childhood [4].

For computers, recognizing emotions is a difficult task because humans can show emotion on different levels and even express compound emotions. Emotional facial expressions can differ between cultures and individuals. Paul Ekman showed that that there are universal facial expressions for basic emotions [5]. These basic facial expressions are used in the datasets of this study.

Whereas the previous studies have been done on large datasets, this study makes use of two relatively small datasets. The aim is to evaluate existing methods and study if they can be used on a smaller crowd-sourced dataset, what characterizes an optimal method and whether these characteristics transfer to a different dataset for the same application. This application is the field of emotion recognition from facial expressions.

2. BACKGROUND

2.1 Residual Network

A convolutional network (CNN) is a deep neural network, often used for recognizing images. A regular neural network consists of layers composed of a set of fully connected neurons, whereas a convolutional neural network makes use of convolutional layers. Such a layer is organized in three dimensions. The first two dimensions are often the width and height of an image. The third dimension is the amount of these images stacked on top of each other, for example the color channels. The convolutional layers are used for feature extraction. A CNN also has a classification part at the end, used as a classifier to predict a corresponding class to the input image.

Residual networks were proposed by He et al. [7]. They use a *skip connection* identity mapping, that adds the output from the previous layer to the layer ahead. This results in the ability to train deeper networks than was previously possible.

This study makes use of a Residual Squeeze and Excitation architecture. This is a residual network that makes use of so called SE blocks. Normally in a CNN, each channel is weighed equally when creating feature maps. An SE block is a content aware mechanism that

32nd *Twente Student Conference on IT* Jan. 31st, 2020, Enschede, The Netherlands.

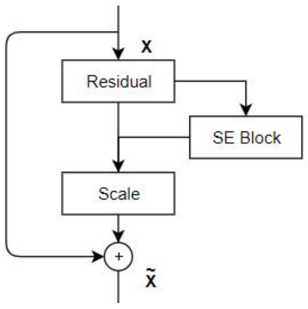


Figure 1: A residual layer with a SE block

weighs each channel adaptively. This way it can decide how relevant each channel is. By stacking these blocks a so-called SENet can be formed, which generalizes effectively across datasets [9]. These blocks can also be used in existing state-of-the-art, like a residual network, where it improves performance. Figure 1 shows how a SE block is fitted into a residual block.

When training neural networks, the goal is to get the highest accuracy while reducing the training time. Training involves tuning the set of inner parameters, or weights, to fit a desired outcome. The weights are learned during the training process and not manually set. This training process is iterative, which means that it progresses in a step-wise fashion.
Optimization algorithms tune these weights. A popular algorithm is called Gradient Descent. It tunes the weights by travelling down the slope of the estimated error in steps until it reaches a minimum. The step size that is taken is calculated by multiplying the gradient with the learning rate. This error function is also called the loss function.
The dataset is often split in a training set and a validation set. The network trains and adjusts it weights based on the performance on the training data, and validates it on the validation data, which does not affect the weight updates. The performance on the validation data represents how the network responds to unseen data, also called generalization.
A network can be really accurate on a training set, but its performance on unseen data is what eventually matters. This training accuracy and validation accuracy comes with two important terms in evaluating the performance of a network:
Overfitting is what happens when a network is really accurate on the training data, but inaccurate on the test data. This means that the model fits the training data too well, and thus increases the generalization error.

Underfitting is when a model fails to be accurate on both the training and the test set.
These terms are illustrated in Figure 2.

2.2 Hyperparameters

Hyperparameters are a different type of parameter than weights. A *hyperparameter* is a parameter that can be configured before the training process. The difference between other parameters and hyperparameters is that a hyperparameter is used in the parameter updating function and thereby specifies how the network parameters are learnt, whereas the values of other parameters are derived in training.
The amount of epochs is a hyperparameter that corresponds to the number of times the whole training set is shown to the network.

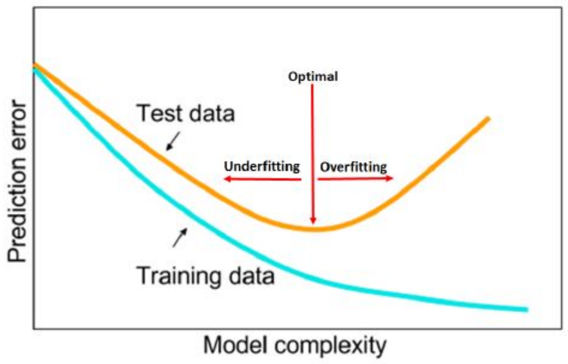


Figure 2: Illustrative explanation of the trade off between overfitting and underfitting. Model complexity refers to the power of the network. Image from Smith [12].

