CBC: NEURAL NETWORKS

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Machine Learning

Author:
Dimitrogiannis Kontogouris, Michael Tarasiou (CID: 00650270, 00661239)

Date: March 8, 2018

1 Linear and Relu layers

All functions discussed below work for any size of input variable or its derivatives. In functions that include linear transormations, to get the dimensionality correct we reshape the input variable X to NxD where D is the unrolled dimensionality of a sample (product of all but the first dimension, the first dimension corresponds to number of samples). The weights of the neural network are matrices of dimensions DxB where B is the dimension of the next layer. Bias terms are $b \in \mathbb{R}^B$ and are broadcasted when added to matrices.

1.1 Linear forward pass

The linear forward function performs a linear transformation to the incoming data matrix $X \in \mathbb{R}^{N \times D}$, transforming it into a matrix $y \in \mathbb{R}^{N \times B}$ where B is the number of neurons in the next layer of the network. Mathematically this is shown below:

$$y = XW + b \tag{1}$$

1.2 Linear backward pass

For the backward pass, we are asked to return the derivatives of the loss function with respect to the weights and biases in the layer and the input X. We have:

$$dw = \frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial w} \tag{2}$$

$$\frac{\partial y}{\partial w} = \frac{\partial (XW + b)}{\partial w} = X$$

We know $\frac{\partial L}{\partial y}$ since it is a function input (dout) and thus we can calculate dw:

$$dw = \frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \times X = dout \times X \tag{3}$$

Similarly, the gradient with respect to the biases is:

$$db = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial b} = \frac{\partial L}{\partial y} \times 1 = \frac{\partial L}{\partial y} = dout$$

Finally, the gradient with respect to the input X:

$$dX = \frac{\partial L}{\partial X} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial X} = \frac{\partial L}{\partial y} \times W = dout \times W \tag{4}$$

The dimensions of the gradients are: $dw \in \mathbb{R}^{D \times B}$, $db \in \mathbb{R}^{N \times B}$ and $dX \in \mathbb{R}^{N \times D}$

1.3 Relu forward pass

The relu forward pass is simply defined as:

$$y = max(0, X) \tag{5}$$

The output of the relu forward pass is the input X, if X is greater than 0, otherwise it is 0. The dimensions of matrices y and X are the same $N \times D$ for some N and D.

1.4 Relu backward pass

The relu backward pass is simply 0 if X less than 0, otherwise it is equal to dout.

2 Dropout

Dropout is a method to prevent over-fitting in the neural network by dropping or ”disabling” neurons in the layers of the network during training. Each neuron has a probability p of being dropped. During testing we do not use dropout. This leads to an inconsistency of the node activations during training and test time. To alleviate this we perform inverted dropout and scale the incoming node activations by a factor of $1/(1-p)$ during train time. At test time we can do a forward pass without dropout or scaling. This method ensures minimal interference with the model during test time.

2.1 Dropout forward pass

To perform the dropout forward pass during training, we create a matrix called *mask* which is of the same shape as the input X, Each element of *mask*, is the equivalent of a random draw from a Bernoulli distribution, with p probability of being 0, and $1-p$ of being 1. The output of the layer is an element-wise multiplication of matrix *mask* and the input X, and thus the result is a matrix equal to X, but with some of its elements changed to 0. During test time, no dropout is performed so we just need to return the input variable.

2.2 Dropout backward pass

For the backward pass during the training phase, we perform the same operation as for the forward pass in the opposite direction. We multiply the scaling factor, with the mask (the same as before!) and with the derivatives from the next layers. In the testing phase, the derivatives simply pass through, without a mask.

3 Softmax

The output of the neural network produces a number for each of the classes we have in our classification problem. These numbers can be interpreted as un-normalized probabilities. Using a softmax function, we can normalize them to add up to 1 and thus define a discrete probability distribution over the network outputs. Mathematically the softmax function is defined as:

$$\sigma_j(y_i) = \frac{e^{y_i[j]}}{\sum_{k=1}^{D} e^{y_i[k]}} \tag{6}$$

where, D is the number of classes and $y_i \in \mathbb{R}^D$ is a vector with the outputs of the network for each class (of the i_{th} sample). For numerical stability, we multiply (6) both the numerator and denominator by a constant K s.t $\log K = -D$ which improves stability under overflow and has no effect on results.

