***Experiment - 7***

**Name: Sushant Tulasi**

**Div: D20B Roll no: 60**

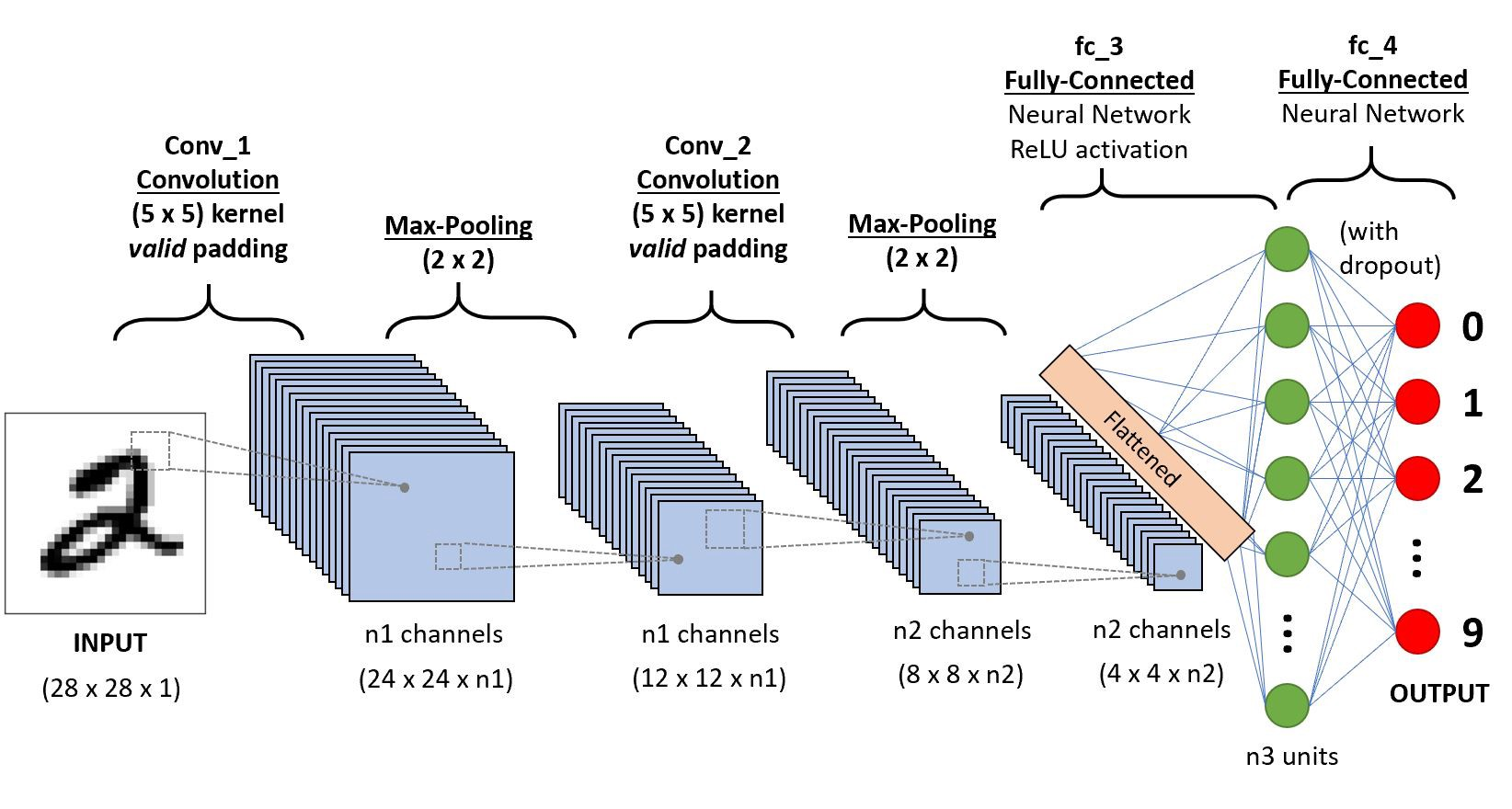
**Aim**:To implement CNN Deep Learning Applications like i) Image Classification System

ii) Handwritten Digit Recognition System (like MNIST Dataset) iii) Traffic Signs Recognition

**Theory:**

# Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing and classifying visual data, such as images and videos. They are inspired by the visual perception process in the human brain and have become the state-of-the-art technique for various computer vision tasks.