The hyperparameters that are being investigated in this work are:
Learning rate depicts how much the weights are changed during the training of the model.
A learning rate too large may result in a sub-optimal weight set, not letting the model get narrower and missing local optima. When the learning rate is too low, it makes the training process very slow and can result in overfitting. During the training process, the learning rate itself can be adjusted using a *learning rate schedule.*

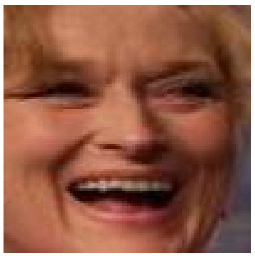
Momentum is a value between 0 and 1 that adds a fraction m of the previous weight update to the new update to make sure a consistent heading through the space is made. It smoothens out small fluctuations. Compared to a ball rolling down a hill, a small bump won't change the rolling direction for this tiny moment, the physical momentum keeps the ball rolling. It is closely related to the learning rate. The optimal learning rate is dependent on the momentum and vice versa [12].

Batch size refers to the size of the training samples processed before the model is being updated. This means the batch size impacts the amount of parameter updates during training, and affects the training time. A large batch size can lead to poor generalization. This difference in performance between training and testing data is called the *generalization gap.* In section 4 a study from Hoffer et al. [8] is discussed, which tells that the generalization gap is more related to the small number of updates rather than the size of a batch.

Weight decay, also known as *L2 Regularization* is used to prevent models from getting overly complex. In the loss function the sum of squares of all weights is multiplied with the weight decay and that result is added to the loss function.

2.3 Datasets

In this study, two datasets are used. RAF-DB and FER+. The Real-world Affective Faces Database (RAF-DB) [10] is a data set which contains around 30000 images of faces from thousands of individuals that have been labeled 40 times. The labels are based on the seven basic emotions from Paul Ekman [6], namely: anger, disgust, fear, happiness, sadness, surprise and neutral. The database consists of two subsets, one single-label subset with the aforementioned labels, and one compound label subset consisting of 12 labels. This study only makes use of the single-label subset, which contains 15339 images divided over a training and testing set with 12271 and 3068 images re-


(a) Image from the RAF dataset portraying happiness

(b) Image from the FER+ dataset portraying anger

Figure 3: Example images from the datasets used

spectively. All images are coloured and 100x100 pixels in size. An example can be found in Figure 3a.

The *FER+* dataset [1] is a dataset of around 35000 greyscale images of faces labeled with an emotion, based upon the original FER dataset that has been prepared by Pierre Luc Carrier and Aaron Courville. Carrier and Courville used web crawling to get face images with related emotion keywords, which were filtered by human labelers. The accuracy was not very high. The FER+ dataset makes use of these images, but they have been re-tagged with the use of crowd sourcing. This dataset uses 8 emotion labels based on the basic emotions from Paul Ekman: neutral, happiness, surprise, sadness, anger, disgust, fear and contempt. All images are 50x50 pixels in size. An example can be found in Figure 3b.

3. AIM OF THIS RESEARCH

The aim of this paper is to evaluate the behaviour of different hyperparameters on two datasets and in relation to each other. The focus is to optimize these hyperparameters and study whether existing methods can be used on a smaller crowd-sourced dataset. Validation is used to see whether these behaviours are dataset depended, or apply to a different dataset in the same context.
The following research questions are addressed in order to reach the aim of this research:

RQ1 How can hyperparameters be optimized to increase learning robustness of a neural network.

- **RQ1.1** How can *momentum* be optimized for the learning process?
- **RQ1.2** How can *batch size* be optimized for the learning process?
- **RQ1.3** How can *learning rate scheduling* be optimized for the learning process?
- **RQ1.4** How can *weight decay* be optimized for the learning process?

RQ2 Will these optimal techniques apply to a similar dataset in the same context?

4. RELATED WORK

The research field of optimization is broad, with both hyperparameters being investigated as well as structures of neural networks. Different optimization methods are discussed in this section.

