**Import necessary libraries:**

First of all we import necessary libraries.

* TensorFlow/Keras: For building and training deep learning models.
* Sklearn: For evaluation metrics (e.g., confusion matrix, classification report).
* Matplotlib and Seaborn: For visualizing results like accuracy, loss, and confusion matrices.
* Numpy: For numerical operations on arrays.
* **Loading dataset:**

And then we load the dataset CIFAR-10 which has 60,000 32x32 images in 10 classes like airplane,car,bird and so on .We used 50,000 images as a training data and remaining 10,000 images as the test data.

* **Normalize pixel values:**

Pixel values in the images range from 0 to 255.We perform the Normalization on pixel values to scale these values [0,1].It helps the model to train faster and improve their performance.

* **Splitting training data into training set and validation set:**

Training data is further split into Training Set (80%) used to train the model.

Validation Set (20%) used to evaluate the model during training to avoid overfitting.

test\_size=0.2: 20% of the original training data is reserved for validation.

random\_state=42: Ensures reproducibility of the split.

* **Our Goal:**

Our goal is to perform CNN on the dataset of images to predict the label of images.This is deep learning model which perform classification on images data.

* **Defining the Model:**

Sequential A linear stack of layers where each layer's output serves as the input to the next layer. Each layer is added sequentially to build the CNN.

* **Layers Explanation:**

1. **Convolutional Layer 1:**

* **Purpose:** Extract features from the input images.
* **32 filters:** The layer has 32 filters to detect patterns (edges, textures, etc.).
* **Kernel size (3, 3):** Each filter scans a 3x3 section of the input.
* **Activation 'relu':** Applies a Rectified Linear Unit activation to introduce non-linearity.
* **Input shape (32, 32, 3):** Images are 32x32 pixels with 3 color channels (RGB).

1. **Max Pooling Layer 1:**

* **Purpose:** Downsample the feature map by selecting the maximum value from each 2x2 region.
* **Effect:** Reduces the spatial dimensions (height and width), making the model computationally efficient and robust.

1. **Convolutional Layer 2:**

* Similar to the first convolutional layer but with 64 filters. It learns more complex patterns from the downsampled feature map.

1. **Max Pooling Layer 2:**

* Further reduces the size of the feature map.

1. **Convolutional Layer 3:**

* Another convolutional layer with 64 filters, extracting even deeper features.

1. **Flatten Layer:**

* **Purpose:** Converts the 2D feature maps into a 1D vector so it can be passed to fully connected layers.

**Example:** A feature map of size (8, 8, 64) becomes a 1D vector of size 8\*8\*64 = 4096.

1. **Dense Layer 1:**

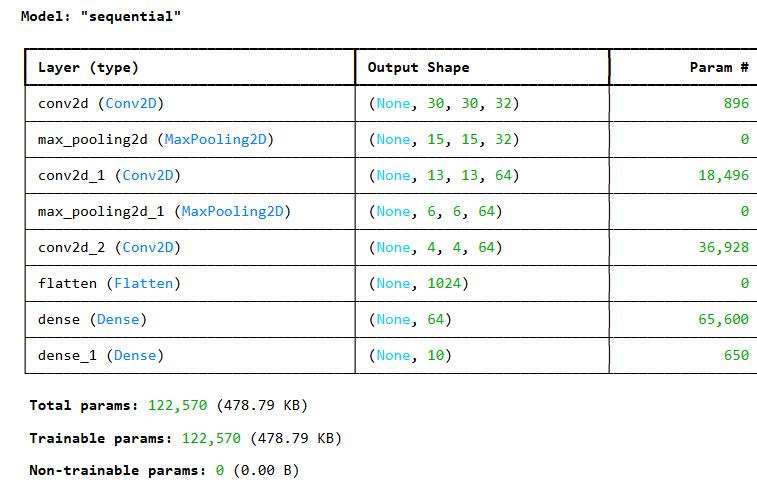
* **Purpose:** A fully connected layer with 64 neurons, learning complex relationships between the features.
* **Activation 'relu':** Introduces non-linearity**.**

1. **Dense Layer 2 (Output Layer):**

* **Purpose:** Final layer with 10 neurons, one for each class in CIFAR-10.
* **Activation 'softmax':** Converts the output into probabilities for each class, summing to 1.
* **Model Summary:**

Displays the architecture, showing each layer’s type, output shape, and the number of trainable parameters.

**For Example:**

****

* **Model Compilation:**

**a. Optimizer: adam**

* **Purpose**: Determines how the model's weights are updated during training.
* **Adam (Adaptive Moment Estimation)**:
  + Combines the benefits of **Momentum** and **RMSProp**.
  + Dynamically adjusts the learning rate for each parameter.
  + Works well for most deep learning tasks.

**b. Loss Function: sparse\_categorical\_crossentropy**

* **Purpose:** Measures how far the model's predictions are from the true labels.
* **Used for multi-class classification problems where:**

Labels are integers (e.g., 0, 1, 2… for CIFAR-10 classes).

