Imperial College
London

COURSEWORK 2

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Machine Learning (Course 395)

Authors:
Mario Zusag (CID: 00834438)
Anna Mitenkova (CID: 00927771)
Jack Collins (CID: 00829530)
Aditya Chaturvedi (CID: 00751251)

Date: March 8, 2018

1 Question 1.

Explain the forward pass.
The forward pass consists of applying of a linear transformation to the incoming data $\boldsymbol{X}$, namely $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, where $\boldsymbol{W}$ is the $weight\ matrix$ and $\boldsymbol{b}$ is the $bias$. The idea behind this transformation is that we want to determine the contribution of each x_i of the input $\boldsymbol{x}$, to the neuron output. The role of the bias term is to create an "adjustable" threshold, where the neuron fires, (by adding the bias term we can always set the threshold at 0).

Explain the backward pass.
The backward pass consists of backpropagation which starts at the last layer of the network and recursively applies the chain rule to compute the gradients of the parameters. The latter allows us to update the weight values, as well as the bias, that minimize the error function of the network over the set of the training examples. The formulas for the parameter updates can be seen in the code in the function $linear\_backwards(dout, X, W, b)$.

Explain the ReLU activations.
The **ReLU** activation consists of applying the Rectified Linear Unit function $max(0, x)$ to the net activation matrix. The idea behind the activation function is that it determines whether the neuron fires or not. The reason for using the **ReLU** function over other activation functions, especially the sigmoid function, is that it is less computationally expensive, as well as leads to faster convergence, which makes it easier to train networks. The **ReLU** function is also a way of restricting the capacity of the network: few neurons in the network do not activate, thereby making the activations sparse and efficient.

2 Question 2.

The idea behind the dropout is to prevent overfitting by preventing the neurons from co-adapting too much. While training, dropout is implemented by only keeping a neuron active with some probability q, or setting it to zero otherwise with probability p, where $p = 1 - q$. Note that when applying the dropout technique, it is crucial to perform the scaling. This is done to make sure that neuron outputs at test/training time are identical to their expected outputs at training/test time.

Explain the forward pass of dropout.
The forward pass of the dropout consists of applying in the forward pass the dropout technique, as described above, to each neuron output with a specified probability (defined p in the code, with the default value of 0.5). Note, that it is important that we differentiate between the training and test phases here. Since the test performance is critical, we leave the test forward pass unchanged (i.e. no dropout). For the training forward pass, we perform the dropout with the inverse scaling factor equal to $\frac{1}{1-p}$.

Explain the backward pass of dropout. The backward pass of the dropout is almost identical to that of the forward pass, but instead of being applied to $\boldsymbol{X}$-the data matrix, it is applied to the upstream derivative matrix.

3 Question 3

Explain the computation of softmax
The softmax classifier is an output activation function that outputs a discrete probability distribution (squashes the raw class scores into normalized positive values summing to one). Essentially we represent the class scores probabilistically. The most suitable error function for softmax is negative log likelihood, and is calculated using:

$$Loss = -\frac{1}{N}\sum_{j=0}^{N-1} ln(o_j) \tag{1}$$

Here, o_j is the j-th softmax output and the negative log loss is calculated for each output (class prediction score). This is averaged over all samples, stabilizing the behavior of the algorithm as the sample size changes.

Explanation of loss gradient
For the loss gradient we have to differentiate the negative log likelihood loss function with respect to output o_j. After applying the chain rule extended to the derivative of the softmax function, which is $o_i(1-o_j)$ for $i = j$ and $-o_j o_i$ for $i \neq j$, we get:

$$\frac{\partial Loss}{\partial o_j} = o_j - y_j \text{ with } \sum_{j=0}^{N-1} y_j = 1 \tag{2}$$

4 Question 4.

Explain parameters used to overfit a small subset of the CIFAR-10data. By using just a single layer with as many hidden nodes as there are data points (50) while keeping L2 regularization and dropout at 0, the model quickly overfit to the training data within 20 epochs. Hyper-parameters used were:

```
hidden_dims=[50],
dropout=0,
reg=0,
weight_scale=1e-2,
update_rule='sgd',
learning_rate=1e-3,
momentum=0.9,
lr_decay=0.95,
```