4.1 CNN Optimization

The scaling of convolutional neural networks is studied in a research by Tan and Le [15]. Scaling a network is often used for models to increase accuracy. However, scaling is not well understood and there exist multiple methods to do it. A popular method is by increasing the depth, other common methods scale by one dimension as well. They propose a scaling method that scales all dimensions of depth/width/resolution using a compound coefficient. This results in much better accuracy and efficiency than achieved with previous convolutional neural networks.

4.2 Multiple hyperparameter optimization

Grid search is a widely used strategy to optimize hyperparameters. This is a sweep through a manually specified subset of the hyperparameters space, trying all combinations. It is an extensive trial and error method. An advantage is that it can be easily parallelized, but it is inefficient and computationally expensive. Bergstra and Bengio [2] suggest using a random search to tune hyperparameters. A random search samples this hyperparameter space randomly, and is able to find models that are as good or better than those found with a grid search. They suggest that not all hyperparameters are equally important to tune, and this importance differs per model and dataset. Grid search would then allocate too much time on exploring dimensions of irrelevant hyperparameters.

Another common technique is Bayesian Optimization. The idea is to build a probabilistic model of the objective function, and using this to find the most promising hyperparameters. By using past results, hyperparameters are mapped to a probability. The most commonly used model for this type of optimization is the Gaussian Process (GP). A drawback of GP is that it scales cubically. Snoek et al. [14] explore the use of neural networks instead of GPs to model distributions over functions. They call this deep networks for global optimization, or DNGO. Their solution scales linearly with the number of data, and allows parallelism. They show this technique is well suited for large scale hyperparameter optimization.

4.3 Batch size

Hoffer et al. [8] researched the influence of a large batch size on the generalization gap. A problem often encountered is when the batch size increases, the performance in generalization goes down. This is a phenomenon called the *generalization gap*. They showed that the generalization gap is not related to the batch size, but comes from the relatively small amount of updates. To close the generalization gap between small and large batch strategies, they propose to:

- Use Stochastic Gradient Descent with momentum, gradient clipping and a decreasing learning rate schedule
- Adapt learning rate to the batch size
- Use a sufficient number of high learning rate iterations

They used a momentum SGD with a fixed learning rate that decreases exponentially every few epochs.

4.4 Learning rate

Smith et al. [13] examine the possibility to, instead of decaying the learning rate according to a schedule, increase the batch size according to the same schedule. So when the learning rate drops by a factor α instead increase the batch size by a factor α. They show that while both options reach the same test accuracy, increasing the batch size significantly reduces the amount of required parameter updates. With B as the batch size, ϵ as the learning rate and m as the momentum, increasing the batch size with the $B \propto \epsilon/(1-m)$ relation turned out to require the least amount of updates while maintaining a high validation accuracy.

Smith [12] discusses early signs of overfitting or underfitting visible in the test loss. A test loss curve that keeps decreasing at the end of training is a sign of underfitting, which can be reduced by increasing the learning rate. A loss curve that increases at the end of training is a sign of overfitting.
Because momentum and learning rate are closely connected, Smith argues that an increasing cyclic learning rate and a decreasing cyclic momentum is an optimal training procedure. In this case, both cycle only once. This is called 1cycle. A decreasing cyclical momentum shows the same result in terms of accuracy as a constant momentum, but stabilizes the training to allow larger learning rates. When a constant learning rate is used, a constant momentum in the range of 0.9 - 0.99 is advised. In addition, he discusses how adjustments to hyperparameters affect overfitting and underfitting. He suggests using the Learning rate range test on different weight decays to find good range for the learning rate and a good value for the weight decay.

This range of learning rates can then be used for a *cyclical learning rate* scheduler, which cycles the learning rate between two bounds.

4.4.1 Learning rate schedules

Step decay is a learning rate schedule that decreases the learning rate step-wise over the duration of the training. The starting learning rate is specified and decreased every few epochs by a factor. This factor and the number of epochs for the drop can be specified before the training.

A **cyclical learning rate** cycles the learning rate between a lower bound and an upper bound. A cycle consists of the increase from the base learning rate to the maximum learning rate, and back to the base learning rate again. This method has been proposed by Leslie Smith [11].
These bounds can be found using the **learning rate range test.** For this test the model runs for a few epochs while the learning rate increases to a maximum set value. When the loss increases extremely rapidly, the test is stopped early. The accuracy will be plotted against the learning rate. The optimal range is the range where the accuracy is rising or stable until it starts decreasing.

The **1cycle policy** is a variation of the cyclical learning rate. Here, only one cycle is made for the duration of the training. For the final few epochs, the learning rate drops to a lower value very quickly. An example is shown in Figure 4.

