Foundations and Frontiers of Machine Learning
Group Assignment 2
Group 1

Contents

List of Figures

List of Tables

1 Table of Contributions

Name	**Student ID**	**Contribution**
Emmanouil Tsolias	239490757	Graded Assignment 2

Table 1: Individual Contribution to Group Work

2 Data Visualisation (Task 2)

2.1 Visualing with PCA

Machine learning models use data to finetune the values of their internal parameters while training. Bigger datasets can contain more information but sometimes multiple features contain the same information scaled. This becomes more likely with larger datasets. Moreover, each additional feature adds a new dimension to the dataset which makes the samples sparse, and visualisation harder. (Khee, 2022).

This can be addressed by constructing new features that encompass the dataset's information without repetition, using with the Principal Components Analysis (PCA) technique. The first new feature is engineered to explain as much variance as possible from all features of the dataset (Maćkiewicz, 1993). Each next one is engineered to contain as much of the information that is left, without repetition, which means that the last features contain little to no additional information (depending on how linearly independent the dataset was) and can be discarded, allowing for easier visualization and reducing the overall noise. Also, the features are now in descending order in terms of variance explained so we can select the first n ones, knowing that we have retain maximum information per feature. We can now select a number that is suitable for visualization. The new features have no actual meaning but can be used to show the distribution of the samples in the space, informing us about the separability of the classes.

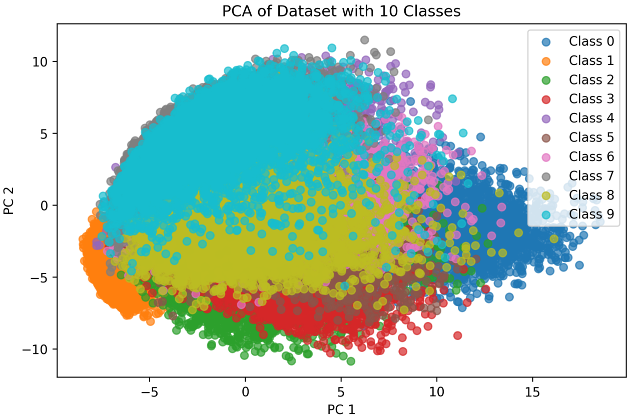
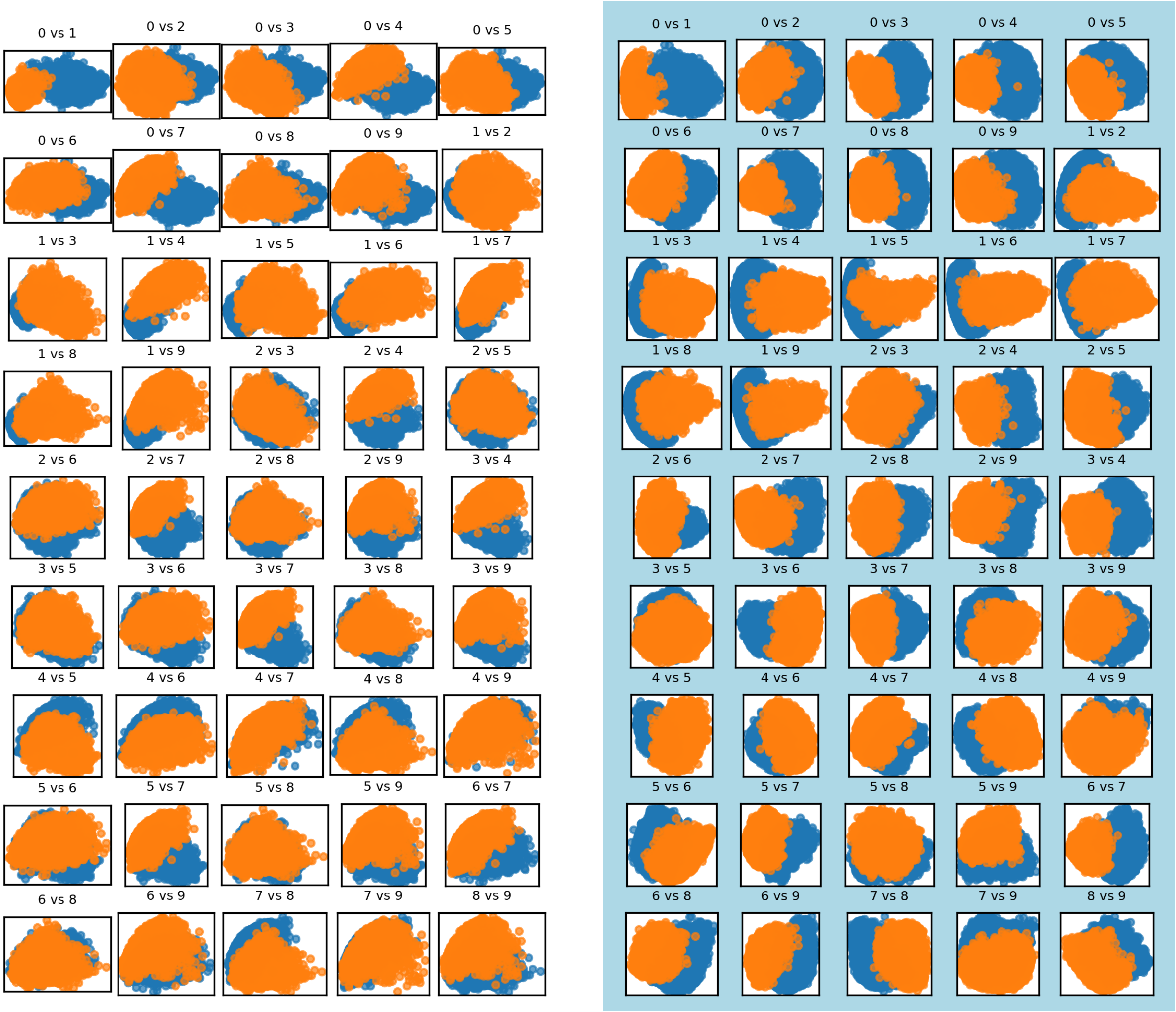


Figure 1: Plot of 2 first components of PCA for all digits of the MNIST dataset.

