Assignment

Neural Network Models for Object Recognition

MSc Artificial Intelligence

Machine Learning

Murthy Kanuri
Student ID: 12696139

INTRODUCTION

Object recognition is integral to AI, in which systems use machine learning techniques to create meaning from the appearance or presence of objects from images and video streams. It acts as the building block upon which many applications are based:

- **Autonomous Vehicle Systems**
- **Surveillance Systems**
- **Healthcare**
- **Retail and E-commerce**
- **Agriculture**

Integrating object recognition into such applications can enhance the circle of automation, accuracy, and efficiency, hence leading to more innovations with improved results in different industry spheres.

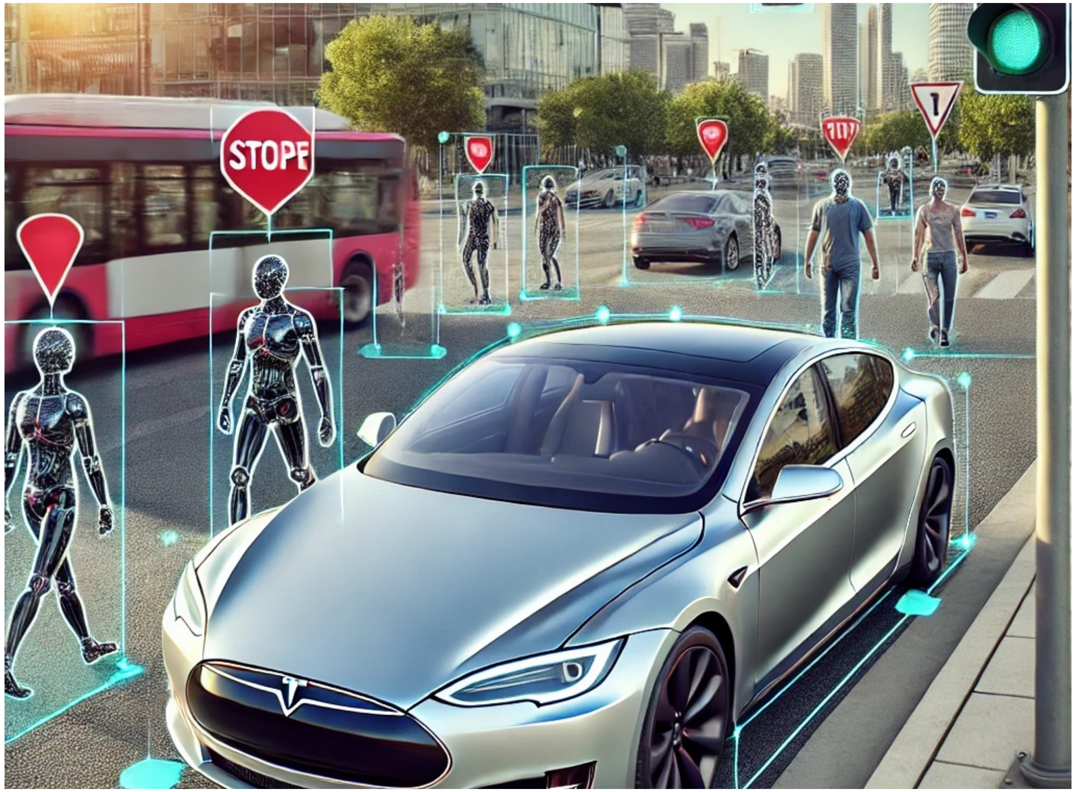


Image Source : Generated through Open AI

OVERVIEWOF THE CIFAR-10 DATASET

CIFAR-10 *(Canadian Institute for Advanced Research, 10 classes)* is a widely used benchmark dataset in machine learning and computer vision, introduced by *Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton in 2009.*

Data Composition	
Total Images	60,000 Colour images
Image Dimensions	32X32 pixels
Number of Classes	10 distinct categories
Images per Class	6000
'RGB' Colour	3

Data Split	
Training Set	50,000 images
Test Set	10,000 images

Class Categories
Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck

CIFAR 10 - IMAGE CLASSIFICATION

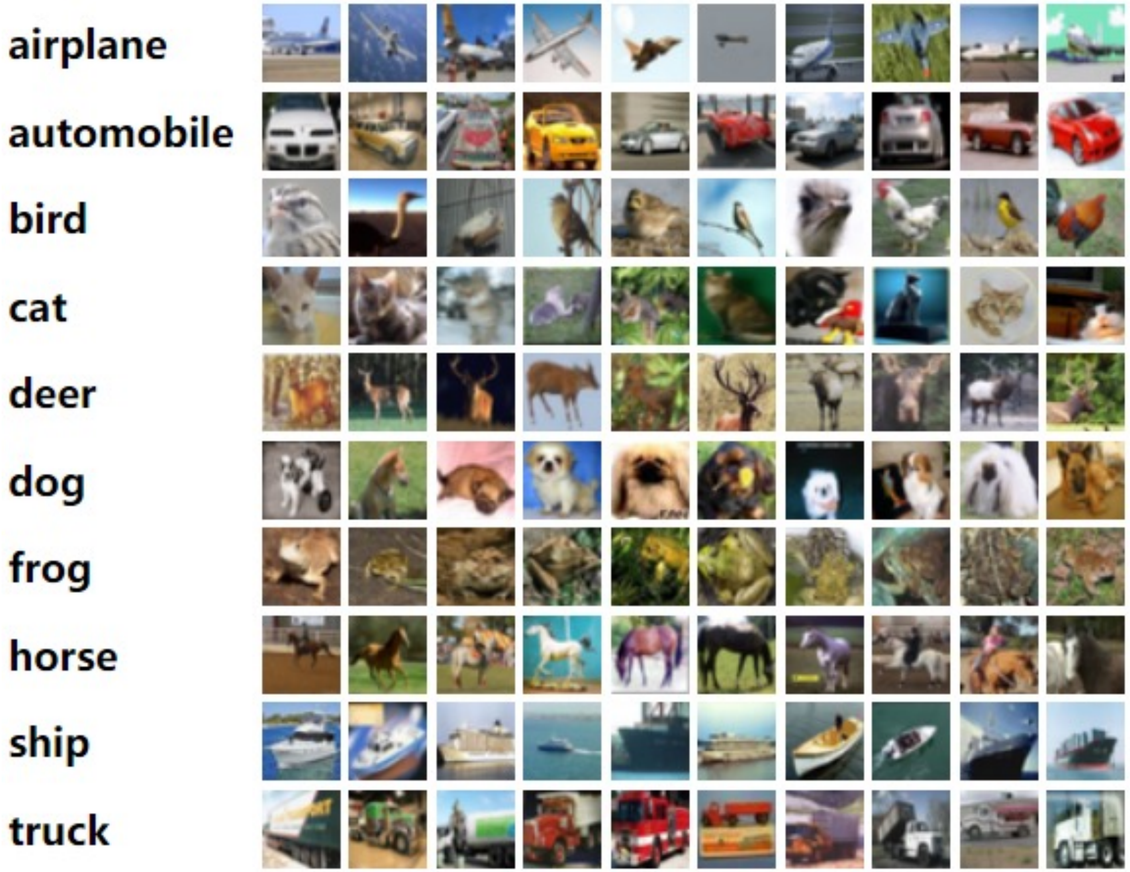


Image Source : Krizhevsky, A., Nair, V. and Hinton, G., 2009. *CIFAR-10 dataset*. Available at: https://www.cs.toronto.edu/~kriz/cifar.html [Accessed 11 January 2025].

