sign-language-mnist : https://www.kaggle.com/datasets/grassknoted/asl-alphabet/data?select=asl_alphabet_train

chatbot dataset : https://www.kaggle.com/datasets/kreeshrajani/3k-conversations-dataset-for-chatbot

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
import pickle
```

Read the data

In [ ]:
```
# Define the directory containing your data
data_dir = '..\\dataset\\asl_alphabet_train'

# Initialize a list to store the data
data = []

# Loop through each folder and file in the directory
for label in os.listdir(data_dir):
    if os.path.isdir(os.path.join(data_dir, label)):  # Check if it's a directory
        for image in os.listdir(os.path.join(data_dir, label)):
            image_path = os.path.join(data_dir, label, image)
            data.append([image_path, label])

# Create a DataFrame
asl_train_df = pd.DataFrame(data, columns=['image_path', 'label'])

# Show the first few rows of the DataFrame
asl_train_df.head()
```

Out[ ]:

	image_path	label
0	..\dataset\asl_alphabet_train\A\A1.jpg	A
1	..\dataset\asl_alphabet_train\A\A10.jpg	A
2	..\dataset\asl_alphabet_train\A\A100.jpg	A
3	..\dataset\asl_alphabet_train\A\A1000.jpg	A
4	..\dataset\asl_alphabet_train\A\A1001.jpg	A

In [ ]:
```
asl_train_df.shape
```

Out[ ]:
```
(87000, 2)
```

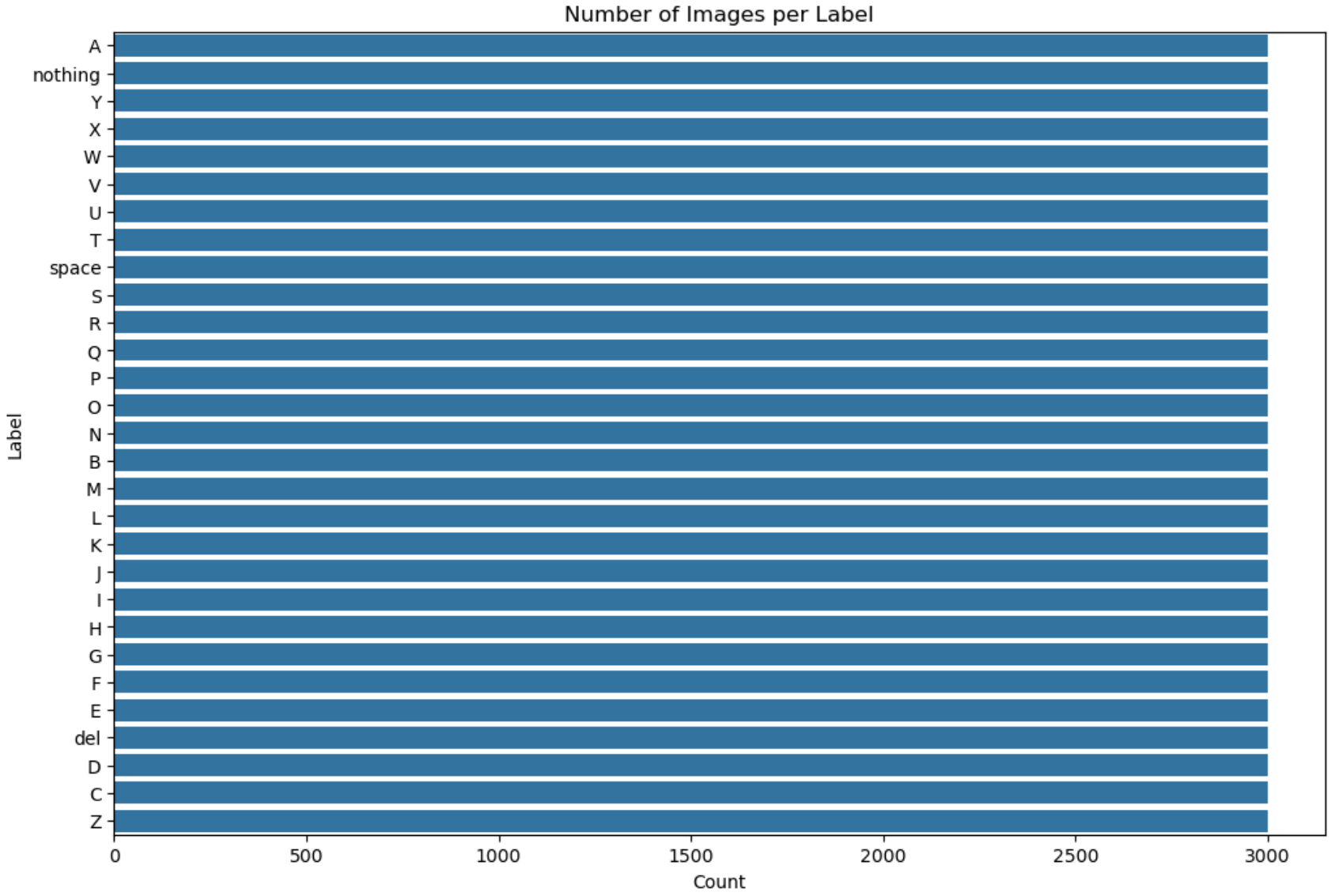
In [ ]:
```
asl_train_df['label'].value_counts()
```

Out[ ]:
```
label
A          3000
nothing    3000
Y          3000
X          3000
W          3000
V          3000
U          3000
T          3000
space      3000
S          3000
R          3000
Q          3000
P          3000
O          3000
N          3000
B          3000
M          3000
L          3000
K          3000
J          3000
I          3000
H          3000
G          3000
F          3000
E          3000
del        3000
D          3000
C          3000
Z          3000
Name: count, dtype: int64
```

Quick EDA

In [ ]:
```
import seaborn as sns
# Plotting
plt.figure(figsize=(12, 8))  # Set the figure size
sns.countplot(y='label', data=asl_train_df, order = asl_train_df['label'].value_counts().index)  # Create a countplot
plt.title('Number of Images per Label')  # Set title
plt.xlabel('Count')  # Set x-axis label
```