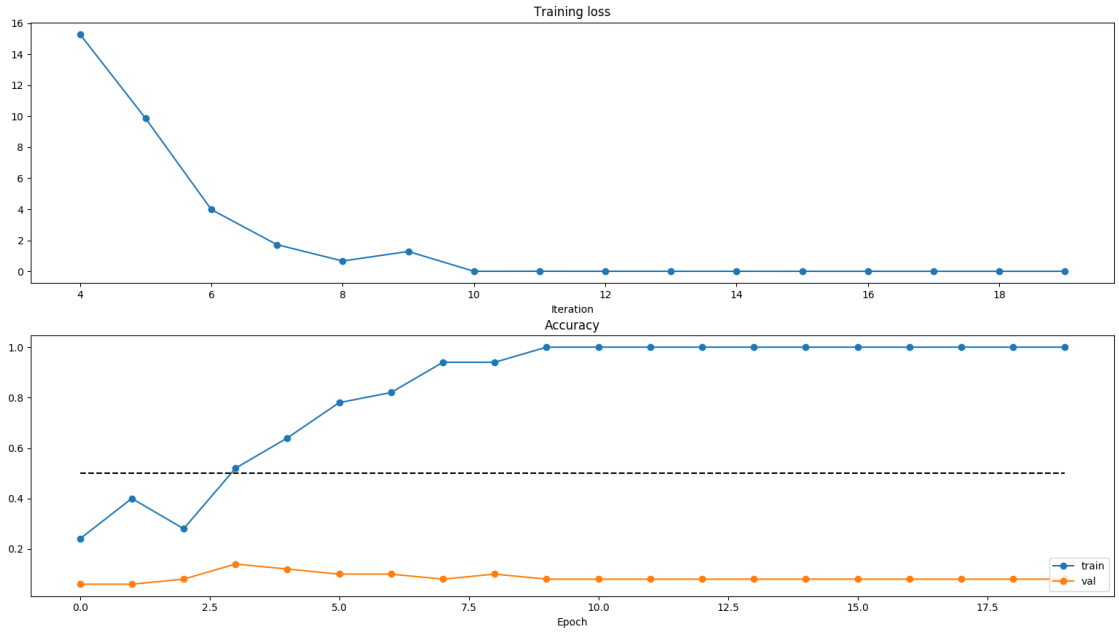

Figure 1: Training loss vs Iteration number and Accuracy vs Epoch of an overfit network on CIFAR10 dataset

```
num_epochs=20,
batch_size=100
```

Explain parameters used to achieve at least 50% accuracy on CIFAR-10. By using the following architecture and hyper-parameters, 51% accuracy was achieved on the CIFAR10 validation dataset (with 57% accuracy on the training set):

```
hidden_dims=[60, 75],
dropout=0,
reg=0.2,
weight_scale=1e-2)
update_rule='sgd_momentum'
learning_rate=2e-3,
momentum=0.9,
lr_decay=0.95,
num_epochs=40,
batch_size=100
```

5 Hyper-parameter Optimization with FER2013

For this part of the coursework we chose to convert the FER images to grayscale after judging that image color should not affect the ability to detect emotions in the

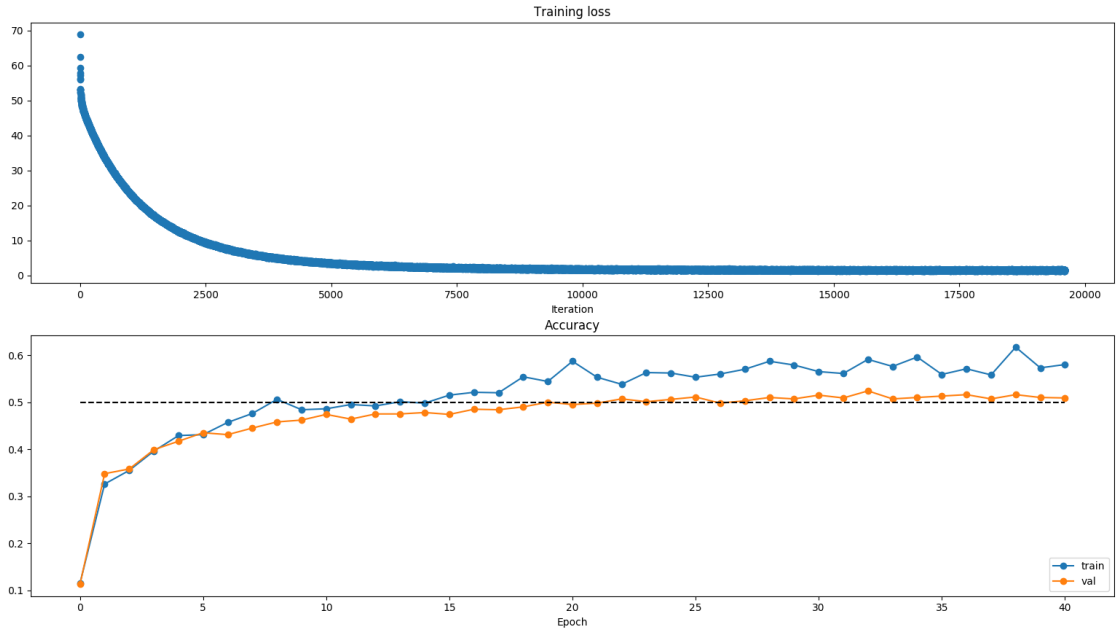

Figure 2: Training loss vs Iteration number and Accuracy vs Epoch of a two-layer fully-connected network for CIFAR10 dataset

image. This reduced the size of the input layer by 66% which decreased the training time. We also chose to subtract the mean image from each image to center the data.