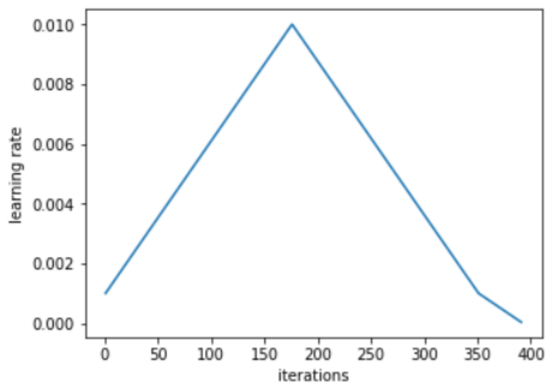


Figure 4: Illustration of the 1cycle policy, where the learning rate cycles once between two bounds and decreases at the end. Image from: S. Gugger, "The 1cycle policy", Another data science student's blog, 2018. https://sgugger.github.io/the-1cycle-policy.htmlthe-1cycle-policy

5. PROPOSED FRAMEWORK

The experiments conducted examine different hyperparameters, and their relations to each other. Constant values and scheduled values are compared.
Most of the research is performed on the RAF-DB dataset. Optimal methods and settings that have been found are used on the FER+ dataset, and an examination is done whether this yields similar results. This way it becomes clear whether these techniques are dataset dependent, or apply to similar datasets in the same context.

The first experiments conducted are the learning rate range tests with different weight decays. The tests increase the learning rate from 0.001 until 1.0. This test can be seen as a learning rate scan which shows information about how the network behaves on different learning rates. The accuracy is plotted as a function to the learning rate. A good range is defined by an increasing or stable accuracy until the accuracy drops. For a constant value, the learning rate with the highest accuracy in this range is selected.
The weight decay corresponding to test with the best range is selected as a constant weight decay value for the remaining experiments.

The optimal range for the learning rate resulting from the previous experiments is selected to perform cyclic learning rate tests. Constant and scheduled learning rates are tested against constant momentum values of 0.99, 0.97, 0.95, 0.9.
The best outcome of these constant values serve as the upper bound for a momentum schedule called *cyclical momentum.* This follows the principle of the 1cycle policy, but in an opposite fashion. When the learning rate increases, the momentum decreases and vice versa. The cyclical momentum has got a lower bound of 0.8.
The batch size remains constant.

The learning rate schedules that have been tested are:

- Constant learning rate
- Cyclic learning rate
- 1Cycle policy
- Step decay

Apart from the learning rate tests, experiments on varying constant batch sizes have been conducted. Tests are performed whether lower or higher batch sizes perform better.

The validation loss and validation accuracy are examined to compare the performance of the different hyperparameter settings.

6. RESULTS

This section presents the results of the experiments proposed in section 5. The results of the experiments are split per hyperparameter, starting with the learning rate range

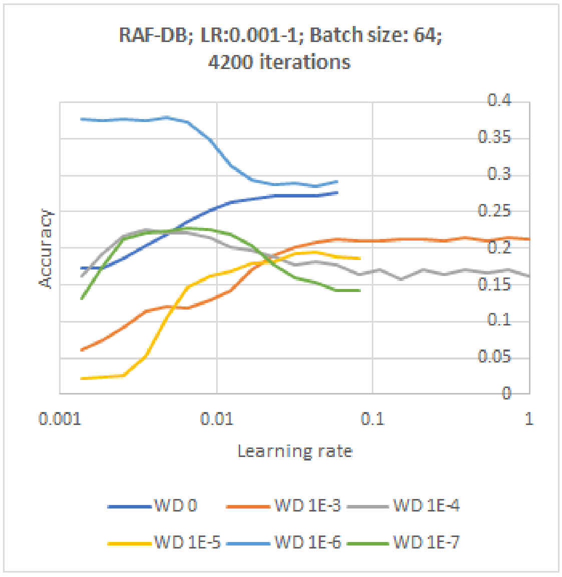


Figure 5: Learning rate range test. The accuracy of multiple learning rate range tests is plotted against the learning rate. The tests vary in weight decay.

test. For the remaining tests, training and test accuracy are presented alongside the validation loss for the experiments. The research is conducted on a SENet50, using a stochastic gradient descent optimization algorithm.

