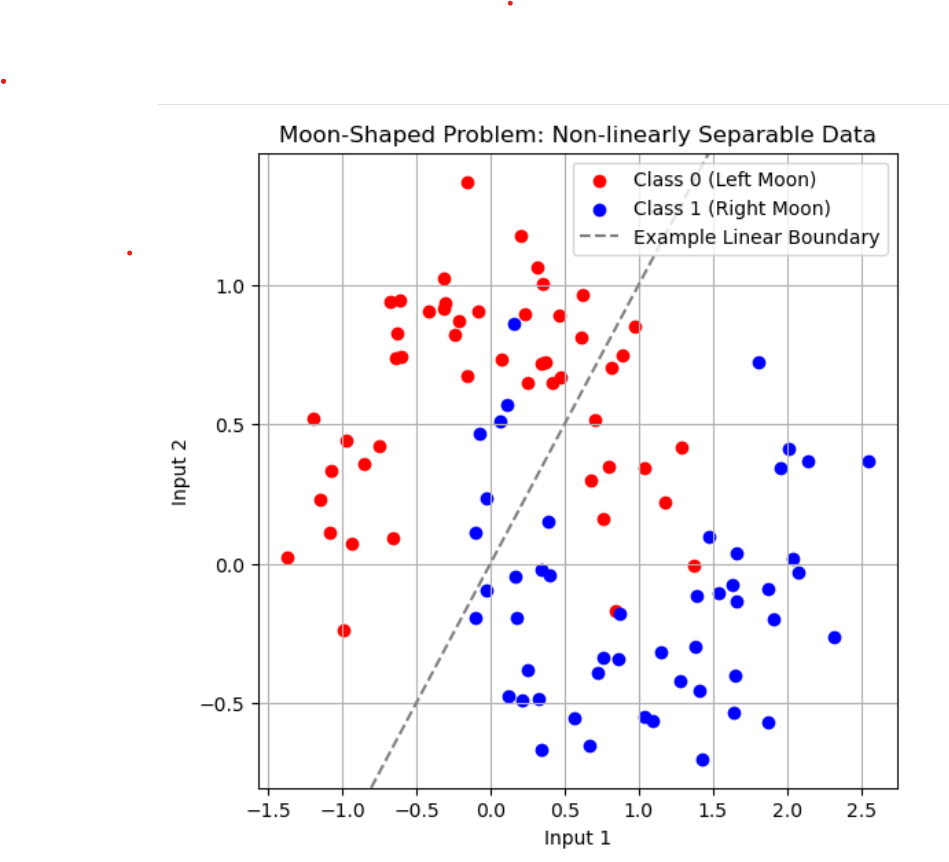
**Submitted by**: SRILEKHA EMMA

**Student Id**:23034668

ANNs are computational models inspired by the human brain. Each artificial neuron processes inputs, computes a weighted sum, applies an activation function, and produces an output. Simple models like perceptrons can solve linearly separable problems (e.g., AND) but fail with non-linear problems like moon perceptron, which necessitates advanced architectures such as Multilayer Perceptrons (MLPs).



**Artificial Neural Networks (ANNs)**

Artificial Neural Networks (ANNs) are computational models inspired by the neural networks found in the human brain. Each artificial neuron performs the following tasks:

1. Receives inputs.
2. Computes a weighted sum of these inputs.
3. Applies an activation function to produce an output.

The simplest form of an ANN is the **perceptron**, which can solve linearly separable problems like the AND gate. However, perceptrons struggle with non-linear problems, such as the **Moon Problem**, highlighting the need for more advanced architectures like **Multilayer Perceptrons (MLPs)**.

**Example: Moon Problem**

The **Moon Problem** is a well-known example that demonstrates the limitations of perceptrons. In the Moon problem, the data points are arranged in two interlocking crescent-shaped classes, which cannot be separated by a straight line. The data for the Moon problem consists of:

* **Inputs (X)**: Points that form two moons or crescent-shaped clusters.
* **Outputs (y)**: Labels indicating which moon each point belongs to (typically two classes).

A perceptron cannot solve the Moon problem because the data points are non-linearly separable. The two classes are intertwined in a way that a single straight line cannot separate them, which demonstrates the need for more sophisticated network architectures. This limitation of the perceptron motivates the use of **Multilayer Perceptrons (MLPs)**, which have hidden layers capable of transforming the data into a space where a linear decision boundary can be drawn.

**Visualization of the Moon Problem**

In a typical visualization of the Moon problem, class 0 (red) and class 1 (blue) points form two crescent shapes. A linear model, such as a straight line (illustrated as a dashed grey line), fails to separate the classes correctly because the moons are interwoven. This failure is due to the non-linear separability of the data, highlighting the limitation of a perceptron, which can only draw straight lines to separate data.

To address this, **Multilayer Perceptrons (MLPs)** are introduced. MLPs consist of hidden layers that can apply non-linear transformations to the data, allowing for the creation of a decision boundary that can correctly separate the moons.

**What is an MLP?**

A **Multilayer Perceptron (MLP)** is a type of Artificial Neural Network (ANN) used for supervised learning tasks, such as classification and regression. Unlike a simple perceptron, which has no hidden layers, an MLP consists of one or more hidden layers placed between the input and output layers. These hidden layers enable MLPs to handle complex, non-linearly separable data like the Moon problem, making them much more powerful and flexible than a simple perceptron.

**Key Characteristics of MLPs**

1.Feedforward Network: MLPs are feedforward neural networks, means the flow of information moves in one direction i.e. from the input layer to the output layer, without looping back.

2.Supervised Learning: MLPs require labelled data during training to learn the mapping between inputs and outputs.

3.Universal Function Approximator: With sufficient hidden layers and neurons, an MLP can approximate any continuous function. MLP Architecture: It is a type of feedforward artificial neural network (ANN) called as “Multilayer” since it contains several layers of neurons, stacked together.

The key components of an MLP are input layer, hidden layers, output layer.

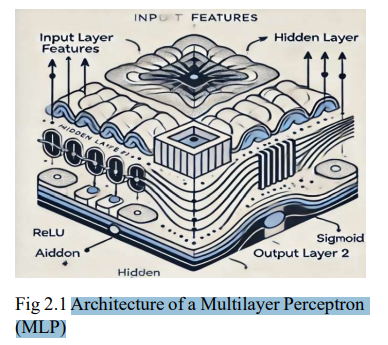
**Architecture of a Multilayer Perceptron (MLP)**

**Input Layer Role**:

This layer receives the raw data. It doesn’t process the data instead it passes it to the next layer. Structure: Each neuron in the input layer corresponds to a feature in the dataset. The raw data is passed to the first hidden layer for further processing. Example: For an image classification task (e.g., Fashion-MNIST dataset), the input layer would have one neuron per pixel in the image, meaning for a 28x28 image, there would be 784 neurons.

**Hidden Layers Role**: These layers where model learns from the data. They perform most of the computation in the network and their job is to transform the input data into a format that model can use to make the predictions. The more hidden layers you have, the more complex patterns the model can learn.

