```
from google.colab import drive
drive.mount('/content/drive')
```

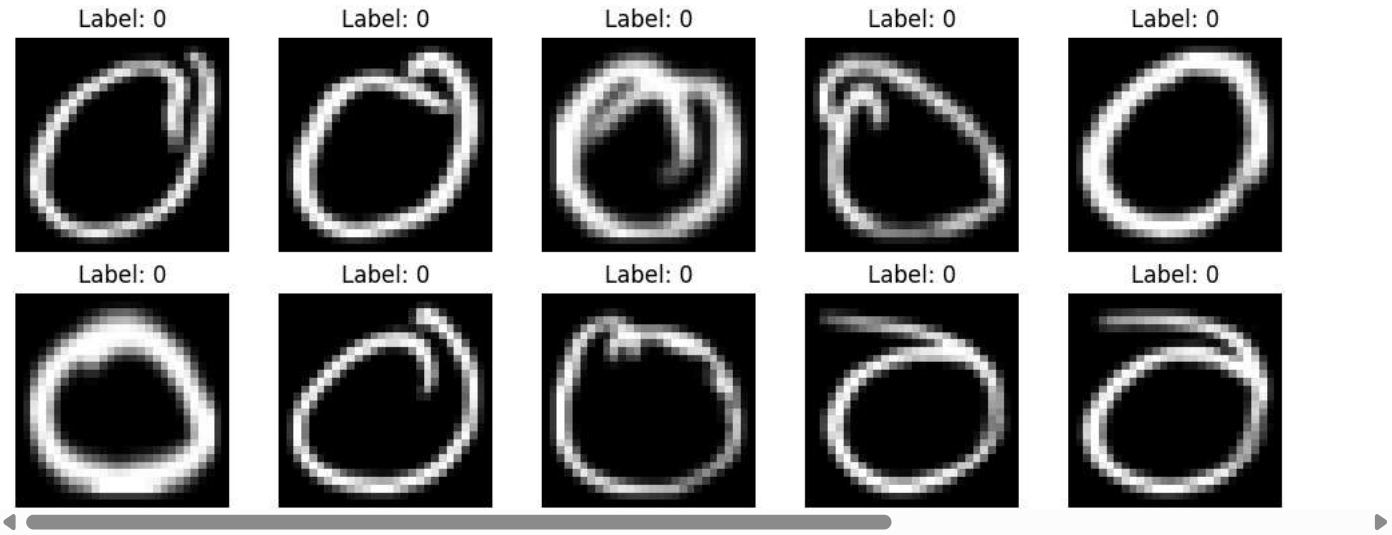
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Task1 Data Preparation

```
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

# Define dataset paths
train_dir = "/content/drive/MyDrive/AI_ML/week_4/DevanagariHandwrittenDigitDataset/Train"
test_dir = "/content/drive/MyDrive/AI_ML/week_4/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 = []

    # Check if directory exists
    if not os.path.exists(folder):
        print(f"Error: Directory '{folder}' not found!")
        return np.array([]), np.array([])

    class_names = sorted(os.listdir(folder))  # Sorted class names
    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)

            # Load image using PIL
            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)

# Ensure data is loaded correctly
if x_train.size == 0 or x_test.size == 0:
    raise ValueError("Dataset loading failed. Check dataset structure.")

# 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")  # Fixed quotes
    plt.title(f"Label: {np.argmax(y_train[i])}")
    plt.axis("off")

plt.tight_layout()
plt.show()
```

```
Training set: (16540, 28, 28, 1), Labels: (16540, 10)
Testing set: (16540, 28, 28, 1), Labels: (16540, 10)
```



Task 2 Build the FCN Model

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

# Define the model
model = Sequential([
    Flatten(input_shape=(28, 28, 1)),  # Flatten the 28x28 image into a 784-dimensional vector
    Dense(64, activation="sigmoid"),   # 1st hidden layer
    Dense(128, activation="sigmoid"),  # 2nd hidden layer
    Dense(256, activation="sigmoid"),  # 3rd hidden layer
    Dense(10, activation="softmax")    # Output layer (10 classes)
])

# Display model architecture
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
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)

Task 3 Compile the Model

```
# Compile the model
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
```

```
    metrics=["accuracy"]
)
```

Task 4 Train the Model

```
# Set training parameters
batch_size = 128
epochs = 500

# Define callbacks
callbacks = [
    tf.keras.callbacks.ModelCheckpoint(filepath="best_model.keras", save_best_only=True),
    tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience=4, restore_best_weights=True)
]

# Train the model
history = model.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2,
    callbacks=callbacks
)
```

```
Epoch 1/500
104/104 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.2814 - loss: 2.0033 - val_accuracy: 0.0000e+00 - val_loss: 5.7632
Epoch 2/500
104/104 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8119 - loss: 0.6613 - val_accuracy: 0.0000e+00 - val_loss: 5.8815
Epoch 3/500
104/104 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9157 - loss: 0.3110 - val_accuracy: 0.0000e+00 - val_loss: 6.0728
Epoch 4/500
104/104 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9424 - loss: 0.2089 - val_accuracy: 0.0088 - val_loss: 5.7910
Epoch 5/500
104/104 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9505 - loss: 0.1745 - val_accuracy: 0.0054 - val_loss: 6.0877
```

