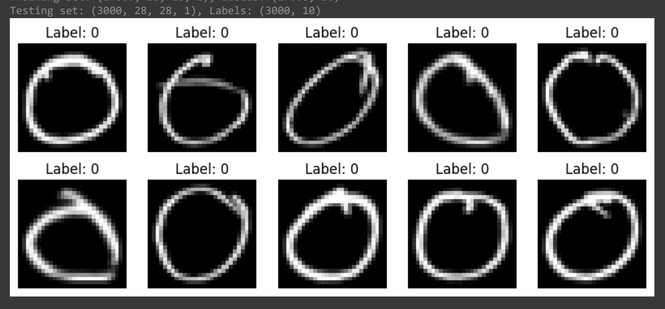
```
!unzip "/content/Copy of devnagari digit.zip"
```

```
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22877.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22878.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22879.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22880.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22881.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22882.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22883.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22884.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22885.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22886.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22887.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22888.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22889.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/22890.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30200.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30201.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30202.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30203.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30204.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30205.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30206.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30207.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30208.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30209.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30210.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30211.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30212.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30213.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30214.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30215.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30216.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30217.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30218.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30219.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30220.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30221.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30222.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30223.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30224.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30225.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30226.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30227.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30228.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30229.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30231.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30232.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30233.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30234.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30236.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30237.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30238.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30239.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30240.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30241.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30242.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30243.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30244.png
  inflating: DevanagariHandwrittenDigitDataset/Train/digit_7/30246.png
```

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from PIL import Image # Import Pillow
# Define dataset paths
train_dir = "/content/DevanagariHandwrittenDigitDataset/Train"
test_dir = "/content/DevanagariHandwrittenDigitDataset/Test"
# Define image size
img_height, img_width = 28, 28
# Function to load images and labels using PIL
def load_images_from_folder(folder):
    images = []
    labels = []
    class_names = sorted(os.listdir(folder))  # Sorted class names (digit_0, digit_1, ...)
    class_map = {name: i for i, name in enumerate(class_names)}  # Map class names to labels

    for class_name in class_names:
        class_path = os.path.join(folder, class_name)
        label = class_map[class_name]

        for filename in os.listdir(class_path):
            img_path = os.path.join(class_path, filename)
            img = Image.open(img_path).convert("L")  # Convert to grayscale
            img = img.resize((img_width, img_height))  # Resize to (28,28)
            img = np.array(img) / 255.0  # Normalize pixel values to [0,1]
            images.append(img)
            labels.append(label)

    return np.array(images), np.array(labels)

# Load training and testing datasets
x_train, y_train = load_images_from_folder(train_dir)
x_test, y_test = load_images_from_folder(test_dir)
# Reshape images for Keras input
x_train = x_train.reshape(-1, img_height, img_width, 1) # Shape (num_samples, 28, 28, 1)
x_test = x_test.reshape(-1, img_height, img_width, 1)
# One-hot encode labels
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
# Print dataset shape
print(f"Training set: {x_train.shape}, Labels: {y_train.shape}")
print(f"Testing set: {x_test.shape}, Labels: {y_test.shape}")
# Visualize some images
plt.figure(figsize=(10, 4))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_train[i].reshape(28, 28), cmap='gray')
    plt.title(f"Label: {np.argmax(y_train[i])}")
    plt.axis("off")
plt.show()
```

```
Training set: (17000, 28, 28, 1), Labels: (17000, 10)
Testing set: (3000, 28, 28, 1), Labels: (3000, 10)
```



Task 1 (Data Preparation)

```
import zipfile
import os

# Define the path to the uploaded zip file and extraction directory
zip_path = '//content/Copy of devnagari digit.zip'
extraction_dir = '/content/DevanagariHandwrittenDigitDataset'

# Unzip the dataset
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extraction_dir)

# List extracted folders to verify
extracted_folders = os.listdir(extraction_dir)
extracted_folders
```

```
['Train', 'Test', 'DevanagariHandwrittenDigitDataset']
```

Checking the contents inside folder

```
# Check the contents inside 'DevanagariHandwrittenDigitDataset' to find train/test folders
dataset_main_dir = os.path.join(extraction_dir, 'DevanagariHandwrittenDigitDataset')
dataset_contents = os.listdir(dataset_main_dir)
dataset_contents
```

```
['Train', 'Test']
```

Data Preparation of 'Train'

```
# List class folders inside the Train directory
train_dir = os.path.join(dataset_main_dir, 'Train')
class_folders = os.listdir(train_dir)
class_folders_sorted = sorted(class_folders)  # Ensure label order is consistent
class_folders_sorted
```