If we plot the first two components with different colour for each class, we can see which classes overlap, and which ones are linearly separable. Thus we have that, for example, the pairs (0, 1), (0, 2), and (2, 9) should be easy to differentiate since the occupy opposing parts on the plane. Still, due to the significant overlap caused by 10 different classes, the distribution of some of the classes is obscured.

Figure 2a provides a pairwise visualisation of the distributions. It Is clear that some pairs of classes, e.g. (0, 1), (0, 7), (3, 7) are relatively distinct, while others (5, 8), (7, 9) are virtually impossible to separate. However this is the result of considering just 2 components and not all of the information contained in the dataset. That means distributions that do not seem differentiable now, may actually be when the model is trained on the original dataset. Another point is that the above results are obtained by performing PCA on the whole dataset. However, when it comes to distinguishing classes in pairs, the datapoints of other classes introduce variance and noise that may be irrelevant. Thus, it is worth producing the same plot with the PCA calculated for each pair. This is shown in figure 2b and it becomes apparent that the results are better in terms of separability. To objectively measure this, we use the silhouette score which takes into account the average distance of a clusters samples between each-other and between the samples of the other class (Shahpure, 2020). Higher scores mean better clustering and for the generic PCA we get 0.337 where as the same score for the pairwise PCA is 0.407. This proves that we can achieve better clustering (and hence prediction performance) by performing pairwise PCA.



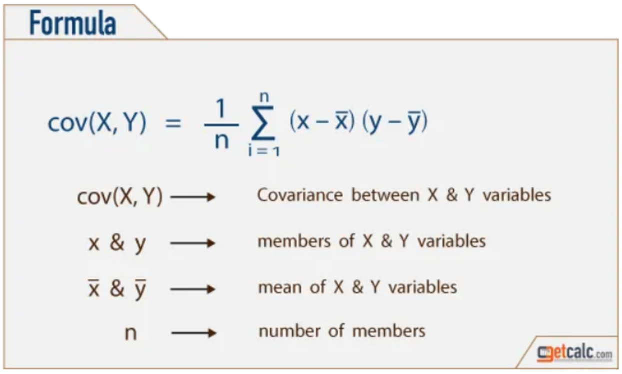
(a) Plot of the first 2 PCA Components for all pairs of digits.

(b) Pair-wise PCA plot.

Figure 2: Side-by-side visualization of PCA components and pair-wise PCA plots.

2.2 Mathematical basis of PCA

Before applying and transformations on the dataset, it is important to normalize it so the features with larger ranges do not affect the calculations disproportionately.



Formula

$$\mathrm{cov}(X, Y) = \frac{1}{n} \sum_{i=1}^{n} (x - \bar{x})(y - \bar{y})$$

cov(X, Y) ⟶ Covariance between X & Y variables

x & y ⟶ members of X & Y variables

$\bar{x}$ & $\bar{y}$ ⟶ mean of X & Y variables

n ⟶ number of members

getcalc.com

Figure 3: Calculation of Covariance Matrix (Dubey, 2018).

The next step is to identify how much each feature correlates with the other ones, as the more they correlate, the less useful information they add. Figure 4 shows the steps of calculation, which essentially shows that the more two variables follow the same trend, then larger the product of their difference to the mean will be

From Linear Algebra we know that a matrix can represent a transformation and its eigenvectors represent the directions in which the data has the most variance. Thus these eigenvectors will form the basis of the new vector space which is now built so that each new dimensions contains the most variance

The actual amount of variance explained by each dimension is given away by the eigenvalues of the eigenvectors. We can create a new matrix by concatenating the desired number of eigenvectors which, when multiplied with our data, produces the PCA-Data, which is our data transformed to the new base.

3 Perceptrons (Task 2)

The human neurons receive multiple stimulations that are then regulated (scaled). The result then is not a linear stimulation but instead a binary state of excitation, equivalent to "making a decision" (Queensland Brain Institute). Rosenblatt's perceptron machine is a software analogous which receives a multidimensional input that it individually scales, and then compares its sum to a bias in order to select a binary response (Rosenblatt, 1958). If the input is the coordinates of some samples the classes of which are known, we can calculate the weights and the bias, which are trainable parameters, to produce the desired response. Since the described operation is the equation of the line or a hyperplane, we can set up an algorithm that hypotheses which plane would separate these classes based on the values of their coordinates, and if this hypothesis is wrong, it changes it so that it would work well for these specific samples. If the samples occupy distinct areas of the feature space and we do that for all samples, the resulting hyperplane would end up satisfying the requirement for all samples (since we know the samples of a cluster have similar feature values) (Rosenblatt, 1958). From Task 1 we know that this applies to a degree, for the digit pairs.

The algorithmic approach consists of two functions. The first implements the matrix multiplication/scaling and the comparison with the threshold to return a result/decision. The second iterates through the samples of the dataset, producing a prediction and then correcting the weights so that if the prediction was wrong, next time it will be closer to right. The algorithms stops upon achieving an MSE goal or reaching the iterations threshold. With each iteration the number of misclassifications reduces on the training dataset, but the same is not always true for the test dataset, as the two contain different samples and too accurate a modeling of the training samples can end up imprinting noise on the model (overfitting). This is demonstrated by the training vs test accuracy, over number of iterations in figure 4.

