# DEEP LEARNING ASSIGNMENT 1

530653709, 540771859

Benjamin Archer Rajiv Mehta

# 1 INTRODUCTION

Recent technological advancements such as the creation of ChatGPT by OpenAI and overall advancements in the use of machine learning as seen with changes to medical image segmentation through the use of deep learning, have resulted in a shift in market demand to new and innovative methods of analysis and prediction modeling through deep learning techniques [1] . Though as with the large strides in this field that have already been made, many multi-layer neural networks have been created into libraries such as Tensorflow and Pytorch which can be easily applied to most datasets. With these implementations a drawback arises, the understanding and overall control of how the neural network is operating can lead to serious issues down the track such as overfitting of the model which can cause data scientists and businesses to make poor predictions and business decisions.

This study aims to explore how a multilayer neural network is implemented from scratch and how different components such as Rectified Linear Unit (ReLu) and Dropout are implemented and can affect the outputs and predictions of the model. The study also aims to then choose the best combination of components to have the most accurate predictions. Through these goals, readers will be able to gain an understanding of what each component does and the basic principles which will then allow future data-scientists to learn how to create and optimize their models in both written code and understanding.

# 2 METHODS

## 2.1 CODE

https://drive.google.com/drive/folders/11W2svlxt7c4xct1FbxCFTk5SlNV88yj8?usp=sharing

## 2.2 PRE-PROCESSING

Data pre-processing is an essential step in the data analysis and prediction pipeline which involves techniques to prepare data by checking for null values and transforming the data to make sure each factor is normalised. It can also include the reduce the dimensionality of the data, improving the quality and consistency of the machine learning models. For this study, as the dataset was provided by the University of Sydney, the data was assumed to be pre-processed before given to students. Nonetheless, before acting on this, the team first double checked for any null values and removed them from the dataset to validate a clean dataset. As the context for the data was unknown no transformation and dimension reduction was performed on the dataset.

## 2.3 HIDDEN LAYERS

Hidden layers in neural networks are an essential component to any deep learning model. These layers are not inputs or outputs of the model, but rather refers to an extra layer of neurons that sits within the model, transforming inputs by applying weights and passing them through activation functions to learn non-linear relationships. Having multiple hidden layers in deep

learning models allows the models to analyse complex relationships within the training data though, having too many hidden layers can cause some drawbacks. The first main issue is that the number of layers is directly proportional to the amount of training data. If there is not enough data, then the training, validation and testing split might cause the model to output in accurate results. This then leads to the second issue as it is then possible to overfit the model with the additional training data, meaning that the model might begin to stop noticing trends in the data, reducing the prediction accuracy.

## 2.4 Rectified Linear Unit (ReLu)

To mitigate the overfitting issue caused by having multiple hidden layers, the ReLu activation function is used to store the maximum positive weighted value of the input. If there is no maximum value which is greater then or equal to zero, then the value passed to the next layer is 0. Therefore, when passed a 0 value, the model will no longer consider those specific parameters as significant in the model and extracts only the important features and patterns in the input data.

$$\text{Relu Activation: } f(x) = \begin{cases} 0 \text{ for x} < 0 \\ \text{x for x} \geq 0 \end{cases}$$

## 2.5 GELU (Gaussian Error Linear Unit)

To enhance the capability of our model to capture complex patterns we used the GELU activation function. It uses a weighted combination of the input values, applying a stochastic regularisation effect based on the Guassian distribution. This probabilistic approach allows GELU to retain some gradient information even when the inputs are negative or near zero, unlike ReLU which entirely nullifies these inputs. As a result GELU has the potential to help in the better integration of uncertainty in the data. [2]

## 2.6 Weight Decay

To further mitigate overfitting in the model, L2 Regularisation also known as Weight Decay can be implemented. Weight decay adds an additional term in the loss function during the training step which is used to limit the maximum weight value that can be taken by the model between layers. This encourages the model to reduce the number of large values which could cause the gradient to become exponentially larger causing instability within the model. Though, with the implementation of weight decay, the overall run-time of the model increases as adding the additional term slows down the learning rate and requires more iterations to get closer to the optimal point on gradient descent.

