base-12

September 21, 2021

[1]:

```
base_transfer_set = ['01', '02', '04', '05', '08', '09', '12', '13', '16',␣
 ↪'17', '18', '20']
target_transfer_set = ['03', '06', '07', '10', '11', '14', '15', '19']

import random
def random_combination(iterable, r):
    "Random selection from itertools.combinations(iterable, r)"
    pool = tuple(iterable)
    n = len(pool)
    indices = sorted(random.sample(range(n), r))
    return tuple(pool[i] for i in indices)

transfers_size_6 = []
for i in range(4):
    transfers_size_6.append(random_combination(target_transfer_set, 6))
print(transfers_size_6)
transfers_size_6 = [('03', '06', '07', '10', '11', '14'), ('03', '06', '07',␣
 ↪'10', '14', '15'), ('03', '06', '07', '10', '14', '15'), ('03', '07', '10',␣
 ↪'14', '15', '19')]
for i, tmp in enumerate(transfers_size_6):
    transfers_size_6[i] = list(transfers_size_6[i])
print(transfers_size_6)

transfers_size_4 = []
for i in range(4):
    transfers_size_4.append(random_combination(target_transfer_set, 4))
print(transfers_size_4)
transfers_size_4 = [('06', '10', '14', '15'), ('03', '10', '14', '19'), ('03',␣
 ↪'06', '10', '15'), ('03', '07', '10', '15')]
for i, tmp in enumerate(transfers_size_4):
    transfers_size_4[i] = list(transfers_size_4[i])
print(transfers_size_4)

transfers_size_3 = []
for i in range(4):
    transfers_size_3.append(random_combination(target_transfer_set, 3))
print(transfers_size_3)
```

```
transfers_size_3 = [('07', '11', '14'), ('06', '07', '10'), ('03', '15', '19'),␣
 ↪('06', '14', '19')]
for i, tmp in enumerate(transfers_size_3):
    transfers_size_3[i] = list(transfers_size_3[i])
print(transfers_size_3)

transfers_size_2 = []
for i in range(4):
    transfers_size_2.append(random_combination(target_transfer_set, 2))
print(transfers_size_2)
transfers_size_2 = [('06', '10'), ('07', '11'), ('06', '15'), ('14', '15')]
for i, tmp in enumerate(transfers_size_2):
    transfers_size_2[i] = list(transfers_size_2[i])
print(transfers_size_2)
```

```
[('03', '06', '07', '10', '15', '19'), ('06', '07', '10', '14', '15', '19'),
('03', '06', '07', '10', '11', '14'), ('07', '10', '11', '14', '15', '19')]
[['03', '06', '07', '10', '11', '14'], ['03', '06', '07', '10', '14', '15'],
['03', '06', '07', '10', '14', '15'], ['03', '07', '10', '14', '15', '19']]
[('03', '06', '14', '19'), ('03', '06', '10', '11'), ('03', '06', '07', '11'),
('03', '07', '11', '14')]
[['06', '10', '14', '15'], ['03', '10', '14', '19'], ['03', '06', '10', '15'],
['03', '07', '10', '15']]
[('03', '14', '15'), ('03', '14', '15'), ('10', '11', '14'), ('03', '06', '10')]
[['07', '11', '14'], ['06', '07', '10'], ['03', '15', '19'], ['06', '14', '19']]
[('06', '10'), ('10', '14'), ('10', '11'), ('10', '14')]
[['06', '10'], ['07', '11'], ['06', '15'], ['14', '15']]
```

[2]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

def make_confusion_matrix(cf,
                          group_names=None,
                          categories='auto',
                          count=True,
                          percent=True,
                          cbar=True,
                          xyticks=True,
                          xyplotlabels=True,
                          sum_stats=True,
                          figsize=None,
                          cmap='Blues',
                          title=None):
    '''
    This function will make a pretty plot of an sklearn Confusion Matrix cm␣
 ↪using a Seaborn heatmap visualization.
```

```
    Arguments
    ---------
    cf:            confusion matrix to be passed in
    group_names:   List of strings that represent the labels row by row to be␣
↪shown in each square.
    categories:    List of strings containing the categories to be displayed on␣
↪the x,y axis. Default is 'auto'
    count:         If True, show the raw number in the confusion matrix.␣
↪Default is True.
    normalize:     If True, show the proportions for each category. Default is␣
↪True.
    cbar:          If True, show the color bar. The cbar values are based off␣
↪the values in the confusion matrix.
                   Default is True.
    xyticks:       If True, show x and y ticks. Default is True.
    xyplotlabels:  If True, show 'True Label' and 'Predicted Label' on the␣
↪figure. Default is True.
    sum_stats:     If True, display summary statistics below the figure.␣
↪Default is True.
    figsize:       Tuple representing the figure size. Default will be the␣
↪matplotlib rcParams value.
    cmap:          Colormap of the values displayed from matplotlib.pyplot.cm.␣
↪Default is 'Blues'
                   See http://matplotlib.org/examples/color/colormaps_reference.
↪html

    title:         Title for the heatmap. Default is None.
    '''


    # CODE TO GENERATE TEXT INSIDE EACH SQUARE
    blanks = ['' for i in range(cf.size)]

    if group_names and len(group_names)==cf.size:
        group_labels = ["{}\n".format(value) for value in group_names]
    else:
        group_labels = blanks

    if count:
        group_counts = ["{0:0.0f}\n".format(value) for value in cf.flatten()]
    else:
        group_counts = blanks

    if percent:
        group_percentages = ["{0:.2%}".format(value) for value in cf.flatten()/
↪np.sum(cf)]
```

```
    else:
        group_percentages = blanks

    box_labels = [f"{v1}{v2}{v3}".strip() for v1, v2, v3 in␣
↪zip(group_labels,group_counts,group_percentages)]
    box_labels = np.asarray(box_labels).reshape(cf.shape[0],cf.shape[1])


    # CODE TO GENERATE SUMMARY STATISTICS & TEXT FOR SUMMARY STATS
    if sum_stats:
        #Accuracy is sum of diagonal divided by total observations
        accuracy  = np.trace(cf) / float(np.sum(cf))

        #if it is a binary confusion matrix, show some more stats
        if len(cf)==2:
            #Metrics for Binary Confusion Matrices
            precision = cf[1,1] / sum(cf[:,1])
            recall    = cf[1,1] / sum(cf[1,:])
            f1_score  = 2*precision*recall / (precision + recall)
            stats_text = "\n\nAccuracy={:0.3f}\nPrecision={:0.3f}\nRecall={:0.
↪3f}\nF1 Score={:0.3f}".format(
                accuracy,precision,recall,f1_score)
        else:
            stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
    else:
        stats_text = ""


    # SET FIGURE PARAMETERS ACCORDING TO OTHER ARGUMENTS
    if figsize==None:
        #Get default figure size if not set
        figsize = plt.rcParams.get('figure.figsize')

    if xyticks==False:
        #Do not show categories if xyticks is False
        categories=False


