```
Start coding or generate with AI.
```

```
import os
import glob
import cv2
import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import sys

DRIVE_DATASET_FOLDER = "/content/drive/MyDrive/dataset"
OUTPUT_DIR = "/content/drive/MyDrive/smart_digit_outputs_folder"
IMAGE_EXTS = ("*.png", "*.jpg", "*.jpeg", "*.bmp")
FRAME_SIZE = (28, 28)
TRAIN_RATIO = 0.7
RANDOM_SEED = 42
BATCH_SIZE = 32
EPOCHS = 12
INTERACTIVE_LABEL = True

def ensure_dir(p):
    os.makedirs(p, exist_ok=True)

def mount_drive_if_colab():
    if 'google.colab' in sys.modules:
        from google.colab import drive
        drive.mount('/content/drive', force_remount=False)


def gather_images_and_auto_labels(dataset_folder):
    dataset_folder = Path(dataset_folder)
    images = []
    labels = []

    subdirs = [d for d in dataset_folder.iterdir() if d.is_dir()]
    digit_subdirs = [d for d in subdirs if d.name.isdigit()]
    if digit_subdirs:
        print("Detected digit subfolders; using subfolder names as labels.")
        for d in sorted(digit_subdirs, key=lambda x: x.name):
            for ext in IMAGE_EXTS:
                for p in d.glob(ext):
                    images.append(str(p))
                    labels.append(int(d.name))
        return images, labels

    labels_csv = dataset_folder / "labels.csv"
    if labels_csv.exists():
        print("Found labels.csv; using it.")
        df = pd.read_csv(str(labels_csv))

        for _, r in df.iterrows():
            fname = r['filename']
            label = int(r['label'])
            p = dataset_folder / fname
            if p.exists():
                images.append(str(p))
                labels.append(label)
        return images, labels

    print("No subfolders or labels.csv found — attempting to infer labels from filenames.")
    cand_files = []
    for ext in IMAGE_EXTS:
        cand_files.extend(sorted(map(str, dataset_folder.glob(ext))))
    if not cand_files:
        return [], []

    for p in cand_files:
        name = os.path.basename(p)

        assigned = None

        import re

        patterns = [
            r'^([0-9])[_\-\.\s]',
            r'[_\-\.\s]([0-9])[_\-\.\s]',
            r'[_\-\.\s]([0-9])$',
            r'^([0-9])$',
        ]
        for pat in patterns:
            m = re.search(pat, name)
            if m:
                assigned = int(m.group(1))
                break

        if assigned is None:
            digits = re.findall(r'([0-9])', name)
            if len(digits) == 1:
                assigned = int(digits[0])
        if assigned is not None:
            images.append(p)
            labels.append(assigned)
    return images, labels


def interactive_label_images(image_paths, labels_csv_path):
    print("Interactive labeling: for each image enter digit 0-9. Type 'q' to quit and save progress.")
    rows = []
    if os.path.exists(labels_csv_path):
        prev = pd.read_csv(labels_csv_path)
        rows = prev.to_dict('records')
        labeled_names = set(prev['filename'].astype(str).tolist())
    else:
        labeled_names = set()
    for p in image_paths:
        fname = os.path.basename(p)
        if fname in labeled_names:
            continue

        img = cv2.imread(str(p))
        if img is None:
            continue
        try:
            from google.colab.patches import cv2_imshow
            disp = cv2.resize(img, (240,240))
            cv2_imshow(disp)
        except Exception:
            tmp = os.path.join(os.path.dirname(labels_csv_path), "preview_tmp.png")
            cv2.imwrite(tmp, cv2.resize(img, (240,240)))
            print("Preview saved to", tmp)
        lab = input(f"Label for {fname} (0-9, q to quit): ").strip()
        if lab.lower() == 'q':
            break
        if lab not in [str(i) for i in range(10)]:
            print("Invalid; skipping.")
            continue
        rows.append({'filename': fname, 'label': int(lab)})
        df = pd.DataFrame(rows)
        df.to_csv(labels_csv_path, index=False)
    return pd.DataFrame(rows)


def load_images_and_labels_from_folder(dataset_folder, out_labels_csv):