The initial architecture chosen for the network was two layers each with 200 neurons. This number of neurons was chosen as it was of an order of magnitude roughly mid-way between the number of input neurons (2304) and the number of output neurons (7). We chose to use two layers to allow some complexity to be learned while keep training time short. The stopping criterion we defined in *solver.py* used the validation dataset accuracy. When this accuracy decreases for four epochs in a row, then the solver stops training. Learning rate decay appeared to achieve good results without need for a learning rate update schedule. For momentum, the update schedule linearly increased the momentum by $0.5/num\_epochs$ every epoch until it reached 0.9.

To find the optimal learning rate, the model was trained for learning rates between 1e-5 and 1e-1. Accuracy on the validation data failed to reach a limit with learning rates of 1e-4 or smaller, while learning rates of 1e-2 and higher caused the loss to become very unstable from one iteration to the next. Through this process 1e-3 was found to be the most effective learning rate.

Using dropout of 0.5 and 0.8 the validation accuracy was found to decrease from 34.78% with no regularization to 27.01% and 24.96% respectively. Using L2 reg-

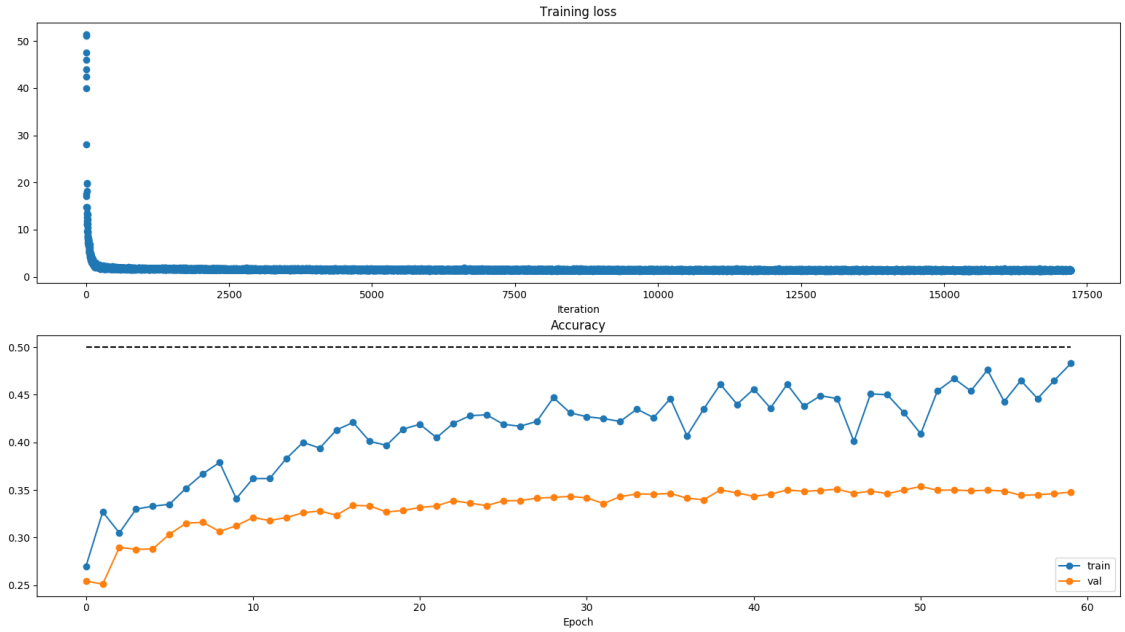

Figure 3: Training loss vs Iteration number and Accuracy vs Epoch of a two-layer fully-connected network with no dropout and no regularization for the FER2013 dataset

ularization of 0.1 and 0.2 the validation accuracies were found to be 38.68% and 40.19% respectively. Validation performance was significantly lower using dropout than L2 regularization.

