Understanding ANN Architecture - Depth and Width Effects on Breast Cancer Classification

STD ID:24060519

1.Introduction

In this work, we use a feedforward ANN for the classification of breast tumours (benign/malignant) from quantitative geomorphometric measurements derived from biopsy images. The point is not just to build a model that works but to observe how the architecture of a Multilayer Perceptron (MLP) – how many layers of neurons it has, and how many neurons are in each layer – actually affects its performance on your data/overfitting of real dataset about medical charges.

The experiments are conducted on the Breast Cancer Wisconsin (Diagnostic) dataset, which is widely used for benchmarking binary classification tasks. Each row in the dataset represents one breast mass with 30 numerical features (mean radius, mean texture, mean perimeter, mean area and so on) discreetised from digitised images of FNA shows. The target column is the diagnosis (M = malignant, B = benign). Its intermediate size of features and dataset as well as the network's simplicity to minimize over-fitting makes it a good playground for architecture experiments.

The included notebook guides you through the whole process: from how to load and explore the data, preprocess it (scale it down), define some ANN architectures, train them with similar settings, and finally evaluate using learning curves and confusion matrices.

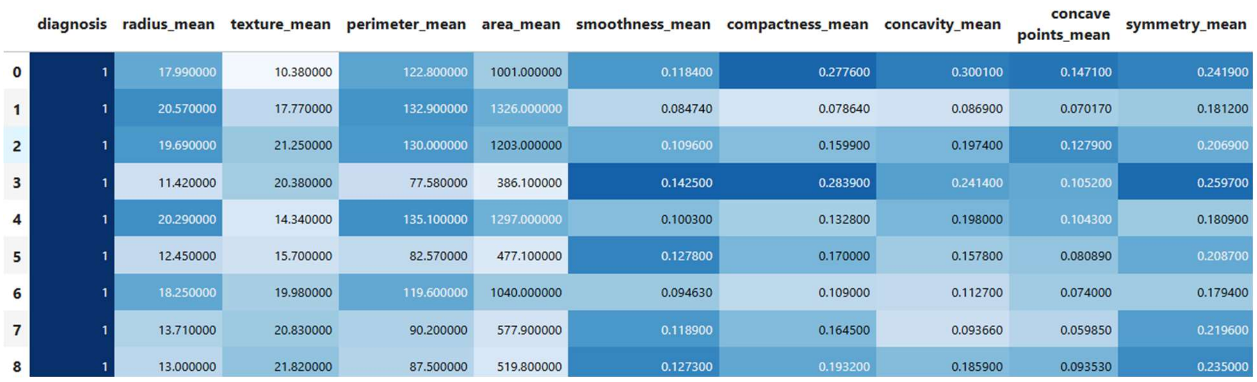
2.Data and Preprocessing

The start with a version of the Breast Cancer Wisconsin dataset in CSV format. Once I loaded it into a pandas dataframe, there are 569 samples over 33 columns; an id column and an empty Unnamed: 32 column. These two columns are dropped, so we have 30 feature columns and the target "diagnosis".

A brief look at the label distribution tells me that they are not exactly balanced – benign cases are slightly more represented than malignant ones (357 benign versus 212 malignant). This slight imbalance is common in medical data and can still be handled (especially with stratified splits).As neural networks work with numeric target the diagnosis column is encoded using label encoder, whereby benign tumors become 0 and malignant tumors become 1. The feature matrix XXX now contains the 30 measurements, and yyy holds the encoded labels.

In order to obtain reliable estimates of how well the two models generalise, data is split into three. First, 20% of the complete dataset is reserved as a test set. Next, 20% of the remaining 80% forms a validation set. This makes approximately 60% training, 20% validation and 20% test. Stratified sampling enables that the proportion of positive and negative cases is balanced in each subset.

ANNs require feature scaling because the raw input features are in different scales, and with outliers. Instead of a standard scaler, we use a RobustScaler in this work. RobustScaler uses the median and interquartile range to center and scale data, ensuring that outliers in medical measurements doesn’t impact the accuracy of your models as they would with standard scaler. The scaler is trained on the training but applied onto validation and test set.



