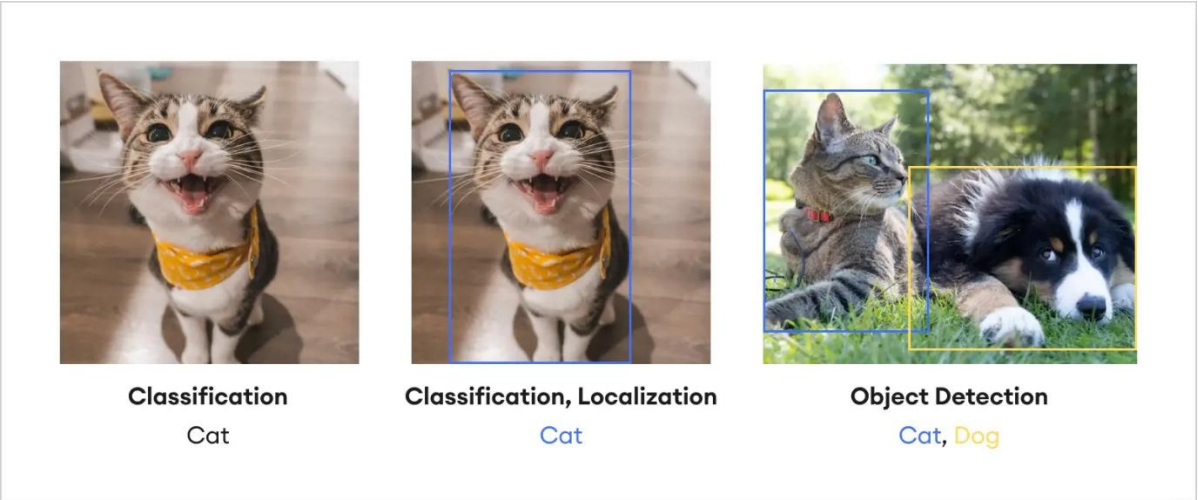
## ASSIGNMENT 2: IMAGE CLASSIFICATION MODEL BY DIVIDING THE MODEL INTO FOLLOWING 4 STAGES



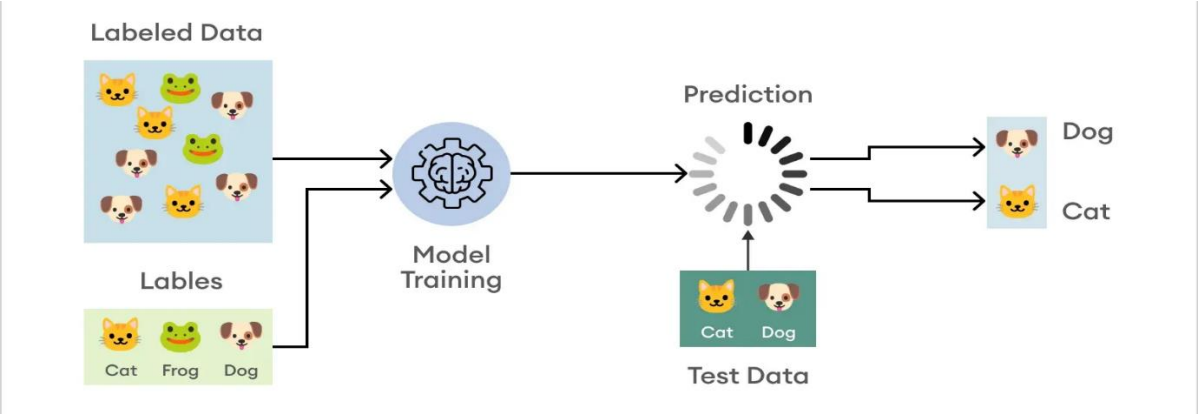




**Image Classification** in deep learning is a task where a model is trained to recognize and classify objects within images. The model learns patterns and features that distinguish one class from another, enabling it to make predictions on new, unseen images.

### 1. Convolutional Neural Networks (CNNs)

- **Purpose**: CNNs are specialized neural networks for image data. They automatically learn spatial hierarchies in images and are highly effective for image classification tasks.

- **Layers in CNN**:

    - **Convolutional Layer**: Applies filters to extract features like edges, textures, and shapes.

    - **Pooling Layer**: Reduces the spatial size of feature maps, preserving important information while reducing computation (e.g., max pooling).

- **Fully Connected (Dense) Layer**: At the end of the network, neurons connect to all outputs from the previous layer to make final classification decisions.

**2. Image Preprocessing**

- **Normalization**: Scaling pixel values (e.g., from [0, 255] to [0, 1]) to make training faster and more stable.
- **Resizing**: Adjusting image dimensions to a fixed size, often required by CNNs for consistent input shapes.
- **Data Augmentation**: Artificially expanding the training set by applying transformations like rotation, flipping, cropping, or adding noise. This helps reduce overfitting and improves generalization.

```
[1]  import numpy as np
     import tensorflow as tf
     from tensorflow.keras.datasets import mnist
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
     import matplotlib.pyplot as plt

[2]  print("[INFO] accessing MNIST...")
     (x_train, y_train), (x_test, y_test) = mnist.load_data()

     x_train = x_train.reshape((x_train.shape[0], 28, 28, 1)).astype('float32') / 255
     x_test = x_test.reshape((x_test.shape[0], 28, 28, 1)).astype('float32') / 255
```

 **Loading Data**:

- mnist.load_data() loads the dataset, giving training and test sets.
- x_train, y_train: Training images and labels.
- x_test, y_test: Test images and labels.

 **Reshaping and Normalizing**:

- **Reshape**: Adjusts images to a shape of (28, 28, 1), with 1 as the channel dimension since the images are grayscale.
- **Normalization**: Scales pixel values from [0, 255] to [0, 1] to improve training efficiency and stability.

```
[3]  model = Sequential()

     # Convolutional layers
     model.add(Conv2D(28, kernel_size=(3, 3), input_shape=(28, 28, 1)))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Flatten())

     # Fully connected layers
     model.add(Dense(200, activation="relu"))
     model.add(Dropout(0.3))
     model.add(Dense(10, activation="softmax"))
     model.summary()
```

**Layers Explained:**

- **Convolutional Layers**:
    - Conv2D(28, kernel_size=(3, 3), input_shape=(28, 28, 1)): Adds a convolutional layer with 28 filters and a 3x3 kernel size. The input_shape specifies the shape of each input image as (28, 28, 1).
    - **Purpose**: Extracts low-level features such as edges and textures from images.
    - MaxPooling2D(pool_size=(2, 2)): Adds a pooling layer with a 2x2 pool size. This reduces the feature map dimensions by half, retaining only the most important information from each feature map.
- **Flatten Layer**:
    - Flatten(): Converts the 2D feature maps from the convolutional layers into a 1D vector, preparing it for the fully connected layers.
- **Fully Connected Layers**:
    - Dense(200, activation="relu"): A dense (fully connected) layer with 200 neurons and ReLU activation, which introduces non-linearity and helps the model learn complex patterns.
    - Dropout(0.3): Randomly drops 30% of the neurons during training to prevent overfitting.
    - Dense(10, activation="softmax"): The output layer with 10 neurons (one for each digit class, 0-9) and softmax activation to output probabilities for each class.
- **Model Summary**:
    - model.summary(): Prints a summary of the model architecture, including the layer types, shapes, and parameters. **REST IS SAME AS ASSIGNMENT @**