REQUIRED LIBRARIES

All the essential libraries used are Python based and provide essential tools for for data handling, visualization, model development, and evaluation.

```
import matplotlib.pyplot as plt # For plotting and visualising data, it allows plotting graphs, charts and images.
import numpy as np # For numerical computations and array manipulations
from sklearn.model_selection import train_test_split # For splitting the dataset into training and test sets,
optionally with a validation set
from tensorflow.keras.datasets import cifar10 # To load the CIFAR-10 dataset, which contains 60,000 32x32 colour
images in 10 classes
from tensorflow.keras.utils import to_categorical # used to convert class labels into a one-hot encoded format
from tensorflow.keras.models import Sequential # For creating a sequential neural network.
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization,
Input # For adding layers to the neural network, including convolutional, pooling, flattening, dense, dropout, batch
normalization, and input layers.
from tensorflow.keras.callbacks import EarlyStopping # Monitor validation loss and stop training when no
improvement is observed and prevent overfitting
import seaborn as sns # Seaborn is used to create attractive and informative statistical visualisations, such as
confusion matrices and accuracy trends.
from sklearn.metrics import confusion_matrix, classification_report # For evaluating model performance by
generating a confusion matrix and classification report, which provide accuracy, precision, recall, and F1-score for each
class.
```

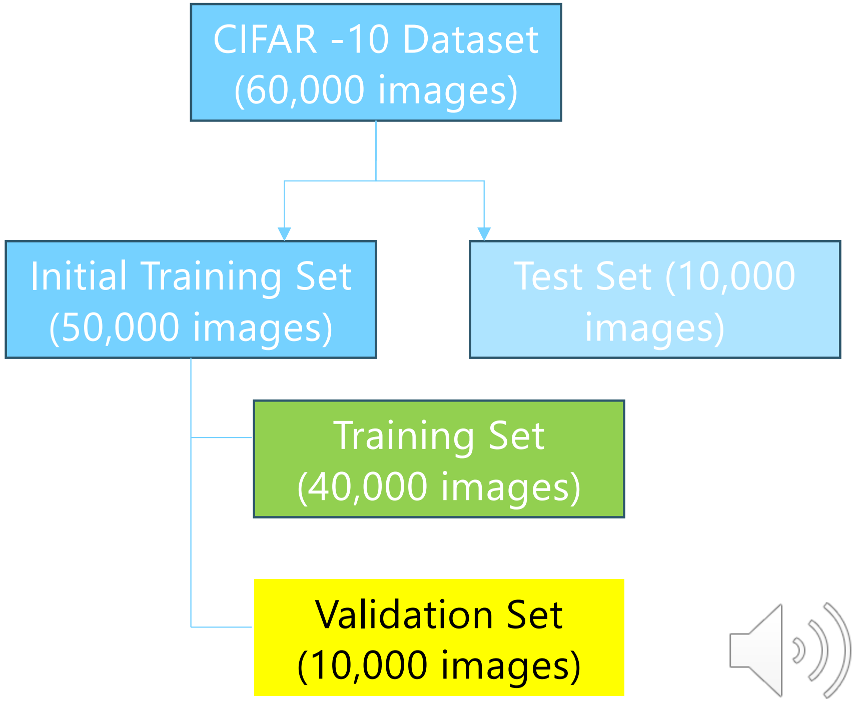
PARTITIONING THE VALIDATION SET AND DATA INSIGHTS

The train_test_split() function in scikit-learn is a handy utility that splits the dataset into training and testing sets. This is essential for training the model on one subset of data and validating it on another to evaluate its performance (Zaczyński, 2023).

Dataset is partitioned into three distinct subsets—80% for training, 10% for validation, and 10% for testing using the scikit-learn's train_test_split

```
# Load CIFAR-10 dataset
(X_train_full, y_train_full), (X_test, y_test) = cifar10.load_data()

#  Confirm dataset dimensions
print("Original Training set shape:", X_train_full.shape)  # (50000, 32, 32, 3)
print("Original Testing set shape:", X_test.shape)        # (10000, 32, 32, 3)

# Split training data into 80% training and 20% validation sets
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full, y_train_full, test_size=0.2, random_state=42, stratify=y_train_full
)

# Verify the dimensions of the partitions
print("Training set shape:", X_train.shape)  # (40000, 32, 32, 3)
print("Validation set shape:", X_val.shape)  # (10000, 32, 32, 3)
print("Testing set shape:", X_test.shape)    # (10000, 32, 32, 3)
```

```
Original Training set shape: (50000, 32, 32, 3)
Original Testing set shape: (10000, 32, 32, 3)
Training set shape: (40000, 32, 32, 3)
Validation set shape: (10000, 32, 32, 3)
Testing set shape: (10000, 32, 32, 3)
```



TRAINING, VALIDATION AND TESTING SET DETAILS

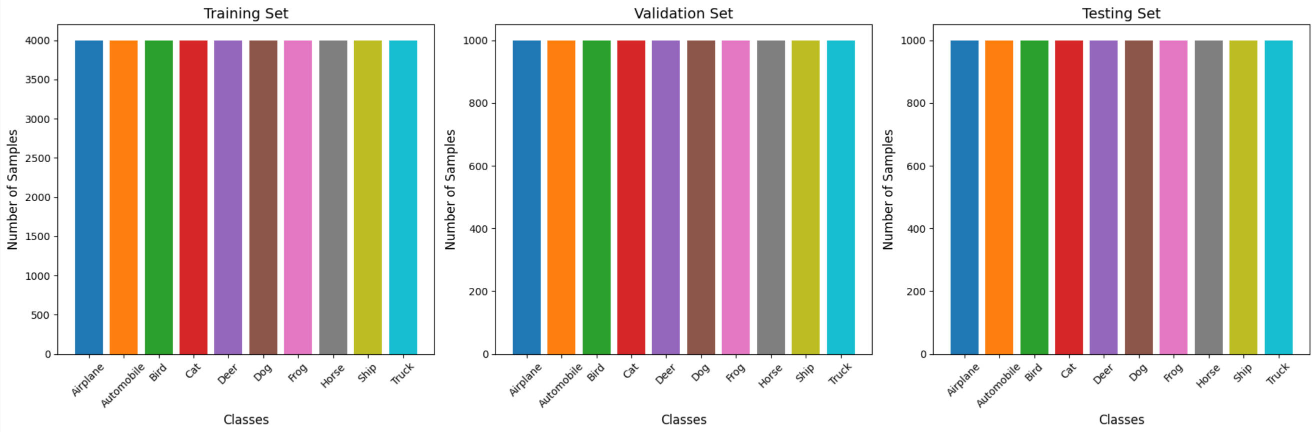
The **training set**, **validation set**, and **testing set** are subsets of a dataset used in the process of training and evaluating machine learning models. Each serves a distinct purpose to ensure the model generalizes well to unseen data.