	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	1	17.990000	10.380000	122.800000	1001.000000	0.118400	0.277600	0.300100	0.147100	0.241900
1	1	20.570000	17.770000	132.900000	1326.000000	0.084740	0.078640	0.086900	0.070170	0.181200
2	1	19.690000	21.250000	130.000000	1203.000000	0.109600	0.159900	0.197400	0.127900	0.206900
3	1	11.420000	20.380000	77.580000	386.100000	0.142500	0.283900	0.241400	0.105200	0.259700
4	1	20.290000	14.340000	135.100000	1297.000000	0.100300	0.132800	0.198000	0.104300	0.180900
5	1	12.450000	15.700000	82.570000	477.100000	0.127800	0.170000	0.157800	0.080890	0.208700
6	1	18.250000	19.980000	119.600000	1040.000000	0.094630	0.109000	0.112700	0.074000	0.179400
7	1	13.710000	20.830000	90.200000	577.900000	0.118900	0.164500	0.093660	0.059850	0.219600
8	1	13.000000	21.820000	87.500000	519.800000	0.127300	0.193200	0.185900	0.093530	0.235000

Figure 1: Breast Cancer Wisconsin dataset

3.Model Architecture

The class of the notebook is experimenting various ANN architectures to see how they perform. All of the architectures are plain MLPs created using Keras and here's what they all have in common:

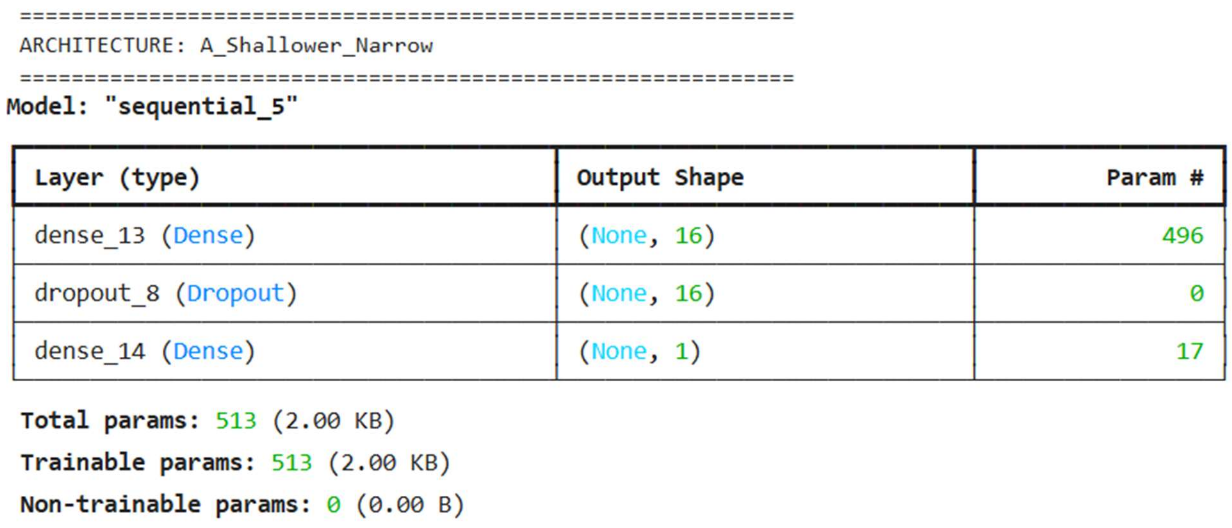
- Hidden layers activated by the ReLU function
- One sigmoid output neuron to estimate the probability of tumour being malignant
- Those matrices can be compared by the loss binary cross-entropy.

Adam as a Method for Stochastic Optimization time t by among all rangon only the “t” presented Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions.