6.1 Learning rate range test

A learning rate range test has been performed on different weight decays. The test ran for 22 epochs, where the learning rate increases exponentially from 0.001 to 1.0. The results can be seen in Figure 5, where the accuracy is plotted against the learning rate.
Concluding from this test, a learning rate of 0.006 with a weight decay of 1x10-6 is the best option. For a cyclical learning rate schedule, a range from 0.001 - 0.006 would be optimal.

6.2 Learning rate schedules

Different learning rate schedules are tested under the same conditions. All were tested with a weight decay of 1E-6, momentum of 0, batch size of 64 and all for 64 epochs. See Table 1 for the results highlighted in this section. The full test results can be found in Appendix A.

We can see that a learning rate schedule that varies with small steps performs better than a schedule with big variations in the learning rate. The step decay with a drop factor of 0.5 has the biggest variations in the learning rate, and the worst performer. A constant learning rate shows a validation accuracy. Since the 1cycle policy changes the learning rate with very small steps, it performs better than the other schedules.
A cyclical learning rate which cycles multiple times results in an unstable training, which can be seen in Figure 6 of training loss. The 1cycle policy however, results in a stable training but takes longer to converge than the other schedules.

6.3 Momentum

Several experiments have been done that tested multiple learning rate schedules against different momenta, of which the complete results can be found in Appendix A. The results presented here in Table 2 are focused on the learning rate policy which gave the highest accuracy: the 1cycle policy.

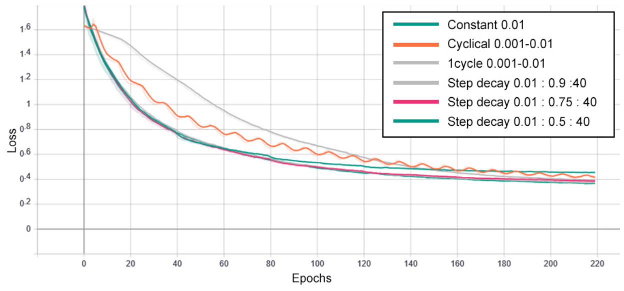


Figure 6: Loss graph of experiments on different learning rate schedules. The training loss is plotted against the epochs. The instability of the cyclical learning rate can be seen by the pulsing behaviour in the graph. The 1cycle policy converges slower, as can be seen by the slower decline.

Table 1: Results achieved with different learning rate schedules. Weight decay = 1E-6, Momentum = 0, Batch size = 64. For 220 Epochs

Schedule	Training Accuracy (%)	Validation Accuracy (%)	Validation loss
Constant: 0.01	93.75	75.15	0.8272
Cyclical: 0.001-0.01	91.75	73.28	0.8533
1cycle: 0.001-0.01	92.65	75.34	0.8097
Step decay: 0.01 : 0.9 : 40	93.41	74.49	0.8411
Step decay: 0.01 : 0.75 : 40	92.94	73.61	0.8186
Step decay: 0.01 : 0.5 : 40	89.62	74.10	0.8261

The addition of momentum makes a significant improvement in the performance of the network. It makes sure the network converges faster, increases the accuracy and decreases the loss. This is validated with the other experiments on three different learning rate policies.
When the momentum is too large for the learning rate, the loss function starts to increase after hitting a minimum and thus the run ends with a high loss value. This is a sign of overfitting. The higher the momentum, the steeper this increase. This means a balance should be found between high validation accuracy and a low validation loss.

6.4 Batch size

Tests have been done on both increasing as well as lowering the batch size. This can be seen in Table 3. The full test results are in Appendix B.

Lowering the batch size results in a slightly higher validation accuracy. This is because when the sample size is lower, more noise of the dataset is being included in the

Table 2: Results achieved by testing the 1cycle learning rate policy against different momenta. Weight decay = 1E-6, Batch size = 64. For 220 epochs

Momentum	Training Accuracy (%)	Validation Accuracy (%)	Validation loss
0	92.65	75.34	0.8097
0.9	97.57	80.42	0.751
0.95	97.99	80.62	0.7736
0.97	98.48	82.65	0.7753
0.99	98.79	83.95	0.8119
Cyclic: 0.9-0.8	96.75	79.93	0.7614

Table 3: Results achieved by batch size tests for a constant and a varying learning rate. The tests ran for 220 epochs