## 2.7 Momentum in Stochastic Gradient Descent (SGD)

Though, with the reduction in the learning rate caused by the weight decay function, it is important to also try and optimise the processing speed of the model. To do this, momentum can be introduced into SGD by adding a velocity factor into the equation. This velocity term allows the model to take bigger jumps on the gradient descent by using the average of the previous updates. With these bigger jumps, the number of descents steps the model has to take is reduced and is more linear between points. Though, with these larger jumps and extra parameters, the implementation of momentum can cause the model to overshoot the optimal

gradient leading a convergence from forward and back propagation in an suboptimal location. This also means that it is sensitive to hyper-parameters and can take a large amount of time and resources to get correct plus if the coefficient becomes too large can lead to memory issues dependent on available hardware.

$$\text{Momentum implementation: } W_{t+1} = W_t + \rho V_t - \eta g(W_t)$$

## 2.8 DROPOUT

Dropout is another regularisation technique that could be implemented to help reduce overfitting within the model, especially those with large or continuous datasets. During the training step, dropout activation removes nodes in each layer based on either a specified or random chance, usually between 0.2-0.5. This comes with two main advantages, the first being that as neurons are randomly dropped at each iteration, the model does not become overly reliant on specific nodes based on inputs reducing overfitting of the model. The second advantage is because nodes are being removed, the model is pushed to learn patterns and trends rather than direct input equals a certain output, making the model robust to unseen data. Though, as nodes are being removed at each iteration, more iterations of the model need to occur for training to make sure the scope of the trends are seen which could increase the overall processing time of the model.

## 2.9 SOFTMAX AND CROSS-ENTROPY LOSS

Softmax is an activation function commonly used in the output layer of neural networks for multi-class classification problems. It transforms the raw output scores of the network into a probability distribution that sums to one. This allows the network to return interpretable outputs that represent the likelihood of each class. Crossentropy loss provides a measure of the error between the predicted probability distribution and the true label distribution. By minimising the cross-entropy loss during training the network learns to produce probability distributions that closely match the true label distribution.

## 2.10 MINI-BATCH TRAINING

Mini-batch training is a technique for optimising neural network performance by dividing the dataset into smaller subsets called mini-batches.Mini-batch training improves memory efficiency by processing smaller subsets making it beneficial for large, high-dimensional data and neural networks with many parameters. Mini-batch training also provides a balance between convergence speed and weight update stability by averaging gradients over each mini-batch. However mini-batch training can also lead to noisy updates and slow convergence if the batch size is too small and high memory usage if the batch size is too large. The batch size for mini-batch training is often between 16 and 128 with 32 being a common default.

## 2.11 BATCH NORMALISATION

Batch Normalisation is a technique used to improve the training speed and stability of neural networks by normalising the inputs to each layer. Every layer of a neural network transforms its input data as it learns, potentially leading to a significant shift in the distribution of the data as it flows through the network. This shift can slow down the training process significantly because

each layer must adapt to a new data distribution at every training step. By applying batch normalisation the inputs to each layer are standardised to have a mean of zero and variance of one. This accelerates the training by reducing the number of epochs needed to train but also stabilises the learning process across different layers by maintaining a consistent distribution of input values. Our implementation makes a separate BatchNormalization class in which the batch normalisation backward and forward pass are calculated. These functions are called from the backward pass in the Hidden Layer.

# 3 EXPERIMENT/RESULTS

## 3.1 DEFAULT MODEL PERFORMANCE

Once the model was programmed we created a default model based on standard values for each of the hyperparameters. Here is the structure and hyperparameter values of the default model:

- # of Hidden Layers = 2
- Hidden Layer Node Count = 64, 64
- Hidden Layer Activations = GELU, ReLU
- Learning Rate = 0.001
- Batch Size = 32
- Weight Decay = 0.001
- Momentum = 0.9
- Dropout Probability = 0.5
- Batch Norm = True

When the model was run the validation accuracy was 40.74% which is not good but it shows that it is learning significantly better than random chance (10%). The training and validation loss graph shows that after around 50 epochs there are negligible improvements in training loss while validation loss actually appears to go up indicating likely overfitting.

