1. Introduction.¶ The CIFAR-10 dataset contains 60,000 color images of 32 x 32 pixels in 3 channels divided into 10 classes. Each class contains 6,000 images. The training set contains 50,000 images, while the test sets provides 10,000 images. This image taken from the CIFAR repository (https://www.cs.toronto.edu/~kriz/cifar.html). This is a classification problem with 10 classes(muti-label classification). We can take a view on this image for more comprehension of the dataset image.pngimage.png .

```
import tensorflow as tf

# CIFAR-10 Image Classification using CNN
# Step 1: Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import datasets, layers, models

(x_train, y_train), (x_test, y_test) = datasets.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}")
```

```
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
x_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
```

Here we can see we have 5000 training images and 1000 test images as specified above and all the images are of 32 by 32 size and have 3 color channels i.e. images are color images. As well as it is also visible that there is only a single label assigned with each image.

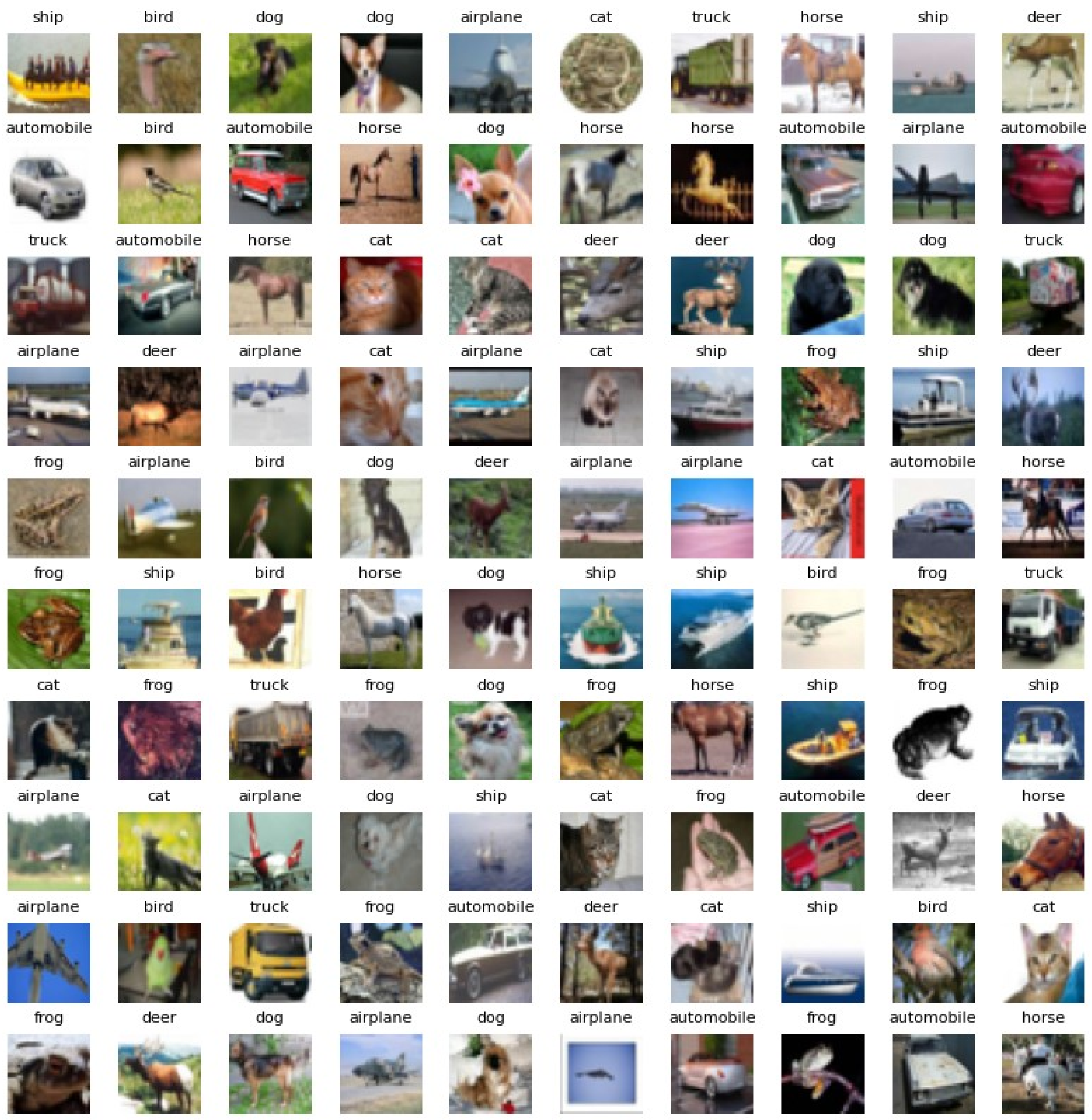
Until now, we have our data with us. But still, we cannot be sent it directly to our neural network. We need to process the data in order to send it to the network. The first thing in the process is to reduce the pixel values. Currently, all the image pixels are in a range from 1-256, and we need to reduce those values to a value ranging between 0 and 1. This enables our model to easily track trends and efficient training. We can do this simply by dividing all pixel values by 255.0. Another thing we want to do is to flatten(in simple words rearrange them in form of a row) the label values using the flatten() function.

```
# Reduce pixel values
# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# flatten the label values
y_train, y_test = y_train.flatten(), y_test.flatten()

# number of classes
K = len(set(y_train))
```

