Image Classification Using CNN (CIFAR-10 Dataset)

(GROUP – 9)

Presented by: :

- SATYAJEET SAHU
- ANIRBAN DUTTA
- SATYAM DIBYAJYOTI NAYAK

Content:

- ﷺ CIFAR-10: 60,000 images (32×32 px) across 10 classes (e.g., cat, dog, car, plane).
- ﷺ CNNs: Deep learning models ideal for image pattern recognition.
- ﷺ Project Focus: Load data, build CNN, train, evaluate, and predict.
- ﷺ Goal: Classify images accurately using a custom CNN in TensorFlow/Keras.

- ❑ **Data Preparation**: Load and normalize CIFAR-10 images
- ❑ **Model Design**: Build a CNN using TensorFlow & Keras
- ❑ **Training & Evaluation**: Train the model and check accuracy
- ❑ **Result Analysis**: Visualize performance and predictions

Environment Setup :

Platform :

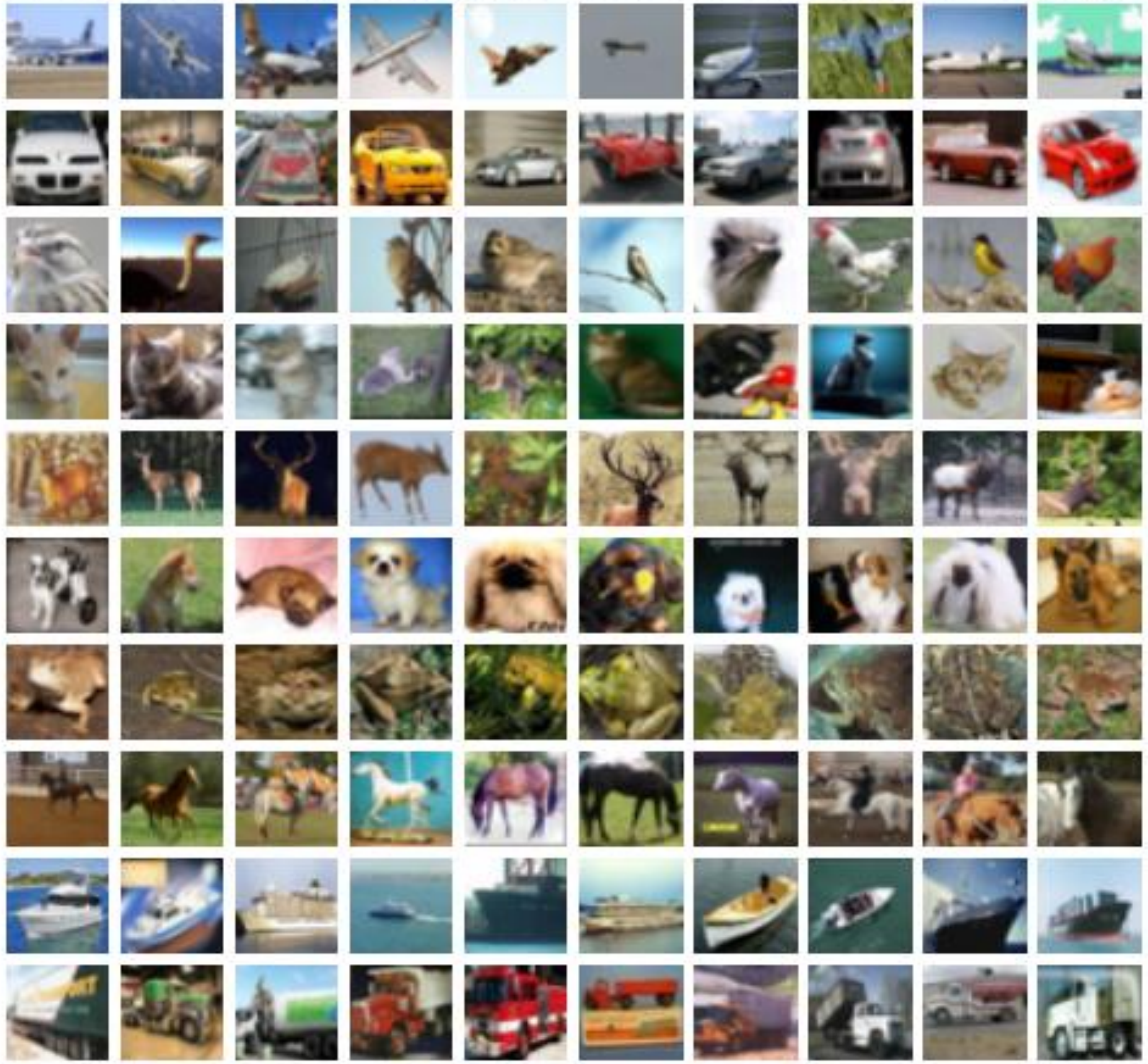
Jupyter Notebook with GPUs: T4, V100, A100