3.1 Gradient of softmax

For the combination of softmax and the negative log likelihood cost function we use, for a label $l \in \mathbb{R}^D$ we have the following:

$$L = -\sum_j l_j \log \sigma_j$$

$$\frac{\partial \sigma_j}{\partial y_j} = \frac{\exp y_j \exp \sum_k y_k - (\exp y_j)^2}{(\sum_k \exp y_k)^2} = \sigma_j(1-\sigma_j)$$

$$\frac{\partial \sigma_j}{\partial y_i} = \frac{-\exp y_j \exp y_i}{(\sum_k \exp y_k)^2} = -\sigma_j \sigma_i, \quad \forall i \neq j$$

$$\begin{aligned}\frac{\partial L}{\partial y_i} &= -\frac{\partial l_i \sigma_i + \sum_{j\neq i} l_j \log \sigma_j}{\partial y_i} \\ &= -l_i \frac{1}{\sigma_i}(1-\sigma_i)\sigma_i + \sum_{j\neq i} l_j \frac{1}{\sigma_j}\sigma_j\sigma_i \\ &= \sigma_i \sum_j l_j - l_i = \sigma_i - l_i\end{aligned}$$

4 Question 4

Both experiments used default learning rate $n = 0.001$ and learning rate decay 0.95 applied per epoch.

For the overfitting task we selected the top 50 images and labels from CIFAR10 and used just those to fit a model. Since the objective was to fit the training data as good as possible with no consideration of generalization capabilities of our network we

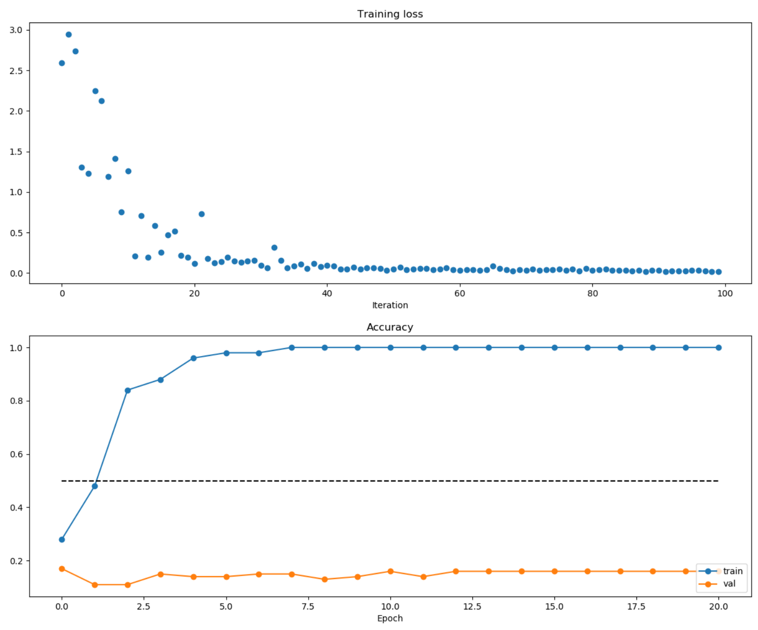

Figure 1: Network overfitting a subset of CIFAR10 dataset

selected a network with two hidden layers with 1024 and 512 nodes respectively which allow for very complicated functions to be fit. We did not apply any type of regularization as this would penalize the complexity of the model. The selected model achieves 100% accuracy over the training set at the sixth epoch as can be shown in Fig 1 below. Momentum was set at 0.5.

To get 50% accuracy over the validation set, we selected a network with two hidden layers, 512 and 256 nodes per respective layer. We didn't use dropout, but we did use a 0.2 L_2 regularization multiplier. The model achieved > 50% over the validation set after 4 epochs as can be seen in Fig 2 below. Momentum was set at 0.3.