The input layer is always 30 units as there are 30 scaled features. In addition, several hidden-layer structures are examined to investigate depth and width:

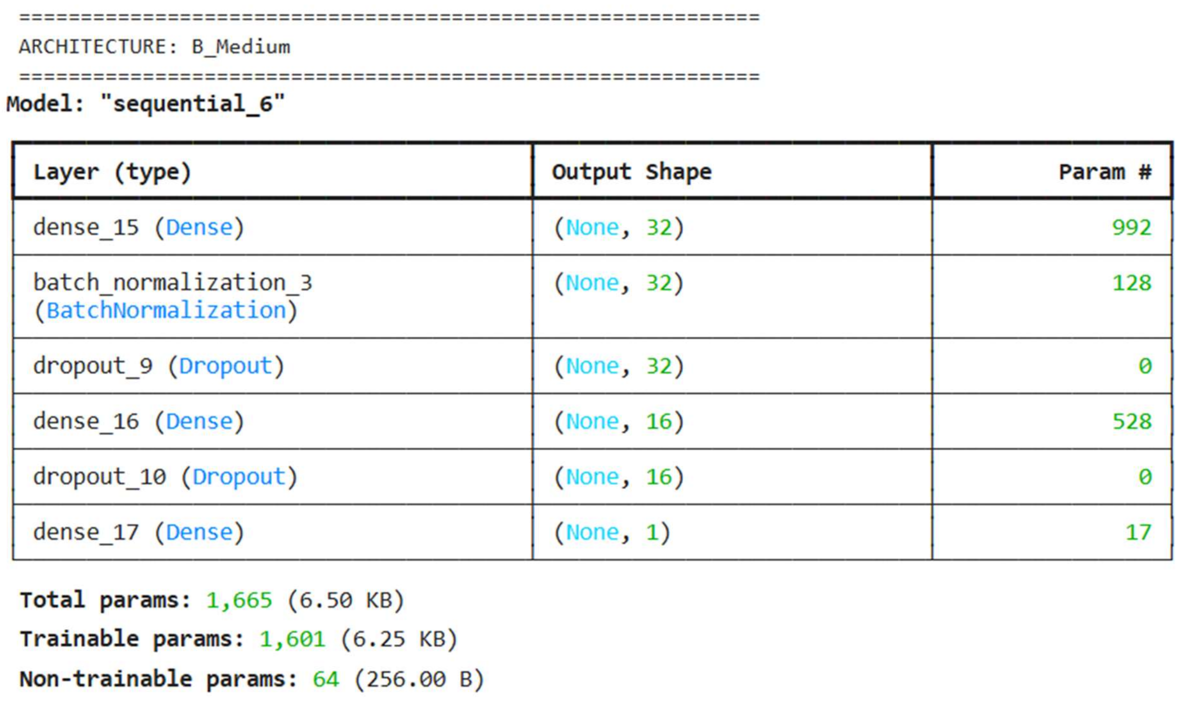
- An under-parametrized network with small size and small width
- A somewhat deeper network with two middle-sized hidden layers
- A wider and deeper network with larger layers, providing more capacity but also as being at higher risk of overfitting.

Methods have included Batch Normalization and Dropout to achieve better stability, as well as reduced overfitting. Dropout randomly turns off a percentage of neurons during training, acting as a regulariser and Batch Normalization keeps the activations within a layer normalised which can accelerate and stabilise learning.All models are compiled with the same loss and optimiser, so differences in performance mainly reflect architectural changes rather than training-setting changes.