LR	Batch size	Momentum	Validation Accuracy (%)	Validation loss
0.01	32	0.9	82.97	0.8063
0.01	64	0.9	80.09	0.8008
0.01	128	0.9	80.78	0.7862
0.01	256	0.9	74.1	0.7768
0.001-0.01 (1cycle)	16	0.9-0.8	82.62	0.7373
0.001-0.01 (1cycle)	32	0.9-0.8	81.96	0.7373
0.001-0.01 (1cycle)	64	0.9-0.8	79.93	0.7614
0.001-0.01 (1cycle)	128	0.9-0.8	76.8	0.8065

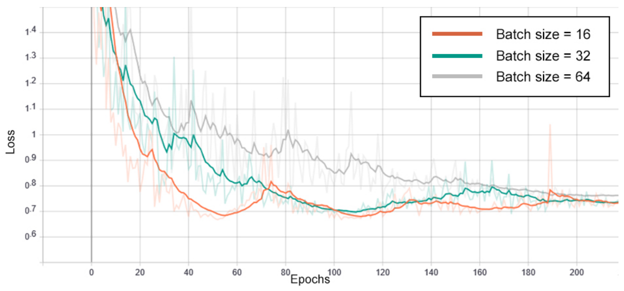


Figure 7: Loss graphs of three batch size experiments. The instability caused by a low batch size can be observed by looking at the multiple minima.

training of the network.
However, too low of a batch size results in an unstable network, which can be seen in the loss graph of Figure 7. This way it starts fitting too much to the training data. A cyclical momentum increases this instability. This means that a cyclical momentum needs a higher batch size for the network to be stable.
So, a balance must be found between the network stability and accuracy. In this context a cyclical momentum with batch sizes of 32 and 16 are unstable. 64 is a good value, maintaining a good stability and a decent validation accuracy and validation loss.

6.5 Validation on FER+

Interesting outcomes of the experiments on the RAF dataset have been tested on the FER+ dataset for means of validating these results. The validation is done on the momentum and batch size experiments. So whether the addition of momentum results in a significant performance improvement and whether lower batch sizes increase accuracy and causes instability.

6.5.1 Momentum validation

The momentum tests are validated on a 1cycle policy that cycles between 0.001 and 0.01 with a batch size of 64, weight decay of 1E-6 and is trained for 220 epochs. This policy has been chosen because of the good results in the previous tests on the RAF dataset. The results can be found in Table 4.

We can see that momentum is a valuable addition in terms of accuracy. With the increase in momentum, the accuracy increases and the network converges faster. For higher momentum values, the loss function starts to increase again after hitting a minimum. The validation loss of the experiment without momentum is lower than the experiments with a momentum above 0.8, this can be explained by the aforementioned phenomena. When using cyclical momentum, the training became unstable.

Table 4: Results of validation on FER+: Testing different momenta for 220 epochs. 1cycle policy (0.001-0.01); Batch size = 64; Weight decay = 1E-6

Momentum	Training Accuracy (%)	Validation Accuracy (%)	Validation loss
0	91.65	80.81	0.7086
0.8	93.63	83.46	0.6835
0.9	94.73	83.95	0.7118
0.95	95.62	84.53	0.7758
0.97	95.84	85.52	0.792
0.99	96.12	86.02	0.8277
Cyclic: 0.9-0.8	92.78	83.26	0.7003

Table 5: Results of validation on FER+: Testing different batch sizes for 220 epochs; 1cycle policy (0.001-0.01); Cyclical momentum (0.9-0.8); Weight decay = 1E-6

Batch size	Training Accuracy (%)	Validation Accuracy (%)	Validation loss
32	93.23	83.63	0.7037
64	92.78	83.26	0.7003
128	92.15	81.29	0.6728
256	91.21	79.94	0.7628

6.5.2 Batch size validation

Different constant batch sizes are tested for 220 epochs with a 1cycle policy that cycles between 0.001 and 0.01, with a cyclical momentum (0.8-0.9) and a weight decay of 1E-6. The results are shown in Table 5

Increasing the batch size results in a decrease of the accuracy, just like on the RAF dataset. However, for batch size 32 and 64 the loss and accuracy graphs show instabilities in the training. When the batch size is too low the network fits too much to the training data. Increasing the batch size results in a more stable network. This explains why a batch size of 128 has the best validation loss and a stable training.

7. DISCUSSION

After testing both on the RAF dataset and the FER+ dataset, some similarities can be noticed.