Dataset Subset	Purpose	Interaction with Model	Typical Dataset Splits
Training Set	Train the model and adjust weights.	Directly affects the model's parameters.	~60-80%
Validation Set	Tune hyperparameters and prevent overfitting.	Guides decisions during training but doesn't affect weights.	~10-20%
Testing Set	evaluate the model's final performance	Used only for final assessment, never during training.	~10-20%

Reference : Murphy, K.P. (2012) Machine Learning: A Probabilistic Perspective. Cambridge: MIT Press.

VISUALIZING CLASS DISTRUBUTIONS

```python
# Class names corresponding to CIFAR-10 labels (0-9)
class_names = [
    "Airplane", "Automobile", "Bird", "Cat", "Deer",
    "Dog", "Frog", "Horse", "Ship", "Truck"
]

# Function to plot the distribution of classes in a dataset
def plot_class_distribution(y, title, ax):
    unique, counts = np.unique(y, return_counts=True)
    ax.bar(class_names, counts, color=plt.cm.tab10.colors)
    ax.set_title(title, fontsize=14)
    ax.set_xlabel("Classes", fontsize=12)
    ax.set_ylabel("Number of Samples", fontsize=12)
    ax.tick_params(axis='x', rotation=45)

# Create a grid for three plots: Training, Validation, Testing
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Plot class distributions for each dataset
plot_class_distribution(y_train, "Training Set", axes[0])
plot_class_distribution(y_val, "Validation Set", axes[1])
plot_class_distribution(y_test, "Testing Set", axes[2])

plt.tight_layout()
plt.show()
```



IMPORTANCE OF A VALIDATION SET

- **Model Evaluation :** Helps track the model's learning progress and provides an unbiased estimate of its performance during training *(Goodfellow et al., 2016; Bishop, 2006)*.
- **Hyperparameter Tuning :** Used to optimize parameters such as learning rate, batch size, and model architecture, ensuring the best combination for accuracy and minimal loss *(Hastie, Tibshirani, & Friedman, 2009; Géron, 2019)*.
- **Prevent Overfitting :** Detects overfitting when training performance improves but validation performance declines, ensuring the model learns meaningful patterns instead of memorizing data *(Kohavi, 1995; Goodfellow et al., 2016)*.
- **Early Stopping :** Monitors validation loss and stops training automatically if performance stops improving, preventing overtraining *(Prechelt, 1998; Géron, 2019)*.
- **Model Selection :** Helps compare different models or configurations, selecting the one with the best validation performance for final testing and deployment *(Hastie, Tibshirani, & Friedman, 2009; Chollet, 2021)*.

DATA PREPARATION-NORMALISATION

In the CIFAR-10 dataset, images have pixel intensity values ranging from 0 to 255. Normalizing values to the [0,1] range involves dividing each pixel value by 255.

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

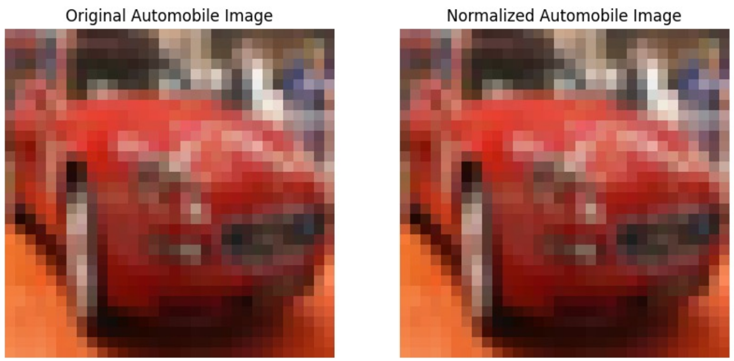
Where:

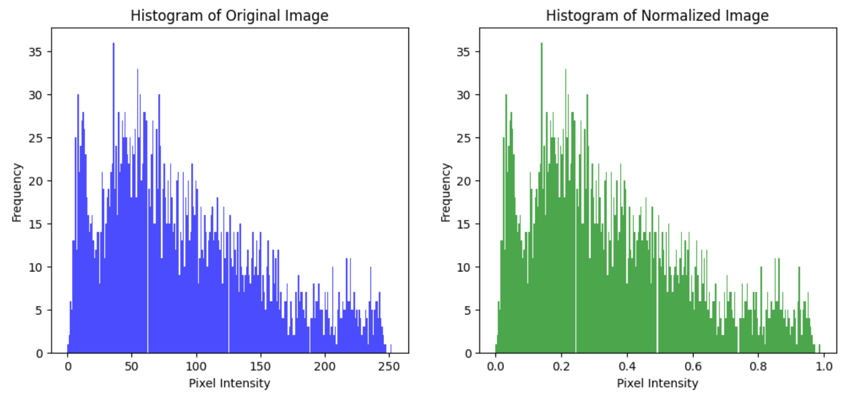
•X is the original pixel value.

•X′ is the normalized pixel value.

•X_{min} is the minimum pixel value in the image (0 for 8-bit images).

•X_{max} is the maximum pixel value in the image (255 for 8-bit images).





Data Normalisation significantly influences the accuracy of machine learning algorithms (Cabello-Solorzano et al., 2023).

DATA PREPARATION-NORMALISATION

Before Normalisation

```
# Show the pixel values of the first image before normalization
X_train[0]
X_val[0]
X_test[0]
```

```
ndarray (32, 32, 3) hide data
array([[[158, 112,  49],
        [159, 111,  47],
        [165, 116,  51],
        ...,
        [137,  95,  36],
        [126,  91,  36],
        [116,  85,  33]],

       [[152, 112,  51],
        [151, 110,  40],
        [159, 114,  45],
        ...,
        [136,  95,  31],
        [125,  91,  32],
        [119,  88,  34]],

       [[151, 110,  47],
        [151, 109,  33],
        [158, 111,  36],
```

```
#  Data Preprocessing - normalization
X_train = X_train / 255.0
X_val = X_val / 255.0
X_test = X_test/ 255.0
```

