dear Team ##,

I found your work almost a model case of a very well done semester project. You did a most insightful study of a modeling task that just had the right (limited) difficulty and an affordable-sized dataset, with existing comparison metrics out there, and - super! - implemented your models and trainings from scratch. The only options for improvement concern the technical writing: some relevant information is not given, see our comments. Also I was left with some mysteries concerning the loss function that you used (or invented?). But altogether very good!

For displaying our comments I recommend using Adobe Acrobat - other pdf readers may not display all comments properly. - You find the grading breakdown on the next page.

Kind regards, Herbert Jaeger (July 17, 2022)

Evaluation form for semester project report, Neural Networks (AI), Spring 2022

Name of students: nn
Name of grader: Celestine P. Lawrence, Herbert Jaeger
Date: 11.07.2022 / 17.7. 2022

Fill in violet fields with grades on scale 0-10. If not applicable, put weight to zero. For bonus, fill in green field (as grade addition); this bonus will be added to the report grade ("Report Grade", capped at 10.0)
For Latex use, enter a 1 in the red field when Latex was used, else 0.

	grade	weight %	total	criteria
Presentation (40%)				
Figures, tables, pseudocode	9.50	5	0.475	good quality figs? Captions? Symbols explained in caption? Figs etc. referenced in text? Sources of imported graphics given? Legible in B/W printout? Figs informative (not redundant)? Pseudocode transparent?
Technical writing	8.00	35	2.8	clear formulations? Concise formulation? Mastership of technical formulations / math formulations? Good text flow? Appropriate language (not sloppy in technical sections, not too dry in motivation sections?) Clear flow of argumentation? Contents well structured? Correct English, typos? All used external sources (literature, code) referenced? Reference list uniformly and correctly formatted? are experiments/implementations clearly, completely, succintly described? Can experiments be checked/reproduced? results clearly, completely, succintly documented? Technical descriptions done in clear English and/or formulas and/or pseudocoe, without relying on toolbox terminology?
Latex	1.00		0	was Latex used? If yes leave the 1.00, else write 0 in the red field
Technical quality (60%)				
Penetration of subject	9.00	20	1.8	Challenges of data / task well recognized? Used data transparently described and inherent challenges discerned? Clear perception of goals for the prooject? Good task statement? Challenges inherent in dataset perceived and adequately addressed? Appropriate choice of model type? choice of preprocessing, feature extraction, learning algorithm motivated? Motivational examples?
Technical work	9.00	30	2.7	Results connected back to starting question? Statistical basics done properly where applicable (error bars!)? Maths correct? Reasonable choice of model type? Appropriate pre-processing of raw data? Implementations / experiments reasonably thought out? Cross-validation / early stopping appropriately used? Planned professionally (modular, efficient, transparent, documented)? Realistic assessment of qualities / deficiencies of results?
Exhaustiveness	10.00	10	1	Is the amount of work done adequate for an extended "lab exercise"? (if ok, full score here - for extra effort give some bonus below)
Bonus	0.50	100	0.5	extraordinary achievements, e.g. much extra work, very independent work, very difficult topic, interdisciplinary connections, original thinking. Max 1.5 bonus grade points possible; typically fractions of 1.0
Report Grade			***9.28***	(bonus for implementation from scratch)

university of groningen / faculty of science and engineering / bernoulli institute

Breast Cancer Prediction

Semester Project for the Neural Networks course

student 1	student 2	student 3	student 4
s1234567	s1234567	s1234567	s1234567

Abstract: Breast cancer prediction from cell samples is an important medical task that mathematical models or neural networks can nowadays solve. We used the Breast Cancer Wisconsin Data Set from the UCI ML Repository to train a neural architecture to classify cancer cells as malignant (M) or benignant (B). We then compared the accuracy of these this network with a baseline logistic regression model. Furthermore, we experimented with transfer learning.