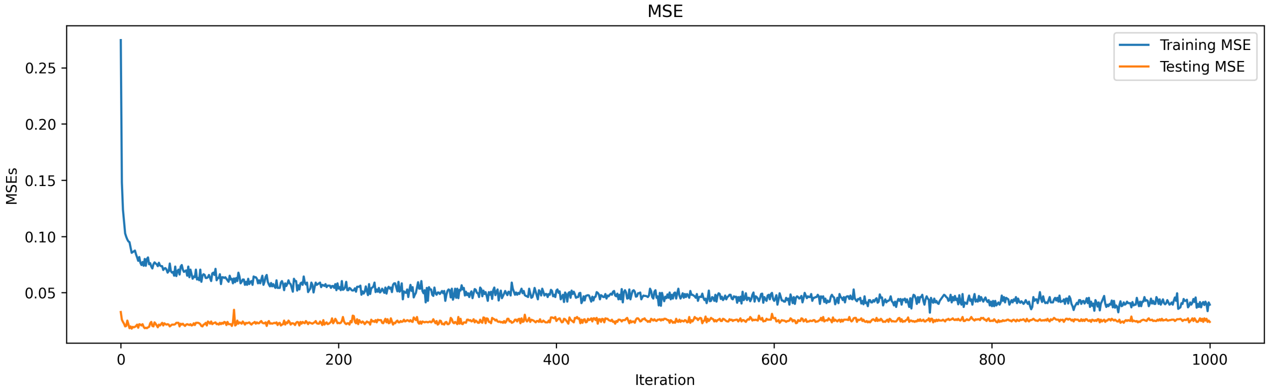
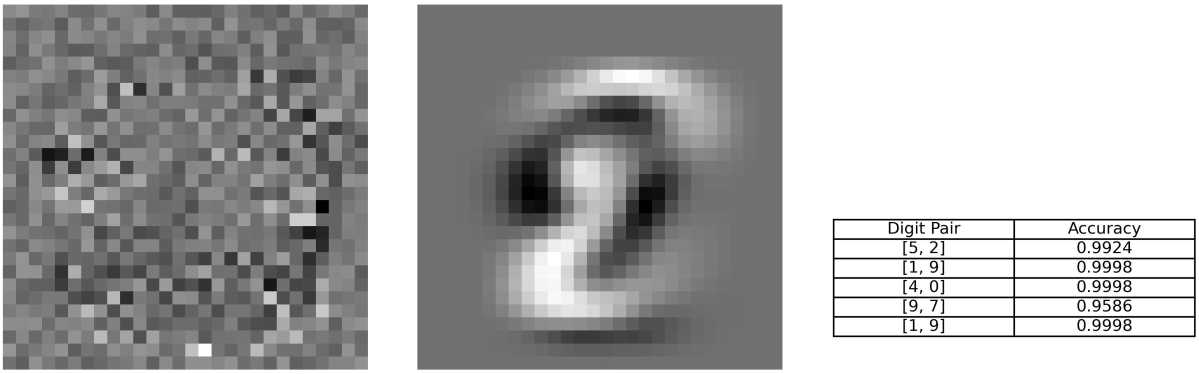


Figure 4: Training vs Testing Mean Square Error (MSE).

On one hand it is clear that peak accuracy is achieved at 10-20 iterations, after which the reduction in training MSE seems to indicate overfitting rather than actual improvement. Since the weights are randomly initialized at each run, multiple runs equal to simulated annealing and can serve to expose entrapment to local minimum. As the MSE is repeatedly similar, we can infer that the algorithm does not converge to a poor local minimum.

Given the baseline MSE calculated in the notepad as 3.6, it is evident that the model has actually learned to distinguish between the two classes (8 and 9). The weights matrix has the same properties as the input matrix and thus, be reversing the process of transforming the digit images to input arrays, we can produce an image of the learned weights. Figure 5a provides this visualization, and makes clear that this particular simple model does not learn any high level features a human would use. Instead it may opportunistically utilize whichever pixels help it correctly split groups of samples that the training so far has not been able to assist it with, similar to nodes that split high impurity leaves in decision trees. Figure 4 shows the difference between the average image of 8 and 9. Areas with high or low brightness indicate consistent differences between the classes that could be used for a rule based system, yet we see that the model has focused elsewhere.



Digit Pair	Accuracy
[5, 2]	0.9924
[1, 9]	0.9998
[4, 0]	0.9998
[9, 7]	0.9586
[1, 9]	0.9998

(a) Visualisation of weights learned by the Perceptron

(b) Difference between average hand-drawn 8 and 9 characters

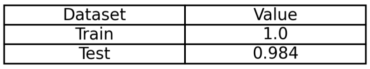
(c) Accuracies for various digit pairs

Figure 5: How the model 'sees' the samples vs How we would expect it to see. Achieved accuracies for random pairs

Figure 6 is a compilation of other pairs of digits with the respective achieved accuracy after training. We see variations that are aligned with how visually similar the two digits are, (that is the handwritten versions as the digital ones can differ e.g. handwritten '5' often lookls more like an 'S'

4 Multilayer Perceptrons (Task 3)

To solve the problem of the perceptron's simplicity, we can organize large groups of them into Neural Networks (NN). The motivation stems from a property called "emergence" where the capabilities of a structure exceed the sum of its parts (Pedder, 2023). In the software domain, the model can learn complex curves to separate the classes more effectively than hyperplanes can (Choi, 2020). While setting up the network with the Keras Library, there are multiple hyperparamters to select. Number of epochs is similar to the iterations in task 2, and batch size allows the model to work with groups of samples speeding up training. The amount of time needed to find the best combination increases exponentially so using guidelines (Bengio, 2012) and some tests, we find one that is sufficient. The resulting training and test accuracy is as follows:.