* **Example:** If the true label is 2, this function penalizes the model if it predicts something other than class 2.

**c. Metrics: accuracy**

* **Purpose:** Tracks the proportion of correctly classified images during training and evaluation.
* **Training the Model:**

**a. model.fit():**

* **Purpose:** Trains the model on the dataset using the optimizer, loss function, and metrics defined during compilation.
* **Arguments:**

**train\_images & train\_labels:** Training data and corresponding labels.

* **epochs=100:** The model will go through the entire training dataset 100 times.
* **validation\_data=(val\_images, val\_labels):**
* During training, the model's performance is evaluated on the validation dataset after each epoch.
* Helps track overfitting and fine-tune hyperparameters.

**b. history:**

* **Purpose:** Stores the training process details, such as:

Training and validation accuracy.

* Training and validation loss for each epoch**.**
* **Usage:** Useful for visualizing the training process (e.g., plotting accuracy or loss curves).
* **Model Evaluation (model.evaluate):**

This line evaluates the trained model on the test data **(test\_images and** **test\_labels).** It calculates how well the model performs by returning two metrics **test\_loss (the loss value)** and **test\_acc (the accuracy of the model on the test** **data).** It gives you an overall score of the model's performance on data it hasn't seen before.

* **Classification Report (classification\_report):**

First, the model makes predictions on the test data **(model.predict(test\_images)).** The **np.argmax** function is used to convert the model's output **(probabilities for** **each class)** into predicted class labels **(the class with the highest probability).**

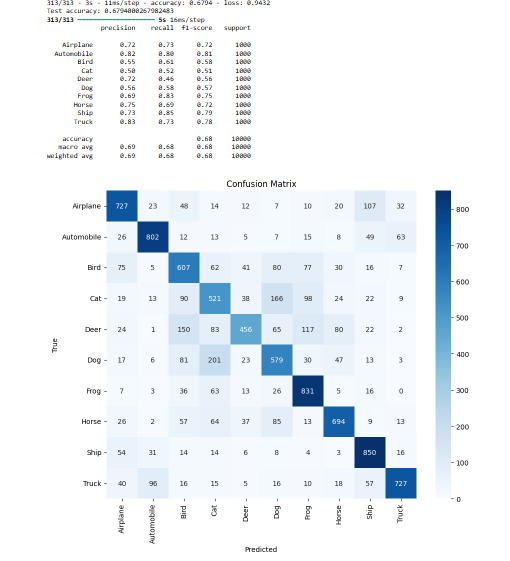
Then, classification\_report generates a report showing the precision, recall, F1-score, and support for each class (e.g., 'Airplane', 'Automobile', etc.).

It gives a detailed breakdown of the model’s performance for each individual class, allowing you to see how well it is doing in terms of both correct and incorrect predictions for each class.

* **Confusion Matrix (confusion\_matrix):**

**confusion\_matrix** creates a matrix that shows how many times each class was predicted correctly and how many times it was confused with another class.

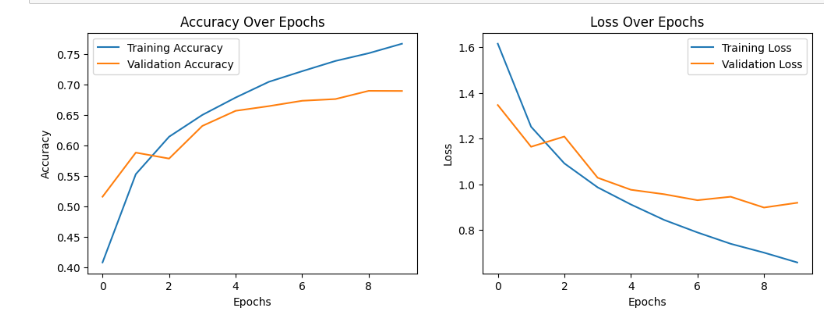
The **sns.heatmap** is used to visualize this confusion matrix. Each cell of the matrix shows the number of instances for that combination of true and predicted class labels.

This visualization helps you understand where the model is making mistakes, showing you the counts of correct predictions and misclassifications for each class.

* **Visualization:**

**Training vs. Validation:** The two plots (accuracy and loss) compare the model's performance on both the training data and the validation data.

**Purpose:** These visualizations help you track the model's learning progress. If the training accuracy increases while the validation accuracy stays the same or decreases, it might indicate overfitting. Similarly, if the training loss is much lower than the validationloss, the model might not generalize well to unseen data.



* **Performance:**

The code provides a summary of the model's performance, highlighting its strengths and weaknesses.It saves important information (like training metrics and confusion matrix visualizations) into files for future reference or reporting purposes. The model achieved an accuracy of **67.94%** on the test set.

**Strengths:** The model performed well in classes like 'Airplane' and 'Automobile' with high precision.

**Limitations:** Misclassifications were frequent in 'Cat' vs 'Dog', possibly due to similar features.