

✓ Hand Gesture Recognition with openCV

```
# Import necessary libraries
import numpy as np
import pandas as pd
import os
import cv2
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle # Import the shuffle function
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import seaborn as sns
import random
from scipy.stats import norm
import keras
from keras import backend as K
K.clear_session()
import kagglehub
```

```
image_shape = (80, 80, 1) # Adjusted to match your hand gesture images' size and grayscale
batch_size = 64
latent_dim = 16
epochs = 5
```

```
!wget https://www.kaggle.com/api/v1/datasets/download/gti-upm/leapgestrecog?dataset_version_number=1
```

```
--2024-12-15 11:43:11-- https://www.kaggle.com/api/v1/datasets/download/gti-upm/leapgestrecog?dataset_version_number=1
Resolving www.kaggle.com (www.kaggle.com)... 35.244.233.98
Connecting to www.kaggle.com (www.kaggle.com)|35.244.233.98|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://storage.googleapis.com/443/kaggle-data-sets/39466/61155/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Algorithm=GOOG4-RS
--2024-12-15 11:43:12-- https://storage.googleapis.com/kaggle-data-sets/39466/61155/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RS
Resolving storage.googleapis.com (storage.googleapis.com)... 142.251.170.207, 173.194.174.207, 74.125.23.207, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|142.251.170.207|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2286085038 (2.1G) [application/zip]
Saving to: 'leapgestrecog?dataset_version_number=1'

leapgestrecog?datas 100%[=====>] 2.13G 29.1MB/s in 75s

2024-12-15 11:44:28 (29.0 MB/s) - 'leapgestrecog?dataset_version_number=1' saved [2286085038/2286085038]
```

```
!unzip leapgestrecog?dataset_version_number=1
```

 [Afficher la sortie masquée](#)

```
# Parameters
IMG_SIZE = 80 # Image size to resize to
data_path = '/content/leapGestRecog/' # Path to dataset folder

# Initialize lists to store images and labels
x_data = []
y_data = []

# Dictionaries for label encoding and decoding
lookup = {}
reverselookup = {}
label_count = 0

# Create label mappings based on subfolders
```

```

for folder_name in os.listdir(data_path + '00/'):
    if not folder_name.startswith('.'):
        lookup[folder_name] = label_count
        reverselookup[label_count] = folder_name
        label_count += 1

# Load images and labels
for i in range(10): # Assumes folders named '00' to '09'
    folder_path = f'{data_path}0{i}/'
    for gesture_folder in os.listdir(folder_path):
        if not gesture_folder.startswith('.'):
            for image_file in os.listdir(f'{folder_path}{gesture_folder}/'):
                # Load and resize each image
                img_path = f'{folder_path}{gesture_folder}/{image_file}'
                img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

                # Check if image is loaded successfully
                if img is None:
                    print(f"Warning: Image {img_path} could not be loaded.")
                    continue

                # Resize and add to dataset
                img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
                x_data.append(img)
                y_data.append(lookup[gesture_folder])

# Convert lists to numpy arrays and normalize
x_data = np.array(x_data, dtype='float32').reshape(-1, IMG_SIZE, IMG_SIZE, 1) / 255.0
y_data = np.array(y_data)

# One-hot encode labels for classification
y_data = to_categorical(y_data, num_classes=len(lookup))

# Shuffle the dataset
x_data, y_data = shuffle(x_data, y_data, random_state=42)

# Split into training and validation sets (80% train, 20% validation)
x_train, x_val, y_train, y_val = train_test_split(x_data, y_data, test_size=0.2, random_state=42)

# Display dataset info
print("Dataset loaded.")
print(f"Total images: {len(x_data)}")
print(f"Image shape: {x_data[0].shape}")
print(f"Number of classes: {len(lookup)}")

# Optional: Check label mappings
print("Label mappings:", lookup)

# Optional: Check the shapes of the train/validation split
print(f"Training set shape: {x_train.shape}, {y_train.shape}")
print(f"Validation set shape: {x_val.shape}, {y_val.shape}")