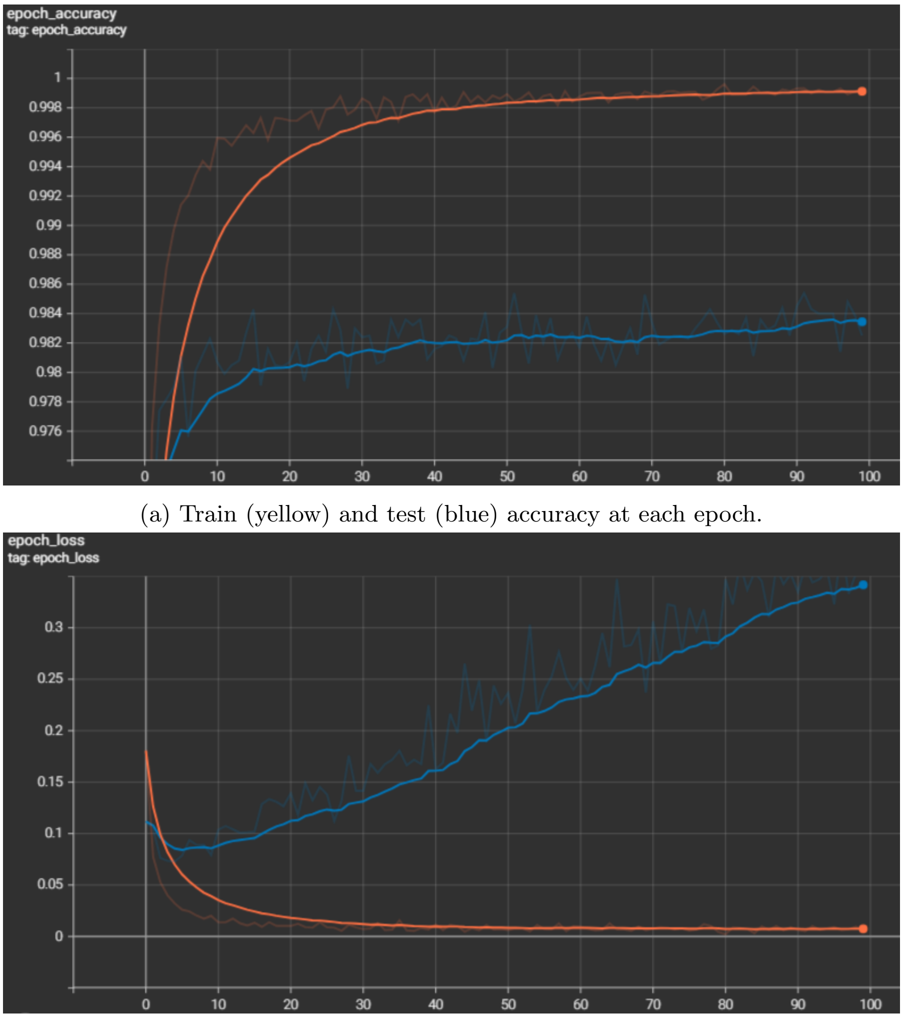


Dataset	Value
Train	1.0
Test	0.984

Figure 6: MLP accuracy on train and test datasets

At first sight it might seem as if the accuracy is lower that with the perceptron. However, the model now performs multiclass classification, not binary classification, which is significantly more demanding.

We leave the model to train for 100 epochs to monitor accuracy and loss. We can see that accuracy score



(a) Train (yellow) and test (blue) accuracy at each epoch.

(b) Train (yellow) and test (blue) loss at each epoch.

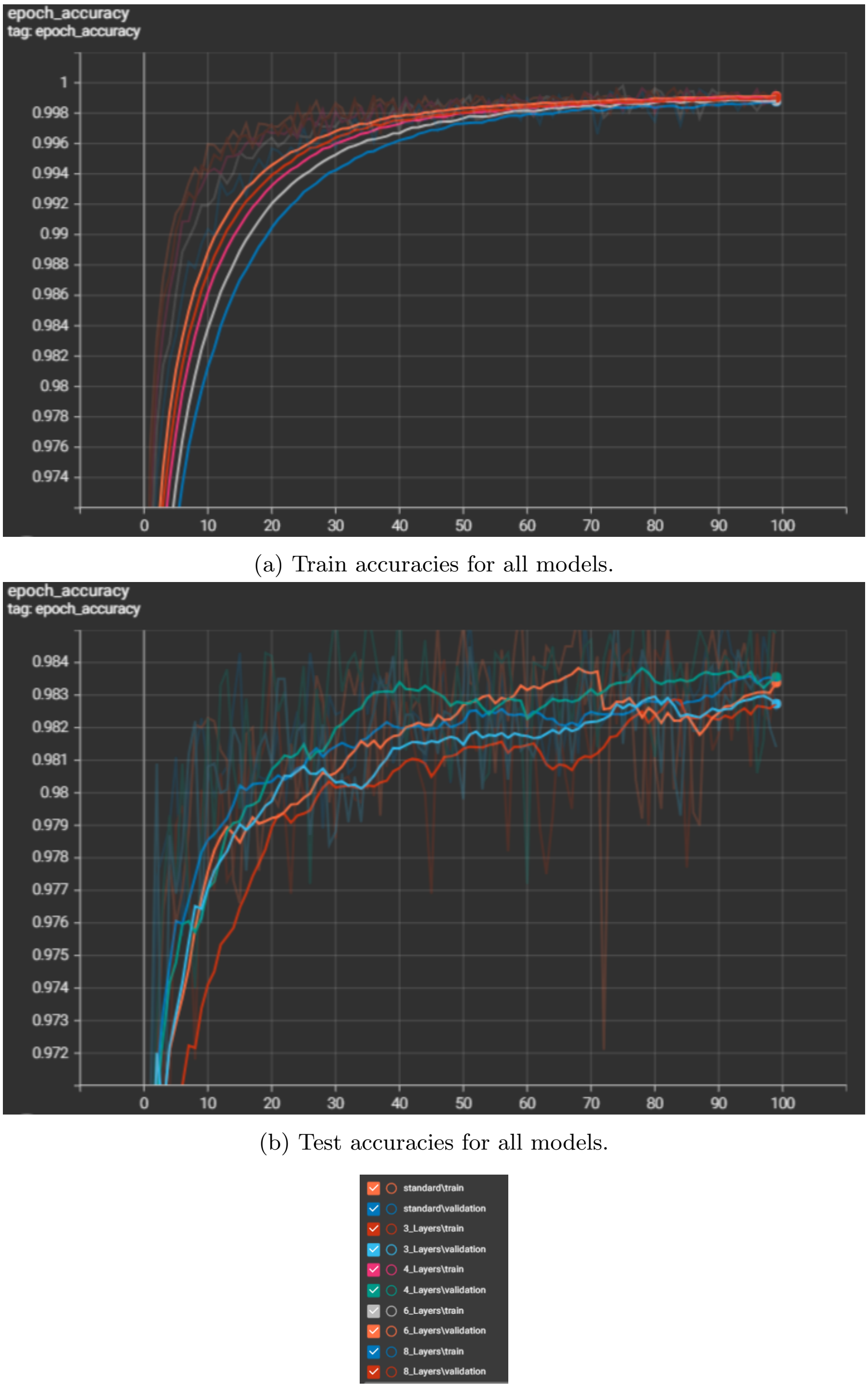
Figure 7: Train and test metrics vs epochs for standard MLP. Smoothing is set to 0.9.

on the validation dataset gets close to its maximum within 30 epochs, but does not stop improving till the end, albeit very slowly. On the other hand, test loss reaches it minimum at around 5 epochs, after which it steadily increases while training loss keeps reducing, indicating overfitting. It is noteworthy that comparing (validation) accuracy and loss we get contradicting results. This is due to the fact that loss evaluates the output probabilities while accuracy evaluates the results. As the model overfits, the model becomes “confused” and less certain but not so much so that it completely misses the target (Smith, 2017).

