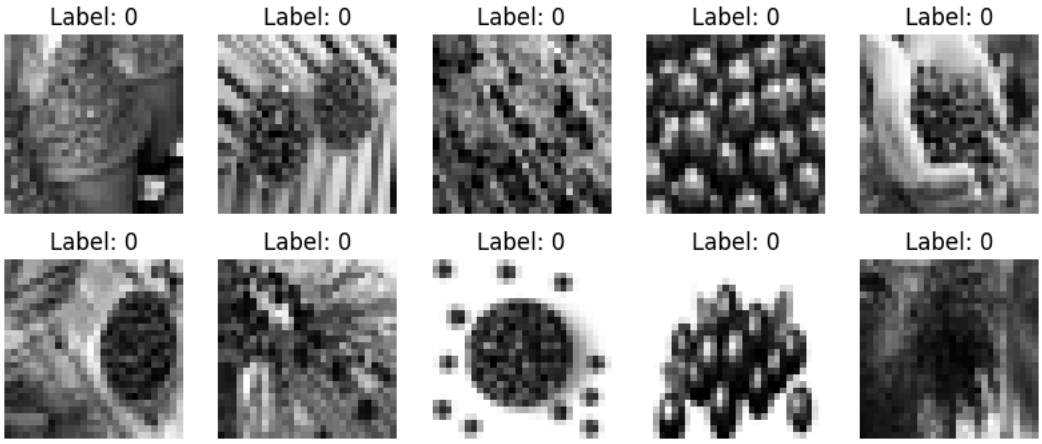
Task 1: Data Preparation

```
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

# Training and testing directory
train_dir = "/content/drive/MyDrive/AI and Machine Learning/train"
test_dir = "/content/drive/MyDrive/AI and Machine Learning/test"

# Defining the 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([name for name in os.listdir(folder) if os.path.isdir(os.path.join(folder, name))])
    print(f"Class names: {class_names}")
    class_map = { name: i for i, name in enumerate(class_names) }
    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)
            try:
                img = Image.open(img_path).convert("L")
                img = img.resize((img_width, img_height))
                img = np.array(img) / 255.0
                if img.shape != (img_height, img_width):
                    print(f"Skipping image {img_path}: incorrect shape {img.shape}")
                    continue
                images.append(img)
                labels.append(label)
            except Exception as e:
                print(f"Error loading image {img_path}: {e}")
                continue
    images = np.array(images, dtype=np.float32)
    labels = np.array(labels, dtype=np.int32)
    print(f"Loaded {len(images)} images with shape {images.shape}, labels shape {labels.shape}")
    return images, 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)
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()
```

```
Class names: ['acai', 'cupuacu', 'graviola', 'guarana', 'pupunha', 'tucuma']
Loaded 90 images with shape (90, 28, 28), labels shape (90,)
Class names: ['acai', 'cupuacu', 'graviola', 'guarana', 'pupunha', 'tucuma']
Loaded 30 images with shape (30, 28, 28), labels shape (30,)
Training set: (90, 28, 28, 1), Labels: (90, 10)
Testing set: (30, 28, 28, 1), Labels: (30, 10)
```




```
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).
```

