```python
import tensorflow as tf
import flwr as fl
import numpy as np

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Normalize pixel values to [0, 1]
x_train, x_test = x_train / 255.0, x_test / 255.0

print("Train shape:", x_train.shape)
print("Test shape:", x_test.shape)
```

```
Train shape: (60000, 28, 28)
Test shape: (10000, 28, 28)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.dat
  return datetime.utcnow().replace(tzinfo=utc)
```

```
!pip install flwr
```

```
Requirement already satisfied: flwr in /usr/local/lib/python3.12/dist-packages (1.22.0)
Requirement already satisfied: click<8.2.0 in /usr/local/lib/python3.12/dist-packages (from flwr) (8.1.
Requirement already satisfied: cryptography<45.0.0,>=44.0.1 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: grpcio!=1.65.0,<2.0.0,>=1.62.3 in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: grpcio-health-checking<2.0.0,>=1.62.3 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: iterators<0.0.3,>=0.0.2 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: numpy<3.0.0,>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from fl
Requirement already satisfied: pathspec<0.13.0,>=0.12.1 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: protobuf<5.0.0,>=4.21.6 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: pycryptodome<4.0.0,>=3.18.0 in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: pyyaml<7.0.0,>=6.0.2 in /usr/local/lib/python3.12/dist-packages (from fl
Requirement already satisfied: requests<3.0.0,>=2.31.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: rich<14.0.0,>=13.5.0 in /usr/local/lib/python3.12/dist-packages (from fl
Requirement already satisfied: tomli<3.0.0,>=2.0.1 in /usr/local/lib/python3.12/dist-packages (from flw
Requirement already satisfied: tomli-w<2.0.0,>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from f
Requirement already satisfied: typer<0.13.0,>=0.12.5 in /usr/local/lib/python3.12/dist-packages (from f
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.12/dist-packages (from cryptography
Requirement already satisfied: typing-extensions~=4.12 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requ
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requ
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from r
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from type
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-packages (from cffi>=1.12->c
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-
# Split training data into 3 clients
client_data = []
num_clients = 3
split_size = len(x_train) // num_clients

for i in range(num_clients):
    start = i * split_size
    end = (i + 1) * split_size
    client_data.append((x_train[start:end], y_train[start:end]))

print("Each client has:", len(client_data[0][0]), "training samples")
```

```
Each client has: 20000 training samples
```

```
def create_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Reshape((28, 28, 1), input_shape=(28, 28)),
        tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
        tf.keras.layers.Conv2D(64, kernel_size=(3,3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dense(10, activation="softmax")
    ])
    model.compile(optimizer="adam",
                  loss="sparse_categorical_crossentropy",
                  metrics=["accuracy"])
    return model
```

```
class MnistClient(fl.client.NumPyClient):
    def __init__(self, model, train_data, test_data):
        self.model = model
        self.x_train, self.y_train = train_data
        self.x_test, self.y_test = test_data

    def get_parameters(self, config):
        return self.model.get_weights()

    def fit(self, parameters, config):
        self.model.set_weights(parameters)
        self.model.fit(self.x_train, self.y_train, epochs=1, batch_size=32, verbose=0)
        return self.model.get_weights(), len(self.x_train), {}

    def evaluate(self, parameters, config):
        self.model.set_weights(parameters)
        loss, acc = self.model.evaluate(self.x_test, self.y_test, verbose=0)
        return loss, len(self.x_test), {"accuracy": acc}
```

```
def client_fn(cid: str):
    model = create_model()
    train_data = client_data[int(cid)]
    test_data = (x_test, y_test)
    return MnistClient(model, train_data, test_data)

# Start Federated Learning Simulation
fl.simulation.start_simulation(
    client_fn=client_fn,
    num_clients=num_clients,
    config=fl.server.ServerConfig(num_rounds=3),
)
```

```
(ClientAppActor pid=3885)
(ClientAppActor pid=3885)
(ClientAppActor pid=3885) WARNING :   Deprecation Warning: The `client_fn` function must return an ins
(ClientAppActor pid=3885) WARNING :   DEPRECATED FEATURE: `client_fn` now expects a signature `def cl
(ClientAppActor pid=3885)             This is a deprecated feature. It will be removed
(ClientAppActor pid=3885)             entirely in future versions of Flower.