    # MAKE THE HEATMAP VISUALIZATION
    plt.figure(figsize=figsize)
    sns.
↪heatmap(cf,annot=box_labels,fmt="",cmap=cmap,cbar=cbar,xticklabels=categories,yticklabels=c

    if xyplotlabels:
        plt.ylabel('True label')
        plt.xlabel('Predicted label' + stats_text)
    else:
        plt.xlabel(stats_text)

    if title:
        plt.title(title)
```

[3]:

```
import os
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

def create_best_model(gesture_subset):
    gesture_subset.sort()
    print("Loadind Dataset for gestures: ", gesture_subset)
    path = 'gestures-dataset'
    dataset = None

    samples = 0
    for subject in os.listdir(path):
        if os.path.isfile(os.path.join(path, subject)):
            continue
        if subject in ('U01', 'U02', 'U03', 'U04', 'U05', 'U06', 'U07', 'U08'):
            for gesture in os.listdir(os.path.join(path, subject)):
                if os.path.isfile(os.path.join(path, subject, gesture)):
                    continue
                gesture = str(gesture)
                if gesture not in gesture_subset:
                    continue
                for samplefile in os.listdir(os.path.join(path, subject,␣
 ↪gesture)):
                    if os.path.isfile(os.path.join(path, subject, gesture,␣
 ↪samplefile)):
                        df = pd.read_csv(os.path.join(path, subject, gesture,␣
 ↪samplefile), \
                            sep = ' ', \
                            names = ['System.currentTimeMillis()', \
                            'System.nanoTime()', \
                            'sample.timestamp', \
                            'X', \
                            'Y', \
                            'Z' \
                            ])
                        df = df[["sample.timestamp", "X", "Y", "Z"]]

                        start = df["sample.timestamp"][0]
                        df["sample.timestamp"] -= start
                        df["sample.timestamp"] /= 10000000
                        df["subject"] = subject
                        df["gesture"] = gesture
                        df["sample"] = str(samplefile[:-4])
                        samples += 1
                        #print(df)
                        if dataset is None:
                            dataset = df.copy()
                        else:
                            dataset = pd.concat([dataset, df])

    dataset = dataset.sort_values(by=['gesture','subject','sample','sample.
↪timestamp'])
    data = dataset
    print(str(samples) + " samples loaded")

    print("Scaling Dataset for gestures: ", gesture_subset)
    from sklearn.preprocessing import StandardScaler

    scaler = StandardScaler()
    dataset_scaled = None

    samples = 0
    for i, gesture in enumerate(gesture_subset):
        df_gesture=data[data['gesture']==gesture]
        for j, subject in enumerate(df_gesture['subject'].unique()):
            df_subject=df_gesture[df_gesture['subject']==subject]
            for k, sample in enumerate(df_subject['sample'].unique()):
                df_sample=df_subject[df_subject['sample']==sample].copy()
                df_sample.sort_values(by=['sample.timestamp'])

                sc = scaler
                sc = sc.fit_transform(df_sample[["X", "Y", "Z"]])
                sc = pd.DataFrame(data=sc, columns=["X", "Y", "Z"])
                df_sample['X'] = sc['X']
                df_sample['Y'] = sc['Y']
                df_sample['Z'] = sc['Z']
                if dataset_scaled is None:
                    dataset_scaled = df_sample.copy()
                else:
                    dataset_scaled = pd.concat([dataset_scaled, df_sample])
                samples += 1
    print(str(samples) + " samples scaled")
    data = dataset_scaled

    print("Cleaning Dataset for gestures: ", gesture_subset)
    dataset_outliers = None
    dataset_cleaned = None
    samples = 0
    outliers = 0
    for i, gesture in enumerate(gesture_subset):
        df_gesture = data[data['gesture']==gesture]
        for j, subject in enumerate(df_gesture['subject'].unique()):
            df_subject = df_gesture[df_gesture['subject']==subject]

            time_mean = df_subject.groupby(["gesture","subject", "sample"]).
↪count().groupby(["gesture","subject"]).agg({'sample.timestamp': ['mean']})
            time_std = df_subject.groupby(["gesture","subject", "sample"]).
↪count().groupby(["gesture","subject"]).agg({'sample.timestamp': ['std']})
            time_max = time_mean['sample.timestamp'].iloc[0]['mean'] + 1.0 *␣
↪time_std['sample.timestamp'].iloc[0]['std']
            time_min = time_mean['sample.timestamp'].iloc[0]['mean'] - 1.0 *␣
↪time_std['sample.timestamp'].iloc[0]['std']
            for k, sample in enumerate(df_subject['sample'].unique()):
                df_sample=df_subject[df_subject['sample']==sample]
                df_sample_count = df_sample.count()['sample.timestamp']
                if df_sample_count < time_min or df_sample_count > time_max:
                    if dataset_outliers is None:
                        dataset_outliers = df_sample.copy()
                    else:
                        dataset_outliers = pd.concat([dataset_outliers,␣
↪df_sample])
                    outliers += 1
                else:
                    if dataset_cleaned is None:
                        dataset_cleaned = df_sample.copy()
                    else:
                        dataset_cleaned = pd.concat([dataset_cleaned,␣
↪df_sample])
                    samples += 1
    print(str(samples) + " samples cleaned")
    print(str(outliers) + " samples outliers")
    data = dataset_cleaned

    print("Time slicing Cleaned Dataset for gestures: ", gesture_subset)
    dataset_timecut = None
    samples = 0
    damaged = 0
    for i, gesture in enumerate(data['gesture'].unique()):
        df_gesture = data[data['gesture']==gesture]
        for j, subject in enumerate(df_gesture['subject'].unique()):
            df_subject = df_gesture[df_gesture['subject']==subject]
            time_max = 19 # 18 * 11 = 198
            for i, sample in enumerate(df_subject['sample'].unique()):
                df_sample = df_subject[df_subject['sample']==sample]
                df_sample_count = df_sample.count()['sample.timestamp']
                #print(df_sample_count)
                if df_sample_count >= time_max:
                    df_sample = df_sample[df_sample['sample.timestamp'] <= (11␣
↪* (time_max-1))]
                    df_sample_count = df_sample.count()['sample.timestamp']
                    #print(df_sample_count)
                elif df_sample_count < time_max:
                    for tmp in range(df_sample_count * 11, (time_max) * 11, 11):
                        df = pd.DataFrame([[tmp, 0.0, 0.0, 0.0, gesture,␣
↪subject, sample]], columns=['sample.timestamp', 'X', 'Y', 'Z', 'gesture',␣
↪'subject', 'sample'])
                        df_sample = df_sample.append(df, ignore_index=True)
                #print(df_sample)
                df_sample_count = df_sample.count()['sample.timestamp']
                #print(df_sample_count)
                if df_sample_count != time_max:
                    damaged += 1
                    continue
                if dataset_timecut is None:
                    dataset_timecut = df_sample.copy()
                else:
                    dataset_timecut = pd.concat([dataset_timecut, df_sample])
                samples += 1