Libraries :

- TensorFlow
- Numpy
- Matplotlib

Here are the classes in the dataset, as well as 10 random images from each:

#0	airplane
#1	automobile
#2	bird
#3	cat
#4	deer
#5	dog
#6	frog
#7	horse
#8	ship
#9	truck



Data Preprocessing & Visualization

Normalization

(images / 255)

Visualization

Plot sample images with labels

Code Architecture

- **tensorflow / keras**: Core deep learning libraries used to build and train the CNN model.
- **numpy**: Used for numerical operations and array handling.
- **matplotlib.pyplot**: For visualizing images from the dataset.
- **cifar10.load_data()**: Loads the CIFAR-10 dataset, splitting it into training and testing sets.
- **x_train, y_train / x_test, y_test**: Variables that hold training and testing data and their corresponding labels.
- **x_train.shape / x_test.shape**: Used to check the dimensions of the data.
- **plt.imshow()**: Displays image data visually.
- **x_train / 255.0, x_test / 255.0**: Normalizes pixel values to the [0, 1] range for better model performance.
- **Sequential()**: Initializes a linear stack of layers for the CNN model.
- **Conv2D(filters, kernel_size, activation)**: Adds a 2D convolutional layer that extracts features from input images.
- **MaxPooling2D(pool_size)**: Reduces the spatial dimensions of the feature maps.

- **Flatten()**: Converts the 2D feature maps into a 1D vector for input into dense layers.
- **Dense(units, activation)**: Fully connected layer that performs classification based on extracted features.
- **model.compile()**: Configures the model with optimizer, loss function, and evaluation metric.
- **optimizer='adam'**: Optimizer that adjusts learning rate adaptively during training.
- **loss='sparse_categorical_crossentropy'**: Suitable loss function for multi-class classification with integer labels.
- **metrics=['accuracy']**: Tracks the accuracy of the model during training and evaluation.
- **model.fit(x_train, y_train, epochs, validation_data)**: Trains the model using training data over multiple iterations (epochs).
- **model.evaluate(x_test, y_test)**: Evaluates model performance on unseen test data.
- **model.predict(x_test)**: Predicts class probabilities for the test images.
- **np.argmax()**: Converts prediction probabilities into final class labels.

