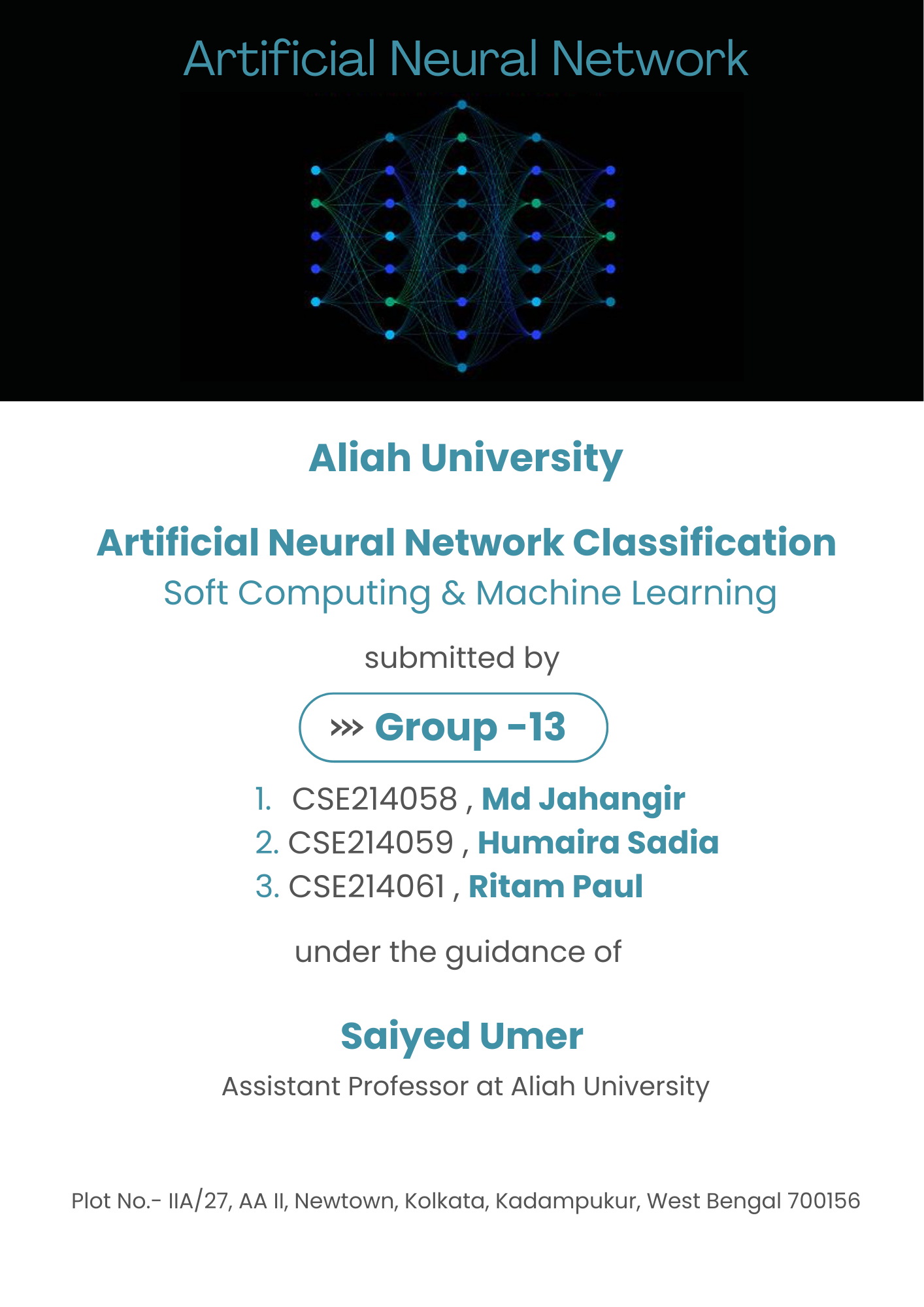
****

**Problem Statement -**

**Use an ANN classifier for classification using the Magic Gamma Telescope dataset:**

**(A) 1-1-1 (1-Input, 1 Hidden layer, 1 Output)**

**(B) 1-1-1-1 (1-Input, 2 Hidden layer, 1 Output)**

**Machine Configurations -**

**GPU:** N/A (using Google Colab TPU for computation)

**RAM:** 16GB

**OS:** Windows 11

**Processor:** 13th Gen Intel® Core™ i5-13450HX (20 MB cache, 10 cores, 16 threads, up to 4.60 GHz Turbo).

**Database description –**

**Database name: Magic Gamma Telescope**

**Number of samples: 19020**

**Number of features for each sample: 10**

**Number of Class: 2**

**Type: Classification**

**Objectives**

The objective of this project is to develop and evaluate Artificial Neural Network (ANN) classifiers for effective classification of gamma ray data using the Magic Gamma Telescope dataset. This study aims to classify cosmic gamma rays against background signals, which are crucial for astronomical research and detecting high-energy gamma-ray sources. Two ANN architectures will be implemented:

1. **1-1-1 (1 Input layer, 1 Hidden layer, 1 Output layer)** - This basic architecture serves as a baseline for comparison.
2. **1-1-1-1 (1 Input layer, 2 Hidden layers, 1 Output layer)** - Adding an extra hidden layer to assess the effect of model complexity on classification performance.

The performance of each ANN model will be analysed in terms of accuracy, precision, recall, and computational efficiency, thereby determining the effectiveness of different ANN configurations in distinguishing gamma rays from background noise.