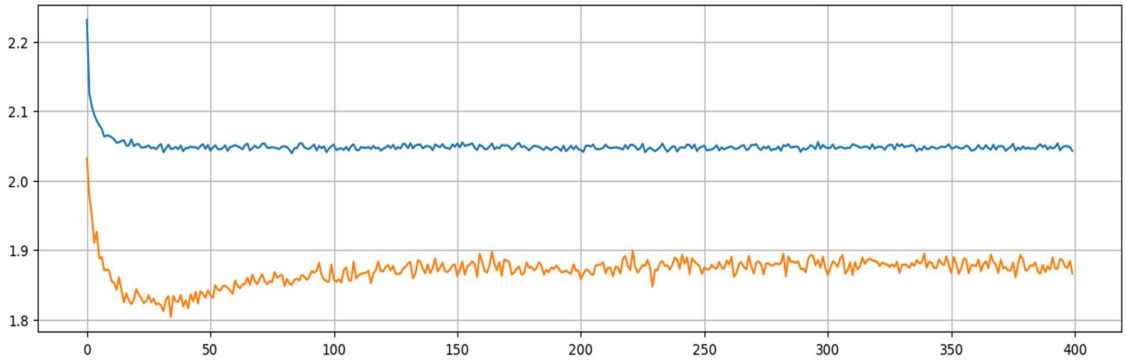


*Figure 1: Training Loss (Blue) and Validation Loss (Orange) over 400 epoch*

## 3.2 HYPERPARAMETER TUNING

For hyperparameter tuning we decided to go with One-at-a-time Tuning for its simplicity and low computational requirements. This method consists of varying one model parameter at a time while keeping all other parameters constant. While it is fast and efficient it has the drawback of overlooking the interactions between parameters. We compared the different hyperparameter tunings on the validation accuracy with the lowest accuracy being 10.33% and the highest accuracy being 50.50% when the dropout probability was lowered to 0.2.

When tuning learning rate there was an interesting effect where the accuracies increase as the learning rate gets smaller until it gets to 0.00001 at which point the accuracy takes a major decline. There was very little difference in the accuracies of the models with different weight decay values. This could be for a number of reasons. There could be a problem with the implementation causing it to have a smaller effect than it should. There could be different combinations of hyperparameters that make weight decay more consequential than is shown here. Differences in batch sizes were also very small which could be due to the same reasons. In this case it could also be because the effect of batch normalisation makes the differences in batch size less consequential. It could also be because we did not try more extreme batch size values. Tuning the momentum value was more effective at lowering the validation accuracy albeit only slightly. The accuracy went up as the momentum value went down with a peak of 41.08% which is less than 1% better than the default model. Tuning the dropout probability had a larger effect on the accuracy with a difference of more than 10% between the worst performing (60% dropout) and best performing (20% dropout).

When batch normalisation is on the default model gives a modest validation accuracy of 40.72%. However when it is turned off the model gives a validation accuracy of 10.33% which means that it is not learning and is randomly guessing output values. The reason for this is almost certainly faulty implementation which we were unable to rectify. Given the mini-batch training the model is likely performing better with batch normalisation than it would've without, although we cannot examine the metrics at this time. Performance on the two activation functions we implemented show that overall ReLU activation worked better with this model than did GELU by about 7%. This could be because GELU is most suited for NLP while ReLU is useful for simpler models and is more general purpose. Interestingly when one hidden layer used GELU activation and the other used ReLU activation the model performs better when ReLU activation is before the GELU activation by about 5%.

The layer and node counts are a special case because these values have large effects on the runtime and overall computational load of the model. They can also have significant effects on the degree of overfitting of the model which can have serious consequences. Three node counts were tested with the validation accuracy increasing as the node count increased. However the spread between the lowest validation accuracy (39.93%) and the highest (43.40%) is less than 4% indicating low significance. The increase in computing time, however, is considerable. Similarly adding hidden layers to the network has a notable effect on the validation accuracy with one layer having the lowest (33.06%) and three layers having the highest (41.15%). As with increased node counts the increase in layers comes at the expense of longer run times and heavier resource usage with the largest model having a run time of more than triple the smallest model.

Below are the validation results for each of the hyperparameters that were tuned. The results from the default model are included:

*Table 1. Factors Validation Accuracy*

| Factor and Validation | | | | | |
|---|---|---|---|---|---|
| **Momentum** | 0.85 | 0.9 | 0.95 | 0.99 | |
| Validation Accuracy | 41.08% | 40.74% | 40.43% | 40.35% | |
| **Dropout Probability** | 0 | 0.2 | 0.3 | 0.5 | 0.6 |
| Validation Accuracy | 50.21% | 50.50% | 46.29% | 40.74% | 39.10% |
| **Batch Sizes** | 16 | 32 | 64 | 128 | |
| Validation Accuracy | 40.72% | 40.71% | 40.74% | 40.64% | |
| **Node Counts** | [64,32] | [64,64] | [128,64] | | |
| Validation Accuracy | 39.93% | 40.74% | 43.50% | | |
| **Activations** | [gelu,gelu] | [relu,gelu] | [gelu,relu] | [relu,relu] | |
| Validation Accuracy | 39.75% | 44.29% | 40.74% | 47.02% | |
| **Batch Normalisation** | TRUE | FALSE | | | |
| Validation Accuracy | 40.72% | 10.33% | | | |
| **Layer Count** | One Hidden Layer | Two Hidden Layers | Three Hidden Layers | | |
| Validation Accuracy | 33.06% | 40.74% | 41.15% | | |

