

Instituto Tecnológico de Estudios Superiores de Monterrey

School of Engineering and Sciences



Master of Science in Computer Science

Homework 3

Denisse Chacón Ramírez — A01562077

Emilio Rios Ochoa — A01378965

Luis Ignacio Ferro Salinas — A01378248

Computer Vision (CS5050)

Dr. José Gerardo Tamez Peña

November 15th, 2024

1.- Letter recognition using three ML models.

Load the letter recognition dataset.

```
In [ ]: from ucimlrepo import fetch_ucirepo  
  
# fetch dataset  
letter_recognition = fetch_ucirepo(id=59)  
  
# data (as pandas dataframes)  
X = letter_recognition.data.features  
y = letter_recognition.data.targets  
  
# metadata  
print(letter_recognition.metadata)
```

```
letter  feature_1  feature_2  feature_3  feature_4  feature_5  feature_6  \  
0      T          2          8          3          5          1          8  
1      I          5         12          3          7          2         10  
2      D          4         11          6          8          6         10  
3      N          7         11          6          6          3          5  
4      G          2          1          3          1          1          8  
  
    feature_7  feature_8  feature_9  feature_10  feature_11  feature_12  \  
0        13          0          6          6         10          8  
1        5          5          4         13          3          9  
2        6          2          6         10          3          7  
3        9          4          6          4          4         10  
4        6          6          6          6          5          9  
  
    feature_13  feature_14  feature_15  feature_16  
0          0          8          0          8  
1          2          8          4         10  
2          3          7          3          9  
3          6         10          2          8  
4          1          7          5         10
```

```
In [ ]: # variable information  
print(letter_recognition.variables)
```

	name	role	type	demographic	description \
0	lettr	Target	Categorical	None	capital letter
1	x-box	Feature	Integer	None	horizontal position of box
2	y-box	Feature	Integer	None	vertical position of box
3	width	Feature	Integer	None	width of box
4	high	Feature	Integer	None	height of box
5	onpix	Feature	Integer	None	total # on pixels
6	x-bar	Feature	Integer	None	mean x of on pixels in box
7	y-bar	Feature	Integer	None	mean y of on pixels in box
8	x2bar	Feature	Integer	None	mean x variance
9	y2bar	Feature	Integer	None	mean y variance
10	xybar	Feature	Integer	None	mean x y correlation
11	x2ybr	Feature	Integer	None	mean of x * x * y
12	xy2br	Feature	Integer	None	mean of x * y * y
13	x-ege	Feature	Integer	None	mean edge count left to right
14	xegvy	Feature	Integer	None	correlation of x-ege with y
15	y-ege	Feature	Integer	None	mean edge count bottom to top
16	yegvx	Feature	Integer	None	correlation of y-ege with x
	units	missing_values			
0	None	no			
1	None	no			
2	None	no			
3	None	no			
4	None	no			
5	None	no			
6	None	no			
7	None	no			
8	None	no			
9	None	no			
10	None	no			
11	None	no			
12	None	no			
13	None	no			
14	None	no			
15	None	no			
16	None	no			

Split the data into traind and test (using 70% of the data for training and 30% for testing)

In []: `#!pip install -U scikit-learn`

```
Collecting scikit-learn
  Downloading scikit_learn-1.5.2-cp312-cp312-macosx_12_0_arm64.whl.metadata (13 kB)
Requirement already satisfied: numpy>=1.19.5 in /Users/cdr_c/anaconda3/envs/my_env/lib/python3.12/site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /Users/cdr_c/anaconda3/envs/my_env/lib/python3.12/site-packages (from scikit-learn) (1.14.1)
Collecting joblib>=1.2.0 (from scikit-learn)
  Using cached joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Using cached threadpoolctl-3.5.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.5.2-cp312-cp312-macosx_12_0_arm64.whl (11.0 MB)
    11.0/11.0 MB 6.1 MB/s eta 0:00:0000:0100:01
Using cached joblib-1.4.2-py3-none-any.whl (301 kB)
Using cached threadpoolctl-3.5.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, joblib, scikit-learn
Successfully installed joblib-1.4.2 scikit-learn-1.5.2 threadpoolctl-3.5.0
```

```
In [ ]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

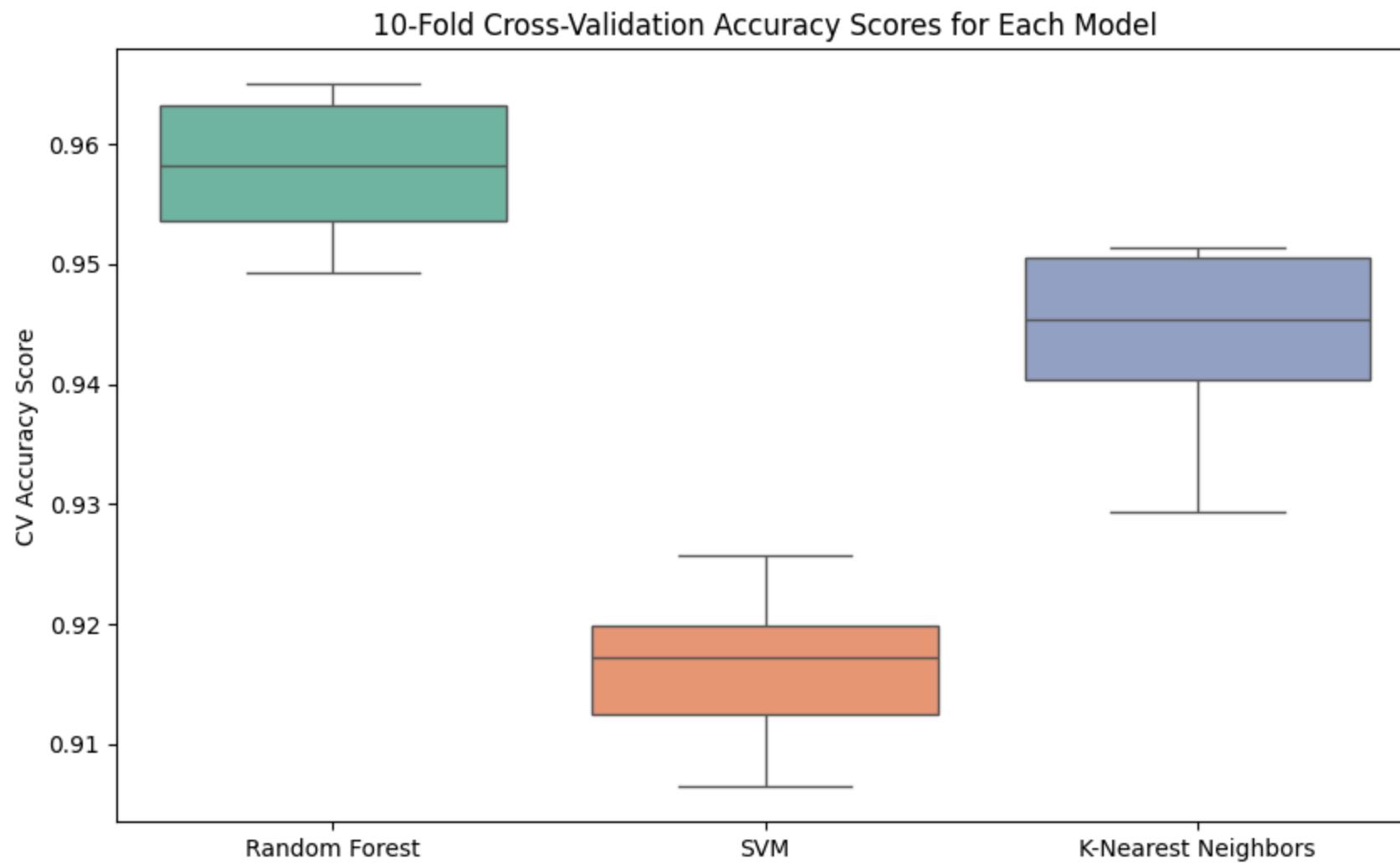
Train and evaluate the models Random Forest, SVM and KNN

```
In [ ]: models = {
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
    'K-Nearest Neighbors': KNeighborsClassifier()
}

cv_results = {}
for model_name, model in models.items():
    cv_score = cross_val_score(model, X_train, y_train.values.ravel(), cv=10, scoring='accuracy')
    cv_results[model_name] = cv_score
    print(f'{model_name} 10-fold CV Mean Accuracy: {np.mean(cv_score):.3f}')
```

```
Random Forest 10-fold CV Mean Accuracy: 0.958
SVM 10-fold CV Mean Accuracy: 0.916
K-Nearest Neighbors 10-fold CV Mean Accuracy: 0.944
```

```
In [ ]: plt.figure(figsize=(10, 6))
sns.boxplot(data=list(cv_results.values()), palette="Set2")
plt.xticks(range(len(models)), list(models.keys()))
plt.ylabel("CV Accuracy Score")
plt.title("10-Fold Cross-Validation Accuracy Scores for Each Model")
plt.show()
```



Hyperparameter tuning

```
In [ ]: param_grids = {
    'Random Forest': {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20]},
    'SVM': {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']},
    'K-Nearest Neighbors': {'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance']}
}
```

```
best_models = {}
for model_name, model in models.items():
    grid_search = GridSearchCV(model, param_grids[model_name], cv=10, scoring='accuracy')
    grid_search.fit(X_train, y_train.values.ravel())
    best_models[model_name] = grid_search.best_estimator_
    print(f'{model_name} Best Hyperparameters: {grid_search.best_params_}')
```

Random Forest Best Hyperparameters: {'max_depth': None, 'n_estimators': 200}

SVM Best Hyperparameters: {'C': 10, 'kernel': 'rbf'}

K-Nearest Neighbors Best Hyperparameters: {'n_neighbors': 5, 'weights': 'distance'}

Evaluate the models with the best parameters

```
In [ ]: for model_name, model in best_models.items():
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'{model_name} Hold-out Test Accuracy: {accuracy:.3f}')
```

Random Forest Hold-out Test Accuracy: 0.961

SVM Hold-out Test Accuracy: 0.962

K-Nearest Neighbors Hold-out Test Accuracy: 0.955

As we can see, the model with the highest accuracy was Random Forest.

```
In [ ]:
```

Satellite Data

Now use a statistical-based filter selection method to select the most discriminant feature.

Load the necessary libraries and the data

```
In [ ]: # Step 1: Load and Preprocess Data
from ucimlrepo import fetch_ucirepo
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Fetch the dataset
statlog_landsat_satellite = fetch_ucirepo(id=146)

# data (as pandas dataframes)
X = statlog_landsat_satellite.data.features
y = statlog_landsat_satellite.data.targets
```

```
In [ ]: df = X.assign(y=y)
```

Now, we report one-to other t-test p-values

```
In [ ]: from scipy import stats

class_data = {}
for class_label in y["class"].unique():
    class_data[class_label] = df[df["y"] == class_label]

results = []
classes = list(class_data.keys())

for i in range(len(classes)):
    for j in range(i+1, len(classes)):
        class1 = class_data[classes[i]]
        class2 = class_data[classes[j]]
        for column in df.columns[:-1]:
```

```

t_statistic, p_value = stats.ttest_ind(class1[column], class2[column])
results.append((column, classes[i], classes[j], t_statistic, p_value))

result_t_test = pd.DataFrame(results, columns = ['Feature', 'Class 1', 'Class2', 't-statistic', 'p-value'])
#significance 0.05
#alpha = 0.05

#selected_features_t_test = result_t_test[result_t_test['p-value'] < alpha]

#print(selected_features_t_test)

```

Out[]:

	Feature	Class 1	Class2	t-statistic	p-value
0	Attribute1	3	4	30.096091	3.281795e-164
1	Attribute2	3	4	29.127384	1.453090e-155
2	Attribute3	3	4	32.072758	3.645964e-182
3	Attribute4	3	4	30.919016	1.227980e-171
4	Attribute5	3	4	32.688717	7.787227e-188
...
535	Attribute32	2	1	44.236078	1.618743e-307
536	Attribute33	2	1	-36.648910	1.163164e-230
537	Attribute34	2	1	-79.278275	0.000000e+00
538	Attribute35	2	1	8.362560	1.065254e-16
539	Attribute36	2	1	41.741003	5.159350e-282

540 rows × 5 columns

Analogously, we performed the One-to-Other KS- p-values

In []:

```

ks_results = []
classes = list(class_data.keys())

for i in range(len(classes)):
    for j in range(i+1, len(classes)):
        class1 = class_data[classes[i]]
        class2 = class_data[classes[j]]
        for column in df.columns[:-1]:
            #ks

```

```

    ks_statistic, ks_p_value = stats.ks_2samp(class1[column], class2[column])
    ks_results.append((column, classes[i], classes[j], ks_statistic, ks_p_value))

result_ks_test = pd.DataFrame(ks_results, columns = ['Feature', 'Class 1', 'Class2', 't-statistic', 'p-value'])
result_ks_test

```

Out[]:

	Feature	Class 1	Class2	t-statistic	p-value
0	Attribute1	3	4	0.590553	1.081019e-139
1	Attribute2	3	4	0.618637	1.782161e-154
2	Attribute3	3	4	0.620436	1.980902e-155
3	Attribute4	3	4	0.596374	1.128059e-142
4	Attribute5	3	4	0.614752	2.349314e-152
...
535	Attribute32	2	1	0.781465	1.822375e-294
536	Attribute33	2	1	0.690157	3.951711e-221
537	Attribute34	2	1	0.865572	4.051338e-322
538	Attribute35	2	1	0.177414	9.829564e-14
539	Attribute36	2	1	0.762973	2.865326e-278

540 rows × 5 columns

Now using ROC AUC

In []:

```

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.metrics import roc_auc_score

#separate in train and test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

roc_auc_results = []

for feature in X_train.columns:
    clf = DecisionTreeClassifier()

    clf.fit(X_train[feature].fillna(0).to_frame(), y_train)

```

```
y_scored = clf.predict_proba(X_test[feature].to_frame())

roc_auc_results.append(roc_auc_score(y_test, y_scored, multi_class='ovr'))

results_roc = pd.Series(roc_auc_results)

results_roc.index = X_train.columns

results_roc_df = pd.DataFrame({
    'Feature': results_roc.index,
    'roc_auc': results_roc.values
})

# Muestra el DataFrame
results_roc_df
```

	Feature	roc_auc
0	Attribute1	0.841729
1	Attribute2	0.841859
2	Attribute3	0.796173
3	Attribute4	0.827412
4	Attribute5	0.858386
5	Attribute6	0.851289
6	Attribute7	0.796095
7	Attribute8	0.827831
8	Attribute9	0.845229
9	Attribute10	0.837111
10	Attribute11	0.797725
11	Attribute12	0.820930
12	Attribute13	0.859329
13	Attribute14	0.853800
14	Attribute15	0.806963
15	Attribute16	0.842519
16	Attribute17	0.876212
17	Attribute18	0.878900
18	Attribute19	0.805027
19	Attribute20	0.845358
20	Attribute21	0.869825
21	Attribute22	0.861839
22	Attribute23	0.805540
23	Attribute24	0.835885
24	Attribute25	0.840513
25	Attribute26	0.834317
26	Attribute27	0.786526

	Feature	roc_auc
27	Attribute28	0.826860
28	Attribute29	0.855004
29	Attribute30	0.848189
30	Attribute31	0.797210
31	Attribute32	0.820969
32	Attribute33	0.853111
33	Attribute34	0.840684
34	Attribute35	0.792385
35	Attribute36	0.826401

Adjust the p_values for FDR

```
In [ ]: from statsmodels.stats.multitest import fdrcorrection

# Adjust p-values for FDR (both t-test and KS-test)
t_test_p_values = result_t_test['p-value'].values
ks_p_values = result_ks_test['p-value'].values

t_test_adjusted = multipletests(t_test_p_values, method='fdr_bh')[1]
ks_adjusted = multipletests(ks_p_values, method='fdr_bh')[1]

result_t_test['FDR-adjusted p-value'] = t_test_adjusted
result_ks_test['FDR-adjusted p-value'] = ks_adjusted
```

Compare the ranking of the p-values for the three different methods.

First, we are going to calculate the mean of the corrected p values for each feature for the t-test and ks test.

```
In [ ]: # ks
mean_p_values_ks = result_ks_test.groupby('Feature')['FDR-adjusted p-value'].mean().reset_index()
mean_p_values_ks.rename(columns={'p-value': 'FDR-adjusted p-value'}, inplace=True)
mean_p_values_df_ks = mean_p_values_ks

# t-test
mean_p_values_t_test = result_t_test.groupby('Feature')['FDR-adjusted p-value'].mean().reset_index()
mean_p_values_t_test.rename(columns={'p-value': 'FDR-adjusted p-value'}, inplace=True)
mean_p_values_df_t = mean_p_values_t_test
```

Now, we combined the results in a dataframe

```
In [ ]: ranking_df = pd.merge(mean_p_values_df_t[['Feature', 'FDR-adjusted p-value']],
                           mean_p_values_df_ks[['Feature', 'FDR-adjusted p-value']],
                           on='Feature', suffixes=('_t_test', '_ks'))
ranking_df = pd.merge(ranking_df, results_roc_df, on='Feature')
```

