**IMAGE CLASSIFICATION(Using CNN)**

**A PROJECT REPORT**

*In partial fulfilment of the requirements for the award of the degree*

## BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE ENGINEERING

*Under the guidance of*

**SHOUVIK SARKAR**

**BY**

**PARSHANT KUMAR**

# INDIAN INSTITUTE of INFORMATION TECHNOLOGY, GUWAHATI

**In association with**



#### 

|  |  |  |
| --- | --- | --- |
| 1. | Title of the Project: | **IMAGE CLASSIFICATION (Using CNN)** |
| 2. | Project Members: | **PARSHANT KUMAR** |
| 3. | Name of the guide: | **Mr. SHOUVIK SARKAR** |
| 4. | Address: | Ardent Computech Pvt. Ltd (An ISO 9001:2015 Certified)  A-1/20, Ramgarh, Ganguly Bagan, Kolkata,  West Bengal 700047 |

(Note: All entries of the proforma of approval should be filled up with appropriate and complete information. Incomplete proforma of approval in any respect will be rejected.)

**Project version control History:-**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Members** | **Description of version** | **Date Completed** |
| **1.0** | **Parshant kumar** | Which dataset to use.  (either to pick the random images and transform them to array using [Matplotlib](https://matplotlib.org/stable/tutorials/introductory/images.html#image-tutorial) or pick the CIFAR-10 or CIFAR-100 dataset) | 10th June, 2023 |
| **1.1** | **Parshant kumar** | Preprocessing steps like,  Scaling of the data  **For example,**  X\_train = X\_train / 255.0  And handling the categorical variables using one hot encoding. | 11th June, 2023 |
| **2.0** | **Parshant kumar** | **Introducing the dropout layers in between the convolutional layers in order to overcome the overfitting of model on the data provided.** | 14th June, 2023 |
| **3.0** | **Parshant kumar** | **Final project.**  **Final evaluation and testing of the model and the accuracy of approx. 84% after iterating over 26-30 epochs.** | 16th June, 2023 |

**Signature of Team Member** **Signature of Approver**

Date: Date:

For Office Use Only

**Approved**

**SHOUVIK SARKAR**

**Not Approved**

Evaluator Project Proposal

# DECLARATION

I hereby declare that the project work being presented in the project proposal entitled **“IMAGE CLASSIFICATION (Using CNN)”** in partial fulfilment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** at **ARDENT COMPUTECH PVT. LTD, JADAVPUR, KOLKATA, WEST**

**BENGAL,** is an authentic work carried out under the guidance of **MR. SHOUVIK SARKAR**. The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

Date: 6th June, 2023 **Signature of thestudent**

Name of the Student: **PARSHANT KUMAR**



**Ardent ComputechPvt. Ltd (An ISO 9001:2015 Certified)**

# CERTIFICATE

This is to certify that this proposal of minor project entitled **“IMAGE CLASSIFICATION (Using CNN)”** is a record of bonafied work, carried out by **PARSHANT KUMAR** under my guidance at **ARDENT COMPUTECH PVT LTD**. In my opinion, the report in its present form is in partial fulfilment ofthe requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** and as per regulations of the **ARDENT*®.*** To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report.

**Guide / Supervisor**

## MR. SHOUVIK SARKAR



#### Ardent ComputechPvt. Ltd (An ISO 9001:2015 Certified)

# ACKNOWLEDGEMENT

Success of any project depends largely on the encouragement and guidelines of many others. I take this sincere opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project work.

I would like to show our greatest appreciation to ***Mr. SHOUVIK SARKAR***, Project Engineer at Ardent, Kolkata. I always feel motivated and encouraged every timeby his valuable advice and constant inspiration; without his encouragement and guidance this project would not have materialized.

Words are inadequate in offering our thanks to the other trainees, project assistants and other members at Ardent ComputechPvt. Ltd. for theirencouragement and cooperation in carrying out this project work. The guidance and support received from all the members and who are contributing to this project, was vital for the success of this project.

**INDEX**

* **OVERVIEW**
* **HISTORY OF PYTHON**
* **FEATURES OF PYTHON**
* **OBJECTIVE**
* **INTRODUCTION TO ML**
* **DEFINITION OF MACHINE LEARNING**
* **IMPORTANCE OF MACHINE LEARNING**
* **PROJECT ORIENTED**
* **MACHINE LEARNING ALGORITHMS**
* **COMPONENTS**
* **NumPy**
* **Pandas**
* **TensorFlow and Keras**
* **Scikit-learn**
* **STEPS INVOLVED IN MODEL CREATION**
* **PROJECT CODE**
* **RESULTS**
* **REFERENCES**

# OVERVIEW

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and has fewer syntactical constructions than other languages.

**Python is interpreted**: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to Perland PHP.

**Python is Interactive**: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

**Python is Object-Oriented**: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

**Python is a Beginner's Language**: Python is a great language for the beginner- level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

# HISTORY OF PYTHON

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, UNIX shell, and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

**FEATURES OF PYTHON**

Easy-to-learn: Python has few Keywords, simple structure and clearly defined syntax. This allows a student to pick up the language quickly.

Easy-to-Read: Python code is more clearly defined and visible to the eyes.

Easy -to-Maintain: Python's source code is fairly easy-to-maintain.

**A broad standard library:** Python's bulk of the library is very portable and cross platform compatible with UNIX, Windows, and Macintosh.

**Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

**Portable:** Python can run on the wide variety of hardware platforms and has the same interface on all platforms.

**Extendable:** You can add low level modules to the python interpreter. These modules enables programmers to add to or customize their tools to be more 0efficient.

**Databases:** Python provides interfaces to all major commercial databases.

**GUI Programming**: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

**Scalable:** Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

* It supports functional and structured programming methods as well as OOP.
* It can be used as a scripting language or can be compiled to byte code for building large applications.
* It provides very high level dynamic datatypes and supports dynamic type checking.
* It supports automatic garbage collections.
* It can be easily integrated with C, C++, COM, ActiveX, CORBA and JAVA.

**OBJECTIVE**

The objective of this project is to accurately classify and label images into predefined categories or classes. CNNs are specifically designed to analyze visual data and have become the go-to approach for image classification tasks due to their ability to automatically learn relevant features directly from raw pixel data.

Our motive is to design an efficient model using convolutional neural networks which can predict the given image.

**INTRODUCTION TO ML**

**Definition of Machine Learning:** Machine learning is a subfield of artificial intelligence that focuses on developing algorithms and models that allow computers to learn from data and make predictions or decision without being explicitly programmed. It involves creating mathematical models and leveraging statistical techniques to enable computers to automatically improve their performance on a specific task through experience or training.