    dataset_cleaned = dataset_timecut
    print(str(samples) + " cleaned samples sliced")
    print(str(damaged) + " cleaned samples damaged")

    data = dataset_outliers
    print("Time slicing Outliers Dataset for gestures: ", gesture_subset)
    dataset_timecut = None
    samples = 0
    damaged = 0
    for i, gesture in enumerate(data['gesture'].unique()):
        df_gesture = data[data['gesture']==gesture]
        for j, subject in enumerate(df_gesture['subject'].unique()):
            df_subject = df_gesture[df_gesture['subject']==subject]
            time_max = 19 # 18 * 11 = 198
            for i, sample in enumerate(df_subject['sample'].unique()):
                df_sample = df_subject[df_subject['sample']==sample]
                df_sample_count = df_sample.count()['sample.timestamp']
                #print(df_sample_count)
                if df_sample_count >= time_max:
                    df_sample = df_sample[df_sample['sample.timestamp'] <= (11␣
↪* (time_max-1))]
```

```
                    df_sample_count = df_sample.count()['sample.timestamp']
                    #print(df_sample_count)
                elif df_sample_count < time_max:
                    for tmp in range(df_sample_count * 11, (time_max) * 11, 11):
                        df = pd.DataFrame([[tmp, 0.0, 0.0, 0.0, gesture,␣
↪subject, sample]], columns=['sample.timestamp', 'X', 'Y', 'Z', 'gesture',␣
↪'subject', 'sample'])
                        df_sample = df_sample.append(df, ignore_index=True)
                #print(df_sample)
                df_sample_count = df_sample.count()['sample.timestamp']
                #print(df_sample_count)
                if df_sample_count != time_max:
                    damaged += 1
                    continue
                if dataset_timecut is None:
                    dataset_timecut = df_sample.copy()
                else:
                    dataset_timecut = pd.concat([dataset_timecut, df_sample])
                samples += 1

    dataset_outliers = dataset_timecut
    print(str(samples) + " outliers samples sliced")
    print(str(damaged) + " outliers samples damaged")

    data = dataset_cleaned

    from keras.models import Sequential
    from keras.layers import Bidirectional
    from keras.layers import LSTM
    from keras.layers import Dense
    from keras.layers import Dropout
    from keras.optimizers import adam_v2
    from keras.wrappers.scikit_learn import KerasClassifier
#     from scikeras.wrappers import KerasClassifier
    from sklearn.model_selection import StratifiedGroupKFold
    from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import GridSearchCV
    from keras.utils import np_utils
    from sklearn.preprocessing import LabelEncoder
    from sklearn.pipeline import Pipeline
    import numpy as np

    # fix random seed for reproducibility
    seed = 1000
    np.random.seed(seed)
    # create the dataset
    def get_dataset(data):
```

```
    X_train = []
    Y_train = []
    groups = []
    for i, gesture in enumerate(data['gesture'].unique()):
        df_gesture = data[data['gesture']==gesture]
        for j, subject in enumerate(df_gesture['subject'].unique()):
            df_subject = df_gesture[df_gesture['subject']==subject]
            for k, sample in enumerate(df_subject['sample'].unique()):
                df_sample = df_subject[df_subject['sample']==sample]
                accel_vector = []
                for index, row in df_sample.sort_values(by='sample.
↪timestamp').iterrows():
                    accel_vector.append([row['X'],row['Y'],row['Z']])
                accel_vector = np.asarray(accel_vector)
                X_train.append(accel_vector)
                Y_train.append(gesture)
                groups.append(subject)
    X_train = np.asarray(X_train)
    Y_train = LabelEncoder().fit_transform(Y_train)
    #print(Y_train)
    return X_train, Y_train, groups

# Function to create model, required for KerasClassifier
def create_model(dropout_rate=0.8, units=128, optimizer=adam_v2.
↪Adam(learning_rate=0.001)):
    model = Sequential()
    model.add(
        Bidirectional(
            LSTM(
                units=units,
                input_shape=[19, 3]
            )
        )
    )
    model.add(Dropout(rate=dropout_rate))
    model.add(Dense(units=units, activation='relu'))
    model.add(Dense(len(gesture_subset), activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy',␣
↪optimizer=optimizer, metrics=['accuracy'])
    #print(model.summary())
    return model

model = KerasClassifier(build_fn=create_model, verbose=0)
cv = StratifiedGroupKFold(n_splits=5, shuffle=True, random_state=1000)
# get the dataset
X, y, g = get_dataset(dataset_cleaned)
#cv = cv.split(X, y, g)
```

```
    batch_size = [19]
    epochs = [64, 128]
    #epochs = [128]
    units = [32,64,128]
#    units = [16]
    dropout_rate = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
#    dropout_rate = [0.5]
    param_grid = dict(epochs=epochs, units=units, batch_size=batch_size,␣
↪dropout_rate=dropout_rate)
    print("Hyperparameter tunning started for Dataset for gestures: ",␣
↪gesture_subset)
    grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1,␣
↪cv=cv, verbose=1)
    grid_result = grid.fit(X, y, groups=g)
    # summarize results
    print("Best: %f using %s" % (grid_result.best_score_, grid_result.
↪best_params_))
    means = grid_result.cv_results_['mean_test_score']
    stds = grid_result.cv_results_['std_test_score']
    train_mean = grid_result.cv_results_['mean_fit_time']
    train_std = grid_result.cv_results_['std_fit_time']
    score_mean = grid_result.cv_results_['mean_score_time']
    score_std = grid_result.cv_results_['std_score_time']
    params = grid_result.cv_results_['params']
    for mean, stdev, train_mean, train_std, score_mean, score_std, param in␣
↪zip(means, stds, train_mean, train_std, score_mean, score_std, params):
        print("accuracy: %f (%f) train time: %f (%f) score time: %f (%f) with:␣
↪%r" % (mean, stdev, train_mean, train_std, score_mean, score_std, param))
    print("Hyperparameter tunning completed for Dataset: ", gesture_subset)


    model = grid_result.best_estimator_
    import pickle

    def save_model(model, gesture_subset):
        gesture_subset.sort()
        name = '-'.join(gesture_subset)
        # saving model
        pickle.dump(model.classes_, open(name + '_model_classes.pkl','wb'))
        model.model.save(name + '_lstm')
    print("Saving model to disk started for Dataset gestures: ", gesture_subset)
    save_model(model, gesture_subset)
    print("Saving model to disk completed for Dataset gestures: ",␣
↪gesture_subset)