5 Hyper-parameter Optimization

For an initial model architecture we have decided to use 2 hidden layers. That choice was motivated by the following. Any single hidden layer feed-forward neural network is proven to be a universal function approximator for a large enough number of hidden units so in theory a single hidden layer network is enough to represent the function we want to find. In practice, a single hidden layer network may require up to an exponential (to the input) number of hidden units to express a function and deep networks can approximate the same function using fewer parameters. Our problem is classifying human emotion which is very likely a complicated function that can benefit from multiple hidden layers. For all things equal simpler is better

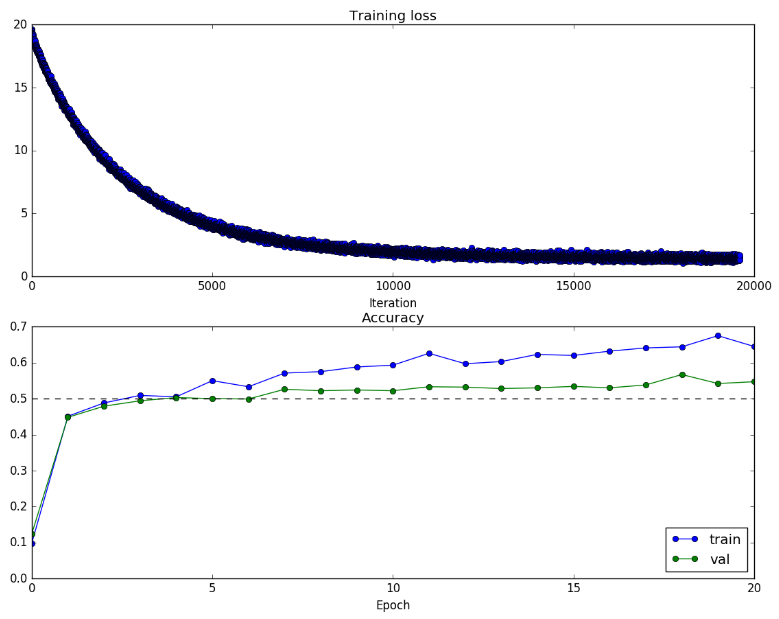

Figure 2: Network achieving > 50% validation accuracy in CIFAR10 dataset

and less computation and complication are easier to manage. For that reason we decided not to use many hidden layers at the beginning. In addition, a two layer neural network was used in the CIFAR dataset achieving classification scores above 50%. The CIFAR dataset has a comparable number of input units to FER2013 so using an NN of similar size seems like a good starting point.

In terms of the activation units for the hidden layers, we are using rectified linear units (ReLu). Networks using ReLu allow more efficient back propagation and have been proven to outperform networks with other activation functions. For that reason we will not be experimenting with different activation functions. For the output layer alone we are using a softmax layer which generalizes sigmoid activations to multiple output units and is ideal for the multi class classification problem at hand.

In general we want to reduce the learning rate as time progresses and we are reaching the local minimum. We use the default learning rate decay update schedule, reducing the learning rate by a given factor after each epoch, which satisfies this requirement and is simple enough.

The **stopping criterion** we choose is early stopping, that is to stop training if the validation set accuracy has not increased over a number of 10 consecutive training epochs.

To optimize the hyper-parameters of the neural network, we used a 2 layer network configuration with **512 and 256 neurons** in the first and second hidden layer respectively. To train the network, we used **stochastic gradient descent with momentum 0.9** and a **batch size of 100**, since it seemed to provide the best compromise between training time and validation performance. Finally, we used **50 epochs**, but in many cases the training stopped earlier due to the stopping criterion.

For the optimization of each parameter, we fix all other parameters, and manually optimize only the variable at hand by repeated cycles of training and assessing the performance of the neural network.

5.1 Learning rate optimization

To optimize the learning rate we trained the network with the configuration as described above using 4 different learning rates and observed the training and validation accuracy. For this optimization, we did not include L2 regularization or dropout. In addition, we did not have any rate decay between epochs.

LR	*Validation Accuracy (%)*	*Train Accuracy (%)*	*Minimum loss achieved*
0.0001	40.28	51.1	1.1770
0.001	42.09	79.1	0.4454
0.01	45.08	98.0	0.0069
0.1	26.90	27.5	1.5881

Table 1: Learning rate optimization

As it is shown from table 1, the best performing model was achieved with a learning rate of 0.01, with a validation accuracy of 45.08%. Figure 3 shows the training and validation accuracy vs the number of epochs, as well as the training loss for learning rate 0.01. Figure 4 shows the best validation accuracy achieved for different learning rates between 10^{-5} and 10^{-1}. Once again the plot shows that the best choice of learning rate is around 10^{-2}.

5.2 Dropout optimization

Using the optimized learning rate from the previous section, 0.01, we trained the network once again fixing all parameters except the probability of dropout. We tested four probabilities of dropout 0.01, 0.02, 0.05 and 0.1.