Mathematics plays a fundamental role in machine learning as it provides the theoretical foundation and tools necessary to understand and develop these algorithms. Some key mathematical concepts and techniques in machine learning include:

* **Linear Algebra:** Linear Algebra is used to represent and manipulate high-dimensional data, such as images or text, as vectors and matrices. Operations like matrix multiplication, vector dot products, and eigenvalue decomposition are frequently used in machine learning algorithms.
* **Calculus:** Calculus is used to optimize machine learning models and algorithms. Gradient descent, a commonly used optimization algorithm, relies on derivatives from calculus to iteratively update model parameters and minimize the loss function.
* **Probability and Statistics:** Probability theory and statistical analysis are essential for understanding uncertainty, estimating parameters, ad making predictions in machine learning. Concepts such as probability distributions, hypothesis testing, regression analysis, and Bayesian inference are commonly employed in various machine learning algorithms.
* **Optimization:** Optimization techniques are used to find the best possible values for the parameters of a machine learning model. Techniques like gradient descent, stochastic gradient descent, and convex optimization algorithms aim to minimize the error or loss function and improve the model’s performance.
* **Algorithms and Algorithms Analysis:** Machine learning involves the development and analysis of algorithms that can efficiently process and learn from large datasts. Understanding algorithm complexity, time complexity, and space complexity is crucial for assessing the scalability and efficiently of machine learning algorithms.

Mathematics forms the foundation of many machine learning algorithms, allowing researchers and practitioners to create models that can learn from data, generalize to unseen examples, and make accurate predictions. The mathematical principles and techniques enable the formulation and optimization of models that effectively capture patters and relationships within the data, leading to improved performance and insights.

**Importance of Machine Learning:** Machine learning has become increasingly important in various domains and industries due to its ability to analyze large amounts of data, discover patterns, and make prediction or decisions. It has revolutionized fields such as healthcare, finance, marketing, image and speech recognition, recommendation systems, and autonomous vehicles.

Machine learning enables organizations to:

* **Extract Insights:** By leveraging machine learning techniques, businesses can extract valuable insights form vast amounts of data. This allows them to identify patterns, trends, and correlations that may not be apparent through traditional analysis methods.
* **Enhance Decision Making:** ML models can process and analyze complex data to provide informed recommendations and support decision-making processes. This leads to more accurate and data-driven decisions that can improve efficiency and outcomes.
* **Improve Efficiency and Automation:** ML enables automation of various tasks and processes, reducing manual effort and increasing efficiency. It can automate repetitive and time- consuming tasks, allowing human resources to focus on more complex ans strategic activities.
* **Personalize Experiences:** ML powers personalized experiences by analyzing user behavior, preferences and historical data. This enables businesses to deliver tailored recommendations, targeted advertising, and personalized product offerings, enhancing customer satisfaction and engagement.
* **Predictive Analytics:** ML algorithms can analyze historical data to make predictions and forecasts about future outcomes. This is particularly valuable for tasks such as demand forecasting, risk assessment, fraud detection, and predictive maintenance.
* **Adaptive Systems:** ML models can adapt and improve over time as they receive new data. This adaptability enables systems to continuously learn from user interactions, feedback, and changing environments, leading to improve performance and accuracy.

The importance of machine learning lies in its ability to leverage the power of data and algorithms to extract valuable inisights, make accurate predictions, automate processes and drive informed decision-making. By harnessing ML techniques, organizations can gain a competitive edge, optimize operations and unlock new opportunities in the ever-expanding digital landscape.

**Project Oriented:** Convolutional Neural Networks (CNNs) are designed to analyze visual data and have become the go-to approach for image classification tasks due to their ability to automate learn relevant features directly from raw pixel data.

CNNs are a type of deep learning model inspired by the organization of the visual cortex in animals. They consist of multiple layers, including convolutional layers, pooling layers and fully connected layers. These layers work together to extract meaningful features from image and make accurate predictions.

* **Convolutional Layers:** Convolutional layers apply filters or kernels to input images, performing local operations that capture spatial patterns and features. These filters convolve over the image, computing dot products between the filter weights and image pixels, producing features maps that highlight important visual patterns.
* **Pooling Layers:** Pooling layers reduce the spatial dimensions of feature maps generated by convolutional layers. They downsample the feature maps by aggragating nearby values, such as taking the maximum or average value within a region. Pooling helps to extract the most important features and achieve translation invariance, making the model more robust to variation in object position or orientation.
* **Fully Connected Layers:** Fully connected layers take the extracted feature from the previous layers and map them to the output classes. These layers connect every neuron in one layer to every neuron in the next layer, allowing the model to learn complex relationships and make predictions.

Mathematically, CNNs and neural networks rely on linear algebra operations, such as matrix multiplications, dot products, and elementwise operations. These operations are used to compute the activations and transformations that take place in each layer of the network.

During training, CNNs use a process called backpropagation, which involves calculating gradient of the loss function with respect ot the model’s parameters. This allows the mode to adjust its parameters iteratively using optimization algorithms, such as gradient descent, to minimize the difference between predicted and true labels.

The power of CNNs lies in their ability to automatically learn hierarchical representations of visual data through the convolutional and pooling layers. This hierarchical learning enables them to capture low-level features, such as edges and textures, and progressively build more abstract and complex representations, improving the model’s ability to classify images accurately.

**Some additional points:**

* **Activation Functions:** Activation functions introduce non-linearity to the neural network, enabling it to learn complex relationships between input data and output predictions. Common activation functions used in CNNs include ReLU (Rectified Linear Unit), sigmoid, and tanh. These functions help the network learn and model non-linear mappings between image features and class labels.
* **Loss Functions:** Loss functions quantify the discrepancy between predicted and true labels during training. Different loss functions can be used based on the nature of the classification problem, such as cross-entropy loss for multi-class classification tasks. The choice of an appropriate loss function affects the optimization process and the model’s ability to converge to accurate predictions.
* **Optimization Algorithms:** Optimization algorithms, such as gradient descent and its variants (e.g., Adam, RMSprop), are used to update the model’s parameters during training. These algorithms iteratively adjust the weights and biases of the network based on the calculated gradients of the loss function, aiming to minimize the loss and improve the model’s accuracy.
* **Dropout and Regularization:** Dropout is a regularization technique commonly used in neural networks, including CNNs. It randomly drops out a certain percentage of neurons during training, preventing overfitting and enhancing the generalization ability of the model. Regularization techniques like L1 and L2 regularization can also be applied to control model complexity and prevent overfitting.
* **Pretrained Models and Transfer Learning:** Pretrained models, such as VGG, ResNet, or Inception, are CNN architectures that have been trained on large datasets like ImageNet. They have learned a wide range of image features and can be used as a starting point for your image classification task. Transfer learning involves fine-tuning these pretrained models by adjusting their parameters to adapt to your specific classification problem, potentially improving performance even with limited data.
* **Data Augmentation:** Data augmentation techniques are used to artificially increase the diversity and size of the training dataset. By applying random transformations such as rotations, flips zooms, or translations to the images, the model is exposed to a large variety of data and become more robust to different variations and presepectives.
* **Interpretabiltiy and Explainability:** CNNs are often regarded as “black box” models due to their complexity. However, there are techniques, such as gradient-based class activation maps (CAM) or saliency maps, that provide visual explanations decision. These techniques can help understand na interpret the model’s predictions.

**Machine Learning Algorithms:** Machine learning algorithms are computational procedures that learn patterns and relationships from data, allowing computers to make predictions or take actions based on that learned knowledge. There are several types of machine learning algorithms:

1. **Supervised Learning Algorithms:** These algorithms learn from labeled training data, where each data point is associated with a corresponding label or target value. Common supervised learning algorithms include:

* **Support Vector Machines (SVM):** SVMs aim to find the best hyperplane that separates different classes in a high-dimensional space.
* **Decision Trees:** Decision trees create a flowchart-like structure that splits the data based on different attribute values to make predictions.
* **Random Forest:** It is an ensemble method that combines multiple decision trees to improve the accuracy and robustness of predictions.
* **Gradient Boosting:** Gradient boosting builds an ensemble of weak prediction models, sequentially improving upon the errors made by previous models.
* **Naïve Bayes:** Naïve Bayes algorithms apply Bayes’ theorem and assume independence among features to make probabilistic prediction.