Task 5 Evaluate the Model

```
# Evaluate the model on test data
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_acc:.4f}")
```

```
517/517 - 1s - 3ms/step - accuracy: 0.6071 - loss: 1.8398
Test accuracy: 0.6071
```

Task 6 Save and Load the Model

```
# Import TensorFlow
import tensorflow as tf

# Save the trained model in the native Keras format (.keras)
model.save("devnagari_fcn_model.keras")

# Load the saved model
loaded_model = tf.keras.models.load_model("devnagari_fcn_model.keras")

# Re-evaluate the loaded model on the test set
loaded_test_loss, loaded_test_acc = loaded_model.evaluate(x_test, y_test, verbose=2)
print(f"Loaded model test accuracy: {loaded_test_acc:.4f}")
```

```
517/517 - 2s - 3ms/step - accuracy: 0.6071 - loss: 1.8398
Loaded model test accuracy: 0.6071
```

Task 7: Making Predictions

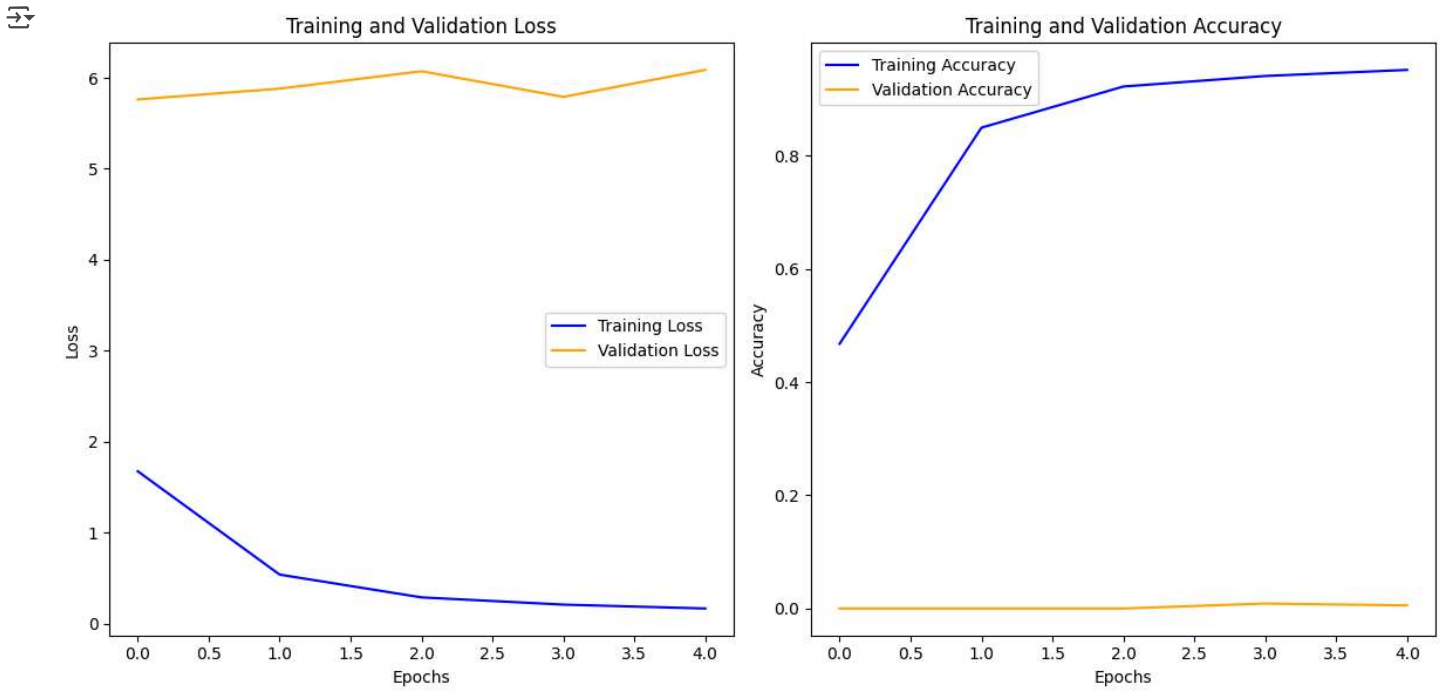
```
# Make predictions on test images
predictions = model.predict(x_test)

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

```
# Print first prediction
print(f"Predicted label for first image: {predicted_labels[0]}")
print(f"True label for first image: {np.argmax(y_test[0])}")
```

```
517/517 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Predicted label for first image: 0
True label for first image: 0
```

Visualizing

```
# Extracting training and validation loss and accuracy
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# Plotting training and validation loss and accuracy
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(train_loss, label="Training Loss", color="blue")
plt.plot(val_loss, label="Validation Loss", color="orange")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_acc, label="Training Accuracy", color="blue")
plt.plot(val_acc, label="Validation Accuracy", color="orange")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()

plt.tight_layout()
plt.show()
```




```
model.save('/content/drive/MyDrive/devnagari_fcn_model.h5')
loaded_model = tf.keras.models.load_model('/content/drive/MyDrive/devnagari_fcn_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you t
```