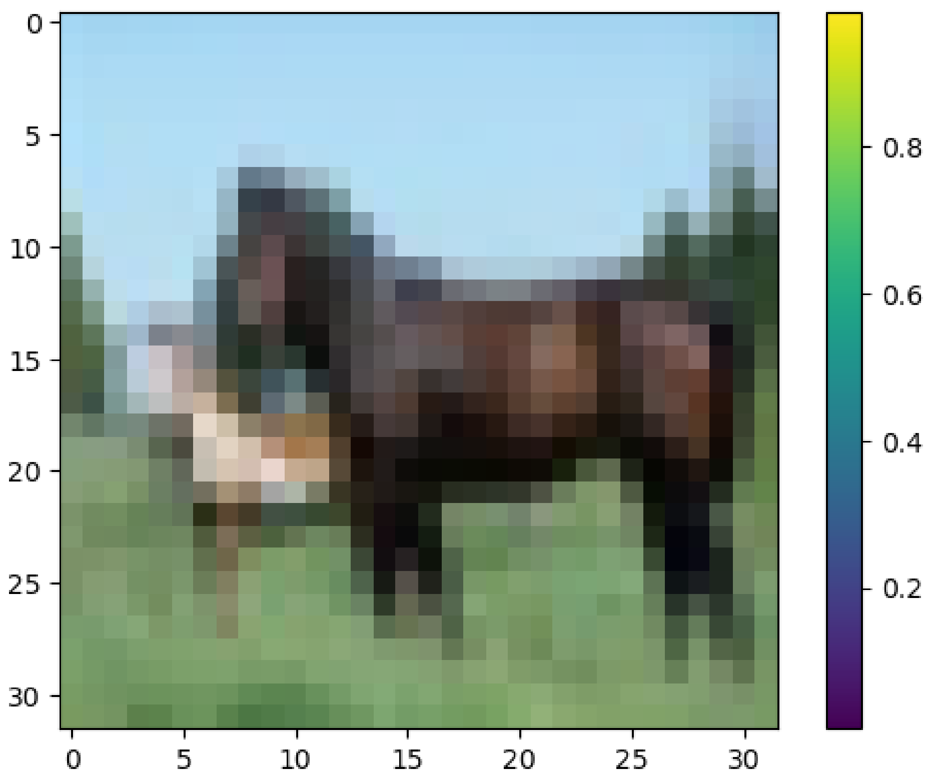
```
# calculate total number of classes
# for output layer
print("number of classes:", K)

number of classes: 10
```