## 3.3 JUSTIFICATION OF FINAL MODEL

The final model was chosen based on a number of factors. Most of the hyperparameter values were chosen because they had the highest performance on validation accuracy including learning rate, dropout probability, momentum, node count and activation type orientation. The first exception we made was with weight decay which had negligible differences in accuracy between values. Because the results were so close we decided to keep the default value of 0.001 in case it had a stronger effect in interaction with non-default values in the final model. Likewise, the variation in validation accuracy as a result of changes in batch size were so small that we decided to keep the default value. Another exception we made was to build our model with two hidden layers instead of three despite its better performance. This is because the model with three hidden layers took approximately twice as long to run as the model with two hidden layers and resulted in less than 1% better performance in terms of validation accuracy. The final model took approximately 30 minutes to run on Google Colab. Here are the specifications for the final model:

- # of Hidden Layers = 2
- Hidden Layer Node Count = 128, 64
- Hidden Layer Activations = ReLU, ReLU
- Learning Rate = 0.0001

- Batch Size = 32
- Weight Decay = 0.001
- Momentum = 0.85
- Dropout Probability = 0.2
- Batch Norm = True

## 3.4 Test Results

The tuned final model has a test accuracy of 53.89% which is significantly higher than random guessing (10%). Unlike in the default model the training and validation loss graph for the final tuned model does not show the same increase in validation loss after a certain amount of epochs. This may mean that the model is less prone to overfitting than the default model. This graph also shows that there is negligible decrease in loss after epoch 200 which results in a large amount of computing time effectively wasted.

The Classification Report in Figure 3 shows the overall performance of the model is acceptable but has serious room for improvement. The macro average for precision, recall and f1-score are 0.53, 0.54 and 0.54 respectively which are all significantly better than random guessing indicating that the model has learned. Although there was no class imbalance, the model performed worse on certain labels like 2 and 3 and better on labels like 1 and 8. The lowest values for precision, recall and f1-score were 0.41, 0.33 and 0.37 respectively indicating low predictive performance on these classes. The highest values for precision, recall and f1-score were 0.62, 0.72 and 0.68 respectively, still indicating moderate performance.

The Confusion Matrix in Figure 4 shows the confusion matrix for the test set predictions. The model shows significantly different performances for different classes. The 9 class for example was only mistaken for the 4 class 4 times while the 3 class was mistaken for the 5 class 202 times. Figure 5 shows actual predictions and labels from the code.