Structure: Each neuron in the hidden layers receives weighted inputs from the previous layer applies a bias and passes the result through an activation function.:



Mathematical Representation: For a neuron in the 𝑙 − 𝑡ℎhidden layer:

where: 𝑧() is the weighted sum of inputs,

𝑊() is the weight matrix,

𝑎( )is the activation of the previous layer,

𝑏() is the bias for layer 𝑙 ,

𝑓(𝑧() is the activation function.

Why Hidden Layers Matter: They allows network to capture non-linear relationships between inputs and outputs. Each hidden layer processes information in a way that creates a more abstract representation of the input.

**Output Layer Role:** Here in this layer network's predictions are made. This layer determines the outcome based on the learned weights.

Structure: For classification problems, the number of neurons in the output layer is equal to the number of classes. For binary classification, you may use a single neuron with a Sigmoid activation function. In multi-class classification, you’ll use one neuron per class, and typically for each class a SoftMax activation function is used to output probabilities. For regression problems, there is usually just one neuron that predicts a continuous value.

**Mathematical Representation**: For a neuron in the output layer: � �( ) is the weighted sum at the output layer, 𝑊( ) is the weight matrix, 𝑎( )is the activation of the previous layer, 𝑦 is the final output. For binary classification, the output might use the Sigmoid activation produces a value between 0 and 1 which represents the probability of a certain class.

For multi-class classification, SoftMax is typically used to ensure the output sums to 1, representing the class probabilities. Activation Functions in MLP Importance of Activation Functions Activation functions are essential because they adds non-linearity to the network, allows to model complex relationships. Without them, network would only be able to learn linear mappings, which is insufficient for most real world tasks. Sigmoid: Often used for binary classification, the sigmoid function squashes the output between 0 and 1, which makes useful for probability outputs.

**Tanh**: Purpose: Like sigmoid but with output values between -1 and 1, maybe suitable for tasks where the data has a centered distribution. Effect: Like sigmoid, but more powerful as it ensures the output is centered around 0.

Range: (-1, 1)

**ReLU (Rectified Linear Unit)**: One of the most used activation functions due to its simplicity and efficiency.

Formula: f(z)=max (0, z), Range: (0, ∞).

Purpose: Used in hidden layers of MLPs, ReLU is efficient and helps in mitigating the vanishing gradient problem. It outputs 0 for negative values and the input value for positive values, allows it computationally efficient.

**Effect**: ReLU adds non-linearity, allowing MLPs capturing the complex patterns in the data.

Visual Representation: Input: -5, -3, 0, 2, 5 → Output: 0, 0, 0, 2, 5

ReLU activation creates a threshold at 0, ensuring positive values pass through as they are, while negative values are clipped.

SoftMax Formula: 𝑓(𝑧) = ∑ Purpose: Used in the output layer for multi-class classification tasks. SoftMax helps in converts logits into probabilities, here the sum of the outputs equals 1.

Effect: Useful for classification problems where each class has a probability, and you need the model to select the most likely class.

Let’s have a look how these are implemented in Multilayer Perceptron (MLP) for the image classification using the CIFAR-10 dataset

**Dataset:**

* **CIFAR-10**: 60,000 images (50,000 training, 10,000 testing), 32x32 pixels, 10 classes (e.g., airplane, car, cat, etc.).

**Why CNNs?**

* CNNs are ideal for images as they detect patterns (edges, textures) and learn hierarchical features efficiently.

**Model Architecture:**

1. **Input Layer**: Images (32x32x3).
2. **Convolutional Layers**: Extract features using filters.
3. **Pooling Layers**: Reduce spatial dimensions while preserving key features.
4. **Fully Connected Layers**: Combine features for classification.
5. **Output Layer**: Softmax activation for 10-class probabilities.

**Training:**

* **Forward Propagation**: Pass images through the network to compute predictions.
* **Loss Function**: Categorical Crossentropy measures the difference between predicted and actual labels.
* **Optimization**: Adam optimizer adjusts weights using gradients.

**Regularization:**

1. **Dropout**: Randomly deactivate neurons to prevent overfitting.
2. **Early Stopping**: Stop training when validation performance plateaus.
3. **Data Augmentation**: Enhance training data with transformations like flipping and rotation.

**Evaluation:**

**Metrics**: Accuracy, Precision, Recall, F1-Score.

**Confusion Matrix**: Visualize correct/incorrect predictions.

**Training Curves**: Compare training and validation loss to detect overfitting.

**Advanced Techniques:**

**Transfer Learning**: Use pre-trained models for better results.

**Learning Rate Scheduling**: Gradually reduce learning rate for efficient convergence.

**This report includes:**

CNNs, combined with regularization and augmentation techniques, effectively classify CIFAR-10 images, achieving robust and generalizable results.

1. **Moon-Shaped Problem: Non-Linearly Separable Data**
   1. The scatter plot shows two classes:
      1. Class 0 (red points) and Class 1 (blue points).
   2. A **linear decision boundary** (gray dashed line) is drawn to demonstrate that linear models cannot separate these classes effectively.

**Key Concepts to we can learn from above Moon-Shaped Problem:**

**1.Linear vs. Non-Linear Separability:**

* + Linear models (e.g., Logistic Regression, Linear SVM) fail to classify this data because the boundary required to separate the classes is **non-linear**.
  + This introduces the idea of **kernel tricks** or non-linear models.

1. **Real-World Analogies:**
   * Many real-world problems (e.g., image recognition, voice classification) are non-linearly separable.
   * The moon dataset is a simple proxy for these complex scenarios.
2. **Potential Solutions:**
   * **Kernel SVMs:** Use kernels (e.g., RBF) to project the data into a higher dimension where it becomes linearly separable.
   * **Neural Networks:** Learn complex non-linear decision boundaries directly from data.

By going from the Moon dataset to CIFAR-10:

1. **Scalability**: We can understand why simple models (linear, kernel SVM, or shallow MLPs) work well for toy datasets but struggle with complex, high-dimensional data.
2. **Power of Neural Networks**:
   * Neural networks (MLPs) can adapt to non-linear boundaries (Moon dataset).
   * With deeper architectures, they can handle real-world challenges (CIFAR-10).
3. **Techniques for Complexity**:
   * Regularization (L2, dropout) prevents overfitting on large datasets like CIFAR-10.
   * Batch normalization ensures stable training and faster convergence.
4. **Representation**:
   * For simple structured data (Moon), we rely on feature engineering.
   * For CIFAR-10, the model learns hierarchical features automatically.

**2. Transition to CIFAR-10: A Real-World Challenge**

* The CIFAR-10 dataset takes the learning to a **real-world problem**:
  + **Data Complexity**: Unlike the two-dimensional Moon dataset, CIFAR-10 contains high-dimensional, color images (32×32×3 pixels, i.e., 3,072 features per image).
  + **Task Complexity**: Instead of a binary classification (two classes like Moon), CIFAR-10 has **10 classes** (e.g., airplanes, cars, birds, etc.), making the decision boundaries far more intricate.
  + **Data Size**: CIFAR-10 has tens of thousands of images, requiring models that can scale and generalize well.

