Import Libraries

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
from google.colab import files
import zipfile
import cv2
from skimage.feature import hog, local_binary_pattern
from skimage.io import imread
```

Upload Files

```
print("Upload train.zip and test.zip")
uploaded = files.upload()
```

```
Upload train.zip and test.zip
Choose Files 2 files
```
- **test.zip**(application/x-zip-compressed) - 31355611 bytes, last modified: 2/8/2025 - 100% done
- **train.zip**(application/x-zip-compressed) - 136185743 bytes, last modified: 2/8/2025 - 100% done

```
Saving test.zip to test.zip
Saving train.zip to train.zip
```

```
with zipfile.ZipFile('train.zip', 'r') as zip_ref:
    zip_ref.extractall('content/train')

with zipfile.ZipFile('test.zip', 'r') as zip_ref:
    zip_ref.extractall('content/test')



train_dir = 'content/train'
test_dir = 'content/test'

print(f"Train Directory: {os.listdir(train_dir)}")
print(f"Test Directory: {os.listdir(test_dir)}")
```

```
Train Directory: ['train']
Test Directory: ['test']
```

Feature Extraction Methods

```
# SIFT Feature Extraction
def extract_sift_features(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    sift = cv2.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(img, None)
    return descriptors


# SURF Feature Extraction
def extract_surf_features(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    surf = cv2.xfeatures2d.SURF_create()
    keypoints, descriptors = surf.detectAndCompute(img, None)
    return descriptors


#LBP Feature Extraction
def extract_lbp_features(image_path, radius=3, n_points=24):
    img = imread(image_path, as_gray=True)
    lbp = local_binary_pattern(img, n_points, radius, method='uniform')
    return lbp


# HOG Feature Extraction
def extract_hog_features(image_path):
    img = imread(image_path, as_gray=True)
    features, hog_image = hog(img, visualize=True, block_norm='L2-Hys')
    return features
```

Data Preprocessing

```
datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

train_data = datagen.flow_from_directory(
    train_dir,
    target_size=(227, 227),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_data = datagen.flow_from_directory(
    train_dir,
    target_size=(227, 227),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)

test_datagen = ImageDataGenerator(rescale=1.0/255.0)
test_data = test_datagen.flow_from_directory(
    test_dir,
    target_size=(227, 227),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 1124 images belonging to 1 classes.
Found 281 images belonging to 1 classes.
Found 327 images belonging to 1 classes.
```

Build AlexNet Architecture

```
def build_alexnet(input_shape=(227, 227, 3), num_classes=2):
    model = Sequential([
        Conv2D(96, (11, 11), strides=(4, 4), activation='relu', input_shape=input_shape),
        MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
        Conv2D(256, (5, 5), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
        Conv2D(384, (3, 3), activation='relu', padding='same'),
        Conv2D(384, (3, 3), activation='relu', padding='same'),
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),
        Flatten(),
        Dense(4096, activation='relu'),
        Dropout(0.5),
        Dense(4096, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='sigmoid')
    ])
    return model

alexnet = build_alexnet()
alexnet.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 55, 55, 96)	34,944
max_pooling2d_3 (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_6 (Conv2D)	(None, 27, 27, 256)	614,656
max_pooling2d_4 (MaxPooling2D)	(None, 13, 13, 256)	0
conv2d_7 (Conv2D)	(None, 13, 13, 384)	885,120
conv2d_8 (Conv2D)	(None, 13, 13, 384)	1,327,488
conv2d_9 (Conv2D)	(None, 13, 13, 256)	884,992
max_pooling2d_5 (MaxPooling2D)	(None, 6, 6, 256)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_3 (Dense)	(None, 4096)	37,752,832
dropout_2 (Dropout)	(None, 4096)	0
dense_4 (Dense)	(None, 4096)	16,781,312
dropout_3 (Dropout)	(None, 4096)	0
dense_5 (Dense)	(None, 2)	8,194

Total params: 58,289,538 (222.36 MB)
Trainable params: 58,289,538 (222.36 MB)
Non-trainable params: 0 (0.00 B)

Compile and Train the Model

```
alexnet.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = alexnet.fit(
    train_data,
    epochs=10,
    validation_data=val_data,
    callbacks=[early_stopping]
)
```

```
Epoch 1/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 258s 7s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 259s 7s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 253s 7s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 248s 7s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 5/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 250s 7s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 6/10
36/36 ━━━━━━━━━━━━━━━━━━━━ 251s 7s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
```

Evaluate and Save the Model

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

alexnet.save('alexnet_skin_disease_classifier.h5')
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 16s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
Test Accuracy: 100.00%
```

Test on a Sample Image

```
from google.colab import files
import cv2
```

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

# Function to extract HOG features
def extract_hog_features(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    hog = cv2.HOGDescriptor()
    hog_features = hog.compute(img)
    return hog_features

# Upload and save sample image
print("Upload a sample image for testing:")
uploaded = files.upload()
sample_image_path = list(uploaded.keys())[0]

# Extract HOG features from the sample image
print("Extracting HOG features...")
hog_features = extract_hog_features(sample_image_path)
print(f"HOG Features Shape: {hog_features.shape}")

# Load and preprocess image for prediction
img = cv2.imread(sample_image_path)
img = cv2.resize(img, (227, 227))  # Resize to model input size
img_array = img_to_array(img) / 255.0  # Normalize
img_array = np.expand_dims(img_array, axis=0)

# Load trained model
model = load_model('/content/alexnet_skin_disease_classifier.h5')  # Replace with your model path

# Predict class
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions, axis=1)[0]
print(f"Predicted Class: {predicted_class}")
```

```
Upload a sample image for testing:
Choose Files Lichen-planus-body.jpg
• Lichen-planus-body.jpg(image/jpeg) - 258772 bytes, last modified: 2/8/2025 - 100% done
Saving Lichen-planus-body.jpg to Lichen-planus-body (2).jpg
Extracting HOG features...
HOG Features Shape: (34783560,)
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until y
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 225ms/step
```

Start coding or generate with AI.