```

Dataset loaded.
 Total images: 20000
 Image shape: (80, 80, 1)
 Number of classes: 10
 Label mappings: {'10_down': 0, '01_palm': 1, '04_fist_moved': 2, '06_index': 3, '03_fist': 4, '02_1': 5, '07_ok': 6, '09_c': 7, '08_2': 8, '05_3': 9}
 Training set shape: (16000, 80, 80, 1), (16000, 10)
 Validation set shape: (4000, 80, 80, 1), (4000, 10)

```

def get_images_1_to_10(x_data, y_data):
    # Initialize empty lists to store selected images and labels
    selected_images = []
    selected_labels = []

    # Loop through labels 0 to 9
    for label in range(10):
        # Get indices where the label is equal to the current label
        label_indices = [i for i, lbl in enumerate(y_data) if np.argmax(lbl) == label]

```

```

# Randomly select one image from the indices of that label
selected_index = random.choice(label_indices)

# Append the image and its label to the selected lists
selected_images.append(x_data[selected_index])
selected_labels.append(label)

return np.array(selected_images), np.array(selected_labels)

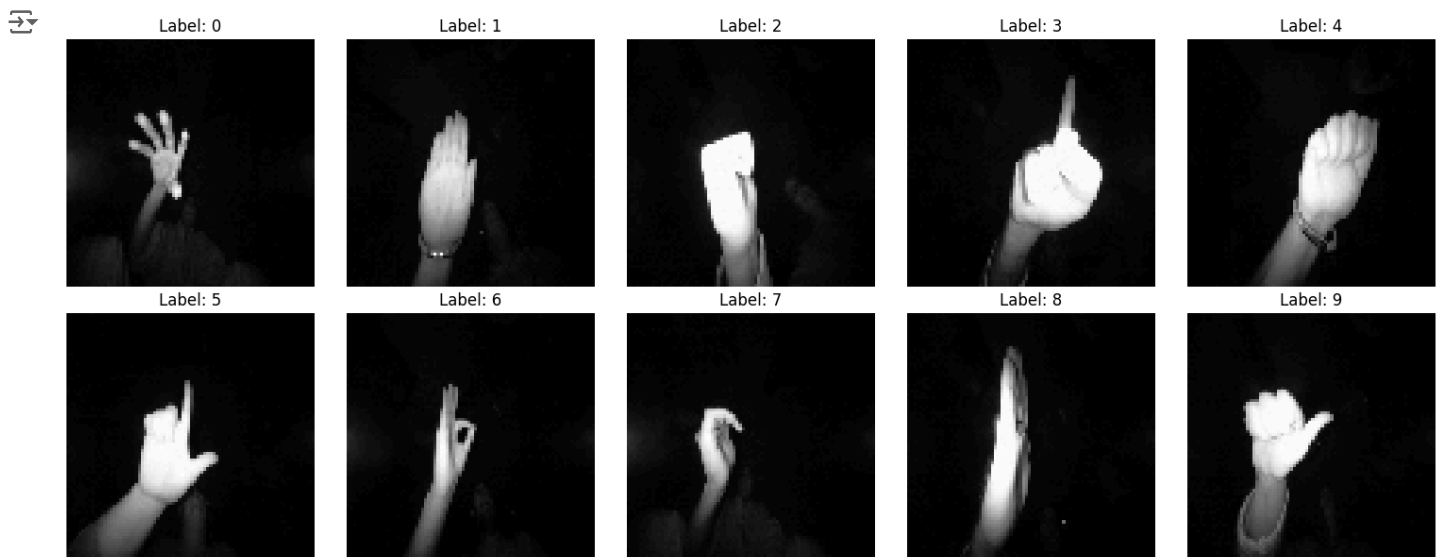
# Select random 10 images of labeled 0 to 9
selected_x, selected_y = get_images_1_to_10(x_train, y_train)

# To visualize the selected images, you can use matplotlib
import matplotlib.pyplot as plt

# Display the selected images
fig, axes = plt.subplots(2, 5, figsize=(15, 6))
for ax, img, label in zip(axes.ravel(), selected_x, selected_y):
    ax.imshow(img.reshape(image_shape[0], image_shape[1]), cmap='gray')
    ax.set_title(f'Label: {label}')
    ax.axis('off')

plt.tight_layout()
plt.show()