1 Introduction

For our semester project we chose the task of breast cancer cell classification. The dataset we work with is very small (by today's standards) which allows us to further experiment with different architectures and parameters and see how they influence the accuracy. To successfully complete the semester project we followed the standard procedure for machine learning tasks:

1. Find an appropriate dataset
2. Explore the dataset using visualizations and basic statistics
3. Clean and normalize the dataset to ensure high quality
4. Pre-process the data (split and reduce dimensions using PCA)
5. Decide on and build a neural network architecture
6. Split the dataset into training and testing set and train the network
7. Use k-fold cross validation to prevent overfitting (as well as regularization elements)
8. Play with the parameters of neural network to obtain the highest accuracy (eg. number of neurons/layers, learning rate)
9. Experiment with different architectures and baseline models
10. Report our findings

1.1 Implementation decisions

We have decided, despite the primordial fear, to go through the project *without* the use of external libraries such as Keras, Tensorflow, or PyTorch. Starting from the lecture notes and Goodfellow's book on deep learning, we set up the network architecture, wrote the forward pass function and the backpropagation algorithm from scratch. The only help being numpy for useful matrix operations and some other libraries for side-tasks such as parallelization, PCA, or plotting. The code can be found **here**.

2 Background and Data

We used the Breast Cancer Wisconsin Data Set for our task, which can be freely accessed in the UCI Machine Learning Repository (5). The data is comprised of fine-needle aspirates data from 569 patients. The detailed construction of the dataset is presented in Wolberg *et al.*(6), we present a summary.

2.1 Background

A fine needle is inserted in a potentially suspicious mass located in the breast area. The cells present in the mass are then aspirated and pot under a microscope, hence the name fine needle aspirate. The benefit of this procedure is to offer a less invasive method of checking potentially suspicious clumps since it involves using only a small needle instead of more invasive histological procedures. The task is then to classify the extracted sample. In particular, we are interested in whether it is

cancerous or not.

Wolberg *et al.* write that the human performance on the classification task is over 90%(6). Furthermore, if the medical professional performing the fine needle extraction is unsure of the results, they can prescribe a more invasive procedure in order to have better accuracy.

2.2 Data

One fine needle aspirate was extracted for each of the 569 patients present in the data set. Fine needle aspirate is then spread on a microscope slide. This makes it so that many cells cover each other, break or are squished by their neighbours. Furthermore, the fine needle aspirates cells indiscriminately. Therefore even if the mass is cancerous, the sample may contain 'healthy' cells. Therefore, it is essential to consider the most abnormal-looking cells, which are more likely to be cancerous. A human user selected three cells based on these facts and drew a rough outline. The outline is further improved using an active contour model known as 'snake'. The final boundary is then analyzed for the creation of the following data points:

- *Radius.* The average length from the centre to the boundary of the contour.
- *Perimeter.* The total length of the contour.
- *Area.* Naively calculated by counting the pixels that the contour encompasses.
- *Compactness.* Calculated as $\frac{Perimeter^2}{Area}$. It measures how 'close' the shape is to a circle, the most compact shape with the compactness of 0.5.
- *Smoothness.* A measure of the smoothness of the contour.
- *Concavity.* If we take two points on the contour and we draw a line through it such that it never intersects with the contour, we call the shape concave. In this case, Wolberg *et al.* check weather concavities exit between points that are close and measure their severity(6).
- *Concave points.* Similar as above.
- *Symmetry.* A measure of how symmetric the contour is based on taking the longest possible line through the area enclosed in the contour as the axis of symmetry.
- *Fractal dimension.* A measure of how regular the contour is.
- *Texture.* The variance of the colour of the pixels contained in the contour.

If we assume that all cancerous and non-cells are of the same area, then given the characteristics mentioned above, we note that a perfectly circular cell would score the lowest in all the categories. This is very promising since Wolberg *et al.* mention that it is the case that cancerous cells are identified by how more abnormal they look in general(6).

The final data set, which we use, is constructed from the raw data of the fine needle aspirates. In particular, for each patient and each feature, the mean and standard deviation are given. Furthermore, an extra data point is given, representing the worst measured value. As we mentioned above, the more a cell feature is evaluated higher, the more abnormal-looking the cell is. This is also true for the area (6). Therefore, the worst value is simply the highest one of the one measured.

The data available for each patient is composed of a 30-dimensional feature vector, which contains three values for each of the features measured on the cells, the average, the standard deviation and the highest value.

3 Data reduction

We perform feature selection, with the aim of training the model on less feature that are more important. This should reduce training time as well as performance, since the architecture is being fed only the most important features. Before setting up the network architecture, we wanted to further analyse/visualize our data and decide on a data reduction method. In this section we will discuss three preprocessing / data exploration steps: correlation matrices, logistic regression, and dimensionality reduction with visualization.