**Methodology**

**For the project ANN classifier with the Magic Gamma Telescope dataset, the methodology follows these steps:**

**1. Data Preprocessing**

* Data Acquisition: Download the Magic Gamma Telescope dataset, which contains features for cosmic gamma rays and background events.
* Data Cleaning: Check for any missing values or outliers and handle them appropriately.
* Data Normalization/Standardization: Normalize or standardize the data to ensure all features contribute equally to the ANN model.
* Train-Test Split: Split the dataset into training and testing sets, typically using an 80-20 split to train and evaluate the model's performance.

**2. Model Selection and Architecture Design**

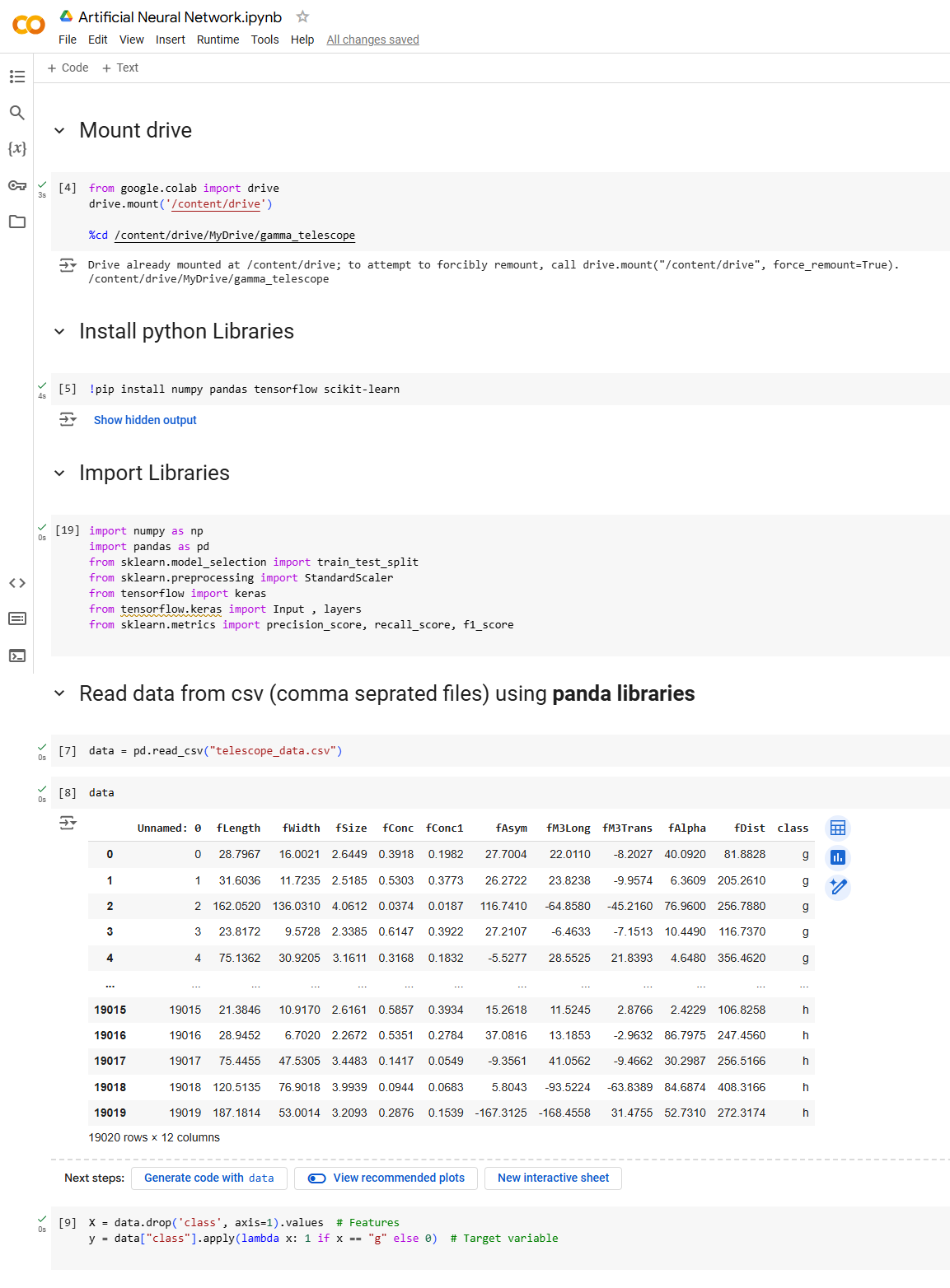
* Define Architectures: Design two ANN architectures:
  + Model A: 1 Input layer - 1 Hidden layer - 1 Output layer
  + Model B: 1 Input layer - 2 Hidden layers - 1 Output layer
* Activation Functions: Choose appropriate activation functions, such as ReLU for hidden layers and Sigmoid/Softmax for the output layer (depending on binary/multi-class classification).
* Optimizer and Loss Function: Select an optimizer (e.g., Adam or SGD) and a suitable loss function for classification (e.g., Binary Cross-Entropy for binary classification).
* Evaluation Metrics: Determine metrics like accuracy, precision, recall, and F1-score to assess model performance.