**Key Components of a CNN:**

****

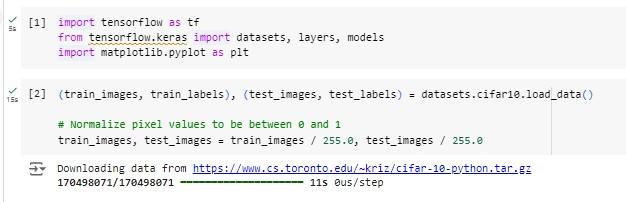
1. **Convolutional Layers:** These layers apply convolutional operations to the input image. Convolutional filters (also known as kernels) slide over the input image to extract features like edges, textures, and patterns.
2. **Pooling Layers:** Pooling layers (e.g., Max Pooling) reduce the spatial dimensions of the feature maps, helping to make the network invariant to small changes in the input.
3. **Fully Connected Layers:** These layers connect every neuron in one layer to every neuron in the next layer, allowing the network to learn complex combinations of features.

# - Working of a CNN:

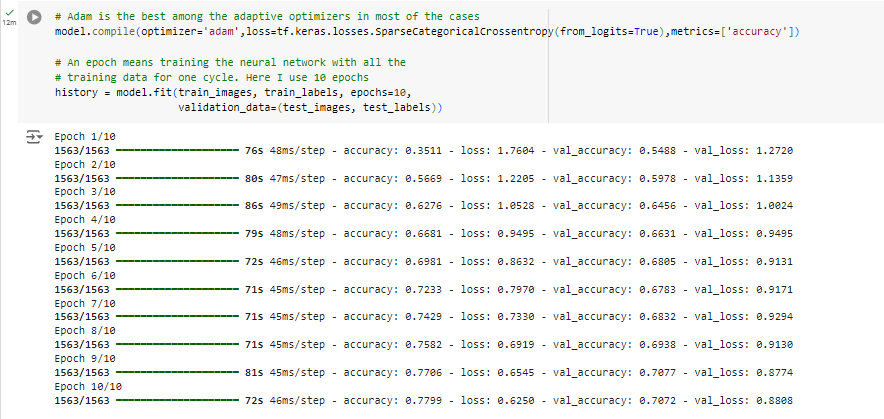
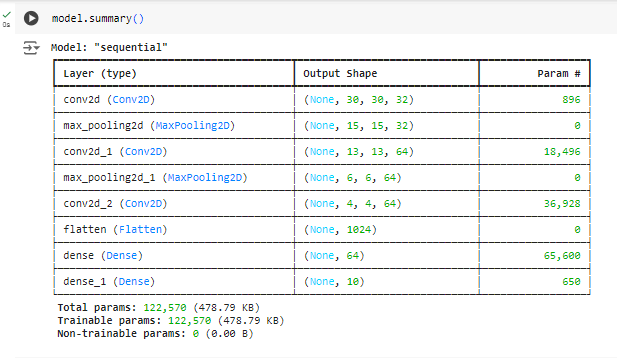
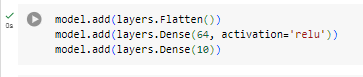
1. **Input:** A CNN starts with an input image, which is passed through a series of convolutional and pooling layers. These layers extract hierarchical features from the input.
2. **Flattening:** The final convolutional or pooling layer is flattened into a vector.
3. **Fully Connected Layers:** The flattened vector is passed through one or more fully connected layers to make predictions. These layers learn to classify the input based on the extracted features.
4. **Output:** The output layer provides the final classification result, often using softmax activation for multi-class classification tasks.

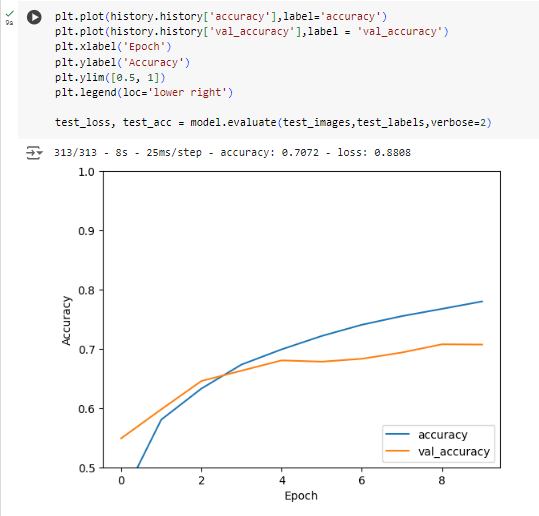
**To implement CNN Deep Learning Applications**