```
plt.ylabel('Label')  # Set y-axis label
plt.show()  # Show the plot
```



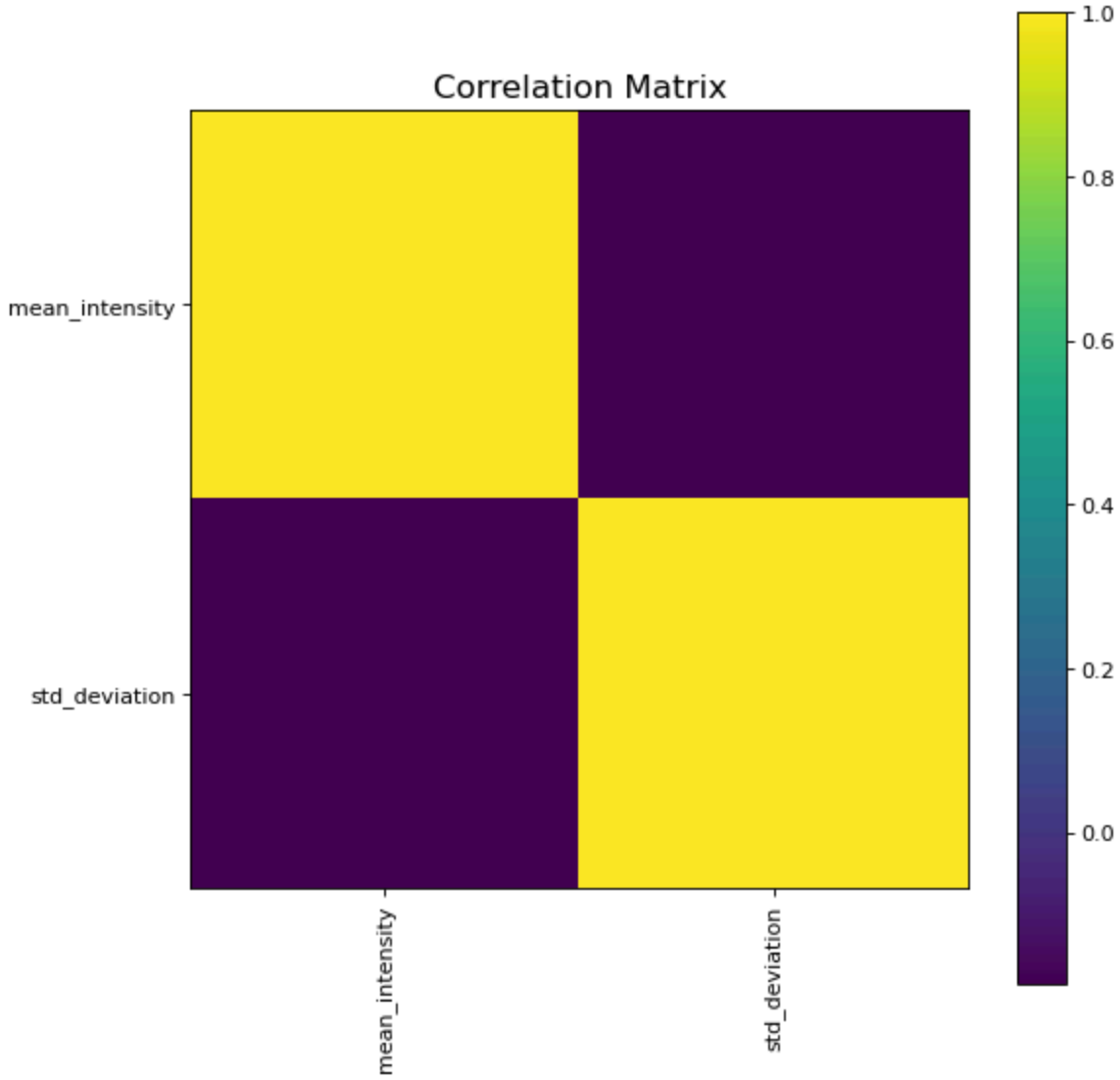
Feature Extraction (Using Mean Pixel Intensity & Standard Deviation)

In [ ]:
```
from PIL import Image
import numpy as np
import pandas as pd

# Define a function to extract mean intensity from an image
def extract_mean_intensity(image_path):
    image = Image.open(image_path)
    image = image.convert('L')  # Convert to grayscale
    array = np.array(image)
    return np.mean(array)

# Define a function to extract standard deviation of intensity from an image
def extract_std_deviation(image_path):
    image = Image.open(image_path)
    image = image.convert('L')  # Convert to grayscale
    array = np.array(image)
    return np.std(array)


# Apply feature extraction to each image in the DataFrame
asl_train_df['mean_intensity'] = asl_train_df['image_path'].apply(extract_mean_intensity)
asl_train_df['std_deviation'] = asl_train_df['image_path'].apply(extract_std_deviation)
```

Correlation Matrix plot between Brightness and Contrast

In [ ]:
```
import matplotlib.pyplot as plt

def plotCorrelationMatrix(df, graphWidth):
    df_numeric = df.select_dtypes(include=[np.number])  # Select only numeric columns
    if df_numeric.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or constant columns ({df_numeric.shape[1]}) is less
        return
    corr = df_numeric.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w', edgecolor='k')
    corrMat = plt.matshow(corr, fignum=1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
```