Data Visualization¶ Verify the data: To verify that the dataset looks correct, let's plot the first 25 images from the training set and display the class name below each image:

```
# Define the labels of the dataset
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',
'frog', 'horse', 'ship', 'truck']

# Let's view more images in a grid format
# Define the dimensions of the plot grid
W_grid = 10
L_grid = 10

# fig, axes = plt.subplots(L_grid, W_grid)
# subplot return the figure object and axes object
# we can use the axes object to plot specific figures at various
locations

fig, axes = plt.subplots(L_grid, W_grid, figsize = (10,10))

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

n_train = len(x_train) # get the length of the train dataset

# Select a random number from 0 to n_train
for i in np.arange(0, W_grid * L_grid): # create evenly spaces
variables

    # Select a random number
    index = np.random.randint(0, n_train)
    # read and display an image with the selected index
    axes[i].imshow(x_train[index,1:])
    label_index = int(y_train[index])
    axes[i].set_title(labels[label_index], fontsize = 8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```



```
plt.figure()
plt.imshow(x_train[12])
plt.colorbar()

<matplotlib.colorbar.Colorbar at 0x1bed55bee70>
```



```
# Step 4: Build the CNN Model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,
3)),
    layers.MaxPooling2D((2, 2)),

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

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

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 output units for the
10 classes
])

# View the model summary
model.summary()

Model: "sequential_1"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                 ┃
```

```
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ conv2d_3 (Conv2D)                     │ (None, 30, 30, 32)           │
896 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ max_pooling2d_3 (MaxPooling2D)        │ (None, 15, 15, 32)           │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ conv2d_4 (Conv2D)                     │ (None, 13, 13, 64)           │
18,496 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ max_pooling2d_4 (MaxPooling2D)        │ (None, 6, 6, 64)             │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ conv2d_5 (Conv2D)                     │ (None, 4, 4, 64)             │
36,928 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ max_pooling2d_5 (MaxPooling2D)        │ (None, 2, 2, 64)             │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ flatten_1 (Flatten)                   │ (None, 256)                  │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_2 (Dense)                       │ (None, 64)                   │
16,448 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_3 (Dense)                       │ (None, 10)                   │
650 │
└───────────────────────────────────────┴──────────────────────────────┴
─────────────────┘

 Total params: 73,418 (286.79 KB)

 Trainable params: 73,418 (286.79 KB)

 Non-trainable params: 0 (0.00 B)
```

Conv2D Layer (conv2d):

Output Shape: (None, 30, 30, 32) Parameters: 896 This is the first convolutional layer. It takes an input of shape (32, 32, 3) and applies 32 filters of size (3x3). MaxPooling2D Layer (max_pooling2d):

Output Shape: (None, 15, 15, 32) Parameters: 0 This layer performs max pooling with a (2x2) window, reducing the spatial dimensions from (30, 30) to (15, 15). Conv2D Layer (conv2d_1):

Output Shape: (None, 13, 13, 64) Parameters: 18,496 The second convolutional layer, applying 64 filters of size (3x3). MaxPooling2D Layer (max_pooling2d_1):

Output Shape: (None, 6, 6, 64) Parameters: 0 Another max pooling layer, reducing the spatial dimensions from (13, 13) to (6, 6). Conv2D Layer (conv2d_2):

Output Shape: (None, 4, 4, 64) Parameters: 36,928 A third convolutional layer, applying 64 filters of size (3x3). MaxPooling2D Layer (max_pooling2d_2):

Output Shape: (None, 2, 2, 64) Parameters: 0 The final max pooling layer, reducing the spatial dimensions to (2, 2). Flatten Layer (flatten):

Output Shape: (None, 256) Parameters: 0 Flattens the (2x2x64) output into a single vector of size 256. Dense Layer (dense):

Output Shape: (None, 64) Parameters: 16,448 A fully connected layer with 64 units and ReLU activation. Dense Layer (dense_1):

Output Shape: (None, 10) Parameters: 650 The output layer with 10 units (for 10 classes in CIFAR-10) and softmax activation.

The CNN model shown in the summary has 9 layers in total. Here is the breakdown:

Conv2D (conv2d) MaxPooling2D (max_pooling2d) Conv2D (conv2d_1) MaxPooling2D (max_pooling2d_1) Conv2D (conv2d_2) MaxPooling2D (max_pooling2d_2) Flatten Dense (dense) Dense (dense_1)