INFO :      aggregate_evaluate: received 3 results and 0 failures
INFO :
INFO :      [ROUND 3]
INFO :      configure_fit: strategy sampled 3 clients (out of 3)
(ClientAppActor pid=3885) WARNING :   DEPRECATED FEATURE: `client_fn` now expects a signature `def cl
(ClientAppActor pid=3885)
(ClientAppActor pid=3885)             This is a deprecated feature. It will be removed
(ClientAppActor pid=3885)             entirely in future versions of Flower.
(ClientAppActor pid=3885)
(ClientAppActor pid=3890)
(ClientAppActor pid=3890)
(ClientAppActor pid=3890)
(ClientAppActor pid=3890)
(ClientAppActor pid=3890) WARNING :   Deprecation Warning: The `client_fn` function must return an ins
(ClientAppActor pid=3890) WARNING :   DEPRECATED FEATURE: `client_fn` now expects a signature `def cl
(ClientAppActor pid=3890)             This is a deprecated feature. It will be removed [repeated 2x ac
(ClientAppActor pid=3890)             entirely in future versions of Flower. [repeated 2x across clus
INFO :      aggregate_fit: received 3 results and 0 failures
INFO :      configure_evaluate: strategy sampled 3 clients (out of 3)
(ClientAppActor pid=3890)
(ClientAppActor pid=3890)
(ClientAppActor pid=3890) WARNING :   DEPRECATED FEATURE: `client_fn` now expects a signature `def cl
(ClientAppActor pid=3890)             This is a deprecated feature. It will be removed
(ClientAppActor pid=3890)             entirely in future versions of Flower.
(ClientAppActor pid=3885) WARNING :   DEPRECATED FEATURE: `client_fn` now expects a signature `def cl
(ClientAppActor pid=3885)
(ClientAppActor pid=3885)             This is a deprecated feature. It will be removed
(ClientAppActor pid=3885)             entirely in future versions of Flower.
(ClientAppActor pid=3885)
(ClientAppActor pid=3885) WARNING :   Deprecation Warning: The `client_fn` function must return an ins
(ClientAppActor pid=3890)
(ClientAppActor pid=3890)
(ClientAppActor pid=3890) WARNING :   DEPRECATED FEATURE: `client_fn` now expects a signature `def cl
(ClientAppActor pid=3890)             This is a deprecated feature. It will be removed
(ClientAppActor pid=3890)             entirely in future versions of Flower.
(ClientAppActor pid=3890) WARNING :   Deprecation Warning: The `client_fn` function must return an ins
(ClientAppActor pid=3890) WARNING :   Deprecation Warning: The `client_fn` function must return an ins
INFO :      aggregate_evaluate: received 3 results and 0 failures
INFO :
INFO :      [SUMMARY]
INFO :      Run finished 3 round(s) in 179.22s
INFO :          History (loss, distributed):
INFO :                  round 1: 0.10901021957397461
INFO :                  round 2: 0.0466558113694191
INFO :                  round 3: 0.03390353173017502
INFO :
History (loss, distributed):
        round 1: 0.10901021957397461
        round 2: 0.0466558113694191
        round 3: 0.03390353173017502
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.da
  return datetime.utcnow().replace(tzinfo=utc)
```

```
# Evaluate the global model on the test set
global_model = create_model()
```