1. **Image Classification System**



# Data





**Code for classification of input data**

**import numpy as np**

**# Convert logits to probabilities and select the class with the highest probability predicted\_labels = np.argmax(predictions, axis=1)**

**# Making predictions**

**predictions = model.predict(test\_images)**

**# Convert logits to class labels**

**predicted\_labels = np.argmax(predictions, axis=1)**

**# Flatten test labels**

**true\_labels = test\_labels.flatten()**

**# Calculate accuracy**

**correct\_predictions = np.sum(predicted\_labels == true\_labels) total\_predictions = len(true\_labels)**

**accuracy = correct\_predictions / total\_predictions print(f"Classification Accuracy: {accuracy \* 100:.2f}%")**

**# Visualize predictions plt.figure(figsize=(10,10)) for i in range(25):**

**plt.subplot(5,5,i+1) plt.xticks([])**

**plt.yticks([]) plt.grid(False)**

**plt.imshow(test\_images[i])**

**predicted\_label = class\_names[predicted\_labels[i]] true\_label = class\_names[true\_labels[i]]**

**color = 'blue' if predicted\_label == true\_label else 'red' plt.xlabel(f"Pred: {predicted\_label}\nTrue: {true\_label}", color=color)**

**plt.show()**

**To implement CNN Deep Learning Applications**

1. **Handwritten Digit Recognition System (like MNIST Dataset)**

**# Import required libraries import tensorflow as tf**

**from tensorflow.keras import datasets, layers, models import matplotlib.pyplot as plt**

**import numpy as np**

**# Load the MNIST dataset**

**(train\_images, train\_labels), (test\_images, test\_labels) = datasets.mnist.load\_data()**

**# Preprocess the data: Normalize the pixel values to be between 0 and 1 train\_images = train\_images / 255.0**

**test\_images = test\_images / 255.0**

**# Reshape the images to fit the input format of CNN (28x28x1 for grayscale images) train\_images = train\_images.reshape((train\_images.shape[0], 28, 28, 1))**

**test\_images = test\_images.reshape((test\_images.shape[0], 28, 28, 1))**

**# Define the class names for digits 0-9 class\_names = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']**

**# Visualize the first 25 images from the training set plt.figure(figsize=(8,8))**

**for i in range(25): plt.subplot(5, 5, i+1) plt.xticks([])**

**plt.yticks([]) plt.grid(False)**

**plt.imshow(train\_images[i].reshape(28, 28), cmap=plt.cm.binary) plt.xlabel(class\_names[train\_labels[i]])**

**plt.show()**

**# Build the CNN model model = models.Sequential()**

**model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)))**

**model.add(layers.MaxPooling2D((2, 2)))**

**model.add(layers.Conv2D(64, (3, 3), activation='relu'))**

**model.add(layers.MaxPooling2D((2, 2)))**

**model.add(layers.Conv2D(64, (3, 3), activation='relu'))**

**# Add Dense layers on top for classification model.add(layers.Flatten()) model.add(layers.Dense(64, activation='relu')) model.add(layers.Dense(10))**

**# Show the model summary model.summary()**

**# Compile the model model.compile(optimizer='adam',**

**loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True), metrics=['accuracy'])**

**# Train the model**

**history = model.fit(train\_images, train\_labels, epochs=5, validation\_data=(test\_images, test\_labels))**

**# Evaluate the model on test data**

**test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2) print(f"Test accuracy: {test\_acc \* 100:.2f}%")**

**# Plot accuracy over the training epochs plt.plot(history.history['accuracy'], label='accuracy') plt.plot(history.history['val\_accuracy'], label='val\_accuracy') plt.xlabel('Epoch')**