**Why We Are Using CIFAR-10 After Moon?**

The CIFAR-10 example builds on the fundamental principles learned with the Moon dataset but introduces the following:

**A. Higher Dimensionality**

* **Moon Dataset**: Low-dimensional (2 features, easy to visualize).
* **CIFAR-10**: High-dimensional (3,072 features per image).
  + Linear methods, or even basic polynomial features, **cannot handle such high-dimensional data** effectively. Here, we need advanced models like **Deep Neural Networks**.

**B. More Complex Data Structure**

* **Moon Dataset**: Data points are simple and structured (curved boundaries).
* **CIFAR-10**: Images are highly unstructured, with intricate patterns (e.g., shape, texture, and color of objects).
  + A **Multilayer Perceptron (MLP)** or **Convolutional Neural Network (CNN)** can learn these complex patterns and relationships.

**C. Scaling to Larger Datasets**

* **Moon Dataset**: Small dataset (100 samples in our example), easily handled by simpler models.
* **CIFAR-10**: Larger dataset (60,000 images), requiring more computational power and efficient models to process and classify.

**The CIFAR-10 Model Relates to the Moon Dataset**

The Moon dataset introduces the **concept of non-linear decision boundaries**, while the CIFAR-10 model shows how to extend these ideas to much more complex scenarios:

**A. Moving to Neural Networks**

* The Moon dataset demonstrates **basic non-linear models** like MLPs to handle non-linear boundaries.
* For CIFAR-10, we use a **deeper, more robust MLP** with:
  + **Regularization** (L2 regularization, dropout).
  + **Batch Normalization** (to stabilize training on larger data).
  + These techniques address challenges like overfitting and training instability, which become critical for more complex datasets.

**B. Input Representation**

* **Moon Dataset**: Simple 2D coordinates (x, y).
* **CIFAR-10**: 32×32 pixel images (3,072 features), requiring more sophisticated transformations and layers.

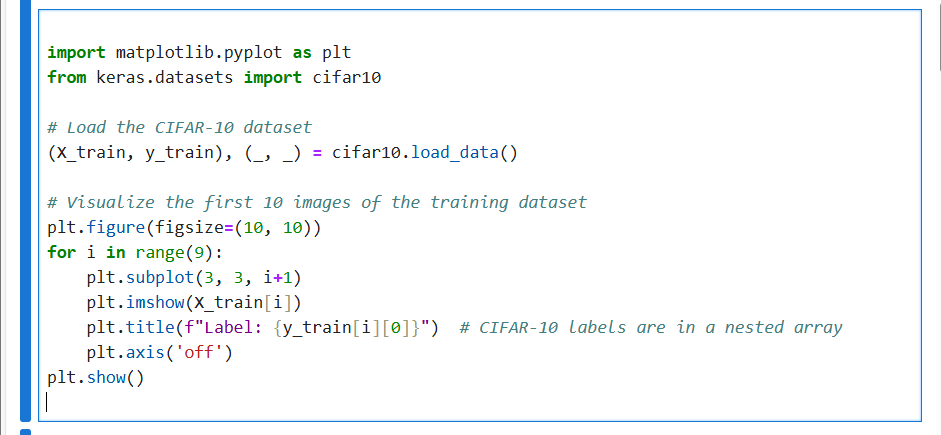
**C. Decision Boundaries**

* **Moon Dataset**: The boundary is a simple curve separating two classes.
* **CIFAR-10**: The boundaries are intricate and multi-dimensional, requiring deeper layers in the network to progressively learn features (e.g., edges, shapes, textures).

**D. Generalization**

* **Moon Dataset**: The model generalizes to new points in 2D space.
* **CIFAR-10**: The model generalizes to unseen images by learning complex features like edges and object structures.

**Now**, we are going through the code of CIFAR-10 DATA SET



This code loads the **CIFAR-10 dataset**, which consists of **50,000 training images** and **10,000 test images** across **10 categories** (e.g., airplane, car, bird, cat). The training images are used to teach a machine learning model to classify new images into these categories.

The code then **visualizes the first 9 images** from the training set in a 3x3 grid. Each image is labeled with its class number (e.g., 0 for airplane, 1 for automobile, etc.).

**Purpose:**

1. **Understand the data**: Shows what the images look like (e.g., small, colorful, and low-resolution).
2. **Check diversity**: Highlights the variety in the dataset and ensures it is properly loaded.
3. **Prepare for training**: Helps us think about the features the model must learn (e.g., shapes, colors, or textures) to classify images.

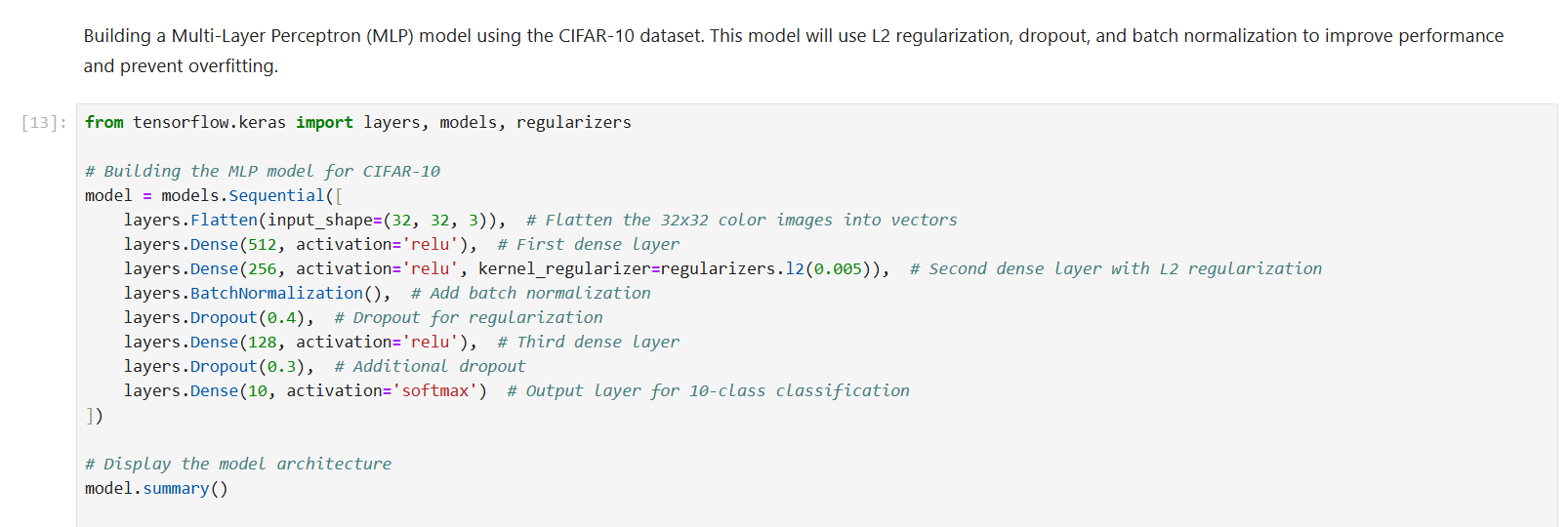
By doing this, we get a quick overview of the dataset and its challenges, such as overlapping features (e.g., similar colors or shapes in different categories). This prepares you to design and train a model effectively.