1. **Unsupervised Learning:** These algorithms learn from unlabeled data and discover patterns or grouping without explicit target labels. Common unsupervised learning algorithms include:

* **K-means Clustering:** K-means Clustering builds a hierarchy of clusters based on similarity or distance measures.
* **Hierarchical Clustering:** Hierarchical clustering builds a hierarchy of clusters by recursively merging or splitting them based on similarity.
* **Principal Component Analysis (PCA):** PCA reduces the dimensionality of data by identifying orthogonal axes that capture the most variance.
* **Generative Adversarial Networks (GAN):** GANs consist of two competing neural networks, a generator and a discriminator, that learn to generate realistic synthetic data.

For image classification, the most commonly used algorithm is Convolutional Neural Networks (CNNs). CNNs are specifically designed to analyze visual data and have demonstrated remarkable performance in image recognition tasks. They are well-suited for capturing hierarchical and spatial patterns within images, making them highly effective for image classification problems.

CNNs have become the state-of-the-art approach for image classification due to their ability to automatically learn relevant features directly from raw pixel data, leveraging convolutional and pooling layers. They are capable of capturing local patterns, detecting edges, textures, and more complex visual structures, ultimately leading to accurate and robust image classification.

Therefore, for this project, CNNs are the best choice.

**COMPONENTS**

**NUMPY(NUMERIACAL PYTHON)**: NumPy is a fundamental library for scientific computing in Python. It provides a powerfull array object and functions for working with large, multi-dimensional arrays and matrices. NumPy’s array object, called ndarray, enables efficient numerical operations and supports various mathematical functions. It also offers tools for linear algebra, Fourier transforms, random number generation, and more. NumPy is a core dependency for many other scientific and machine learning libraries in Python.

**PANDAS**: Pandas is a library build on a top of NumPy that provides high-performance data manipulation and analysis tools. It introduces two primary data structures: the ‘Series’, which is a one-dimensional labeled array capable of holding any data type, and the ‘DataFrame’, which is a two-dimensional labeled data structure resembling a table or a spreadsheet. Pandas simplifies data cleaning, transformation, exploration, and analysis tasks by offering a wide range of functions and methods. It also integrates well with other machine learning libraries, making it popular choice for data preprocessing and data wrangling tasks.

**TENSORFLOW AND KERAS**: TensorFlow is an open-source machine learning framework developed by Google. It is widely used for building an training machine learning models, including deep learning models, with a focus on numerical computation graph representation, allowing you to define and execute complex mathematical operations efficiently.

1. **Low-level operations:** TensorFlow provides a wide range of low-level operations and functions that are essential for image processing and manipulation. These include operations for image loading, resizing, cropping, and transformations. TensorFlow’s computational graph enables efficient execution of these operations on large datasets.
2. **TensorFlow’s Estimator API:** It provides a high-level interface for defining and training machine learning models. For image classification, you can use the Estimator API to define your model architecture, specify the input pipeline for loading and preprocessing images, and configure training parameters. The Estimator API simplifies the training process by handling aspects such as batching, shuffling, and data parallelism automatically.
3. **TensorFlow Hub:** It is a library and platform that allows you to easily discover, share, and reuse pre-trained models and their components. It provides a collection of pre-trained models specifically designed for image classification tasks. You can leverage these models as a starting point and fine-tune them on your specific dataset, saving significant time and computational resources.
4. **Integration with Keras:** TensorFlow integrates with Keras, a high-level neural networks API, making it even more convenient for image classification tasks. Keras provide a user-friendly and intuitive interface for building, training, and evaluating deep learning models. It simplifies the process of defining the model architecture, configuring layers, and specifying loss functions and optimizers. Keras also offers a variety of pre-built layers, including those specific to convolutional neural networks(CNNs), which are commonly used in image classification.
5. **GPU support:** It allows you to leverage the power of GPUs to train deep learning models faster. GPYs excel at performing parallel computations, which is beneficial for tasks like image classification that involve processing layers volumes of data and performing numerous mathematical operations simultaneously. TensorFlow seamlessly integrates with GPUs, enabling efficient training and inference on compatible hardware.

**Keras** is an open-source high-level neural networks API that runs on top of TensorFlow (as well as other backend engines line Theano and Microsoft Cognitive Toolkit). It provides a user-friendly interface for designing, training, and evaluating deep learning models.

1. **User-friendly API:** It offers a simple an intuitive API that enables rapid development of deep learning models. It provides a modular approach for defining models, allowing you to stack layers together to create complex architectures. Keras offers a wide range of pre-built layers and activation function that can be easily combined to construct various types of neural networks.
2. **Model Visualization:** Keras provides tools for visualizing model architectures, allowing you to inspect and understand the structure for neural network. You can generate visual representations for you models as graphs, which help in identifying potential issues, such as missing connections or incorrect layer configurations.
3. **Model training and evaluation:** Keras simplifies the process of training and evaluating models. It provides a high-level interface for specifying loss functions, optimizers, and metrics. Keras also offers convenient methods, and performing model evaluation on test datasets.
4. **Transfer Learning:** Keras supports transfer learning, a technique where pre-trained model are used as a starting point for solving new tasks. By leveraging pre-trained models trained on large datasets, you can benefit from the learned features and fine-tune the model on your specific classification problem. Keras integrates with TensorFlow Hub and other pre-trained model repositories, making it easy to import and use pre-trained models for transfer learning.

**tensorflow.keras.models.Sequential:** It is a linear stack of neural network layers, allowing us to build models layer by layer in a sequential manner.

**tensorflow.keras.layers:** It provides various types of layers for construction neural networks, including dense (fully connected) layers, convolutional layers, pooling layers, flatten layers, dropout layers, and batch normalization layers.

**tensorflow.keras.callbacks.EarlyStopping:** It is callback that stops the model training process if a specified metric (such as validation loss) stops improving, preventing overfitting and saving computational resources.

**tensorflow.Keras.preprocessing.image.ImageDataGenerator:** It is a utility for preprocessing and augmenting image data during model training. It provides functionalities like rescaling, resizing, data augmentation, and data shuffling, enhancing the diversity and generalization of the training data.

**tensorflow.keras.datasets.cifar10:** It is a module that provides access to the CIFAR-10 dataset, a popular benchmark dataset for image classification tasks. It allows us to conveniently load the dataset, which consists of 50,000 training and 10,000 test images, divide into 10 classes.

**tensorflow.keras.utils.to\_categorical:** It is a utility function that converts integer labels into one-hot encoded vectors. This function is commonly used for converting class labels into a binary matrix representation, suitable for training classification models. It simplifies the process of encoding categorical data for training neural networks.

**SCIKIT-LEARN:** scikit-learn often abbreviated as sklearn, is a widely used open-source machine learning library for Python. It provides a comprehensive collection of tools and algorithms for tasks such as classification, regression, clustering, dimensionality reduction, and model evaluation.