Next we investigate the relationship between network layers and accuracy by training and testing multiple MPLs of various depths with all other hyperparameters constant The number of neurons per layer is decreased to 500 to achieve training in reasonable time. We produce 4 models with 3, 4, 6 and 8 such layers.

Looking at the train curves, we see that the train performance decreases with each added layer, indicating overfitting. The fact that the standard model is still the best could indicate that trading ”width for ”depth” is the wrong thing to do here. Similar things apply for the test dataset but with less consistency. The differences are of the order of 0.1% and there is some randomness involved (wheights init, train/test split), but the trend is reproducable accross different runs, so it would appear that all models hit a ceiling at around 98.3% accuracy. We would expect some drop to accuracy for larger models owning to overfiting but as we saw before, the degradation to performance appears on the loss.

Regarding the number of parameters for each model, the results are as follows: For fully connected layers, each neuron corresponds to one parameter for each input, plus the bias. Therefore, for the first layer, we



(a) Train accuracies for all models.

(b) Test accuracies for all models.

(c) Legand.

Figure 8: Train and test accuracies of all models. Smoothing is set to 0.9.

calculate:

$$(784 + 1) \cdot 500 = 392,500.$$

Subsequently, each layer adds:

$$(500 + 1) \cdot 500 = 250,500$$

parameters, and the softmax layer adds:

$$(500 + 1) \cdot 10 = 5,010$$

parameters.

Thus, the total number of parameters works out to:

$$397,510 + 250,500 \cdot n,$$

where n is the number of layers.

We can plot these values along with the respective accuracies.

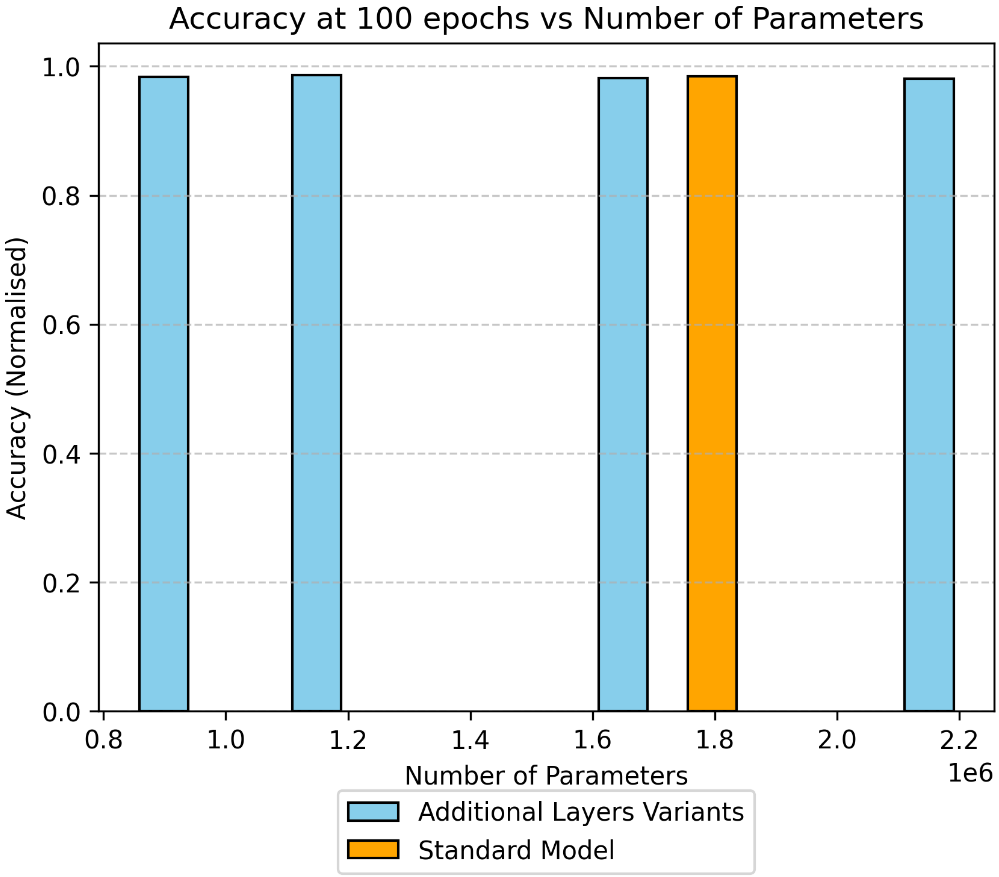


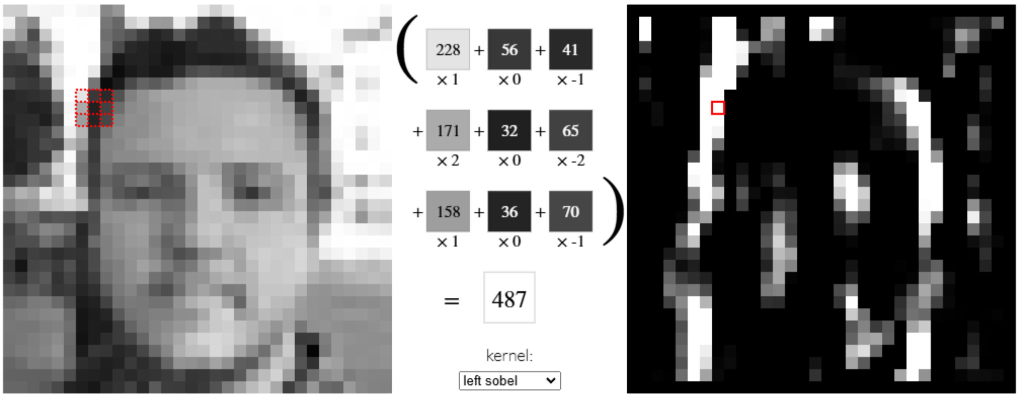
Figure 9: Accuracy for different architectures. No linear trend apparent but the models in the center seem to perform slightly better.

The main model is second with most parameters because of the wider layers. At the large scale of things, all architectures perform similarily in terms of accuracy and as discussed, overfitting manifests itself as higher loss only. We conclude that for this problem, lighter configurations would be beneficial.