To optimize the network further, Bayesian Optimization was used. Here the model is treated as a 'black box' function and the parameters adjusted in a way that investigates regions of most potential increase in validation accuracy. This was done just for two layered networks, after manual testing of deeper networks appeared to give poorer results. The hyper-parameters adjusted were batch size, number of neurons in each layer, learning rate, momentum, and the regularization coefficient. Due to previous poor results from dropout it was not included in this process. Using this method, several different sets of optimal hyper-parameters were found:

step num	validation accuracy	batch size	hidden 1 size	hidden 2 size	learning rate	mom-entum	reg coeff
9	0.43339	50	361	1000	1e-3.0000	0.0000	0.5000
27	0.43395	50	1000	525	1e-3.0000	0.0000	0.5000
28	0.43757	50	535	555	1e-3.0000	0.0000	0.4293
104	0.43841	50	901	543	1e-3.1557	0.6612	0.2854

Each of these solutions was investigated separately. The three solutions with a large number of neurons in either of the hidden layers were found to overfit the training data. So the remaining solution (third entry in above table) was take to be the opti-

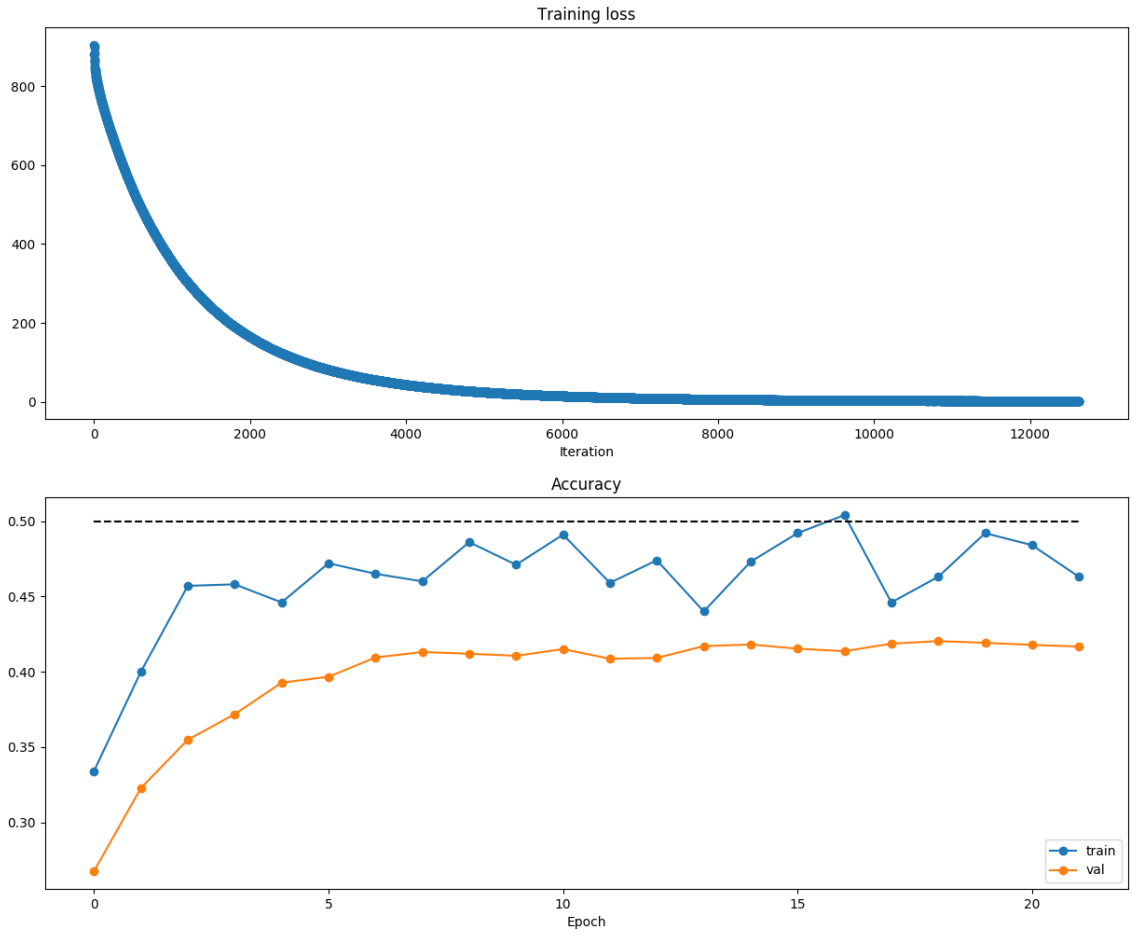

Figure 4: Training loss vs Iteration number and Accuracy vs Epoch of network with final hyper-parameters for the FER2013 dataset

mal. It is interesting that each of these solutions for hyper-parameters found batch size of 50 (minimum bound we specified) to work best. Smaller batch size leads to more stochastic behavior in the gradient descent. This likely helped the model to avoid local minimums. It is also interesting that the algorithm found larger learning rate (1e-3 upper bound) and 0 initial momentum (note: still increasing linearly according to update schedule) to be optimal in most cases. The fact that bayesian optimization set some of the hyper-parameters to the bounds specified indicates that a search with less strict bounds could yield better results. This method could be extended to find the optimum network depth, but this would significantly increase the running time.