```
['digit_0',
 'digit_1',
 'digit_2',
 'digit_3',
 'digit_4',
 'digit_5',
 'digit_6',
 'digit_7',
 'digit_8',
 'digit_9']
```

Data Preparation of 'Test'

```
# List class folders inside the Train directory
train_dir = os.path.join(dataset_main_dir, 'Test')
class_folders = os.listdir(train_dir)
class_folders_sorted = sorted(class_folders)  # Ensure label order is consistent
class_folders_sorted
```

```
['digit_0',
 'digit_1',
 'digit_2',
 'digit_3',
 'digit_4',
 'digit_5',
 'digit_6',
 'digit_7',
 'digit_8',
 'digit_9']
```

```
class_folders = sorted(os.listdir(train_dir))
```

```
def load_images(base_dir, class_folders, image_size=(28, 28)):
    images = []
    labels = []
    for label_index, folder in enumerate(class_folders):
        folder_path = os.path.join(base_dir, folder)
        for filename in os.listdir(folder_path):
            if filename.endswith('.png'):
                img_path = os.path.join(folder_path, filename)
                img = Image.open(img_path).convert('L')  # Grayscale
                img = img.resize(image_size)
                img_array = np.array(img) / 255.0  # Normalize
                images.append(img_array)
                labels.append(label_index)
    return np.array(images), np.array(labels)
```

```
# Load data
X_train, y_train = load_images(train_dir, class_folders)
X_test, y_test = load_images(test_dir, class_folders)
```

```
X_train = X_train.reshape(-1, 28*28)
X_test = X_test.reshape(-1, 28*28)
```

```
y_train_ohe = to_categorical(y_train, num_classes=10)
y_test_ohe = to_categorical(y_test, num_classes=10)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout

# --- Build FCN Model with Input Layer ---
model = Sequential([
    Input(shape=(784,)),  # Explicit input layer
    Dense(512, activation='relu'),
    Dropout(0.2),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')  # Output layer for 10 classes
])
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
model.fit(X_train, y_train_ohe, epochs=10, batch_size=64, validation_split=0.1)
```

```
Epoch 1/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - accuracy: 0.6067 - loss: 1.2813 - val_accuracy: 0.0000e+00 - val_loss: 11.5484
Epoch 2/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9089 - loss: 0.2853 - val_accuracy: 0.0000e+00 - val_loss: 10.8329
Epoch 3/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9337 - loss: 0.2177 - val_accuracy: 0.0000e+00 - val_loss: 10.7518
Epoch 4/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9658 - loss: 0.1067 - val_accuracy: 0.0000e+00 - val_loss: 10.8907
Epoch 5/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9761 - loss: 0.0869 - val_accuracy: 0.0000e+00 - val_loss: 11.8627
Epoch 6/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9849 - loss: 0.0469 - val_accuracy: 0.0000e+00 - val_loss: 12.0082
Epoch 7/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9868 - loss: 0.0459 - val_accuracy: 0.0000e+00 - val_loss: 12.0336
Epoch 8/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9940 - loss: 0.0251 - val_accuracy: 0.0000e+00 - val_loss: 12.5872
Epoch 9/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9907 - loss: 0.0481 - val_accuracy: 0.0000e+00 - val_loss: 13.1477
Epoch 10/10
43/43 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9922 - loss: 0.0261 - val_accuracy: 0.0000e+00 - val_loss: 12.0445
<keras.src.callbacks.history.History at 0x7ee51f9a0a50>
```

```
test_loss, test_accuracy = model.evaluate(X_test, y_test_ohe)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9894 - loss: 0.0952
Test Accuracy: 89.67%
```

Task 2: Build the FCN Model (as per specs)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense

# --- Build FCN with specified architecture ---
model = Sequential([
    Input(shape=(784,)),  # 28x28 images flattened
    Dense(64, activation='sigmoid'),
    Dense(128, activation='sigmoid'),
    Dense(256, activation='sigmoid'),
    Dense(10, activation='softmax')  # Output layer for 10 classes
])

print("Model built successfully.")
model.summary()
```

```
Model built successfully.
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 64)	50,240
dense_5 (Dense)	(None, 128)	8,320
dense_6 (Dense)	(None, 256)	33,024
dense_7 (Dense)	(None, 10)	2,570

```
Total params: 94,154 (367.79 KB)
Trainable params: 94,154 (367.79 KB)
Non-trainable params: 0 (0.00 B)
```

Task 3: Compile the Model