    import tensorflow as tf
    def load_model(gesture_subset):
        gesture_subset.sort()
        name = '-'.join(gesture_subset)
        # loading model
        build_model = lambda: tf.keras.models.load_model(name + '_lstm')
        classifier = KerasClassifier(build_fn=build_model, epochs=1,␣
 ↪batch_size=10, verbose=0)
        classifier.classes_ = pickle.load(open(name + '_model_classes.
 ↪pkl','rb'))
        classifier.model = build_model()
        return classifier
    print("Loading model to disk started for Dataset gestures: ",␣
 ↪gesture_subset)
    model = load_model(gesture_subset)
    #print(model.model.summary())
    print("Loading model to disk completed for Dataset gestures: ",␣
 ↪gesture_subset)

    print("Testing model against outliers for Dataset gestures: ",␣
 ↪gesture_subset)
    data = dataset_outliers
    X, y, g = get_dataset(dataset_outliers)
    y_pred = model.predict(X)
    #print(y)
    #print(y_pred)

    from sklearn.metrics import classification_report
    print(classification_report(y, y_pred, target_names=gesture_subset))

    from sklearn.metrics import confusion_matrix
    cf_matrix = confusion_matrix(y, y_pred)
    make_confusion_matrix(cf_matrix, categories=gesture_subset, figsize=[8,8])
    return grid_result
base_transfer_set = ['01', '02', '04', '05', '08', '09', '12', '13', '16',␣
 ↪'17', '18', '20']
dataset = base_transfer_set

results = create_best_model(dataset)
```

```
Loadind Dataset for gestures:  ['01', '02', '04', '05', '08', '09', '12', '13',
'16', '17', '18', '20']
1942 samples loaded
Scaling Dataset for gestures:  ['01', '02', '04', '05', '08', '09', '12', '13',
'16', '17', '18', '20']
1942 samples scaled
Cleaning Dataset for gestures:  ['01', '02', '04', '05', '08', '09', '12', '13',
'16', '17', '18', '20']
```

```
1493 samples cleaned
449 samples outliers
Time slicing Cleaned Dataset for gestures:  ['01', '02', '04', '05', '08', '09',
'12', '13', '16', '17', '18', '20']
1493 cleaned samples sliced
0 cleaned samples damaged
Time slicing Outliers Dataset for gestures:  ['01', '02', '04', '05', '08',
'09', '12', '13', '16', '17', '18', '20']
446 outliers samples sliced
3 outliers samples damaged
Hyperparameter tunning started for Dataset for gestures:  ['01', '02', '04',
'05', '08', '09', '12', '13', '16', '17', '18', '20']
Fitting 5 folds for each of 54 candidates, totalling 270 fits
```

[ ]:

```python
import os
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

baseset = base_transfer_set

def evaluate_model(baseset):
    print("Baseset: ", baseset)
    print("Loadind Dataset: ", baseset)
    path = 'gestures-dataset'
    dataset = None

    samples = 0
    for subject in os.listdir(path):
        if os.path.isfile(os.path.join(path, subject)):
            continue
        if subject in ('U01', 'U02', 'U03', 'U04', 'U05', 'U06', 'U07', 'U08'):
            for gesture in os.listdir(os.path.join(path, subject)):
                if os.path.isfile(os.path.join(path, subject, gesture)):
                    continue
                gesture = str(gesture)
                if gesture not in baseset:
                    continue
                for samplefile in os.listdir(os.path.join(path, subject,␣
 ↪gesture)):
                    if os.path.isfile(os.path.join(path, subject, gesture,␣
 ↪samplefile)):
                        df = pd.read_csv(os.path.join(path, subject, gesture,␣
 ↪samplefile), \
                            sep = ' ', \
                            names = ['System.currentTimeMillis()', \
                            'System.nanoTime()', \
                            'sample.timestamp', \
                            'X', \
                            'Y', \
                            'Z' \
                            ])
                        df = df[["sample.timestamp", "X", "Y", "Z"]]

                        start = df["sample.timestamp"][0]
                        df["sample.timestamp"] -= start
                        df["sample.timestamp"] /= 10000000
                        df["subject"] = subject
                        df["gesture"] = gesture
                        df["sample"] = str(samplefile[:-4])
                        samples += 1
                        #print(df)
                        if dataset is None:
                            dataset = df.copy()
                        else:
                            dataset = pd.concat([dataset, df])

    dataset = dataset.sort_values(by=['gesture','subject','sample','sample.
↪timestamp'])
    data = dataset
    print(str(samples) + " samples loaded")

    print("Scaling Dataset: ", baseset)
    from sklearn.preprocessing import StandardScaler

    scaler = StandardScaler()
    dataset_scaled = None

    samples = 0
    for i, gesture in enumerate(baseset):
        df_gesture=data[data['gesture']==gesture]
        for j, subject in enumerate(df_gesture['subject'].unique()):
            df_subject=df_gesture[df_gesture['subject']==subject]
            for k, sample in enumerate(df_subject['sample'].unique()):
                df_sample=df_subject[df_subject['sample']==sample].copy()
                df_sample.sort_values(by=['sample.timestamp'])

                sc = scaler
                sc = sc.fit_transform(df_sample[["X", "Y", "Z"]])
                sc = pd.DataFrame(data=sc, columns=["X", "Y", "Z"])
                df_sample['X'] = sc['X']
                df_sample['Y'] = sc['Y']
                df_sample['Z'] = sc['Z']
                if dataset_scaled is None:
                    dataset_scaled = df_sample.copy()
                else:
                    dataset_scaled = pd.concat([dataset_scaled, df_sample])
                samples += 1
    print(str(samples) + " samples scaled")
    data = dataset_scaled

    print("Cleaning Dataset: ", baseset)
    dataset_outliers = None
    dataset_cleaned = None

    samples = 0
    outliers = 0
    for i, gesture in enumerate(baseset):
        df_gesture = data[data['gesture']==gesture]
        for j, subject in enumerate(df_gesture['subject'].unique()):
            df_subject = df_gesture[df_gesture['subject']==subject]

            time_mean = df_subject.groupby(["gesture","subject", "sample"]).
↪count().groupby(["gesture","subject"]).agg({'sample.timestamp': ['mean']})
            time_std = df_subject.groupby(["gesture","subject", "sample"]).
↪count().groupby(["gesture","subject"]).agg({'sample.timestamp': ['std']})
            time_max = time_mean['sample.timestamp'].iloc[0]['mean'] + 1.0 *␣
↪time_std['sample.timestamp'].iloc[0]['std']
            time_min = time_mean['sample.timestamp'].iloc[0]['mean'] - 1.0 *␣
↪time_std['sample.timestamp'].iloc[0]['std']
            for k, sample in enumerate(df_subject['sample'].unique()):
                df_sample=df_subject[df_subject['sample']==sample]
                df_sample_count = df_sample.count()['sample.timestamp']
                if df_sample_count < time_min or df_sample_count > time_max:
                    if dataset_outliers is None:
                        dataset_outliers = df_sample.copy()
                    else:
                        dataset_outliers = pd.concat([dataset_outliers,␣
↪df_sample])
                    outliers += 1
                else:
                    if dataset_cleaned is None:
                        dataset_cleaned = df_sample.copy()
                    else:
                        dataset_cleaned = pd.concat([dataset_cleaned,␣
↪df_sample])
                    samples += 1
    print(str(samples) + " samples cleaned")
    print(str(outliers) + " samples outliers")
    data = dataset_cleaned
```