After Normalisation

```
# Show the pixel values of the first image after normalization
X_train[0]
X_val[0]
X_test[0]
```

```
array([[[0.61960784, 0.43921569, 0.19215686],
        [0.62352941, 0.43529412, 0.18431373],
        [0.64705882, 0.45490196, 0.2       ],
        ...,
        [0.5372549 , 0.37254902, 0.14117647],
        [0.49411765, 0.35686275, 0.14117647],
        [0.45490196, 0.33333333, 0.12941176]],

       [[0.59607843, 0.43921569, 0.2       ],
        [0.59215686, 0.43137255, 0.15686275],
        [0.62352941, 0.44705882, 0.17647059],
        ...,
        [0.53333333, 0.37254902, 0.12156863],
        [0.49019608, 0.35686275, 0.1254902 ],
        [0.46666667, 0.34509804, 0.13333333]],

       [[0.59215686, 0.43137255, 0.18431373],
        [0.59215686, 0.42745098, 0.12941176],
        [0.61960784, 0.43529412, 0.14117647],
        ...,
        [0.54509804, 0.38431373, 0.13333333],
        [0.50980392, 0.37254902, 0.13333333],
        [0.47058824, 0.34901961, 0.12941176]],

       ...,

       [[0.26666667, 0.48627451, 0.69411765],
        [0.16470588, 0.39215686, 0.58039216],
        [0.12156863, 0.34509804, 0.5372549 ],
```

DATA PREPARATION - ONE-HOT ENCODING

One-hot encoding converts categorical labels into binary vectors, ensuring each class is represented uniquely with a '1' in one position and '0s' elsewhere. This transformation prevents unintended ordinal relationships and makes the data compatible with classification models (Géron, 2019).

```
y_train = to_categorical(y_train,num_classes=10)
y_val = to_categorical(y_val, num_classes=10)
y_test = to_categorical(y_test,num_classes=10)
```

```
print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
print(f"X_val shape: {X_val.shape}, y_val shape: {y_val.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")
```

```
X_train shape: (40000, 32, 32, 3), y_train shape: (40000, 10, 10, 10)
X_val shape: (10000, 32, 32, 3), y_val shape: (10000, 10, 10, 10)
X_test shape: (10000, 32, 32, 3), y_test shape: (10000, 10, 10, 10)
```

ARTIFICIAL NEURAL NETWORKS (ANN)

ANN Components : Input Layer, Hidden Layers and Output Layer

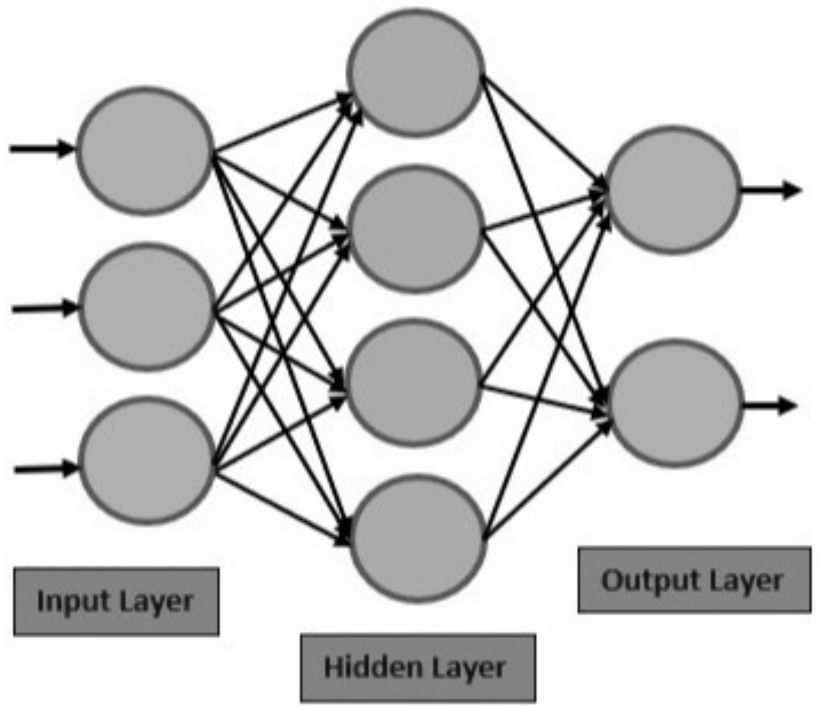


Image Source : As illustrated in the architecture of Artificial Neural Networks (Raj, Ravi & Kos 2023)..."

CONVOLUTIONAL NEURAL NETWORKS (CNN)

Convolutional Neural Networks (CNN) has two steps Feature Learning and Classification.

The combination of **feature learning** and **classification** enables CNNs to excel at complex visual tasks like object recognition and image classification.

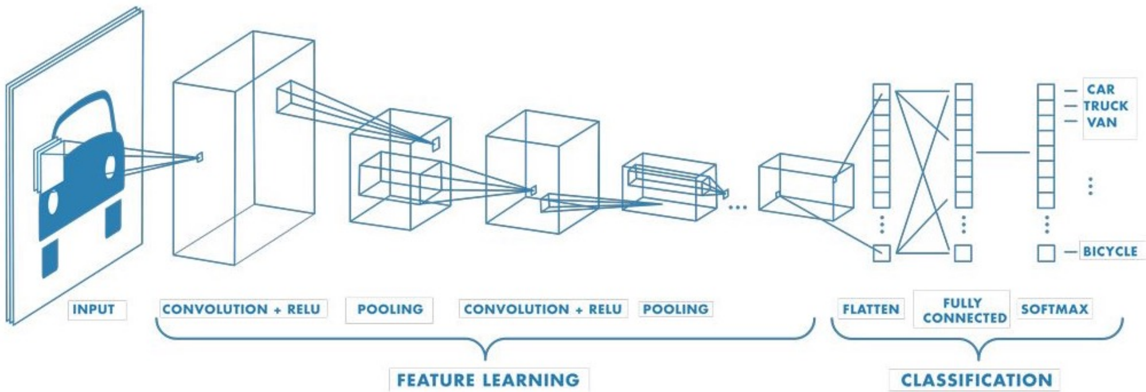


Image Source : Mathworks (2018)

CHOSEN ACTIVATION FUNCTIONS - RECTIFIED LINEAR UNIT (ReLU & Softmax)

An activation function in a neural network determines whether a neuron is activated, introducing non-linearity to help the model learn complex patterns (Goodfellow, Bengio & Courville, 2016).