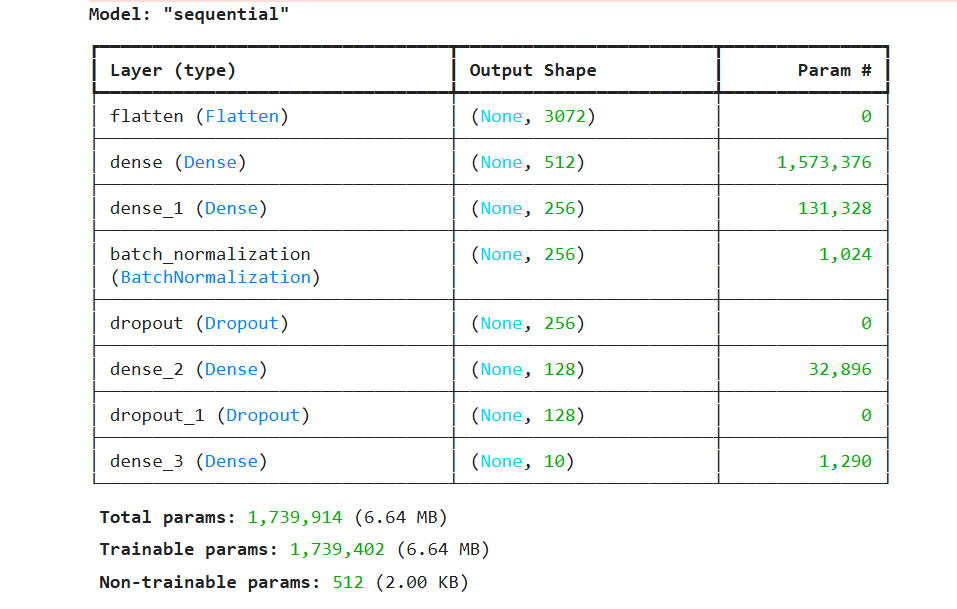
* OUTPUT: We’ll see a grid of 9 images from the training set displayed in color, each representing one of the CIFAR-10 classes (e.g., airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).
* The **labels** (numerical values 0-9) for these images are displayed as titles, corresponding to the classes.

**Building a Multi-Layer Perceptron (MLP) model** using the CIFAR-10 dataset. This model will use L2 regularization, dropout, and batch normalization to improve performance and prevent overfitting.

Let us,discuss some more advance techniques in MLPs Regularization Techniques:

**Dropout**: In this technique we randomly "dropping out" neurons during training to prevent overfitting. It helps network generalize better by preventing it from becoming too reliant on specific neurons or features.



Fig.Dropout Technique to prevent overfitting

**Explanation of the Model Architecture**

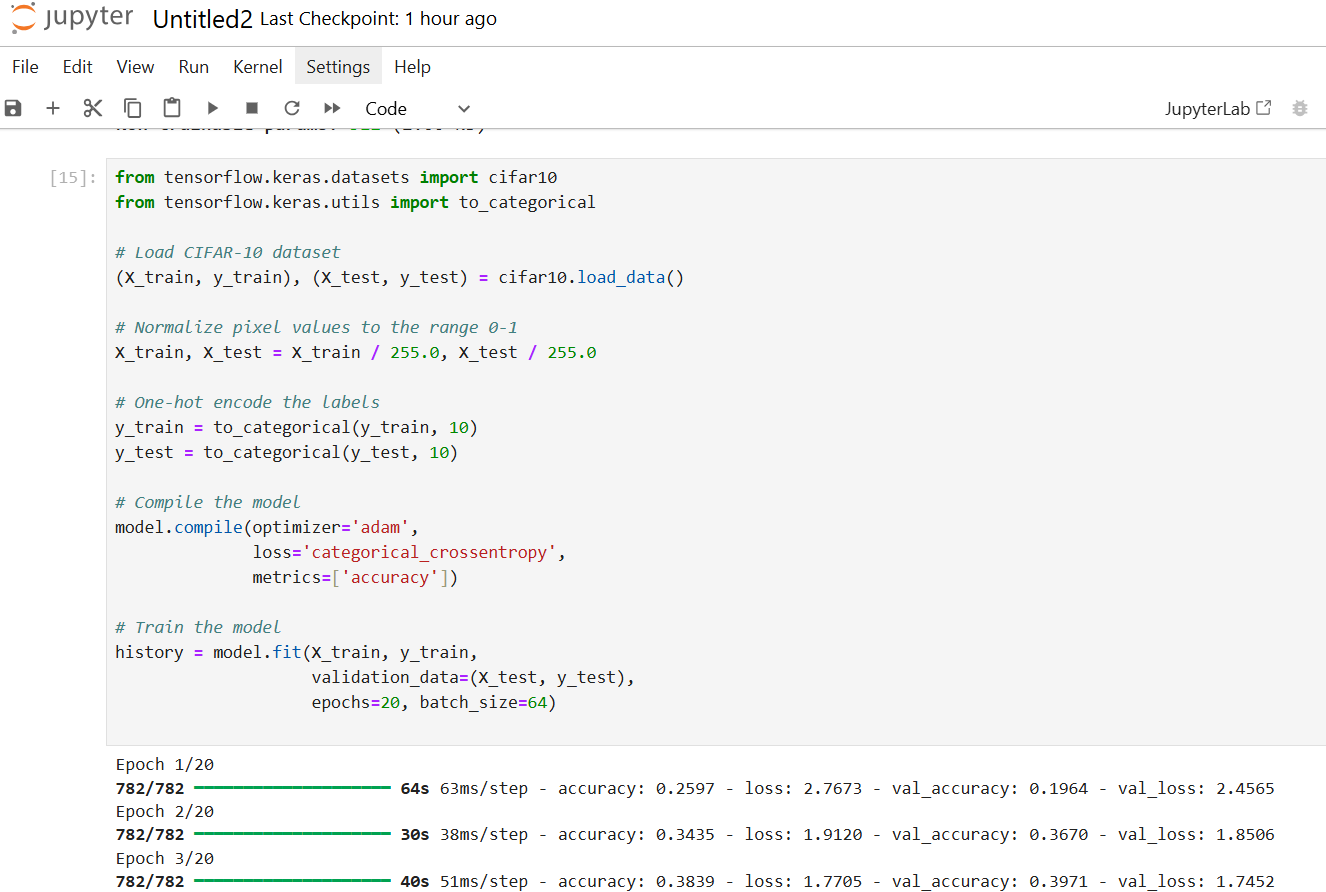
1. **Flatten Layer**:
   * **Input shape**: (32, 32, 3) represents the CIFAR-10 images (32x32 pixels with 3 color channels).
   * The **Flatten** layer converts these 2D images into 1D vectors, so they can be processed by dense layers.
2. **Dense Layer 1**:
   * **512 neurons** with **ReLU activation**: A fully connected layer that learns 512 features from the flattened image data.
   * ReLU allows the model to learn non-linear relationships.
3. **Dense Layer 2**:
   * **256 neurons** with **ReLU activation**.
   * **L2 regularization**: It helps prevent overfitting by adding a penalty for large weights during training.
4. **Batch Normalization**:
   * Normalizes the inputs to the next layer, improving training speed and stability.
5. **Dropout Layer 1**:
   * **40% dropout**: Randomly drops 40% of neurons during training to prevent overfitting.
6. **Dense Layer 3**:
   * **128 neurons** with **ReLU activation**: Further processes the features learned by previous layers.
7. **Dropout Layer 2**:
   * **30% dropout**: Further regularization to help with generalization.
8. **Output Layer**:
   * **10 neurons** with **softmax activation**: One neuron for each class (e.g., airplane, cat) in CIFAR-10. Softmax converts the output into probability values for each class.