3.1 Correlation matrix

We start by creating a correlation matrix of each feature set to see if and how our features are correlated. This is important because two highly correlated features might be redundant and hence one of these variables can be omitted. However, this is not always the case since the two correlated variables might still include different key information important for the training.

From Figure 3.1 we see that very highly correlated features are f1, f3 and f4 although there are some other high correlations such as f3 with f8. This indicates that there might be room for dimensionality reduction without significant decrease in accuracy. We will be testing this in section 3.3.

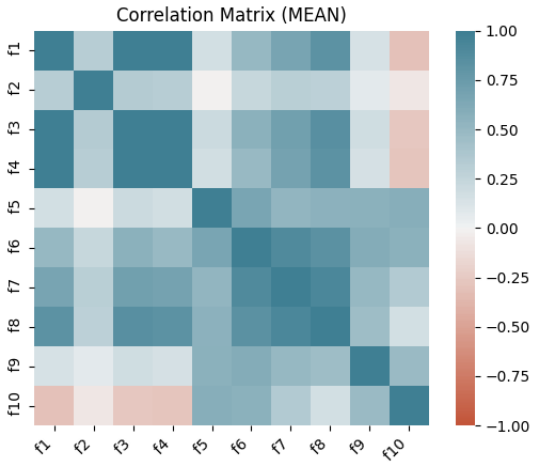

Figure 3.1: Correlation matrix of 10 features representing the means of the features.

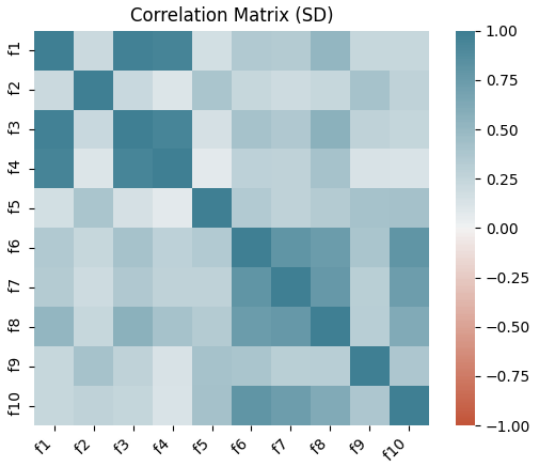

Figure 3.2: Correlation matrix of 10 features representing the standard deviation of the features.

3.2 Logistic regression

Based on the paper (6) claiming to have obtained accuracy score of 0.975 using a regression model, we wanted to reproduce this result and compare it to the performance of our neural network model.

3.2.1 Implementation

We first obtain a subset of our dataset corresponding to the means of features (first 10 features). Then we split it in the training (500) and testing (68) set. Then, for our logistic regression model we have used the model from *sklearn.linear_model* as $LogisticRegression(solver = `lbfgs`, max\_iter = 1000)$. Lastly, we let the model predict the correct labels and computed the accuracy score.

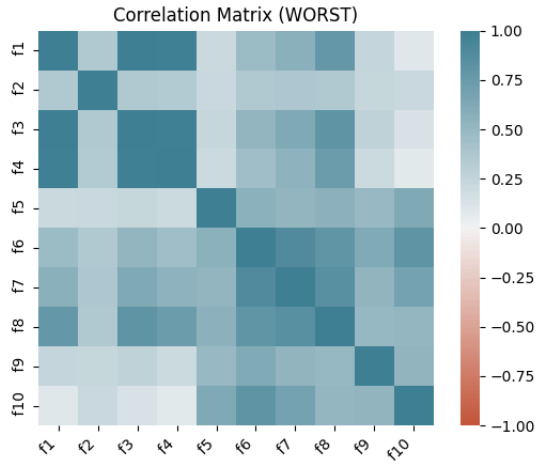

Figure 3.3: Correlation matrix of 10 features representing the worst (highest) values of the features.