```
    plt.title('Correlation Matrix', fontsize=15)
    plt.show()

# Plot the correlation matrix
plotCorrelationMatrix(asl_train_df, graphWidth=8)
```



In image processing and analysis, feature extraction such as mean pixel intensity and standard deviation is crucial for distilling complex image data into more interpretable, numerical forms.

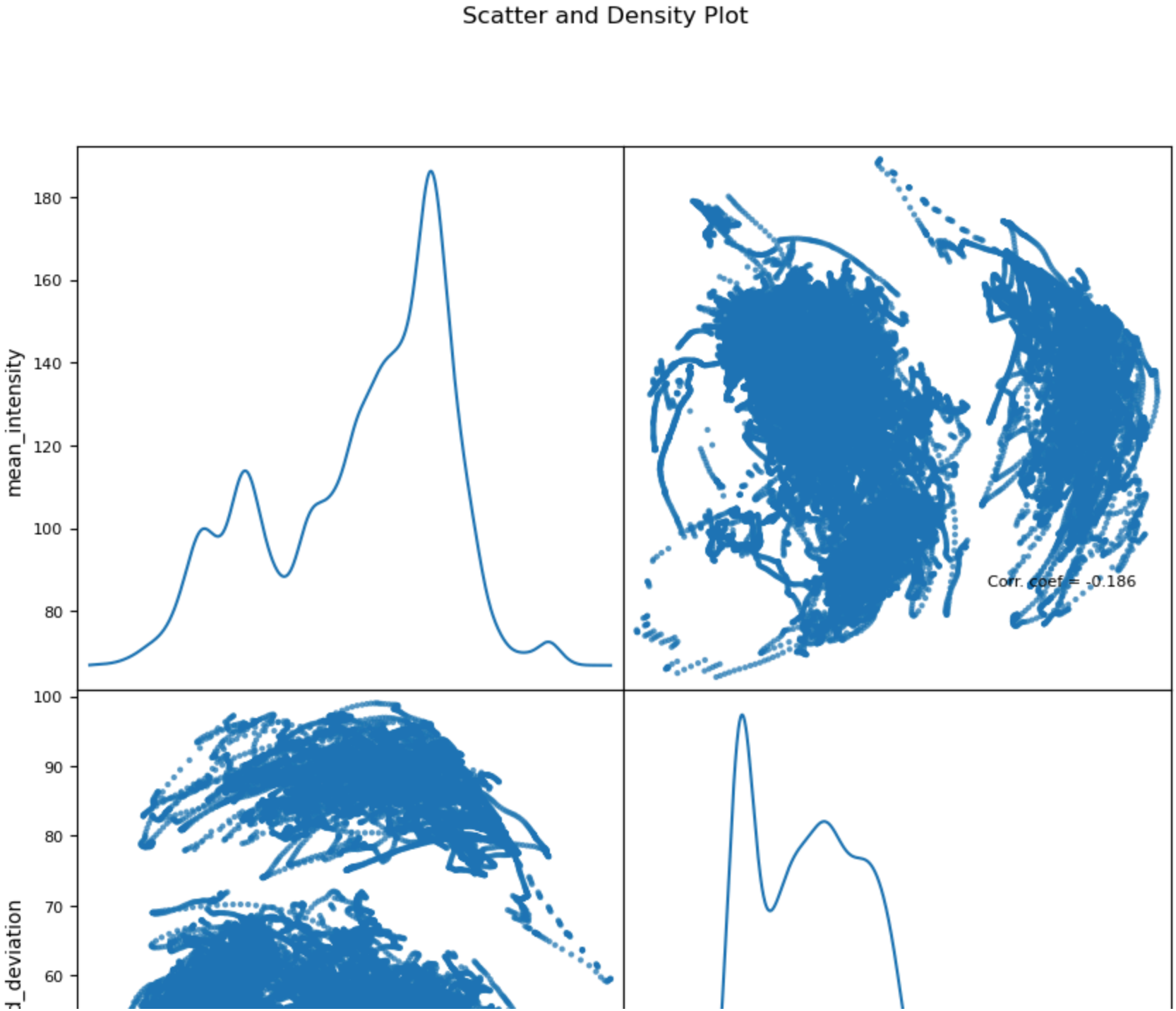
Mean pixel intensity provides a measure of the overall brightness or luminance of an image, which can be critical for tasks that involve brightness normalization or thresholding.

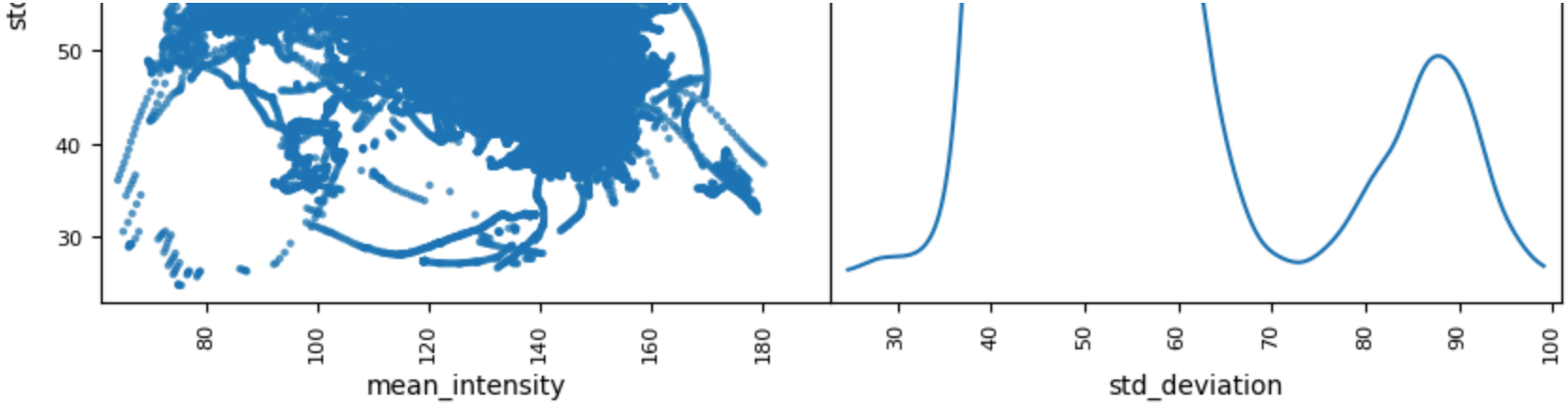
Standard deviation, on the other hand, measures the variability or contrast within the pixel values, offering insights into the texture and detail present in the image.

The correlation matrix plot, showing a negative correlation between these two features, suggests that images with higher average brightness tend to have less contrast and variability in their pixel values. Understanding this relationship helps in optimizing image processing tasks and improving algorithms for tasks like image classification, where consistent image quality can influence performance.

Scatter and Density Plot

In [ ]:
```
import matplotlib.pyplot as plt

