```
import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
from PIL import Image
import matplotlib.pyplot as plt
import zipfile
```

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

```
Mounted at /content/drive
```

Task 1: Data Preparation Key Considerations

Use PIL (Python Imaging Library) for image processing Normalize images to 0-1 range Resize to consistent 28x28 pixels Extract labels from directory structure

```
# Unzip the dataset if needed
zip_path = '/content/drive/MyDrive/Ai and ML/Copy of Copy of devnagari digit.zip'
extract_path = '/content/drive/MyDrive/Level 6/AI & ML/w4/'

# Unzip the file if not already extracted
try:
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)
except FileExistsError:
    print("Dataset already extracted.")

# Set paths
train_data_dir = '/content/drive/MyDrive/Level 6/AI & ML/w4/DevanagariHandwrittenDigitDataset/Train'
test_data_dir = '/content/drive/MyDrive/Level 6/AI & ML/w4/DevanagariHandwrittenDigitDataset/Test'

# Data Preparation Function
def load_data(data_dir):
    """
    Load Devnagari digit images and labels from the specified directory

    Args:
    data_dir (str): Path to the directory containing digit images

    Returns:
    tuple: (images, labels)
    """
    images = []
    labels = []

    # Iterate through subdirectories (each representing a digit)
    for label, digit_dir in enumerate(sorted(os.listdir(data_dir))):
        digit_path = os.path.join(data_dir, digit_dir)

        # Ensure it's a directory
        if os.path.isdir(digit_path):
            for img_file in os.listdir(digit_path):
                img_path = os.path.join(digit_path, img_file)

                # Open and process image
                try:
                    img = Image.open(img_path).convert('L')  # Convert to grayscale
                    img = img.resize((28, 28))  # Resize to 28x28
                    img_array = np.array(img) / 255.0  # Normalize to 0-1

                    images.append(img_array)
                    labels.append(label)
                except Exception as e:
                    print(f"Error processing {img_path}: {e}")

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

```
# Load training and testing data
print("Loading training data...")
X_train, y_train = load_data(train_data_dir)
print(f"Training data shape: {X_train.shape}")
```

```
print("Loading testing data...")
X_test, y_test = load_data(test_data_dir)
print(f"Testing data shape: {X_test.shape}")

# Reshape images for FCN (flatten)
X_train = X_train.reshape(X_train.shape[0], 28*28)
X_test = X_test.reshape(X_test.shape[0], 28*28)

# Convert labels to one-hot encoding
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

```
Loading training data...
Training data shape: (17000, 28, 28)
Loading testing data...
Testing data shape: (3000, 28, 28)
```

Task 2: Build FCN Model

3 Hidden Layers

1. 1st Layer: 64 neurons
2. 2nd Layer: 128 neurons
3. 3rd Layer: 256 neurons

- Sigmoid activation for hidden layers
- Softmax activation for output layer

```
# Build the FCN Model
model = keras.Sequential([
    # Input layer
    keras.layers.InputLayer(input_shape=(28*28,)),

    # First hidden layer
    keras.layers.Dense(64, activation='sigmoid'),

    # Second hidden layer
    keras.layers.Dense(128, activation='sigmoid'),

    # Third hidden layer
    keras.layers.Dense(256, activation='sigmoid'),

    # Output layer
    keras.layers.Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/input_layer.py:27: UserWarning: Argument `input_shape` is
  warnings.warn(
```

summery of the model

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	50,240
dense_1 (Dense)	(None, 128)	8,320
dense_2 (Dense)	(None, 256)	33,024
dense_3 (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: Model Compilation

Configuration

Optimizer: Adam Loss Function: Categorical Crossentropy Metric: Accuracy

```
# Compile the Model
model.compile(
```

```
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Task 4: Model Training

Training Parameters

Batch Size: 128 Epochs: 20 Validation Split: 0.2

Callbacks for Optimization