Algorithm 3.1 Calculate regression accuracy

Require: x_train, y_train, x_test, y_test
  $scores = []$
  $reg \Leftarrow LogisticRegression()$
  $reg.fit(x\_train, y\_train)$
  for *every training instance* **do**
    $pred \Leftarrow reg.predict(training\ instance)$
    if $pred = y\_train\ instance$ **then**
      $scores.append(1)$
    else
      $scores.append(0)$
    end if
  end for
  return $\frac{sum(scores)}{len(scores)}$

3.2.2 Regression results

After averaging the results over 1000 runs we obtained an accuracy score of 0.92658. Most of the data points were classified correctly, however, there is room for improvement. This result indicates that our data might be, to some extent, linearly separable but pure logistic regression model of 10 features is not sufficient to predict malignant or benign cancers with a *very* high accuracy.

3.3 PCA

Principal Component Analysis (3) is a standard and popular dimensionality reduction algorithm widely used in many machine learning applications, including ours. We have first used PCA to help visualize our data in 2D and 3D and check the alleged linear separability from other papers.

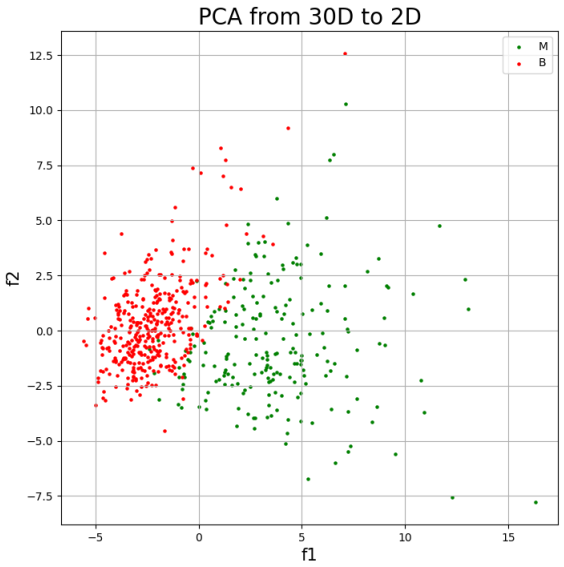

Figure 3.4: PCA reduction of the original dataset from 30 dimensions to 2.

From Figure 3.4 the linear separability is becoming evident although there is some small overlap between the two classes. Reducing the data from 30 dimensions to merely 2 leads to information loss. Hence, we have also plotted 3D graph of 3 features.

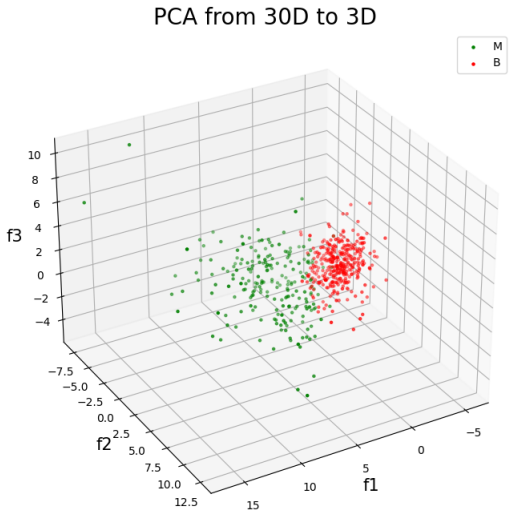

Figure 3.5: PCA reduction of the original dataset from 30 dimensions to 3.

From Figure 3.5 we can see that the two groups can be linearly separated.

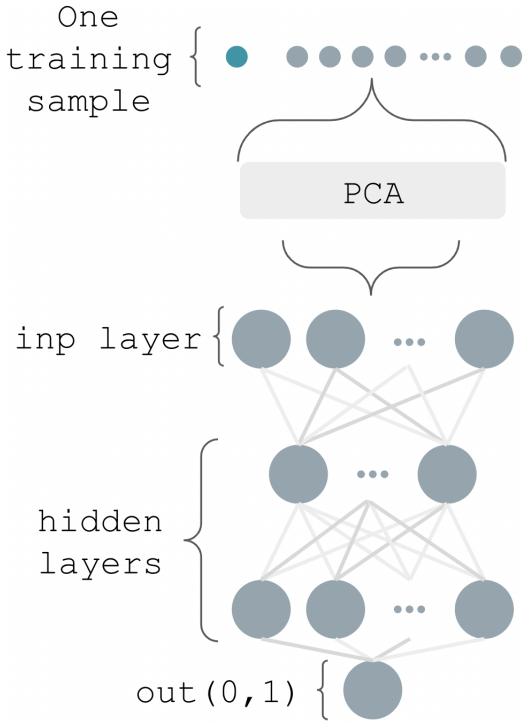

Figure 4.1: A graphical representation of the architecture of the MLP we used. The ~~size~~ of the training ~~sample~~ is 30.