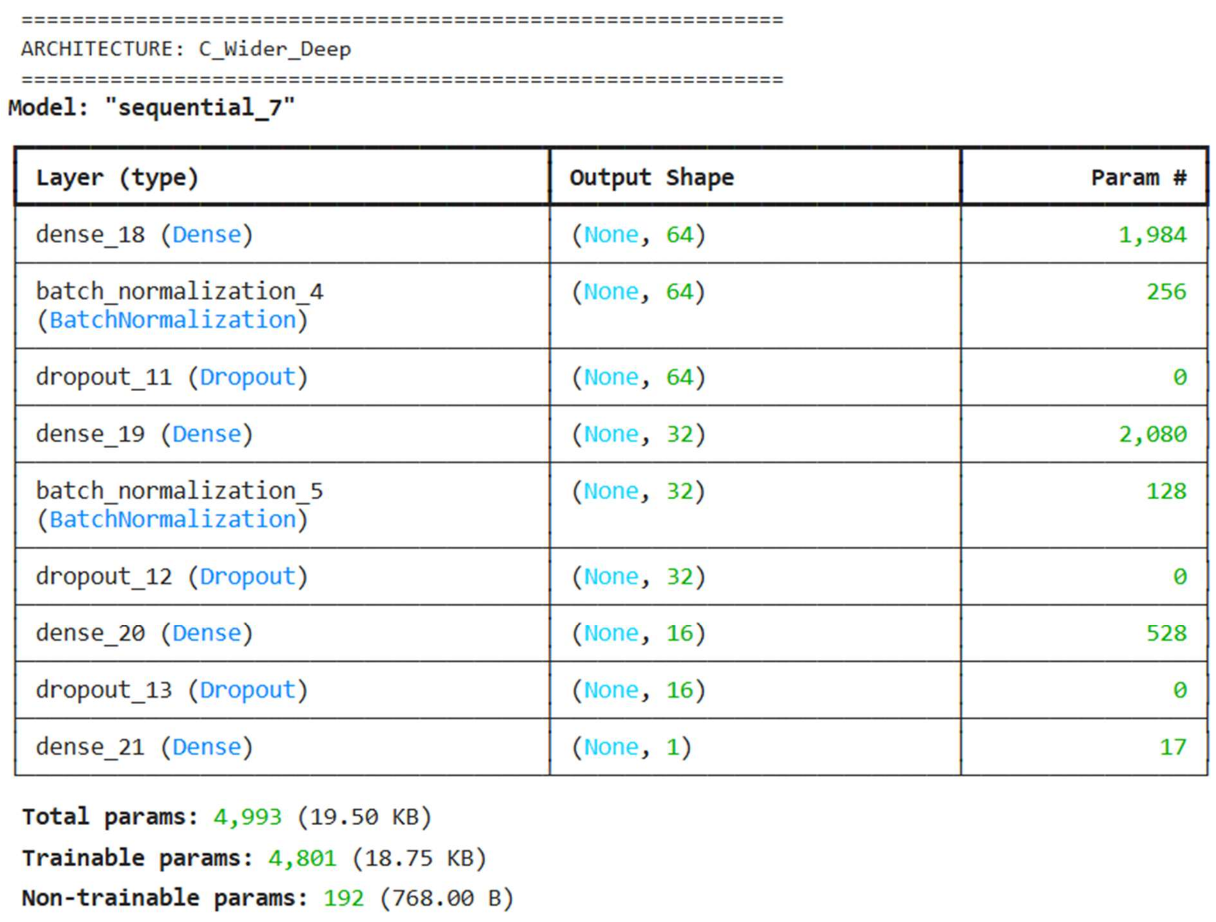
```
============================================================
ARCHITECTURE: A_Shallower_Narrow
============================================================
Model: "sequential_5"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense_13 (Dense)                │ (None, 16)             │           496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_8 (Dropout)             │ (None, 16)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_14 (Dense)                │ (None, 1)              │            17 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 513 (2.00 KB)
 Trainable params: 513 (2.00 KB)
 Non-trainable params: 0 (0.00 B)
```

Figure 2: A Shallower Narrow architecture



```
============================================================
ARCHITECTURE: B_Medium
============================================================
Model: "sequential_6"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense_15 (Dense)                │ (None, 32)             │           992 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_3           │ (None, 32)             │           128 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_9 (Dropout)             │ (None, 32)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_16 (Dense)                │ (None, 16)             │           528 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_10 (Dropout)            │ (None, 16)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_17 (Dense)                │ (None, 1)              │            17 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 1,665 (6.50 KB)
 Trainable params: 1,601 (6.25 KB)
 Non-trainable params: 64 (256.00 B)
```