    images, labels = gather_images_and_auto_labels(dataset_folder)
    if images and labels and len(images) >= 1:
        print(f"Auto-detected {len(images)} labeled images.")
        return list(images), list(labels)

    all_images = []
    for ext in IMAGE_EXTS:
        all_images.extend(sorted(map(str, Path(dataset_folder).glob(ext))))
    if not all_images:
        print("No images found in the dataset folder.")
        return [], []
    print(f"Found {len(all_images)} images but couldn't auto-assign labels.")
    if INTERACTIVE_LABEL:

        labels_df = interactive_label_images(all_images, out_labels_csv)
        labeled_images = []
        labeled_labels = []
        for _, r in labels_df.iterrows():
            p = os.path.join(dataset_folder, r['filename'])
            if os.path.exists(p):
                labeled_images.append(p)
                labeled_labels.append(int(r['label']))
        return labeled_images, labeled_labels
    else:
        print("Interactive labeling disabled. Place a labels.csv in the dataset folder or use subfolders r
        return [], []

def preprocess_image(path, target_size=(28,28)):
    img = cv2.imread(str(path), cv2.IMREAD_GRAYSCALE)
    if img is None:
        return None
    img = cv2.resize(img, target_size, interpolation=cv2.INTER_AREA)
    img = cv2.bitwise_not(img)
    img = img.astype(np.float32) / 255.0
    img = img[..., np.newaxis]
    return img

def build_simple_cnn(input_shape=(28,28,1), num_classes=10):
    model = models.Sequential([
        layers.Input(shape=input_shape),
        layers.Conv2D(32, (3,3), activation='relu', padding='same'),
        layers.MaxPool2D((2,2)),
        layers.Conv2D(64, (3,3), activation='relu', padding='same'),
        layers.MaxPool2D((2,2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

def compute_and_save_metrics(y_true, y_pred, labels_list, excel_path, plots_dir):
    cm = confusion_matrix(y_true, y_pred, labels=labels_list)
    precision = precision_score(y_true, y_pred, labels=labels_list, average=None, zero_division=0)
    recall = recall_score(y_true, y_pred, labels=labels_list, average=None, zero_division=0)
    f1 = f1_score(y_true, y_pred, labels=labels_list, average=None, zero_division=0)
    overall_acc = np.sum(np.diag(cm)) / np.sum(cm) if np.sum(cm)>0 else 0.0

    rows = []
    for i, lab in enumerate(labels_list):
        rows.append({
            'digit': lab,
            'detection_rate_recall': recall[i],
            'precision': precision[i],
            'f1_score': f1[i],
            'support': int(cm[i].sum())
        })
    df = pd.DataFrame(rows)
    ensure_dir(os.path.dirname(excel_path))
    with pd.ExcelWriter(excel_path) as writer:
        df.to_excel(writer, sheet_name='per_class_metrics', index=False)
        cm_df = pd.DataFrame(cm, index=[f"true_{l}" for l in labels_list], columns=[f"pred_{l}" for l in l
        cm_df.to_excel(writer, sheet_name='confusion_matrix')
        pd.DataFrame([{'overall_accuracy': overall_acc, 'total_samples': int(np.sum(cm))}]).to_excel(write

    ensure_dir(plots_dir)
    plt.figure(figsize=(8,6))
    plt.imshow(cm, interpolation='nearest')
    plt.title("Confusion Matrix")
    plt.colorbar()
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.xticks(range(len(labels_list)), labels_list)
    plt.yticks(range(len(labels_list)), labels_list)
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i,j], 'd'), ha='center', va='center', color='white' if cm[i,j] > cm.m
    plt.tight_layout()
    cm_plot = os.path.join(plots_dir, "confusion_matrix.png")
    plt.savefig(cm_plot)
    plt.close()
    print("Saved confusion matrix plot to", cm_plot)
    return df, cm, overall_acc


def main():
    np.random.seed(RANDOM_SEED)
    tf.random.set_seed(RANDOM_SEED)

    mount_drive_if_colab()
    ensure_dir(OUTPUT_DIR)
    labels_csv_out = os.path.join(OUTPUT_DIR, "labels.csv")

    imgs, labs = load_images_and_labels_from_folder(DRIVE_DATASET_FOLDER, labels_csv_out)
    if len(imgs) == 0:
        print("No labeled images available. Exiting.")
        return

    X = []
```