1 Importing Necessary Libraries

```python
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

2 Loading the Dataset

```python
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 1113s 7us/step

(50000, 32, 32, 3)
```

```python
# Traing sample : 50000
# Each Sample :32 , 32
# RGB Channel : 3
```

3 Normalization of dataset

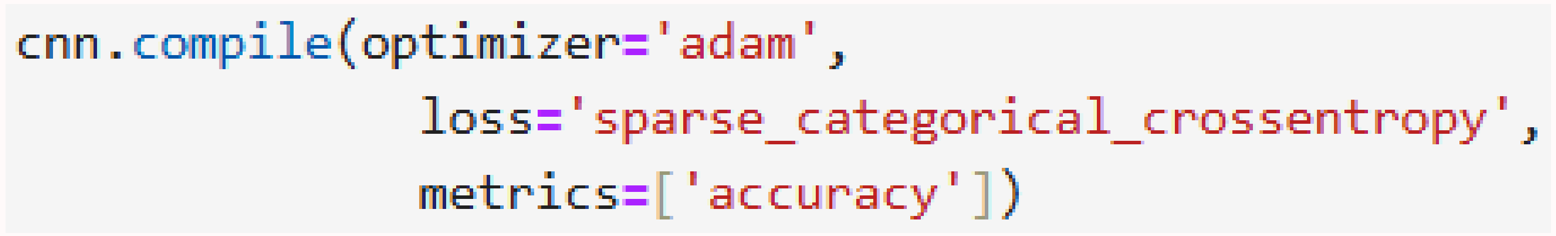
```python
X_train = X_train / 255.0
X_test = X_test / 255.0
#the values are now normalized
```

4 Defining the CNN Architecture

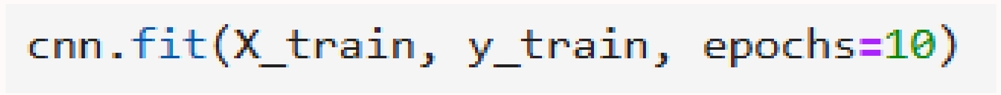
```python
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # softmax normalized the pobability (activation method)
])
```

5

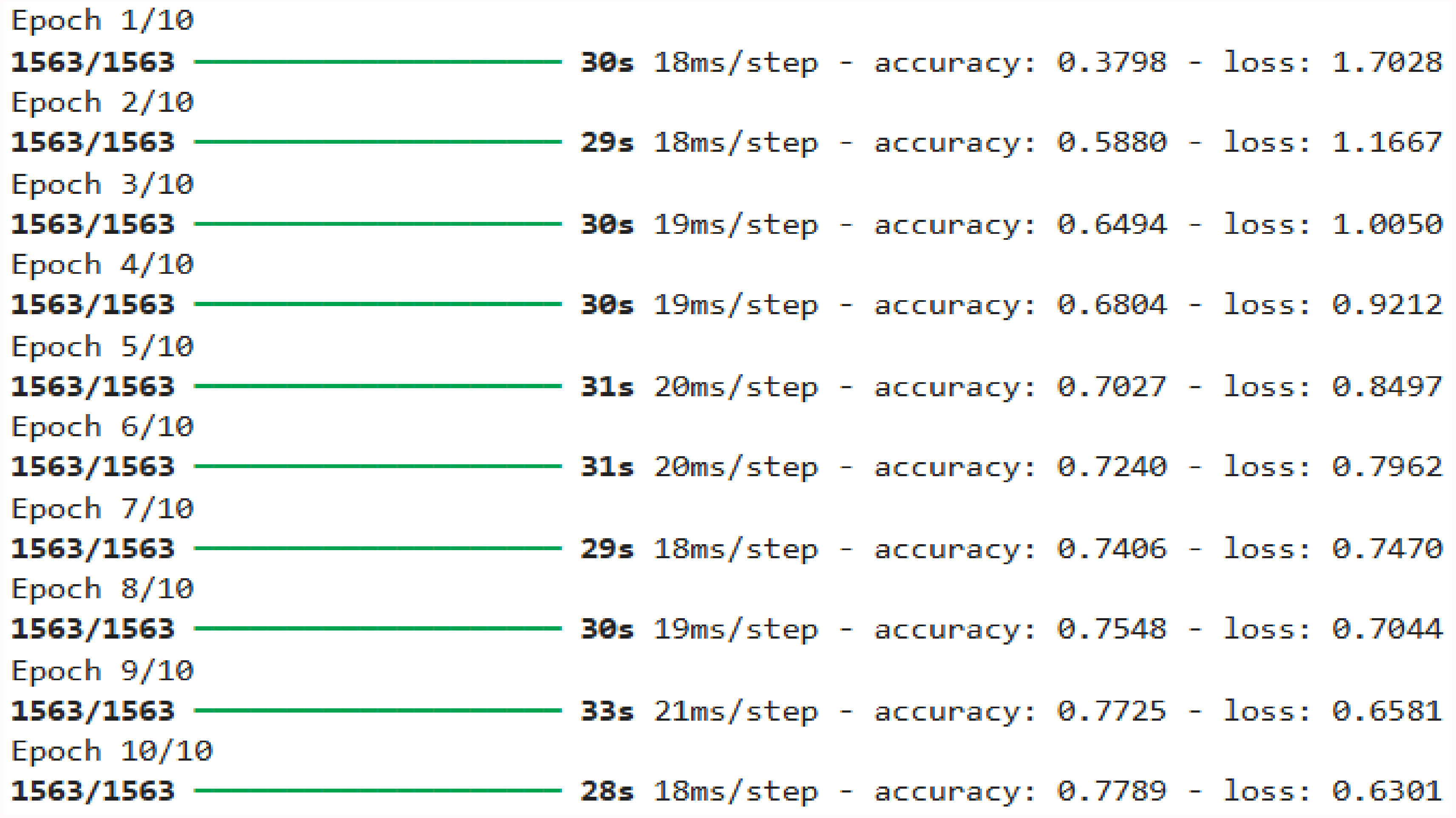
Compiling and Training the Model