Figure 3: B Medium architecture



```
============================================================
ARCHITECTURE: C_Wider_Deep
============================================================
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_18 (Dense)	(None, 64)	1,984
batch_normalization_4 (BatchNormalization)	(None, 64)	256
dropout_11 (Dropout)	(None, 64)	0
dense_19 (Dense)	(None, 32)	2,080
batch_normalization_5 (BatchNormalization)	(None, 32)	128
dropout_12 (Dropout)	(None, 32)	0
dense_20 (Dense)	(None, 16)	528
dropout_13 (Dropout)	(None, 16)	0
dense_21 (Dense)	(None, 1)	17

```
Total params: 4,993 (19.50 KB)
Trainable params: 4,801 (18.75 KB)
Non-trainable params: 192 (768.00 B)
```

Figure 4: C Wider Deep architecture

4.Training Procedure

Training is with the scaled training set, and we monitor at the validation how each individual model performs as it learns. In order to keep the training performance optimal and avoid overfitting, two Keras callbacks are utilised:

- EarlyStopping: It will give the validation loss to look at and stop training if it hasn't improved in a while (restoring the best weights seen so far).
- ReduceLROnPlateau : It will decrease your learning rate when you see that a variable doesn't change — in many cases (but not all) this allows to have more measured steps during training.

All architectures are allowed to be trained for the same fixed maximum number of epochs with a batch size of 32, but in practice training terminates early due to early stopping. The training history for each model (i.e., the accuracy and loss on both the training and validation data) is written to disk so it can be plotted at a later time.

After training for a specific model is completed, the model is evaluated on the held-out test set to provide an unbiased estimate of the performance. We aggregate all these results into a DataFrame for convenient comparison.

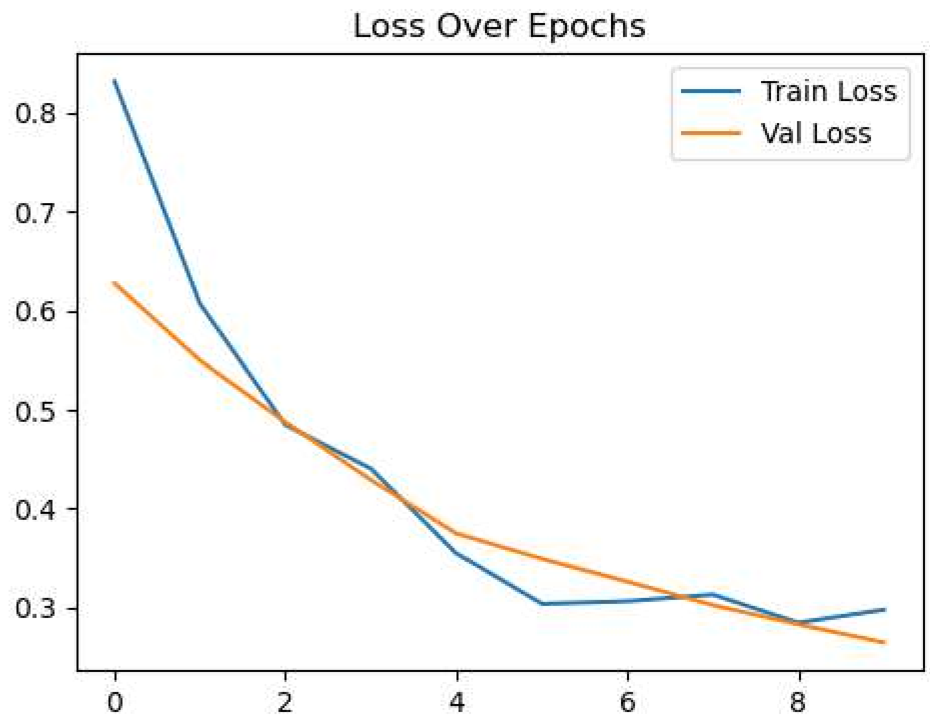


Figure 5 : Learning curves for each network (training and validation accuracy and loss)

5.Results and Evaluation

In the notebook results table, for each architecture, the best validation accuracy achieved during training and the final test set accuracy is found. In general the shallow model with a single small hidden layer does well and provides a useful baseline, but is not fully squeezing all possible performance. Introducing another hidden layer with a reasonable number of neurons improves the accuracy on both datasets, confirming that this little bit bigger network turns out to strike just the right balance between exibility and simplicity.