```
# Step 5: Compile the Model
model.compile(optimizer='adam',

loss='sparse_categorical_crossentropy',metrics=['accuracy'])

# Step 6: Train the Model
history = model.fit(x_train, y_train,
epochs=10,validation_data=(x_test, y_test))


Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 12ms/step - accuracy: 0.2870 -
loss: 2.7930 - val_accuracy: 0.4809 - val_loss: 1.4280
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - accuracy: 0.4915 -
loss: 1.4182 - val_accuracy: 0.5390 - val_loss: 1.2972
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.5559 -
```

```
loss: 1.2550 - val_accuracy: 0.5484 - val_loss: 1.3016
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 14ms/step - accuracy: 0.5900 -
loss: 1.1668 - val_accuracy: 0.5921 - val_loss: 1.1709
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.6137 -
loss: 1.0946 - val_accuracy: 0.6058 - val_loss: 1.1391
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.6403 -
loss: 1.0351 - val_accuracy: 0.6244 - val_loss: 1.0790
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.6619 -
loss: 0.9652 - val_accuracy: 0.6369 - val_loss: 1.0687
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.6786 -
loss: 0.9334 - val_accuracy: 0.6302 - val_loss: 1.0774
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 13ms/step - accuracy: 0.6882 -
loss: 0.8946 - val_accuracy: 0.6575 - val_loss: 1.0149
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.6974 -
loss: 0.8531 - val_accuracy: 0.6379 - val_loss: 1.0576
```

```python
# Step 7: Evaluate the Model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"\nTest accuracy: {test_acc}")
```

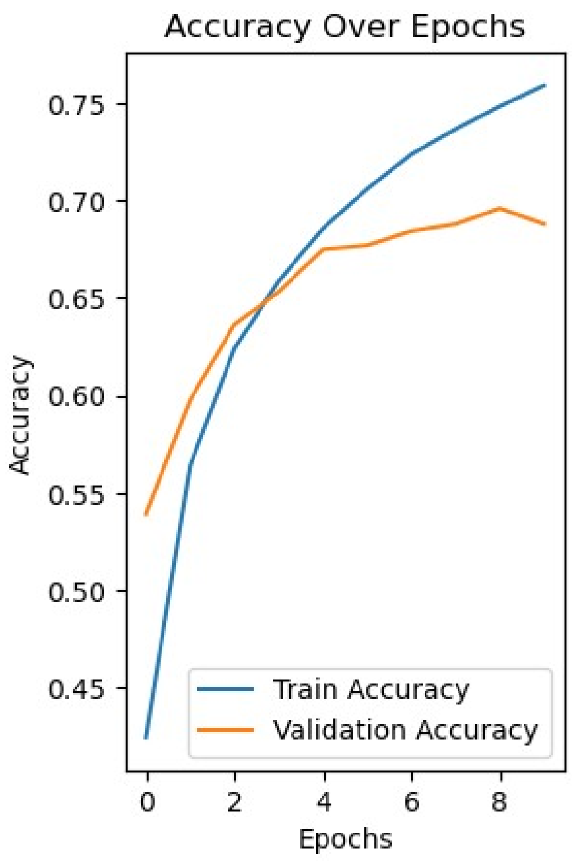
```
313/313 - 1s - 4ms/step - accuracy: 0.6379 - loss: 1.0576