The library follows a consistent API design, making it easy to use and integrate into machine learning workflows. It provides efficient implementations of various algorithms and techniques, including preprocessing, feature selection, model selection, and model evaluation. Additionally, scikit-learn supports interoperability with other scientific Python libraries such as NumPy and pandas.

1. **ConfusionMatrixDisplay:** It is a class within the ‘**sklearn.metrics**’ module that allows for the visualization of a confusion matrix. The confusion matrix is a tabular representation that show the performance of a classification mode by comparing predicted labels against actual labels. The ‘ConfusionMatrixDisplay’ class provides methods to plot the confusion matrix in a visually informative way, allowing for easier interpretation and analysis of the model’s performance.
2. **classification\_report:** It is a function from ‘**sklearn.metrics**’ that generates a comprehensive report summarizing the performance of a classification model. The ‘classification\_report’ function calculates various evaluation metrics, including precision, recall, F1-score, and support, for each class in the classification problem. It also provides overall metrics such as accuracy and macro/micro averages, giving a detailed overview of the model’s performance on different classes and aiding in understanding its strengths and weakness.
3. **confusion\_matrix:** It is a function from ‘**sklearn.metrics**’ that computes a confusion matrix based on the predicted labels and true labels of a classification model. The confusion matrix is a square matrix where each row represents the instances in a predicted class, and each column repreents the instancs in an actual class. It helps in understanding the distribution of predictions and misclassifications across different classes, providing insights into the model’s performance and error patterns.

**STEPS INVOLVED IN CREATING THE MODEL**

1. **Import the necessary libraries and modules:** Import TensorFlow, the required layers from **tensorFlow.keras.layers**, and any other relevant libraries.
2. **Load and preprocessing the dataset:** Use the ‘**tensorflow.keras.datasets.cifar10**’ dataset. Perform any necessary preprocessing steps, such as scaling the pixel values, converting labels to categorical form using ‘**tensorflow.keras.utils.to\_categorical**’, and splitting the dataset into training and testing sets.
3. **Create the model architecture:**

* Start by initializing a sequential model using ‘**tensorflow.keras.models.Sequenial()**’.
* Add convolutional layers (‘**tensoflow.keras.layers.Con2D**’) with appropriate filters, kernel sizes, and activation functions.
* Add pooling layers (‘**tensorflow.keras.layers.MaxPool2D**’) to reduce spatial dimensions and capture dominant features.
* Optionally, insert batch normalization layers (‘**tensorflow.keras.layers.BatchNormalization**’) between the convolutional layers to prevent overfitting.
* Flatten the output from the convolutional layers using ‘**tensorflow.keras.layers.Flatten**’ to prepare for the fully connected layers.
* Add dense (fully connected) layers (‘**tensorflow.keras.layers.Dense**’) with appropriate units and activation functions.
* Optionally, add dropout layers between the dense layers to further combat overfitting.
* Finally, add a dense output layer with the number of units equal to the number of classes and an appropriate activation function (e.g., **softmax** for multi-class classification).

1. **Compile the model:** Configure the model for training by specific the loss function (‘**loss**’), optimizer (‘**optimizer**’), and evaluation metric (‘**metrics**’). Common choices include categorical cross-entropy loss, Adam optimizer, and accuracy metric.
2. **Train the model:** Use the ‘**model.fit()**’ method to train the model on the training dataset. Specify the number of epochs, batch size, and any necessary callbacks (e.g., ‘**tensorflow.keras.callbacks.EarlyStopping**’) for easly stopping or model checkpoint.
3. **Evaluate the model:** Use the ‘**model.evaluate()**’ methods to evaluate the trained model on the testing dataset. Obtain metrics such as accuracy, loss, and any other relevant evaluation measures.
4. **Make predictions:** Use the trained model to make predictions on new/unseen images using the ‘**model.predict()**’ method.
5. **Fine-tuning and optimization (optional):** If desired, you can experiment with hyperparameter tuning, model architecture modifications, or techniques like transfer learning to further improve the model’s performance.

**PROJECT CODE:-**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import tensorflow as tf

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNormalization

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.metrics import classification\_report, confusion\_matrix

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

print(f"X\_train shape: {X\_train.shape}")

print(f"y\_train shape: {y\_train.shape}")

print(f"X\_test shape: {X\_test.shape}")

print(f"y\_test shape: {y\_test.shape}")

print(X\_train[:5])

X\_train shape: (50000, 32, 32, 3)

y\_train shape: (50000, 1)

X\_test shape: (10000, 32, 32, 3)

y\_test shape: (10000, 1)

[[[[ 59 62 63]

[ 43 46 45]

[ 50 48 43]

...

[158 132 108]

[152 125 102]

[148 124 103]]

[[ 16 20 20]

[ 0 0 0]

[ 18 8 0]

...

[123 88 55]

[119 83 50]

[122 87 57]]

[[ 25 24 21]

[ 16 7 0]

[ 49 27 8]

...

[118 84 50]

...

...

[ 75 79 82]

[ 71 75 78]

[ 73 77 80]]]]

labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

W\_grid = 10

L\_grid = 10

fig, axes = plt.subplots(L\_grid, W\_grid, figsize = (17,17))

axes = axes.ravel() # flaten the 15 x 15 matrix into 225 array

n\_train = len(X\_train)

# showing image

for i in np.arange(0, W\_grid \* L\_grid):

    index = np.random.randint(0, n\_train)

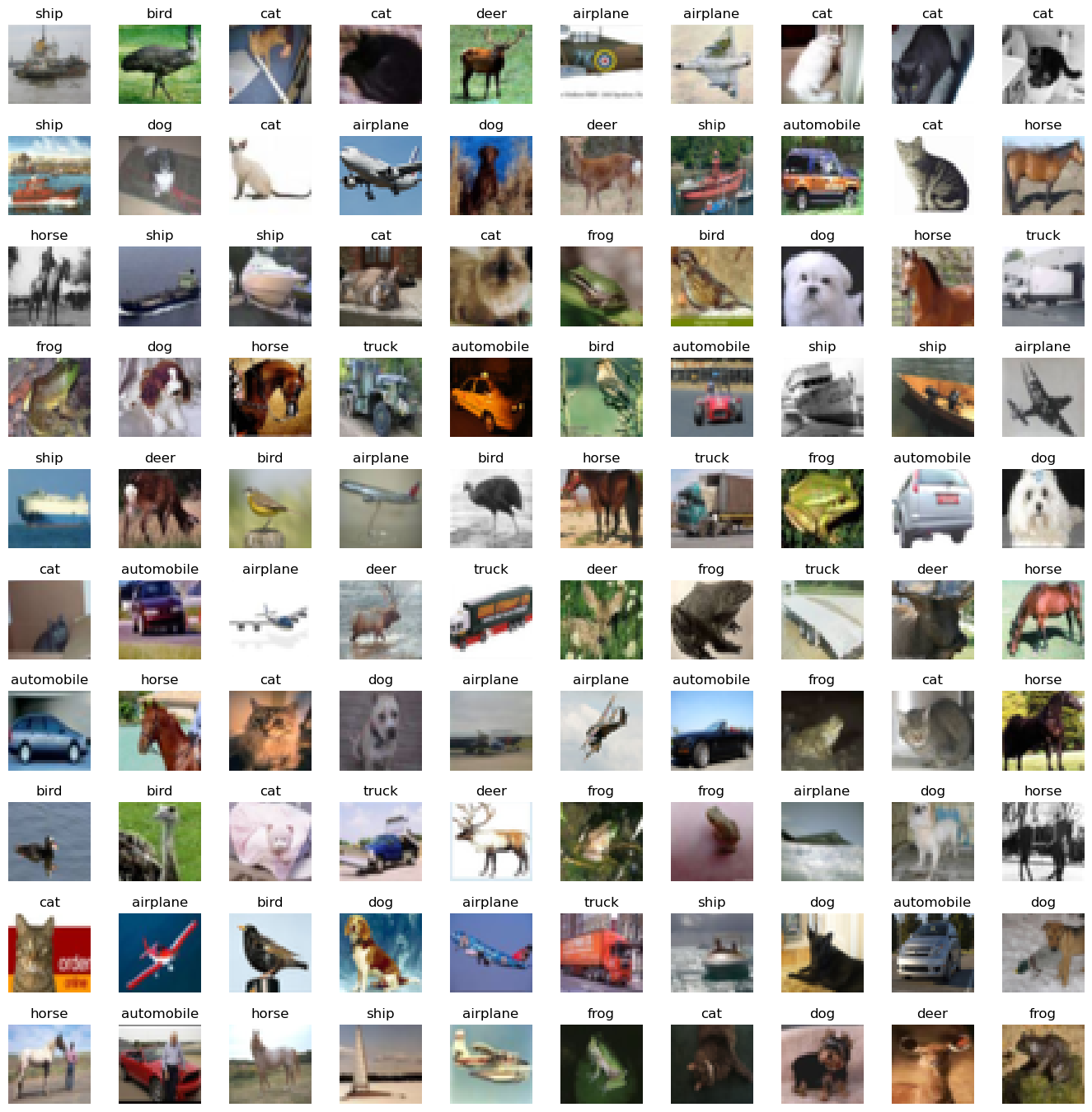
    axes[i].imshow(X\_train[index,1:])