```
print("Time slicing Cleaned Dataset: ", baseset)
dataset_timecut = None
samples = 0
damaged = 0
for i, gesture in enumerate(data['gesture'].unique()):
    df_gesture = data[data['gesture']==gesture]
    for j, subject in enumerate(df_gesture['subject'].unique()):
        df_subject = df_gesture[df_gesture['subject']==subject]
        time_max = 19 # 18 * 11 = 198
        for i, sample in enumerate(df_subject['sample'].unique()):
            df_sample = df_subject[df_subject['sample']==sample]
            df_sample_count = df_sample.count()['sample.timestamp']
            #print(df_sample_count)
            if df_sample_count >= time_max:
                df_sample = df_sample[df_sample['sample.timestamp'] <= (11␣
↪* (time_max-1))]
                df_sample_count = df_sample.count()['sample.timestamp']
                #print(df_sample_count)
            elif df_sample_count < time_max:
                for tmp in range(df_sample_count * 11, (time_max) * 11, 11):
                    df = pd.DataFrame([[tmp, 0.0, 0.0, 0.0, gesture,␣
↪subject, sample]], columns=['sample.timestamp', 'X', 'Y', 'Z', 'gesture',␣
↪'subject', 'sample'])
                    df_sample = df_sample.append(df, ignore_index=True)
            #print(df_sample)
            df_sample_count = df_sample.count()['sample.timestamp']
            #print(df_sample_count)
            if df_sample_count != time_max:
                damaged += 1
                continue
            if dataset_timecut is None:
                dataset_timecut = df_sample.copy()
            else:
                dataset_timecut = pd.concat([dataset_timecut, df_sample])
            samples += 1

dataset_cleaned = dataset_timecut
print(str(samples) + " cleaned samples sliced")
print(str(damaged) + " cleaned samples damaged")

data = dataset_outliers
print("Time slicing Outliers Dataset: ", baseset)
dataset_timecut = None
samples = 0
damaged = 0
for i, gesture in enumerate(data['gesture'].unique()):
    df_gesture = data[data['gesture']==gesture]
    for j, subject in enumerate(df_gesture['subject'].unique()):
        df_subject = df_gesture[df_gesture['subject']==subject]
        time_max = 19 # 18 * 11 = 198
        for i, sample in enumerate(df_subject['sample'].unique()):
            df_sample = df_subject[df_subject['sample']==sample]
            df_sample_count = df_sample.count()['sample.timestamp']
            #print(df_sample_count)
            if df_sample_count >= time_max:
                df_sample = df_sample[df_sample['sample.timestamp'] <= (11␣
↪* (time_max-1))]
                df_sample_count = df_sample.count()['sample.timestamp']
                #print(df_sample_count)
            elif df_sample_count < time_max:
                for tmp in range(df_sample_count * 11, (time_max) * 11, 11):
                    df = pd.DataFrame([[tmp, 0.0, 0.0, 0.0, gesture,␣
↪subject, sample]], columns=['sample.timestamp', 'X', 'Y', 'Z', 'gesture',␣
↪'subject', 'sample'])
                    df_sample = df_sample.append(df, ignore_index=True)
            #print(df_sample)
            df_sample_count = df_sample.count()['sample.timestamp']
            #print(df_sample_count)
            if df_sample_count != time_max:
                damaged += 1
                continue
            if dataset_timecut is None:
                dataset_timecut = df_sample.copy()
            else:
                dataset_timecut = pd.concat([dataset_timecut, df_sample])
            samples += 1

dataset_outliers = dataset_timecut
print(str(samples) + " outliers samples sliced")
print(str(damaged) + " outliers samples damaged")

from keras import backend as K
data = dataset_cleaned
from keras.models import Sequential
from keras.layers import Bidirectional
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Dropout
from keras.optimizers import adam_v2
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedGroupKFold
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
import numpy as np
import tensorflow as tf

# fix random seed for reproducibility
seed = 1000
np.random.seed(seed)
# create the dataset
def get_dataset(data, index=[]):
    X_train = []
    Y_train = []
    groups = []
    samples_idx=0
    for i, gesture in enumerate(data['gesture'].unique()):
        df_gesture = data[data['gesture']==gesture]
        for j, subject in enumerate(df_gesture['subject'].unique()):
            df_subject = df_gesture[df_gesture['subject']==subject]
            for k, sample in enumerate(df_subject['sample'].unique()):
                df_sample = df_subject[df_subject['sample']==sample]
                accel_vector = []
                for idx, row in df_sample.sort_values(by='sample.
↪timestamp').iterrows():
                    accel_vector.append([row['X'],row['Y'],row['Z']])
                accel_vector = np.asarray(accel_vector)
                if len(index)==0:
                    X_train.append(accel_vector)
                    Y_train.append(gesture)
                    groups.append(subject)
                else:
                    if samples_idx in index:
                        X_train.append(accel_vector)
                        Y_train.append(gesture)
                        groups.append(subject)
                samples_idx+=1
    X_train = np.asarray(X_train)
    Y_train = LabelEncoder().fit_transform(Y_train)
    #print(Y_train)
    return X_train, Y_train, groups