**Summary Output**

* The **model.summary()** will display the following details:
  + **Layer types**: Flatten, Dense, BatchNormalization, Dropout.
  + **Output shapes**: Shape of the data after each layer.
  + **Parameters**: Number of weights and biases in the model.
  + **Total parameters**: Total number of trainable weights in the entire model.

**Total Parameters**: The total number of parameters (weights and biases) in the model is shown (e.g., 1.7 million parameters).

**Training the model, we will see output related to each training epoch.**



**Epoch X/20**:

* Shows the current epoch number (e.g., Epoch 1/20 means it's the first epoch out of 20 total epochs).

**782/782 [==============================]**:

* This represents the number of steps taken to process the entire training dataset. The number 782 corresponds to the number of batches (since the batch size is 64, 50,000 training samples are divided by 64).

**50s 64ms/step**:

* Time taken to complete one step (64ms per batch), and the total time for the epoch (50 seconds).

**loss: 1.9074**:

* The **loss** value represents the difference between the model's predicted output and the true labels. A lower value is better, as it indicates the model's predictions are getting closer to the actual labels.

**accuracy: 0.3215**:

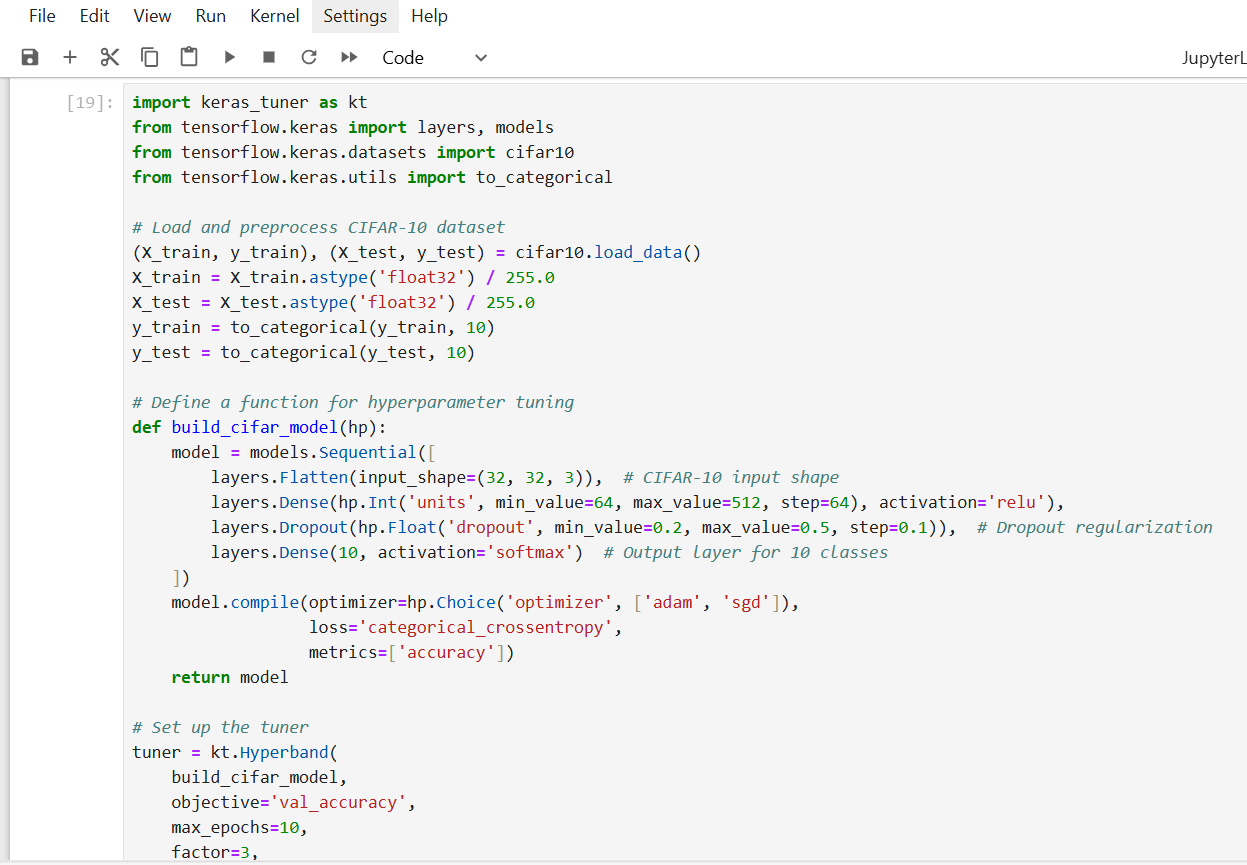
* This is the **training accuracy** — the percentage of correct predictions made on the training set. For example, accuracy: 0.3215 means the model correctly predicted 32.15% of the training samples.

**val\_loss: 1.7592**:

* This is the **validation loss**, which indicates how well the model is performing on the test set. It's similar to the training loss but calculated on the unseen validation data.

**val\_accuracy: 0.3672**:

* This is the **validation accuracy**, indicating how well the model is performing on the test set. For example, val\_accuracy: 0.3672 means the model correctly predicted 36.72% of the test samples after that epoch.



**Hyperparameter Tuning**: The code performs hyperparameter tuning to find the best combination of model parameters that maximizes the validation accuracy.

**Model Evaluation**: After finding the best model, it is tested on the test set, and the test accuracy is printed.

NOW, It uses **convolutional layers** to extract spatial features, **max pooling** to reduce dimensionality, and **dense layers** to classify the digits.

After training for 5 epochs, the model is evaluated on the test set, and the accuracy and loss are printed. The higher the accuracy and the lower the loss, the better the model's performance.

The **confusion matrix** will have the true classes along the y-axis and the predicted classes along the x-axis.

The diagonal elements (from top-left to bottom-right) represent correct predictions, where the predicted class matches the true class.

Off-diagonal elements represent misclassifications, where the model predicted one class but the true label was a different class.