    label\_index = int(y\_train[index])

    axes[i].set\_title(labels[label\_index], fontsize = 12)

    axes[i].axis('off')

plt.subplots\_adjust(hspace=0.4)

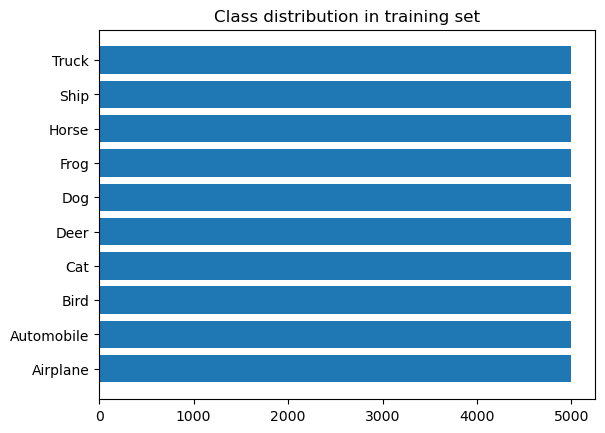


classes\_name = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

classes, counts = np.unique(y\_train, return\_counts=True)

plt.barh(classes\_name, counts)

plt.title('Class distribution in training set')



classes, counts = np.unique(y\_test, return\_counts=True)

plt.barh(classes\_name, counts)

plt.title('Class distribution in testing set')



# Scale the data

X\_train = X\_train / 255.0

X\_test = X\_test / 255.0

# transforming to one hotencoded

y\_cat\_train = to\_categorical(y\_train, 10)

y\_cat\_test = to\_categorical(y\_test, 10)

print(y\_cat\_train)

y\_cat\_train.shape

[[0. 0. 0. ... 0. 0. 0.]

[0. 0. 0. ... 0. 0. 1.]

[0. 0. 0. ... 0. 0. 1.]

...

[0. 0. 0. ... 0. 0. 1.]

[0. 1. 0. ... 0. 0. 0.]

[0. 1. 0. ... 0. 0. 0.]]

(50000, 10)

INPUT\_SHAPE = (32, 32, 3)

KERNEL\_SIZE = (3, 3)

model = Sequential()

# Convolutional Layer

model.add(Conv2D(filters=32, kernel\_size=KERNEL\_SIZE, input\_shape=INPUT\_SHAPE, activation='relu', padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(filters=32, kernel\_size=KERNEL\_SIZE, input\_shape=INPUT\_SHAPE, activation='relu', padding='same'))

model.add(BatchNormalization())

# Pooling layer

model.add(MaxPool2D(pool\_size=(2, 2)))

# Dropout layers

model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel\_size=KERNEL\_SIZE, input\_shape=INPUT\_SHAPE, activation='relu', padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(filters=64, kernel\_size=KERNEL\_SIZE, input\_shape=INPUT\_SHAPE, activation='relu', padding='same'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(filters=128, kernel\_size=KERNEL\_SIZE, input\_shape=INPUT\_SHAPE, activation='relu', padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(filters=128, kernel\_size=KERNEL\_SIZE, input\_shape=INPUT\_SHAPE, activation='relu', padding='same'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

# model.add(Dropout(0.2))

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.25))

model.add(Dense(10, activation='softmax'))

METRICS = [

    'accuracy',

    tf.keras.metrics.Precision(name='precision'),

    tf.keras.metrics.Recall(name='recall')

]

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=METRICS)

model.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 32, 32, 32) 896

batch\_normalization (BatchN (None, 32, 32, 32) 128

ormalization)

conv2d\_1 (Conv2D) (None, 32, 32, 32) 9248

batch\_normalization\_1 (Batc (None, 32, 32, 32) 128

hNormalization)

max\_pooling2d (MaxPooling2D (None, 16, 16, 32) 0

)

dropout (Dropout) (None, 16, 16, 32) 0

conv2d\_2 (Conv2D) (None, 16, 16, 64) 18496

batch\_normalization\_2 (Batc (None, 16, 16, 64) 256

hNormalization)

conv2d\_3 (Conv2D) (None, 16, 16, 64) 36928

...

Total params: 552,362

Trainable params: 551,466

Non-trainable params: 896

batch\_size = 32

data\_generator = ImageDataGenerator(width\_shift\_range=0.1, height\_shift\_range=0.1, horizontal\_flip=True)

train\_generator = data\_generator.flow(X\_train, y\_cat\_train, batch\_size)

steps\_per\_epoch = X\_train.shape[0] // batch\_size

r = model.fit(train\_generator,

              epochs=20,

              steps\_per\_epoch=steps\_per\_epoch,

              validation\_data=(X\_test, y\_cat\_test),

#               callbacks=[early\_stop],

#               batch\_size=batch\_size,

             )

Epoch 1/20

1562/1562 [==============================] - 106s 66ms/step - loss: 1.6122 - accuracy: 0.4181 - precision: 0.6287 - recall: 0.2102 - val\_loss: 1.2116 - val\_accuracy: 0.5668 - val\_precision: 0.7252 - val\_recall: 0.4154

Epoch 2/20

1562/1562 [==============================] - 120s 77ms/step - loss: 1.2101 - accuracy: 0.5749 - precision: 0.7306 - recall: 0.4145 - val\_loss: 1.6112 - val\_accuracy: 0.5207 - val\_precision: 0.5816 - val\_recall: 0.4441