```
# --- Compile model ---
model.compile(optimizer='adam',
              loss='categorical_crossentropy',  # Since labels are one-hot encoded
              metrics=['accuracy'])

print("Model compiled.")
```

```
Model compiled.
```

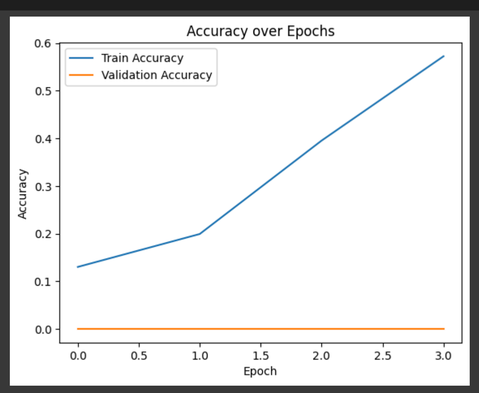
Task 4: Train the Model

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt

# --- Callbacks ---
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
checkpoint = ModelCheckpoint('best_devnagari_model.h5', save_best_only=True, monitor='val_loss')

# --- Train the model ---
history = model.fit(X_train, y_train_ohe,
                    epochs=20,
                    batch_size=128,
                    validation_split=0.2,
                    callbacks=[early_stop, checkpoint])

print("Training complete.")
```

```
Epoch 1/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.1239 - loss: 2.2922WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
19/19 ━━━━━━━━━━━━━━━━━━━━ 5s 144ms/step - accuracy: 0.1242 - loss: 2.2862 - val_accuracy: 0.0000e+00 - val_loss: 5.8536
Epoch 2/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.1576 - loss: 2.0367 - val_accuracy: 0.0000e+00 - val_loss: 6.6991
Epoch 3/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.3244 - loss: 1.8903 - val_accuracy: 0.0000e+00 - val_loss: 6.8387
Epoch 4/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5686 - loss: 1.5598 - val_accuracy: 0.0000e+00 - val_loss: 6.8928
Training complete.
```

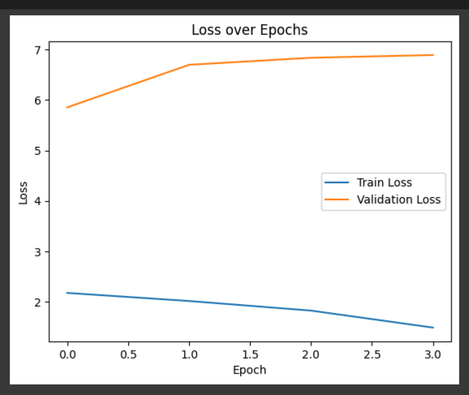
Plot accuracy

```
# --- Plot accuracy ---
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Plot Loss

```
# --- Plot loss ---
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Task 5: Evaluate the Model on Test Data

```
# --- Evaluate on test set ---
test_loss, test_acc = model.evaluate(X_test, y_test_ohe)
```

```
print(f"Test Accuracy: {test_acc * 100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.0444 - loss: 2.2066
Test Accuracy: 10.00%
Test Loss: 2.8160
```

Task 6: Save and Load the Model

Save the Model

```
# --- Save model ---
model.save('devnagari_digit_model.keras')
print("Model saved as 'devnagari_digit_model.keras'")
```

```
Model saved as 'devnagari_digit_model.keras'
```

Load the Model and Re-evaluate

```
from tensorflow.keras.models import load_model

# --- Load model ---
loaded_model = load_model('devnagari_digit_model.keras')
print("Model loaded.")

# --- Recompile model ---
loaded_model.compile(optimizer='adam',
                     loss='categorical_crossentropy',
                     metrics=['accuracy'])

# --- Re-evaluate ---
loss, accuracy = loaded_model.evaluate(X_test, y_test_ohe)
print(f"Loaded Model Test Accuracy: {accuracy * 100:.2f}%")
print(X_test.shape)  # Should be (num_samples, 784)
print(X_test.min(), X_test.max())  # Should be 0.0 and 1.0
print(y_test_ohe.shape)  # Should be (num_samples, 10) for one-hot
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/saving/saving_lib.py:757: UserWarning: Skipping variable loading for optimizer 'rmsprop', because it has 10 variables whereas the saved optimizer has 18 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
Model loaded.
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.0444 - loss: 2.2066
Loaded Model Test Accuracy: 10.00%
(3000, 784)
0.0 1.0
(3000, 10)
```

Task 7: Make Predictions

```
import numpy as np

# --- Predict on test images ---
predictions = loaded_model.predict(X_test)

# --- Convert probabilities to labels ---
predicted_labels = np.argmax(predictions, axis=1)

# --- Display first 10 predictions and true labels ---
print("Predicted labels:", predicted_labels[:10])
print("True labels     :", np.argmax(y_test_ohe[:10], axis=1))
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
Predicted labels: [6 6 6 6 6 6 6 6 6 6]
True labels     : [0 0 0 0 0 0 0 0 0 0]
```