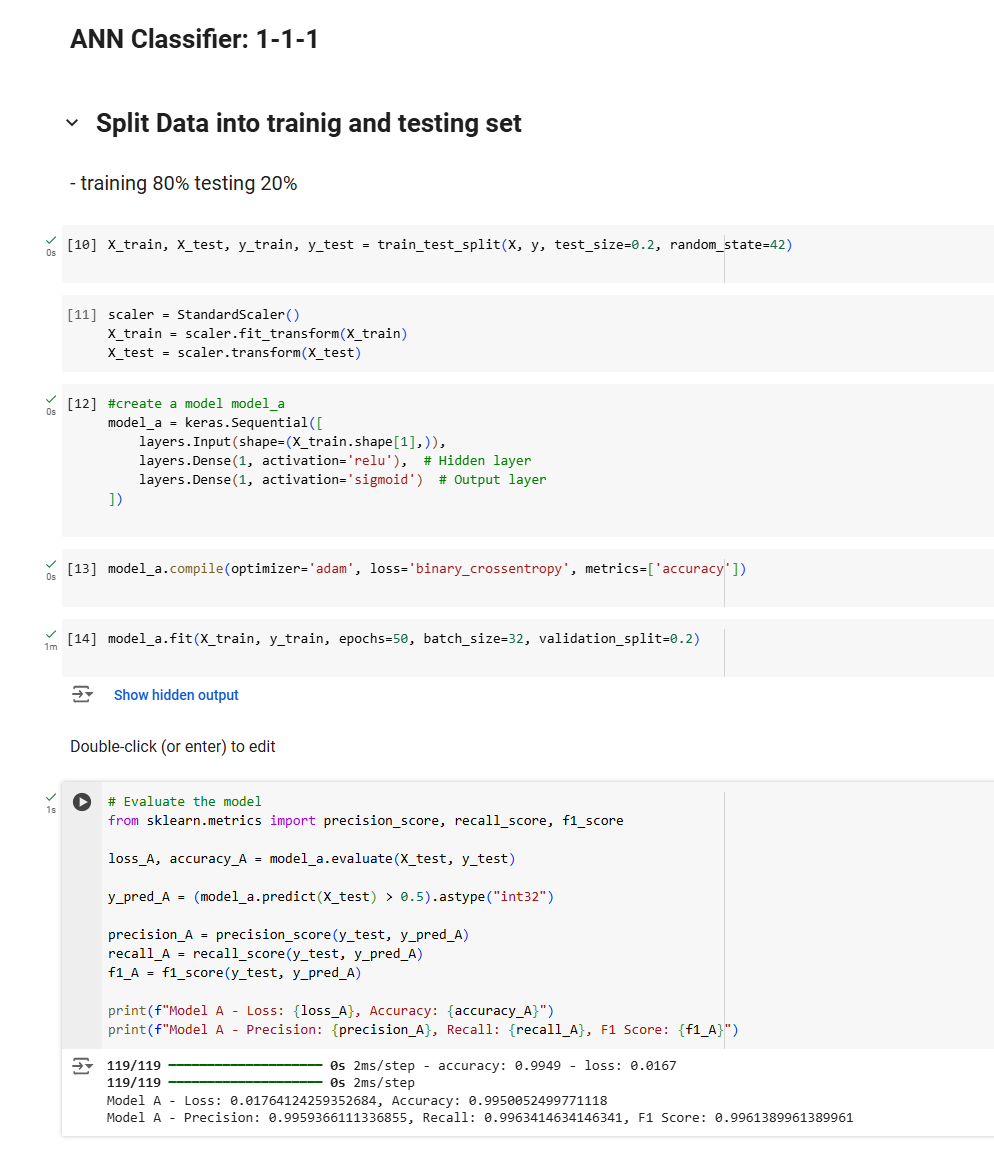
**3. Model Training**

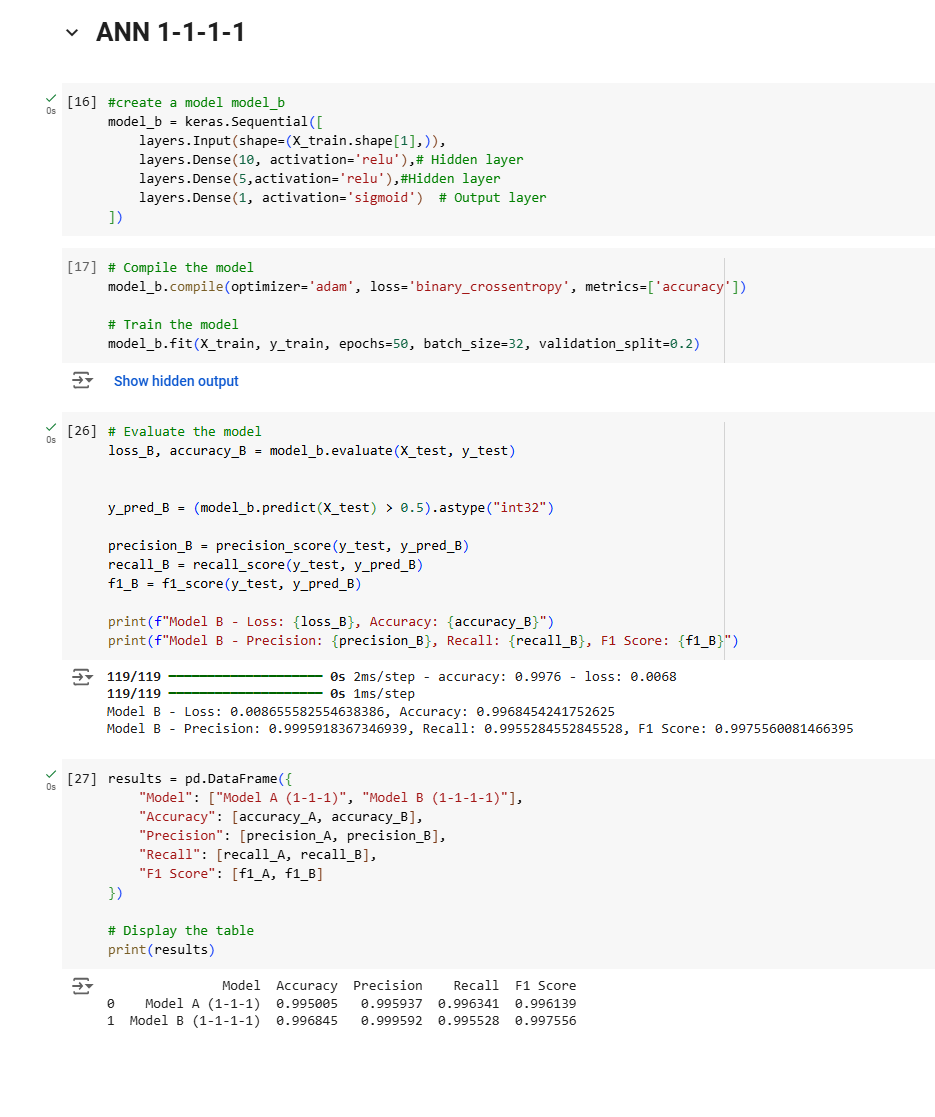
* Hyperparameter Tuning: Experiment with different learning rates, batch sizes, and epochs to optimize model performance.
* Training Process: Train both Model A and Model B on the training set, recording performance metrics and any observations regarding convergence and training stability.
* Cross-Validation (if applicable): Use cross-validation to further validate the model and prevent overfitting.