```
cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```



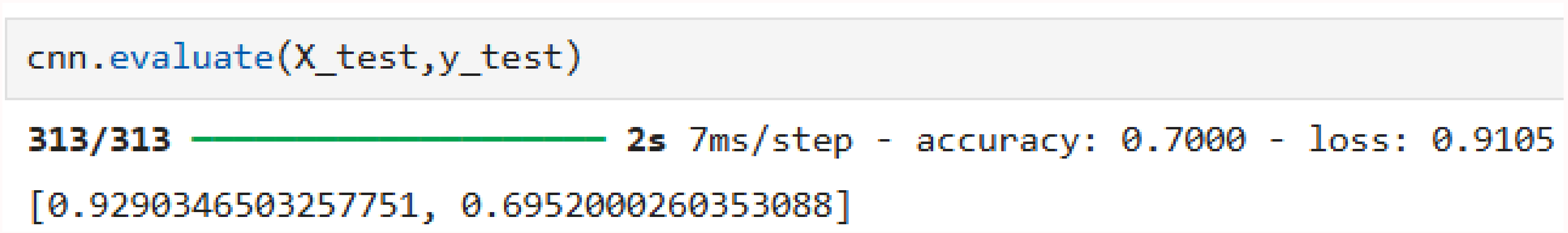
```
cnn.fit(X_train, y_train, epochs=10)
```



```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 30s 18ms/step - accuracy: 0.3798 - loss: 1.7028
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.5880 - loss: 1.1667
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 30s 19ms/step - accuracy: 0.6494 - loss: 1.0050
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 30s 19ms/step - accuracy: 0.6804 - loss: 0.9212
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 31s 20ms/step - accuracy: 0.7027 - loss: 0.8497
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 31s 20ms/step - accuracy: 0.7240 - loss: 0.7962
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 29s 18ms/step - accuracy: 0.7406 - loss: 0.7470
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 30s 19ms/step - accuracy: 0.7548 - loss: 0.7044
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 33s 21ms/step - accuracy: 0.7725 - loss: 0.6581
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 28s 18ms/step - accuracy: 0.7789 - loss: 0.6301
```