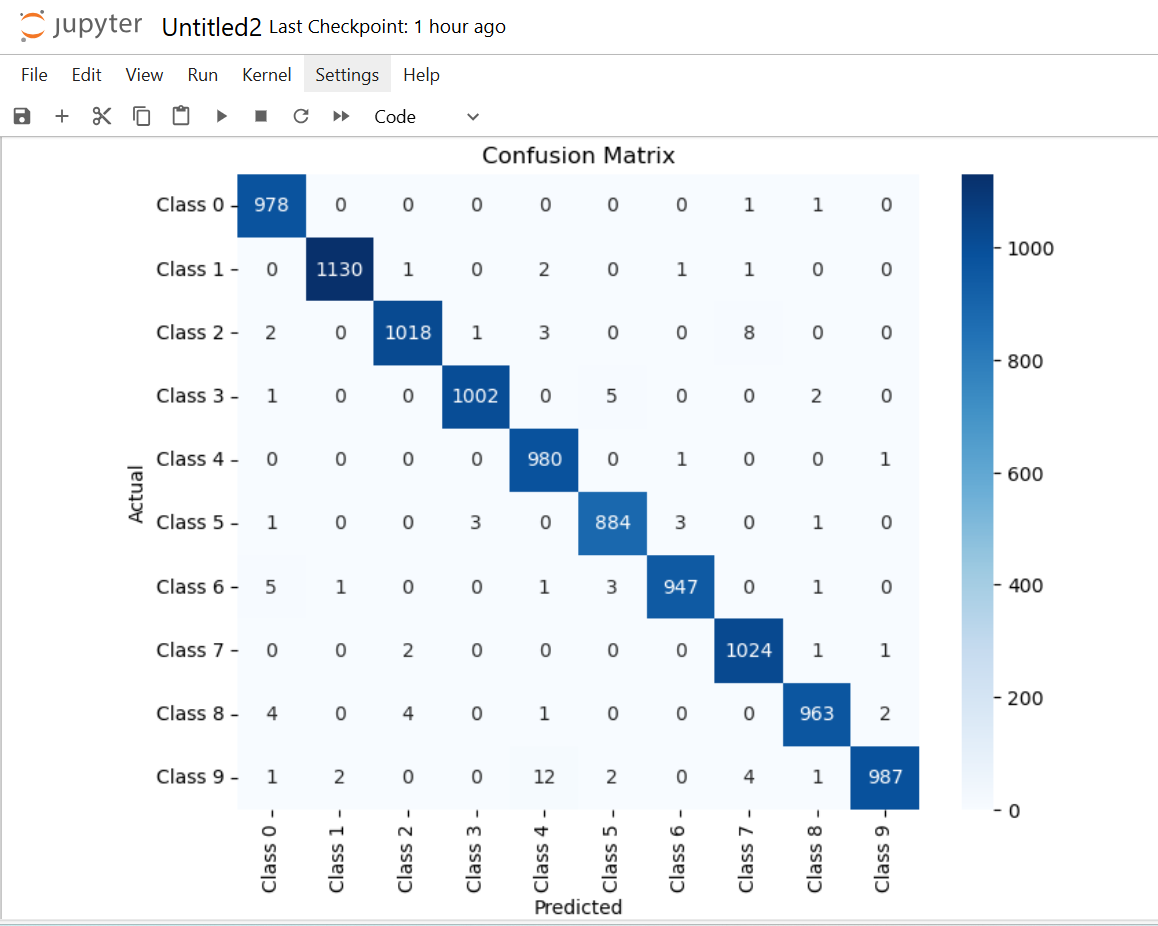


Fig. confusion Matrix for Visualising the predictions

**Significance**: We can Visualize the number of correct and incorrect predictions in each class. Highlights which classes are most often confused, guiding improvements in the model.

**Training and Validation Curves**

Why: Identify overfitting or underfitting by comparing training and validation accuracy