```
In [ ]: ranking_df
```

Out[]:

	Feature	FDR-adjusted p-value_t_test	FDR-adjusted p-value_ks	roc_auc
0	Attribute1	6.610641e-15	1.516973e-30	0.841729
1	Attribute10	6.834457e-26	1.164221e-46	0.837111
2	Attribute11	1.156428e-02	4.040214e-04	0.797725
3	Attribute12	2.865018e-10	4.172099e-23	0.820930
4	Attribute13	8.625106e-15	3.112649e-31	0.859329
5	Attribute14	6.338784e-15	8.703658e-44	0.853800
6	Attribute15	5.174310e-06	1.646539e-07	0.806963
7	Attribute16	1.464519e-05	9.195123e-30	0.842519
8	Attribute17	4.161674e-25	1.460148e-42	0.876212
9	Attribute18	4.891669e-16	4.130507e-51	0.878900
10	Attribute19	7.059655e-04	6.700984e-06	0.805027
11	Attribute2	1.095815e-16	1.029653e-38	0.841859
12	Attribute20	1.254657e-06	2.694846e-32	0.845358
13	Attribute21	6.702344e-20	1.783788e-43	0.869825
14	Attribute22	1.158223e-20	4.202383e-52	0.861839
15	Attribute23	2.418061e-04	1.764358e-05	0.805540
16	Attribute24	1.325681e-07	3.410477e-23	0.835885
17	Attribute25	2.869168e-06	3.574524e-24	0.840513
18	Attribute26	6.989594e-14	4.846886e-39	0.834317
19	Attribute27	1.456329e-09	1.064026e-11	0.786526
20	Attribute28	6.229255e-04	5.057471e-17	0.826860
21	Attribute29	1.638265e-12	2.536568e-35	0.855004
22	Attribute3	9.339383e-03	8.058272e-05	0.796173
23	Attribute30	6.638249e-18	2.059393e-45	0.848189
24	Attribute31	3.458041e-06	3.072608e-09	0.797210
25	Attribute32	1.596755e-06	1.752727e-20	0.820969
26	Attribute33	3.242728e-12	2.537384e-38	0.853111

	Feature	FDR-adjusted p-value_t_test	FDR-adjusted p-value_ks	roc_auc
27	Attribute34	8.513060e-24	1.197893e-45	0.840684
28	Attribute35	1.936563e-05	5.499561e-08	0.792385
29	Attribute36	4.333297e-09	1.543257e-19	0.826401
30	Attribute4	1.002249e-10	2.849858e-25	0.827412
31	Attribute5	5.457235e-21	2.118687e-38	0.858386
32	Attribute6	1.381265e-19	8.581139e-44	0.851289
33	Attribute7	2.843596e-02	2.863931e-05	0.796095
34	Attribute8	1.330187e-10	4.516820e-20	0.827831
35	Attribute9	5.993331e-16	2.855948e-34	0.845229

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

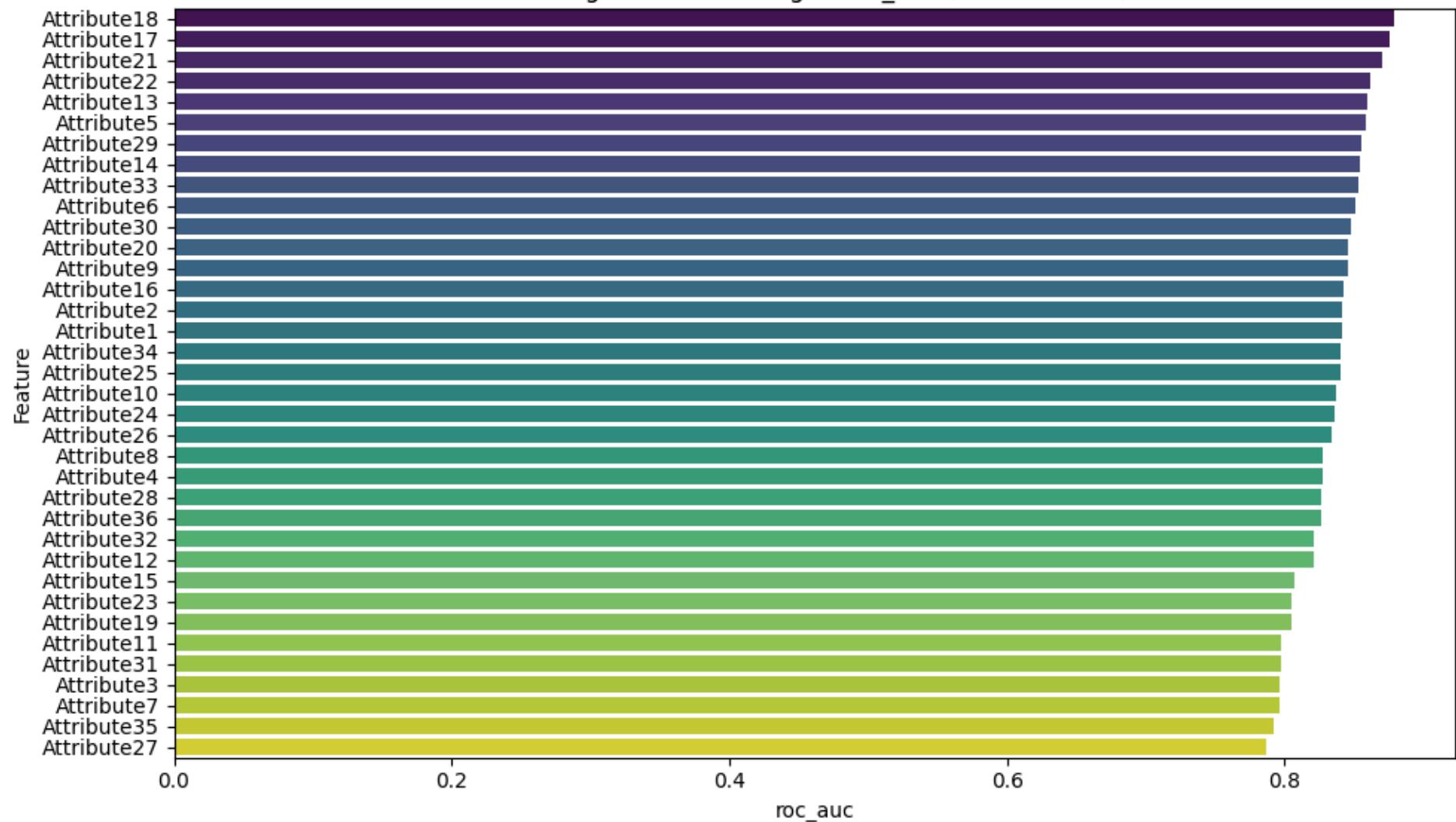
#SORTED BY AUC
plt.figure(figsize=(10, 6))
sns.barplot(x='roc_auc', y='Feature', data=ranking_df.sort_values('roc_auc', ascending=False), palette='viridis')
plt.title('Ranking de Features según roc_auc (Descendente)')
plt.xlabel('roc_auc')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```

/var/folders/h4/v6kv36fs44lg2yrrlf2z2jyr0000gn/T/ipykernel_90249/1083825134.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

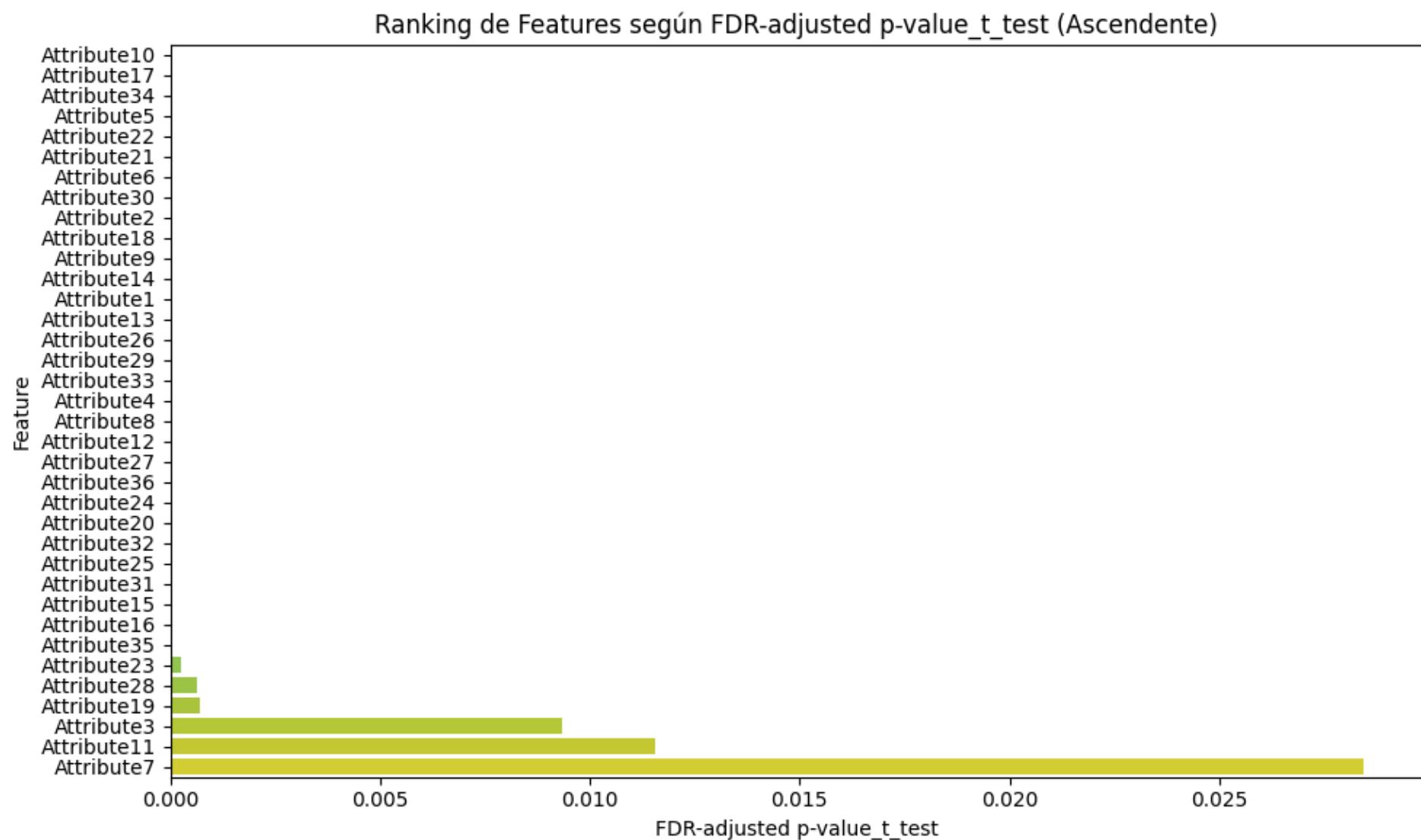
```
sns.barplot(x='roc_auc', y='Feature', data=ranking_df.sort_values('roc_auc', ascending=False), palette='viridis')
```

Ranking de Features según roc_auc (Descendente)



```
In [ ]: #SORTED BY T-TEST
plt.figure(figsize=(10, 6))
sns.barplot(x='FDR-adjusted p-value_t_test', y='Feature', data=ranking_df.sort_values('FDR-adjusted p-value_t_test'), palette='viridis')
plt.title('Ranking de Features según FDR-adjusted p-value_t_test (Ascendente)')
plt.xlabel('FDR-adjusted p-value_t_test')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```

```
/var/folders/h4/v6kv36fs44lg2yrrlf2z2jyr0000gn/T/ipykernel_90249/1207628925.py:3: FutureWarning:  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue`  
` and set `legend=False` for the same effect.  
sns.barplot(x='FDR-adjusted p-value_t_test', y='Feature', data=ranking_df.sort_values('FDR-adjusted p-value_t_test'),  
palette='viridis')
```



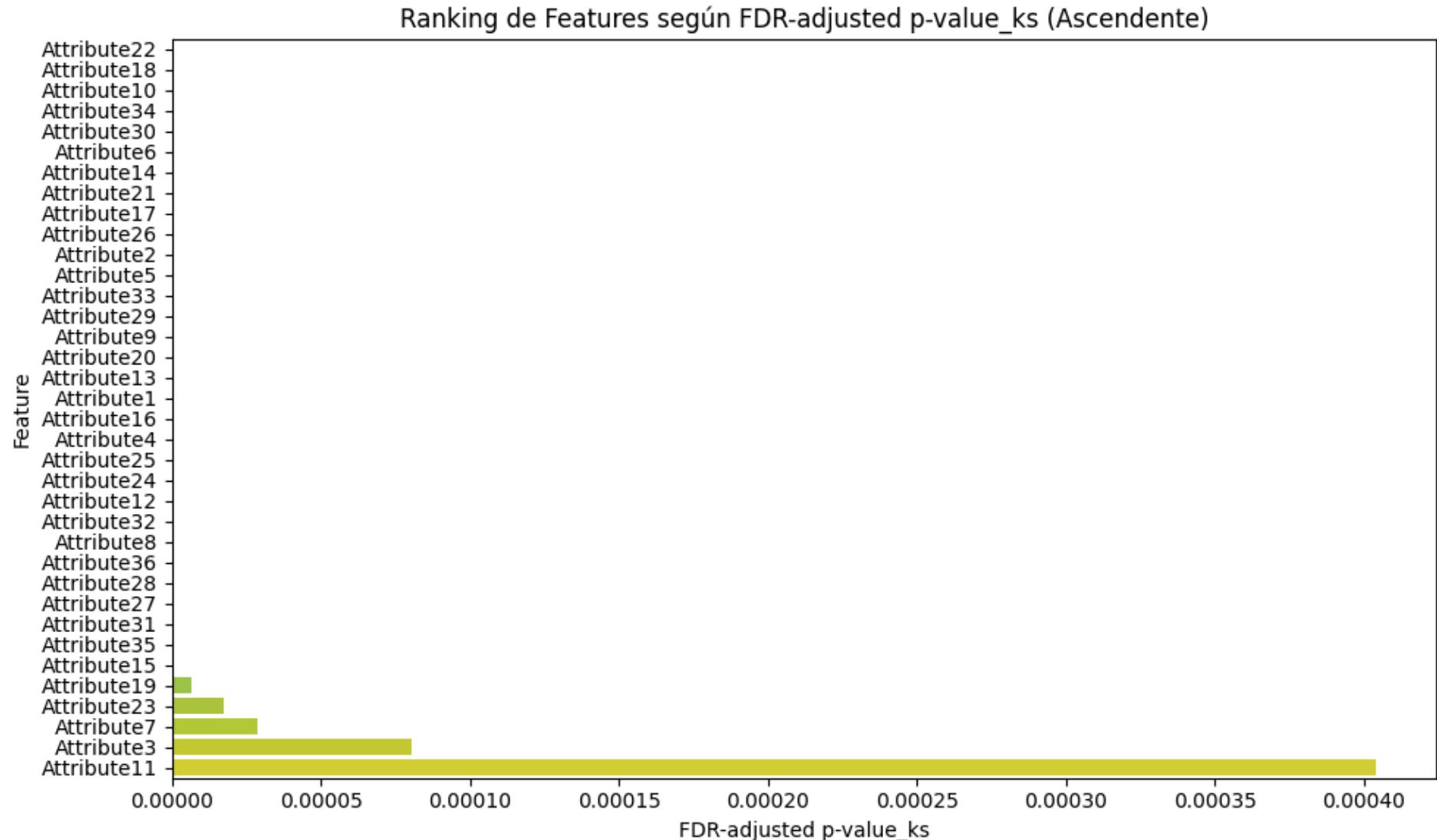
```
In [ ]: #sorted by ks  
plt.figure(figsize=(10, 6))  
sns.barplot(x='FDR-adjusted p-value_ks', y='Feature', data=ranking_df.sort_values('FDR-adjusted p-value_ks'), palette='viridis')  
plt.title('Ranking de Features según FDR-adjusted p-value_ks (Ascendente)')  
plt.xlabel('FDR-adjusted p-value_ks')  
plt.ylabel('Feature')
```

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

/var/folders/h4/v6kv36fs44lg2yrrlf2z2jyr0000gn/T/ipykernel_90249/2410965441.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x='FDR-adjusted p-value_ks', y='Feature', data=ranking_df.sort_values('FDR-adjusted p-value_ks'), palette='viridis')
```



Select any heuristic feature selection method. Compare the ranking to the statistical methods. We are going to use the Chi-Square Test method

```
In [ ]: import sklearn.feature_selection as FS  
  
chi_values, p_values = FS.chi2(X, y)  
  
indices_chi = np.argsort(chi_values)[::-1]  
  
chi_df = pd.DataFrame({  
    'Feature': X.columns[indices_chi], # Use the indices to sort the feature names  
    'chi': chi_values[indices_chi] # Use the sorted chi values  
})  
  
print(chi_df)
```

	Feature	chi
0	Attribute18	31767.095869
1	Attribute22	30510.985521
2	Attribute14	29725.448977
3	Attribute6	29233.716925
4	Attribute30	29215.811159
5	Attribute34	28780.786126
6	Attribute2	28208.488641
7	Attribute10	27851.344253
8	Attribute26	27238.571806
9	Attribute20	20208.148464
10	Attribute16	19385.145654
11	Attribute24	19358.263583
12	Attribute32	18444.822271
13	Attribute8	18371.176465
14	Attribute28	18054.859025
15	Attribute12	17974.249954
16	Attribute36	17596.603313
17	Attribute4	17555.029307
18	Attribute17	13296.630900
19	Attribute21	12813.377998
20	Attribute13	12542.874406
21	Attribute5	12336.219215
22	Attribute29	12323.142085
23	Attribute33	11974.142740
24	Attribute1	11856.141364
25	Attribute9	11853.039207
26	Attribute25	11772.071611
27	Attribute19	11144.625537
28	Attribute15	10698.193478
29	Attribute23	10574.750434
30	Attribute31	10202.998419
31	Attribute7	10199.845914
32	Attribute27	10008.605838
33	Attribute11	9896.457428
34	Attribute3	9793.304309
35	Attribute35	9683.338290

```
In [ ]: ranking_df = pd.merge(ranking_df, chi_df, on='Feature')
ranking_df
```

Out[]:

	Feature	FDR-adjusted p-value_t_test	FDR-adjusted p-value_ks	roc_auc	chi
0	Attribute1	6.610641e-15	1.516973e-30	0.841729	11856.141364
1	Attribute10	6.834457e-26	1.164221e-46	0.837111	27851.344253
2	Attribute11	1.156428e-02	4.040214e-04	0.797725	9896.457428
3	Attribute12	2.865018e-10	4.172099e-23	0.820930	17974.249954
4	Attribute13	8.625106e-15	3.112649e-31	0.859329	12542.874406
5	Attribute14	6.338784e-15	8.703658e-44	0.853800	29725.448977
6	Attribute15	5.174310e-06	1.646539e-07	0.806963	10698.193478
7	Attribute16	1.464519e-05	9.195123e-30	0.842519	19385.145654
8	Attribute17	4.161674e-25	1.460148e-42	0.876212	13296.630900
9	Attribute18	4.891669e-16	4.130507e-51	0.878900	31767.095869
10	Attribute19	7.059655e-04	6.700984e-06	0.805027	11144.625537
11	Attribute2	1.095815e-16	1.029653e-38	0.841859	28208.488641
12	Attribute20	1.254657e-06	2.694846e-32	0.845358	20208.148464
13	Attribute21	6.702344e-20	1.783788e-43	0.869825	12813.377998
14	Attribute22	1.158223e-20	4.202383e-52	0.861839	30510.985521
15	Attribute23	2.418061e-04	1.764358e-05	0.805540	10574.750434
16	Attribute24	1.325681e-07	3.410477e-23	0.835885	19358.263583
17	Attribute25	2.869168e-06	3.574524e-24	0.840513	11772.071611
18	Attribute26	6.989594e-14	4.846886e-39	0.834317	27238.571806
19	Attribute27	1.456329e-09	1.064026e-11	0.786526	10008.605838
20	Attribute28	6.229255e-04	5.057471e-17	0.826860	18054.859025
21	Attribute29	1.638265e-12	2.536568e-35	0.855004	12323.142085
22	Attribute3	9.339383e-03	8.058272e-05	0.796173	9793.304309
23	Attribute30	6.638249e-18	2.059393e-45	0.848189	29215.811159
24	Attribute31	3.458041e-06	3.072608e-09	0.797210	10202.998419
25	Attribute32	1.596755e-06	1.752727e-20	0.820969	18444.822271
26	Attribute33	3.242728e-12	2.537384e-38	0.853111	11974.142740

	Feature	FDR-adjusted p-value_t_test	FDR-adjusted p-value_ks	roc_auc	chi
27	Attribute34	8.513060e-24	1.197893e-45	0.840684	28780.786126
28	Attribute35	1.936563e-05	5.499561e-08	0.792385	9683.338290
29	Attribute36	4.333297e-09	1.543257e-19	0.826401	17596.603313
30	Attribute4	1.002249e-10	2.849858e-25	0.827412	17555.029307
31	Attribute5	5.457235e-21	2.118687e-38	0.858386	12336.219215
32	Attribute6	1.381265e-19	8.581139e-44	0.851289	29233.716925
33	Attribute7	2.843596e-02	2.863931e-05	0.796095	10199.845914
34	Attribute8	1.330187e-10	4.516820e-20	0.827831	18371.176465
35	Attribute9	5.993331e-16	2.855948e-34	0.845229	11853.039207

In []:

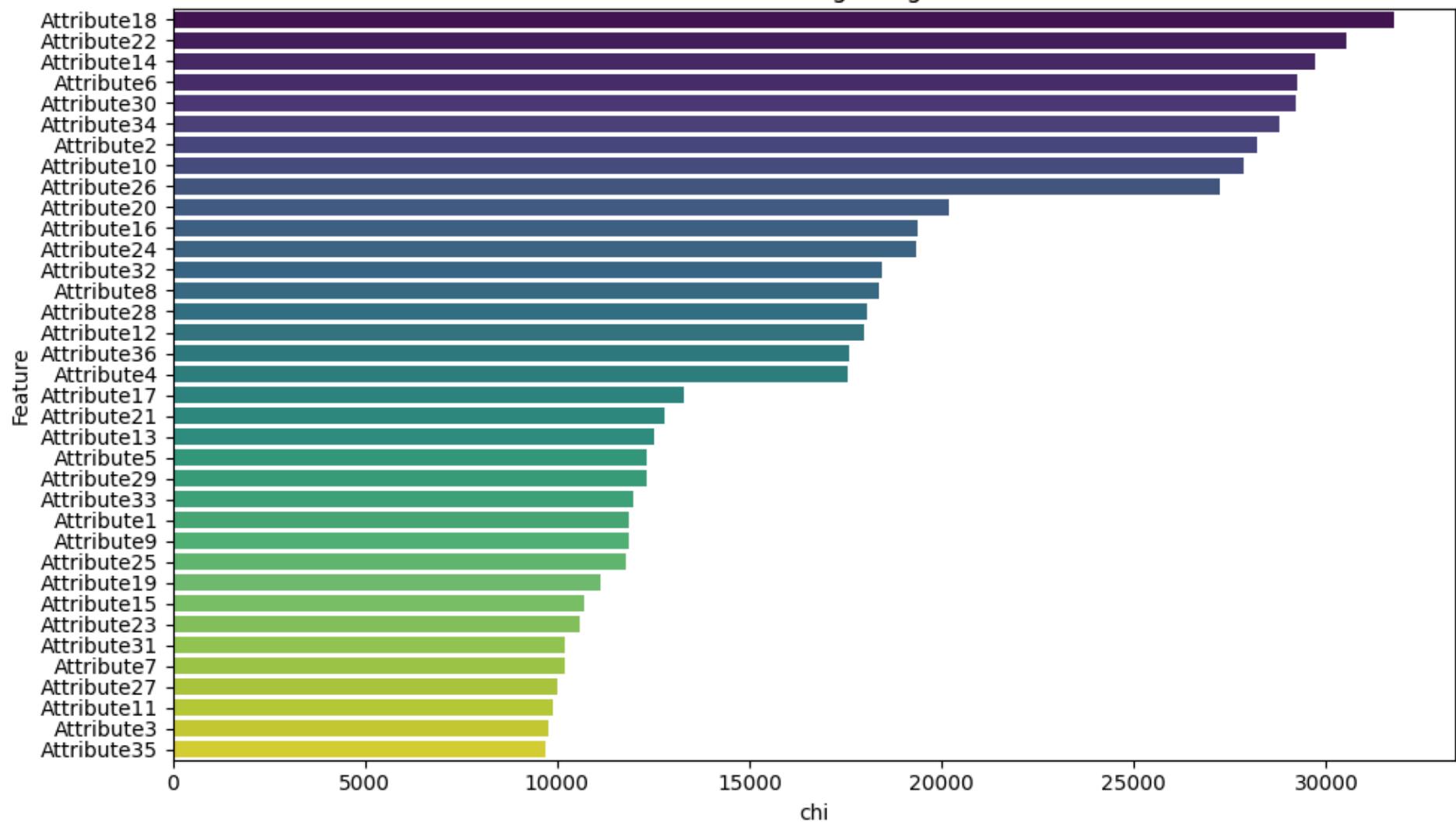
```
#SORTED BY chi
plt.figure(figsize=(10, 6))
sns.barplot(x='chi', y='Feature', data=ranking_df.sort_values('chi', ascending=False), palette='viridis')
plt.title('Feature ranking using chi ')
plt.xlabel('chi')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```

/var/folders/h4/v6kv36fs44lg2yrrlf2z2jyr0000gn/T/ipykernel_90249/668243646.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x='chi', y='Feature', data=ranking_df.sort_values('chi', ascending=False), palette='viridis')
```

Feature ranking using chi



As we can see each method ranked the features in a different order. Nevertheless we can observe that attributes 3, 11 and 25 are in the last places of the 4 rankings, while features 18, 22 and 17 are in the first places in most of the rankings.

Do a PCA transform on the data

```
In [ ]: from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score

# Step 1: Perform PCA on the data
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
```

Select the PCA components that explain 95% of the variance

```
In [ ]: explained_variance_ratio = np.cumsum(pca.explained_variance_ratio_)
n_components = np.argmax(explained_variance_ratio >= 0.95) + 1
print(f"Number of PCA components selected: {n_components}")
```

Number of PCA components selected: 6

```
In [ ]: X_pca_selected = X_pca[:, :n_components]
```

Perform Linear Discriminant Analysis (LDA)

```
In [ ]: y_encoded = LabelEncoder().fit_transform(df['y'])

lda = LinearDiscriminantAnalysis()
lda.fit(X_pca_selected, y_encoded)
```

Out[]:

▼ LinearDiscriminantAnalysis ⓘ ?

LinearDiscriminantAnalysis()

Report the performance of LDA

```
In [ ]: lda_cv_score = cross_val_score(lda, X_pca_selected, y_encoded, cv=5)
print(f"Cross-validated accuracy score: {lda_cv_score.mean():.4f}")
```

Cross-validated accuracy score: 0.8107

```
In [ ]: y_pred = lda.predict(X_pca_selected)
accuracy = accuracy_score(y_encoded, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

Accuracy: 0.8194

Part 2. Digit Recognition

Using Traditional ML Classifiers (i.e. No Deep Learning)

In []:

```
import math
import numpy as np
import cv2 as cv
import time
from scipy import stats
from matplotlib import pyplot as plt
import skfuzzy as fuzz
import csv
import os
import pandas as pd
from sklearn.manifold import TSNE
import plotly.express as px
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score, recall_score, precision_score, f1_score

import warnings
warnings.filterwarnings('ignore')
```

Dataset Creation

In this section, starting off from a set of 10 images (one for each digit) containing 50 samples each, we extract the contours of each digit and compute their Euler's number and Hu moments to create a dataset using those properties as the describing features of each digit sample, as they are invariant to transformations.