def build_model(baseset):
    baseset.sort()
    basename = '-'.join(baseset)
    basemodel = tf.keras.models.load_model(basename + '_lstm')
```

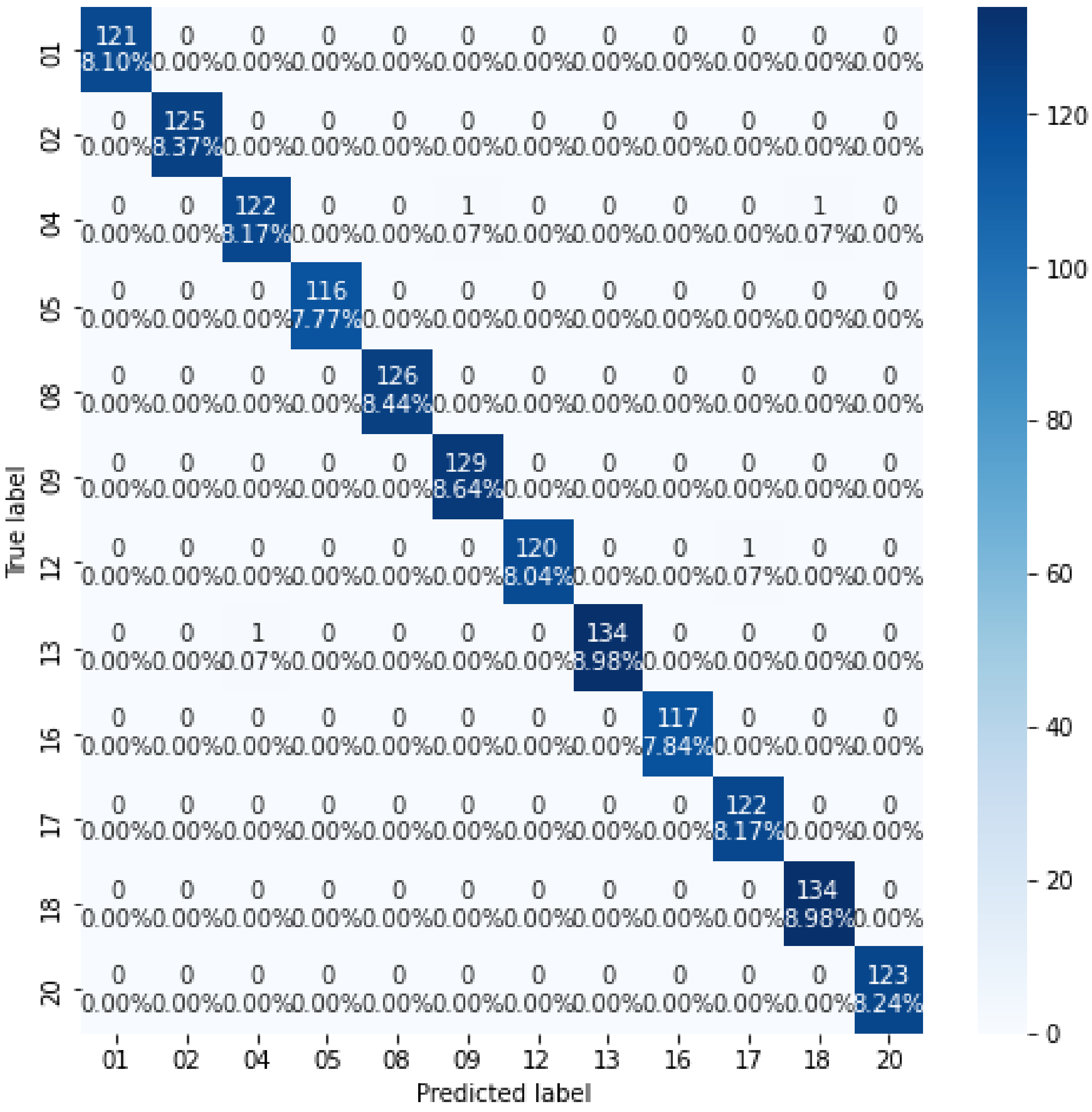
```
        basemodel.build([None, 19, 3])
        #print(model.summary())
        basemodel.compile(loss='sparse_categorical_crossentropy',␣
↪optimizer=adam_v2.Adam(learning_rate=0.001), metrics=['accuracy'])
        return basemodel

    # Function to create model, required for KerasClassifier
    import pickle
    def load_classifier(baseset):
        baseset.sort()
        basename = '-'.join(baseset)
        classifier = KerasClassifier(build_fn=build_model, baseset=baseset,␣
↪epochs=64, batch_size=19, verbose=0)
        classifier.classes_ = pickle.load(open(basename + '_model_classes.
↪pkl','rb'))
        classifier.model = build_model(baseset)
        return classifier

    #print(model.model.summary())
    #print(model.classes_)
    from sklearn.metrics import classification_report
    from sklearn.metrics import confusion_matrix

    for n_splits in [5]:
        for epoch in [[results.best_params_['epochs']]]:
            cv = StratifiedGroupKFold(n_splits=n_splits, shuffle=True,␣
↪random_state=(1000+epoch[0]))
            X, y, g = get_dataset(dataset_cleaned)

            # Initialize the accuracy of the models to blank list. The accuracy␣
↪of each model will be appended to this list
            accuracy_model = []
            best_estimator = None
            # Initialize the array to zero which will store the confusion matrix
            array = None
            outliers = None

            report_cleaned = None
            report_outliers = None

            print("Processing started for split estimator: " + str(n_splits) +␣
↪", epochs: " + str(epoch))
            # Iterate over each train-test split
            fold = 1
            for train_index, test_index in cv.split(X, y, g):
                #print(test_index)
                if len(test_index) == 0 or len(train_index) == 0:
                    continue
                print("Processing ", fold, "-fold")
                fold += 1

                classifier = load_classifier(baseset)
                # Split train-test (Inverted)
                X_train, y_train, group_train = get_dataset(dataset_cleaned,␣
↪train_index)
                X_test, y_test, group_test = get_dataset(dataset_cleaned,␣
↪test_index)
                X_outliers, y_outliers, group_test =␣
↪get_dataset(dataset_outliers)
                # Train the model
                History = classifier.fit(X_train, y_train, epochs=epoch[0])
                # Append to accuracy_model the accuracy of the model
                accuracy_model.append(accuracy_score(y_test, classifier.
↪predict(X_test), normalize=True))
                if accuracy_model[-1] == max(accuracy_model):
                    best_estimator = classifier
                # Calculate the confusion matrix
                c = confusion_matrix(y_test, classifier.predict(X_test))
                # Add the score to the previous confusion matrix of previous␣
↪model
                if isinstance(array, np.ndarray) == False:
                    array = c.copy()
                else:
                    array = array + c

                # Calculate the confusion matrix
                c = confusion_matrix(y_outliers, classifier.predict(X_outliers))
                # Add the score to the previous confusion matrix of previous␣
↪model
                if isinstance(outliers, np.ndarray) == False:
                    outliers = c.copy()
                else:
                    outliers = outliers + c

                #Accumulate for classification report
                if isinstance(report_cleaned, list) == False:
                    report_cleaned = [y_test, classifier.predict(X_test)]
                else:
                    report_cleaned[0] = np.append(report_cleaned[0],y_test)
                    report_cleaned[1] = np.append(report_cleaned[1],classifier.
↪predict(X_test))
                #Accumulate for classification report
```

```
               if isinstance(report_outliers, list) == False:
                   report_outliers = [y_outliers, classifier.
↪predict(X_outliers)]
               else:
                   report_outliers[0] = np.
↪append(report_outliers[0],y_outliers)
                   report_outliers[1] = np.
↪append(report_outliers[1],classifier.predict(X_outliers))

           # Print the accuracy
           print("At split estimator: " + str(n_splits) + ", epochs: " +␣
↪str(epoch))
           print("Accurace mean(std): " + str(np.mean(accuracy_model)) + "(" +␣
↪str(np.std(accuracy_model)) + ")")

           # To calculate the classification reports

           print("Classification report for all valid cross_validations␣
↪against their tests sets")
           print(classification_report(report_cleaned[0], report_cleaned[1],␣
↪target_names=baseset))

           print("Classification report for all valid cross_validations␣
↪against outliers")
           print(classification_report(report_outliers[0], report_outliers[1],␣
↪target_names=baseset))