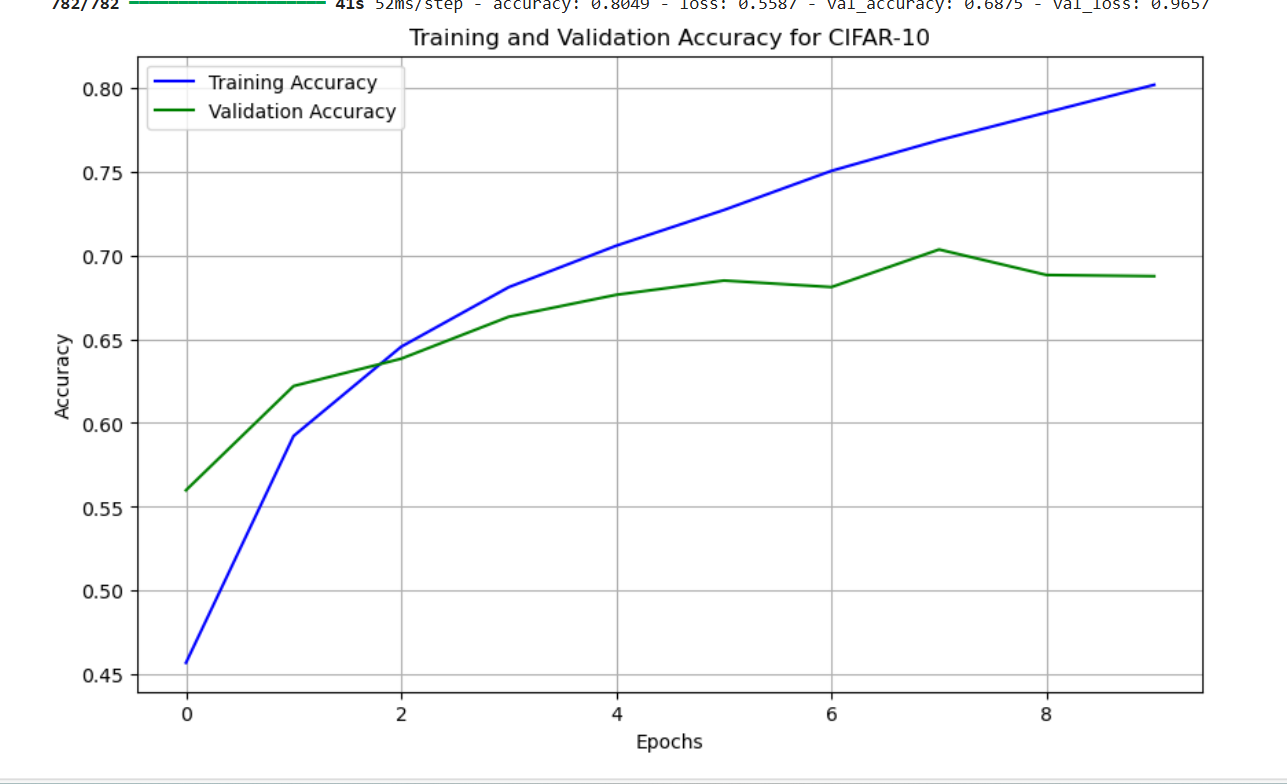


Fig .Training and Validation Accuracy plots

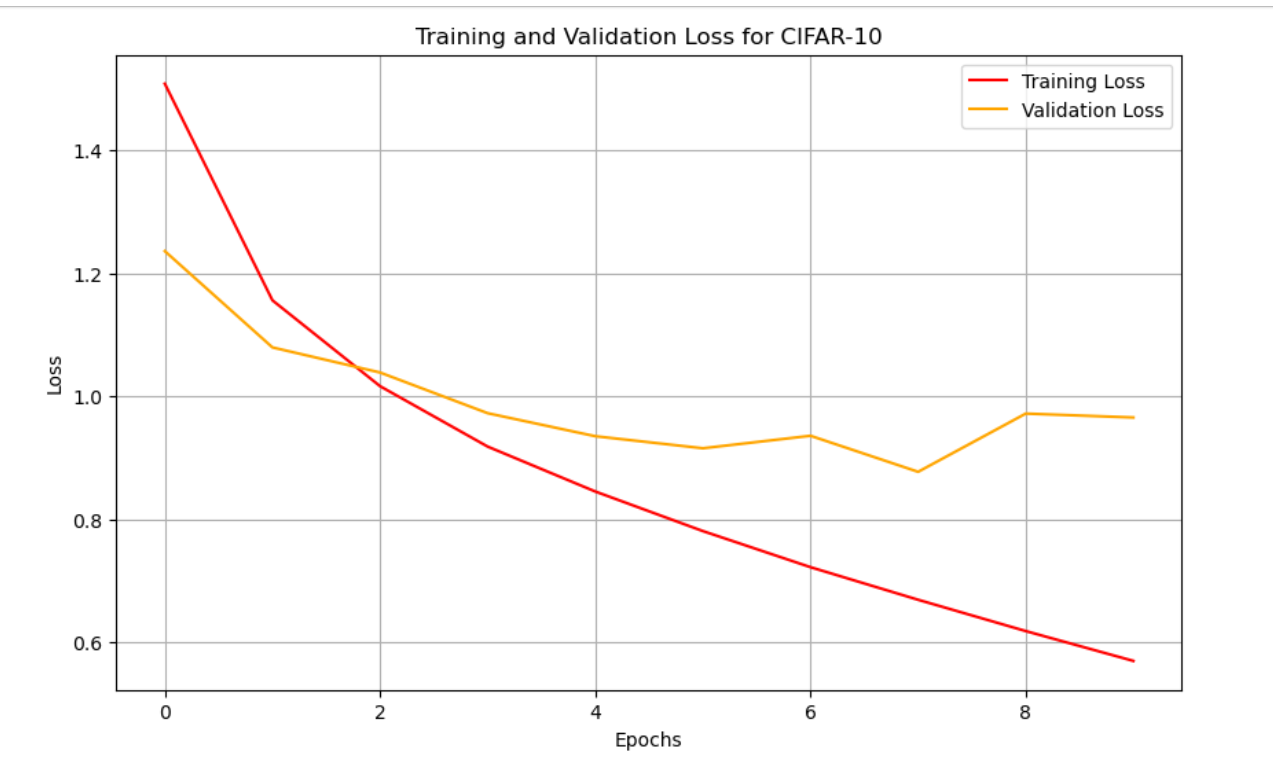
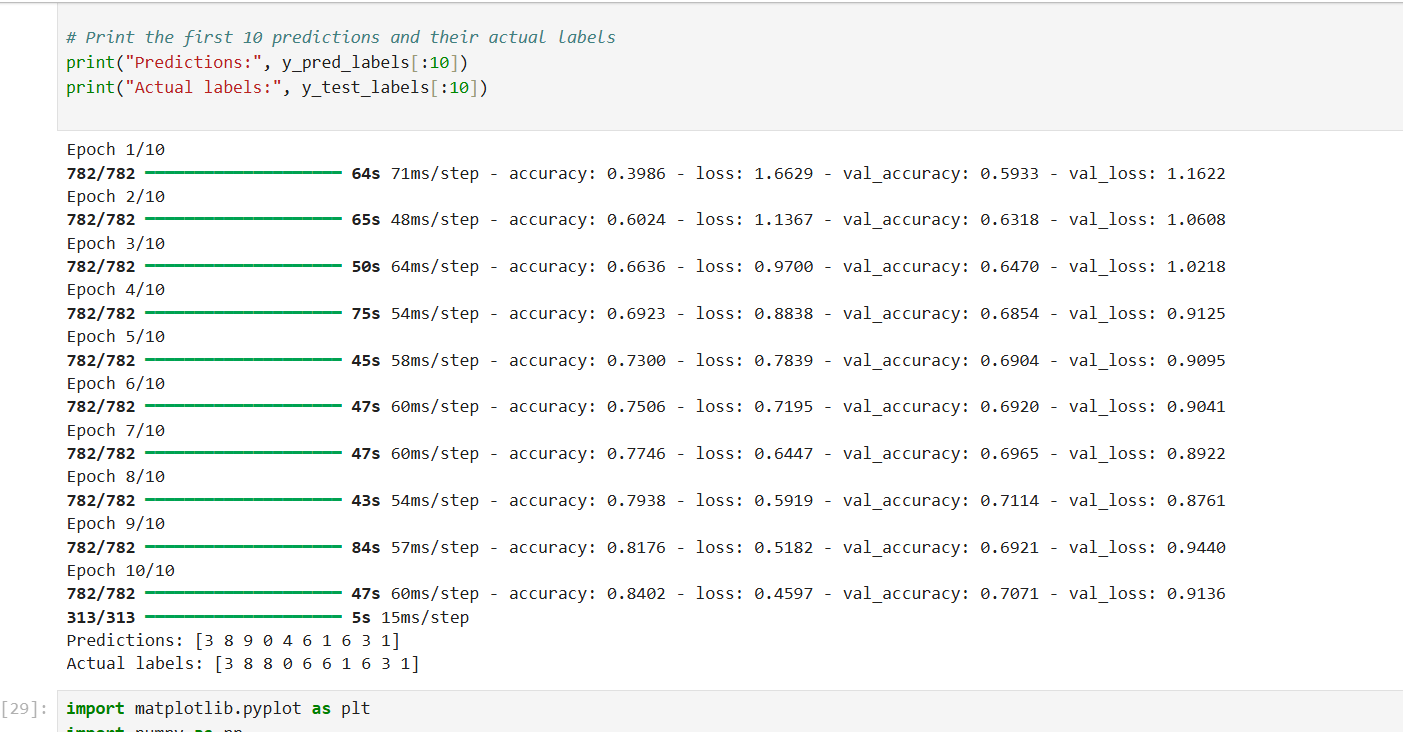
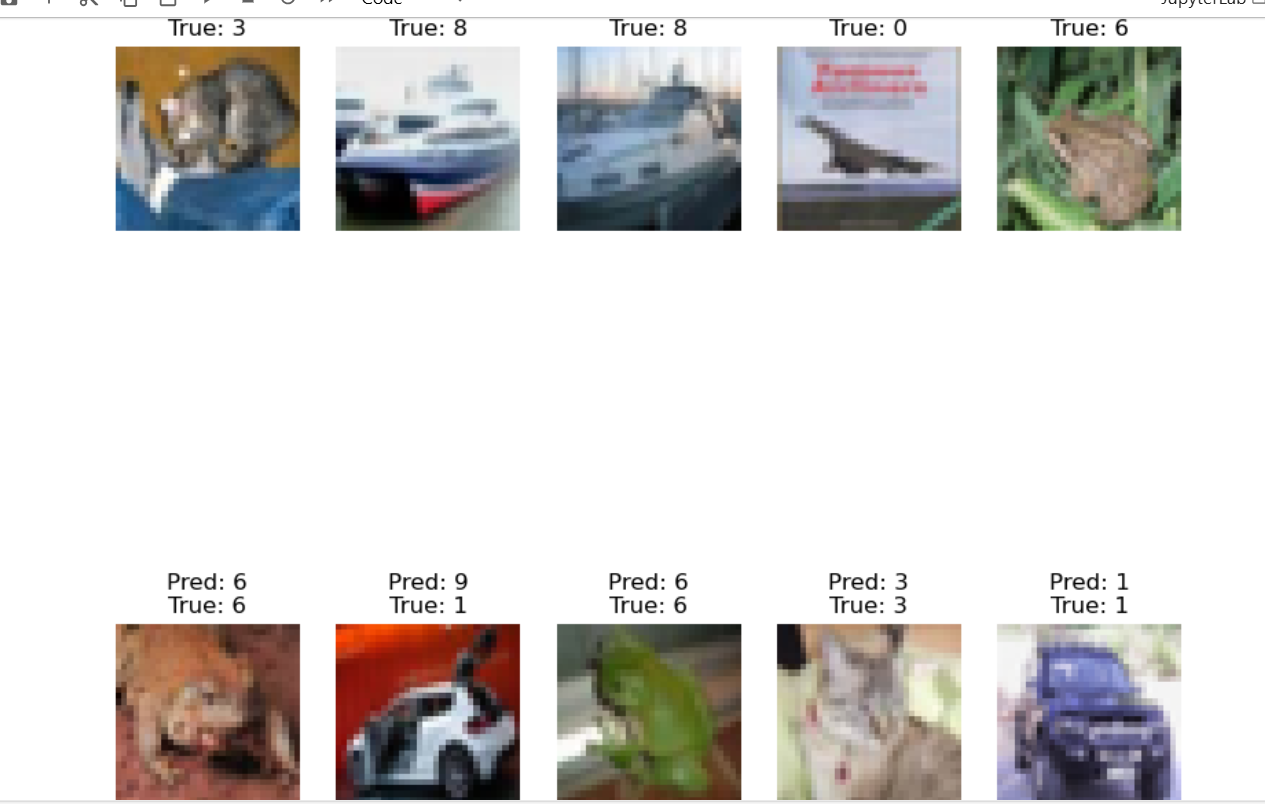