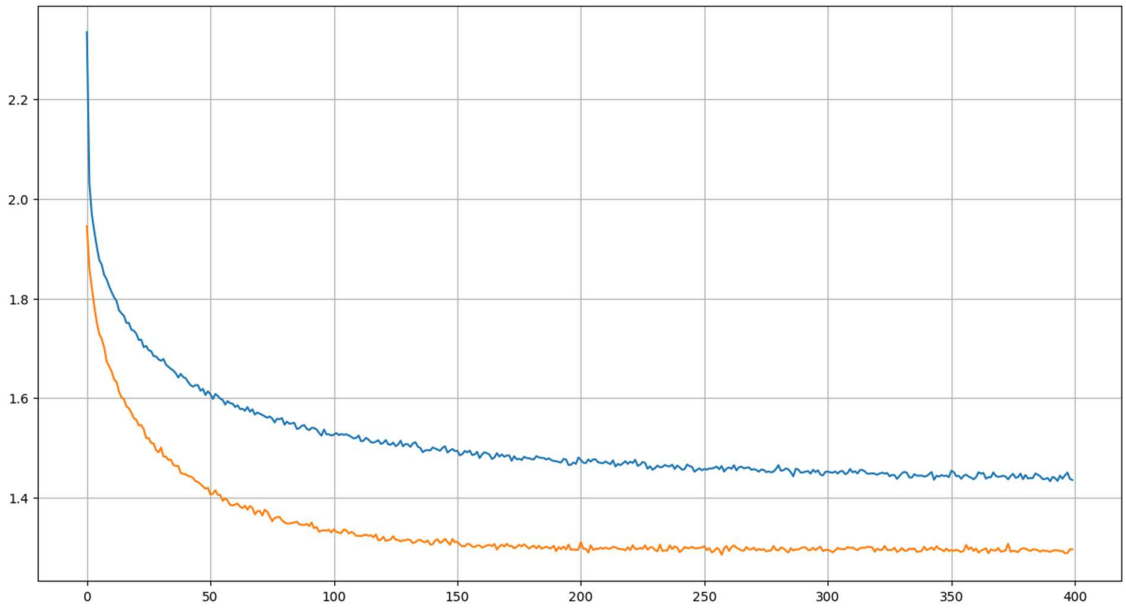

*Figure 2: Training Loss (Blue) and Validation Loss (Orange) over 400 epochs (x-axis)*

*Table 2. Classification Report for test set predictions*

| | Precision | Recall | f1-Score |
|---|---|---|---|
| 0 | 0.62 | 0.57 | 0.60 |
| 1 | 0.62 | 0.69 | 0.65 |
| 2 | 0.45 | 0.44 | 0.45 |
| 3 | 0.41 | 0.33 | 0.37 |
| 4 | 0.44 | 0.46 | 0.45 |
| 5 | 0.47 | 0.44 | 0.46 |
| 6 | 0.61 | 0.61 | 0.61 |
| 7 | 0.61 | 0.61 | 0.61 |
| 8 | 0.65 | 0.72 | 0.68 |
| 9 | 0.56 | 0.63 | 0.59 |
| Macro Average | 0.53 | 0.54 | 0.54 |
| Weighted Average | 0.53 | 0.54 | 0.54 |