4 Classifier

This section describes the classifier we used to complete the task. The dataset classifies the cell samples as binary in benign or malign cancer. Cancer is a severe illness; therefore, it would be better for an application to output its confidence and the diagnosis since, in case of an inconclusive fine needle aspirate, an invasive test can be carried out. We also describe how dropout was implemented as a regularization procedure.

4.1 The Architecture

Since this is a course on Neural Networks, we choose a model from the class of multi-layer Perceptrons with a single output neuron. We varied the number and size of hidden layers throughout the experiment, and the input layer, whose size changed based on the input data size. For the model to be more accurate, we added a trainable bias unit at each layer. The activation function we used is the logistic sigmoid: $\sigma(a) = \frac{1}{1+e^{-a}}$, which is a standard choice(2). We also note that it is particularly appropriate in this case since the final output will already be in the $(0, 1)$ interval. Therefore, we can interpret it as a probability that the sample is malignant.

4.2 The Loss Function

The last remark of the previous part is contingent on choosing an appropriate loss function. Following Goodfellow, we choose the loss function in the following way(1). First, we consider a model θ. Therefore for input x, we want $\theta(x)$ to represent the probability that x is malign, which in the dataset is denoted as $y = 1$, reserving the value 0 for the benign diagnosis. Therefore $\theta(x) = P(y = 1|x)$. Since y can take two values it follows that $P(y = 0|x) = 1 - \theta(x)$. We also let z represent the potential of the last neuron, therefore $\theta(x) = \sigma(z)$. This is a convention from Goodfellow since it makes writing the loss easier and computationally safer as it avoids under-flows in computations(1). Then we pick the loss function:

$$L(\theta(x), y) = \log(1 + e^{(1-2y)z}).$$

We will use the backbpropagation algorithm as presented in the lecture notes for the course(2). In order to do so, we need to compute $\frac{\partial L}{\partial \theta(x)}$. We can use the fact that $\frac{\partial y}{\partial \theta(x)} = 0$, in order to write that:

$$\frac{\partial L}{\partial \theta(x)} = \left(\frac{\partial L}{\partial z}\right)\left(\frac{\partial \theta(x)}{\partial z}\right)^{-1}.$$

And hence:

$$\frac{\partial L}{\partial \theta(x)} = \frac{(1-2y)e^{(1-2y)z}}{(1-e^{(1-2y)z})} \cdot \frac{1}{\theta(x)(1-\theta(x))}.$$

With some arithmetic we can simplify the above expression to:

$$\frac{\partial L}{\partial \theta(x)} = (1 - y - \sigma(-z)) \cdot \frac{1}{\theta(x)(1-\theta(x))}.$$

We note that for $y = 1$ then as $z \to \infty$, as it should in learning, we have that $\frac{1}{e^z - 1} \to 0$ and hence the loss disappears.

Training, Hyper-parameters and Initialization

We train our model using the back-propagation algorithm as presented in the lecture notes(2). In this case, we also want to mention briefly the Hyper-parameters involved in the task. The learning rate, the step size of the gradient descent will be varied through our experiment. The stopping criterion will be given by cross validation as explained in the following Section 5. Furthermore, we initialize the model's weights to small random variables as suggested in the lecture notes(2).

4.2.1 Regularization: Dropout

Regularization is a standard techniques used in ML to prevent overfitting and improve the ability of the model to generalize, as presented in (1).

There are many regularization methods, but amongst them, we selected dropout as it is a powerful yet computationally inexpensive method as mentioned in Goodfellow's book(1). Moreover, using dropout also provides a benefit in efficiently improving a neural network's robustness. A neural network is considered robust if its performance remains stable despite the fluctuations in the testing data. This robustness improvement occurs in the dropout case because different neurons are removed or become inactive every epoch. Hence it is more difficult for the neural network to memorize the training samples. Using dropout further allows the train of multiple neural networks without the typically high cost of memory and runtime, since at each testing run, the neural network is *de facto* smaller as some neurons are inactive.

Depending on the neural network's architecture, the dropout algorithm can vary. In most cases, as in our implementation, to remove non-output units, the activation of those units is multiplied by zero.