Dropout	*Validation Accuracy (%)*	*Train Accuracy (%)*	*Minimum loss achieved*
0.01	43.83	94.3	0.021
0.02	43.97	93.9	0.055
0.05	42.19	81.6	0.296
0.1	41.64	81.4	0.286

Table 2: Results for different dropout probabilities

We observe from table 2 that a dropout rate of 1-2% seem to perform the best on the validation data, so dropout overall does not seem to help our network. This can be taken as an indication that the network we are training does not have the capacity to benefit from dropout. Figure 5 shows the training and validation accuracy as well as the training loss for dropout rate of 0.01.

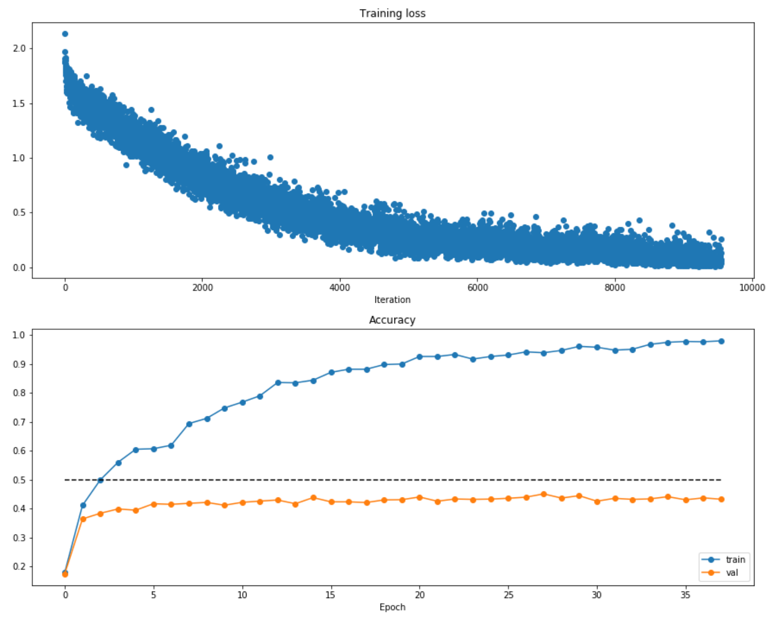

Figure 3: Learning rate: 0.01. **Top**: Training loss vs number of iterations, **Bottom**: Training and Validation accuracy vs epochs