The final model for the FER data gave the following evaluation metrics:

angry	disgust	fear	happy	sad	surprise	neutral
74.	13.	51.	33.	47.	17.	27.
0.	0.	0.	0.	0.	0.	0.
51.	13.	100.	40.	59.	34.	36.
149.	13.	113.	643.	174.	53.	153.
70.	8.	69.	63.	189.	23.	85.
33.	0.	60.	34.	35.	235.	39.
90.	9.	103.	82.	149.	53.	267.

Table 1: The confusion matrix

angry	disgust	fear	happy	sad	surprise	neutral
15.85%	23.21%	10.28%	3.69%	7.2 %	4.1 %	4.45%
0. %	0. %	0. %	0. %	0. %	0. %	0. %
10.92%	23.21%	20.16%	4.47%	9.04%	8.19%	5.93%
31.91%	23.21%	22.78%	71.84%	26.65%	12.77%	25.21%
14.99%	14.29%	13.91%	7.04%	28.94%	5.54%	14. %
7.07%	0. %	12.1 %	3.8 %	5.36%	56.63%	6.43%
19.27%	16.07%	20.77%	9.16%	22.82%	12.77%	43.99%

Table 2: The confusion matrix in %

angry	disgust	fear	happy	sad	surprise	neutral
20.3	0.0	24.13	58.64	32.59	55.23	39.26

Table 3: The f_1 measure

Classification rate: 0.42.

6 Question 6.

Altogether we have implemented 6 different convolutional neural networks for recognizing emotions in the fer2013 dataset. We have used Keras (v 2.1.4) with a Tensorflow (v 1.6.0) backend and let all algorithms run on Google Cloud. The setup for Google Cloud was not included in our submitted code, because Google Cloud requires a specific structure for setup. For preprocessing we converted the images to grayscale (only 1 channel), subtracted the mean and divided by the standard deviation of the training dataset for both - training and validation data.
Our best model achieved a train accuracy of 79% (depicted in figure 5) and a validation accuracy of 63% (depicted in figure 7). We used an architecture with 5 2D convolutional layers that were followed by ReLU activation functions, max-pooling and dropout, similar to the architecture from `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/icmi2015_ChaZhang.pdf`. We started with 64 5×5 filters to learn the features of the faces without padding. After convolution the filtered images were padded, max-pooling was used with strides of 2 and the dropout-rate was set to 0.2. This was followed by two 64 3×3 with zero-padding of size (1,1) and two 128 3×3 filters with zero-padding of size (1,1)

all followed by max-pooling and a dropout-rate of 0.2. The filtered activations were flattened and followed by a dense layer of size 1024 with a dropout rate of 0.25 and finally an output dense layer of size 7 (number of labels) with a dropout-rate of 0.25. As the final activation function we used softmax with the accompanying categorical crossentropy.