An important parameter of the dropout algorithm is the dropout rate, the percentage of ignored neurons in a layer. In ML, the standard dropout rate is 0.5 for hidden layers and 0.8 for the input layer, as presented in Goodfellow's 'Deep Learning'(1). However, we used very low dropout rates during our tests, as we have very few neurons. We chose these dropout rates because when using the common dropout rate values, the accuracy was lower than 50%. This low accuracy is most likely due to the low number of neurons in each layer and the size of our dataset, which is (very) small.

During our experiment, we tested different dropout rates and recorded the accuracy of our network. We only varied the number of units in the non-output layers according to the dropout rate while having the other components of the architecture & parameters fixed for the whole period. As in the case of the standard dropout rates, the recorded accuracy scores were consistently lower than those generated without dropout. The only accuracy scores that were close to those recorded with no dropout were in the case of a 0.05 dropout rate, equivalent to a keeping rate of 0.95. In Section 6, the latter results are described in more detail.

4.2.2 Normalization of the dataset

We also decided to normalize our input data. We did this normalization for the hidden layer(s) to avoid giving a feature more important than others due to its bigger range when it is untrained.
We have used the most basic normalization technique to keep our feature vectors as simple as possible. This was also easier to achieve since our model did not have any instances of negative numbers. Consequently, we decided to normalize all feature vectors on a scale from 0 to 1. We used the following formula to do so:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Where x is the current datapoint, x' is the normalized datapoint, x_{max} and x_{min} are respectively the maximum and the minimum values of the current feature vector.

4.3 Accuracy

How do we measure how accurate our model is? Of course, we could take the average loss function value on the testing dataset, but what does this value represent? Given the loss function we have, it is hard to imagine what a loss of 2 means. Furthermore, doctors should use the trained model in practice to make informed decisions about patients. Therefore we decided to measure accuracy slightly differently than just computing the average loss. In particular, the trained model output the probability for the given sample to be malignant. Therefore, we decided to measure accuracy based on how close this probability is to the correct 1 or 0. Intuitively this makes sense since knowing that you have a 55% or 95% percentage of having a serious illness matters. Through the report, we computed three variations on this idea. One that classifies the points based on whether the probability is above or below 50% is the most likely to be according to the model. For the other two, we set up stricter criteria. That is, the model should not only be in the correct half, but also in the last 30%(respectively 10%) from outputting a round 100% or 0% of probability. That is, all the samples which were classified between those 30% and 70%(respectively 10% and 90%) were marked as wrong since they do not provide enough confidence to base a diagnosis on them. In the paper by Woldber *et al.* that motivated our project, they only use the lowest level of accuracy. That is, they take that the model classifies samples based on which probability is higher and rarely consider where this probability lies, except when discussing practical use of the model where they say that it was used in an exceptional case where the cells were not cancerous at all, and the model gave inconclusive results(6).

5 Cross validation

To reach our results and perform our hyperparameter search in a fair and precise way, we conducted a cross validation (CV) scheme. This CV scheme allowed a fair comparison between all combinations in hyperparameters. First, a basic hold-out validation is conducted for each combination of hyperparameters. After it has been tested, if the accuracy of this initial test was higher than 90% on the sharpest accuracy measure as discussed above, the tested hyperparameter combination is passed to the CV scheme. In our CV scheme, we used *k-fold cross validation* with $k = 10$. This means that our set was split into ten disjoint and (approximately) equal subsets. Using Python's *scikit-learn* library's *random_state* feature, we ensured that the dataset was split in the same way for every different hyperparameter combination. In each iteration of the *10-fold cross validation*, a unique set was held out as a testing set while the nine others were merged to be used to train the neural network. Finally, the accuracies for the ten folds are averaged to assess whether the model consistently performs well or the initial performance accuracy was just due to lucky weight initialization in the training phase. The accuracies were calculated using 3 different error thresholds as explained in the legend captions under tables 6.1, 6.2, 6.3.

6 Results

We ran a hyperparameter search to find the best-performing model and investigated combinations of various parameters that influence the performance. These parameters include the architecture (number of neurons and layers), the keep rate for the dropout regularization, learning rate, number of epochs and the number of features reduced by PCA. After running the search, we selected a few best-performing models discussed above and ran 10-fold cross validation on them. All the results for various number of features are present in tables 6.1, 6.2, 6.3.