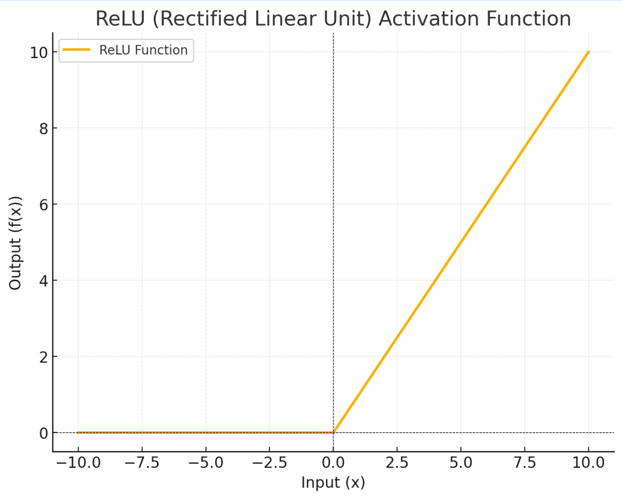
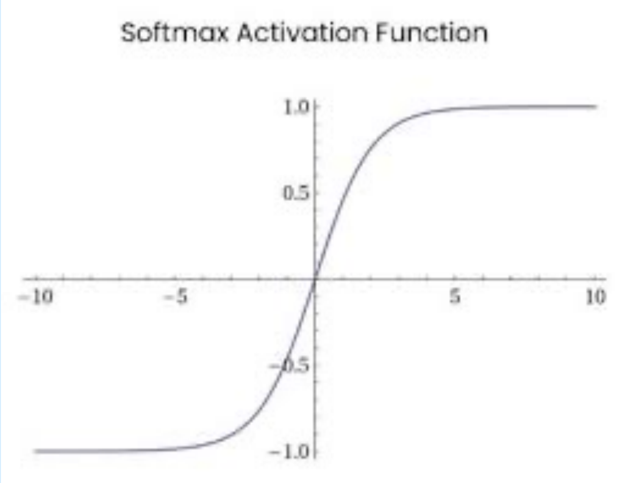
Activation Function	Mathematical Expression	Range	Key Characteristics
Rectified Linear Unit (ReLU)	$f(x) = \max(0, x)$	[0,∞)	Non-linear, allows sparse activations, introduces non-linearity, Prevents vanishing gradient problem, Less computationally complex compared to sigmoid or tanh
Softmax	$f(x)_i = \frac{e^{Z_i}}{\sum_{j=1}^{K} e^{Z_j}}$	(0,1)	Outputs a probability distribution over multiple classes; used in the output layer of classification models.

ReLU was chosen for its efficiency in deep networks, and Softmax for its probabilistic output interpretation in classification tasks.

Both ReLU and Softmax are used in conjunction, with ReLU applied to the hidden layers and Softmax to the output layer.

Reference : Goodfellow, I., Bengio, Y. and Courville, A. (2016) Deep Learning. Cambridge: MIT Press.

CHOSEN ACTIVATION FUNCTIONS - RECTIFIED LINEAR UNIT (ReLU & Softmax)

ReLU	Softmax
	
The graph shows a clear linear increase for positive values while remaining flat for negative inputs.	The S-shaped curve demonstrates how it transforms raw values into probabilities, with a steep transition in the middle and saturation at extremes.

ReLU Image Source : Towards Data Science, 2021. Understanding Activation Functions in Neural Networks. [online] Available at: https://towardsdatascience.com

Softmax Image Source : Analytics Vidhya, 2021. Activation Functions in Neural Networks. [online] Available at: https://www.analyticsvidhya.com

IMPLEMENTED LOSS FUNCTION

Categorical Cross-Entropy (CCE) is a **loss function** commonly used in classification problems where the target variable belongs to one of several possible classes. It measures the dissimilarity between the true label distribution and the predicted probability distribution generated by a model (Goodfellow, Bengio, and Courville, 2016).

CCE is particularly useful when the outputs of a model are **probabilities**, such as those produced by a **softmax activation function** in the final layer of a neural network.

$$CCE = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

Where:

- C : Number of classes
- y_i : The true label for class iii (1 if the class is the true label, otherwise 0, i.e., one-hot encoded).
- $\hat{y}_i$: The predicted probability for class iii (output of softmax).

BUILD THE MODEL

```
model = Sequential([
    # Define the input layer with image dimensions for CIFAR-10 (32x32 pixels, 3 color channels)
    Input(shape=(32, 32, 3)),

    # Convolutional Block 1 : Two convolutional layers to capture basic image patterns, followed by max pooling
    Conv2D(32, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(32, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Convolutional Block 2 : Deeper feature extraction with 64 filters, followed by pooling and dropout
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Convolutional Block 3 : Even deeper features captured with 128 filters
    Conv2D(128, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(128, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    # Fully Connected Layers : Flatten the 3D feature maps into a 1D vector for the fully connected layers
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')  # Output layer for 10 classes
])
```

MODEL SUMMARY

```python
# Display a detailed summary of the model's architecture, including layer types,
# output shapes, and the total number of parameters
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 256)	524,544
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1,290

Total params: 847,530 (3.23 MB)
Trainable params: 846,634 (3.23 MB)
Non-trainable params: 896 (3.50 KB)

COMPILE THE MODEL

```python
# Compile the model with Adam optimizer and categorical cross-entropy loss
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Categorical Cross - Entropy

'Adam' (Adaptive Moment Estimation)optimiser, is a cornerstone in advanced model training (Ogundokun et al., 2022). It dynamically adjusts learning rates based on historical gradient information and combines AdaGrad and RMSProp (Zou et al., 2019), making it highly adaptive and efficient, especially for complex datasets.

TRAIN THE MODEL

```
# Set up early stopping to halt training when validation loss stops improving
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
# Train the model using the training data and validate on the validation set
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    epochs=50,
                    batch_size=64,
                    callbacks=[early_stopping])
```