6

Evaluating Model Performance

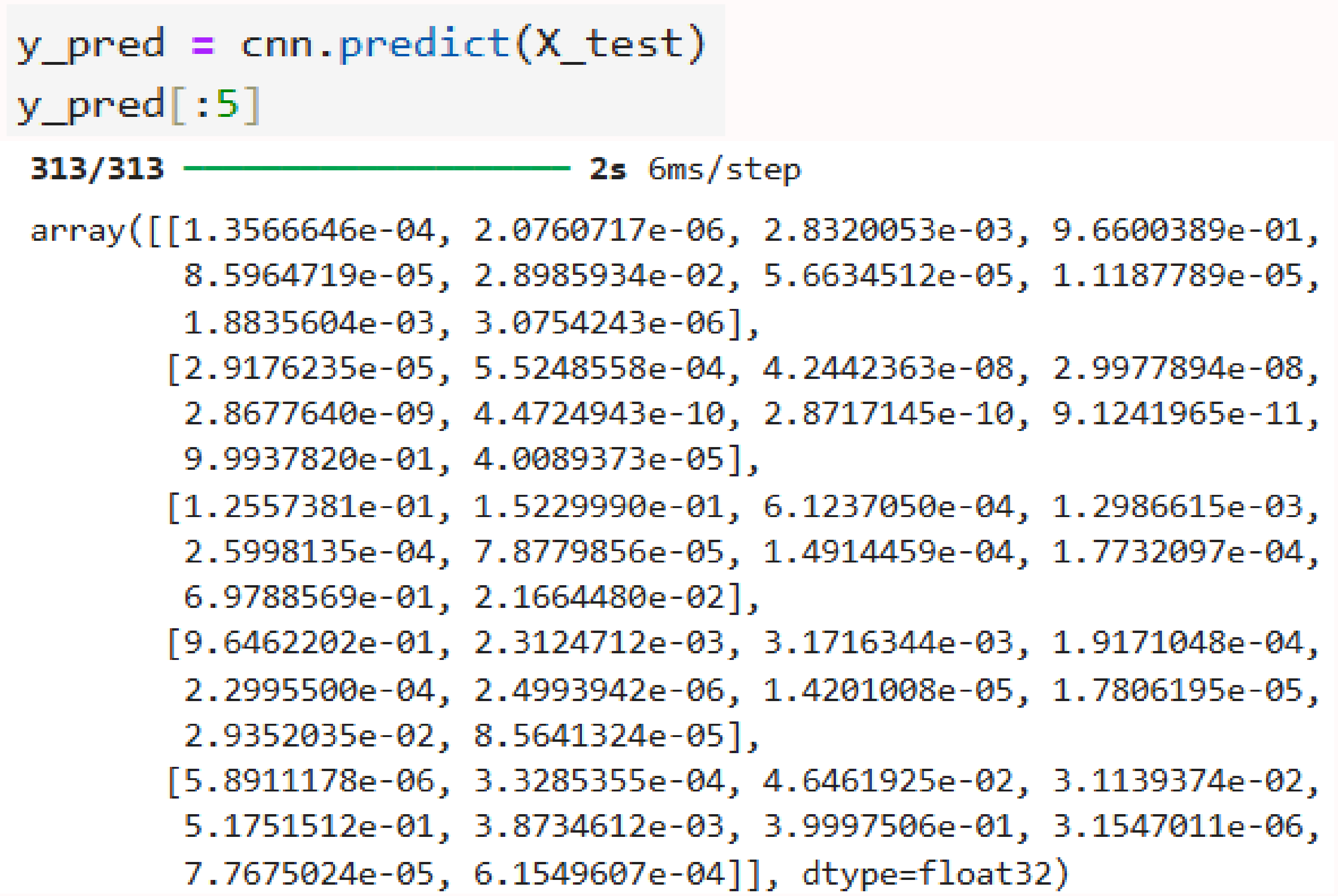


```
cnn.evaluate(X_test,y_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.7000 - loss: 0.9105
[0.9290346503257751, 0.6952000260353088]
```