```



```

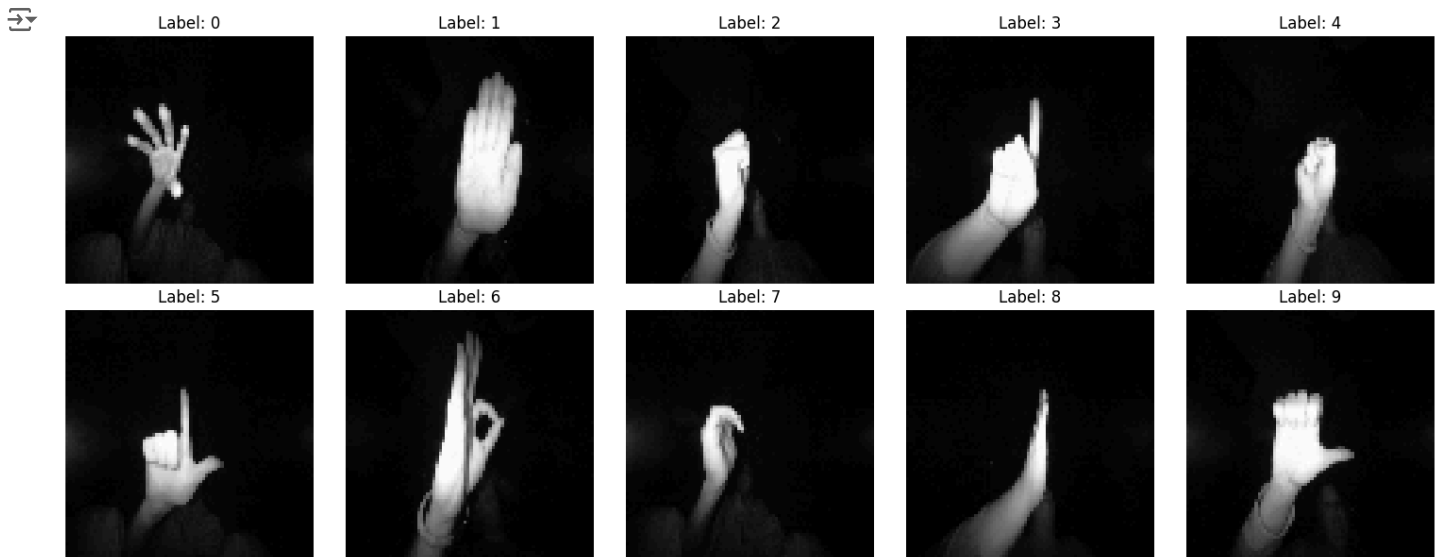
# Select random 10 images of labeled 0 to 9
selected_x, selected_y = get_images_1_to_10(x_train, y_train)

# To visualize the selected images, you can use matplotlib
import matplotlib.pyplot as plt

# Display the selected images
fig, axes = plt.subplots(2, 5, figsize=(15, 6))
for ax, img, label in zip(axes.ravel(), selected_x, selected_y):
    ax.imshow(img.reshape(image_shape[0], image_shape[1]), cmap='gray')
    ax.set_title(f'Label: {label}')
    ax.axis('off')

plt.tight_layout()
plt.show()

```



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Define the CNN model architecture
model = Sequential()

# Add Convolutional layers
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten the output of the last pooling layer
model.add(Flatten())

# Add fully connected layers
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5)) # Dropout for regularization

model.add(Dense(len(lookup), activation='softmax')) # Output layer with softmax activation

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Model summary
model.summary()
```

→ /usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`
super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 78, 78, 32)	320
max_pooling2d (MaxPooling2D)	(None, 39, 39, 32)	0
conv2d_1 (Conv2D)	(None, 37, 37, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 18, 18, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 128)	1,048,704
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 1,142,666 (4.36 MB)

Trainable params: 1,142,666 (4.36 MB)

Non-trainable params: 0 (0.00 B)

```
# Train the CNN model
history = model.fit(
    x_train, y_train,
    epochs=5,
    batch_size=64,
    validation_data=(x_val, y_val),
    verbose=2
)