Test accuracy: 0.6378999948501587
```

```python
# Step 8: Visualize Training and Validation Accuracy and Loss
plt.figure(figsize=(12, 4))
```

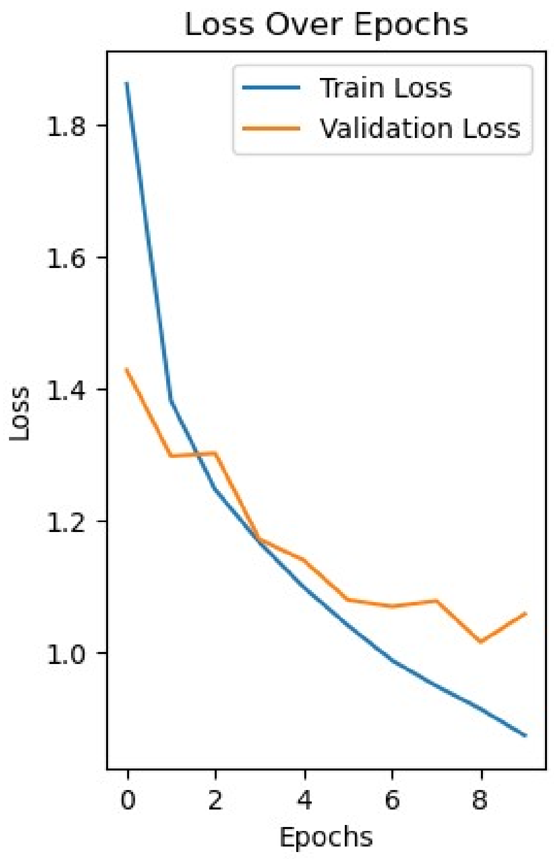
```
<Figure size 1200x400 with 0 Axes>

<Figure size 1200x400 with 0 Axes>
```

```python
# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy Over Epochs')
```

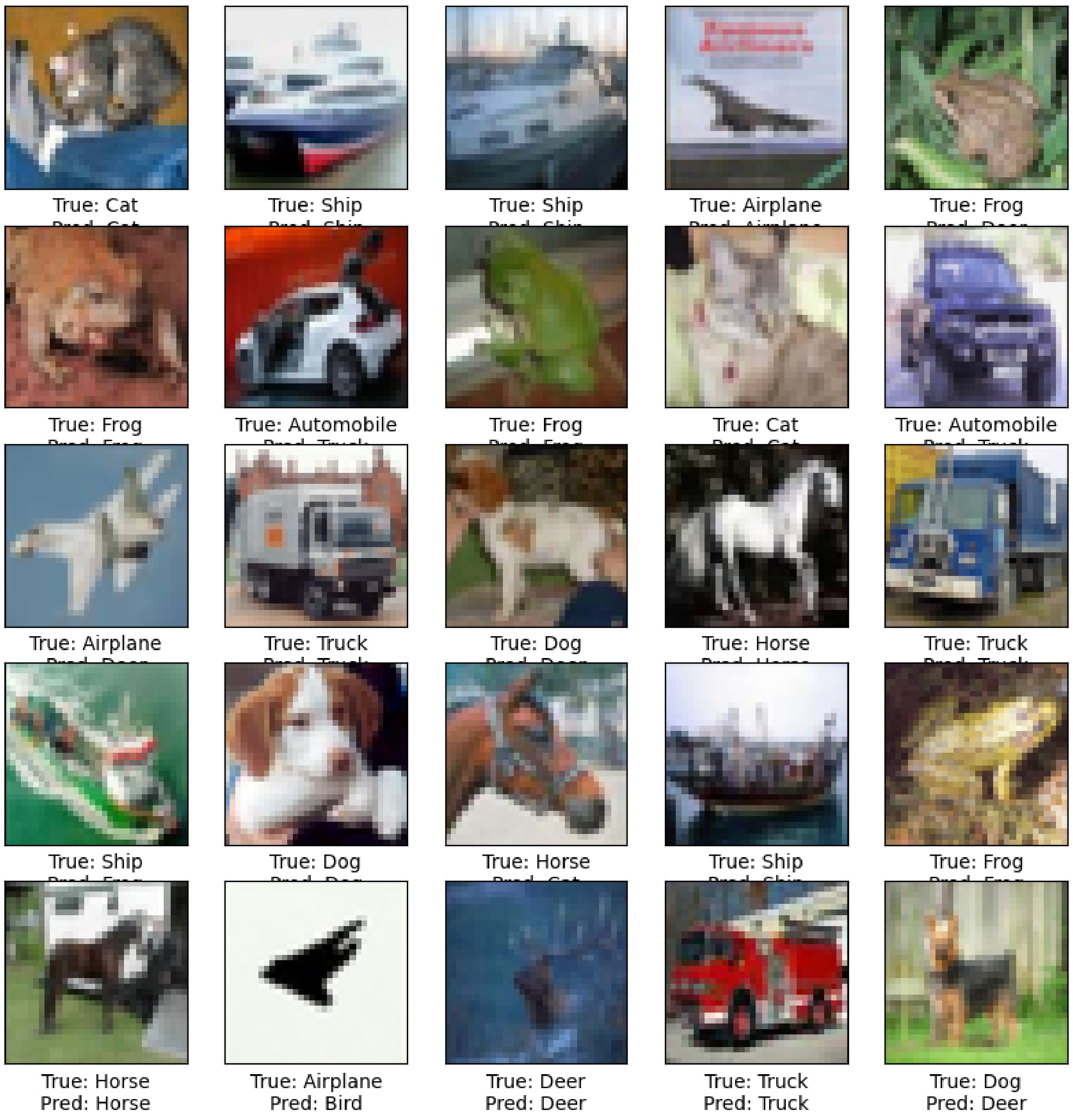
```
Text(0.5, 1.0, 'Accuracy Over Epochs')
```