7

Model Predictions



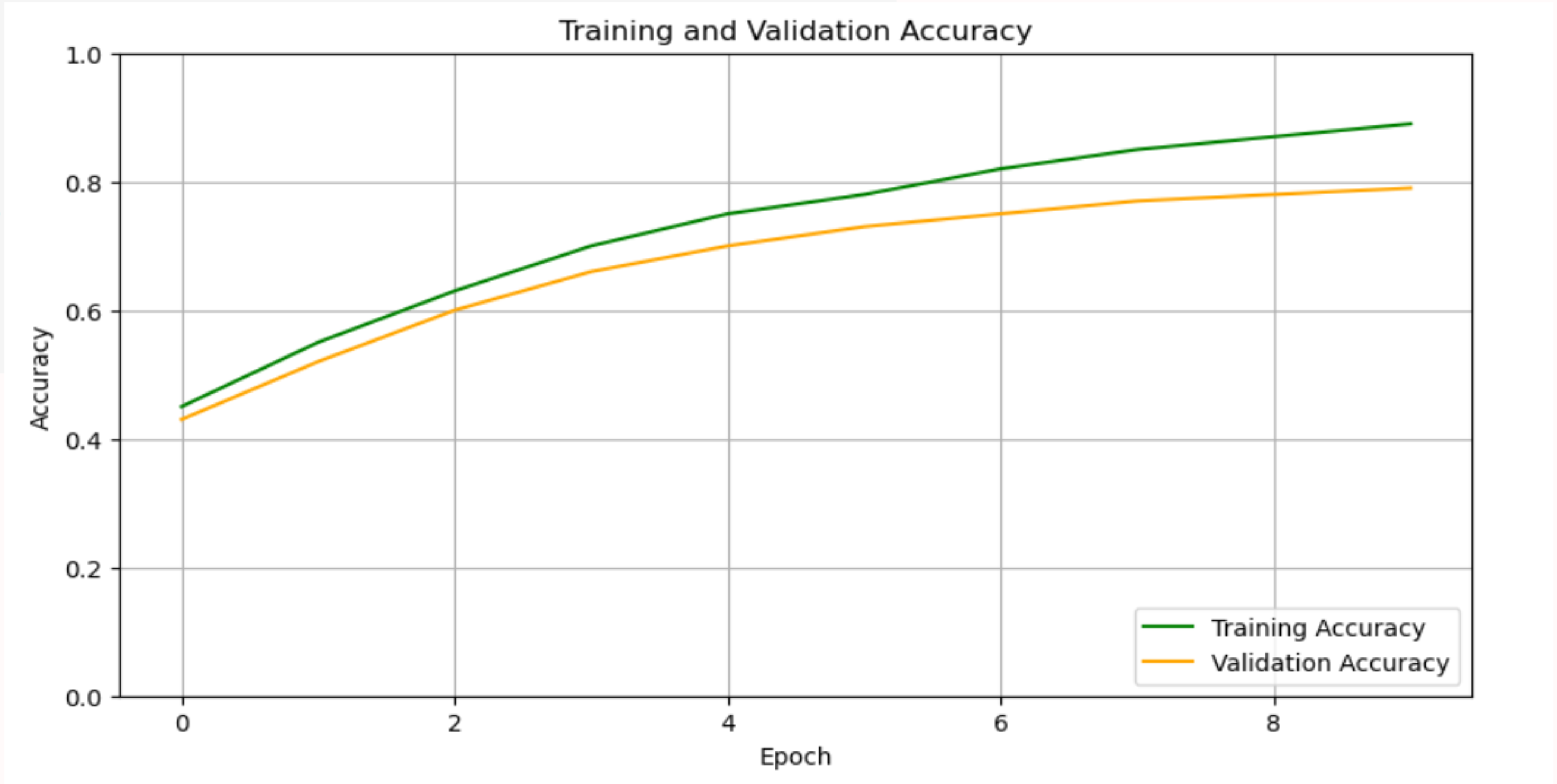
```
y_pred = cnn.predict(X_test)
y_pred[:5]
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
array([[1.3566646e-04, 2.0760717e-06, 2.8320053e-03, 9.6600389e-01,
        8.5964719e-05, 2.8985934e-02, 5.6634512e-05, 1.1187789e-05,
        1.8835604e-03, 3.0754243e-06],
       [2.9176235e-05, 5.5248558e-04, 4.2442363e-08, 2.9977894e-08,
        2.8677640e-09, 4.4724943e-10, 2.8717145e-10, 9.1241965e-11,
        9.9937820e-01, 4.0089373e-05],
       [1.2557381e-01, 1.5229990e-01, 6.1237050e-04, 1.2986615e-03,
        2.5998135e-04, 7.8779856e-05, 1.4914459e-04, 1.7732097e-04,
        6.9788569e-01, 2.1664480e-02],
       [9.6462202e-01, 2.3124712e-03, 3.1716344e-03, 1.9171048e-04,
        2.2995500e-04, 2.4993942e-06, 1.4201008e-05, 1.7806195e-05,
        2.9352035e-02, 8.5641324e-05],
       [5.8911178e-06, 3.3285355e-04, 4.6461925e-02, 3.1139374e-02,
        5.1751512e-01, 3.8734612e-03, 3.9997506e-01, 3.1547011e-06,
        7.7675024e-05, 6.1549607e-04]], dtype=float32)
```