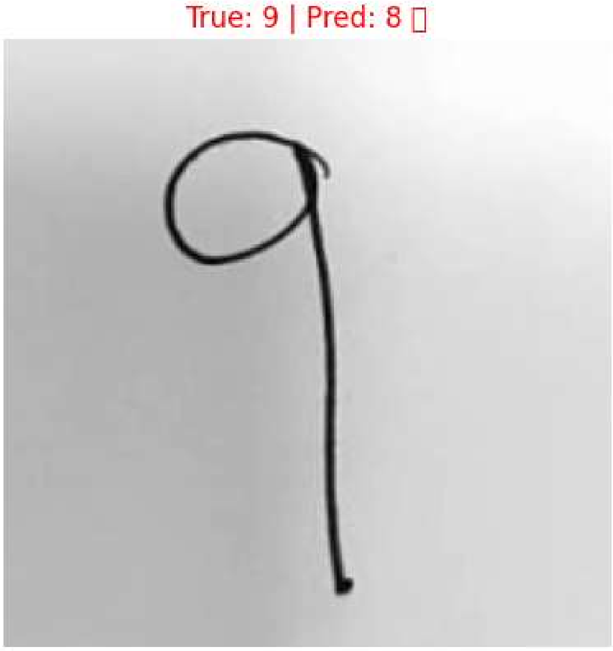
```
y = []
filenames = []
for p, lab in zip(imgs, labs):
    img = preprocess_image(p, target_size=FRAME_SIZE)
    if img is None:
        continue
    X.append(img)
    y.append(int(lab))
    filenames.append(os.path.basename(p))
X = np.stack(X, axis=0)
y = np.array(y)
print("Total loaded samples:", len(y))

X, y, filenames = shuffle(X, y, filenames, random_state=RANDOM_SEED)


unique, counts = np.unique(y, return_counts=True)
per_class_counts = dict(zip(unique, counts))
stratify = y if min(counts) >= 2 else None
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=TRAIN_RATIO, random_state=RANDOM_
print(f"Train: {len(y_train)}  Test: {len(y_test)}")
print("Per-class counts (overall):", per_class_counts)


model = build_simple_cnn(input_shape=(FRAME_SIZE[1], FRAME_SIZE[0], 1), num_classes=10)
model.summary()


train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)).shuffle(1000).batch(BATCH_SIZE).pref
val_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(BATCH_SIZE).prefetch(tf.data.AUTOT
history = model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS)

model_path = os.path.join(OUTPUT_DIR, "digit_model.h5")
model.save(model_path)
print("Model saved to", model_path)


y_pred_probs = model.predict(X_test)
y_pred = np.argmax(y_pred_probs, axis=1)
labels_list = list(range(10))
excel_path = os.path.join(OUTPUT_DIR, "results_metrics_confusion.xlsx")
plots_dir = os.path.join(OUTPUT_DIR, "plots")
metrics_df, cm, overall_acc = compute_and_save_metrics(y_test, y_pred, labels_list, excel_path, plots_
print("Overall accuracy (system) = {:.4f}".format(overall_acc))


preds_df = pd.DataFrame({
    'filename': filenames[-len(y_test):],
    'true': list(y_test),
    'pred': list(y_pred)
})
preds_df.to_excel(os.path.join(OUTPUT_DIR, "predictions_per_sample.xlsx"), index=False)
print("Saved per-sample predictions.")

plt.figure()
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history.get('val_loss', []), label='val_loss')
plt.legend()
plt.title('Training Loss')
plt.savefig(os.path.join(plots_dir, "training_loss.png"))
plt.close()

plt.figure()
```

```
    plt.plot(history.history.get('accuracy', []), label='acc')
    plt.plot(history.history.get('val_accuracy', []), label='val_acc')
    plt.legend()
    plt.title('Training Accuracy')
    plt.savefig(os.path.join(plots_dir, "training_accuracy.png"))
    plt.close()
if __name__ == "__main__":
    main()
```