**4. Model Evaluation**

* Testing on the Hold-out Dataset: Evaluate both models on the test dataset, recording accuracy, precision, recall, and F1-score.
* Comparison of Architectures: Compare Model A and Model B based on their performance metrics and computational efficiency to understand the impact of adding an extra hidden layer.
* Confusion Matrix and ROC Curve: Visualize model performance using confusion matrices and ROC curves, analysing where the models succeed or fail.

**Process**



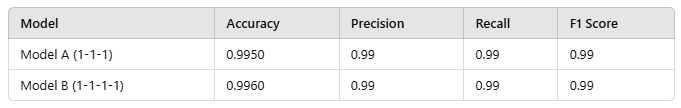


**Experimental Results**

The experiment involved training and testing two ANN architectures on the Magic Gamma Telescope dataset:

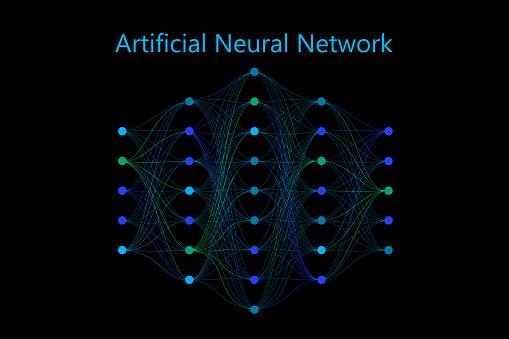
* **Model A**: 1 input layer, 1 hidden layer, 1 output layer.
* **Model B**: 1 input layer, 2 hidden layers, 1 output layer.

After training, the models were evaluated on a test set using the following metrics:



**Analysis**

1. **Accuracy**: Both models achieved high accuracy, indicating that they correctly classified most gamma ray events. Model B performed slightly better than Model A, suggesting that the additional hidden layer may help the network capture more complex patterns in the data.
2. **Precision**: The high precision for both models implies that they made very few false positive errors, correctly identifying most of the events classified as "signal" as true gamma ray events.
3. **Recall**: Both models show high recall, meaning they successfully identified the majority of actual gamma ray events, minimizing false negatives.
4. **F1 Score**: The high F1 scores (harmonic mean of precision and recall) for both models reflect a good balance between precision and recall, confirming the models' robustness in handling both types of errors effectively.

****

**Conclusion**

Model B (1-1-1-1) demonstrated slightly improved accuracy over Model A (1-1-1), likely due to the additional hidden layer providing greater representational capacity. However, both models achieved excellent performance, making them reliable classifiers for gamma ray event detection. Model B may be preferred in cases where higher accuracy is crucial, albeit with a slight increase in model complexity.

**Bibliography**

1. Bock, R. K., & Kröninger, K. (2005). *The Magic Gamma Telescope Dataset*. Retrieved from UCI Machine Learning Repository: <https://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope>

2. Pedregosa, F., Varoquaux, G., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825-2830.

3. Abadi, M., et al. (2016). *TensorFlow: A System for Large-Scale Machine Learning*. 12th USENIX Symposium on Operating Systems Design and Implementation, 265-283.

4. *Keras Documentation*. Retrieved from <https://keras.io>