This largest and deepest network should reach the highest training accuracy which is occasionally close to perfection, but it never has a corresponding highest validation and test context-dependent agreement. Indeed, those scores can plateau or even slightly decline. This is a classic overfitting situation: the model has the capacity to store all of the training data, but its ability to provide predictions for new data samples is degraded. It’s pretty easy to spot in this case due to the increasingly divergent training and validation accuracy, as well as increasing validation loss.

Once the best architecture is chosen for validation and test accuracy, we take a deeper look at predictions in the testing set by the notebook. A confusion matrix will be created, displaying true positives, true negatives, false positives and false negatives. *(i) Classification report for both benign and malignant classes is also printed out which shows precision, recall and F1-scores.

For a medical task such as breast cancer diagnosis, recall for the malignant class is of vital importance since false negatives (predicting benign when in fact the tumour is malignant) could potentially lead to delaying or avoiding treatment. The selected network is able to achieve high overall accuracy while maintaining the recall for malignant cases high as well, so it is reasonably adequate as a decision-support tool.

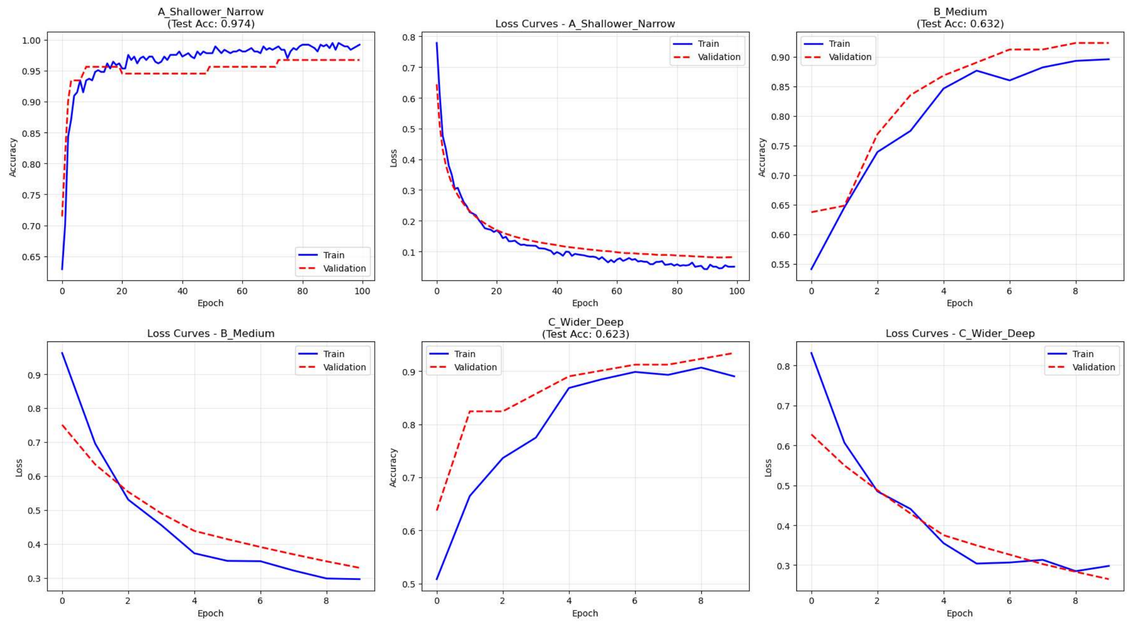


Figure 6 : Validation and Test accuracy for each architecture.

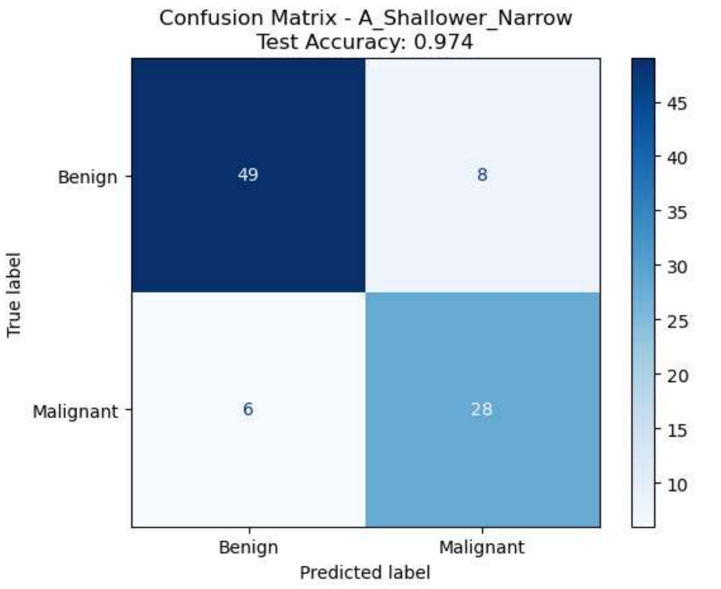


Figure 7 : Confusion matrix plot for the best model on the test set.

6.Discussion and Conclusion

The experiments in this notebook clearly demonstrate how the performance of an ANN is highly sensitive to architectural decisions, particularly over a small tabular dataset like this. There are only a couple of hundreds of samples there really is a trade-off between model capacity and generalisation:

- A shallow network can underfit the data and overlook fine patterns.
- A medium-sized network (with two or even more hidden layers) trained with a moderate number of neurons obtains the best compromise, learning useful structure without overfitting noise.