6.1 Discussion

During development, we experienced a lot of inconsistencies with the performance of some models we have worked with (two runs gave very different accuracy scores). One reason was that the implementation of our loss function was incorrect, and the second reason was that

Best accuracy scores with 5 features

layer 1	layer 2	keep rate	λ	epochs	accuracy
2	0	1	0.1	10000	0.972/0.961/0.924
4	2	0.95	0.1	10000	0.972/0.961/0.924
4	8	1	0.1	10000	0.974/0.968/0.93
8	2	1	0.1	10000	0.974/0.965/0.935
8	4	1	0.1	8000	0.974/0.965/0.926

Table 6.1: Multiple models with highest overall performance for 5 features (the three accuracy scores represent three different confidence ratios - 50/50, 70/30, 90/10 to test how confident the model is in its predictions). λ is the learning rate.

Best accuracy scores with 10 features

layer 1	layer 2	keep rate	λ	epochs	accuracy
8	8	1	0.1	2000	0.975/0.959/0.923
16	4	0.95	0.1	10000	0.972/0.961/0.945
8	0	1	0.1	8000	0.977/0.963/0.937
8	4	1	0.1	10000	0.974/0.963/0.949
8	2	1	0.1	2000	0.975/0.958/0.917

Table 6.2: Multiple models with highest overall performance for 10 features (the three accuracy scores represent three different confidence ratios - 50/50, 70/30, 90/10 to test how confident the model is in its predictions). λ is the learning rate.

too small models had such local minima, which gave poor results (the valleys were not "deep" enough). In such a setting were suffering from the issue of random lucky/unlucky weight initialization.

However, fixing the loss function and adding more neurons created more local minima, many of which gave good results (similarly to deep learning - the cost landscape has such deep local minima that the training needs to be stopped earlier to prevent overfitting). Nonetheless, stopping earlier does not always refer to a small number of epochs, but it could refer to thousands or even tens of thousands, as 6.1 shows. In this figure, we can observe some signs of overfitting after approximately 20000 epochs.

7 Transfer Learning

This section discusses the motivation for our experiment with transfer learning and the results we got.

Best accuracy scores with 30 features

layer 1	layer 2	keep rate	λ	epochs	accuracy
2	0	1	0.1	2000	0.974/0.965/0.944
8	8	0.95	0.1	2000	0.977/0.959/0.921
2	2	1	0.1	8000	0.975/0.966/0.952
4	8	0.95	0.1	8000	0.977/0.965/0.949
4	4	1	0.1	10000	0.979/0.968/0.944

Table 6.3: Multiple models with highest overall performance for 30 features (the three accuracy scores represent three different confidence ratios - 50/50, 70/30, 90/10 to test how confident the model is in its predictions). λ is the learning rate.

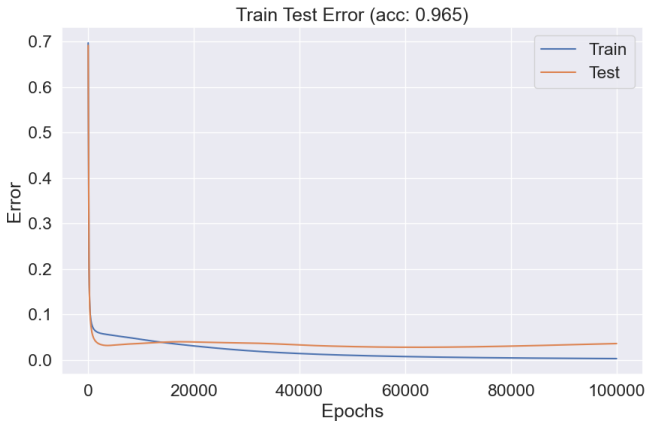

Figure 6.1: Train and test errors for a configuration with 10 features, 8 neurons in the first layer, no neurons in the second layer, no dropout, learning rate of 0.1, and 100000 epochs.

7.1 Background

In this section, we overview the application of transfer learning to our problem. Transfer learning is a method used in deep learning in which we re-initialize the weights of the last layer of an already trained neural network, as it was presented to us by Herbert Jaeger. After that, we re-train the architecture on a new but similar task. The idea of this approach is that the intermediate layers extract features that are also relevant to the new task. Given that we could not find a comparable data set for our task, we decided to split the data set into two parts. We use the first more significant part to train the same architectures as presented in section 6. After that, we attempt to "transfer" the learning to the second data set.