```
# Loss plot
plt.subplot(1, 2, 2)
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('Loss Over Epochs')

plt.show()
```



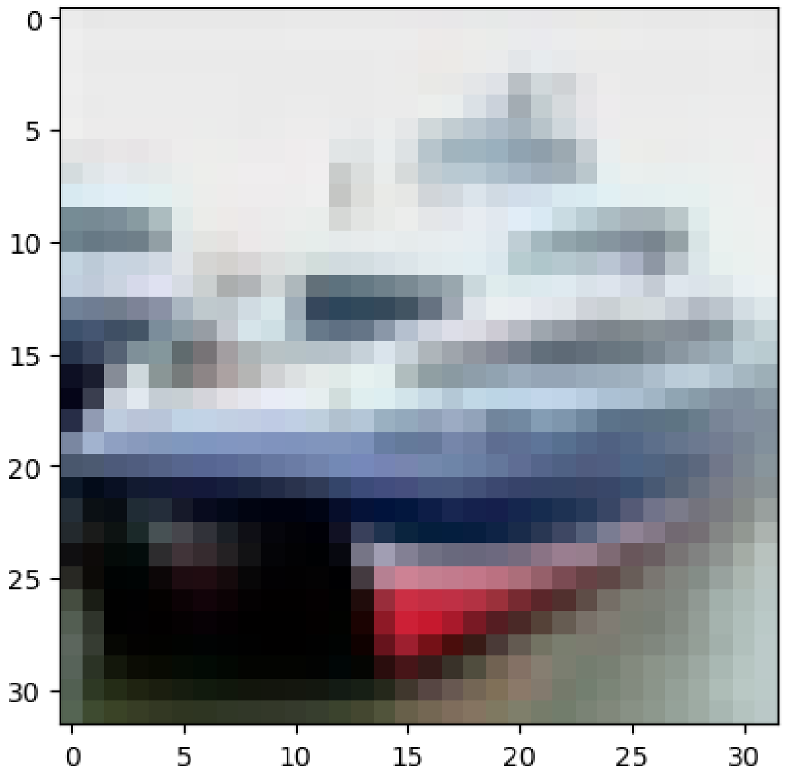
```
# Step 9: Make Predictions on the Test Data
predictions = model.predict(x_test)


313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step

# Visualize some predictions
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test[i])
    plt.xlabel(f"True: {class_names[y_test[i][0]]}\nPred:
{class_names[np.argmax(predictions[i])]}")
plt.show()
```



```python
import numpy as np
# Function to predict and display an image from test data
def predict_image(index):
    img = x_test[index]
    prediction = model.predict(np.expand_dims(img, axis=0))
    predicted_class = np.argmax(prediction)
    plt.imshow(img)
    plt.title(f"Predicted: {class_names[predicted_class]},
Actual:↪{class_names[y_test[index][0]]}")
    plt.show()
```

```
# Predict an example image from test set
predict_image(1)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
```

Predicted: Automobile, Actual:↪Ship



```
To increase the accuracy from 63 %, we Use a Deeper Model
Increase the depth of your CNN by adding more convolutional layers or
increase the number of filters in existing layers to capture more
complex features.

