In [4]:

```
import os
import csv
import cv2
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, export_graphvi
from sklearn.neural_network import MLPClassifier

# Setting the path of the training dataset (that was already pr

running_local = True if os.getenv('JUPYTERHUB_USER') is None el
DATASET_PATH = "."

# Set the location of the dataset
if running_local:
    # If running on your local machine, the sign_lang_train fol
    local_path = "sign_lang_train"
    if os.path.exists(local_path):
        DATASET_PATH = local_path
else:
    # If running on the Jupyter hub, this data folder is alread
    # You DO NOT need to upload the data!
        DATASET_PATH = "/data/mlproject21/sign_lang_train"

# Setting the path of the training dataset (that was already pr

running_local = True if os.getenv('JUPYTERHUB_USER') is None el
DATASET_PATH = "."

# Set the location of the dataset
if running_local:
    # If running on your local machine, the sign_lang_train fol
    local_path = "sign_lang_train"
    if os.path.exists(local_path):
        DATASET_PATH = local_path
else:
    # If running on the Jupyter hub, this data folder is alread
    # You DO NOT need to upload the data!
        DATASET_PATH = "/data/mlproject21/sign_lang_train"

# Utility function

def read_csv(csv_file):
    with open(csv_file, newline='') as f:
        reader = csv.reader(f)
        data = list(reader)
    return data
```

In [5]:

```
import torch
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision import transforms, utils, io
from torchvision.utils import make_grid

from string import ascii_lowercase

class SignLangDataset(Dataset):
    """Sign language dataset"""

    def __init__(self, csv_file, root_dir, class_index_map=None
        """
        Args:
            csv_file (string): Path to the csv file with annota
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform
        """
        self.data = read_csv(os.path.join(root_dir,csv_file))
        self.root_dir = root_dir
        self.class_index_map = class_index_map
        self.transform = transform
        # List of class names in order
        self.class_names = list(map(str, list(range(10)))) + li

    def __len__(self):
        """
        Calculates the length of the dataset-
        """
        return len(self.data)

    def __getitem__(self, idx):
        """
        Returns one sample (dict consisting of an image and its
        """
        if torch.is_tensor(idx):
            idx = idx.tolist()

        # Read the image and labels
        image_path = os.path.join(self.root_dir, self.data[idx]
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        # Shape of the image should be H,W,C where C=1
        image = np.expand_dims(image, 0)
        # The label is the index of the class name in the list
        # because we should have integer labels in the range 0-
        label = self.class_names.index(self.data[idx][0])

        sample = {'image': image, 'label': label}

        if self.transform:
            sample = self.transform(sample)

        return sample
```

In [6]:
```
sign_lang_dataset = SignLangDataset(csv_file="labels.csv", root
#print(sign_lang_dataset[1]['label'])
data_len = len(sign_lang_dataset)
train_ratio = 0.8
train_size = int(train_ratio * data_len)
val_size = data_len - train_size
train_dataset, val_dataset = random_split(sign_lang_dataset, [t
```

In [7]:
```
def build_data_for_scikit(dataset):
    X_list = list()
    y_list = list()
    for data in dataset:
        X_list.append(data['image'])
        y_list.append(data['label'])
    return np.array(X_list), np.array(y_list)
```

In [8]:
```
X_train, y_train = build_data_for_scikit(train_dataset)
X_train = X_train.reshape(train_size, -1)
X_test, y_test = build_data_for_scikit(val_dataset)
X_test = X_test.reshape(val_size, -1)
print(X_train.shape)
print(X_test.shape)
```

```
(7744, 16384)
(1936, 16384)
```

In [9]:
```
#experimental
clf = DecisionTreeClassifier(criterion='gini')
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

```

In [10]:
```
clfs = []
for i in range(200, len(ccp_alphas), 300):
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_
    clf.fit(X_train, y_train)
    clfs.append(clf)
print("Number of nodes in the last tree is: {} with ccp_alpha:
      clfs[-1].tree_.node_count, ccp_alphas[-1]))
```

```
Number of nodes in the last tree is: 291 with ccp_alpha: 0.0144829
25279712844
```

In [13]:
```
print(clfs)
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]
print(train_scores)
print(test_scores)
```

```
[DecisionTreeClassifier(ccp_alpha=0.00012913223140495868, random_s
tate=0), DecisionTreeClassifier(ccp_alpha=0.00023243801652892576,
random_state=0), DecisionTreeClassifier(ccp_alpha=0.00032283057851
23967, random_state=0), DecisionTreeClassifier(ccp_alpha=0.0004958
677685950414, random_state=0), DecisionTreeClassifier(ccp_alpha=0.
0010433475320776988, random_state=0)]
[0.8929493801652892, 0.8607954545454546, 0.7957128099173554, 0.700
4132231404959, 0.49599690082644626]
[0.37964876033057854, 0.38016528925619836, 0.37706611570247933, 0.
381198347107438, 0.3517561983471074]
```

In [44]:
```
tree_sklearn = DecisionTreeClassifier(criterion='gini')
tree_sklearn.fit(X_train, y_train)
```

Out[44]:
```
DecisionTreeClassifier()
```

In [46]:
```
accuracy_train = tree_sklearn.score(X_train, y_train)
accuracy_test = tree_sklearn.score(X_test, y_test)
print(accuracy_train)
print(accuracy_test)
```

```
1.0
0.368801652892562
```

In [ ]:
```

```