In []:

```
class Region:
    def __init__(self):
        self.xm = 0
        self.ym = 0
        self.xx = 0
        self.yy = 0
        self.xy = 0
```

```
self.xxx = 0
self.yyy = 0
self.xxy = 0
self.xyy = 0
self.Perimetro = 0
self.Area = 0
self.closed = False
self.edgePoints = []
self.red = 0
self.green = 0
self.blue = 0
def addPoint(self,x,y,isedge,color):
    self.xm = self.xm + x
    self.ym = self.ym + y
    self.xx = self.xx + x*x
    self.yy = self.yy + y*y
    self.xy = self.xy + x*y
    self.xxx = self.xxx + x*x*x
    self.yyy = self.yyy + y*y*y
    self.xxy = self.xxy + x*x*y
    self.xyy = self.xyy + x*y*y
    self.Area = self.Area + 1
    self.red = self.red + color[0]
    self.green = self.green + color[1]
    self.blue = self.blue + color[2]

    if isedge > 0:
        self.Perimetro = self.Perimetro + 1
        if (isedge & 1) > 0: #up
            self.edgePoints.append((x+0.5,y+0.001))
        if (isedge & 2) > 0: #left
            self.edgePoints.append((x+0.001,y+0.5))
        if (isedge & 4) > 0: #down
            self.edgePoints.append((x+0.5,y+0.999))
        if (isedge & 8) > 0: #right
            self.edgePoints.append((x+0.999,y+0.5))
def addRegion(self,region):
    if (self.closed == False) and (region.closed == False) and (region.Area > 0):
        self.xm = self.xm + region.xm
        self.ym = self.ym + region.ym
        self.xx = self.xx + region.xx
        self.yy = self.yy + region.yy
        self.xy = self.xy + region.xy
        self.xxx = self.xxx + region.xxx
        self.yyy = self.yyy + region.yyy
        self.xxy = self.xxy + region.xxy
        self.xyy = self.xyy + region.xyy
        self.Area = self.Area + region.Area
```

```

        self.Perimetro = self.Perimetro + region.Perimetro
        self.red = self.red + region.red
        self.green = self.green + region.green
        self.blue = self.blue + region.blue
        for n in range(len(region.edgePoints)):
            self.edgePoints.append(region.edgePoints[n])
    def closeRegion(self):
        if (self.Area > 0) and (self.closed == False):
            self.xm = self.xm/self.Area
            self.ym = self.ym/self.Area
            self.xx = self.xx/self.Area - self.xm*self.xm
            self.yy = self.yy/self.Area - self.ym*self.ym
            self.xy = self.xy/self.Area - self.xm*self.ym
            self.red = self.red/self.Area
            self.green = self.green/self.Area
            self.blue = self.blue/self.Area
            self.closed = True

```

Function to transform the points that conform a digit and make their connection smoother (useful for visualization purposes)

```
In [ ]: def smoothPointSet(points, steps):
    lpts = len(points)
    for stp in range(steps):
        apt = np.array(points)
        for n in range(lpts):
            bidx = (n + lpts - 1) % lpts
            uidx = (n + lpts + 1) % lpts
            points[n] = ((apt[bidx][0]+apt[n][0]+apt[uidx][0])/3,(apt[bidx][1]+apt[n][1]+apt[uidx][1])/3)
    return points
```

Function to get the properties of a digit (which comes as a set of points of the same connected component)

```
In [ ]: def getInvariantFeatures(points, displayIndividual = True):
    outlinePoints = points

    # Convert the list of points to integer coordinates
    points = np.array([(int(pt[0]), int(pt[1])) for pt in outlinePoints[0]])

    xMin, yMin = np.min(points, axis=0)
    xMax, yMax = np.max(points, axis=0)

    # Create a binary image that fits the bounding box (a black BG image)
    binary_image = np.zeros((yMax - yMin + 1, xMax - xMin + 1), dtype=np.uint8)
    shifted_points = points - [xMin, yMin]
```

```

# Draw the digit in the binary image (white FG)
for x, y in shifted_points:
    binary_image[y, x] = 1

# Compute Hu Moments
moments = cv.moments(binary_image)
hu_moments = cv.HuMoments(moments).flatten()
hu1, hu2, hu3, hu4, hu5, hu6, hu7 = hu_moments

# Compute Euler's number
num_labels, labels, stats, centroids = cv.connectedComponentsWithStats(binary_image, connectivity = 8)
euler_number = num_labels - 1

# Euler's (Professor's version)
euler_number = 2 - len(points)

if displayIndividual:
    print("Hu Moments:", hu_moments)
    print("Euler's Number:", euler_number)

    # Display the generated binary image
    plt.imshow(binary_image, cmap='gray')
    plt.title("Digit Sample from the Training Set")
    plt.show()

return euler_number, hu1, hu2, hu3, hu4, hu5, hu6, hu7, xMin, yMin, xMax, yMax

```

Function to order the different points that make up a component (in this case, the dots that when together, make up a digit)

```

In [ ]: def orderPointSet(opoints, EdgeImage):
    orderpointsList = []
    if len(opoints) > 0:
        points = opoints.copy()
        neighbors = np.array([(1,0),(1,1),(0,1),(-1,1),(-1,0),(-1,-1),(0,-1),(1,-1),(0,0)])*0.5
        plen = len(points)
        n = 0
        last = 0
        inserted = 0
        orderpoints = []
        orderpoints.append(points[0])
        points.pop(0)
        lastidx = 0
        who = 0
        for inserted in range(plen-1):
            n = 0
            cpt = np.array(orderpoints[last])
            edgetype = EdgeImage[int(cpt[1]),int(cpt[0])]


```

```

dis = 1
thepoints = np.array(points)
thesum = np.sum(abs(thepoints-cpt),axis=1)
lastdir = np.sum(np.square(neigbors-neigbors[lastidx]),axis=1)
dis = 10
idx = lastidx;
for n in range(len(points)):
    if thesum[n] <= 2:
        npoints = cpt + neigbors
        theNsum = np.sum(np.abs(npoints-thepoints[n]),axis=1) + 0.01*lastdir
        for nn in range(8):
            nedgetype = EdgeImage[int(npoints[nn,1]),int(npoints[nn,0])]
            dist = theNsum[nn] + 0.001*((edgetype ^ nedgetype) > 0)
            if dist < dis:
                dis = dist
                idx = nn
                who = n
    if dis < 1.0:
        lastidx = idx
        orderpoints.append(points[who])
        points.pop(who)
    else:
        if len(orderpoints) > 2:
            orderpointsList.append(orderpoints)
        orderpoints = []
        orderpoints.append(points[0])
        points.pop(0)
        last = 0
        last = len(orderpoints) - 1
    if len(orderpoints) > 2:
        orderpointsList.append(orderpoints)
return orderpointsList

```

Function to get the segments of the image

```

In [ ]: def SegmentationRegions(image,colorimg):
regions = []
si,sj = image.shape;
fimage = image.astype(np.int16)
EdgeImage = image.astype(np.int16)

regionlabels = image[:,];
regionlabels = regionlabels.astype(np.int32)
shortlabellut = np.zeros((int(si*sj/100+1),), dtype=np.int32)
labellut = shortlabellut
upKernel = np.array([
[0, -1, 0],

```

```
[0, 1, 0],  
[0, 0, 0]), dtype=np.int16)  
upsameI = abs(cv.filter2D(fimage,-1,upKernel)) == 0  
leftKernel = np.array([  
[0, 0, 0],  
[-1, 1, 0],  
[0, 0, 0]), dtype=np.int16)  
leftsameI = abs(cv.filter2D(fimage,-1,leftKernel)) == 0  
downKernel = np.array([  
[0, 0, 0],  
[0, 1, 0],  
[0, -1, 0]), dtype=np.int16)  
downsameI = abs(cv.filter2D(fimage,-1,downKernel)) == 0  
rightKernel = np.array([  
[0, 0, 0],  
[0, 1, -1],  
[0, 0, 0]), dtype=np.int16)  
rightsameI = abs(cv.filter2D(fimage,-1,rightKernel)) == 0  
currentlabel=0  
uplabelArray = regionlabels[0,:]  
upsame = False  
leftsame = False  
thelabel = 0  
uplabel = 0  
leftlabel = 0  
regions.append(Region())  
for i in range(si):  
    leftlabel = labellut[regionlabels[i,0]]  
    for j in range(sj):  
        color = colorimg[i,j,:]  
        thelabel = leftlabel  
        upsame = False  
        leftsame = False  
        EdgeImage[i,j] = 1*(not upsameI[i,j]) + 2*(not leftsameI[i,j]) + 4*(not downsameI[i,j]) + 8*(not rightsameI[i,j])  
        if i>0:  
            uplabel = uplabelArray[j]  
            upsame = upsameI[i,j]  
            if upsame:  
                thelabel = uplabel  
        if j>0:  
            leftsame = leftsameI[i,j]  
            if leftsame:  
                if thelabel > leftlabel:  
                    thelabel = leftlabel  
            if upsame:  
                thelabel = labellut[thelabel]  
                labellut[leftlabel] = thelabel  
                labellut[uplabel] = thelabel
```

```

if (not(upsame) and not(leftsame)):
    currentlabel = currentlabel + 1
    regions.append(Region())
    thelabel = currentlabel
    if (currentlabel >= labellut.size):
        labellut = np.append(labellut, shortlabellut)
    labellut[currentlabel] = currentlabel
else:
    thelabel = labellut[thelabel]
regionlabels[i,j] = thelabel
regions[thelabel].addPoint(j,i,EdgeImage[i,j],color)
leftlabel = thelabel
uplabelArray = regionlabels[i,:]
print(len(regions))
lreg = len(regions)
changes = 1
while changes > 0:
    changes = 0
    for idx in range(lreg):
        newlabel = labellut[idx]
        if regions[idx].Area > 0 and idx != newlabel:
            changes = changes + 1
            regions[newlabel].addRegion(regions[idx])
            regions[idx].__init__()
            labellut[idx] = labellut[newlabel]
    print(changes)
closed = 0
for n in range(lreg):
    if (n == labellut[n]) and (regions[n].Area > 0):
        closed = closed + 1
        regions[n].closeRegion()
print(closed)
regionlabels = labellut[regionlabels]
return regionlabels, regions, EdgeImage

```

Definition of *sensitivity* Perimeters.

We set thresholds to limit the perimeters allowed. It is necessary because some digits have inner figures (like 8, which has 2 circles inside), so we want to filter only the most exterior contour of the digit. Also, considering that the segmentation can introduce noise and detect things in the paper that are not digits, it is necessary to set these thresholds.

```
In [ ]: PERIMETERS = [
    (300, 2000), # 0
    (100, 700),
    (150, 700), # 2
    (250, 500),
    (200, 700), # 4
    (200, 1000),
```

```
(200, 700), # 6  
(150, 700),  
(400, 700), # 8  
(200, 700)  
]
```

For each source image in the format *digit/X.jpeg*, extract the properties of each digit handwritten. There are 50 samples on each image, however, because of the thresholds set previously, in some cases, some samples won't be added, however, they are all around 50 samples approx.

```
In [ ]:  
featuresfile = open(f"trainingDataset_AllDigits.csv", "w", newline="")  
writer = csv.writer(featuresfile)  
colnames=["Digit", "euler", "Hu1", "Hu2", "Hu3", "Hu4", "Hu5", "Hu6", "Hu7"]  
writer.writerow(colnames)  
  
for digit in range(0, 10):  
  
    print("#####")  
    print()  
    print(f"EXTRACTION OF DIGIT {digit}")  
    print()  
    print("#####")  
  
    frame = cv.imread(os.path.join("digits", f"{digit}.jpeg"))  
    rawImg = frame.copy()  
  
    kernel = np.ones((5,5),np.uint8)  
    bkgnd = cv.dilate(frame,kernel,iterations = 1)  
    bkgnd = cv.dilate(bkgnd,kernel,iterations = 1)  
  
    frame = (bkgnd - frame)  
    frgnd = frame.copy()  
  
    kernel = np.ones((3,3),np.uint8)  
    frame = cv.dilate(frame,kernel,iterations = 1)  
  
    green = frame[:, :, 2]  
    ret, img = cv.threshold(green, 32, 100, 0)  
  
    fig, ax = plt.subplots(1, 3, figsize=(15, 5))  
    ax[0].imshow(cv.cvtColor(bkgnd, cv.COLOR_BGR2RGB))  
    ax[0].set_title("Background")  
  
    ax[1].imshow(cv.cvtColor(frgnd, cv.COLOR_BGR2RGB))  
    ax[1].set_title("Foreground (Just Numbers)")  
  
    ax[2].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))  
    ax[2].set_title("Refined Numbers")
```

```

plt.show()

# #####
labeled, regions, EdgeImage = SegmentationRegions(img,frame)

labeled = labeled % 21

img = cv.normalize(labeled, None, 0, 255, cv.NORM_MINMAX, cv.CV_8U)
im_color = cv.applyColorMap(img, cv.COLORMAP_HSV)
im_color = cv.medianBlur(im_color,3)
RGB_im = cv.cvtColor(im_color, cv.COLOR_BGR2RGB)

# #####
fig, ax = plt.subplots(1,2, figsize=(10, 5))
boundaries = []
for n in range(len(regions)-2):
    idx = n+2
    if regions[idx].Perimetro > PERIMETERS[digit][0]:
        if regions[idx].Perimetro < PERIMETERS[digit][1]:
            pointlist = orderPointSet(regions[idx].edgePoints, EdgeImage)
            smoothpoints = np.array(smoothPointSet(pointlist[0],3))
            euler, hu1, hu2, hu3, hu4, hu5, hu6, hu7, xMin, yMin, xMax, yMax = getInvariantFeatures(pointlist, displayIndividual=False)
            rowfeature = [int(digit), euler, hu1, hu2, hu3, hu4, hu5, hu6, hu7]
            writer.writerow(rowfeature)

            boundaries.append(((xMin, yMin), (xMax, yMax)))

            ax[0].plot(smoothpoints[:,0],smoothpoints[:,1])

ax[0].set_ylim(frame.shape[0], 0)
ax[0].set_xlim(0, frame.shape[1])
ax[0].set_title("Extracted Regions")

boundedImg = rawImg.copy()
for startPoint, endPoint in boundaries:
    cv.rectangle(boundedImg, startPoint, endPoint, (0,255,0), 2)
ax[1].imshow(boundedImg)
ax[1].set_title("Detected objects from original image")

plt.show()

featuresfile.close()

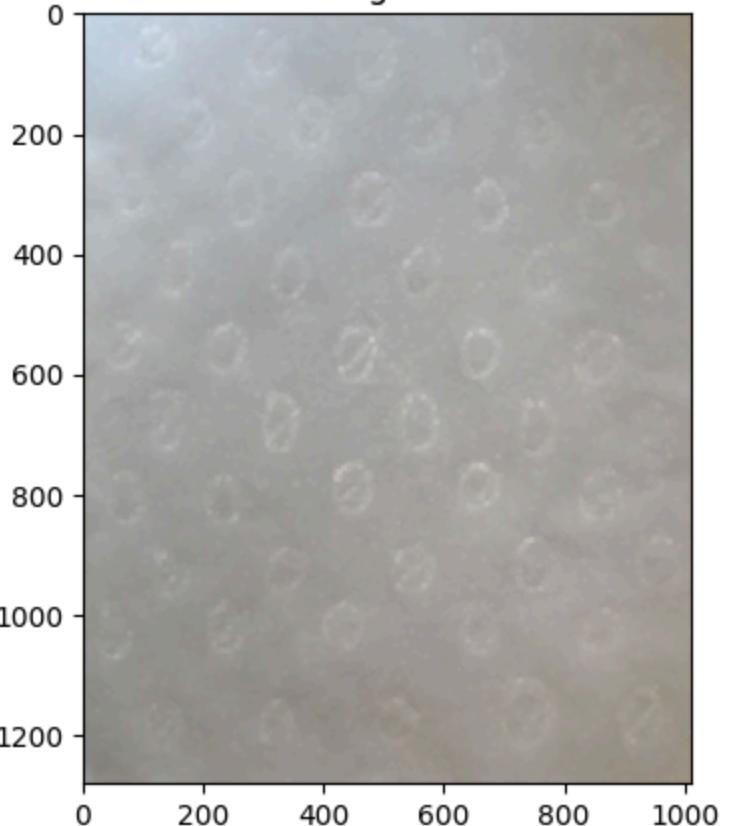
```