Epoch 3/20

1562/1562 [==============================] - 144s 92ms/step - loss: 1.0167 - accuracy: 0.6459 - precision: 0.7767 - recall: 0.5205 - val\_loss: 0.9030 - val\_accuracy: 0.6945 - val\_precision: 0.7980 - val\_recall: 0.6011

Epoch 4/20

1562/1562 [==============================] - 121s 77ms/step - loss: 0.9243 - accuracy: 0.6827 - precision: 0.7951 - recall: 0.5690 - val\_loss: 1.0230 - val\_accuracy: 0.6775 - val\_precision: 0.7434 - val\_recall: 0.6071

Epoch 5/20

1562/1562 [==============================] - 106s 68ms/step - loss: 0.8490 - accuracy: 0.7111 - precision: 0.8129 - recall: 0.6123 - val\_loss: 0.7182 - val\_accuracy: 0.7523 - val\_precision: 0.8346 - val\_recall: 0.6784

Epoch 6/20

1562/1562 [==============================] - 124s 80ms/step - loss: 0.7920 - accuracy: 0.7306 - precision: 0.8253 - recall: 0.6425 - val\_loss: 0.6655 - val\_accuracy: 0.7762 - val\_precision: 0.8605 - val\_recall: 0.6949

Epoch 7/20

1562/1562 [==============================] - 156s 100ms/step - loss: 0.7485 - accuracy: 0.7470 - precision: 0.8336 - recall: 0.6645 - val\_loss: 0.6916 - val\_accuracy: 0.7743 - val\_precision: 0.8376 - val\_recall: 0.7237

Epoch 8/20

1562/1562 [==============================] - 205s 131ms/step - loss: 0.7179 - accuracy: 0.7567 - precision: 0.8384 - recall: 0.6818 - val\_loss: 0.6461 - val\_accuracy: 0.7846 - val\_precision: 0.8452 - val\_recall: 0.7266

Epoch 9/20

1562/1562 [==============================] - 107s 68ms/step - loss: 0.6840 - accuracy: 0.7691 - precision: 0.8467 - recall: 0.6974 - val\_loss: 0.6716 - val\_accuracy: 0.7786 - val\_precision: 0.8426 - val\_recall: 0.7214

Epoch 10/20

1562/1562 [==============================] - 119s 76ms/step - loss: 0.6553 - accuracy: 0.7771 - precision: 0.8514 - recall: 0.7092 - val\_loss: 0.7011 - val\_accuracy: 0.7692 - val\_precision: 0.8311 - val\_recall: 0.7226

Epoch 11/20

1562/1562 [==============================] - 131s 84ms/step - loss: 0.6364 - accuracy: 0.7837 - precision: 0.8546 - recall: 0.7202 - val\_loss: 0.5675 - val\_accuracy: 0.8124 - val\_precision: 0.8709 - val\_recall: 0.7571

Epoch 12/20

1562/1562 [==============================] - 139s 89ms/step - loss: 0.6107 - accuracy: 0.7919 - precision: 0.8602 - recall: 0.7318 - val\_loss: 0.6458 - val\_accuracy: 0.7856 - val\_precision: 0.8417 - val\_recall: 0.7393

Epoch 13/20

...

Epoch 19/20

1562/1562 [==============================] - 133s 85ms/step - loss: 0.5066 - accuracy: 0.8293 - precision: 0.8819 - recall: 0.7816 - val\_loss: 0.4860 - val\_accuracy: 0.8410 - val\_precision: 0.8811 - val\_recall: 0.8036

Epoch 20/20

1562/1562 [==============================] - 106s 68ms/step - loss: 0.5018 - accuracy: 0.8295 - precision: 0.8810 - recall: 0.7821 - val\_loss: 0.4657 - val\_accuracy: 0.8459 - val\_precision: 0.8849 - val\_recall: 0.8079

plt.figure(figsize=(12, 16))

plt.subplot(4, 2, 1)

plt.plot(r.history['loss'], label='Loss')

plt.plot(r.history['val\_loss'], label='val\_Loss')

plt.title('Loss Function Evolution')

plt.legend()

plt.subplot(4, 2, 2)

plt.plot(r.history['accuracy'], label='accuracy')

plt.plot(r.history['val\_accuracy'], label='val\_accuracy')

plt.title('Accuracy Function Evolution')

plt.legend()

plt.subplot(4, 2, 3)

plt.plot(r.history['precision'], label='precision')

plt.plot(r.history['val\_precision'], label='val\_precision')

plt.title('Precision Function Evolution')

plt.legend()

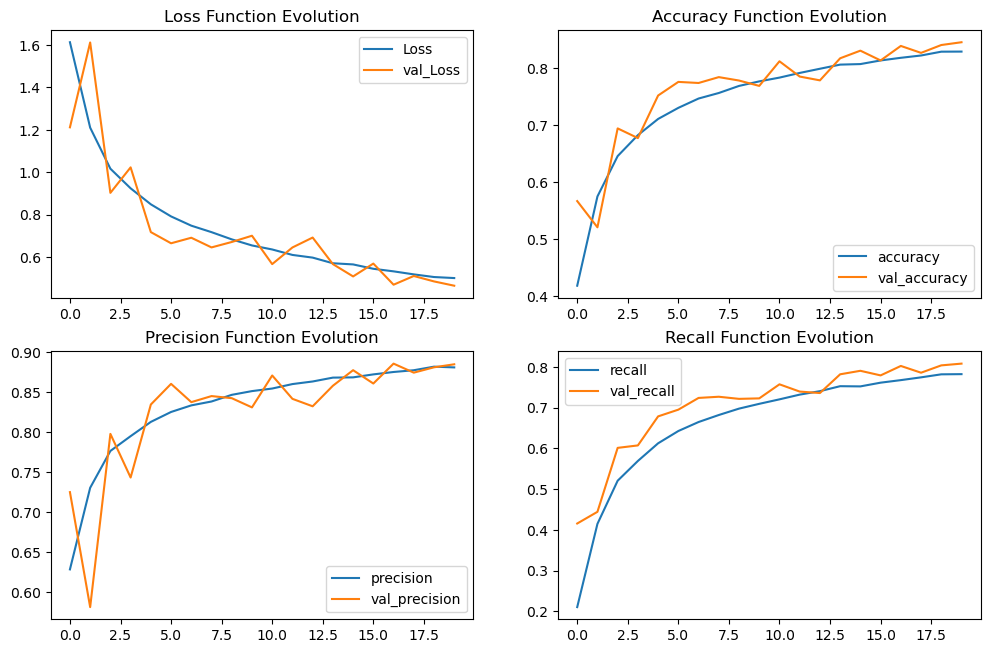
plt.subplot(4, 2, 4)

plt.plot(r.history['recall'], label='recall')

plt.plot(r.history['val\_recall'], label='val\_recall')

plt.title('Recall Function Evolution')

plt.legend()



evaluation = model.evaluate(X\_test, y\_cat\_test)

print(f'Test Accuracy : {evaluation[1] \* 100:.2f}%')

y\_pred = model.predict(X\_test)

y\_pred = np.argmax(y\_pred, axis=1)

cm = confusion\_matrix(y\_test, y\_pred)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=labels)

fig, ax = plt.subplots(figsize=(10, 10))