50 Epochs,
Batch size =64

TRAINING LOG / TRAINING PROGRESS OUTPUT

```
Epoch 1/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 343s 536ms/step - accuracy: 0.2143 - loss: 2.3141 - val_accuracy: 0.3802 - val_loss: 1.8274
Epoch 2/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 380s 534ms/step - accuracy: 0.4220 - loss: 1.5857 - val_accuracy: 0.5038 - val_loss: 1.3827
Epoch 3/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 378s 528ms/step - accuracy: 0.5187 - loss: 1.3445 - val_accuracy: 0.6137 - val_loss: 1.1212
Epoch 4/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 382s 528ms/step - accuracy: 0.5911 - loss: 1.1822 - val_accuracy: 0.6235 - val_loss: 1.0857
Epoch 5/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 383s 529ms/step - accuracy: 0.6475 - loss: 1.0306 - val_accuracy: 0.6724 - val_loss: 0.9425
Epoch 6/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 382s 529ms/step - accuracy: 0.6821 - loss: 0.9399 - val_accuracy: 0.7213 - val_loss: 0.8322
Epoch 7/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 332s 531ms/step - accuracy: 0.7073 - loss: 0.8641 - val_accuracy: 0.7402 - val_loss: 0.7616
Epoch 8/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 381s 529ms/step - accuracy: 0.7322 - loss: 0.8083 - val_accuracy: 0.7187 - val_loss: 0.8285
Epoch 9/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 379s 525ms/step - accuracy: 0.7492 - loss: 0.7520 - val_accuracy: 0.7784 - val_loss: 0.6636
Epoch 10/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 382s 524ms/step - accuracy: 0.7632 - loss: 0.7142 - val_accuracy: 0.7836 - val_loss: 0.6456
Epoch 11/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 380s 522ms/step - accuracy: 0.7796 - loss: 0.6750 - val_accuracy: 0.7811 - val_loss: 0.6751
Epoch 12/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 383s 523ms/step - accuracy: 0.7896 - loss: 0.6407 - val_accuracy: 0.7935 - val_loss: 0.6145
Epoch 13/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 328s 524ms/step - accuracy: 0.7986 - loss: 0.6146 - val_accuracy: 0.7980 - val_loss: 0.6090
Epoch 14/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 381s 523ms/step - accuracy: 0.8074 - loss: 0.5850 - val_accuracy: 0.7977 - val_loss: 0.6055
Epoch 15/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 381s 521ms/step - accuracy: 0.8189 - loss: 0.5543 - val_accuracy: 0.8171 - val_loss: 0.5566
Epoch 16/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 327s 523ms/step - accuracy: 0.8253 - loss: 0.5378 - val_accuracy: 0.8083 - val_loss: 0.5801
Epoch 17/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 380s 520ms/step - accuracy: 0.8368 - loss: 0.5092 - val_accuracy: 0.8225 - val_loss: 0.5395
Epoch 18/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 386s 527ms/step - accuracy: 0.8388 - loss: 0.4906 - val_accuracy: 0.7997 - val_loss: 0.6180
Epoch 19/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 381s 525ms/step - accuracy: 0.8429 - loss: 0.4831 - val_accuracy: 0.8051 - val_loss: 0.5956
Epoch 20/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 382s 525ms/step - accuracy: 0.8481 - loss: 0.4679 - val_accuracy: 0.8028 - val_loss: 0.6046
Epoch 21/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 328s 525ms/step - accuracy: 0.8527 - loss: 0.4422 - val_accuracy: 0.7617 - val_loss: 0.8034
Epoch 22/50
625/625 ━━━━━━━━━━━━━━━━━━━━ 381s 524ms/step - accuracy: 0.8595 - loss: 0.4319 - val_accuracy: 0.8176 - val_loss: 0.5739
```

85.95% training accuracy
81.76% validatio accuracy

EVALUATE THE MODEL

```
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=2)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

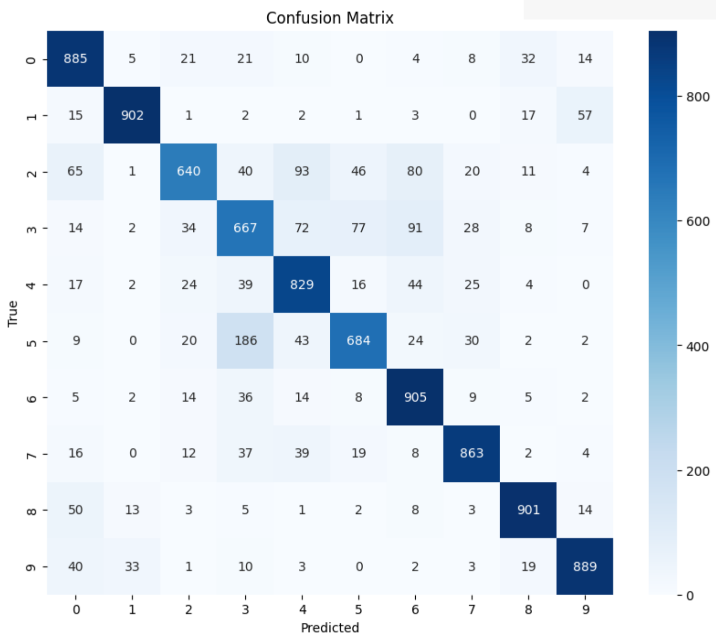
```
313/313 - 23s - 72ms/step - accuracy: 0.8355 - loss: 0.5467
Test Loss: 0.5466769933700562
Test Accuracy: 0.8355000019073486
```

Test Loss is 0.5467
Test Accuracy is 83.55%

Achieved an impressive test accuracy of 83.55%, depicting the model's ability to handle complex image classification tests

CONFUSION MATRIX AND CLASSIFICATION REPORT

```python
# Predict class probabilities
y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1)
y_test_classes = y_test.argmax(axis=1)

# Confusion matrix
cm = confusion_matrix(y_test_classes, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=range(10), yticklabels=range(10))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Classification report
print(classification_report(y_test_classes, y_pred_classes))
```




```
              precision    recall  f1-score   support

           0       0.79      0.89      0.84      1000
           1       0.94      0.90      0.92      1000
           2       0.83      0.64      0.72      1000
           3       0.64      0.67      0.65      1000
           4       0.75      0.83      0.79      1000
           5       0.80      0.68      0.74      1000
           6       0.77      0.91      0.83      1000
           7       0.87      0.86      0.87      1000
           8       0.90      0.90      0.90      1000
           9       0.90      0.89      0.89      1000

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.82     10000
weighted avg       0.82      0.82      0.82     10000
```

CALCULATE METRICS FOR EACH CLASS

```python
# Calculate metrics for each class
num_classes = cm.shape[0]  # Number of classes
for class_idx in range(num_classes):
    # True Positives (TP)
    tp = cm[class_idx, class_idx]

    # False Positives (FP)
    fp = cm[:, class_idx].sum() - tp

    # False Negatives (FN)
    fn = cm[class_idx, :].sum() - tp

    # True Negatives (TN)
    tn = cm.sum() - (tp + fp + fn)

    # Store the results
    metrics[class_idx] = {
        'True Positives': tp,
        'False Positives': fp,
        'False Negatives': fn,
        'True Negatives': tn
    }

# Print the results for each class
for class_idx, values in metrics.items():
    print(f"Class {class_idx}:")
    print(f"  True Positives: {values['True Positives']}")
    print(f"  False Positives: {values['False Positives']}")
    print(f"  False Negatives: {values['False Negatives']}")
    print(f"  True Negatives: {values['True Negatives']}")
    print()
```

```
Class 0:
  True Positives: 885
  False Positives: 231
  False Negatives: 115
  True Negatives: 8769

Class 1:
  True Positives: 902
  False Positives: 58
  False Negatives: 98
  True Negatives: 8942

Class 2:
  True Positives: 640
  False Positives: 130
  False Negatives: 360
  True Negatives: 8870

Class 3:
  True Positives: 667
  False Positives: 376
  False Negatives: 333
  True Negatives: 8624

Class 4:
  True Positives: 829
  False Positives: 277
  False Negatives: 171
  True Negatives: 8723

Class 5:
  True Positives: 684
  False Positives: 169
  False Negatives: 316
  True Negatives: 8831