#####
#####

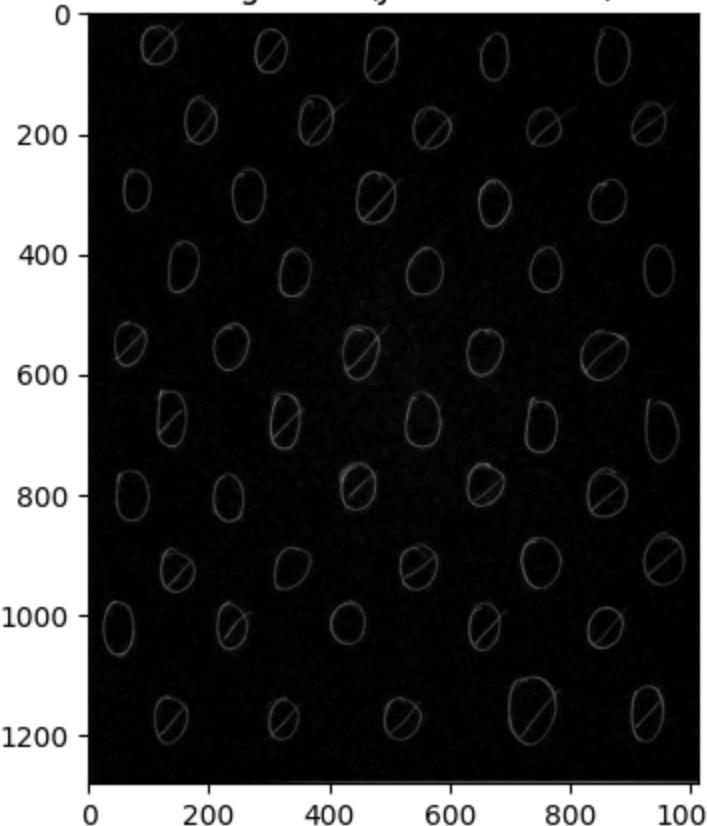
EXTRACTION OF DIGIT 0

#####
#####

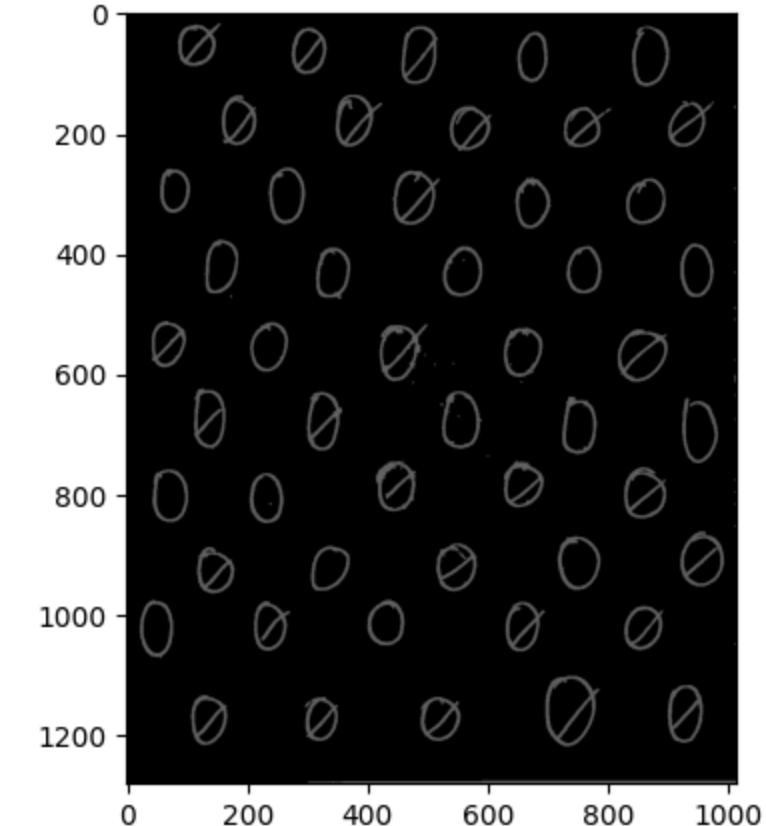
Background



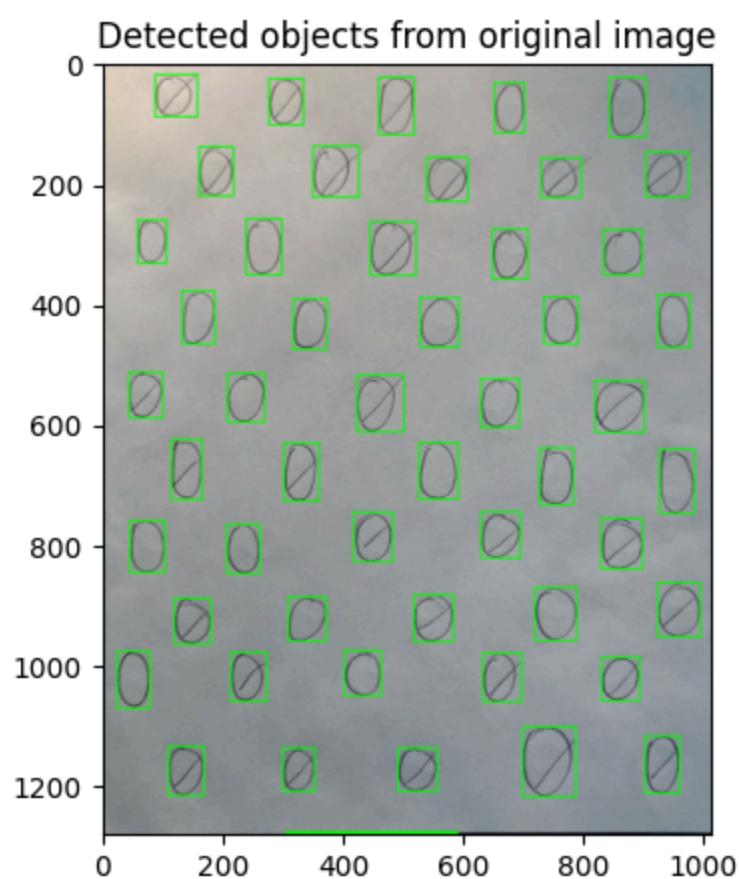
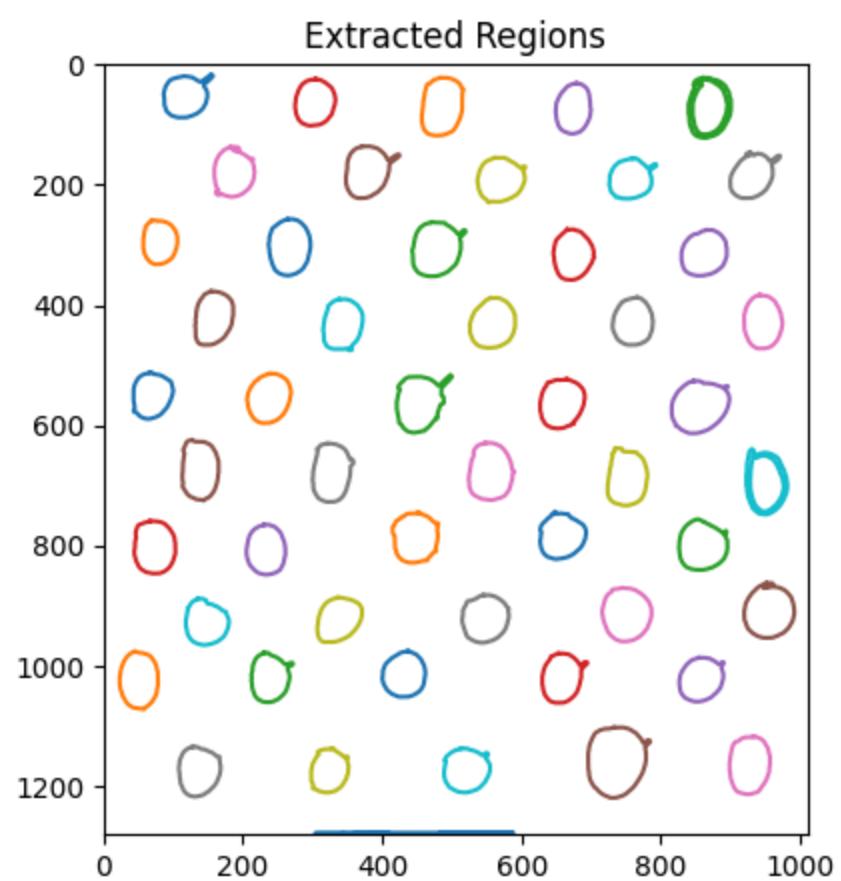
Foreground (Just Numbers)



Refined Numbers



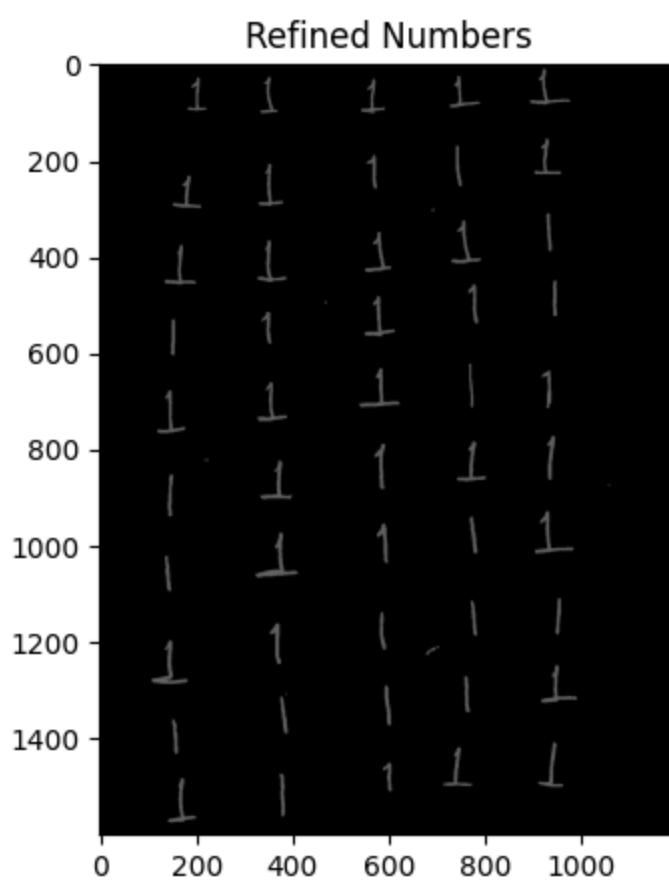
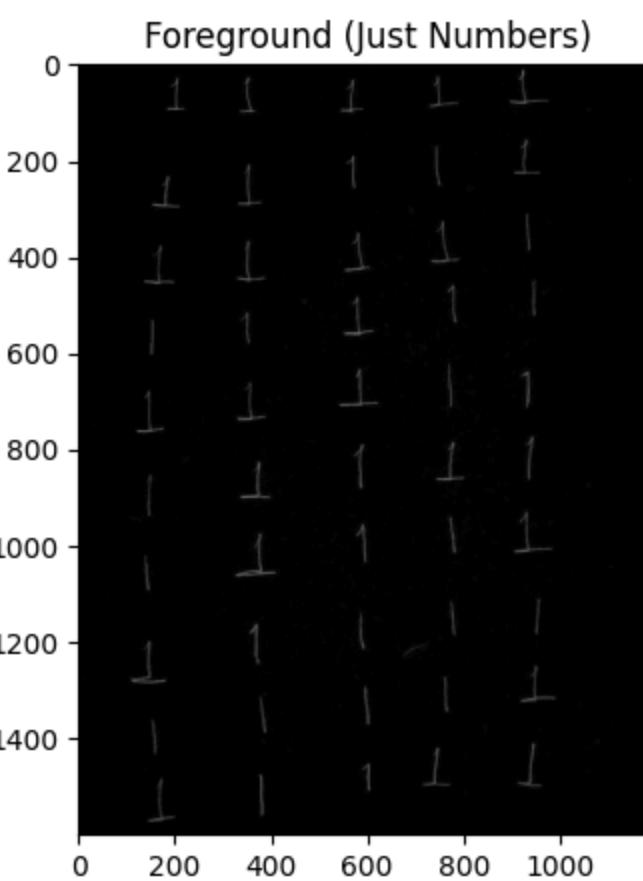
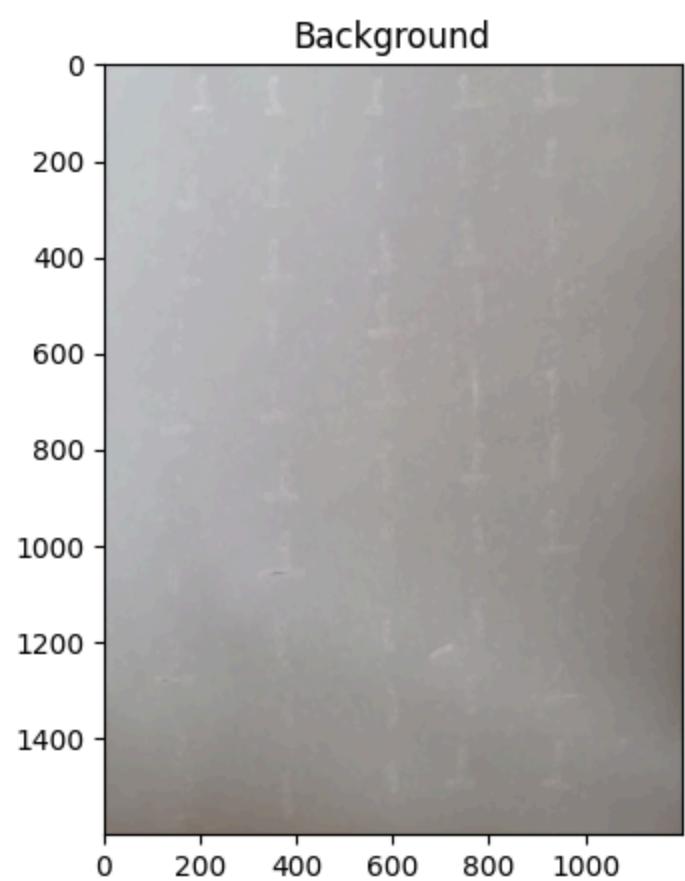
6061
5868
53
0
192



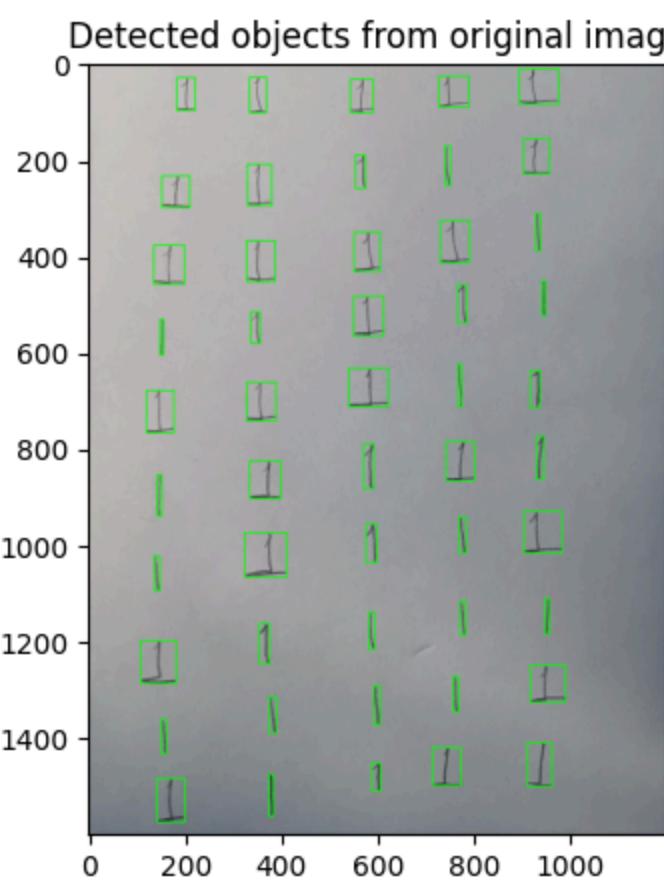
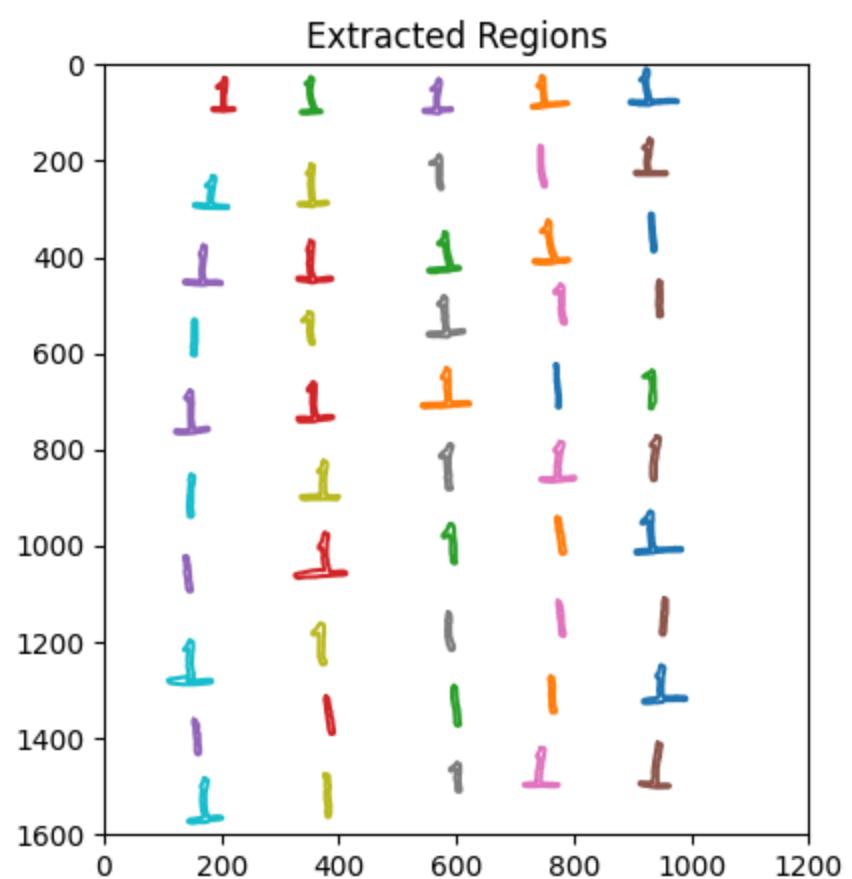
#####

EXTRACTION OF DIGIT 1

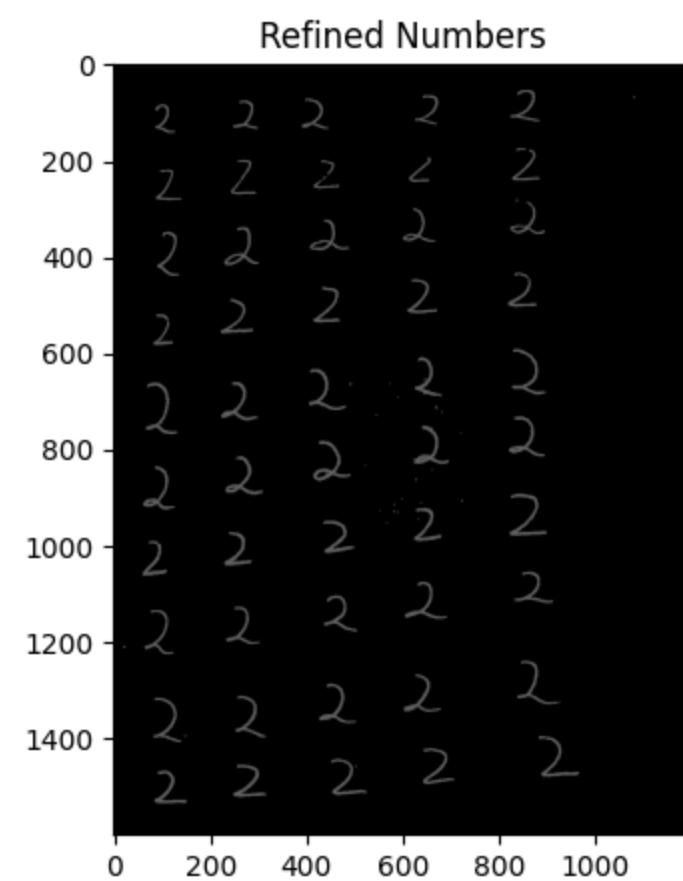
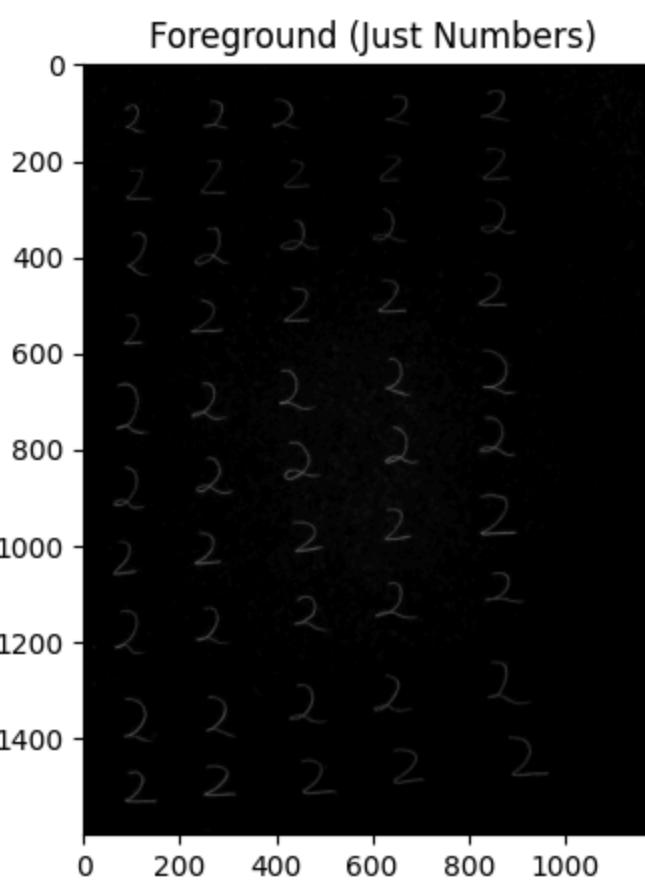
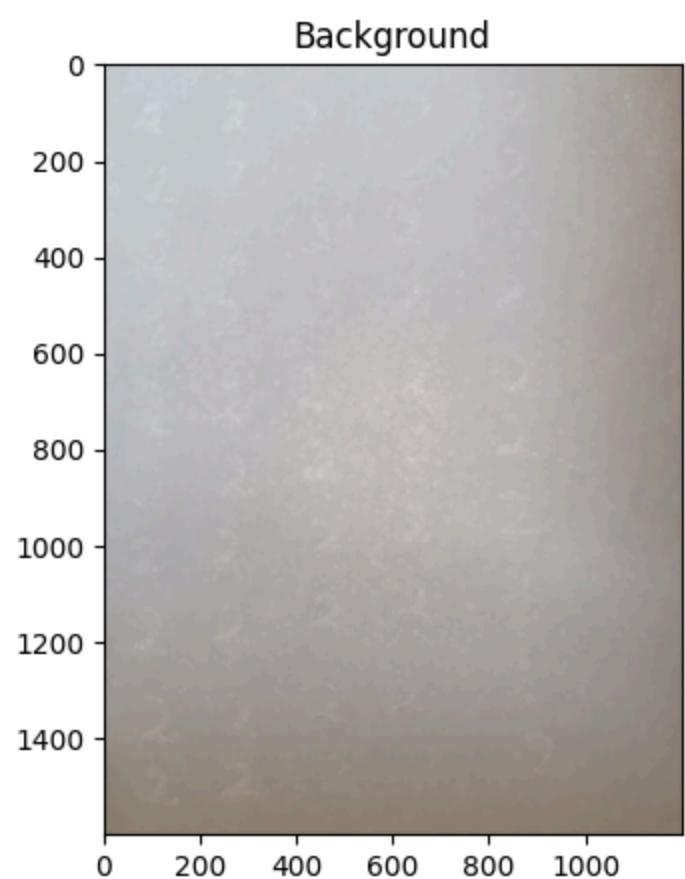
#####



