Code 1:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense
from tensorflow.keras.optimizers import SGD
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.datasets import cifar10

# Load and preprocess dataset
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
x_train = X_train.astype('float32') / 255.0
x_test = X_test.astype('float32') / 255.0

# Class names
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# Display sample images
plt.figure(figsize=(8,8))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i])
    plt.xlabel(class_names[Y_train[i][0]])
plt.show()

# Build CNN model
model = Sequential([
    Conv2D(32, (4, 4), input_shape=(32, 32, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(32, (4, 4), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile model
model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=SGD(learning_rate=0.01, momentum=0.9),
    metrics=['accuracy']
)

# Model summary
model.summary()

# Train model
history = model.fit(
    x_train, Y_train,
    epochs=20,
    batch_size=64,
    validation_data=(x_test, Y_test)
)

# Evaluate model
loss, accuracy = model.evaluate(x_test, Y_test)
print(f"Loss: {loss}")
print(f"Accuracy: {accuracy}")

# Prediction and visualization
plt.figure(figsize=(16, 10))
for i in range(25):
    index = np.random.randint(0, len(x_test) - 1)
    img = x_test[index]
    pred = model.predict(np.expand_dims(img, axis=0))
    predicted_label = class_names[np.argmax(pred)]
    true_label = class_names[Y_test[index][0]]

    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(img)
    color = 'green' if predicted_label == true_label else 'red'
    plt.xlabel(f"Pred: {predicted_label}\nTrue: {true_label}",
color=color)
plt.tight_layout()
plt.show()
```

Code 2

```
import os
import random
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, Sequential, optimizers
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split

tf.random.set_seed(42)
np.random.seed(42)
random.seed(42)

train_dir = "cifar-10-img/train"
test_dir  = "cifar-10-img/test"

ds_train = keras.preprocessing.image_dataset_from_directory(
    train_dir, labels="inferred", label_mode="int", image_size=(32,32),
    color_mode="rgb", batch_size=128, shuffle=True, seed=42
)
ds_test = keras.preprocessing.image_dataset_from_directory(
    test_dir, labels="inferred", label_mode="int", image_size=(32,32),
    color_mode="rgb", batch_size=128, shuffle=False
)

class_names = ds_train.class_names

def ds_to_numpy(dataset):
    imgs, labels = [], []
    for batch in dataset:
        b_x, b_y = batch
        imgs.append(b_x.numpy())
        labels.append(b_y.numpy())
    return np.vstack(imgs), np.hstack(labels)

x_all_train, y_all_train = ds_to_numpy(ds_train)
x_test, y_test = ds_to_numpy(ds_test)
x_all_train = x_all_train.astype("float32") / 255.0
```

```
x_test = x_test.astype("float32") / 255.0

x_train, x_val, y_train, y_val = train_test_split(
    x_all_train, y_all_train, test_size=0.1, random_state=42, shuffle=True
)

datagen = keras.preprocessing.image.ImageDataGenerator(
    width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True
)

batch_size = 64
train_gen = datagen.flow(x_train, y_train, batch_size=batch_size, shuffle=True, seed=42)
validation_data = (x_val, y_val)
steps_per_epoch = train_gen.n // batch_size
validation_steps = max(1, len(x_val) // batch_size)

def build_model():
    return Sequential([
        layers.Input(shape=(32,32,3)),
        layers.Conv2D(32, (3,3), padding='same', activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, (3,3), padding='same', activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2,2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

model = build_model()
sgd = optimizers.SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=sgd, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(
    train_gen,
    epochs=15,
    steps_per_epoch=steps_per_epoch,
    validation_data=validation_data,
    validation_steps=validation_steps,
    verbose=2
)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"\nTest loss: {test_loss:.4f} | Test accuracy: {test_acc:.4f}")

y_pred_probs = model.predict(x_test, batch_size=128, verbose=0)
y_pred = np.argmax(y_pred_probs, axis=1)
```

```
print("\nClassification report (precision / recall / f1):")
print(classification_report(y_test, y_pred, digits=4))
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix shape:", cm.shape)

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss'); plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], label='train acc')
plt.plot(history.history['val_accuracy'], label='val acc')
plt.title('Accuracy'); plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.legend()
plt.tight_layout()
plt.show()

def plot_image(img, pred_label, true_label, ax):
    ax.imshow(img)
    ax.set_title(f"Pred: {class_names[pred_label]}\nTrue: {class_names[true_label]}")
    ax.axis('off')

plt.figure(figsize=(16,10))
for i in range(20):
    idx = random.randint(0, len(x_test) - 1)
    image = x_test[idx]
    pred = np.argmax(model.predict(image[np.newaxis, ...], verbose=0)[0])
    ax = plt.subplot(4, 5, i + 1)
    plot_image(image, pred_label=pred, true_label=y_test[idx], ax=ax)
plt.tight_layout()
plt.show()
```

Theory:

Convolutional Neural Networks (CNNs) are a class of deep learning models widely used for image recognition and classification. CNNs consist of convolutional layers that automatically learn spatial hierarchies of features from images.

In this experiment, we use the **CIFAR-10** dataset, which contains 60,000 color images in 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).

The model uses **two convolutional layers** followed by **max-pooling layers** for feature extraction, and **fully connected layers** for classification.
The **Stochastic Gradient Descent (SGD)** optimizer is used for efficient weight updates, and the **softmax** activation function is used for multi-class output prediction.

Algorithm Steps:

1. Import TensorFlow, Keras, and other required libraries.
2. Load and normalize the CIFAR-10 dataset.
3. Visualize sample images from the dataset.
4. Build a CNN model with:
   - Convolutional + ReLU layers
   - MaxPooling layers
   - Flatten and Dense layers
5. Compile the model using SGD optimizer and categorical cross-entropy loss.
6. Train the model on the training data.
7. Evaluate model performance on test data.
8. Display predictions with true vs. predicted labels.

Output:

- Displays 25 sample CIFAR-10 images with class labels.
- Trains CNN for 20 epochs showing training and validation accuracy.
- Prints test accuracy and loss.
- Shows predicted vs. true labels for 25 random test images.

Result:

The CNN model successfully classifies CIFAR-10 images with good accuracy using the **SGD optimizer**.

Would you like me to add a **short “Conclusion” paragraph** for your practical file (1–2 lines summary)?