# Evaluate the model on validation data
val_loss, val_acc = model.evaluate(x_val, y_val, verbose=2)
print(f'Validation Accuracy: {val_acc * 100:.2f}%')
```

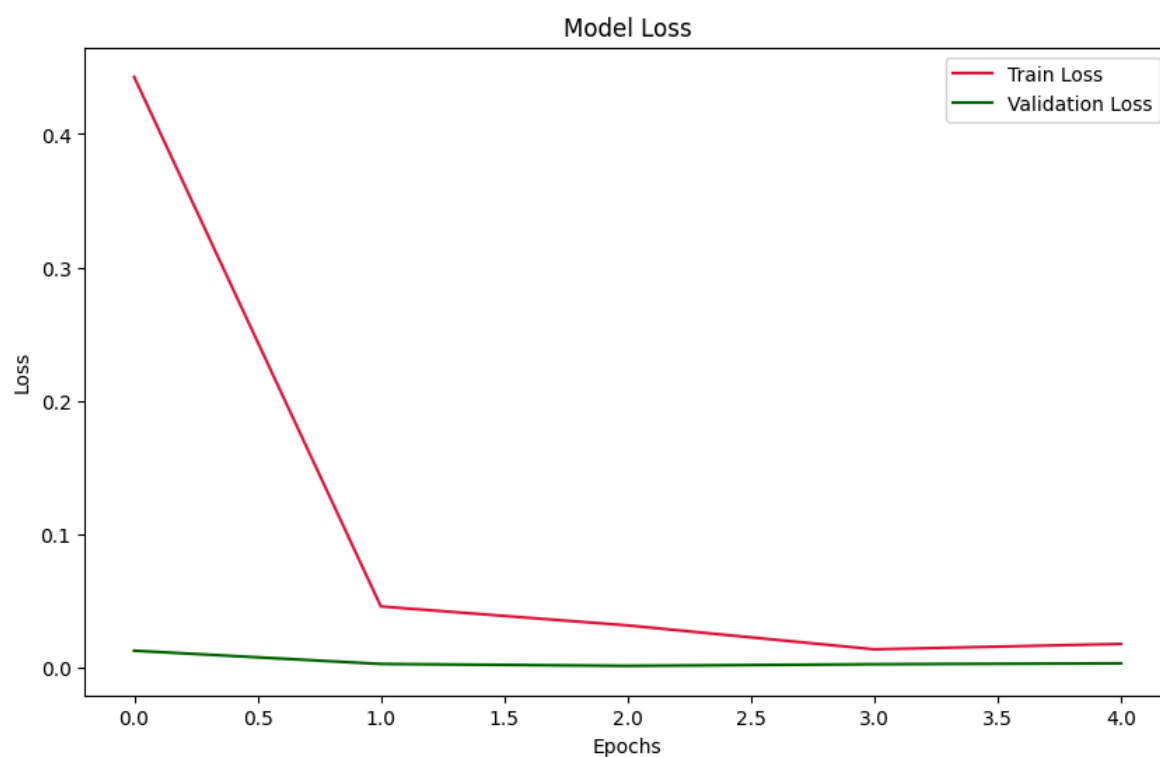
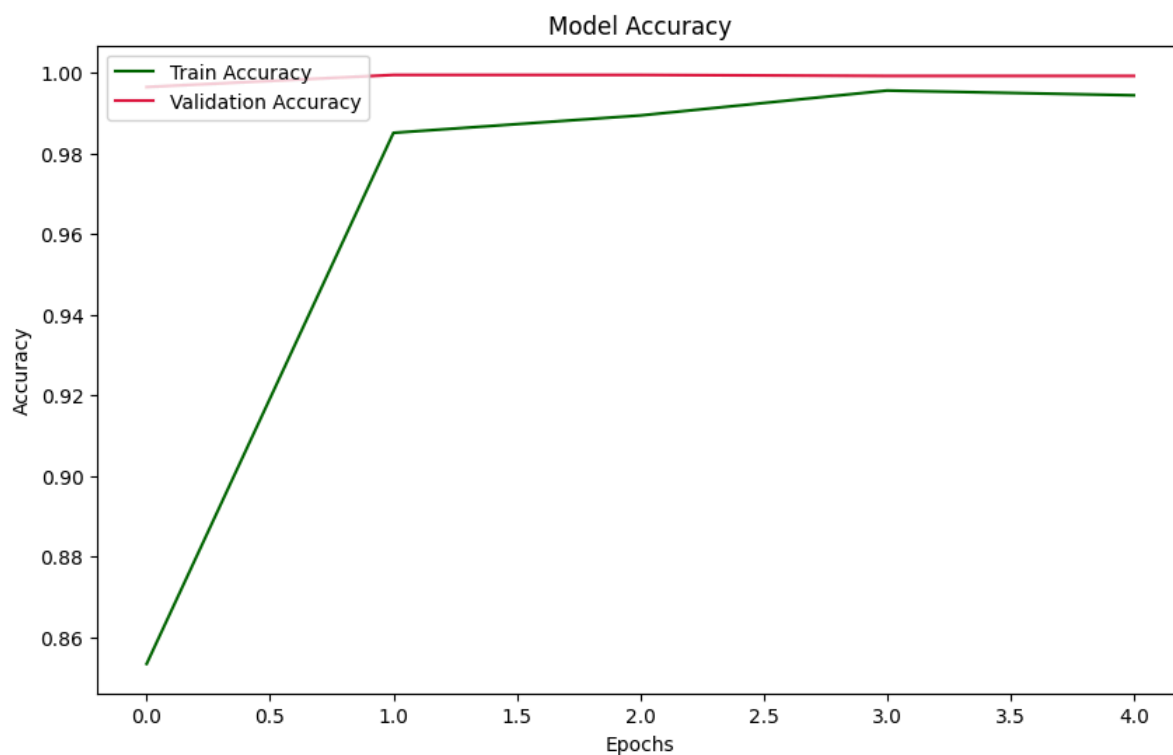
→ Epoch 1/5
250/250 - 198s - 793ms/step - accuracy: 0.8533 - loss: 0.4427 - val_accuracy: 0.9965 - val_loss: 0.0129
Epoch 2/5
250/250 - 195s - 778ms/step - accuracy: 0.9851 - loss: 0.0461 - val_accuracy: 0.9995 - val_loss: 0.0030
Epoch 3/5
250/250 - 194s - 776ms/step - accuracy: 0.9894 - loss: 0.0319 - val_accuracy: 0.9995 - val_loss: 0.0016
Epoch 4/5
250/250 - 202s - 806ms/step - accuracy: 0.9956 - loss: 0.0140 - val_accuracy: 0.9992 - val_loss: 0.0028
Epoch 5/5
250/250 - 203s - 812ms/step - accuracy: 0.9944 - loss: 0.0180 - val_accuracy: 0.9992 - val_loss: 0.0035
125/125 - 12s - 94ms/step - accuracy: 0.9992 - loss: 0.0035
Validation Accuracy: 99.92%