**plt.ylabel('Accuracy') plt.ylim([0.8, 1]) plt.legend(loc='lower right') plt.show()**

**# Make predictions on the test dataset predictions = model.predict(test\_images)**

**# Function to visualize predictions along with true labels def plot\_image(i, predictions\_array, true\_label, img):**

**true\_label, img = true\_label[i], img[i].reshape(28, 28) plt.grid(False)**

**plt.xticks([])**

**plt.yticks([])**

**plt.imshow(img, cmap=plt.cm.binary)**

**predicted\_label = np.argmax(predictions\_array) if predicted\_label == true\_label:**

**color = 'blue' else:**

**color = 'red'**

**plt.xlabel(f"{class\_names[predicted\_label]} ({class\_names[true\_label]})", color=color)**

**# Visualize the first 10 test images, their predicted labels, and the true labels plt.figure(figsize=(10,10))**

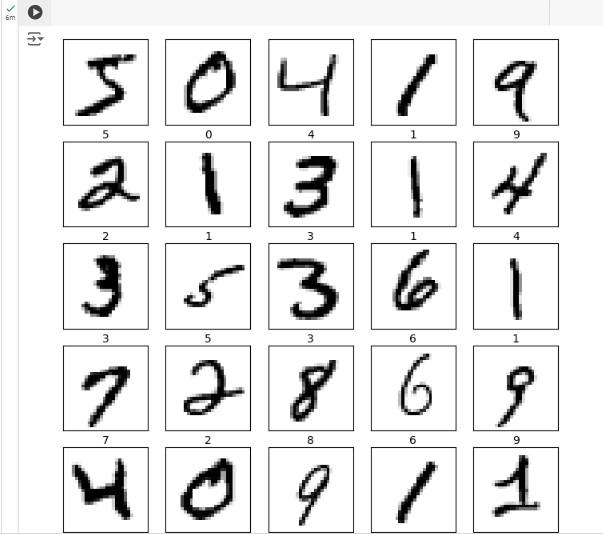
**for i in range(10): plt.subplot(5, 2, i+1)**

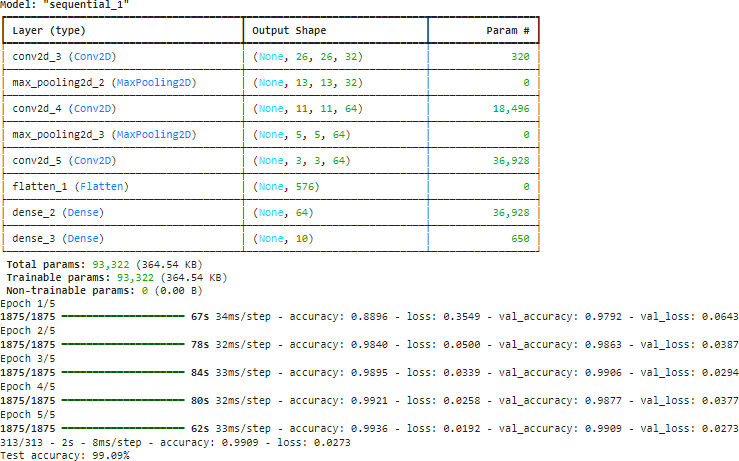
**plot\_image(i, predictions[i], test\_labels, test\_images) plt.tight\_layout()**

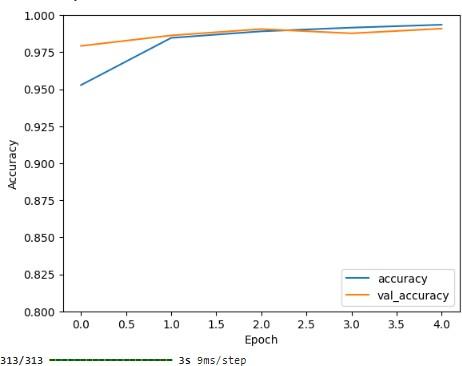
**plt.show()**

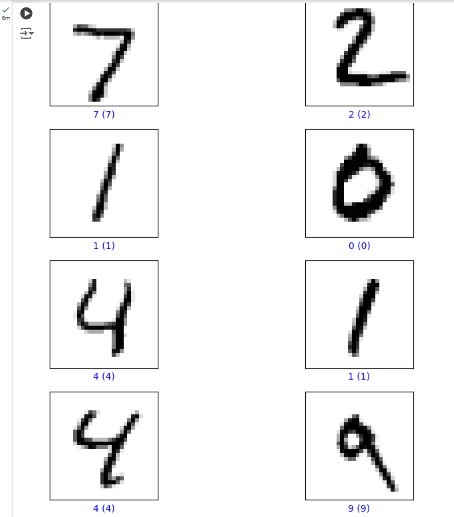
**# Get the predicted digit for the first test image predicted\_digit = np.argmax(predictions[0])**