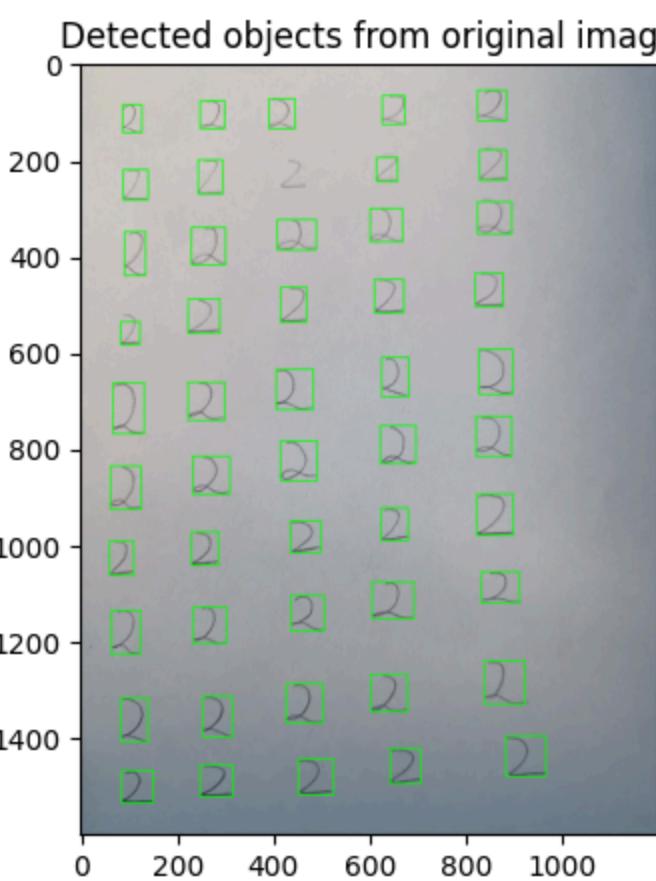
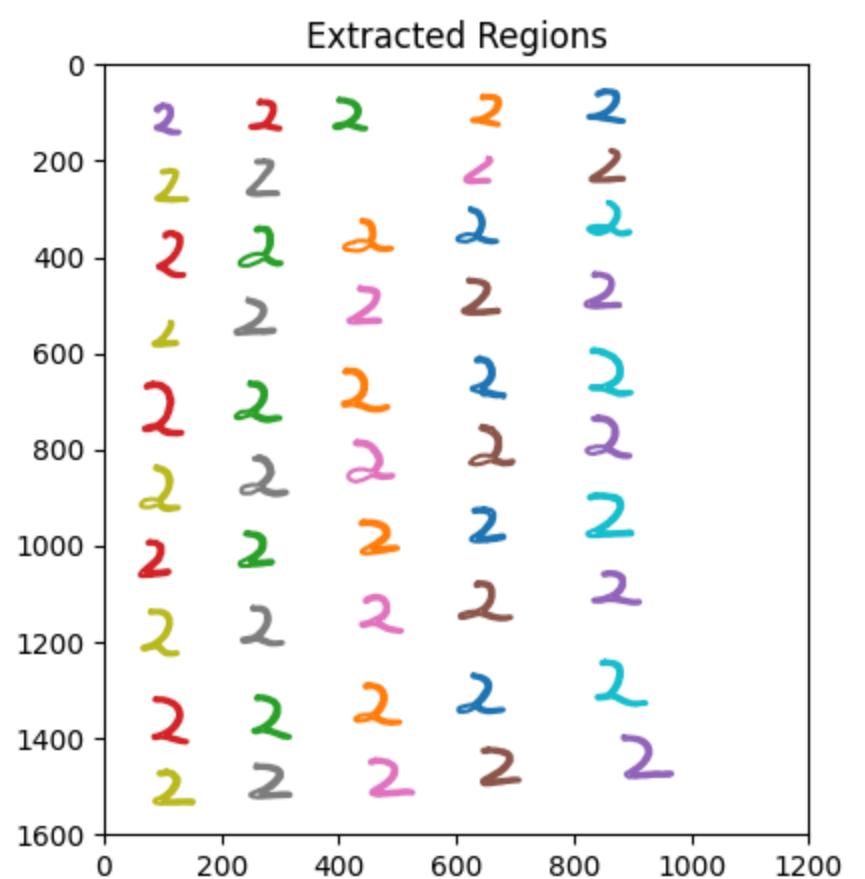
1008
944
39
0
63



#####
EXTRACTION OF DIGIT 2
#####



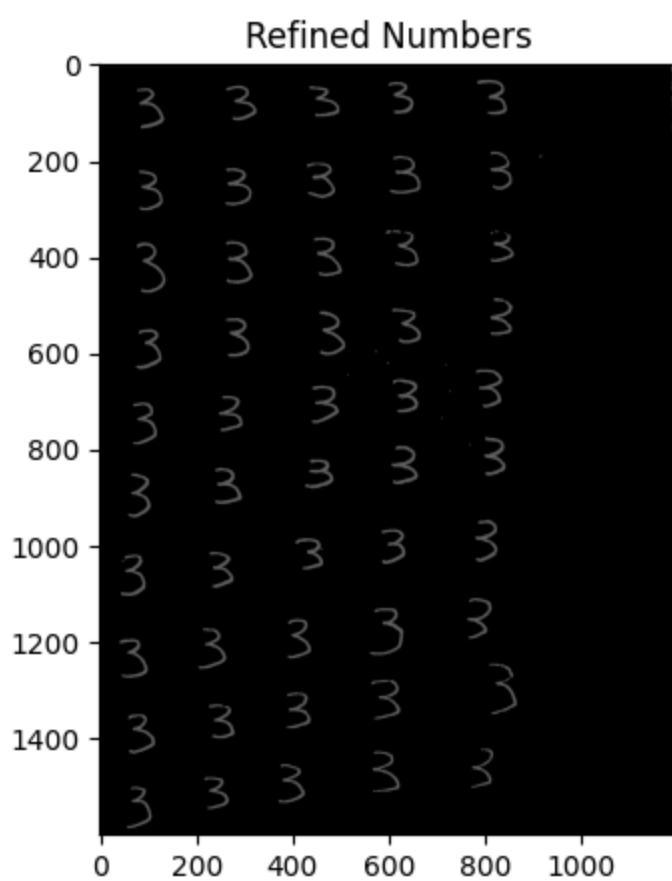
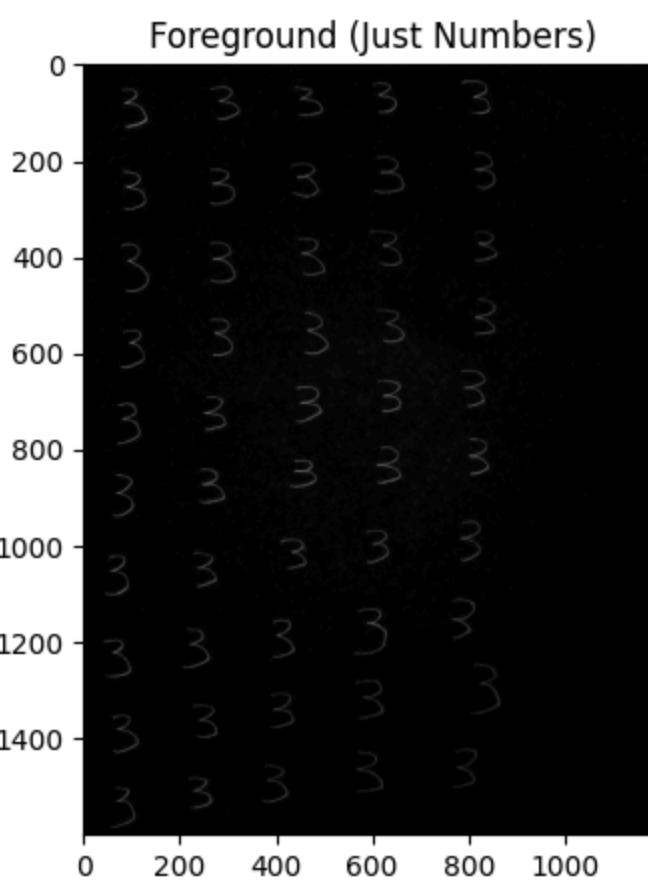
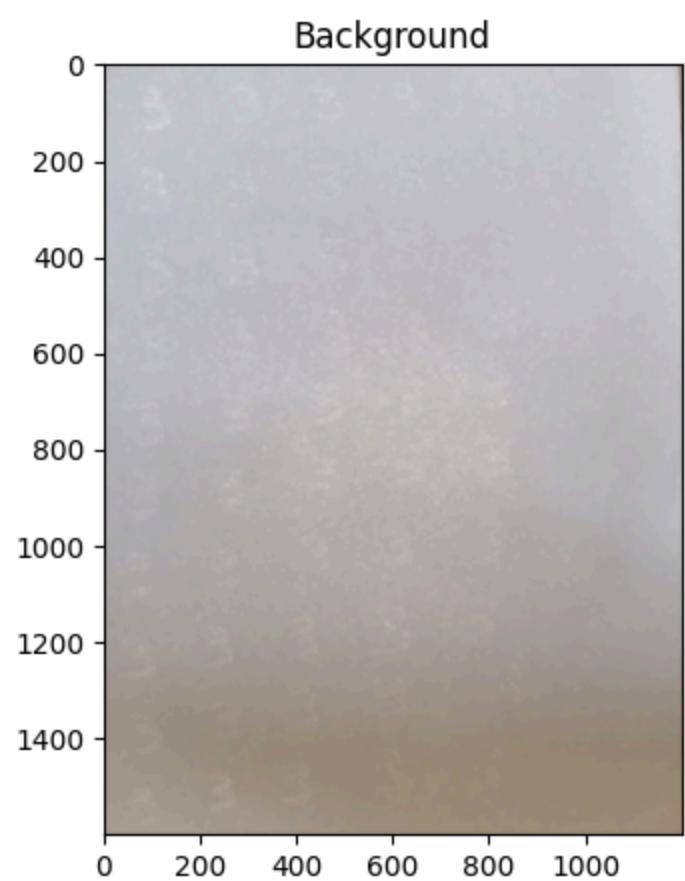
3120
3021
59
0
98



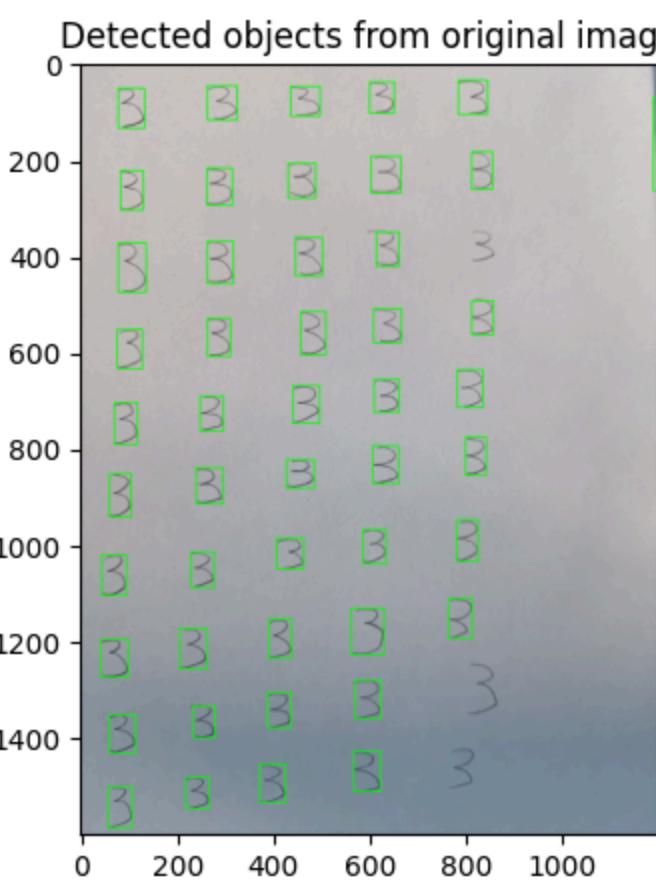
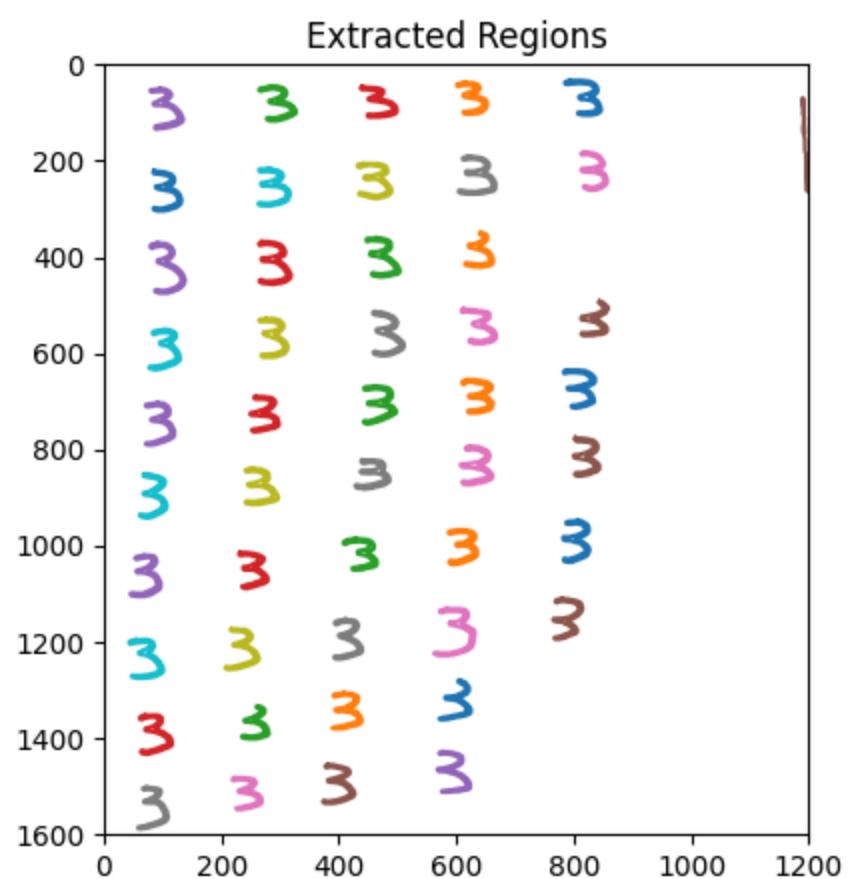
#####
#####

EXTRACTION OF DIGIT 3

#####
#####



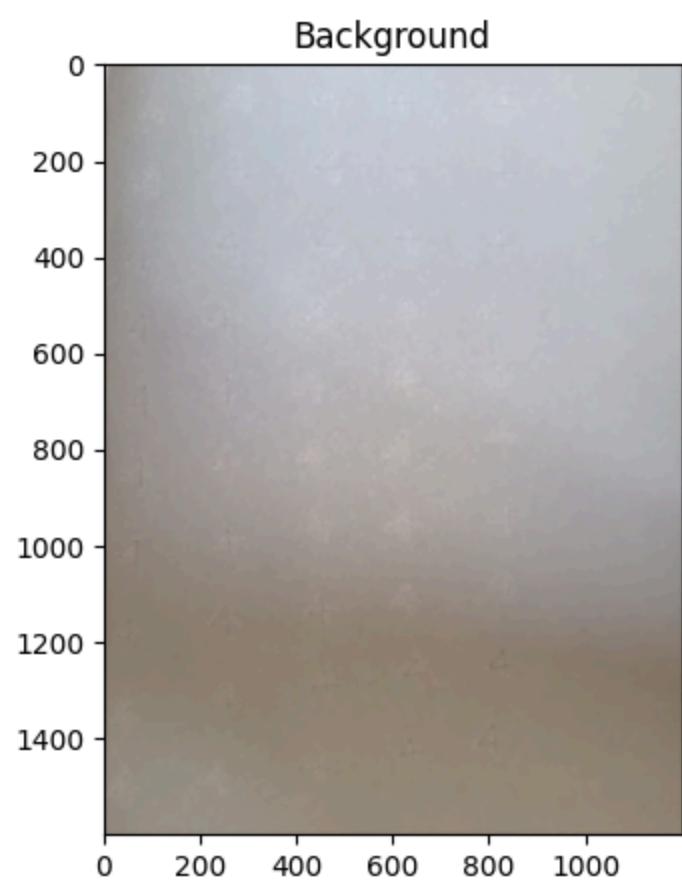
2905
2823
62
0
81



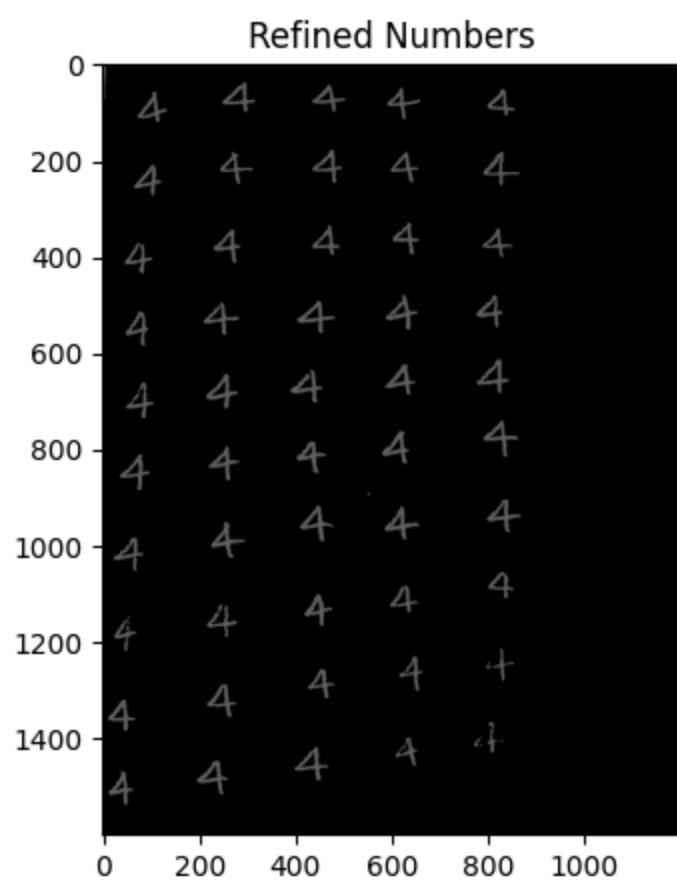
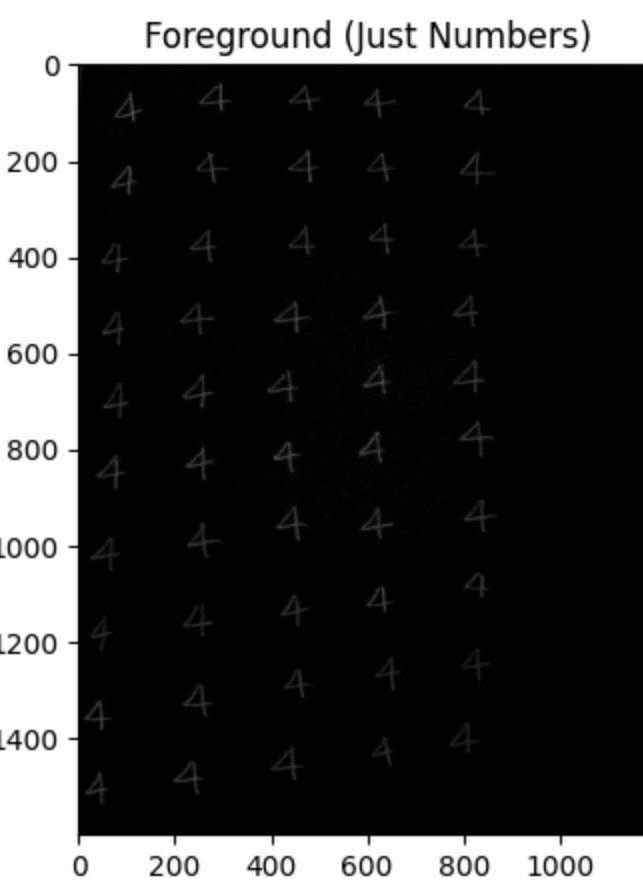
#####