7.2 Motivation

The features we used to train our model are computed from pictures taken by a JVC TK-1070U mounted on an Olympus microscope, as presented in section 2. This camera model is old* and so is the microscope and probably no longer in use. Therefore the model that was computed by us, or by (6) is no longer relevant. We look into transfer learning as a possibility to train a model that would work on new types of cameras and microscopes, given that we precompute the same features. Given that we are transferring on the same data set, this is not a serious application but a proof of concept.

7.3 Results

For the hyperparameter configurations, which looked promising, we decided to split the dataset into three parts, roughly 70/20/10. We performed training on the 70%, which we then transferred on the 20% reserving the last 10% for testing in both cases. In general, we found out that the two models performed similarly on the testing set, up to a difference of some decimal points in their accuracy. This is probably because we are transferring on the same task and dataset; hence, once the last layer was cut and replaced, the training only refines the parameters of the last layer like the previous model. Presenting the

*By old we mean old enough to be a collectable item which goes for a premium, see for example (4).

complete tables as in the previous section would be redundant, and we instead describe the results qualitatively.

In general, it was the case that the accuracy of the second model was slightly lower than in the first, as expected, but this was only a general trend. In some cases, they were somewhat higher. For example, a model with only a single hidden layer of 8 neurons was able to correctly classify 54 of the 57 samples up to the highest accuracy, with a 90% confidence, before transferring. The transfer of the model classified 56 of them correctly. A trend we noted was that all the models which outperformed their pretransfer version used dropout, which intuitively makes sense since we trained them for the same amount of epochs on a significantly smaller dataset and hence overfitting was a possibility.

7.4 Limitations

We only performed one test run for each of the transfers. Hence our results, despite consistency across separate runs, are not correctly verified. As already mentioned, this was more of a side note and proof of concept than a serious application of transfer learning.

8 Conclusion

For our breast cancer cell classification project, we first used various preprocessing techniques such as PCA or correlation matrices to understand our dataset better. Next, we split the dataset into training and testing sets and decided on a neural architecture with a loss function with some default parameters. Then we performed a hyperparameter search and k-fold cross-validation to find the best performing model and compared it to the logistic regression model.

From the section 6 we can conclude that multiple configurations exist for our feedforward network with high accuracy scores. The best accuracies we obtained, 0.979/0.968/0.944 is a configuration that has 30 features, 4 neurons in the first layer, 4 neurons in the second layer, no dropout regularization (keep rate of 1.0), learning rate of 0.1, and 10000 epochs. Teaching us that sometimes tricks such are principal component analysis and dropout are unnecessary. To note also the first row of Table 6.1, in which a neural network with only eight neurons in total archived comparable results.

The performance of the logistic regression model with 10-fold cross validation had an accuracy score of 0.975, which is lower than our model but still very close. To note also that our results are slightly better than the ones from Wolberg *et al.* where a 10-fold cross validation on the same dataset gave an accuracy of 97%(6), unfortunately, we could not retrieve the testing dataset they used on which they had an accuracy of 100%, so this is the best possible comparison.

Furthermore, we experimented with transfer learning and saw that it could yield satisfactory results even on small datasets like ours.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, 2016.

[2] Herbert Jaeger. Neural Networks. `https://www.ai.rug.nl/minds/uploads/LN_NN_RUG.pdf`. Accessed: 03-07-2022.

[3] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal Components Analysis (PCA). *Computers Geosciences*, 19(3):303–342, 1993.

[4] Uknown. TK-1070U. `https://www.radwell.co.uk/en-GB/Buy/JVC/JVC/TK-1070U/?redirect=true`. Accessed: 02-07-2022.

[5] Olvi L. Mangasarian William H. Wolberg, Nick W. Street. Breast Cancer Wisconsin (Diagnostic) Data Set. `https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)`. Accessed: 10-05-2022.

[6] William H Wolberg, W Nick Street, and Olvi L Mangasarian. Machine Learning Techniques to Diagnose Breast Cancer from Image-Processed Nuclear Features of Fine Needle Aspirates. *Cancer Letters*, 77(2-3):163–171, 1994.