           # To calculate the confusion matrix

           print("Confusion Matrix for all valid cross_validations against␣
↪their tests sets")
           make_confusion_matrix(array, categories=baseset, figsize=[8,8])

           print("Confusion Matrix for all valid cross_validations against␣
↪outliers")
           make_confusion_matrix(outliers, categories=baseset, figsize=[8,8])
   def save_model(model, baseset):
       baseset.sort()
       name = '-'.join(baseset)
       # saving model
       pickle.dump(model.classes_, open(name + '_model_classes.pkl','wb'))
       model.model.save(name + '_lstm')
   save_model(best_estimator, baseset)
```

```
baseset = base_transfer_set

model = evaluate_model(baseset)
```

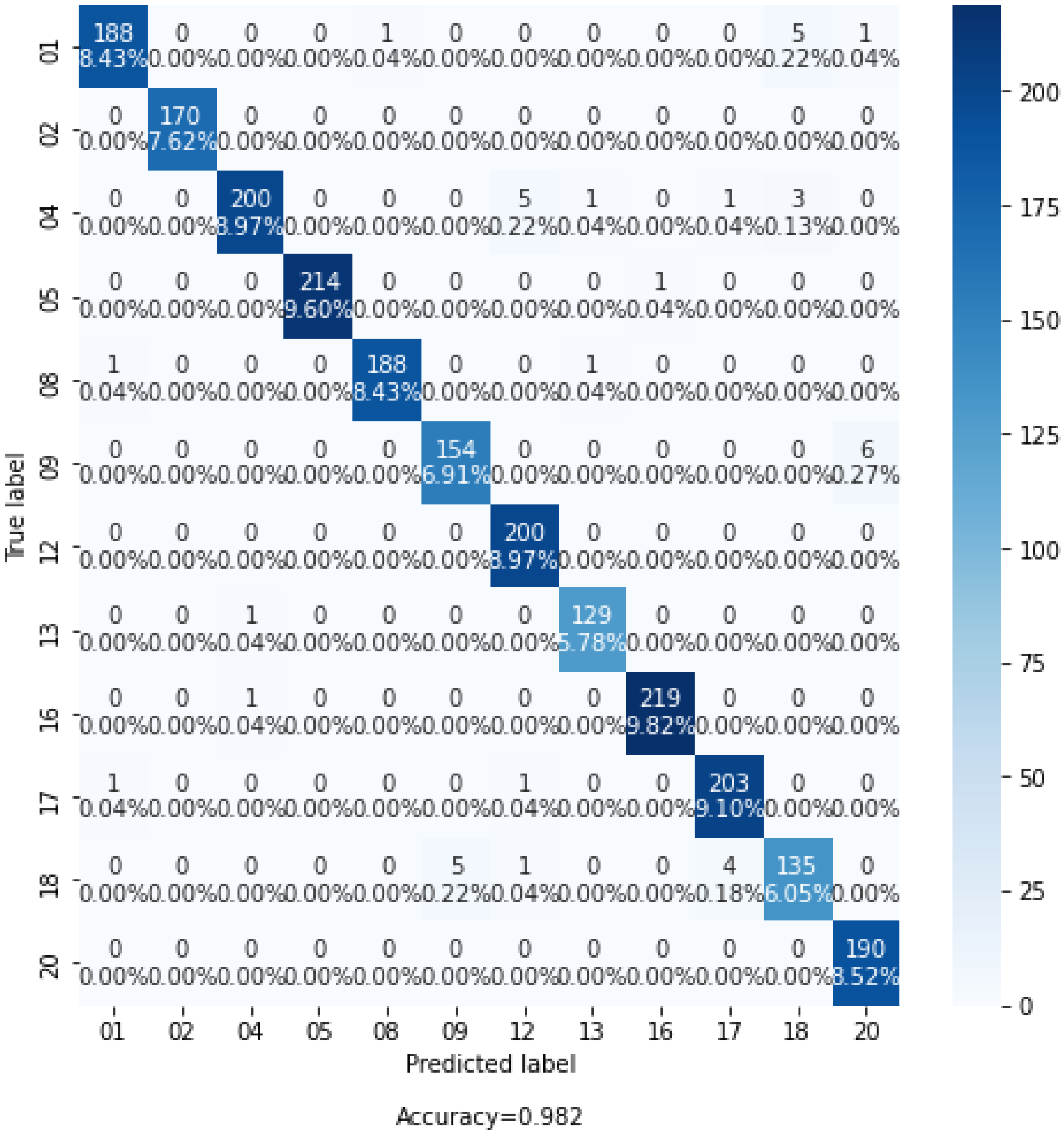
```
Baseset:  ['01', '02', '04', '05', '08', '09', '12', '13', '16', '17', '18',
'20']
Loadind Dataset:  ['01', '02', '04', '05', '08', '09', '12', '13', '16', '17',
'18', '20']
1942 samples loaded
Scaling Dataset:  ['01', '02', '04', '05', '08', '09', '12', '13', '16', '17',
'18', '20']
1942 samples scaled
Cleaning Dataset:  ['01', '02', '04', '05', '08', '09', '12', '13', '16', '17',
'18', '20']
1493 samples cleaned
449 samples outliers
Time slicing Cleaned Dataset:  ['01', '02', '04', '05', '08', '09', '12', '13',
'16', '17', '18', '20']
1493 cleaned samples sliced
0 cleaned samples damaged
Time slicing Outliers Dataset:  ['01', '02', '04', '05', '08', '09', '12', '13',
'16', '17', '18', '20']
446 outliers samples sliced
3 outliers samples damaged
Processing started for split estimator: 5, epochs: [64]
Processing  1 -fold
Processing  2 -fold
Processing  3 -fold
Processing  4 -fold
Processing  5 -fold
At split estimator: 5, epochs: [64]
Accurace mean(std): 0.9965402399198112(0.004691736796150098)
Classification report for all valid cross_validations against their tests sets
              precision    recall  f1-score   support

          01       1.00      1.00      1.00       121
          02       1.00      1.00      1.00       125
          04       0.99      0.98      0.99       124
          05       1.00      1.00      1.00       116
          08       1.00      1.00      1.00       126
          09       0.99      1.00      1.00       129
          12       1.00      0.99      1.00       121
          13       1.00      0.99      1.00       135
          16       1.00      1.00      1.00       117
          17       0.99      1.00      1.00       122
          18       0.99      1.00      1.00       134
          20       1.00      1.00      1.00       123
```

```
    accuracy                           1.00      1493
   macro avg       1.00      1.00      1.00      1493
weighted avg       1.00      1.00      1.00      1493

Classification report for all valid cross_validations against outliers
              precision    recall  f1-score   support