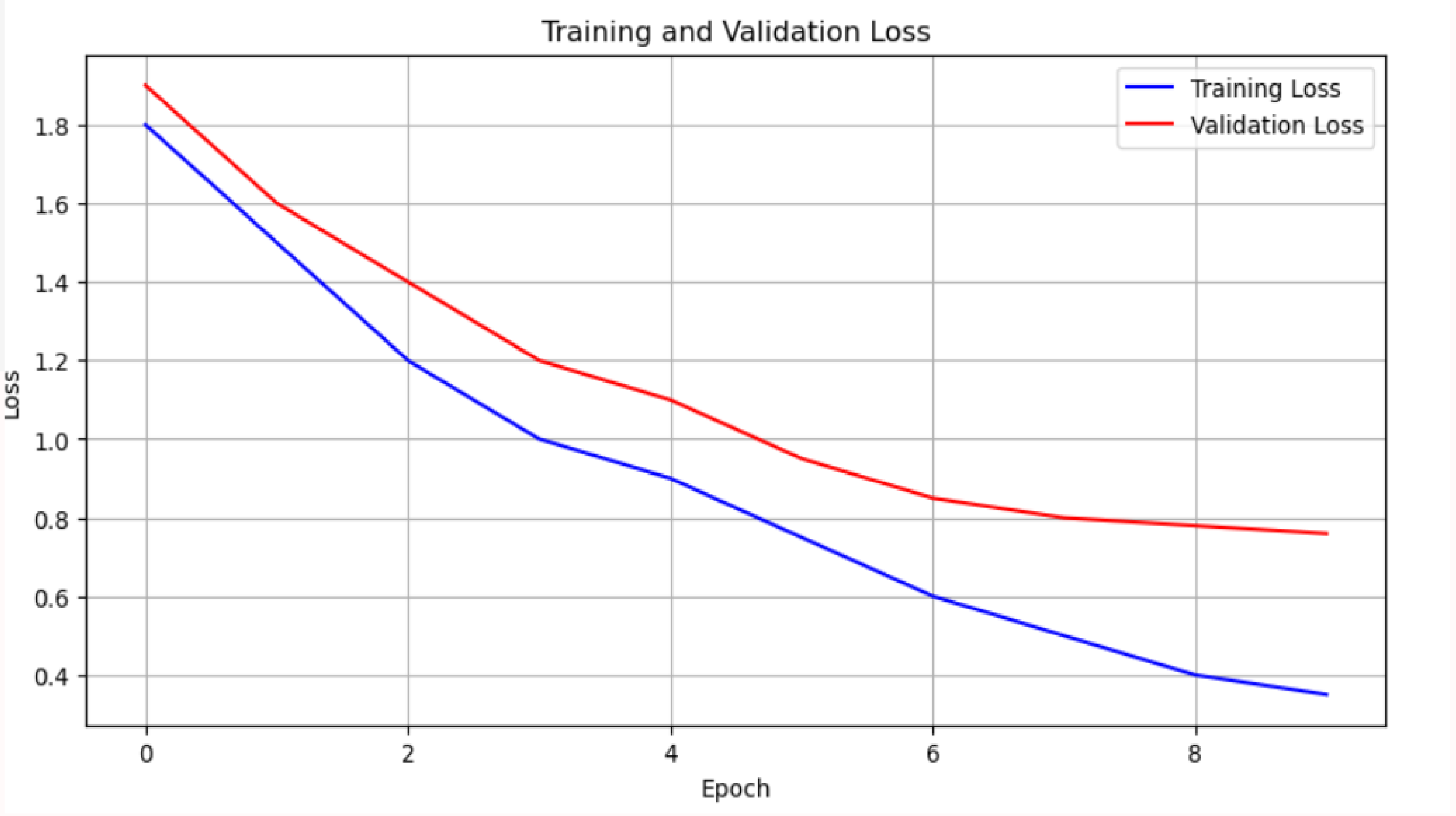
9

Visualizing Training History (GRAPH)

```
# Visualize Accuracy
plt.figure(figsize=(10, 5))
plt.plot(history['accuracy'], label='Training Accuracy', color='green')
plt.plot(history['val_accuracy'], label='Validation Accuracy', color='orange')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



```python
# Visualize Loss
plt.figure(figsize=(10, 5))
plt.plot(history['loss'], label='Training Loss', color='blue')
plt.plot(history['val_loss'], label='Validation Loss', color='red')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```



Summary :

- The model achieved a **test accuracy of ~70%**, indicating it correctly classified over 70% of unseen images.
- Final test loss was **0.9105**, which suggests room for further optimization.
- Future improvements could include:
- Using data augmentation to improve generalization
- Adding dropout layers or batch normalization
- Experimenting with more complex architectures like ResNet or VGG

Thanks for taking the time to view this project on image classification using CNNs.
Feel free to reach out with any questions or suggestions!
Happy Learning!