def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include=[np.number])  # Keep only numerical columns
    df = df.dropna(axis=1)  # Drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]]  # Keep columns with more than 1 unique value
    columnNames = list(df)
    if len(columnNames) > 10:  # Limit the number of columns to prevent overcrowding
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k=1)):
        ax[i, j].annotate(f'Corr. coef = {corrs[i, j]:.3f}', (0.8, 0.2), xycoords='axes fraction', ha='center', va='
    plt.suptitle('Scatter and Density Plot')
    plt.show()

# Call the function to plot
plotScatterMatrix(asl_train_df, plotSize=10, textSize=8)
```


Scatter and Density Plot
mean_intensity
180
160
140
120
100
80
Corr. coef = -0.186
100
90
80
70
60
d_deviation



The scatter and density plot generated from ASL alphabet image dataset provides crucial insights into the relationships between mean intensity and standard deviation of pixel values.

The density plots indicate predominant brightness levels and variability in image contrast, which are important for understanding image characteristics.

The scatter plot reveals a slight positive correlation between mean intensity and standard deviation, suggesting these features are somewhat related but still provide independent information.

This visualization is essential for guiding feature selection and engineering in machine learning models aimed at classifying ASL signs. Overall, it helps ensure that models are trained on informative, non-redundant features, potentially enhancing classification accuracy.

Creating Training & Validation Dataset

In [ ]:
```
asl_train_df.head()
```

Out[ ]:

	image_path	label	mean_intensity	std_deviation
0	..\dataset\asl_alphabet_train\A\A1.jpg	A	113.822725	56.604702
1	..\dataset\asl_alphabet_train\A\A10.jpg	A	113.820850	56.350592
2	..\dataset\asl_alphabet_train\A\A100.jpg	A	105.395600	58.857844
3	..\dataset\asl_alphabet_train\A\A1000.jpg	A	141.781600	43.187863
4	..\dataset\asl_alphabet_train\A\A1001.jpg	A	141.954150	60.112078

In [ ]:
```
new_asl_df = asl_train_df.drop(['mean_intensity', 'std_deviation'], axis =1)
new_asl_df.head()
```

Out[ ]:

	image_path	label
0	..\dataset\asl_alphabet_train\A\A1.jpg	A
1	..\dataset\asl_alphabet_train\A\A10.jpg	A
2	..\dataset\asl_alphabet_train\A\A100.jpg	A
3	..\dataset\asl_alphabet_train\A\A1000.jpg	A
4	..\dataset\asl_alphabet_train\A\A1001.jpg	A

In [ ]:
```
from sklearn.model_selection import train_test_split

# split the dataframe into training and validation sets
asl_train, asl_val = train_test_split(new_asl_df, test_size=0.1, shuffle=True, random_state=42)

# Print the shapes of the training and validation sets
print("Training set shape:", asl_train.shape)
print("Validation set shape:", asl_val.shape)
```

```
Training set shape: (78300, 2)
Validation set shape: (8700, 2)
```

Data Augmentation

In [ ]:
```
%%time

from tensorflow.keras.preprocessing.image import ImageDataGenerator  # Import ImageDataGenerator for data augmentatio

# Define batch size and image size
batch_size = 32
image_size = (64, 64)

# Initialize the ImageDataGenerator for training with data augmentation parameters
train_generator = ImageDataGenerator(rescale=1./255,  # Rescale pixel values to [0, 1]
                                     rotation_range=10,  # Randomly rotate images by up to 10 degrees
                                     height_shift_range=0.1,  # Randomly shift images vertically by up to 10%
                                     width_shift_range=0.1,  # Randomly shift images horizontally by up to 10%
                                     shear_range=0.1,  # Apply shear transformations
                                     zoom_range=0.1,  # Randomly zoom in on images by up to 10%
                                     horizontal_flip=True,  # Randomly flip images horizontally
                                     fill_mode='nearest')  # Fill in new pixels with the nearest pixel values

# Initialize the ImageDataGenerator for validation without data augmentation
val_generator = ImageDataGenerator(rescale=1./255)  # Rescale pixel values to [0, 1]

# Create the training image generator from the dataframe
train_images = train_generator.flow_from_dataframe(asl_train, x_col='image_path', y_col='label',
                                                   color_mode='grayscale', class_mode='categorical',
                                                   batch_size=batch_size, target_size=image_size,
                                                   shuffle=True, seed=0)