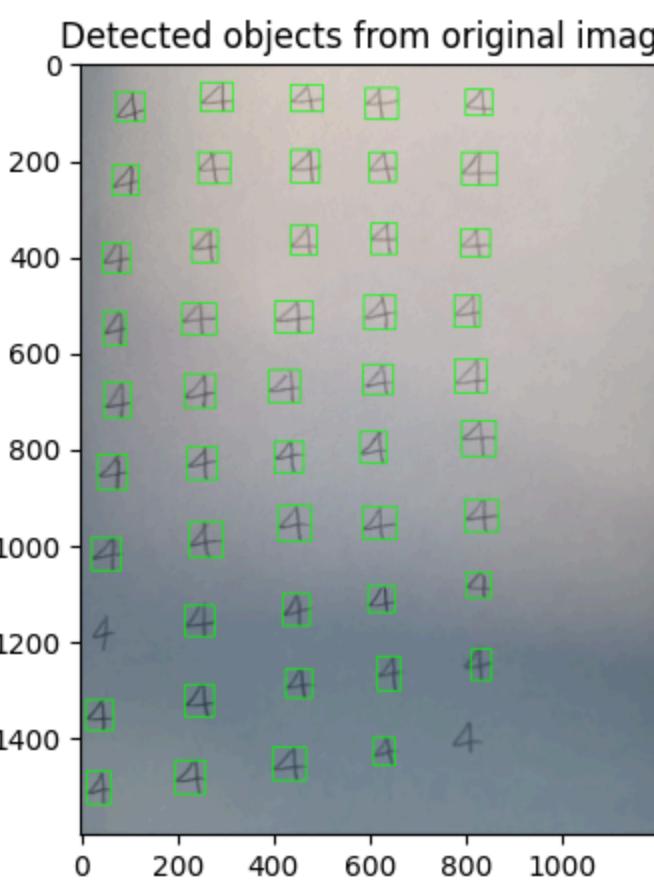
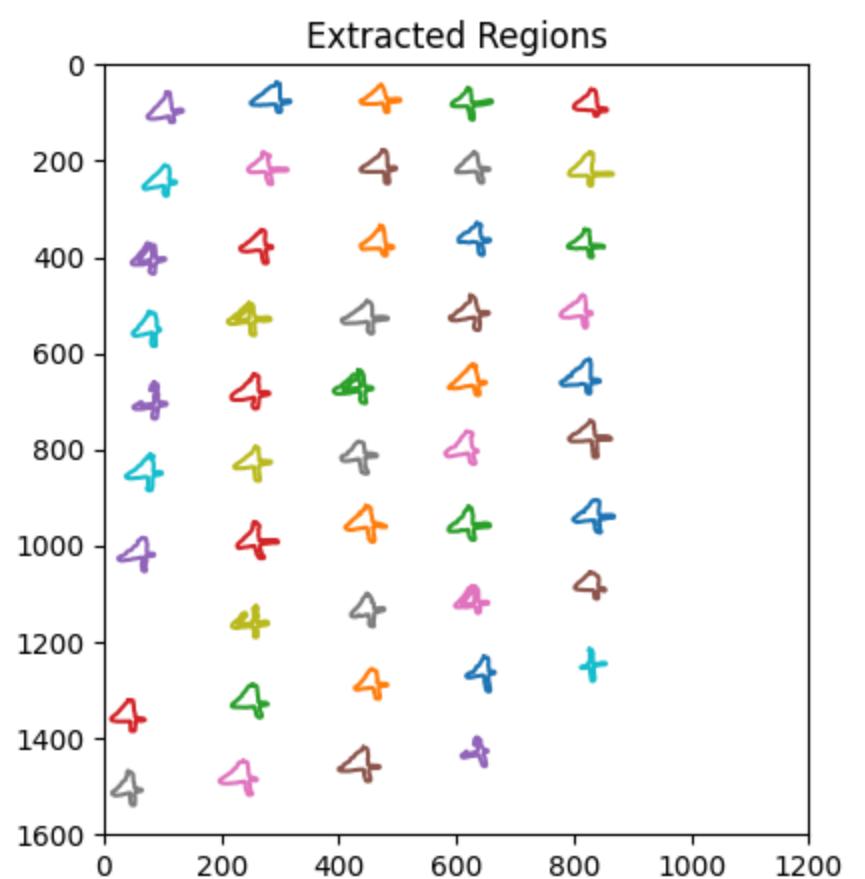
EXTRACTION OF DIGIT 4

#####



2917
2788
74
0
128

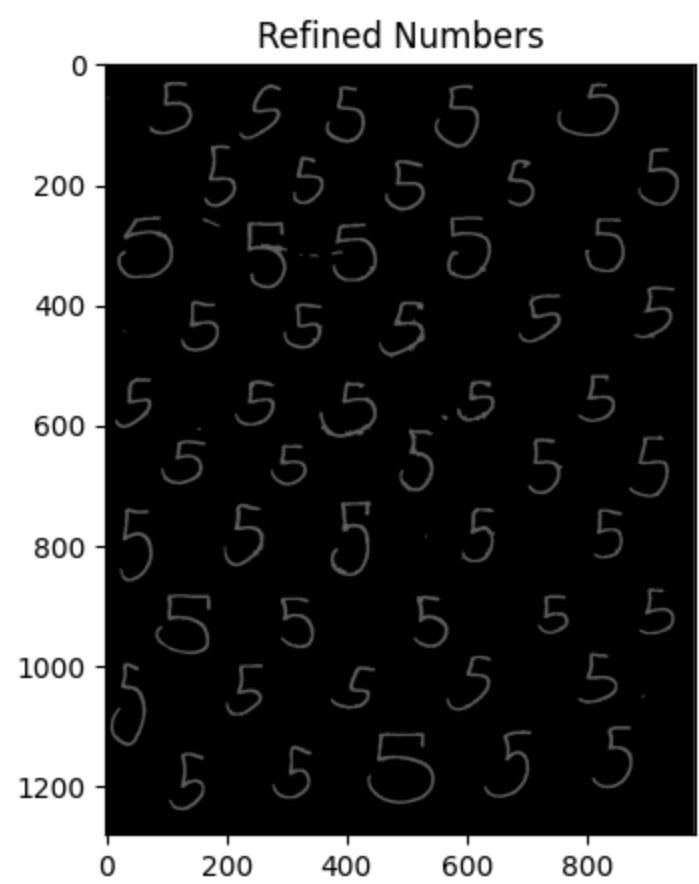
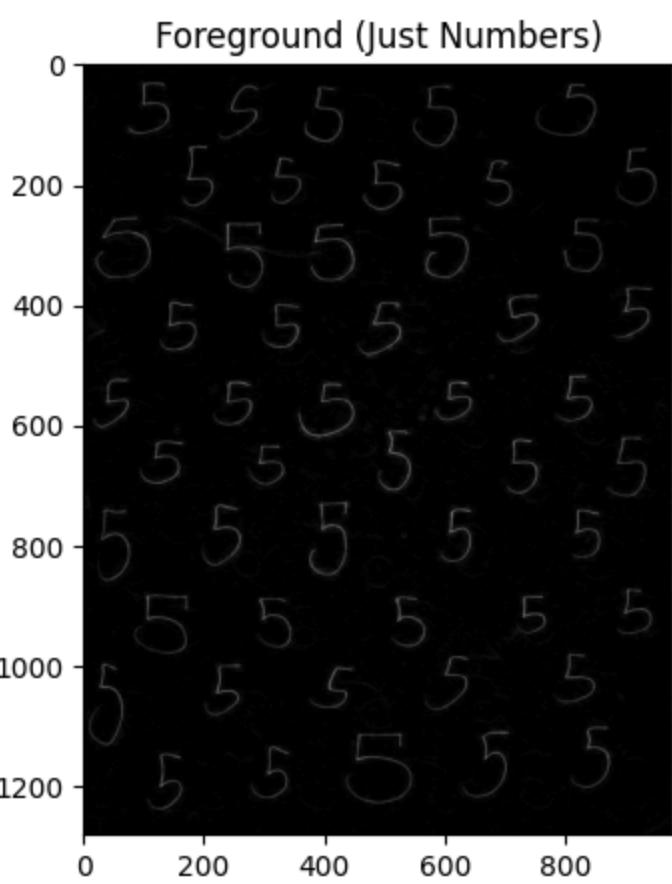
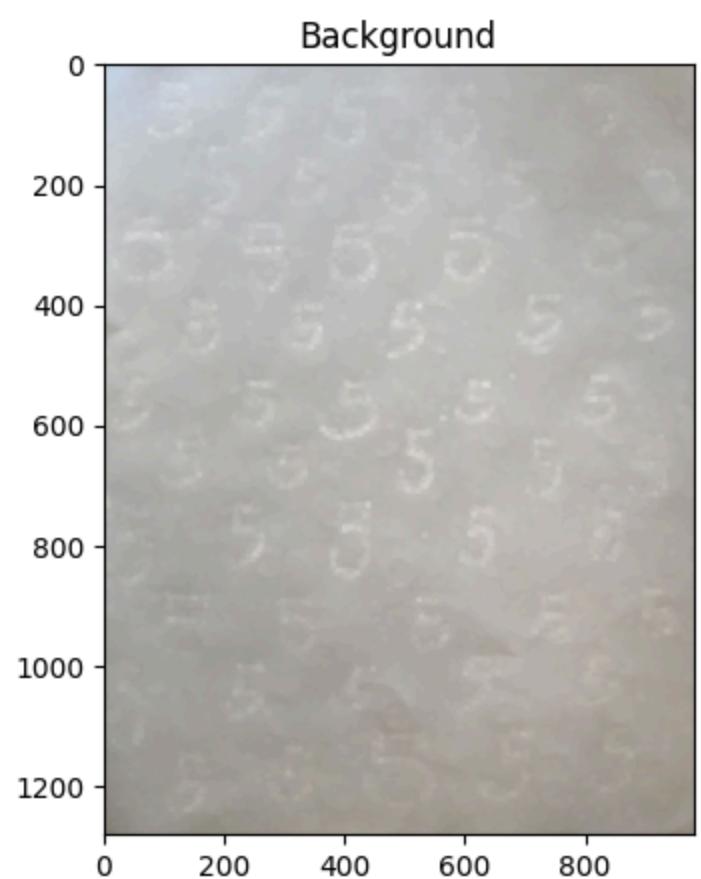




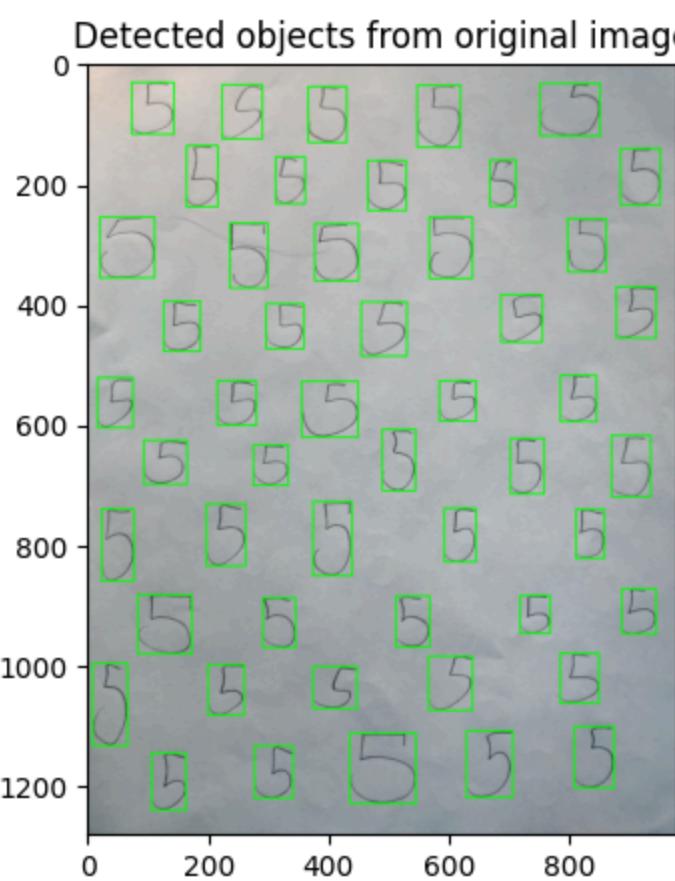
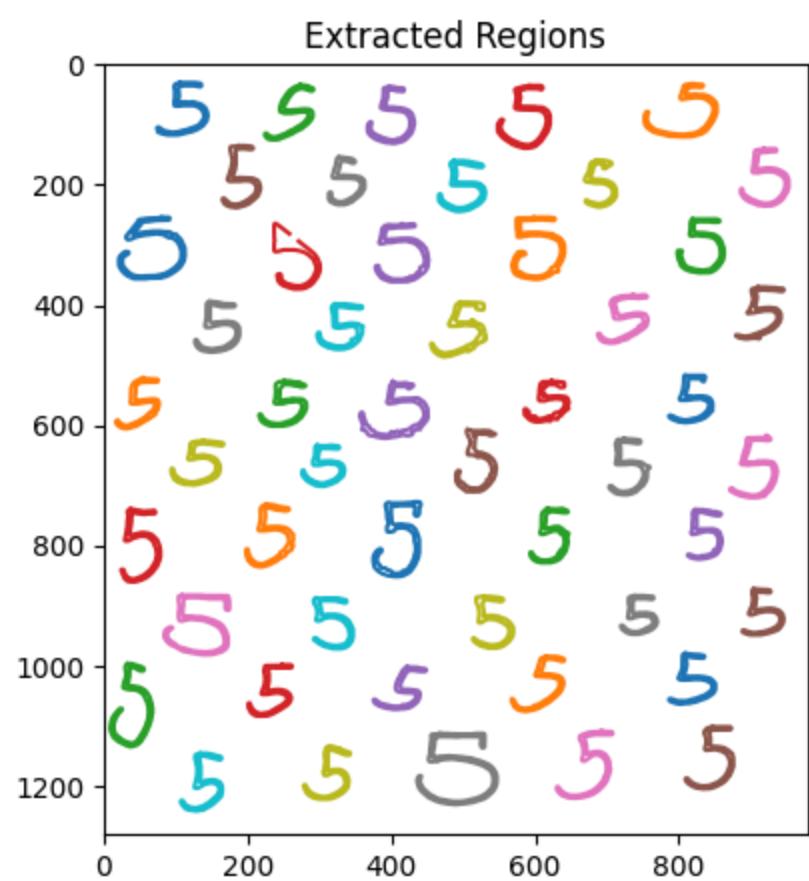
#####

EXTRACTION OF DIGIT 5

#####



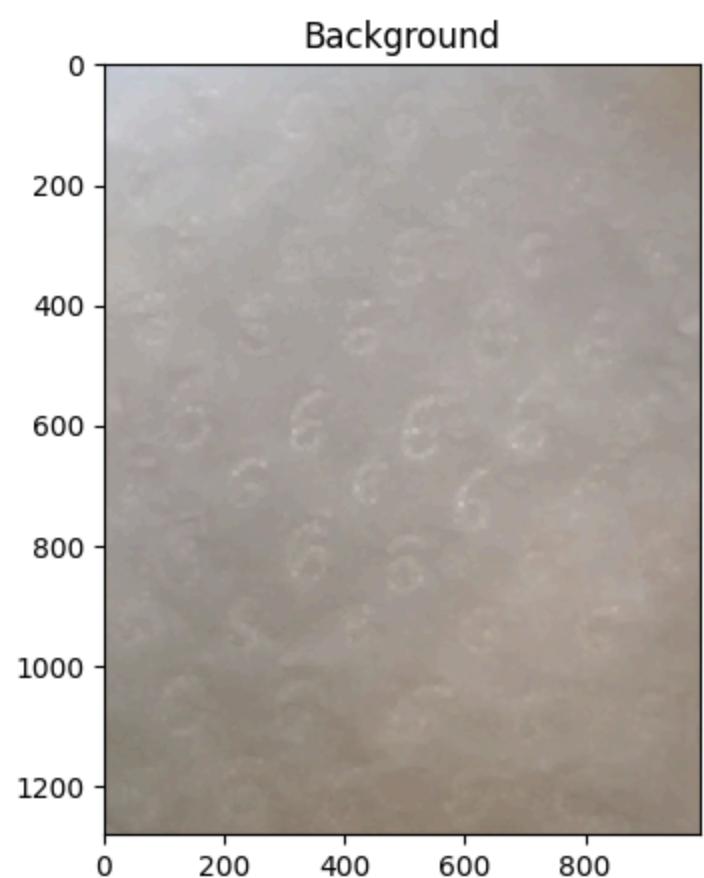
3237
3162
156
0
74



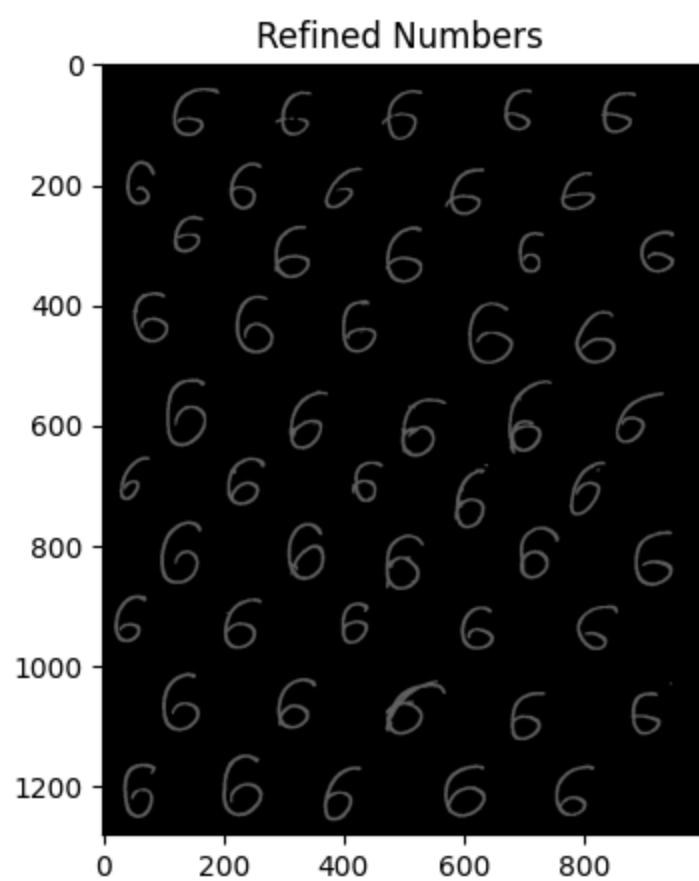
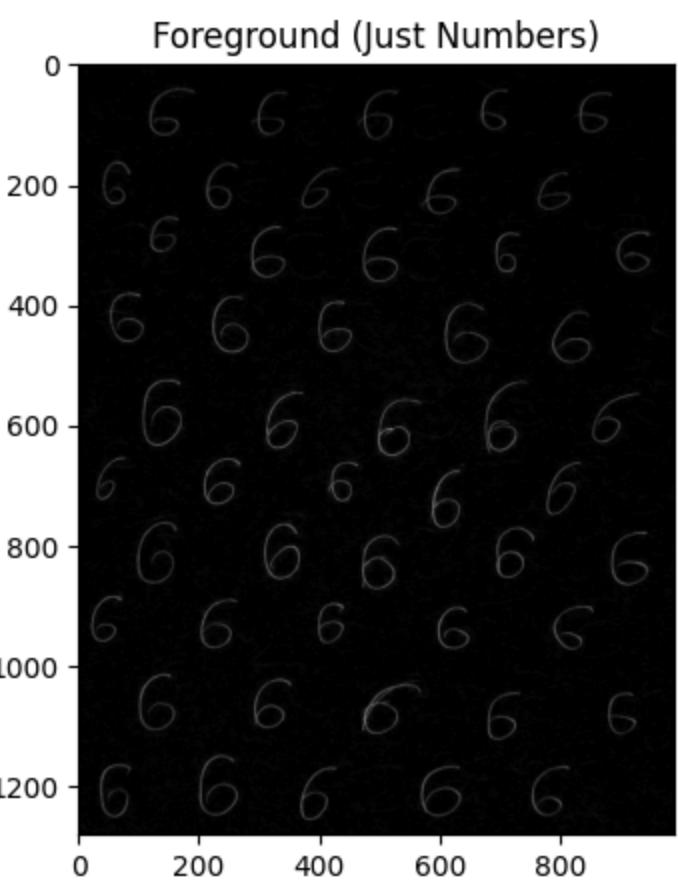
```
#####
#
```