```
# Prepare Callbacks
checkpoint_callback = keras.callbacks.ModelCheckpoint(
    '/content/drive/MyDrive/Level 6/AI & ML/w4/best_model.h5',
    save_best_only=True,
    monitor='val_accuracy'
)

early_stopping_callback = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

Train the Model

```
# Train the Model
history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    batch_size=128,
    epochs=20,
    callbacks=[checkpoint_callback, early_stopping_callback]
)
```

```
Epoch 1/20
107/107 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.2640 - loss: 2.0099WARNING:absl:You are saving your model as an
107/107 ━━━━━━━━━━━━━━━━━━━━ 6s 26ms/step - accuracy: 0.2656 - loss: 2.0067 - val_accuracy: 0.0000e+00 - val_loss: 7.778
Epoch 2/20
107/107 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7912 - loss: 0.7107 - val_accuracy: 0.0000e+00 - val_loss: 9.2926
Epoch 3/20
107/107 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8845 - loss: 0.3419 - val_accuracy: 0.0000e+00 - val_loss: 10.040
Epoch 4/20
107/107 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9346 - loss: 0.2251 - val_accuracy: 0.0000e+00 - val_loss: 10.398
Epoch 5/20
107/107 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9553 - loss: 0.1620 - val_accuracy: 0.0000e+00 - val_loss: 10.834
Epoch 6/20
107/107 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9614 - loss: 0.1299 - val_accuracy: 0.0000e+00 - val_loss: 11.164
```

Task 5: Model Evaluation

Use model.evaluate() on test set Print test accuracy Analyze performance metrics

```
# Evaluate the Model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8074 - loss: 0.9537
Test Accuracy: 60.33%
```

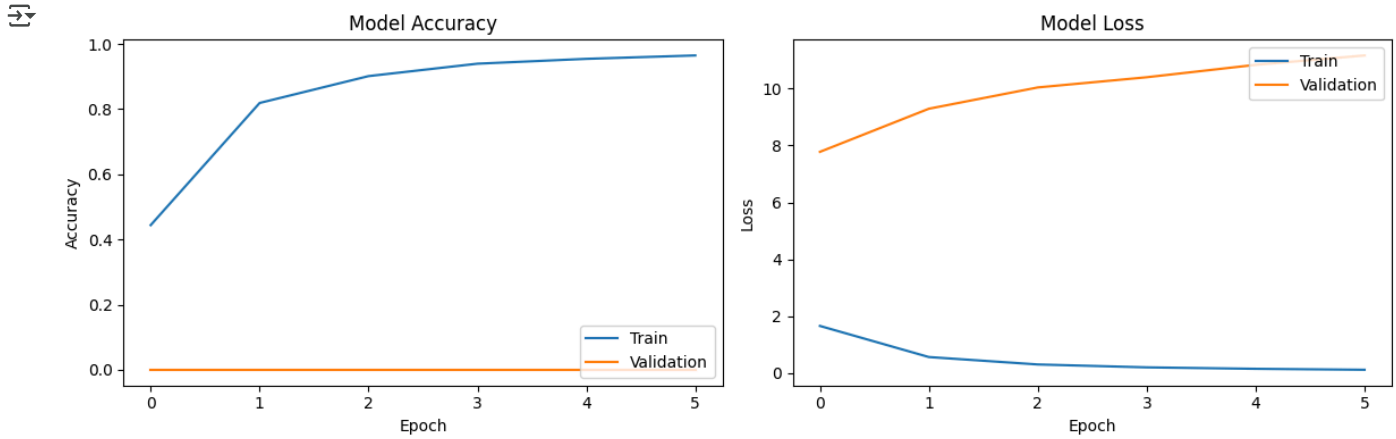
```
# Visualize Training History
plt.figure(figsize=(12, 4))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')

# Loss plot
```

```
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')

plt.tight_layout()
plt.show()
plt.savefig('/content/drive/MyDrive/Level 6/AI & ML/w4/training_history.png')
plt.close()
```