# Create the validation image generator from the dataframe
val_images = val_generator.flow_from_dataframe(asl_val, x_col='image_path', y_col='label',
                                               color_mode='grayscale', class_mode='categorical',
                                               batch_size=batch_size, target_size=image_size)
```

```
Found 78300 validated image filenames belonging to 29 classes.
Found 8700 validated image filenames belonging to 29 classes.
CPU times: total: 328 ms
Wall time: 3.49 s
```

Model Architecture

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, GlobalAveragePooling2D, BatchNormalization, Dense, D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

# Determine the output shape based on the number of unique labels in the training data
output_shape = len(set(train_images.labels))

# Define the model architecture
model = Sequential([
    Input(shape=image_size + (1,), name='input_layer'),  # Input layer with the shape of the images

    Conv2D(16, (3, 3), activation='relu'),  # First convolutional layer with 16 filters
    MaxPooling2D(pool_size=(2, 2)),  # Max pooling layer with pool size of 2x2

    Conv2D(32, (3, 3), activation='relu'),  # Second convolutional layer with 32 filters
    MaxPooling2D(pool_size=(2, 2)),  # Max pooling layer with pool size of 2x2

    Conv2D(64, (3, 3), activation='relu'),  # Third convolutional layer with 64 filters
    MaxPooling2D(pool_size=(2, 2)),  # Max pooling layer with pool size of 2x2

    Conv2D(128, (3, 3), activation='relu'),  # Fourth convolutional layer with 128 filters

    GlobalAveragePooling2D(),  # Global average pooling layer
    BatchNormalization(),  # Batch normalization layer

    Dense(512, activation='relu'),  # First dense layer with 512 units
    Dropout(0.5),  # Dropout layer with 50% dropout rate

    Dense(256, activation='relu'),  # Second dense layer with 256 units
    Dropout(0.5),  # Dropout layer with 50% dropout rate

    Dense(128, activation='relu'),  # Third dense layer with 128 units

    Dense(output_shape, activation='softmax', name='output_layer')  # Output layer with softmax activation
])

# Compile the model with Adam optimizer, categorical crossentropy loss, and accuracy metric
model.compile(optimizer=Adam(learning_rate=1e-3),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])
```

```
# Print the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 16)        160

 max_pooling2d (MaxPooling2D  (None, 31, 31, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_3 (Conv2D)           (None, 4, 4, 128)         73856

 global_average_pooling2d (G  (None, 128)              0
 lobalAveragePooling2D)

 batch_normalization (BatchN  (None, 128)              512
 ormalization)

 dense (Dense)               (None, 512)               66048

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 output_layer (Dense)        (None, 29)                3741

=================================================================
Total params: 331,677
Trainable params: 331,421
```

```
Non-trainable params: 256
_________________________________________________________________
```

Model Training

In [ ]:
```python
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint

# Define the ReduceLROnPlateau callback
reduce_lr = ReduceLROnPlateau(monitor='val_loss',  # Monitor the validation loss
                              factor=0.2,  # Factor by which the learning rate will be reduced
                              patience=10,  # Number of epochs with no improvement after which learning rate will be
                              min_lr=1e-6)  # Lower bound on the learning rate

# Define the ModelCheckpoint callback
model_checkpoint = ModelCheckpoint('best_model_3.h5',  # File path to save the model
                                   monitor='val_accuracy',  # Monitor validation accuracy
                                   save_best_only=True,  # Save only the best model
                                   mode='max',  # Save the model with the maximum accuracy
                                   verbose=1)  # Print messages when saving

# Train the model
history = model.fit(
    train_images,  # Training data
    epochs=100,  # Number of epochs to train
    validation_data=val_images,  # Validation data
    callbacks=[reduce_lr, model_checkpoint]  # List of callbacks to be called during training
)

# Print the training history
print(history.history)
```

Save the trained model

In [ ]:
```python
# Save the model to the current directory
model.save('best_model_3.h5')
```

Visualize the Training and Validation Loss and Accuracy

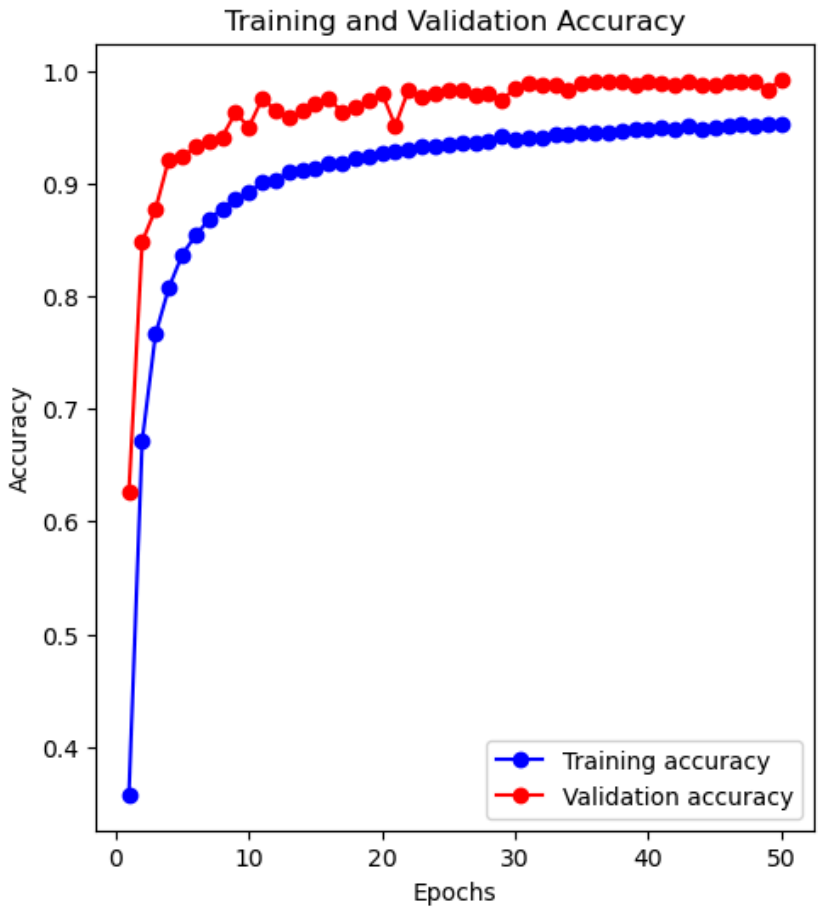
In [ ]:
```
# Load the dictionary from the file
with open('history_dict.pkl', 'rb') as file:
    loaded_history_dict = pickle.load(file)

print(loaded_history_dict)
```