5 Convolutional Neural Network (10 points)

Convolutional Neural Networks (CNNs) perform calculations on groups of features instead of individual ones. The calculation uses a kernel, that is a matrix of weights, that gets applied on the similarily sized slices of the sample using the rolling window approach. More specifically, each value of the kernel is mulitplied with the respective value of the sample's matrix. with all the products being summed. That means that the applied calculations now take geometry or time (or other forms of oganisation) into account. The produced output is a map of the places where the structure of the kernel is found in the original image.

In a typical CNN architecture, multiple kernels are applied to the input layer which itself may consist of multiple layers. The kernels match the input in terms of depth as well and each kernel produces an additional layer for the output. These are visualised in figure 11



kernel: left sobel

Figure 10: Application of the left sobel matrix highlights vertical high to low transitions of the face image (Powell)

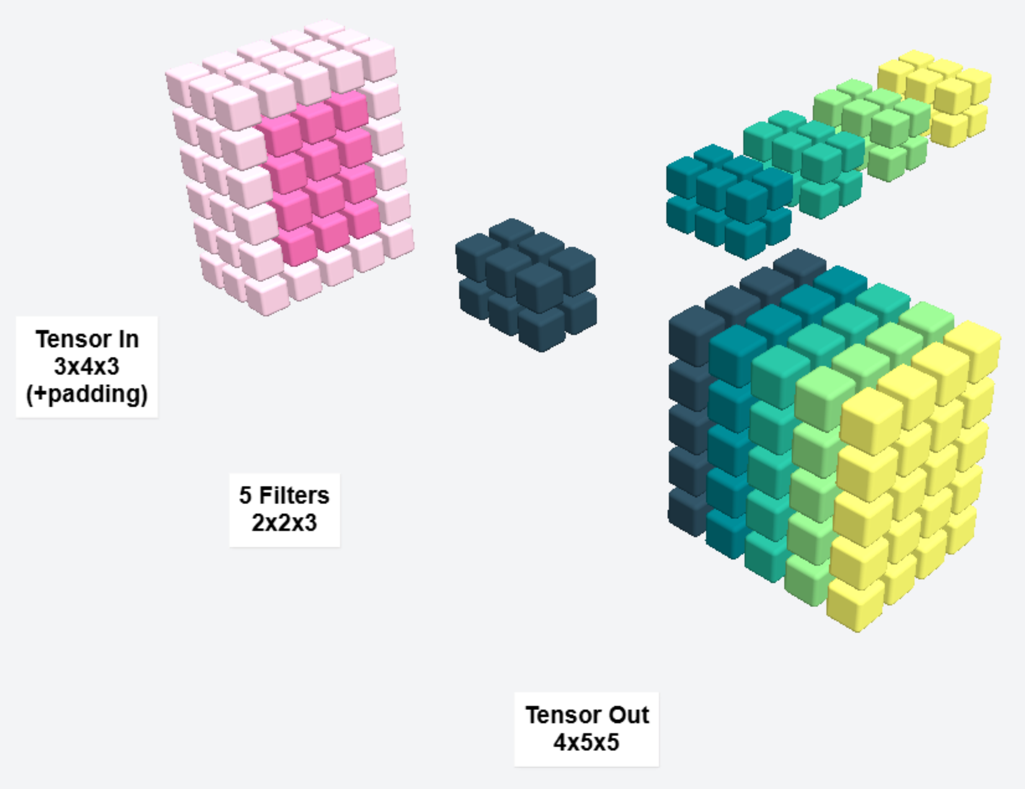


Figure 11: Characteristics of input and output layers in CNNs (made with Convviz)

This array of extracted features can then be used by higher order layers to synthesize more complex features, allowing the identification of complex entities. An example is shown in figure 12 where speed signs are identified as specific combinations of simpler shapes.

We see that the CNN model performs better on the test dataset than any of the MLPs. Plotting the train and test accuracies with tensorboard in figure 14, we can see that for the test data, peak accuracy is reached after 6 epochs. After that, training accuracy keeps increasing while test accuracy has a slight downwards trend, indicating overfiting.

We will now make alterations to the Standard model to see how different hyperparameters affect it's behaviour. Focusing less on the initial epochs, the first thing that stands out is that increasing the kerne size from (4, 4) to (5, 5) is the only tweak that actually harms the accuracy Increasing the model's depth by adding a fourth layer seems to be the best tweak, though the increased number of parameters makes training slower. Same goes for the wider version. In addition, these two variants maintain the upwards trend till the end, sug-

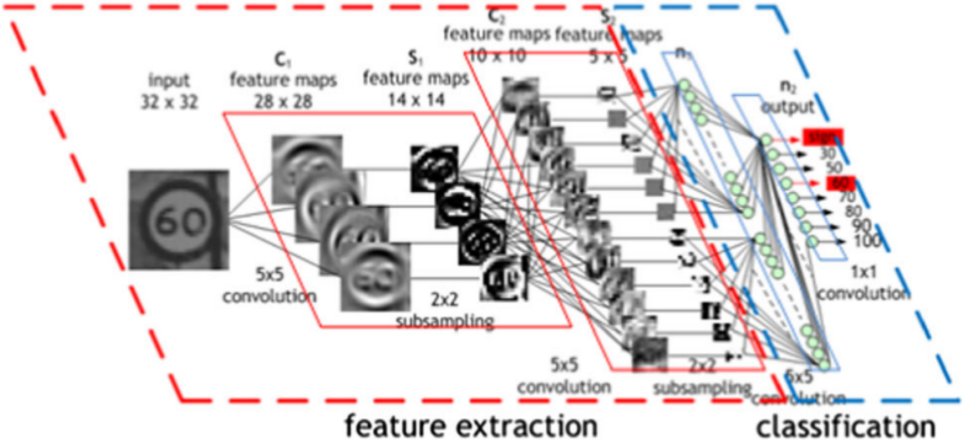
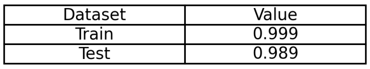


Figure 12: The CNN uses simple features (like the left sobel from figure 10) to construct complex ones (Ghoshal, 2020)