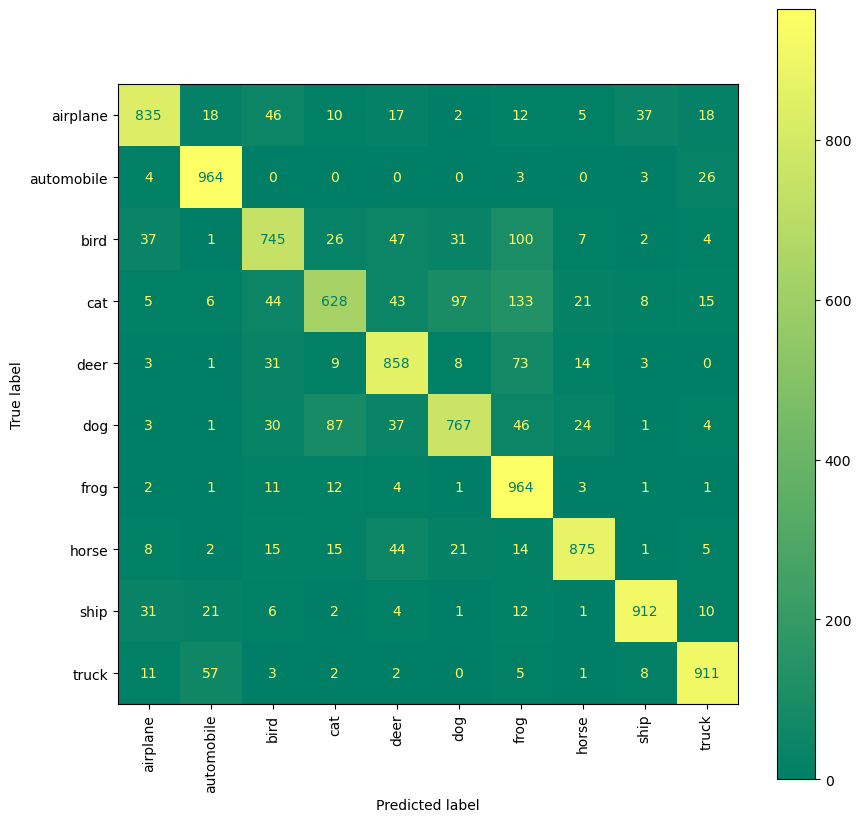
disp = disp.plot(xticks\_rotation='vertical', ax=ax,cmap='summer')

plt.show()

313/313 [==============================] - 4s 14ms/step - loss: 0.4657 - accuracy: 0.8459 - precision: 0.8849 - recall: 0.8079

Test Accuracy : 84.59%

313/313 [==============================] - 4s 12ms/step



print(classification\_report(y\_test, y\_pred))

precision recall f1-score support

0 0.89 0.83 0.86 1000

1 0.90 0.96 0.93 1000

2 0.80 0.74 0.77 1000

3 0.79 0.63 0.70 1000

4 0.81 0.86 0.83 1000

5 0.83 0.77 0.80 1000

6 0.71 0.96 0.82 1000

7 0.92 0.88 0.90 1000

8 0.93 0.91 0.92 1000

9 0.92 0.91 0.91 1000

accuracy 0.85 10000

macro avg 0.85 0.85 0.84 10000

weighted avg 0.85 0.85 0.84 10000

my\_image = X\_test[2000]

plt.imshow(my\_image)

y\_ele = y\_test[2000]

print(f" (Actual)Image is : {labels[int(y\_ele)]}")

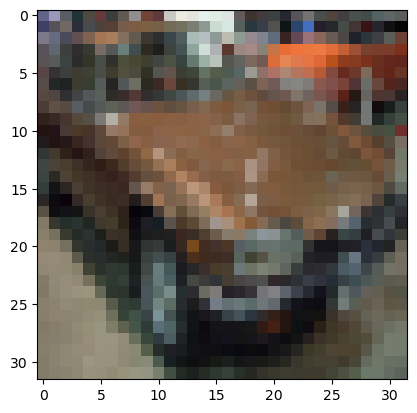
predicted = np.argmax(model.predict(my\_image.reshape(1, 32, 32, 3)))

print(f"(Predicted)Image is {labels[int(predicted)]}")

(Actual)Image is : automobile

1/1 [==============================] - 0s 24ms/step

(Predicted)Image is automobile



labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',

          'dog', 'frog', 'horse', 'ship', 'truck']

W\_grid = 5

L\_grid = 5

fig, axes = plt.subplots(L\_grid, W\_grid, figsize = (17,17))

axes = axes.ravel() # flaten the 15 x 15 matrix into 225 array

n\_test = len(X\_test)

for i in np.arange(0, W\_grid \* L\_grid): # create evenly spaces variables

    index = np.random.randint(0, n\_test)

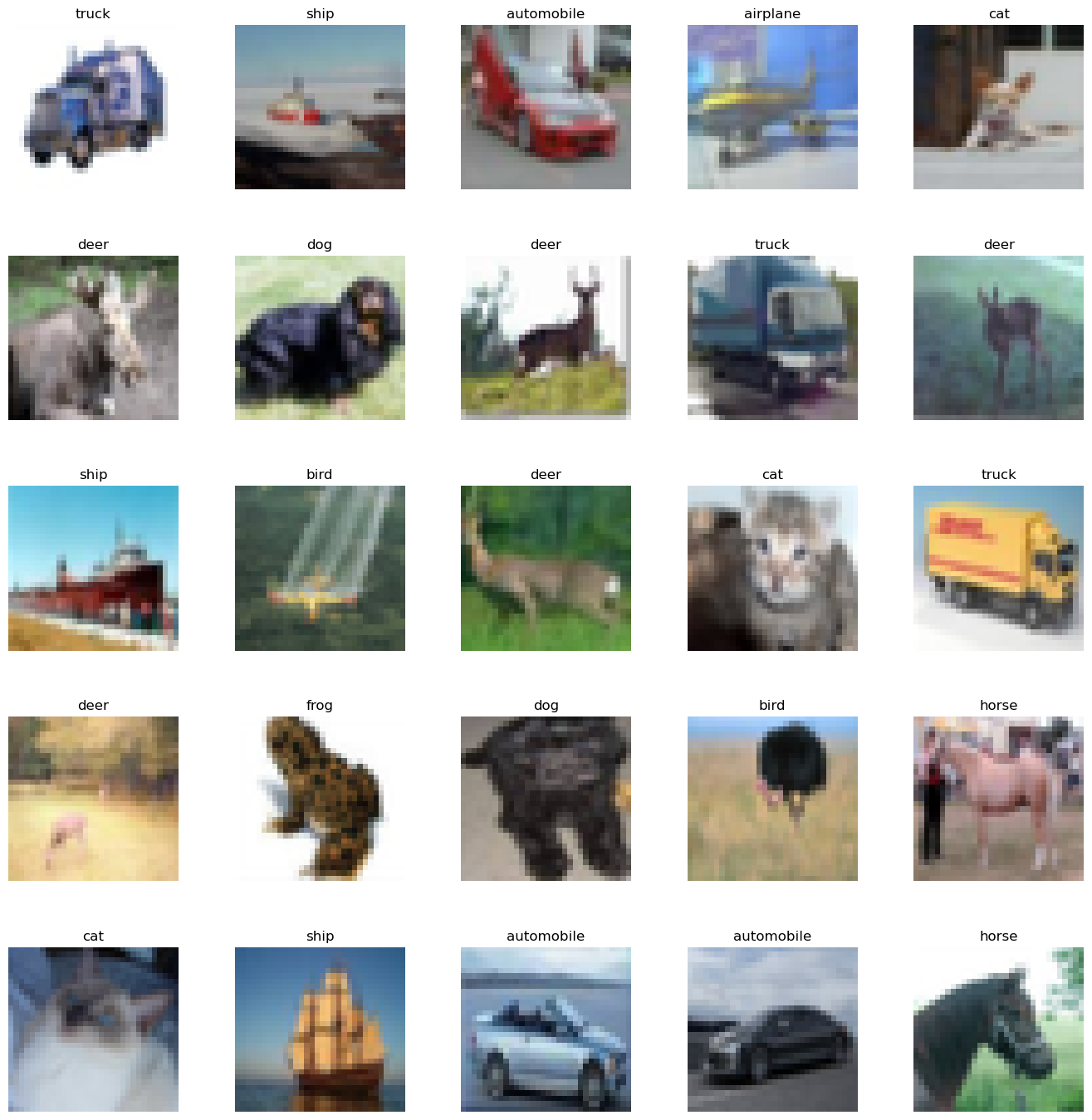
    axes[i].imshow(X\_test[index,1:])