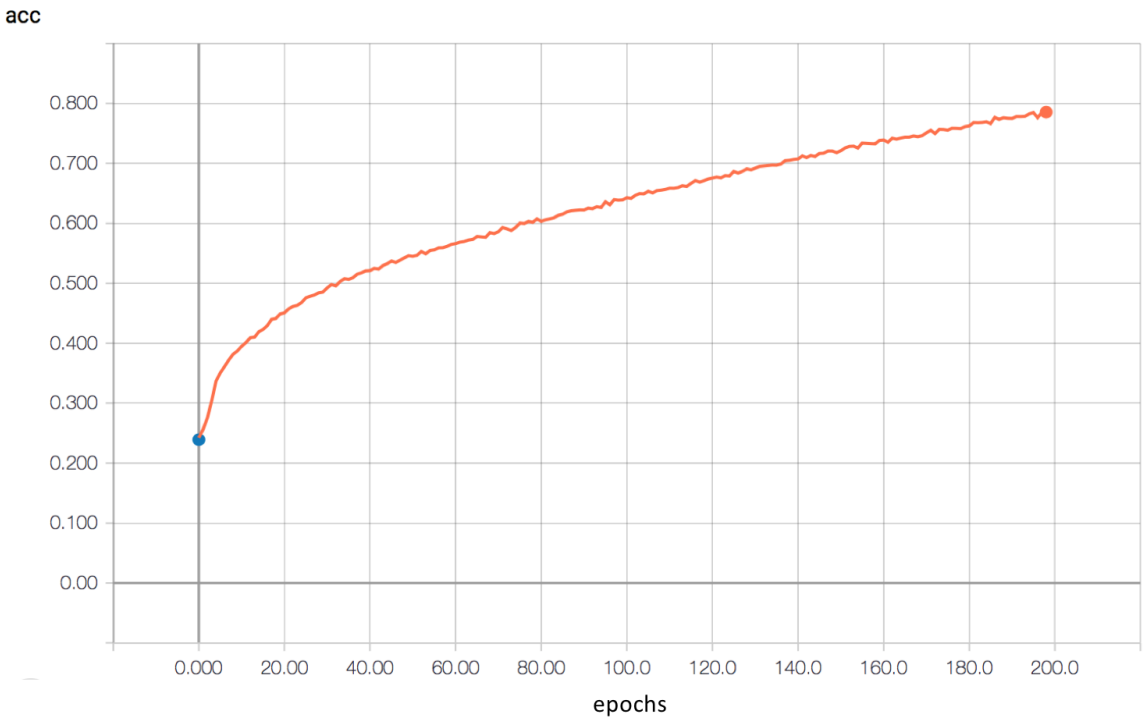

Figure 5: Training accuracy for the FER2013 dataset with CNN

The model had 4,845,063 trainable parameters, was trained for 200 epochs with a batch-size of 128, a validation split of 10% on the training data (train on 25838 samples, validate on 2871 samples) and took 30 hours to train.
Our best model was superior to models without dropout (strong case of overfitting with a training accuracy of 98% and a validation accuracy of 58%), to smaller models (with only 3 hidden layers), to models that used data augmentation methods (horizontal flip and linear transformations - training accuracy of 58% and a validation accuracy of 56%).

angry	disgust	fear	happy	sad	surprise	neutral
246	8.	50.	27.	64.	10.	54.
4.	29.	3.	0.	3.	1.	1.
37.	4.	204.	20.	82.	27.	43.
29,.	2.	24.	721.	42.	20.	51.
62.	8.	102.	41.	327.	16.	82.
15.	0.	38.	18.	15.	321.	6.
74.	5.	75.	68.	120.	20.	370