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,
3)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

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

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.Conv2D(128, (3, 3), activation='relu'),
   # layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

```
C:\Users\Nasreen\anaconda3\Lib\site-packages\keras\src\layers\
convolutional\base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
# View the model summary
model.summary()
```

```
Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                 ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                      │ (None, 30, 30, 32)           │
896 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 28, 28, 32)           │
9,248 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 14, 14, 32)           │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ conv2d_2 (Conv2D)                    │ (None, 12, 12, 64)           │
18,496 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ conv2d_3 (Conv2D)                    │ (None, 10, 10, 64)           │
36,928 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 5, 5, 64)             │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ conv2d_4 (Conv2D)                    │ (None, 3, 3, 128)            │
73,856 │
```

```
├──────────────────────────────────────┼───────────────────────────┼
─────────────────┤
│ conv2d_5 (Conv2D)                    │ (None, 1, 1, 128)         │
147,584 │
├──────────────────────────────────────┼───────────────────────────┼
─────────────────┤
│ flatten (Flatten)                    │ (None, 128)               │
0 │
├──────────────────────────────────────┼───────────────────────────┼
─────────────────┤
│ dense (Dense)                        │ (None, 256)               │
33,024 │
├──────────────────────────────────────┼───────────────────────────┼
─────────────────┤
│ dense_1 (Dense)                      │ (None, 10)                │
2,570 │
└──────────────────────────────────────┴───────────────────────────┴
─────────────────┘

 Total params: 322,602 (1.23 MB)

 Trainable params: 322,602 (1.23 MB)

 Non-trainable params: 0 (0.00 B)
```

```
# Step 5: Compile the Model
model.compile(optimizer='adam',

loss='sparse_categorical_crossentropy',metrics=['accuracy'])

# Step 6: Train the Model
history = model.fit(x_train, y_train, batch_size=64,
epochs=10,validation_data=(x_test, y_test))
```

```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 108s 123ms/step - accuracy: 0.3059 -
loss: 1.8414 - val_accuracy: 0.5381 - val_loss: 1.2718
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 95s 121ms/step - accuracy: 0.5614 - loss:
1.2078 - val_accuracy: 0.6386 - val_loss: 1.0194
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 141s 120ms/step - accuracy: 0.6571 -
loss: 0.9596 - val_accuracy: 0.6752 - val_loss: 0.9128
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 94s 120ms/step - accuracy: 0.7150 - loss:
0.8106 - val_accuracy: 0.6792 - val_loss: 0.9603
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 89s 113ms/step - accuracy: 0.7516 - loss:
0.7121 - val_accuracy: 0.7136 - val_loss: 0.8305
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 91s 116ms/step - accuracy: 0.7842 - loss:
```

```
0.6112 - val_accuracy: 0.7383 - val_loss: 0.7642
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 92s 117ms/step - accuracy: 0.8036 - loss:
0.5572 - val_accuracy: 0.7271 - val_loss: 0.8338
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 93s 119ms/step - accuracy: 0.8311 - loss:
0.4836 - val_accuracy: 0.7450 - val_loss: 0.7842
Epoch 9/10
771/782 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - accuracy: 0.8449 - loss:
0.4384

# Step 7: Evaluate the Model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"\nTest accuracy: {test_acc}")

# Step 8: Visualize Training and Validation Accuracy and Loss
plt.figure(figsize=(12, 4))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Accuracy Over Epochs')

# Loss plot
plt.subplot(1, 2, 2)
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('Loss Over Epochs')

plt.show()
```