Dataset	Value
Train	0.999
Test	0.989

Figure 13: CNN accuracy on train and test datasets

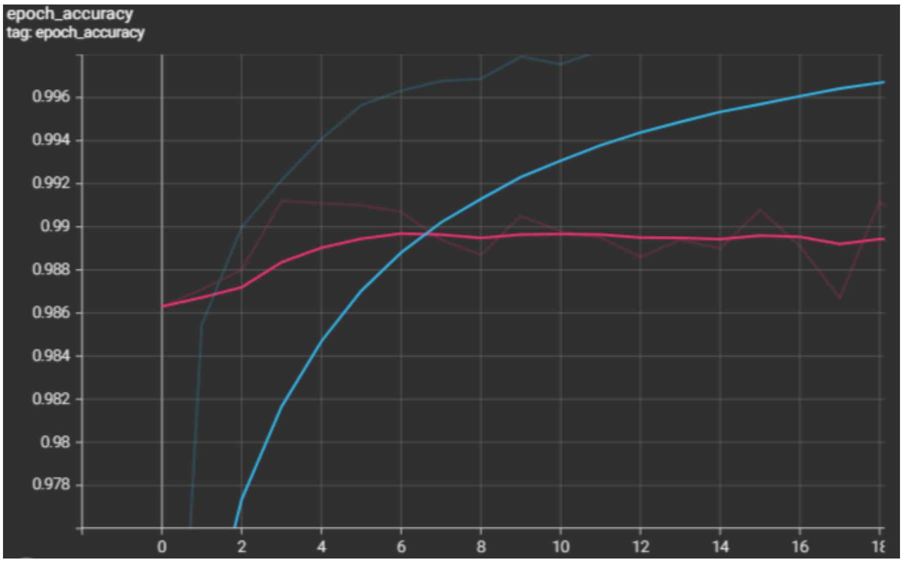
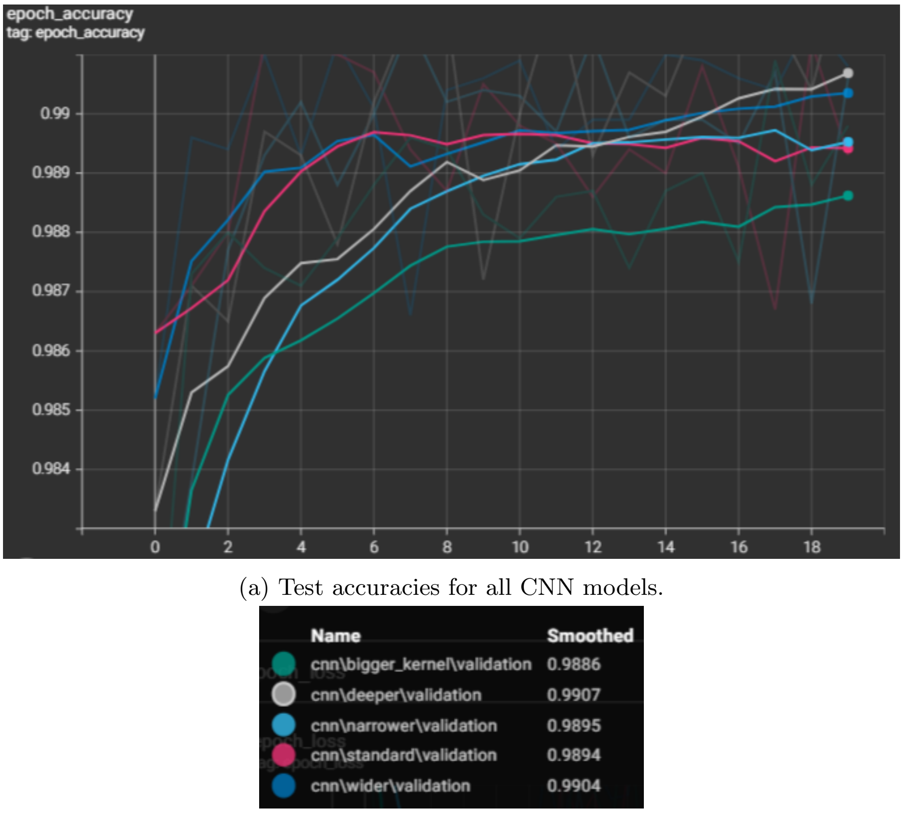


Figure 14: Train (cyan) and test (pink) accuracy at each epoch.

esting that more iterations could further benefit them. However this is also true for the worse (bigger kernel) one. The narrower model seems to perform as well as the standard one. In general the difference between the best and worse models is 0.2% and some noise exists in these results as the experiment was only repeated once. Hence the safer conclusions are deeper and wider are better than bigger kernel for similar number of epochs. However, given more time, more models would be allowd the reach their full potential so the results could be different then. Also, the number of trainable parameters is vastly different between architectures so a performance/parameter graph must also be considered.



(a) Test accuracies for all CNN models.

	Name	Smoothed
●	cnn\bigger_kernel\validation	0.9886
●	cnn\deeper\validation	0.9907
●	cnn\narrower\validation	0.9895
●	cnn\standard\validation	0.9894
●	cnn\wider\validation	0.9904

(b) Legand.

Figure 15: Test accuracies of all CNN models. Smoothing is set to 0.9.

To calculate the number of parameters in a CNN, we have that each $n \cdot n \cdot k$ kernel has

$$n \cdot n \cdot k + 1 = n^2 \cdot k + 1.$$

parameters, where k is the depth of the layer and we take into account the bias.

Hence, for the 32 filters of the first layer we have

$$(4 \cdot 4 \cdot 1) \cdot 32 + 32 = 544.$$

parameters. For the following layers, we need to adjust for the different depths which are now 32 instead of 1 for input and 64 instead of 32 for output:

$$(4 \cdot 4 \cdot 32) \cdot 64 + 64 = 32,832.$$

With similar logic we get

$$131,200$$

for the last convolutional layer.