          01       0.99      0.96      0.98       195
          02       1.00      1.00      1.00       170
          04       0.99      0.95      0.97       210
          05       1.00      1.00      1.00       215
          08       0.99      0.99      0.99       190
          09       0.97      0.96      0.97       160
          12       0.97      1.00      0.98       200
          13       0.98      0.99      0.99       130
          16       1.00      1.00      1.00       220
          17       0.98      0.99      0.98       205
          18       0.94      0.93      0.94       145
          20       0.96      1.00      0.98       190

    accuracy                           0.98      2230
   macro avg       0.98      0.98      0.98      2230
weighted avg       0.98      0.98      0.98      2230

Confusion Matrix for all valid cross_validations against their tests sets
Confusion Matrix for all valid cross_validations against outliers
```



	01	02	04	05	08	09	12	13	16	17	18	20
01	121 8.10%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
02	0 0.00%	125 8.37%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
04	0 0.00%	0 0.00%	122 8.17%	0 0.00%	0 0.00%	1 0.07%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	1 0.07%	0 0.00%
05	0 0.00%	0 0.00%	0 0.00%	116 7.77%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
08	0 0.00%	0 0.00%	0 0.00%	0 0.00%	126 8.44%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
09	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	129 8.64%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
12	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	120 8.04%	0 0.00%	0 0.00%	1 0.07%	0 0.00%	0 0.00%
13	0 0.00%	0 0.00%	1 0.07%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	134 8.98%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
16	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	117 7.84%	0 0.00%	0 0.00%	0 0.00%
17	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	122 8.17%	0 0.00%	0 0.00%
18	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	134 8.98%	0 0.00%
20	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	123 8.24%

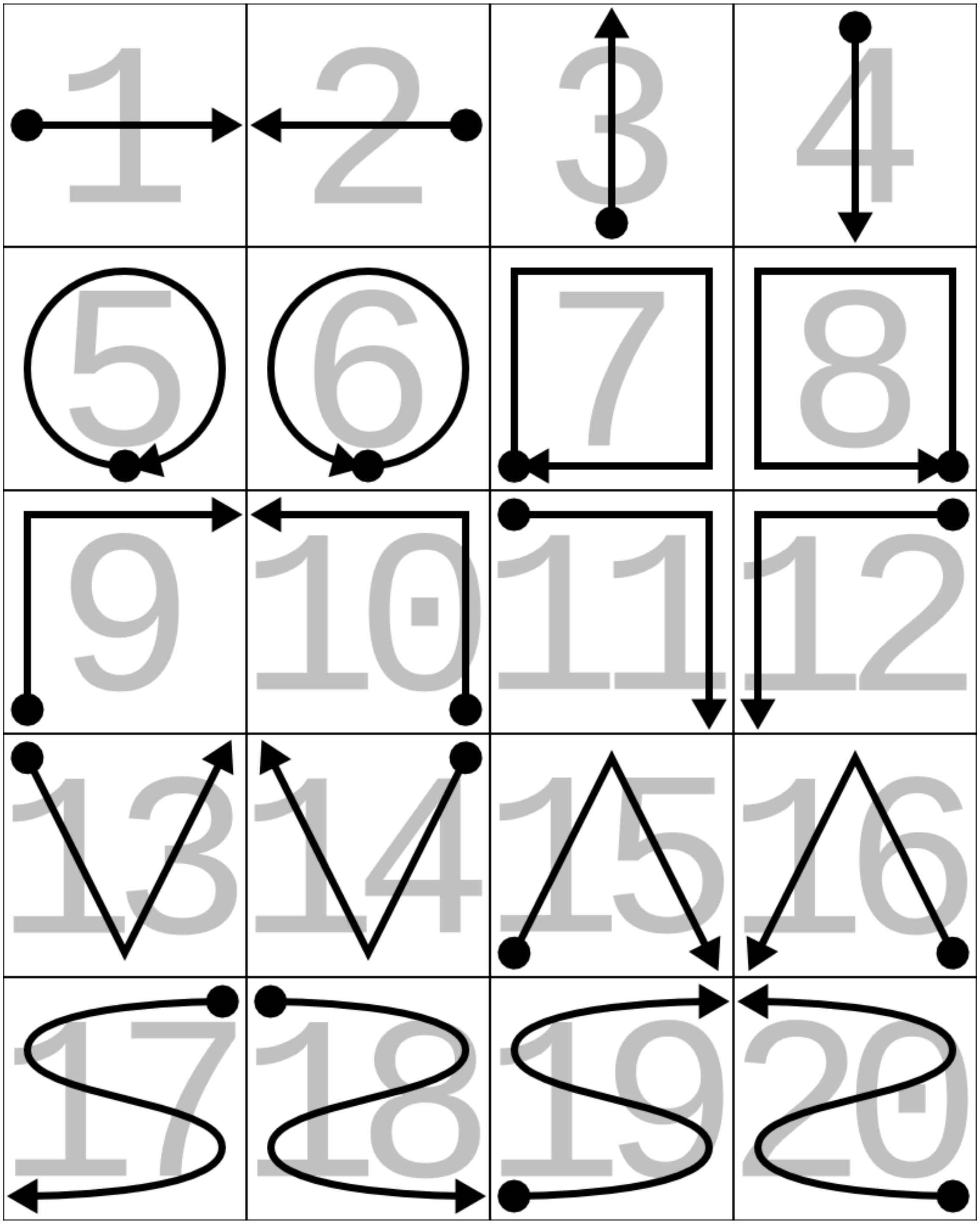
Accuracy=0.997



	01	02	04	05	08	09	12	13	16	17	18	20
01	188 8.43%	0 0.00%	0 0.00%	0 0.00%	1 0.04%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	5 0.22%	1 0.04%
02	0 0.00%	170 7.62%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
04	0 0.00%	0 0.00%	200 8.97%	0 0.00%	0 0.00%	0 0.00%	5 0.22%	1 0.04%	0 0.00%	1 0.04%	3 0.13%	0 0.00%
05	0 0.00%	0 0.00%	0 0.00%	214 9.60%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	1 0.04%	0 0.00%	0 0.00%	0 0.00%
08	1 0.04%	0 0.00%	0 0.00%	0 0.00%	188 8.43%	0 0.00%	0 0.00%	1 0.04%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
09	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	154 6.91%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	6 0.27%
12	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	200 8.97%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
13	0 0.00%	0 0.00%	1 0.04%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	129 5.78%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
16	0 0.00%	0 0.00%	1 0.04%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	219 9.82%	0 0.00%	0 0.00%	0 0.00%
17	1 0.04%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	1 0.04%	0 0.00%	0 0.00%	203 9.10%	0 0.00%	0 0.00%
18	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	5 0.22%	1 0.04%	0 0.00%	0 0.00%	4 0.18%	135 6.05%	0 0.00%
20	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	190 8.52%

[ ]:
```
from IPython.display import Image
Image('gestures-dataset/gestures.png')
```

[ ]:


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20