Task 2: Build the FCN Model

```
import tensorflow as tf
from tensorflow import keras

num_classes = 10
input_shape = (28, 28, 1)
model = keras.Sequential([
    keras.layers.Input(shape=input_shape),
    keras.layers.Flatten(),
    keras.layers.Dense(64, activation="relu"),  # Changed to relu for better performance
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(256, activation="relu"),
    keras.layers.Dense(num_classes, activation="softmax"),
])
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
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: Compile the Model

Compiling the Model

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

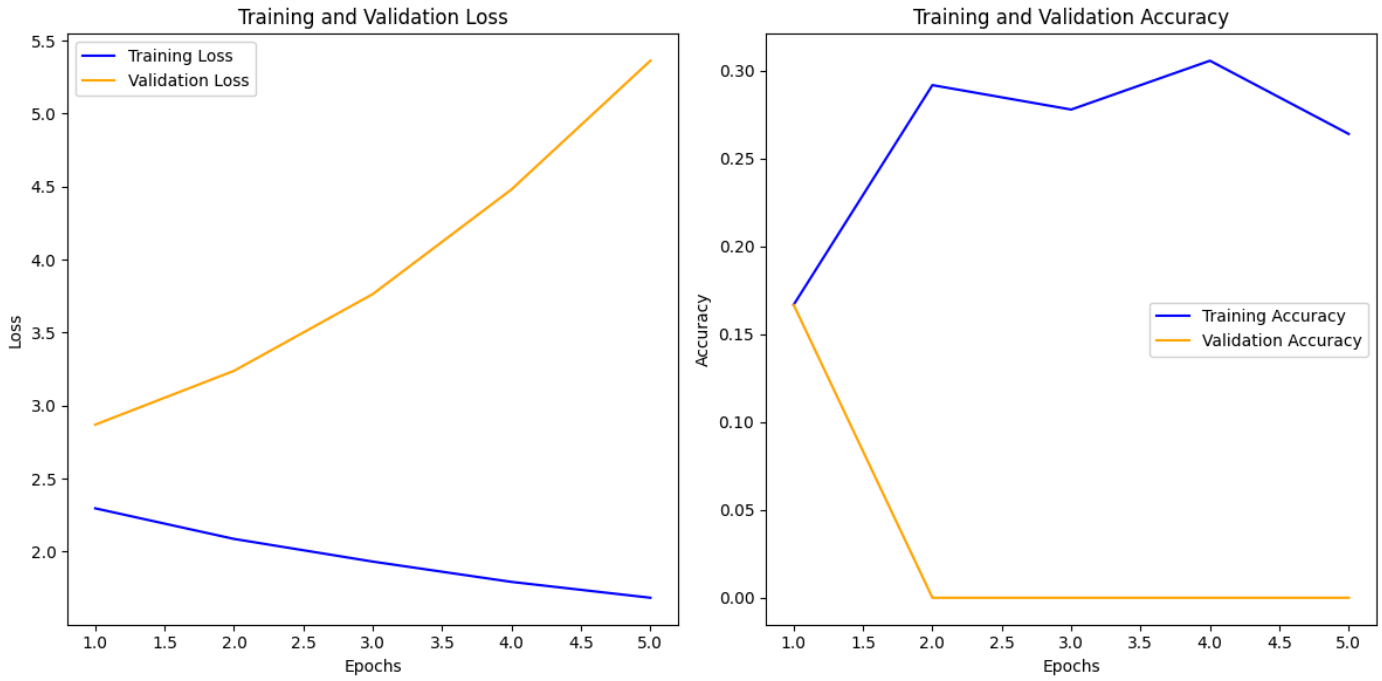
Task 4: Train the Model

```
# Debugging checks
# print(f"x_train type: {type(x_train)}, shape: {x_train.shape}, dtype: {x_train.dtype}")
# print(f"y_train type: {type(y_train)}, shape: {y_train.shape}, dtype: {y_train.dtype}")

# if not isinstance(x_train, np.ndarray) or not isinstance(y_train, np.ndarray):
#     raise ValueError("x_train or y_train is not a NumPy array")
# if x_train.size == 0 or y_train.size == 0:
#     raise ValueError("x_train or y_train is empty")
# if x_train.shape[0] != y_train.shape[0]:
#     raise ValueError(f"Number of samples mismatch: x_train has {x_train.shape[0]} samples, y_train has {y_train.shape[0]} samples")

batch_size = 128
epochs = 20

callbacks = [
    keras.callbacks.ModelCheckpoint(filepath="model_at_epoch_{epoch}.keras"),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=4),
]

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

```
Epoch 1/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.1667 - loss: 2.2968 - val_accuracy: 0.1667 - val_loss: 2.8700
Epoch 2/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step - accuracy: 0.2917 - loss: 2.0874 - val_accuracy: 0.0000e+00 - val_loss: 3.2384
Epoch 3/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 131ms/step - accuracy: 0.2778 - loss: 1.9322 - val_accuracy: 0.0000e+00 - val_loss: 3.7641
Epoch 4/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 131ms/step - accuracy: 0.3056 - loss: 1.7931 - val_accuracy: 0.0000e+00 - val_loss: 4.4810
Epoch 5/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - accuracy: 0.2639 - loss: 1.6846 - val_accuracy: 0.0000e+00 - val_loss: 5.3617
```

```
# Plot training and validation metrics
import matplotlib.pyplot as plt

train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history.get('accuracy', [])
val_acc = history.history.get('val_accuracy', [])

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(range(1, len(train_loss) + 1), train_loss, label="Training Loss", color="blue")
plt.plot(range(1, len(val_loss) + 1), 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)
if train_acc and val_acc:
    plt.plot(range(1, len(train_acc) + 1), train_acc, label="Training Accuracy", color="blue")
    plt.plot(range(1, len(val_acc) + 1), 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()
```



Task 5: Evaluate the Model

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_acc:.4f}")
```

```
1/1 - 0s - 55ms/step - accuracy: 0.1667 - loss: 2.3206
Test accuracy: 0.1667
```

Task 6: Save and Load the Model

1. Saving the Model:

```
model.save("mnist_fully_connected_model.keras")
```

2. Loading the Model:

```
loaded_model = tf.keras.models.load_model("mnist_fully_connected_model.keras")
```

Task 7: Predictions

```
predictions = model.predict(x_test)
predicted_labels = np.argmax(predictions, axis=1)
print(f"Predicted label for first image: {predicted_labels[0]}")
print(f"True label for first image: {np.argmax(y_test[0])}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step
Predicted label for first image: 1
True label for first image: 0
```

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