The flatten layer has no parameters. So far we have ignored the stride and padding (which is zero) as they only affect the dense layer. In the general case, the size of the input after a convolutional layer becomes:

$$\text{Output size} = \frac{\text{Input size} - \text{Kernel size} + 2 \cdot \text{Padding}}{\text{Stride}} + 1$$

so for the stride of 1 of the first layer we get

$$\text{First layer output} = \frac{28 - 4 + 2 \cdot 0}{1} + 1 = 25.$$

This becomes

$$\text{Second layer output} = \frac{25 - 4 + 2 \cdot 0}{2} + 1 = 11.$$

as divisions are rouded down. Finally we get:

$$\text{Third layer output} = \frac{11 - 4 + 2 \cdot 0}{2} + 1 = 4.$$

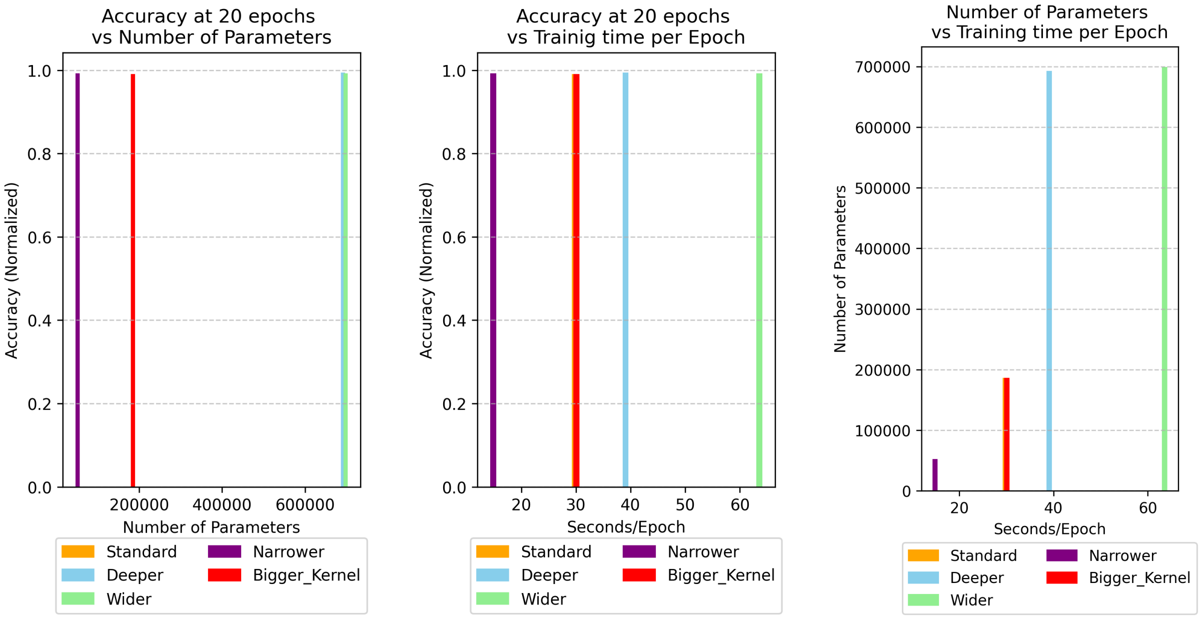
hence the flat layer produces

$$(4 \cdot 4 \cdot 128) = 2,048 \text{ values.}$$

Now, the dense layer calculates 10 parameters for each, plus 10 biases for each neuron, which results in

$$2,048 \cdot 10 + 10 = 20,490 \text{ parameters.}$$

With similar logic we can calculate the number of parameters of all variations, or simply extract them with the provided method. Ploting the Accuracy over Number of Parameters we get Figure 16. On (a) we see that, as expected, the higher performing models posses more parameters. (b) shows that better accuracy is achieved with more trainig time, but the relation is not linear and the tradeoff is questionable when 0.1% improvement requires more that twice the training time. From (c) we see that surprisingly, trainig time is not linear with number of parameters. This is because the wider one has more filters, all of which are processed for each value of each filter.



(a) Accuracy vs Parameters (Standard is behind Bigger Kernel)

(b) Accuracy vs Training Time

(c) Parameters vs Training Time

Figure 16: Comparison of Accuracy, Number of Parameters and Training Time

Compared to MLPs, we see that the performance is better with way less parameters but more training time, as convolutions and back propagation accross many filters and windows are more compute intensive that matrix multiplications

6 Visualising CNN outcomes (10 points)

7 Template Section

7.1 Template Subsection

7.1.1 Template SubSubsection

8 References

Khee, L. K., 2022. *Limitation of Data Visualisation* [Online]. Available from https://medium.com/@e0673935/limitation-of-data-visualization-1e4db068a22a [Accessed 13 October 2024].

Mackiewicz, A., Ratajczak, W., 1993. Principal Components Analysis. *Computers & Geosciences*, 19(3), 303-342.

Shahapure, K. R., Nicholas, C., 2020. Cluster Quality Analysis Using Silhouette Score. *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, 06-09 October 2020, Sydnew, NSW, Australia. pp 747-748.

Dubey, A., 2018. *The mathematics Behind Principal Component Analysis* [Online]. Medium. Available from: https://towardsdatascience.com/the-mathematics-behind-principal-component-analysis-fff2d7f4b643 [Accessed 15 October 2024]

Queensland Brain Institute. *How do neurons work?* [Online]. The University of Queensland. Available from: https://qbi.uq.edu.au/brain-basics/brain/brain-physiology/how-do-neurons-work

Rosenblatt, F., 1958. The Perceptron: A Probabilistic Model for Information Storage and Organisation in the Brain. *Psychological Review*, 65(6), pp.386-408.

Rosenblatt, F., 1958. *The Perceptron: A Theory of Statistical Separability in Cognitive Systems.* Cornel Auronautical Labolatory.

Pedder, C., 2023, *Large language models & emergence* [Online], LinkedIn. Available from: https://www.linkedin.com/pulse/what-emergence-neural-networks-chris-pedder-phd/

Choi, R. C., Coyner, A. S., Kalpathy-Cramer, J., Chiang M. F., Campbell J. P., 2020. Introduction to Machine Learning, Neural Networks, and Deep Learning. *Translational Vision Science & Technology*, 27;9(2), pp.14.

Bengio, Y., 2012. Practical Recommendations for Gradient-Based Training of Deep Architectures. In: G. Montavon, G. B. Orr, K. R. Muller, eds. *Neural Networks: Tricks of the Trade.* Second Edition. Springer.

Smith, L. N., Topin, N., 2017. Exploring Loss Function Topology with Cyclical Learning Rates. *International Conference on Learning Representations*, 14 Febuary 2017, arXiv.org.