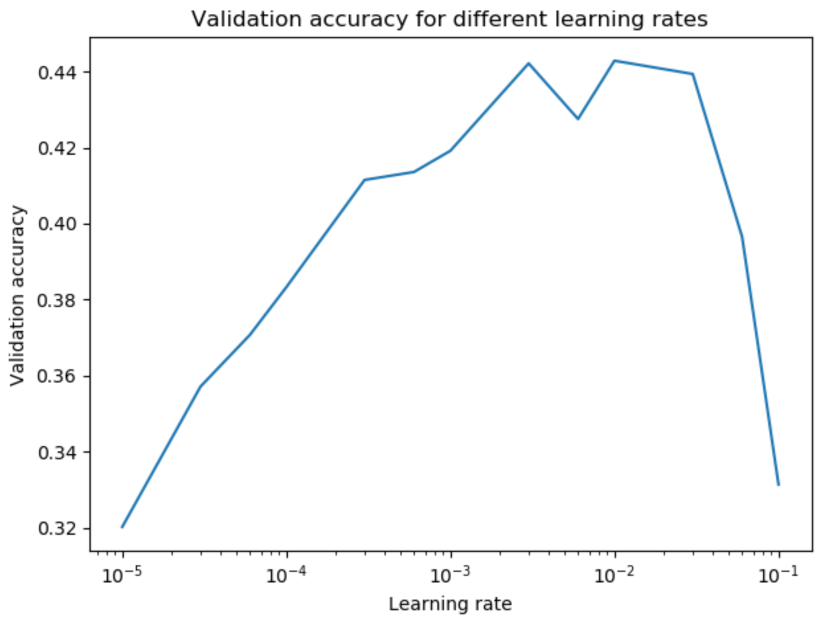

Figure 4: Validation accuracy for different learning rates

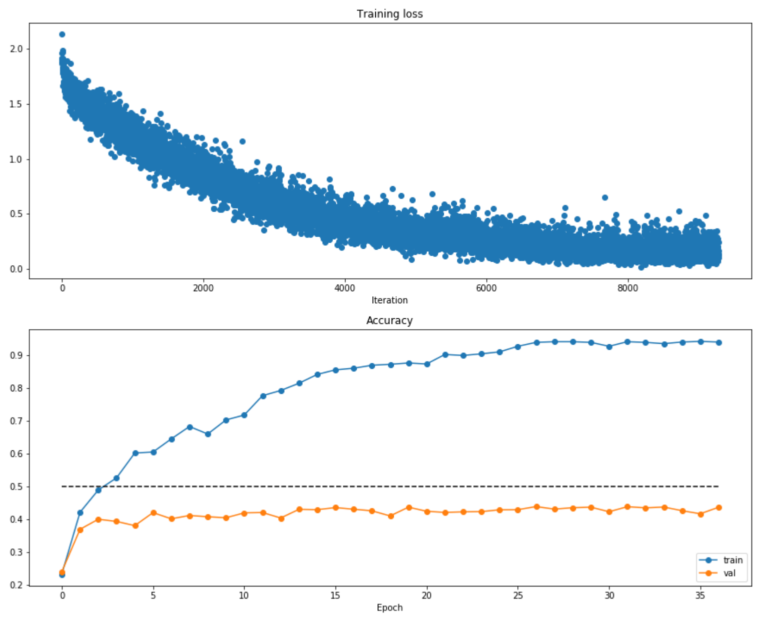

Figure 5: Dropout: 0.01. **Top**: Training loss vs number of iterations, **Bottom**: Training and Validation accuracy vs epochs

5.3 L2 regularization vs dropout

To test L2 regularization, we set dropout to 0 and fix all other parameters once again. The results are shown in table 3.

Regularization	*Validation Accuracy (%)*	*Train Accuracy (%)*	*Minimum loss achieved*
0.001	44.28	97.0	0.1275
0.01	45.43	97.0	0.4844
0.1	43.87	61.9	1.1971
1	35.29	35.8	1.5851

Table 3: Results for different regularization parameters

L2 regularization and dropout serve the same purpose which is to improve the generalisation capacity of the fitted model. It is observable from tables 5 and 3 that as we increase the dropout parameter or the regularization parameter, training accuracy drops, since the model is not ”free” to use the parameters that best fit the training data. We observe the best validation performance with regularization set at 0.01 (45.45%) while for dropout of 0.02 the best validation performance is 43.97%. Overall, again we notice that our model is in need of little regularization.

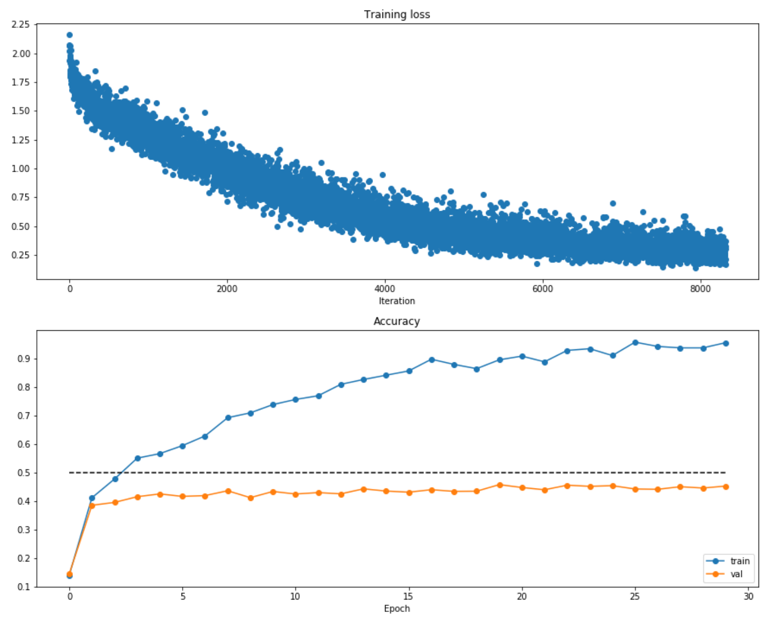

Figure 6: Regularization: 0.001. **Top**: Training loss vs number of iterations, **Bottom**: Training and Validation accuracy vs epochs