Class 6:
  True Positives: 905
  False Positives: 264
  False Negatives: 95
  True Negatives: 8736
```

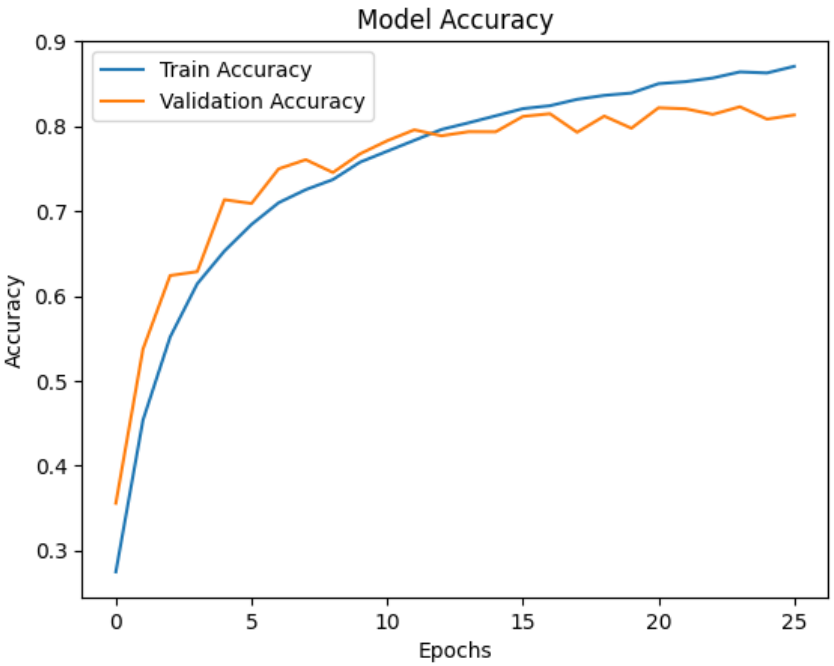
```
Class 7:
  True Positives: 863
  False Positives: 126
  False Negatives: 137
  True Negatives: 8874

Class 8:
  True Positives: 901
  False Positives: 100
  False Negatives: 99
  True Negatives: 8900

Class 9:
  True Positives: 889
  False Positives: 104
  False Negatives: 111
  True Negatives: 8896
```

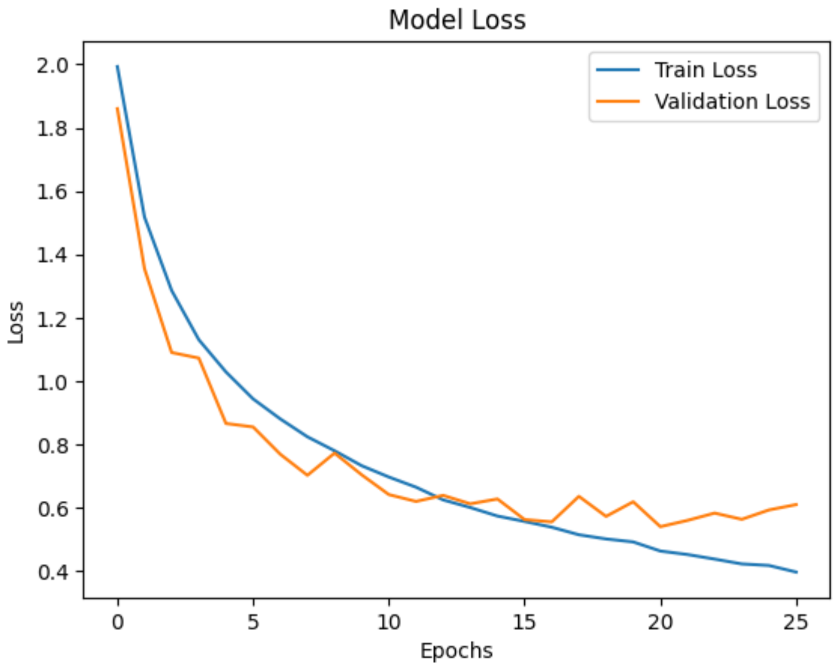
MODEL ACCURACY & MODEL LOSS

The accuracy graph shows how well the model is learning, while the loss graph indicates how well the model is minimizing errors.

```python
# Plot training and validation loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')
plt.show()
```

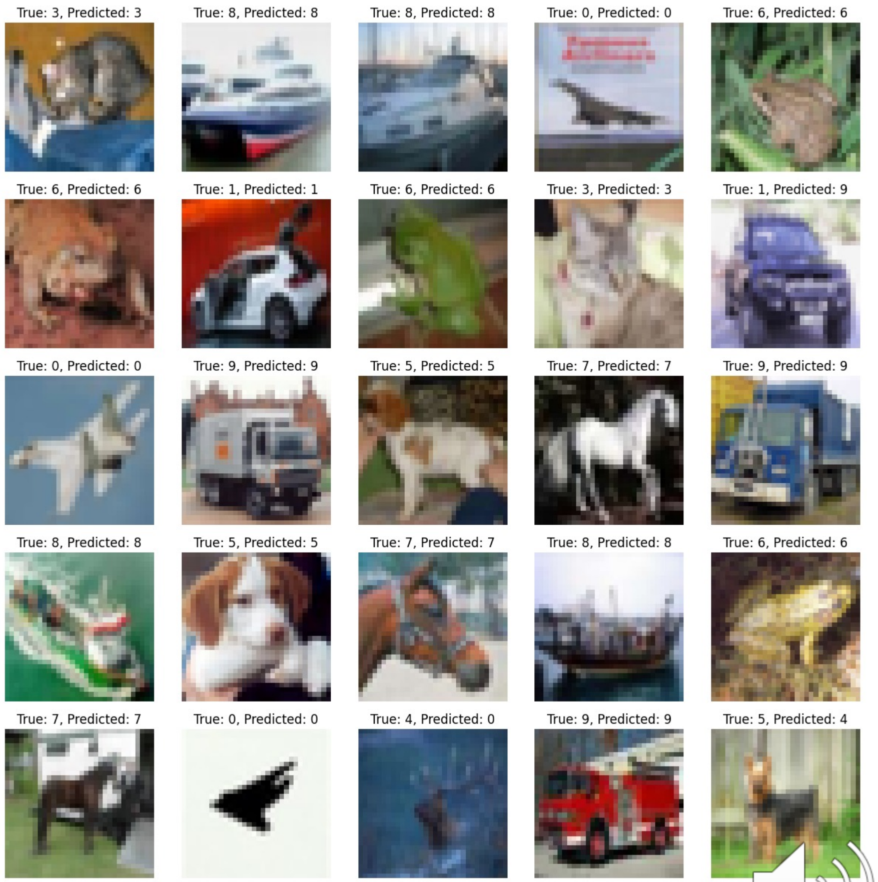

```python
# Plot training and validation loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')
plt.show()
```



VISUALISE SOME PREDICTIONS