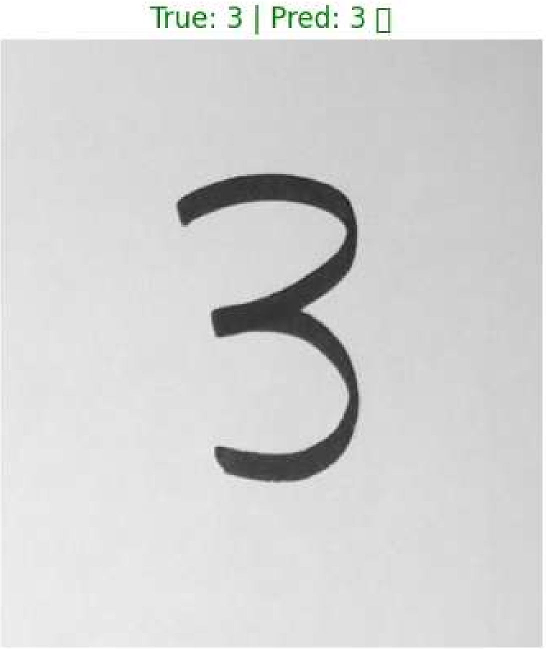
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/driv
Detected digit subfolders; using subfolder names as labels.
```

```
import os
import cv2
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path


output_folder = "/content/drive/MyDrive/smart_digit_outputs_folder"
dataset_folder = "/content/drive/MyDrive/dataset"  # Change if needed
pred_path = os.path.join(output_folder, "predictions_per_sample.xlsx")


pred_df = pd.read_excel(pred_path)
print(f"✅ Loaded {len(pred_df)} predictions from Excel")


all_images = {}
for root, _, files in os.walk(dataset_folder):
    for file in files:
        if file.lower().endswith(('.png', '.jpg', '.jpeg')):
            all_images[file] = os.path.join(root, file)

num_to_show = 10
shown = 0

for i, row in pred_df.iterrows():
    file = str(row["filename"])
    fname = Path(file).name
    img_path = all_images.get(fname, None)

    if img_path is None or not os.path.exists(img_path):
        print(f"⚠️ Image not found for {fname}")
        continue

    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print(f"⚠️ Could not read {fname}")
        continue

    pred = row["pred"]
    true = row["true"]
    correct = (str(pred) == str(true))

    print(f"[{i+1}] File: {fname}")
    print(f"   ➤ True: {true}, Predicted: {pred} {'✅' if correct else '❌'}")

    plt.imshow(img, cmap='gray')
    plt.title(f"True: {true} | Pred: {pred} {'✅' if correct else '❌'}",
              color=("green" if correct else "red"))
    plt.axis("off")
    plt.show()

    shown += 1
    if shown >= num_to_show:
        break
```

```
✅ Loaded 34 predictions from Excel
[1] File: Copy of Screenshot 2025-10-06 at 9.29.12 PM.png
   ➤ True: 9, Predicted: 8 ❌
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 10060 (\N{CRO
  fig.canvas.print_figure(bytes_io, **kw)
```



```
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 9989 (\N{WHIT
  fig.canvas.print_figure(bytes_io, **kw)
[2] File: Copy of IMG_20251007_063831127~5.jpg
   ➤ True: 3, Predicted: 3 ✅
```



```
[3] File: Copy of IMG_20251007_092214759.jpg
   ➤ True: 2, Predicted: 3 ❌
```