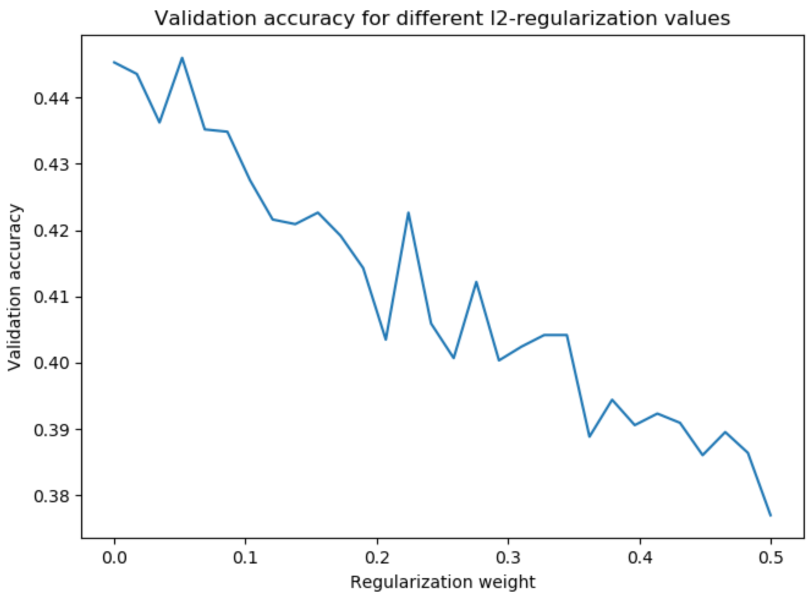

Figure 7: Validation accuracy for different regularization rates

5.4 Number and size of hidden layers

Using the optimized parameters, we performed training using different network configurations of 2 or 3 hidden layers, with different combinations of numbers of nodes per hidden layer. Our general approach was to reduce the number of nodes for deeper layers of the network since our output layer is of only size 7 (7 emotions), while our input layer is 2304 (all the pixels in an image). In addition, we introduce learning rate decay of 95% (learning rate is reduced by 5% in each epoch). The results of the optimization are shown below.

Architecture	*Validation Accuracy (%)*	*Train Accuracy (%)*	*Minimum loss achieved*
[512, 256]	46.93	99.9	0.458
[256,128]	42.47	98.9	0.3287
[512,512,256]	44.49	97.7	0.5950
[512,256,128]	45.54	99.6	0.4841
[256,128,64]	41.67	0.7	0.8411

Table 4: Hidden layers size

From table 4, we observe that the best validation performance is achieved for the 2 hidden layer NN with 512 and 256 in each of the layers. The validation accuracy is 46.94%.

5.5 Performance of NN on test data

The NN parameters that are used for this part are the following:

- 2 hidden layers, 512 and 256 neurons respectively
- learning rate 0.01 with 0.95 decay
- dropout probability 0.01
- regularization 0.01
- momentum 0.9
- batch size 100
- epochs 50

Below we analyse the model performance on the test data of FER2013. To get the performance metrics per emotion, we tranformed the predicted classes and the labels to binary values (1 for the emaotion at hand, 0 for all other emotions). The accuracy per emotion is very high, but this does not indicate that the model performs well, since the amount of positive examples in each binary classification (per emotion) is a lot less than the negative examples (all other emotions). Thus, the accuracy is artificially high by predicting the negative examples correctly. The precision, recall and f1 score are better indicators of the performance of the model for each individual emotion. Emotions 5, 3 and 1 exhibit the best performance. The classification rate for all emotions is 46.17%. The same picture is presented in the confusion matrix shown in Table6, which shows the actual class (column) given a predicted class (row).

Emotion	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	***F1 (%)***
0	82.25	33.0	35.33	34.12
1	98.55	54.76	41.07	46.94
2	82.17	34.87	33.47	34.16
3	80.33	59.89	63.91	61.84
4	78.54	39.38	33.23	36.05
5	91.53	63.70	62.17	62.93
6	78.96	38.78	42.17	40.41

Table 5: Performance of network for each emotion

	P0	***P1***	***P2***	***P3***	***P4***	***P5***	***P6***
A0	165	1	61	86	81	17	56
A1	8	23	4	6	8	1	6
A2	70	5	166	59	71	41	84
A3	70	5	46	572	69	27	106
A4	86	5	90	106	217	30	119
A5	32	0	43	30	19	258	33
A6	69	3	66	96	86	31	256