- A network that is both wide and deep can easily be too complex, achieving close-but-not-perfect training accuracy while validation or test performance seem to plateau even degrade.

Learning curve + confusion matrix is a nice visual way to see what's happening. Observing validation loss and accuracy dynamics across epochs, employing early stopping and monitoring the divergence of training from validation scores are all methods for detecting overfitting sooner. Developed results also emphasize the significance of efficient scaling and rigorous splitting to training, validation and test data.

For tabular classification problems using the same steps as in the notebook is a good starting point:

- Clean and encode the data
- You should also scale your numerical features (hopefully with a method that is robust to noisy data)
- Define some architectures covering a reasonable amount of capacities
- Re-train them with early stopping on as usual gradient descent.
- Compare the validation and test scores for that matter
- Explore learning curves and confusion matrices for the best

In general, the stuff that we do in this file is not limited to just creating a classifier for breast cancer diagnosis. The concrete details demonstrate how to think of network depth and width, and we don't just guess for that anymore.

7. References

- Dua, D. and Graff, C., 2019. Breast Cancer Wisconsin (Diagnostic) Data Set. UCI Machine Learning Repository, University of California, Irvine. Available at: https://archive.ics.uci.edu
- Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep Learning. MIT Press. Chapters 6–7 (deep feedforward networks and regularization).
- Chollet, F., 2018. Deep Learning with Python. 1st ed. Shelter Island: Manning Publications. Examples using Keras for classification and overfitting control.
- Pedregosa, F. et al., 2011. 'Scikit-learn: Machine Learning in Python', Journal of Machine Learning Research, 12, pp. 2825–2830. (Used for train/test split, scaling and evaluation utilities.)
- TensorFlow Developers, 2024. Keras API Reference. TensorFlow Documentation. Available at: https://www.tensorflow.org/api_docs (Accessed 11 December 2025).
- Lu, Z., Pu, H., Wang, F., Hu, Z. and Wang, L., 2017. 'The Expressive Power of Neural Networks: A View from the Width', in Advances in Neural Information Processing Systems (NeurIPS). arXiv:1709.02540. (Background on depth/width trade-offs.)

GITHUB: https://github.com/Rohitha13/breast-cancer-ann-depth-width-tutorial