Task 6: Save and Load Model

- we can use model.save() to save model
- but her Model already saved via ModelCheckpoint callback

```
# Save model
# model.save('/content/drive/MyDrive/Level 6/AI & ML/w4/final_model.h5')
```

Task 7: Predictions

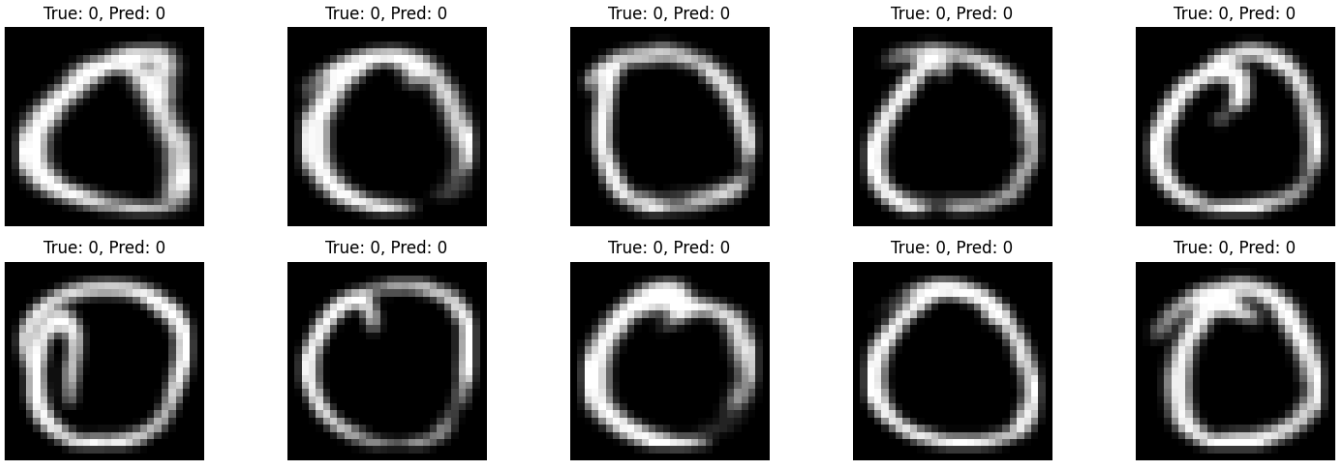
```
# Load the Best Model
loaded_model = keras.models.load_model('/content/drive/MyDrive/Level 6/AI & ML/w4/best_model.h5') # Model already saved via

# Make Predictions
predictions = loaded_model.predict(X_test)
predicted_labels = np.argmax(predictions, axis=1)
true_labels = np.argmax(y_test, axis=1)

# Visualize Some Predictions
plt.figure(figsize=(15, 5))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f'True: {np.argmax(y_test[i])}, Pred: {predicted_labels[i]}')
    plt.axis('off')
plt.tight_layout()
plt.show()
plt.savefig('/content/drive/MyDrive/Level 6/AI & ML/w4/predictions.png')
plt.close()

print("Training and evaluation complete!")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be e
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
```




```
Training and evaluation complete!
```

```
from sklearn.metrics import confusion_matrix, classification_report
cm = confusion_matrix(true_labels, predicted_labels)
print("\nConfusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[293   1   0   0   0   0   6   0   0]
 [  1 294   1   0   2   0   2   0   0]
 [  0   4 276   0   1   8   9   2   0   0]
 [  0   1 270   0   1  19   6   3   0   0]
 [  0   3   0   0 277   7   8   5   0   0]
 [  0   0  51   0  22 222   5   0   0   0]
 [  1   7  15   0   4   2 238  33   0   0]
 [ 77   0   1   0   2   5   5 210   0   0]
 [140   9  33   0  14   3  53  48   0   0]
 [ 87  91   5   0  36   6  25  50   0   0]]
```