Table 6: Confusion matrix of the test data

6 Question 6

For this task we used Tensorflow. The CNN model we submit consists of 4 blocks of convolutional layers (3x3 kernels) followed by max pooling, which are then followed by 2 fully connected layers. We used small 3x3 filters inspired by the VGG architecture that has performed very well in recent years. The stride and padding are selected so that the convolution layer does not modify the size of the features. After each pooling layer we reduce the width and height of the features by a half and double the number of features, effectively halving the total size of the features. We selected a deep convolutional architecture that allows for relatively small size of fully connected layers at the end. The architecture is described in greater detail below:

- input layer of dimensions Nx48x48x1 (batch size x image width x image height x number of channels)

- 2 convolutional layers with 32x3x3x1 filters (number of filters x kernel width x kernel height x number of channels) at stride 1 and 1 zero padding at all dimensions, followed by ReLu activation. Max pool layer of size 2x2 (width x height) at stride 2 that performs dimensionality reduction. After this step each feature's dimensions are halved leading to a tensor of dimensions Nx24x24x32 (batch size x image width x image height x number of features)

- 3 convolutional layers with 64x3x3x1 filters at stride 1 and 1 zero padding at all dimensions, followed by ReLu activation. Max pool layer of size 2x2 (width x height) at stride 2. After this step each feature's dimensions are Nx12x12x64

- 3 convolutional layers with 128x3x3x1 filters at stride 1 and 1 zero padding at all dimensions, followed by ReLu activation. Max pool layer of size 2x2 (width x height) at stride 2. After this step each feature's dimensions are Nx6x6x128

- 3 convolutional layers with 256x3x3x1 filters at stride 1 and 1 zero padding at all dimensions, followed by ReLu activation. Max pool layer of size 2x2 (width x height) at stride 2. After this step each feature's dimensions are Nx3x3x256. This layer is reshaped to a vector of dimension 2034 and dropout with probability of keeping a node 0.6 (dropout 0.4) is applied

- fully connected layer with 512 nodes and dropout 0.4

- fully connected layer with 128 nodes and dropout 0.4

- output layer of dimension 7 equal to the number of classes

Taking advantage of the Tensorflow implementations, we used the Adam optimizer with a learning rate of 0.005 and default parameters $beta1 = 0.9$ $beta2 = 0.999$. Adam was presented in [4] and is a first-order gradient-based optimization that includes adaptive momentum and learning rates.

We experimented with using residual networks (ResNet) presented in [1], [2], specifically the 101-layer bottleneck network. Residual networks are one of the most significant advances in terms of convolutional neural network architecture over the last decade. They use what the authors call skip connections between layers that pass identity mappings between the layers they connect. As a result the convolutional layers between skip connections have to model the residual function which is performed more effectively by the network. This architecture option results in more efficient backpropagation and the capacity to train far deeper models that was though achievable before. Our residual networks performed better however due to size limitations we were not able to submit a trained model.

The training and validation accuracy with respect to the number of epochs can be seen in Fig.8 below. Using the Test data to assess performance we get the confusion matrix shown in Table 7 and the precision, recall and F1 scores as shown in Table 7. Comparing the performancewe can see that the CNN clearly outperforms our best FCN. However that cannot be said regarding all emotions and all performance metrics. For example in the case of emotion A1 (Disgust) the FCN has a higher recall and seems to work equally well when looking at the confusion matrix. Overall, the FER2013 dataset can be difficult to interpret (the human accuracy is about 65% according to [5]) mainly because emotions are rarely present in isolation (human expressions show a mix of emotional states). As such emotion recognition is a complicated task and models are expected to meet a lot of difficulties in fitting it.