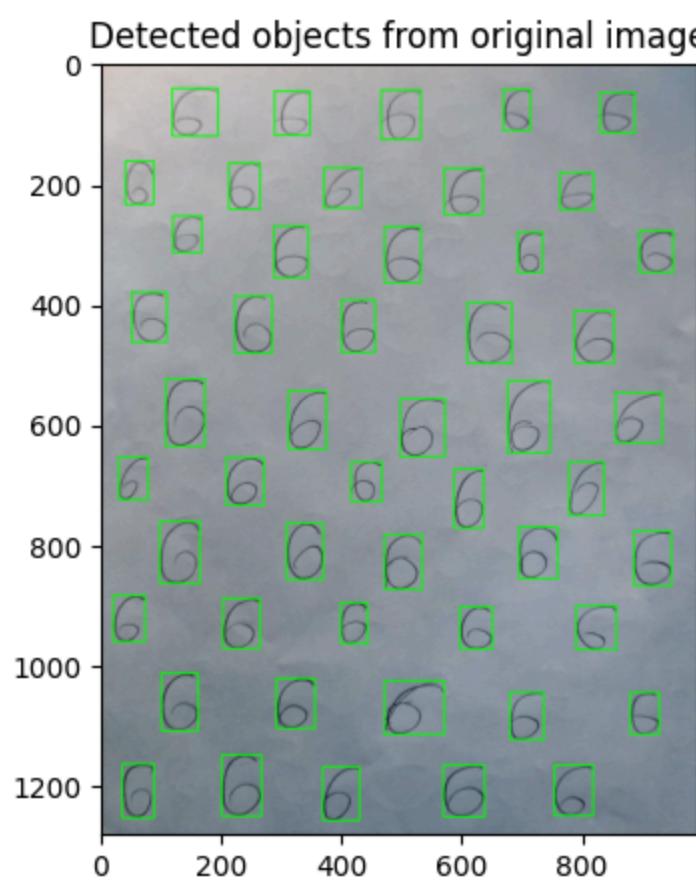
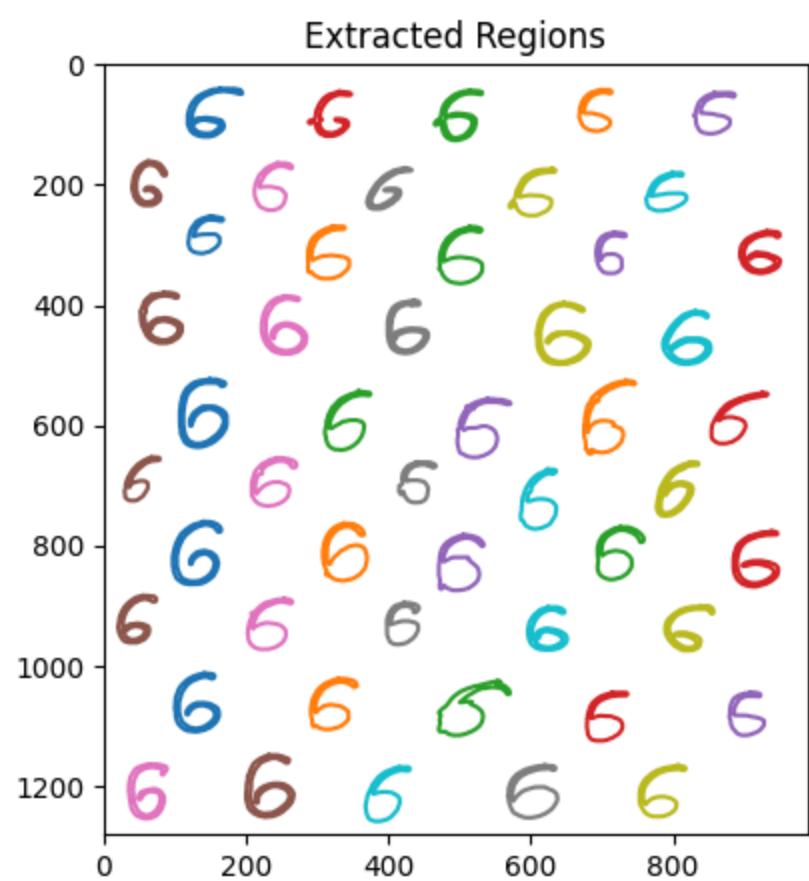
EXTRACTION OF DIGIT 6

```
#####
#
```

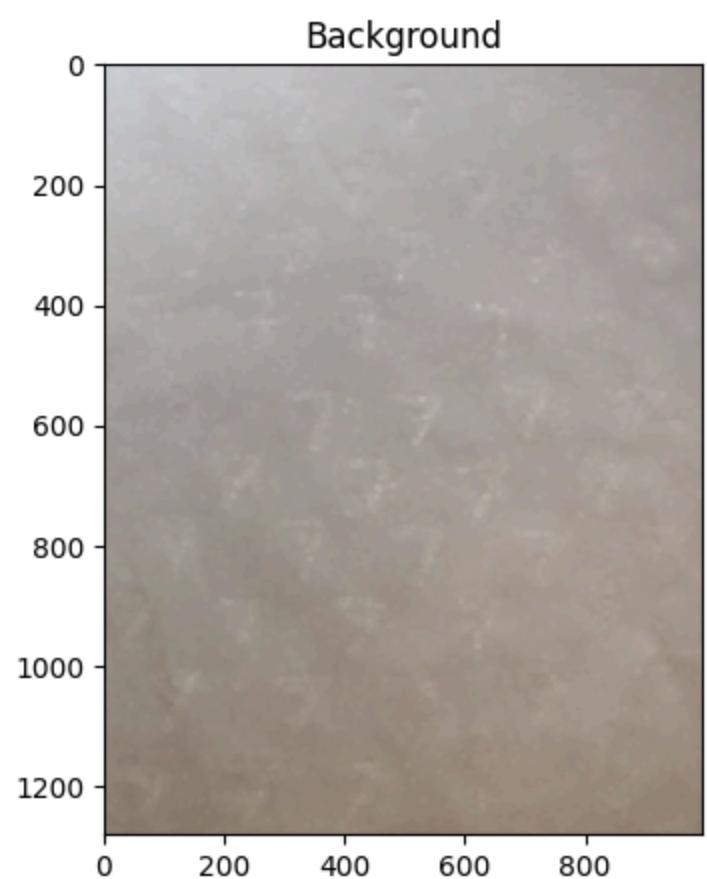


4957
4859
115
0
97

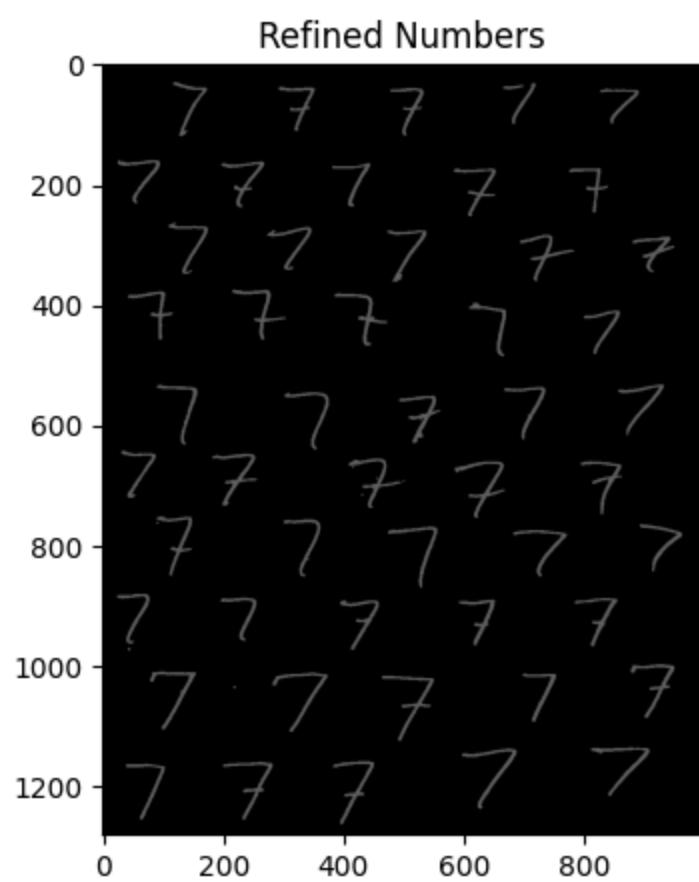
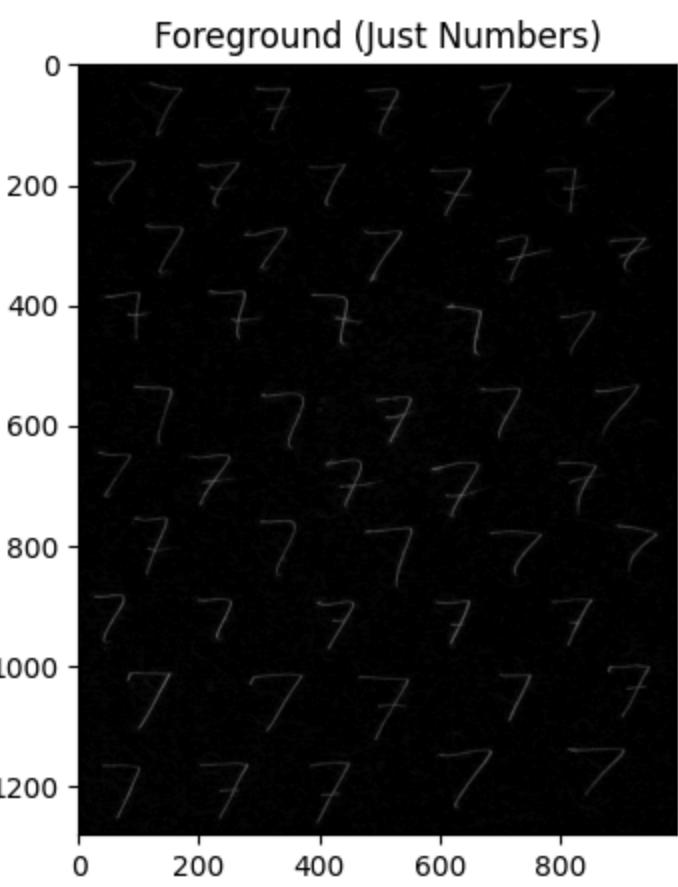


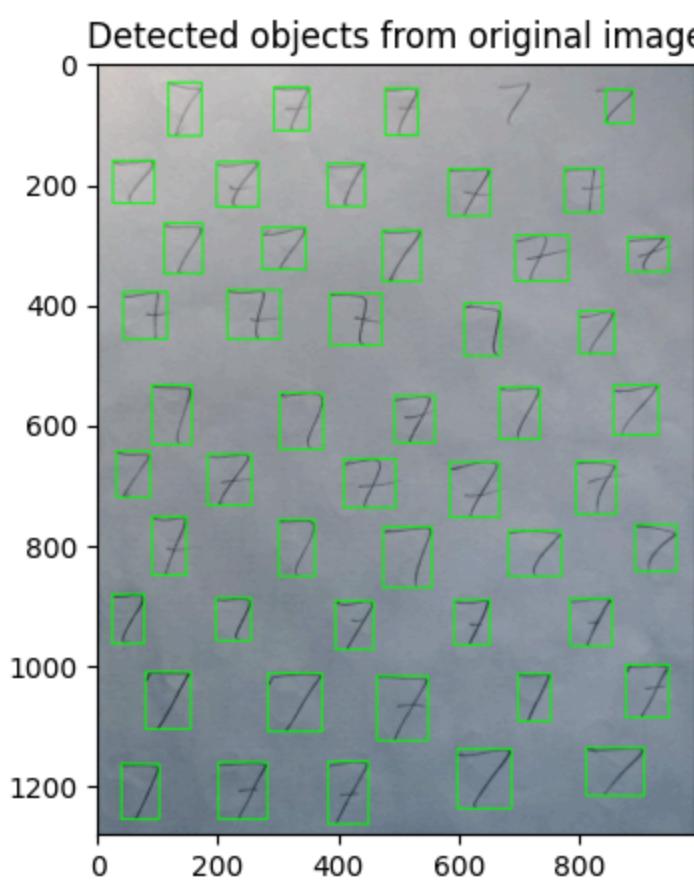
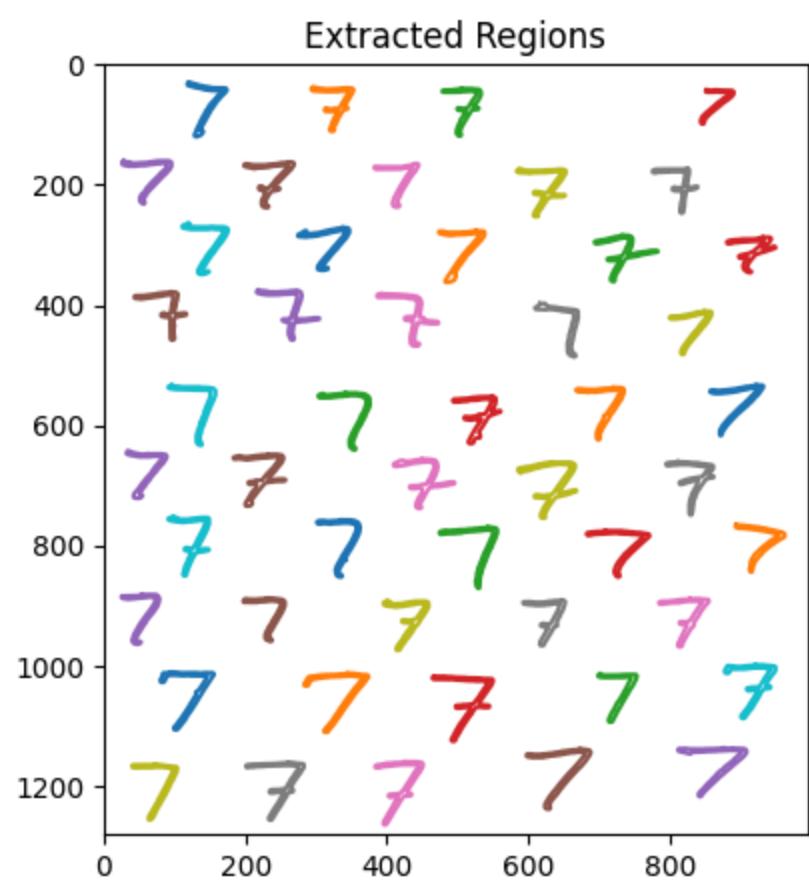


#####
EXTRACTION OF DIGIT 7
#####

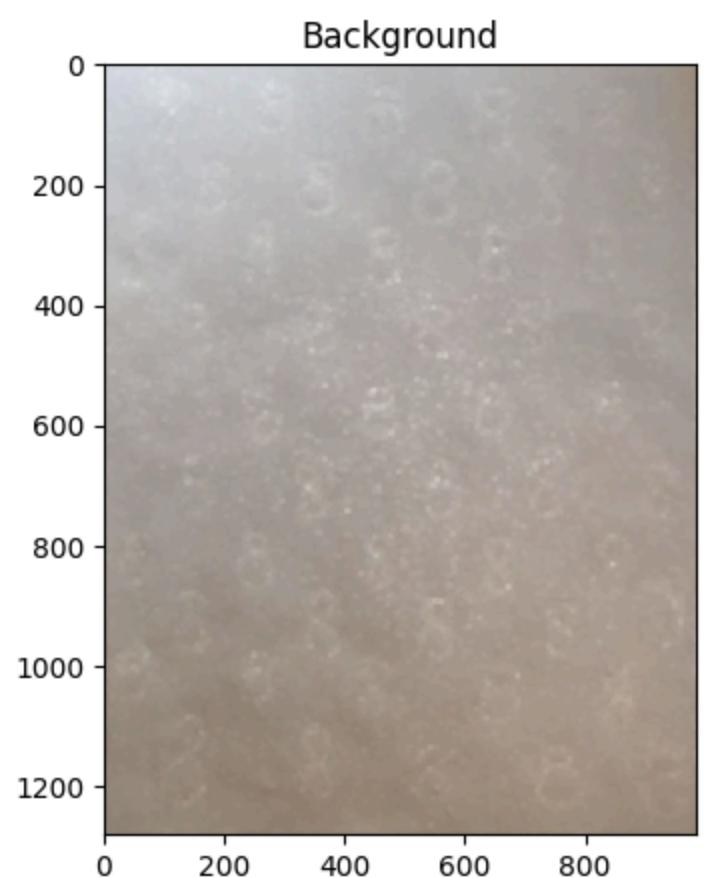


4127
4064
82
0
62

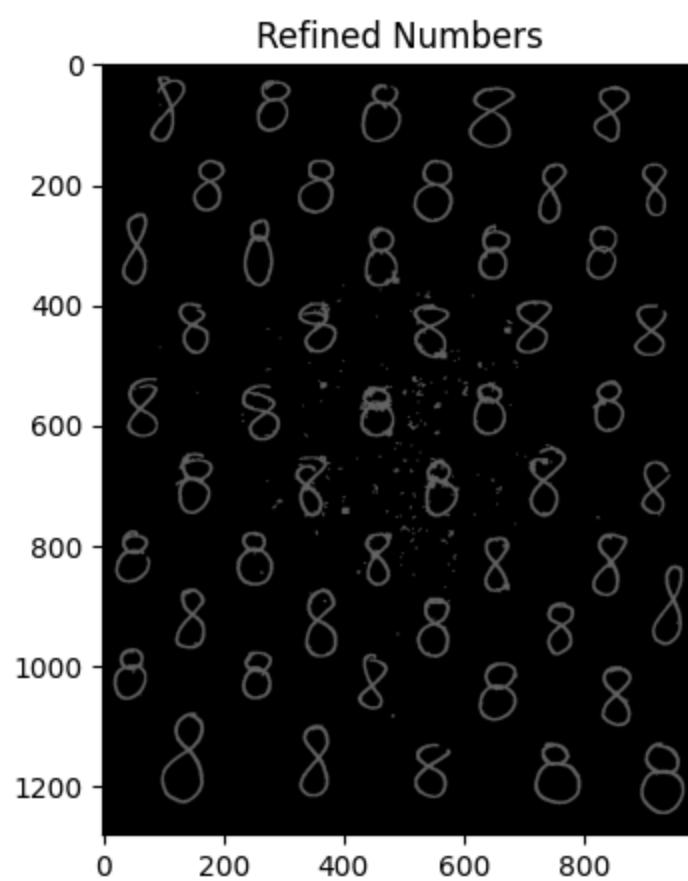
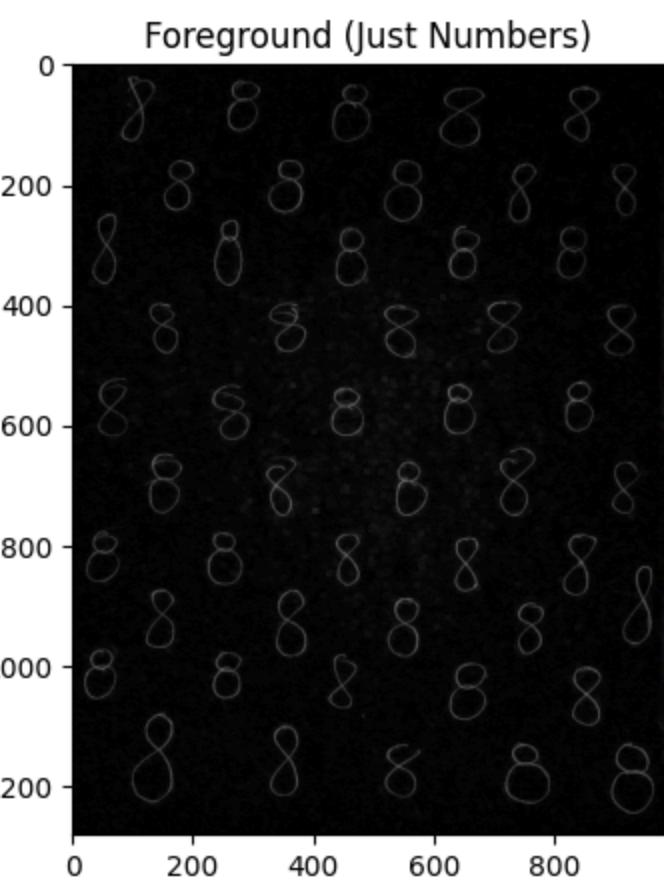


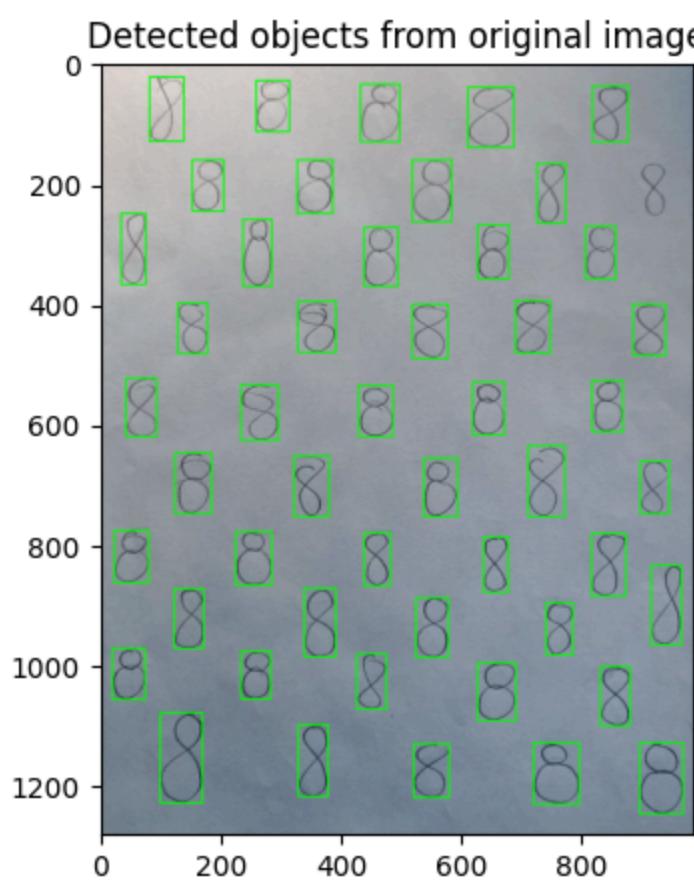
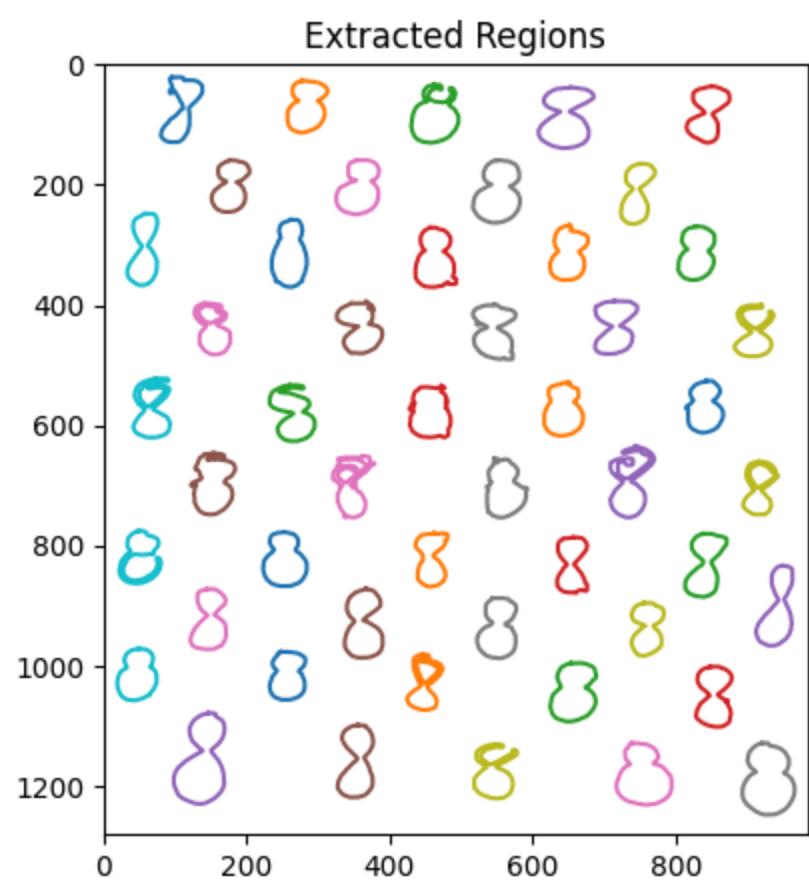