Table 4: The confusion matrix

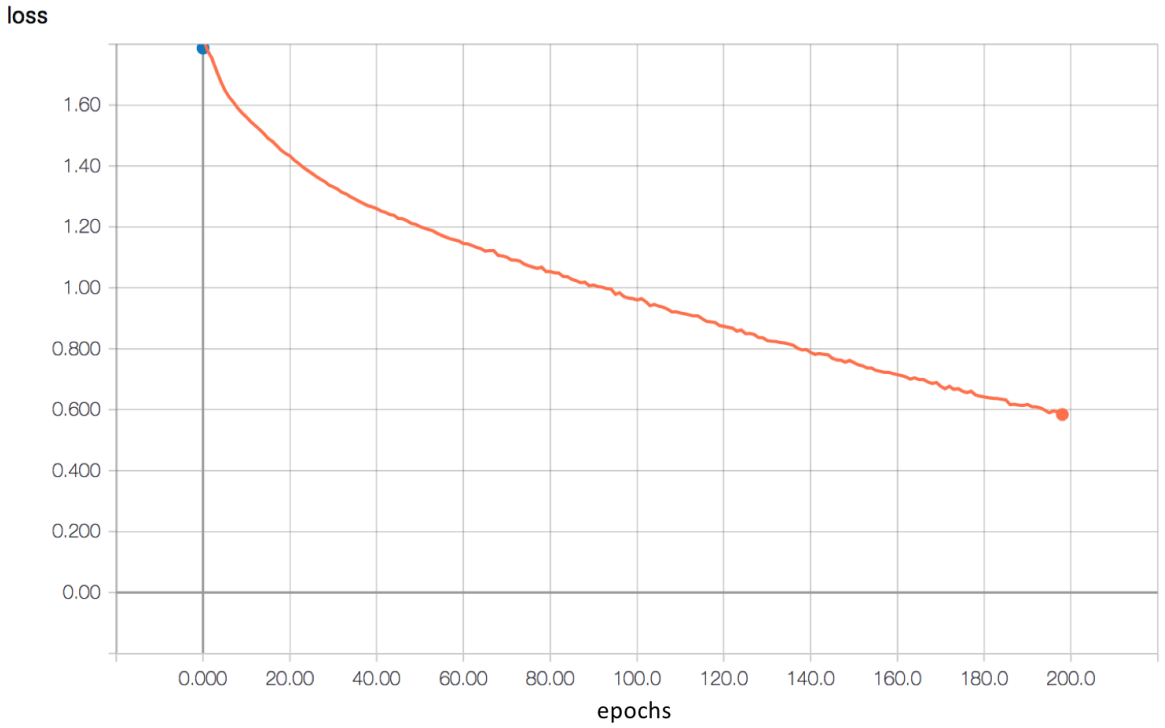

Figure 6: Training loss for the FER2013 dataset with CNN

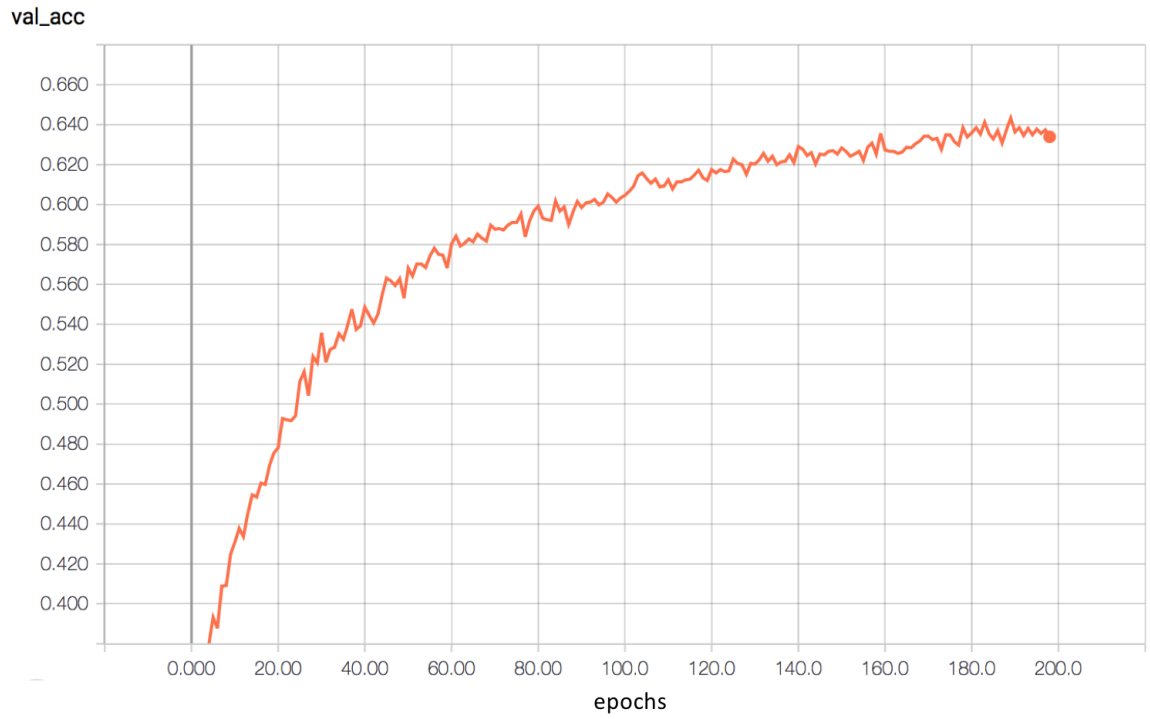

Figure 7: Validation accuracy for the FER2013 dataset with CNN

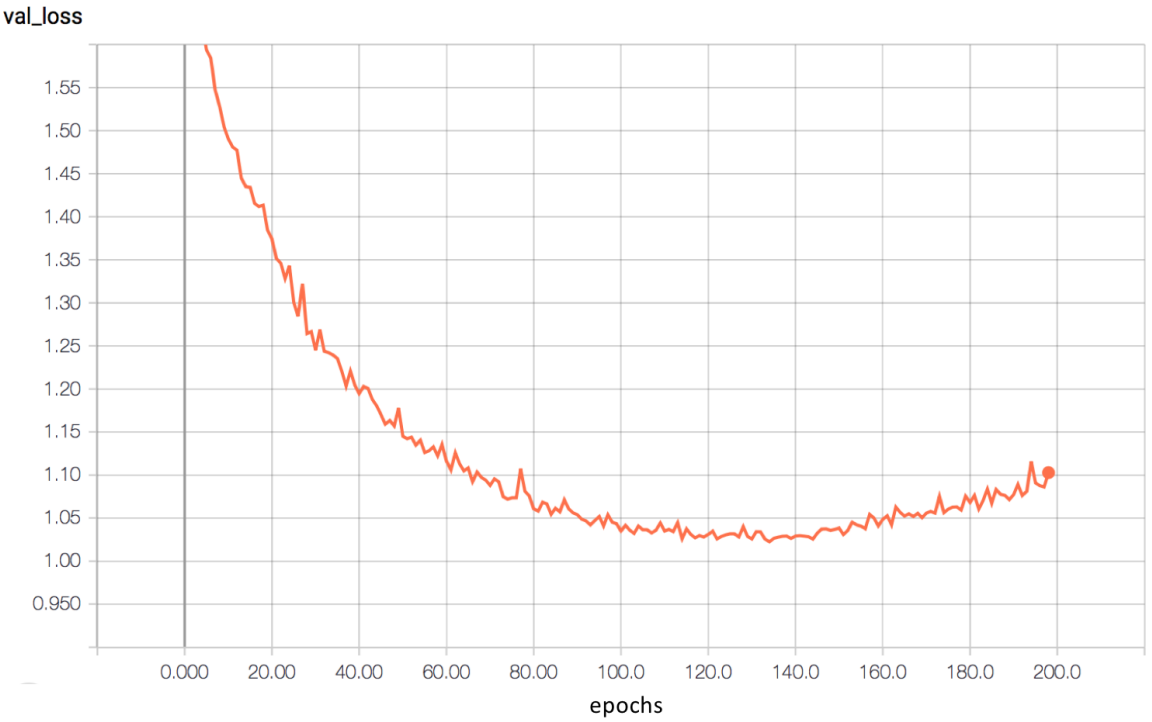

Figure 8: Validation loss for the FER2013 dataset with CNN

angry	disgust	fear	happy	sad	surprise	neutral
52.68 %	14.29 %	10.08 %	3.02 %	9.8 %	2.41 %	8.9%
0.86%	51.79%	0.6 %	0. %	0.46%	0.24 %	0.16%
7.92%	7.14%	41.13 %	2.23 %	12.56 %	6.51 %	7.08%
6.21 %	3.57 %	4.84%	80.56 %	6.43 %	4.82 %	8.4%
13.28 %	14.29%	20.56 %	4.58%	50.08%	3.86 %	13.51%
3.21%	0. %	7.66%	2.01%	2.3 %	77.35 %	0.99%
15.85%	8.93%	15.12 %	7.6 %	18.38%	4.82%	60.96%

Table 5: The confusion matrix in %

angry	disgust	fear	happy	sad	surprise	neutral
53.13	59.79	44.69	80.83	50.66	77.54	55.27

Table 6: The f_1 measure

Classification rate: 0.62.
With over 20% higher accuracy on the validation set, our convolutional neural networks outperform the previous neural network. The structure of convolutional neural networks with activations resulting from filtering on the different layers and their invariance to rotations and shifts due to max-pooling make them very attractive to image classification tasks.
For information on how to run the convolutional neural networks trained, please refer to the readme.md in the submitted code.

7 A1

Assume that you train a neural network classifier using the dataset of the previous coursework. You run cross-validation and you compute the classification error per fold. Lets also assume that you run a statistical test on the two sets of error observations (one from decision trees and one from neural networks) and you find that one algorithms results in better performance than the other. Can we claim that this algorithm is a better learning algorithm than the other in general? Why? Why not?

No, we cannot claim that this algorithm is a better learning algorithm than the other one **in general**. This is due to the fact that we need to consider many different aspects when making a judgement, here we will list just a few.

Firstly, it is very important to consider the time it takes to run and train the algorithm. One algorithm might be giving a better result than the other, however if it takes 10 times longer to run, then we might reconsider using it.