**print(f"Predicted digit for the first test image: {predicted\_digit}")**



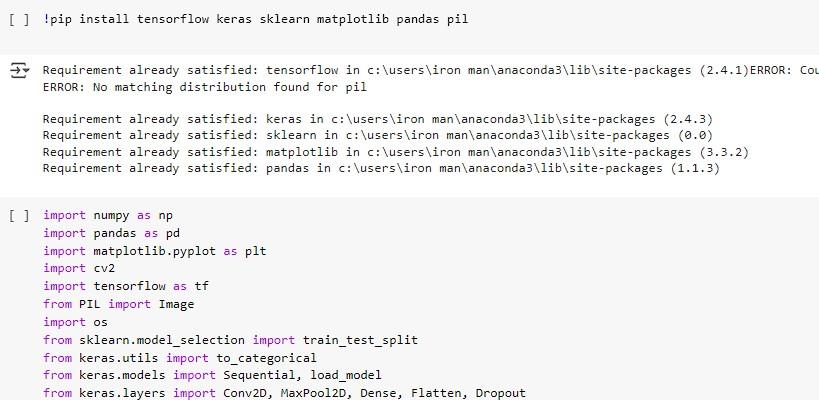




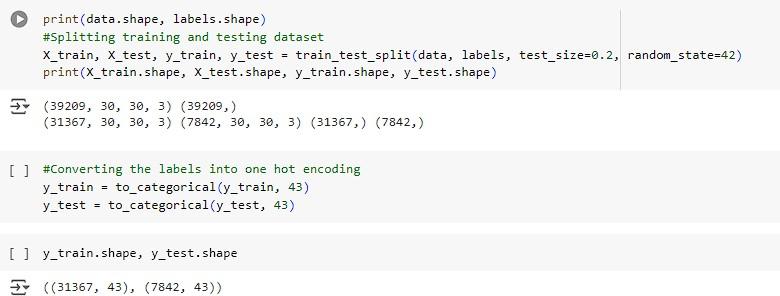
**OUTPUT**

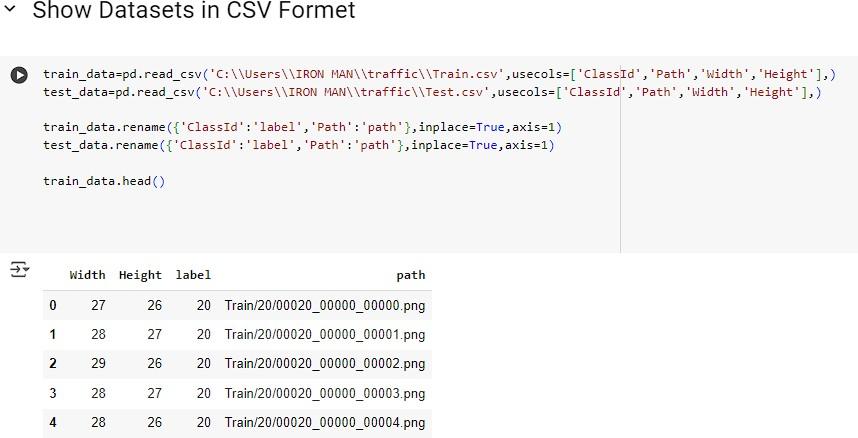
**To implement CNN Deep Learning Applications like**