```
# Load the latest global weights from Flower (server keeps them)
# Since we didn’t persist them automatically, we’ll just train a fresh model normally for now.

global_model.fit(x_train, y_train, epochs=1, validation_data=(x_test, y_test))

# Evaluate on test set
loss, accuracy = global_model.evaluate(x_test, y_test, verbose=0)
print(f"Global Model Accuracy: {accuracy * 100:.2f}%")
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/reshape.py:39: UserWarning: Do not p
  super().__init__(**kwargs)
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step - accuracy: 0.9137 - loss: 0.2862 - val_accuracy: 0.9865 -
Global Model Accuracy: 98.65%
```

```
from keras.models import load_model

# Load the saved model
global_model = load_model("mnist_cnn_model.keras")

# Test a single example
import numpy as np
example = x_test[0].reshape(1, 28, 28, 1)  # reshape for the model
prediction = np.argmax(global_model.predict(example))
print(f"Predicted digit: {prediction}, Actual: {y_test[0]}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 531ms/step
Predicted digit: 7, Actual: 7
```

```
loss, accuracy = global_model.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9830 - loss: 0.0495
Test Accuracy: 98.65%
```

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

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

```
import os
os.environ["KAGGLEHUB_CACHE"] = "/content/drive/MyDrive/kagglehub"
```

```
from google.colab import files
files.upload()  # select kaggle.json
```

Choose Files kaggle.json
kaggle.json(application/json) - 71 bytes, last modified: 9/30/2025 - 100% done

```
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"yaseenjabir2004","key":"a0521279714ef810f64bae33f68c3a11"}'}
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("paultimothymooney/kermany2018")

print("Path to dataset files:", path)
```

```
Using Colab cache for faster access to the 'kermany2018' dataset.
Path to dataset files: /kaggle/input/kermany2018
```

```
import os

root = "/kaggle/input/kermany2018"
for folder in os.listdir(root):
    print(folder)
    full_path = os.path.join(root, folder)
    if os.path.isdir(full_path):
        print("  Subfolders:", os.listdir(full_path))
```

```
OCT2017
  Subfolders: ['val', 'test', 'train']
oct2017
  Subfolders: ['OCT2017 ', '__MACOSX']
```

```
import tensorflow as tf

data_dir = "/kaggle/input/kermany2018/OCT2017 "

img_size = (224, 224)
batch_size = 32

# Training set
train_ds = tf.keras.utils.image_dataset_from_directory(
    directory=f"{data_dir}/train",
    labels="inferred",
    label_mode="categorical",  # one-hot encoding
    image_size=img_size,
    batch_size=batch_size,
    shuffle=True
)

# Validation set
val_ds = tf.keras.utils.image_dataset_from_directory(
    directory=f"{data_dir}/val",
    labels="inferred",
    label_mode="categorical",
    image_size=img_size,
    batch_size=batch_size,
    shuffle=False
)

# Test set
test_ds = tf.keras.utils.image_dataset_from_directory(
    directory=f"{data_dir}/test",
    labels="inferred",
    label_mode="categorical",
    image_size=img_size,
    batch_size=batch_size,
    shuffle=False
)

# Normalize (rescale to 0-1)
normalization_layer = tf.keras.layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds   = val_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds  = test_ds.map(lambda x, y: (normalization_layer(x), y))
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-3028612662.py in <cell line: 0>()
      7 
      8 # Training set
----> 9 train_ds = tf.keras.utils.image_dataset_from_directory(
     10     directory=f"{data_dir}/train",
     11     labels="inferred",

                              5 frames
/usr/lib/python3.12/threading.py in wait(self, timeout)
    353         try:    # restore state no matter what (e.g., KeyboardInterrupt)
    354             if timeout is None:
--> 355                 waiter.acquire()
    356                 gotit = True
    357             else:

KeyboardInterrupt: 
```

```
import tensorflow as tf
from tensorflow.keras import layers, models

data_dir = "/kaggle/input/kermany2018/OCT2017 "
img_size = (224, 224)
batch_size = 32

# Load datasets
train_ds = tf.keras.utils.image_dataset_from_directory(
    directory=f"{data_dir}/train",
    labels="inferred",
    label_mode="categorical",
    image_size=img_size,
    batch_size=batch_size,
    shuffle=True
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    directory=f"{data_dir}/val",
    labels="inferred",
    label_mode="categorical",
    image_size=img_size,
    batch_size=batch_size,
    shuffle=False
)

test_ds = tf.keras.utils.image_dataset_from_directory(
    directory=f"{data_dir}/test",
    labels="inferred",
    label_mode="categorical",
    image_size=img_size,
    batch_size=batch_size,
    shuffle=False
)
```

```
# Normalize
normalization_layer = layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds   = val_ds.map(lambda x, y: (normalization_layer(x), y))
test_ds  = test_ds.map(lambda x, y: (normalization_layer(x), y))
```

```
Found 83484 files belonging to 4 classes.