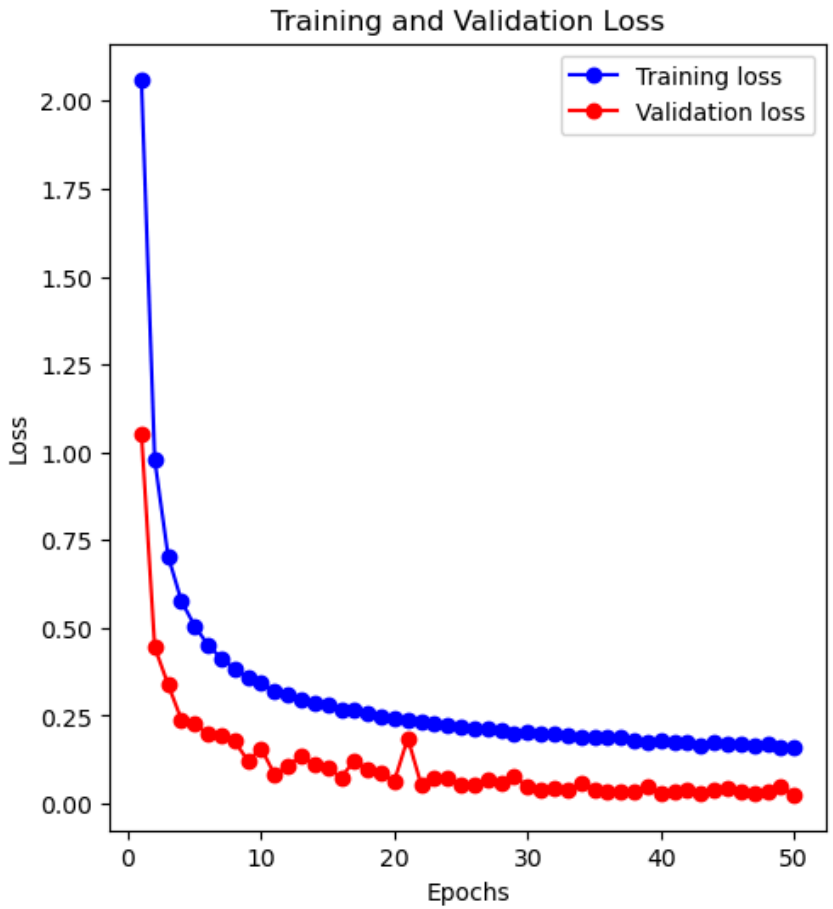
```
{'loss': [2.0612993240356445, 0.9776553511619568, 0.7052192091941833, 0.5767728686332703, 0.5050854086875916, 0.45136
82425022125, 0.4128209054470062, 0.38478416204452515, 0.35735568404197693, 0.34323057532310486, 0.3191038966178894,
0.31109756231307983, 0.29362526535987854, 0.28838837146759033, 0.28270918130874634, 0.2647415101528168, 0.26904365420
34149, 0.2560632824897766, 0.24935513734817505, 0.24345237016677856, 0.23681969940662384, 0.23177726566791534, 0.2275
576889514923, 0.22315919399261475, 0.22021780908107758, 0.21389268338680267, 0.21422119438648224, 0.2078104615211486
8, 0.2003995180130005, 0.20495964586734772, 0.19723382592201233, 0.19952285289764404, 0.19254736602306366, 0.19055277
109146118, 0.1895037740468979, 0.18788780272006989, 0.1867915391921997, 0.17760175466537476, 0.17685775458812714, 0.1
778547316789627, 0.17595550417900085, 0.17344753444194794, 0.16609622538089752, 0.1761433482170105, 0.169077515602111
82, 0.1687229722738266, 0.16293971240520477, 0.16799740493297577, 0.15929579734802246, 0.1616162359714508], 'accurac
y': [0.3575989902019501, 0.671890139579773, 0.7662707567214966, 0.8075861930847168, 0.8367305397987366, 0.85392081737
51831, 0.8678927421569824, 0.8776883482933044, 0.8866794109344482, 0.8924010396003723, 0.9007534980773926, 0.90332055
09185791, 0.909885048866272, 0.911596417427063, 0.9139846563339233, 0.9186078906059265, 0.9173818826675415, 0.9230396
151542664, 0.9246998429298401, 0.9266539216041565, 0.9287228584289551, 0.9296551942825317, 0.932554304599762, 0.93315
45233726501, 0.9340868592262268, 0.9361174702644348, 0.936296284198761, 0.9383907914161682, 0.9416986107826233, 0.939
3231272697449, 0.941123902797699, 0.9406002759933472, 0.943154513835907, 0.9440357685089111, 0.944763720035553, 0.945
0319409370422, 0.945466160774231, 0.9473180174827576, 0.9480332136154175, 0.9483014345169067, 0.9492847919464111, 0.9
489654898643494, 0.951609194278717, 0.9489654898643494, 0.949693500995636, 0.9513665437698364, 0.9527458548545837, 0.
9513282179832458, 0.9532694816589355, 0.9528607726097107], 'val_loss': [1.0541906356811523, 0.444621741771698, 0.3390
044867992401, 0.2391912192106247, 0.22841843962669373, 0.19772784411907196, 0.19614386558532715, 0.17721523344516754,
0.12153453379869461, 0.15693095326423645, 0.08178400248289108, 0.10819496959447861, 0.13468769192695618, 0.1094907447
6957321, 0.10000232607126236, 0.07473992556333542, 0.1225825846195221, 0.09908071160316467, 0.08739268034696579, 0.06
52661845088005, 0.1860141009092331, 0.05432462692260742, 0.07422803342342377, 0.07145791500806808, 0.0522249788045883
2, 0.05560854449868202, 0.0669073835015297, 0.05947130173444748, 0.07903701812028885, 0.047668736428022385, 0.0368778
7055969238, 0.04293560981750488, 0.04073639586567879, 0.05600762367248535, 0.03980637714266777, 0.03538597747683525,
0.032934051007032394, 0.03214767575263977, 0.046203628182411194, 0.029608992859721184, 0.03486257046461105, 0.0397965
3865098953, 0.03152037411928177, 0.04000677168369293, 0.04500666633248329, 0.035874754190444946, 0.03077500313520431
5, 0.034316401928663254, 0.04937422275543213, 0.025663085281848907], 'val_accuracy': [0.6265517473220825, 0.848390817
6422119, 0.8774712681770325, 0.9217241406440735, 0.9234482645988464, 0.932758629322052, 0.9373562932014465, 0.9412643
909454346, 0.9631034731864929, 0.949999988079071, 0.9752873778343201, 0.9656322002410889, 0.9593103528022766, 0.96471
26197814941, 0.9709195494651794, 0.9760919809341431, 0.9632183909416199, 0.9675862193107605, 0.9733333587646484, 0.98
0344831943512, 0.9512643814086914, 0.9836781620979309, 0.977011501789093, 0.9798850417137146, 0.9834482669830322, 0.9
83563244342804, 0.9790804386138916, 0.9806896448135376, 0.9742528796195984, 0.9847126603126526, 0.9887356162071228,
0.9872413873672485, 0.9874712824821472, 0.9832183718681335, 0.9887356162071228, 0.9903448224067688, 0.990229904651641
8, 0.9905747175216675, 0.9873563051223755, 0.9910345077514648, 0.9887356162071228, 0.9877011775970459, 0.990114927291
8701, 0.9880459904670715, 0.9871264100074768, 0.9903448224067688, 0.9902299046516418, 0.9904597997665405, 0.983448266
9830322, 0.9925287365913391], 'lr': [0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.0
01, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.00
1, 0.001, 0.001, 0.001, 0.001, 0.001]}
```