```
import matplotlib.pyplot as plt

# Plot the training & validation accuracy
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Train Accuracy', color='darkgreen')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='crimson')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='upper left')
plt.show()

# Plot the training & validation loss
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Train Loss', color="crimson")
plt.plot(history.history['val_loss'], label='Validation Loss', color="darkgreen")
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend(loc='upper right')
plt.show()
```



```
# Select a random image from the validation set
random_idx = random.randint(0, len(x_val) - 1)
img = x_val[random_idx].reshape(1, IMG_SIZE, IMG_SIZE, 1)

# Predict the class of the image
prediction = model.predict(img)
predicted_class = np.argmax(prediction)
```

```
# Get the label name using reverse lookup
predicted_label = reverselookup[predicted_class]

# Display the predicted label and the image
plt.imshow(x_val[random_idx].reshape(IMG_SIZE, IMG_SIZE), cmap='gray')
plt.title(f"Predicted Label: {predicted_label}")
plt.axis('off')
plt.show()
```

1/1 — 0s 132ms/step

Predicted Label: 07_ok



```
# Save the model to a file
model.save('hand_gesture_cnn_model.h5')

# Load the saved model
from tensorflow.keras.models import load_model
loaded_model = load_model('hand_gesture_cnn_model.h5')

# Verify the model is loaded successfully
loaded_model.summary()
```

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is deprecated. Please use the Keras 2.x format instead.
 WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until model training starts.
 Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 78, 78, 32)	320
max_pooling2d (MaxPooling2D)	(None, 39, 39, 32)	0
conv2d_1 (Conv2D)	(None, 37, 37, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 18, 18, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 128)	1,048,704
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 1,142,668 (4.36 MB)
 Trainable params: 1,142,666 (4.36 MB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 2 (12.00 B)