    label\_index = int(y\_pred[index])

    axes[i].set\_title(labels[label\_index], fontsize = 12)

    axes[i].axis('off')

plt.subplots\_adjust(hspace=0.4)



def plot\_image(i, predictions\_array, true\_label, img):

    predictions\_array, true\_label, img = predictions\_array, true\_label[i], img[i]

    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"{labels[int(predicted\_label)]} {100\*np.max(predictions\_array):2.0f}% ({labels[int(true\_label)]})",

               color=color)

def plot\_value\_array(i, predictions\_array, true\_label):

    predictions\_array, true\_label = predictions\_array, int(true\_label[i])

    plt.grid(False)

    plt.xticks(range(10))

    plt.yticks([])

    thisplot = plt.bar(range(10), predictions\_array, color="#777777")

    plt.ylim([0, 1])

    predicted\_label = np.argmax(predictions\_array)

    thisplot[predicted\_label].set\_color('red')

    thisplot[true\_label].set\_color('blue')

predictions = model.predict(X\_test)

# Plot the first X test images, their predicted labels, and the true labels.

# Color correct predictions in blue and incorrect predictions in red.

num\_rows = 8

num\_cols = 5

num\_images = num\_rows \* num\_cols

plt.figure(figsize=(2 \* 2 \* num\_cols, 2 \* num\_rows))

print("Predicted : red\t Actual : blue")

for i in range(num\_images):

    plt.subplot(num\_rows, 2 \* num\_cols, 2 \* i + 1)

    plot\_image(i, predictions[i], y\_test, X\_test)

    plt.subplot(num\_rows, 2\*num\_cols, 2\*i+2)

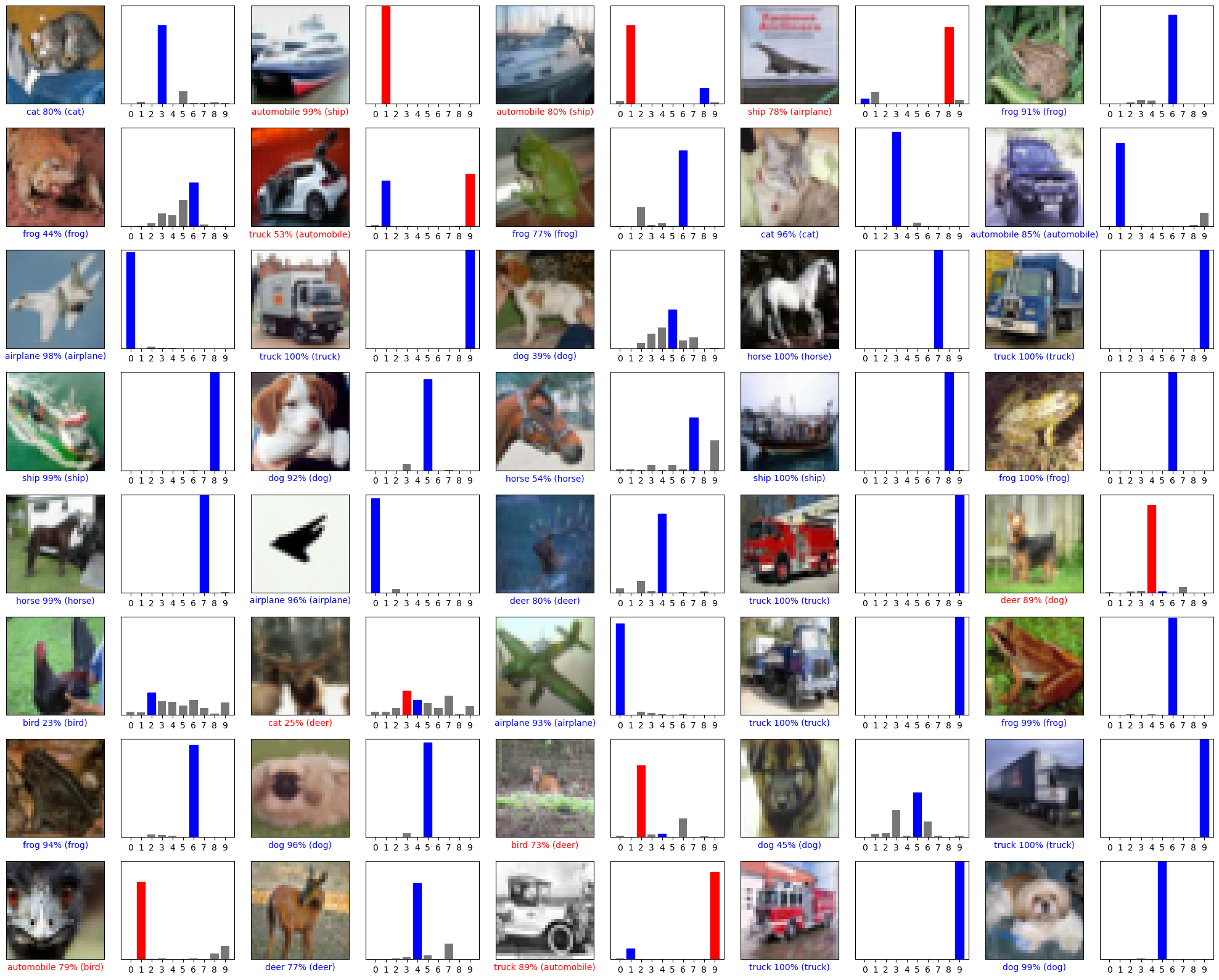
    plot\_value\_array(i, predictions[i], y\_test)

plt.tight\_layout()

plt.show()

313/313 [==============================] - 4s 12ms/step

Predicted : red Actual : blue



**RESULTS:**

**accuracy: 0.8295 ; val\_accuracy : 0.8459**

**precision: 0.8810 ; val\_precision : 0.8849**

**loss: 0.5018 ; val\_loss : 0.4657**

**REFERENCES:**

* [tensorflow.keras](https://www.tensorflow.org/api_docs/python/tf)
* [cifar10](https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)