In [ ]:

```
import matplotlib.pyplot as plt
# Extract the history data
```

```
# history_dict = history.history
history_dict = loaded_history_dict

# Extract the metrics
train_acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
train_loss = history_dict['loss']
val_loss = history_dict['val_loss']

# Create epochs range for x-axis
epochs = range(1, len(train_acc) + 1)

# Plot training and validation accuracy
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)  # 1 row, 2 columns, first subplot
plt.plot(epochs, train_acc, 'bo-', label='Training accuracy')
plt.plot(epochs, val_acc, 'ro-', label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training and validation loss
plt.subplot(1, 2, 2)  # 1 row, 2 columns, second subplot
plt.plot(epochs, train_loss, 'bo-', label='Training loss')
plt.plot(epochs, val_loss, 'ro-', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Display the plots
plt.show()
```





Model Evaluation on ASL Test dataset

In [ ]:
```
from tensorflow.keras.models import load_model

# Load the model
model = load_model("best_model_3.h5")
```

Count the number of images in test folder

In [ ]:
```
import os

# Directory containing test data
test_data_dir = '../dataset/asl_alphabet_test'


# List the files in the directory and count them
num_images = len([name for name in os.listdir(test_data_dir) if os.path.isfile(os.path.join(test_data_dir, name))])
print(f"Number of images in the test folder: {num_images}")
```

```
Number of images in the test folder: 28
```

Visualize images from test folder

In [ ]:
```
import matplotlib.pyplot as plt
import os
from PIL import Image

# test_data_dir is already defined above
image_files = [os.path.join(test_data_dir, f) for f in os.listdir(test_data_dir) if os.path.isfile(os.path.join(test_

#  there are exactly 28 images
assert len(image_files) == 28, "The number of images does not match the expected count of 28."

# Calculate number of rows needed for 5 images per row
num_rows = (len(image_files) + 4) // 5  # This will automatically round up if not a full row

# Plot the images
fig, axes = plt.subplots(nrows=num_rows, ncols=5, figsize=(25, 20))  # the figure size to better fit 5 columns
axes = axes.flatten()  # Flatten the axes array to simplify indexing

for i, img_path in enumerate(image_files):
    img = Image.open(img_path)  # Open the image
    ax = axes[i]  # Get the appropriate subplot
    ax.imshow(img, cmap='gray')  # Display the image
    ax.axis('off')  # Turn off the axis
    ax.set_title(f'Image: {os.path.basename(img_path)}', fontsize=20)  # Set title with the image filename

# Handle any unused axes
for j in range(i + 1, len(axes)):
    axes[j].axis('off')  # Turn off the axis for unused subplot
```

```
plt.tight_layout()
plt.show()
```