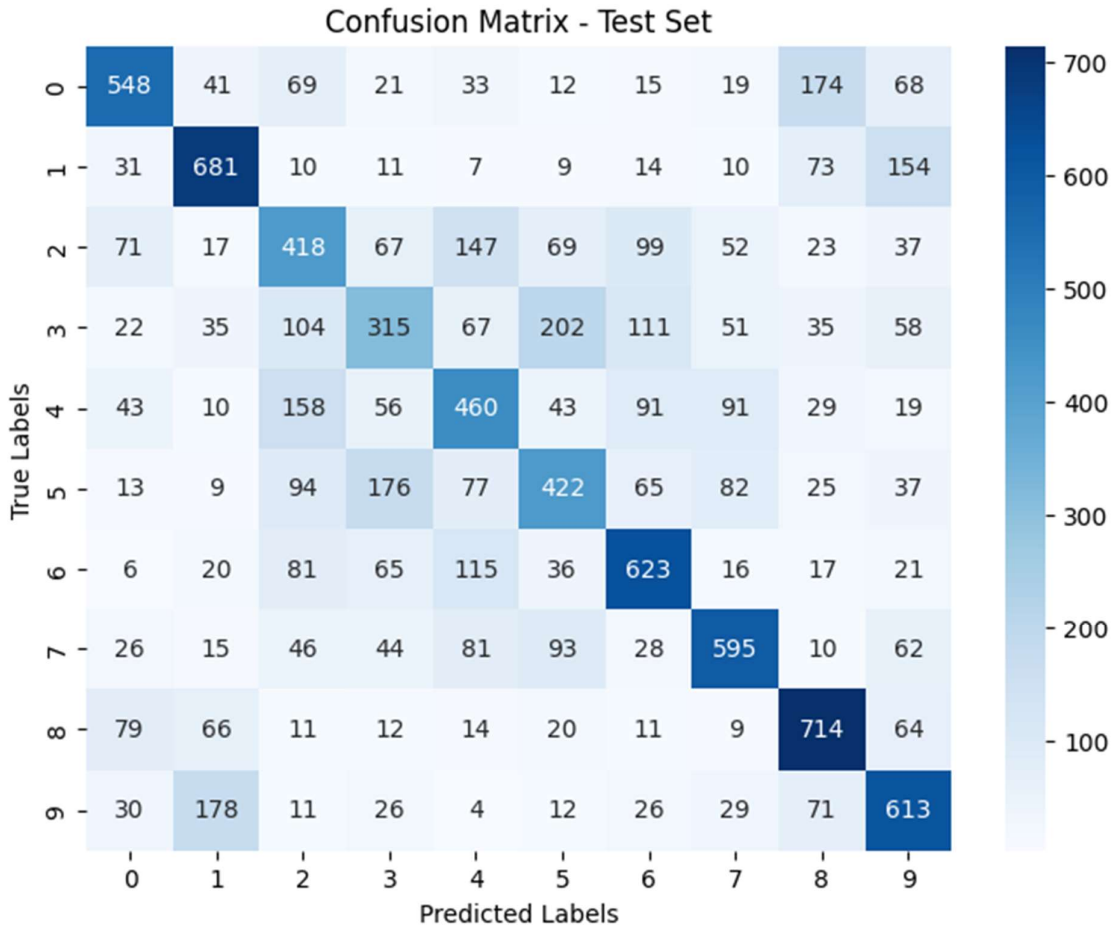


*Figure 3: Confusion Matrix for the test set predictions*

```
Predicted: [7.], Actual: [7]
Predicted: [9.], Actual: [0]
Predicted: [5.], Actual: [3]
Predicted: [7.], Actual: [5]
Predicted: [3.], Actual: [3]
Predicted: [8.], Actual: [8]
Predicted: [5.], Actual: [3]
Predicted: [3.], Actual: [5]
Predicted: [2.], Actual: [1]
Predicted: [7.], Actual: [7]
```

*Figure 4: Last 10 predictions and actual labels*

# 4 Discussion and Conclusion

## 4.1 Discussion

This study aimed to explore the implementation of a multilayer neural network from scratch and investigate the impact of different components on the model's performance. The default model, based on standard hyperparameter values, achieved a validation accuracy of 40.74%, indicating that it learned significantly better than random chance (10%). However, the training and validation loss graph showed that after around 50 epochs, there were negligible improvements in training loss, while validation loss increased, suggesting overfitting. Hyperparameter tuning was performed using the One-at-a-time Tuning method, which varied one model parameter at a time while keeping all others constant. The results showed that tuning the learning rate, weight decay, and batch size had minimal impact on the validation accuracy. Batch normalisation proved to be crucial for the model's performance. When turned off, the model's validation accuracy dropped to 10.33%, indicating that it was not learning and was randomly guessing output values. This could be attributed to faulty implementation, which we were unable to rectify. The final model was chosen based on the best performing hyperparameter values, with some exceptions made for weight decay and batch size due to their negligible impact on accuracy. We also opted for two hidden layers instead of three to balance performance and computing cost.

## 4.2 Conclusion and Reflection

This study demonstrates the importance of understanding the components and hyperparameters of a multilayer neural network when implementing it from scratch. By exploring the impact of different components, it was possible to enhance the model's validation accuracy from 40.74% to 50.50%. Although the final model is significantly better than random guessing, it still has moderate performance on certain classes, as shown by the Classification Report. This suggests there is potential for further improvement, possibly through the use of more advanced techniques like early stopping and hyperparameter grid search. This study also highlights the trade-offs between model complexity, performance and computational efficiency. With increases in the number of hidden layers and nodes the runtime and resource usage increase with the validation score necessitating a tradeoff. The insights gained from this study not only shed light on the importance of hyperparameter tuning but also emphasise the need for a balanced approach to model complexity and computational efficiency.

Once the model was programmed we created a default model based on standard values for each of the hyperparameters. Here is the structure and hyperparameter values of the default model:

- # of Hidden Layers = 2
- Hidden Layer Node Count = 64, 64
- Hidden Layer Activations = GELU, ReLU
- Learning Rate = 0.001
- Batch Size = 32
- Weight Decay = 0.001
- Momentum = 0.9
- Dropout Probability = 0.5
- Batch Norm = True

# 5 REFERENCES

[1] C. G. A.-M. V. S. V. J. D. V. P., “Current and Emerging Trends in Medical Image Segmentation With Deep Learning.,” IEEE transactions on radiation and plasma medical sciences, vol. 7, pp. 545-569, 2023.

[2] Glorot, X., Bordes, A., & Bengio, Y. (2011, June). Deep sparse rectifier neural networks. In Proceedings of the fourteenth international conference on artificial intelligence and statistics (pp. 315-323). JMLR Workshop and Conference Proceedings.

[3] Hendrycks, D., & Gimpel, K. (2016). Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415.

[4] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

[5] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.