In both cases a lower batch size contributes to a higher accuracy than higher batch sizes, while a lower batch size also caused instabilities in the training. In the context of the cyclical momentum, RAF-DB had an ideal batch size of 64, FER+ had an ideal batch size of 128 for a 1cycle policy. If we compare the ideal batch size and the number of iterations with one another, the following constant can be deducted:

RAF: 12211 samples, BS=64, 191 iterations/epoch
FER: 26353 samples, BS=128, 206 iterations/epoch

So we can see that a batch size that results in approximately 200 iterations per epoch is the optimal balance between a stable training and a good accuracy for the 1cycle policy with cyclical momentum.

There were some limitations to this research.
For the learning rate range test, I first looked at the loss graph for my interpretations. But you can not

compare the absolute loss values since the weight decay is an addition to the loss function. Based on this initial interpretation I chose the constant learning rate 0.01 and the range 0.001 - 0.01. When looking at the accuracy, I see that I could have used a smaller range for the cyclical schedules, namely the range of 0.001 - 0.006. Using this range could have resulted in higher accuracy during training.
In the FER+ validation tests on momentum, the batch size used resulted in a small instability in the training. A higher batch size would have resulted in a more stable network to compare these momentum values to. Nevertheless, using this batch size proved the way that cyclical momentum increases the network instability.
The final limitation is the difference in datasets. The FER+ dataset used 8 labels instead of 7, like RAF-DB does. The comparison would be better if both datasets used 7 labels.

8. CONCLUSION AND FUTURE WORK

The results presented in this study show that learning rate schedules that vary little each step perform better than learning rate schedules that make big adjustments to the learning rate. A small step-size is preferred.

Furthermore, the results show that no momentum always performs worse than a training where momentum is added. When the momentum becomes too close to 1, the loss graph will show signs of overfitting. A momentum of 0.97 shows a good balance. Cyclical momentum increases the instability in the training, so it requires a higher batch size to acquire stability than a constant momentum.

This paper shows that a training with a smaller batch size outperforms using a large batch size in terms of accuracy, but results in unstable training when the batch size is too low. For combining 1cycle with a cyclical momentum, aiming at approximately 200 iterations per epoch is a good balance between high accuracy and a stable training.

In future research, validation can be done on other network architectures to confirm whether these findings are characteristic for this SE-ResNet-50, or whether it is a more general phenomenon.
More learning rate schedules can be tested, to investigate the behaviour of these small-step schedules.

9. ACKNOWLEDGEMENTS

I would like to thank my supervisor Nicola Strisciuglio for his guidance and feedback throughout my research. Furthermore, I would like to thank Vincenzo Vigilante for providing me with a neural network to experiment on and helping me along the way.

10. REFERENCES

[1] E. Barsoum, C. Zhang, C. C. Ferrer, and Z. Zhang. Training deep networks for facial expression recognition with crowd-sourced label distribution.

[2] J. Bergstra, J. B. Ca, and Y. B. Ca. Random search for hyper-parameter optimization yoshua bengio, 2012.

[3] L. A. Camras and K. Allison. Children's understanding of emotional facial expressions and verbal labels.

[4] G. Chronaki, J. A. Hadwin, M. Garner, P. Maurage, and E. J. S. Sonuga-Barke. The development of emotion recognition from facial expressions and non-linguistic vocalizations during childhood. *British Journal of Developmental Psychology*, 33:218–236, 6 2015.

[5] P. Ekman. Universal facial expressions of emotion. *California Mental Health Research Digest*, 8:151–158, 1970.

[6] P. Ekman. Facial expression and emotion. *American Psychologist*, 48:384–392, 1993.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. volume 2016-December, pages 770–778. IEEE Computer Society, 12 2016.

[8] E. Hoffer, I. Hubara, and D. Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. 5 2017.

[9] J. Hu. Squeeze-and-excitation networks.

[10] S. Li, W. Deng, and J. Du. Reliable crowdsourcing and deep locality-preserving learning for expression recognition in the wild.

[11] L. N. Smith. Cyclical learning rates for training neural networks.

[12] L. N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. 3 2018.

[13] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. L. G. Brain. Don't decay the learning rate, increase the batch size.

[14] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Mostofa, A. Patwary, and R. P. Adams. Scalable bayesian optimization using deep neural networks prabhat prabhat@lbl.gov.

[15] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks.