Image: A_test.jpg

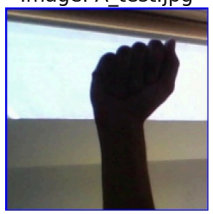


Image: B_test.jpg

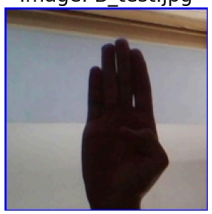


Image: C_test.jpg

Image: D_test.jpg

Image: E_test.jpg

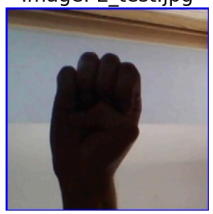


Image: F_test.jpg

Image: G_test.jpg

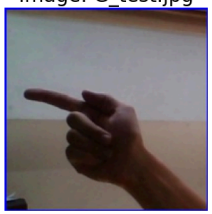


Image: H_test.jpg

Image: I_test.jpg

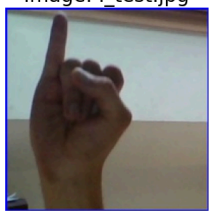


Image: J_test.jpg

Image: K_test.jpg

Image: L_test.jpg

Image: M_test.jpg

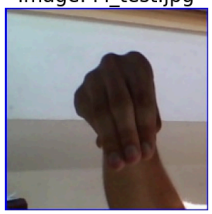


Image: nothing_test.jpg

Image: N_test.jpg

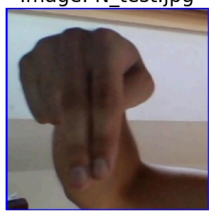


Image: O_test.jpg

Image: P_test.jpg

Image: Q_test.jpg

Image: R_test.jpg

Image: space_test.jpg

Image: S_test.jpg

Image: T_test.jpg

Image: U_test.jpg

Image: V_test.jpg

Image: W_test.jpg

Image: X_test.jpg

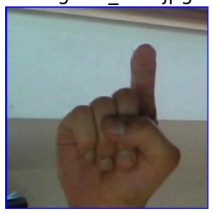


Image: Y_test.jpg

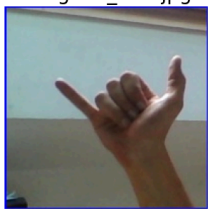


Image: Z_test.jpg

Model Predictions on Test data

In [ ]:
```
# train_images has already been created  above
class_labels = train_images.class_indices
print("Class labels and indices:", class_labels)

# To invert the dictionary for use in predictions (index to label mapping):
index_to_label = {v: k for k, v in class_labels.items()}
print("Index to label mapping:", index_to_label)
```

```
Class labels and indices: {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7, 'I': 8, 'J': 9, 'K': 10,
'L': 11, 'M': 12, 'N': 13, 'O': 14, 'P': 15, 'Q': 16, 'R': 17, 'S': 18, 'T': 19, 'U': 20, 'V': 21, 'W': 22, 'X': 23,
'Y': 24, 'Z': 25, 'del': 26, 'nothing': 27, 'space': 28}
Index to label mapping: {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J', 10: 'K', 11:
'L', 12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q', 17: 'R', 18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W', 23: 'X', 24:
'Y', 25: 'Z', 26: 'del', 27: 'nothing', 28: 'space'}
```

In [ ]:
```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import numpy as np
import os

# Load the model
model = load_model("best_model_3.h5")

# 'train_images' is  training data generator and extract class labels from the generator
class_labels = {v: k for k, v in train_images.class_indices.items()}

# Predict function
def predict_image(image_path, model, class_labels):
    try:
        img = load_img(image_path, target_size=(64, 64), color_mode='grayscale')
        img_array = img_to_array(img)
        img_array = img_array / 255.0
        img_array = np.expand_dims(img_array, axis=0)

        predictions = model.predict(img_array)
        predicted_class_index = np.argmax(predictions, axis=1)[0]
        predicted_class = class_labels[predicted_class_index]
        return predicted_class
```

```
    except Exception as e:
        print(f"Error processing image {image_path}: {e}")
        return "Error"



# List only image files (avoiding possible errors with non-image files)
image_files = [os.path.join(test_data_dir, f) for f in os.listdir(test_data_dir) if os.path.isfile(os.path.join(test_

# Make predictions on each image file
for img_path in image_files:
    predicted_class = predict_image(img_path, model, class_labels)
    print(f"Image: {os.path.basename(img_path)}, Predicted Class: {predicted_class}")
```

```
Image: A_test.jpg, Predicted Class: A
Image: B_test.jpg, Predicted Class: B
Image: C_test.jpg, Predicted Class: C
Image: D_test.jpg, Predicted Class: D
Image: E_test.jpg, Predicted Class: E
Image: F_test.jpg, Predicted Class: F
Image: G_test.jpg, Predicted Class: G
Image: H_test.jpg, Predicted Class: H
Image: I_test.jpg, Predicted Class: I
Image: J_test.jpg, Predicted Class: J
Image: K_test.jpg, Predicted Class: K
Image: L_test.jpg, Predicted Class: L
Image: M_test.jpg, Predicted Class: M
Image: nothing_test.jpg, Predicted Class: nothing
Image: N_test.jpg, Predicted Class: N
Image: O_test.jpg, Predicted Class: O
Image: P_test.jpg, Predicted Class: P
Image: Q_test.jpg, Predicted Class: Q
Image: R_test.jpg, Predicted Class: R
Image: space_test.jpg, Predicted Class: space
Image: S_test.jpg, Predicted Class: S
Image: T_test.jpg, Predicted Class: T
Image: U_test.jpg, Predicted Class: U
Image: V_test.jpg, Predicted Class: V
Image: W_test.jpg, Predicted Class: W
Image: X_test.jpg, Predicted Class: X
Image: Y_test.jpg, Predicted Class: Y
Image: Z_test.jpg, Predicted Class: Z
```