```
# Visualize some predictions
fig, axes = plt.subplots(5, 5, figsize=(12, 12))
for i, ax in enumerate(axes.flat):
 ax.imshow(X_test[i])
 ax.set_title(f"True: {y_test_classes[i]}, Predicted: {y_pred_classes[i]}")
 ax.axis('off')
plt.tight_layout()
plt.show()
```



PERFORMACE INSIGHTS FROM SAMPLE PREDICTIONS

The model demonstrates strong performance in correctly identifying multiple instances across various classes:

Class	Predictions
Class 3 (Cat)	2 correct predictions.
Class 8 (Ship)	4 correct predictions.
Class 0 (Airplane)	3 correct predictions
Class 6 (Frog):	4 correct predictions
Class 1 (Automobile)	2 correct predictions.
Class 9 (Truck)	3 correct predictions
Class 5 (Dog)	2 correct predictions
Class 7 (Horse)	3 correct predictions
Class 4 (Deer)	1 correct prediction.

Misclassification

- Class 5 (Dog): 1 instance incorrectly classified for Class 4 (Deer), showing that it is not trivial to differentiate visually similar grouped classes.

Key Summary

- The model finds clear visual patterns within classes such as ships, airplanes, and frogs.
- Misclassifications, e.g., mistaking dogs as deer, reveal gaps where the model is particularly challenged by visually similar features, suggesting a possible requirement to improve further the process of learning features.

INSIGHTS INTO NEURAL NETWORK DESIGN STRATEGY

- **Data Preprocessing:** The implemented normalisation techniques along with one-hot encoding helped the model to converge faster and more efficiently.

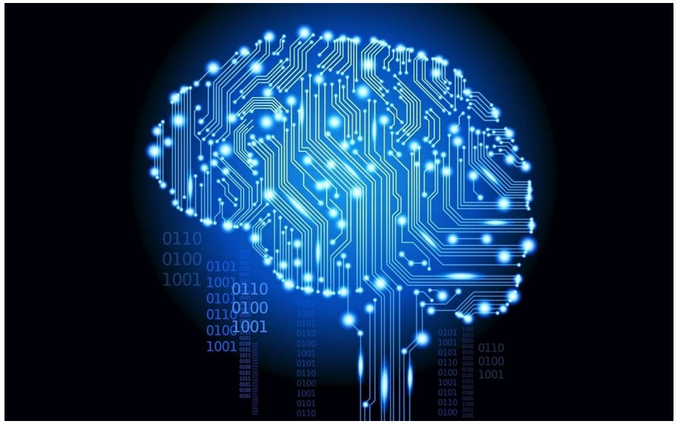
- **Design** : Multiple convolutional blocks were used, followed by fully connected layers to implement the network. The hierarchical nature of CNN allows the model to progressively learn features from low level to high level features. Each convolutional block has
  - Convolutional layers
  - Batch normalization
  - Max-pooling layers
  - Drop out layers

- **Hyperparameters :** Number of filters in convolution layers increase with depth (32 → 64 → 128) It helps capturing more abstract and complex features. Adam optimizer is utilized because as it is suited for image classification, and it automatically adjusts learning rate.

- **Activation & Loss Functions**: ReLU and softmax worked effectively for feature learning and classification.

- **Training :** Early stopping was used to monitor the validation loss, halting training when no improvement was made for 5 consecutive epochs. Early stopping prevents overfitting and reduces unnecessary computations.

REFLECTIONS ON LEARNINGS



- Practical experience on Neural Network Models
- Handling datasets
- Key tools like TensorFlow and Kera
- Evaluating performance
- Utilising a GPU instead of a CPU for training
- Ethics and Fairness
- Collaborating with Peers
- Continuous learning and Industry trends

Github Link : https://github.com/m-kanuri/m-kanuri.github.io/blob/main/NeualNetworkDesign.ipynb

REFERENCES

- Krizhevsky, A., Nair, V. and Hinton, G., 2009. *CIFAR-10 dataset.* [online] Available at: https://www.cs.toronto.edu/~kriz/cifar.html [Accessed 11 January 2025].
- Perez, L. and Wang, J. (2017) 'The Effectiveness of Data Augmentation in Image Classification using Deep Learning', *arXiv preprint arXiv:1712.04621*. Available at: https://arxiv.org/abs/1712.04621 (Accessed: 5 January 2025).
- Cabello-Solorzano, K., Ortigosa de Araujo, I., Peña, M., Correia, L., and Tallón-Ballesteros, A.J. (2023) 'The Impact of Data Normalization on the Accuracy of Machine Learning Algorithms: A Comparative Analysis', in *Advances in Intelligent Data Analysis XXI*. Springer, pp. 399–411. Available at: https://link.springer.com/content/pdf/10.1007/978-3-031-42536-3_33.pdf (Accessed: 5 January 2025).
- Raj, M., Ravi, V. & Kos, A. (2023). 'Artificial Intelligence: Evolution, Developments, Applications, and Future Scope', *Przeglad Elektrotechniczny*, vol. 2023, pp. 1-13. doi: 10.15199/48.2023.02.01.
- Zaczyński, B., 2023. *Split Your Dataset With scikit-learn's train_test_split()*. Real Python. Available at: https://realpython.com/train-test-split-python-data/ [Accessed 12 January 2025].
- Brownlee, J. (2021) *Train, Validation, and Test Sets for Machine Learning,* Machine Learning Mastery. Available at: https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/ (Accessed: 19 January 2025).
- Murphy, K.P. (2012) *Machine Learning: A Probabilistic Perspective.* Cambridge: MIT Press.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* Springer.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning.* Springer.
- Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection.
- Prechelt, L. (1998). "Early Stopping—But When?" *Neural Networks: Tricks of the Trade.*
- Chollet, F. (2021). *Deep Learning with Python.* Manning Publications.
- Géron, A., 2019. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* 2nd ed. Sebastopol, CA: O'Reilly Media.
- Goodfellow, I., Bengio, Y., & Courville, A., 2016. *Deep Learning.* Cambridge, MA: MIT Press.
- Ogundokun, R.O., Maskeliūnas, R., Misra, S. and Damaševičius, R., 2022. *Hybrid InceptionV3-SVM-Based Approach for Human Posture Detection in Health Monitoring Systems*. Algorithms, 15(11), p.410. Available at: https://www.mdpi.com/1999-4893/15/11/410 [Accessed 11 January 2025].
- Zou, Z., Chen, K., Shi, Z., Guo, Y. and Ye, J., 2019. *Object Detection in 20 Years: A Survey*. [online] arXiv. Available at: https://arxiv.org/abs/1905.05055 [Accessed 11 January 2025].
- Towards Data Science, 2021. *Understanding Activation Functions in Neural Networks.* [online] Available at: https://towardsdatascience.com [Accessed 11 January 2025].
- Analytics Vidhya, 2021. *Activation Functions in Neural Networks.* [online] Available at: https://www.analyticsvidhya.com [Accessed 15 January 2025].

Thank You