Another important aspect when comparing two algorithms, is their performance on different statistical tests. While one test might show a stronger preference to one of the algorithms, running a few of them can change this judgment completely. For example in lectures, we saw that just looking at classification rate/error in case of a missing class in the imbalanced data set is misleading. Adding one more statistical test can clarify the situation, however might not represent it fully.

Also, it is very important to consider how the hyperparameters were tuned (see Q5 above), as the algorithm may be performing poorly only due to the fact that hyperparameters were not set up properly.

Simplicity is another important criteria, mainly for practical reasons. For example, if one algorithm has too many parameters that need to be tuned, and mostly this has to be done manually, than we might reconsider using such an algorithm.

When judging if one algporithm is **generally** better than the other, it is also important to consider the nature of the data that we will be using, i.e. dependent/independent, discrete/continuous etc. This might largely affect our choice of the algorithm, however does not prove that one algorithm is **overall** better than the other.

8 A2

Suppose that we want to add some new emotions to the existing dataset. What changes should be made in decision trees and neural networks classifiers in order to include new classes?

If new emotions are added to the existing dataset, then the neural network will need an additional output neuron for each new emotion. This will require re-training of the entire network with the full dataset. This may significantly affect network performance, and therefore may require additional changes to the network architecture. (i.e. tuning hyper-parameters, changing the number of hidden layers etc.)

The advantage with decision trees is that adding new emotions only requires creating additional trees for each emotion. The previous trees and their structure remain intact. Therefore, individual tree performance will remain the same, however overall performance may certainly change.