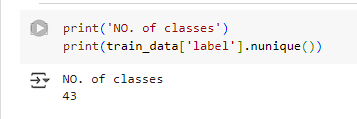
1. **Traffic Signs Recognition:**









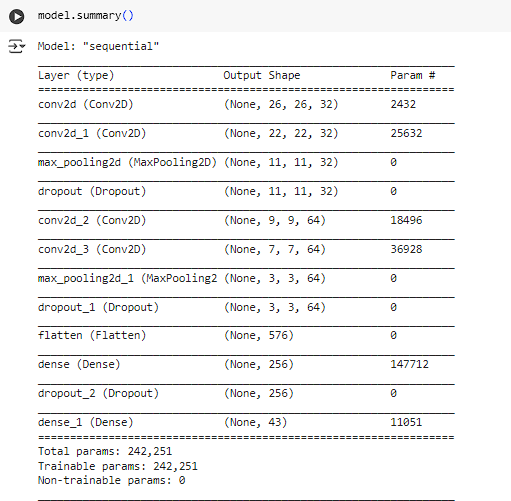


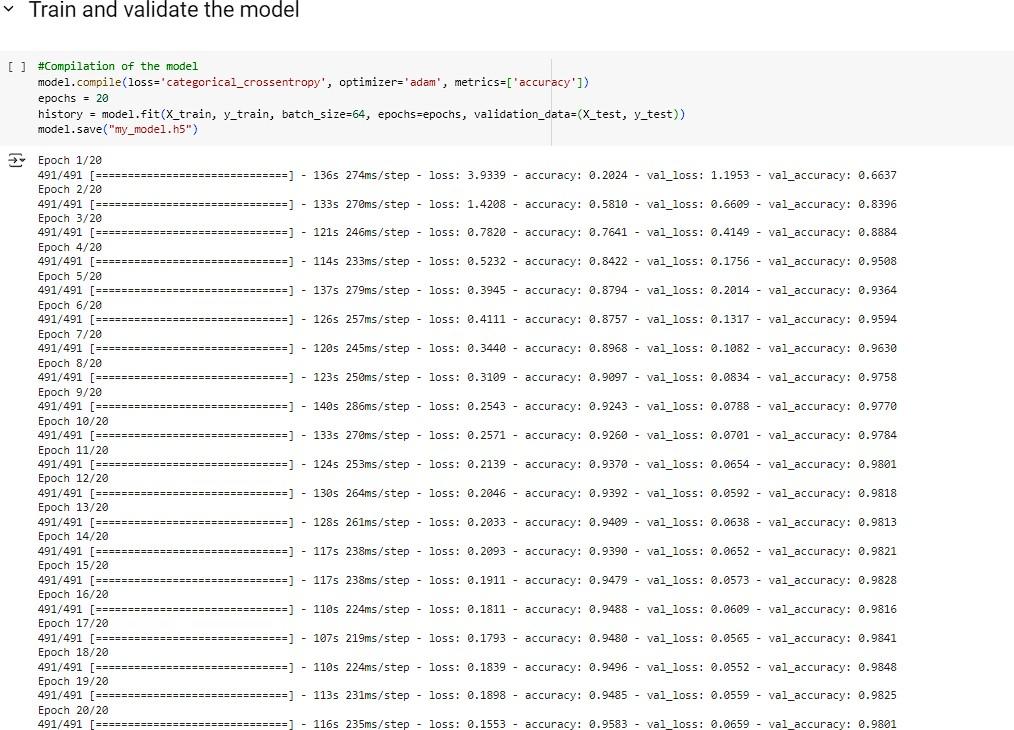
**DATA:**

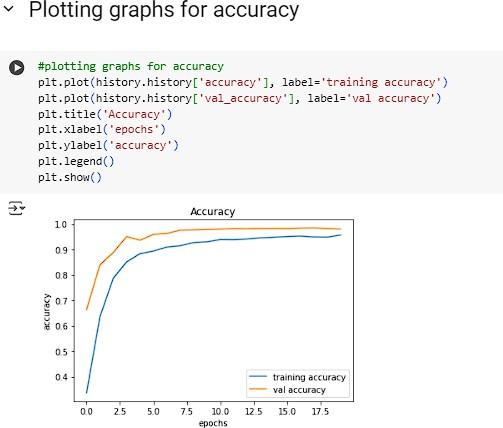


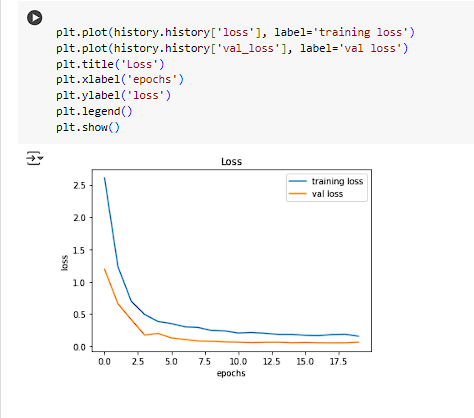


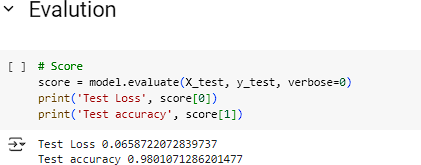












**Conclusion:** Therefore, CNN-based deep learning applications offer efficient and accurate solutions for image classification, handwritten digit recognition, and traffic sign recognition tasks.