Fig . Training and Validation Loss plots

Here training accuracy and validation accuracy continues to improve indicating best fit. (If validation accuracy stagnates or drops, then model is overfitting.) similarly in opposite with loss function A balanced improvement in both curves indicates a well-generalized model. By observing above figure overall model is trained properly and had a good performance.

****

****

**How ML challenges are solved using MLPs**

Machine Learning (ML) has revolutionized industries engaging systems to learn from data and make data-driven predictions. Despite its transformative potential, ML models often face several significant challenges that limit their effectiveness in complex scenarios:

Non-linearity: Many real-world problems involves non-linear relationships between input features and outputs, where traditional models like linear regression fail to capture. For instance, predicting house prices requires modelling intricate, non-linear interactions among factors like location, size, and market trends.

Multilayer Perceptrons(MLPs) overcome thislimitation by introducing the hidden layers with non-linear activation functions (e.g., Fig Training and Validation Loss plots Significance: Here training accuracy and validation accuracy continues to improve indicating best fit. (If validation accuracy stagnates or drops, then model is overfitting.) similarly in opposite with loss function A balanced improvement in both curves indicates a well-generalized model.

By observing above figure overall model is trained properly and had a good performance. Predictions and visualisation: Fig Predictions and visualisation How ML challenges are solved using MLPs Machine Learning (ML) has revolutionized industries engaging systems to learn from data and make data-driven predictions. Despite its transformative potential, ML models often face several significant challenges that limit their effectiveness in complex scenarios: ReLU). These layers transform the input space, enabling MLPs to solve non-linear problems like image classification, having the relationships between pixels are highly complicated**.**

**Feature Engineering**: Traditional ML approaches rely heavily on manual feature engineering to extract meaningful information from raw data, a time-intensive, also error prone process. MLPs eliminate this dependency by automatically learning important features from data during training. For example, in image recognition, MLPs can detect edges, textures, and patterns directly from pixel data, progressing from low-level features to high-level abstractions**.**

**Overfitting:** Overfitting occurs when models memorize training data instead of generalizing patterns, leads to poor performance on unseen data. MLPs address overfitting regularization techniques such as: using

**Dropout:** Randomly deactivating neurons during training to reduce reliance on specific pathways.

**Early Stopping:** Halting training when the validation performance plateaus.

**L2 Regularization**: Penalizing large weights to simplify the model.

I**nterpretability**: While MLPs are powerful, they are often criticized for being "black box" models. This lack of transparency could pose challenges in critical domains like healthcare, finance, where understanding model decisions is essential. Recent advancements, such as Layer-wise Relevance Propagation (LRP) and SHAP (Shapley Additive Explanations), enhance interpretability of MLPs, making it possible to explain predictions and gain insights into the underlying decision-making processes.

**Drawbacks**

**Overfitting**: MLPs can easily be overfit on small datasets or when regularization is not applied. This happens because the model memorizes training data instead of learning generalized patterns. Suggestion: Ensure regularization techniques like Dropout, L2 regularization, and Early Stopping are properly implemented to overcome overfitting**.**

**Black-box Nature Issue:** MLPs are often seen as black-box models, making it hard to interpret their decision-making process.

**Suggestion:** To improve interpretability, introduce methods like SHAP or Layer-wise Relevance Propagation (LRP) to visualize how the model makes predictions. Vanishing Gradient Problem Computational Cost

**Difficulty in Scaling to Large Datasets** For more complex image data, consider using Convolutional Neural Networks (CNNs). While MLPs can handle high-dimensional data, CNNs specialize in extracting hierarchical features from images, leading to better Performance

**References:**

• Aggarwal, C. (2018). Neural Networks and Deep Learning: A Textbook. Springer.

• DataCamp. (2020). Mastering Backpropagation: A Comprehensive Guide for Neural Networks. Retrieved from <https://www.datacamp.com>.

• Scikit-learn Documentation. (2021). Varying Regularization in Multi-layer Perceptron. Retrieved from <https://scikit-learn.org>.

• Chollet, F. (2017). Deep Learning with Python. Manning Publications.

• Towards Data Science. (2020). Understanding the Basics of Multi layer Perceptrons (MLPs). Retrieved from https://towardsdatascience.com.