Found 32 files belonging to 4 classes.
Found 968 files belonging to 4 classes.
```

```
model = models.Sequential([
    layers.Input(shape=(224, 224, 3)),
    layers.Conv2D(32, (3,3), activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(4, activation='softmax')
])
```

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

```
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=10
)
```

```
Epoch 1/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 377s 142ms/step - accuracy: 0.6836 - loss: 0.8484 - val_accuracy: 0.8438
Epoch 2/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 193s 74ms/step - accuracy: 0.8731 - loss: 0.3581 - val_accuracy: 0.8750
Epoch 3/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 191s 73ms/step - accuracy: 0.9377 - loss: 0.1809 - val_accuracy: 0.9375
Epoch 4/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 190s 73ms/step - accuracy: 0.9735 - loss: 0.0830 - val_accuracy: 0.9688
Epoch 5/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 190s 73ms/step - accuracy: 0.9838 - loss: 0.0528 - val_accuracy: 0.9062
Epoch 6/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 194s 74ms/step - accuracy: 0.9866 - loss: 0.0463 - val_accuracy: 0.9062
Epoch 7/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 201s 74ms/step - accuracy: 0.9893 - loss: 0.0353 - val_accuracy: 0.9062
Epoch 8/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 191s 73ms/step - accuracy: 0.9912 - loss: 0.0314 - val_accuracy: 0.9688
Epoch 9/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 191s 73ms/step - accuracy: 0.9915 - loss: 0.0284 - val_accuracy: 0.9062
Epoch 10/10
2609/2609 ━━━━━━━━━━━━━━━━━━━━ 203s 74ms/step - accuracy: 0.9921 - loss: 0.0281 - val_accuracy: 1.0000
```

```
test_loss, test_acc = model.evaluate(test_ds)
print("✅ Test accuracy:", test_acc)
print("Test loss:", test_loss)
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 5s 149ms/step - accuracy: 0.9693 - loss: 0.1646
✅ Test accuracy: 0.9659090638160706
Test loss: 0.17112590372562408
```

```
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np

# Load your saved model
model = load_model("eye_disease_model.keras")  # or your HDF5 file

# Load the new image
img_path = "CNV-1016042-1.jpeg"
img = image.load_img(img_path, target_size=(224, 224))  # match the size used in training
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # add batch dimension
img_array = img_array / 255.0  # normalize same as training

# Get class names from training dataset
# class_names = train_ds.class_names  # This caused the error
# Get class names from the original dataset before normalization (from cell YaRv1o5OM2e6)
# data_dir = "/kaggle/input/kermany2018/OCT2017 " # Need to define data_dir again here
# img_size = (224, 224) # Need to define img_size again here
# batch_size = 32 # Need to define batch_size again here

# original_train_ds = tf.keras.utils.image_dataset_from_directory(
#     directory=f"{data_dir}/train",
#     labels="inferred",
#     label_mode="categorical",
#     image_size=img_size,
#     batch_size=batch_size,
#     shuffle=False # Shuffle is not needed just to get class names
# )
# class_names = original_train_ds.class_names
# Define class names in the same order as training
class_names = ['CNV', 'DME', 'DRUSEN', 'NORMAL']


# Predict
pred = model.predict(img_array)
predicted_index = np.argmax(pred)  # get the index of the highest probability
predicted_class = class_names[predicted_index]  # get class name

for cls, prob in zip(class_names, pred[0]):
    print(f"{cls}: {prob*100:.2f}%")


print("Predicted class:", predicted_class)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 355ms/step
CNV: 100.00%
DME: 0.00%
DRUSEN: 0.00%
NORMAL: 0.00%
Predicted class: CNV
```