APPENDIX

A. RESULTS ON THE LEARNING RATE AND MOMENTUM EXPERIMENTS

Table 6

Schedule	Learning rate	Momentum	Weight Decay	Training Accuracy (%)	Validation Accuracy (%)	Validation loss (0.6 smoothing)
Constant	0.01	0	0.000001	93.75	75.15	0.8272
Constant	0.01	0.9	0.000001	97.85	80.09	0.8008
Cyclical – 22 cycles	0.001 – 0.01	0	0.000001	91.75	73.28	0.8533
Cyclical – 22 cycles	0.001 – 0.01	0.9	0.000001	97.47	82.06	0.7594
Cyclical – 22 cycles	0.001 – 0.01	0.95	0.000001	97.72	81.73	0.7849 (overfit)
Cyclical – 22 cycles	0.001 – 0.01	0.97	0.000001	98.20	83.63	0.7591 (overfit)
Cyclical – 22 cycles	0.001 – 0.01	0.99	0.000001	98.25	82.65	0.7856 (overfit)
Cyclical – 1cycle	0.001 – 0.01	0	0.000001	92.65	75.34	0.8097
Cyclical – 1cycle	0.001 – 0.01	0.9	0.000001	97.57	80.42	0.751
Cyclical – 1cycle	0.001 – 0.01	0.95	0.000001	97.99	80.62	0.7736
Cyclical – 1cycle	0.001 – 0.01	0.97	0.000001	98.48	82.65	0.7753 (small overfit)
Cyclical – 1cycle	0.001 – 0.01	0.99	0.000001	98.79	83.95	0.8119 (overfit)
Cyclical – 1cycle	0.01-0.1	0.9-0.8	0.000001	98.7	83.4	0.779 (small overfit)
Cyclical – 1cycle	0.001-0.01	0.9-0.8	0.000001	96.75	79.93	0.7614
Step decay	0.01 : 0.75 : 40	0	0.0000001	92.54	75.77	0.8078
Step decay	0.01 : 0.5 : 40	0	0.0000001	89.49	74.82	0.7921
Step decay	0.01 : 0.25 : 40	0	0.0000001	83.46	72.72	0.8104
Step decay	0.01 : 0.9 : 40	0	0.000001	93.41	74.49	0.8411
Step decay	0.01 : 0.75 : 40	0	0.000001	92.94	73.61	0.8186
Step decay	0.01 : 0.5 : 40	0	0.000001	89.62	74.10	0.8261
Step decay	0.01 : 0.75 : 40	0.9	0.000001	97.65	82.12	0.768 (small overfit)
Step decay	0.01 : 0.75 : 40	0.95	0.000001	97.81	82.55	0.7546 (small overfit)
Step decay	0.01 : 0.75 : 40	0.97	0.000001	97.47	79.31	0.8161 (small overfit)

B. RESULTS ON THE BATCH SIZE EXPERIMENTS

Table 7

LR	Batch size	Momentum	Training Accuracy (%)	Validation Accuracy (%)	Validation loss (0.6 smoothing)
0.01	32	0.9	97.85	82.97	0.8063
0.01	64	0.9	97.85	80.09	0.8008
0.01	128	0.9	97.12	80.78	0.7862
0.01	256	0.9	94.92	74.1	0.7768
0.01-0.1 (1cycle)	16	0.9-0.8	98.55	84.05	0.7929
0.01-0.1 (1cycle)	32	0.9-0.8	98.71	84.54	0.778
0.01-0.1 (1cycle)	64	0.9-0.8	98.7	83.4	0.779 (small overfit)
0.01-0.1 (1cycle)	128	0.9-0.8	97.89	82.2	0.7717
0.01-0.1 (1cycle)	256	0.9-0.8	96.4	76.93	0.8927
0.001-0.01 (1cycle)	16	0.9-0.8	97.83	82.62	0.7373
0.001-0.01 (1cycle)	32	0.9-0.8	97.31	81.96	0.7373
0.001-0.01 (1cycle)	64	0.9-0.8	96.75	79.93	0.7614
0.001-0.01 (1cycle)	128	0.9-0.8	94.51	76.8	0.8065
0.001-0.01 (1cycle)	16	0.99	98.58	83.93	0.8486
0.001-0.01 (1cycle)	32	0.99	98.81	84.02	0.8238
0.001–0.01 (1cycle)	64	0.99	98.79	83.95	0.8119 (overfit)
0.001-0.01 (1cycle)	16	0.97	98.71	84.62	0.7604
0.001-0.01 (cylic)	32	0.97	98.02	83.2	0.7438
0.01–0.001 (cyclic)	64	0.97	98.20	83.63	0.7591 (overfit)