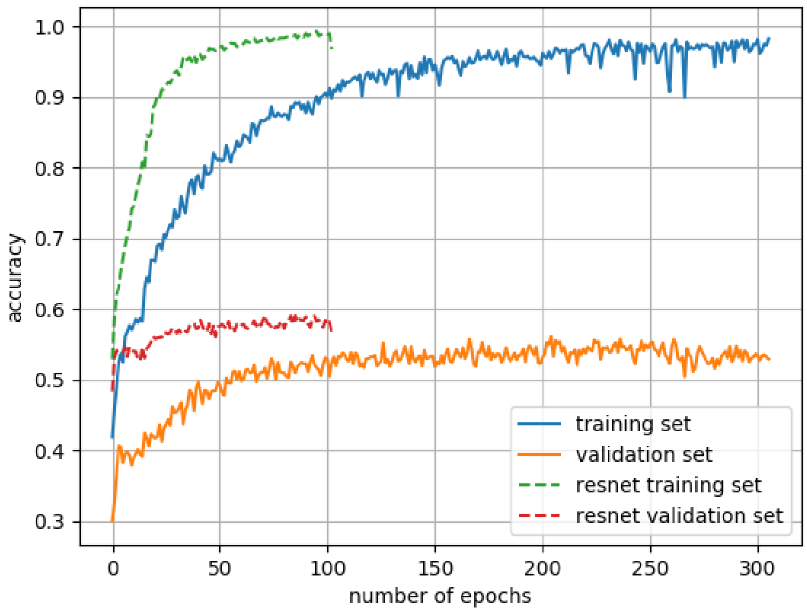

Figure 8: Training and Validation accuracy vs number of epochs

	P0	***P1***	***P2***	***P3***	***P4***	***P5***	***P6***
A0	230	0	54	31	69	14	68
A1	19	20	5	3	6	0	3
A2	72	1	159	42	95	33	94
A3	34	0	21	720	32	21	67
A4	105	0	59	64	271	9	145
A5	24	0	33	24	19	291	24
A6	67	0	45	72	110	5	308

Table 7: Confusion matrix of the test data on the trained CNN

Emotion	Accuracy (%)	Precision (%)	Recall (%)	*F1 (%)*
0	84.48	0.42	0.49	0.45
1	98.97	0.95	0.36	0.52
2	84.56	0.42	0.32	0.36
3	88.55	0.75	0.8	0.78
4	80.13	0.45	0.42	0.43
5	94.26	0.78	0.7	0.74
6	80.49	0.43	0.51	0.47

Table 8: Performance of CNN per emotion

7 Additional questions

7.1 A1

We could not claim one to be a better algorithm than the other in general for the following reasons stemming from the fact that the two methods are very different qualitatively:

- neural networks can be seen as black box function approximators while trees are highly interpretable. In cases where interpretability is crucial we could not use neural networks
- fitting and doing inference with the two models scales differently depending on the software and hardware architecture used. The computing architecture of today might allow one model to be run more efficiently and outperform the other but that will not necessarily hold for future architectures

Finally, the no free lunch theorem for machine learning of [3] states that no apriori distinctions can be made between any two learning algorithms. That is to say that while the inductive principle used by one model might allow it to overperform another at a given problem, when examined under the scope of all possible problems, any two learning algorithms are equivalent.

7.2 A2

For the classification trees, adding more classes would require retraining the whole model as the input space for the new classes has to be taken from space previously occupied by other classes. Other than that, the tree algorithm can be applied without changes.

For the case of neural networks a step we would definitely need to take is to add another node at the output layer of the network allowing the classification of $n + 1$ classes where the previous network was capable of classifying n classes. We will need to initialize the weights and bias connecting the new node with the last hidden layer and we will need to train at least the last layer of the network. Most possibly we will need to retrain the other layers but the previously trained network will provide a good initialization, so it is likely to converge faster than by starting from scratch.

8 Instructions for executing

All code was written in Python3. All scripts were tested and run in the DOC Lab environment. To run the submitted scripts open a terminal window in the project directory and follow the instructions below.

To run the scripts *test.py* and *test_deep.py* which are used to predict using the best FCN and CNN respectively, use the *src* as the home directory and complete the user input section at each script. The parameters to complete are *img_folder* (the path to the test data) and *srcdir*, the global path to the *src* directory. Both scripts load the models which are located in subdirectories of the *src* directory and use the model to make predictions. The predictions are assigned to a variable *preds* which the user can save to disk or pas it further down the pipeline.The best CNN is named *conv*03 and is saved in subdirectory *src/conv*03*_logs*. The best FCN is named *pickle.net* and is saved in subdirectory *src/pickles*.

For the CNN to run tensorflow must be installed.

9 References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Deep Residual Learning for Image Recognition. arXiv:1512.03385

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Identity Mappings in Deep Residual Networks. arXiv: 1603.05027

[3] Wolpert, David (1996), "The Lack of A Priori Distinctions between Learning Algorithms", Neural Computation, pp. 1341-1390

[4] Diederik P. Kingma, Jimmy Ba, " Adam: A Method for Stochastic Optimization", arXiv:1412.6980

[5] Goodfeloow et al. "Challenges in Representation Learning: A report on three machine learning contests", arXiv:1307.0414