#####
EXTRACTION OF DIGIT 8
#####



5635
5293
81
0
341

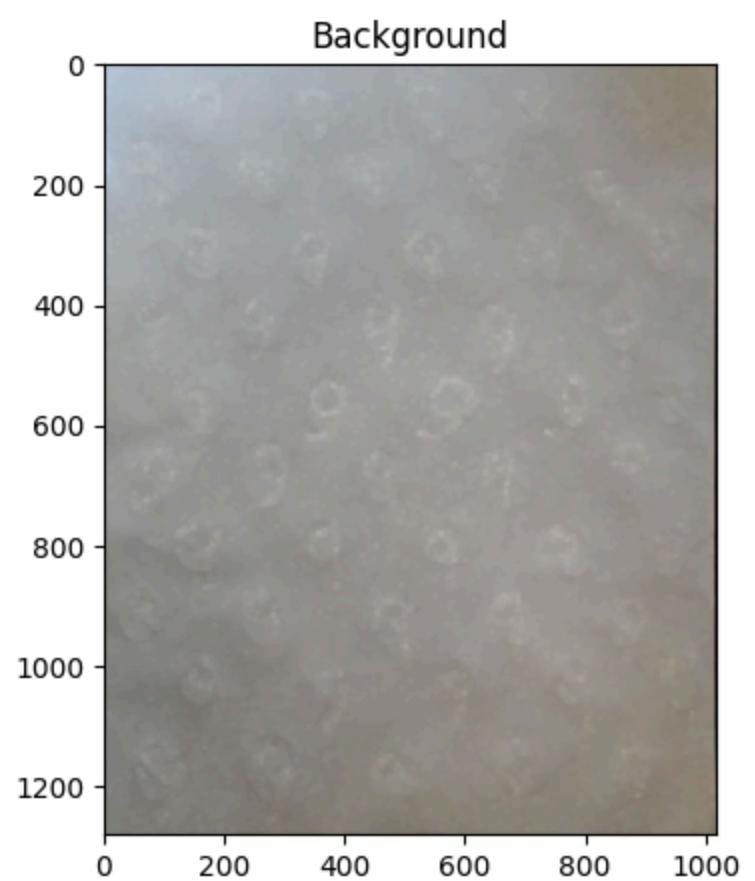




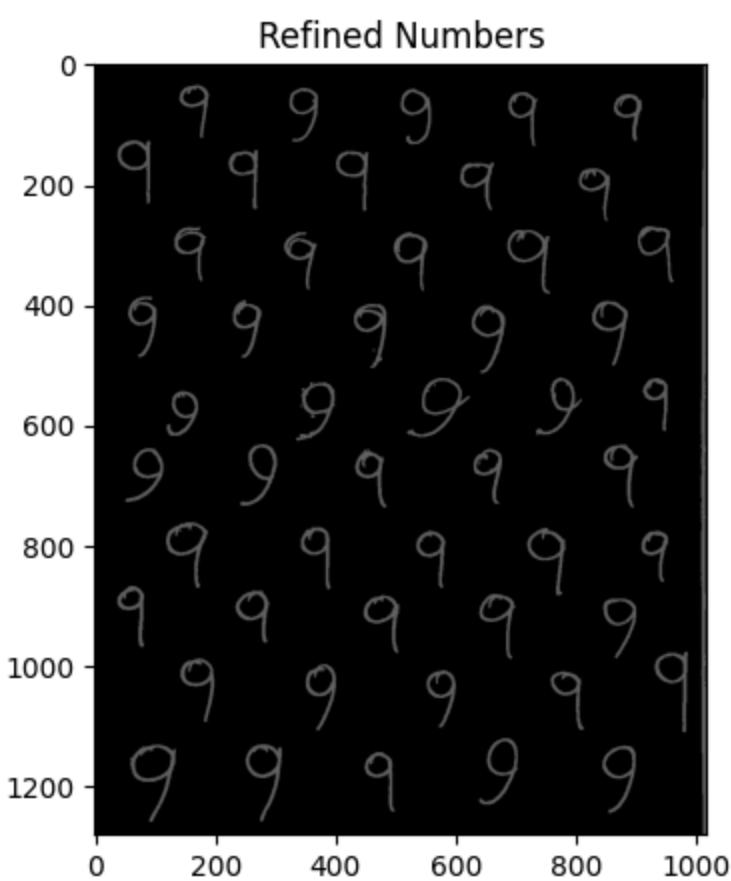
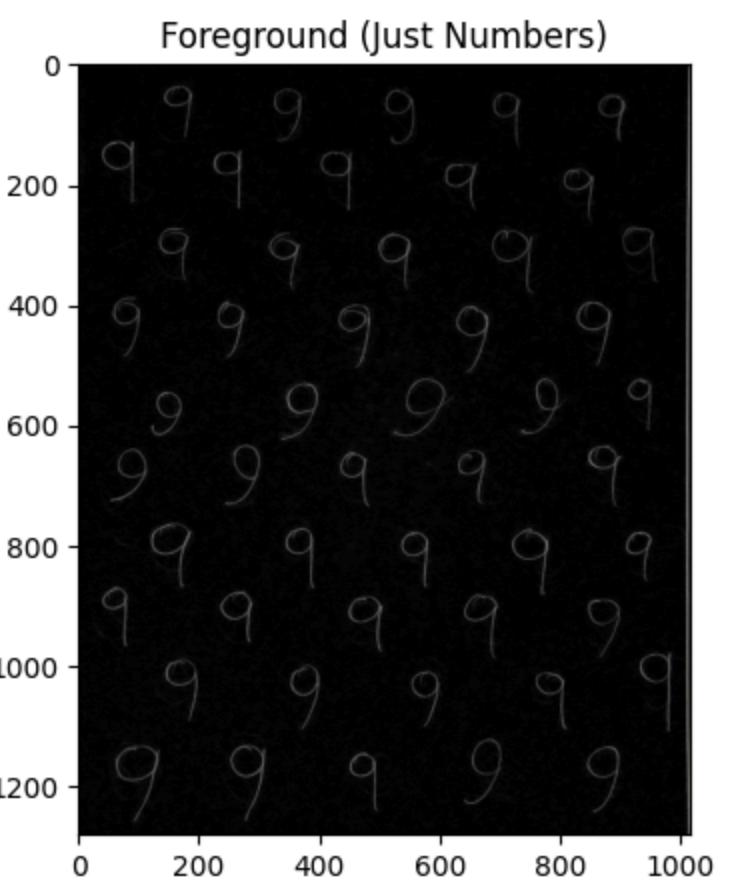
#####

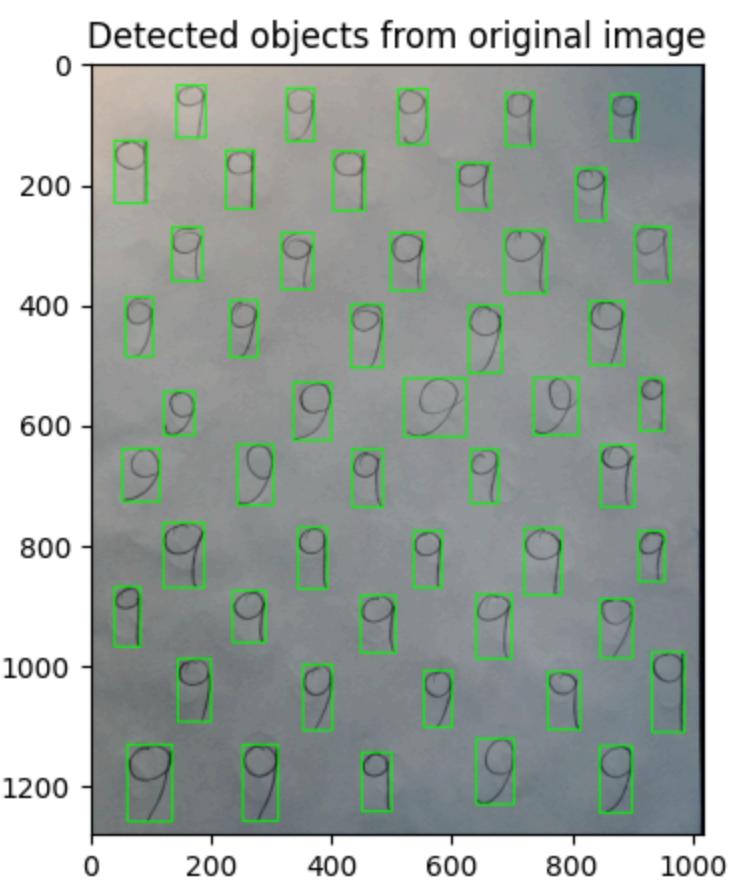
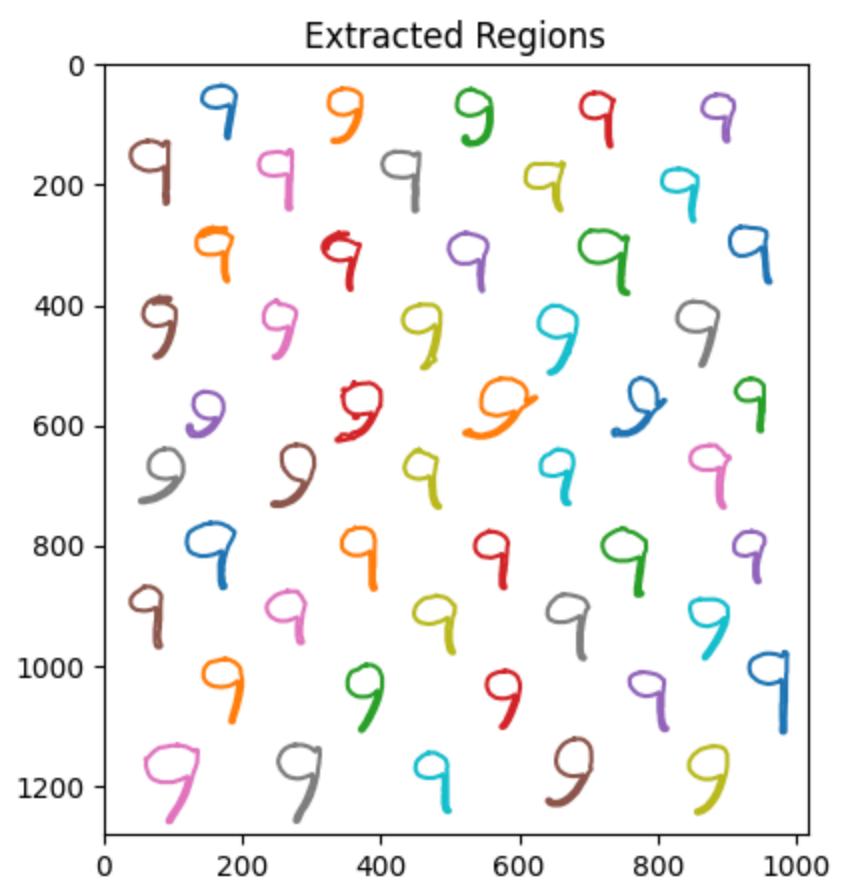
EXTRACTION OF DIGIT 9

#####



3983
3829
116
0
153





Model Training

In this section, based on the created CSV file with approximately 50 samples per digit, we train 3 different classifiers:

- Support Vector Classifier (SVC)
- Decision Trees (DT)
- Random Forest (RF)

We use our Student ID as test sets to assess the quality of recognition using this simpler approach.

```
In [ ]: df = pd.read_csv("trainingDataset_AllDigits.csv")
df
```

Out[]:

	Digit	euler	Hu1	Hu2	Hu3	Hu4	Hu5	Hu6	Hu7
0	0	-314	5.384709	1.489993	5.909982	0.759683	-0.554882	0.881351	-1.511026
1	0	-310	6.474037	5.442867	0.530058	0.013285	0.000879	0.030994	0.000685
2	0	-630	3.107118	1.435844	0.450297	0.047508	0.002103	0.056756	0.006623
3	0	-270	5.802750	1.481768	0.525297	0.012451	-0.000710	0.002768	0.000714
4	0	-270	5.633965	3.978982	0.280501	0.020723	0.001183	0.041180	0.001047
...
489	9	-446	5.136887	10.746555	10.808438	2.923951	8.201701	-3.800582	14.245162
490	9	-496	5.249713	6.847667	28.600078	1.383320	1.851842	-3.518296	8.501622
491	9	-456	5.162316	11.233403	26.504664	2.082517	-15.215564	-4.441049	-2.804761
492	9	-420	4.708165	8.068559	10.216617	1.514683	-1.135186	-4.166123	5.849352
493	9	-316	4.098544	8.365897	11.077894	0.556120	-0.610414	-1.608316	1.238018

494 rows × 9 columns

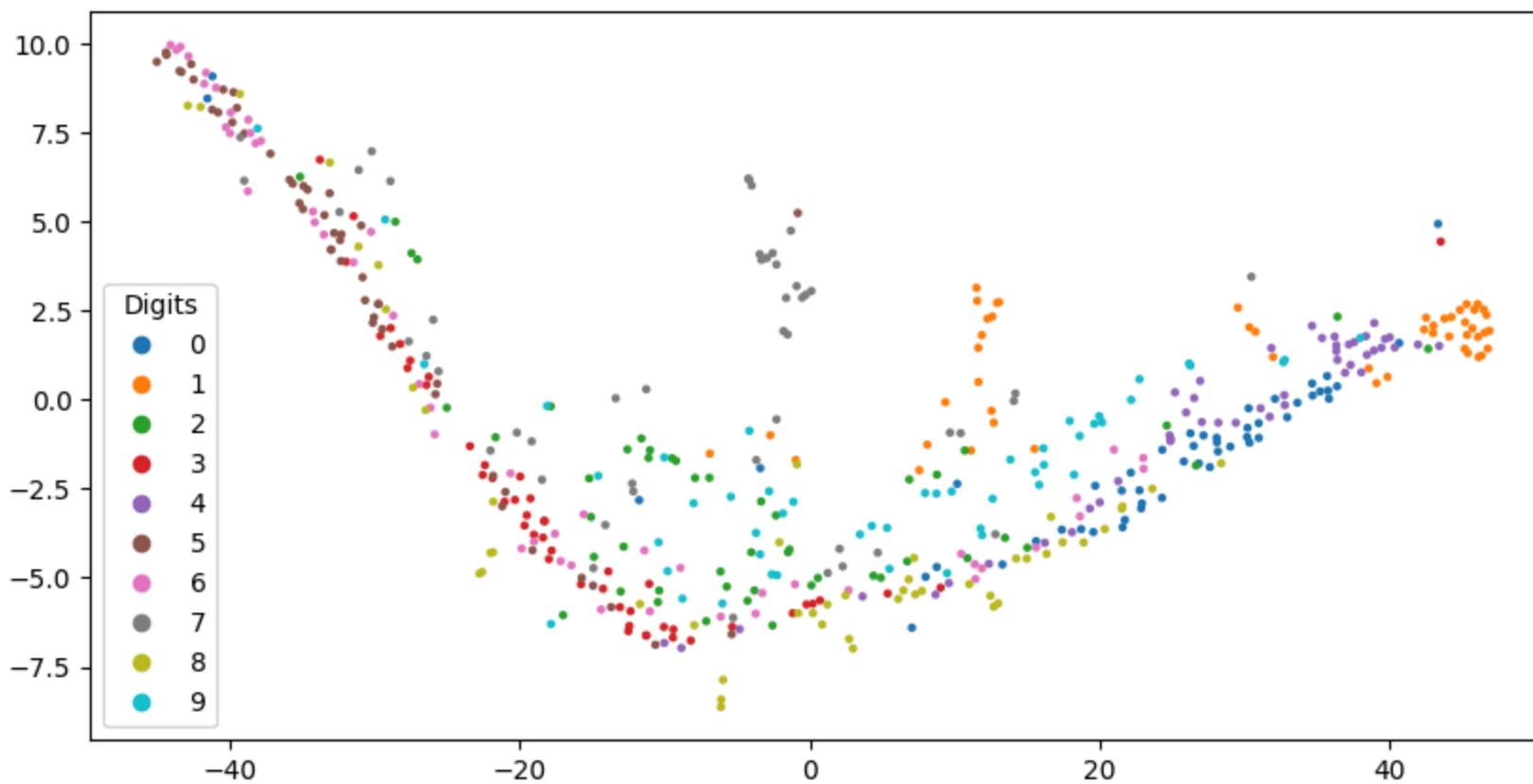
Simple dimensionality reduction visualization to get a glimpse into the separation of classes (digits), as there are some that could be easily confused (like 1 and 7).

In []:

```
tsne = TSNE(n_components=2, random_state=42)
projections = tsne.fit_transform(df.drop(columns=["Digit"]))

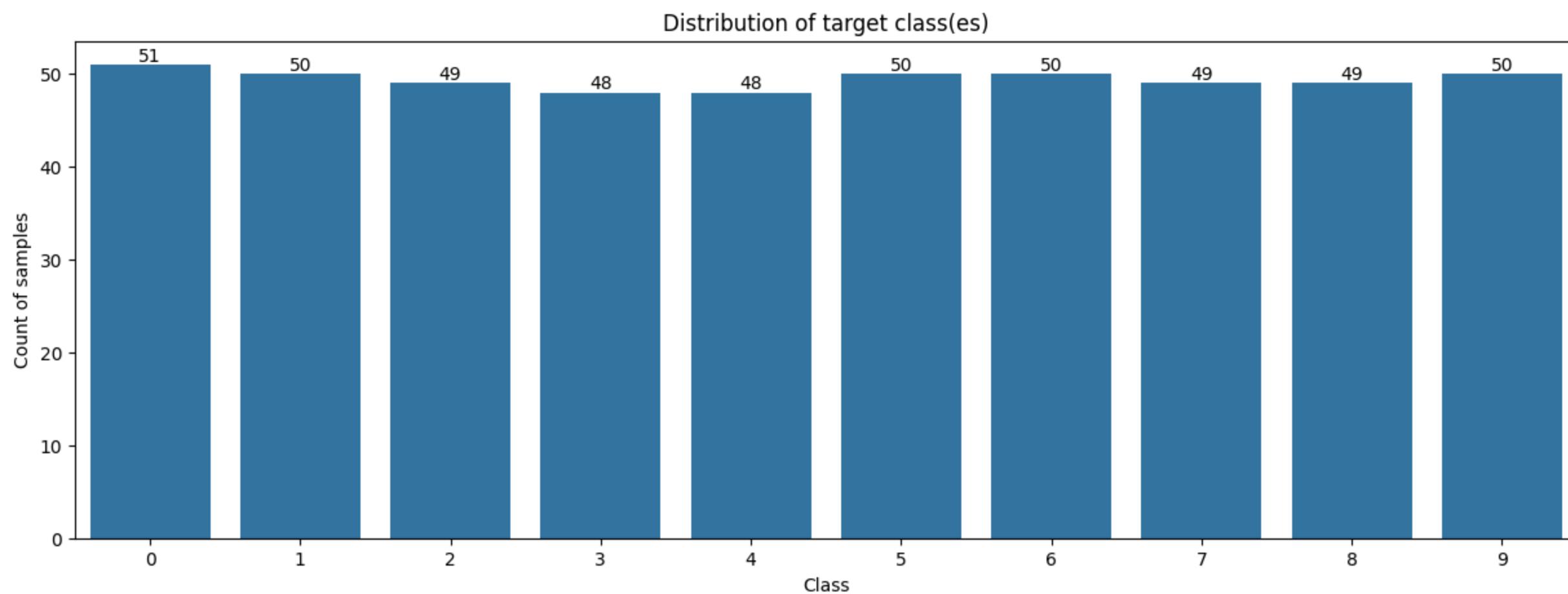
plt.figure(figsize=(10, 5))
scatter = plt.scatter(projections[:, 0], projections[:, 1], c=df["Digit"].astype(int), cmap="tab10", s=5)
plt.legend(*scatter.legend_elements(), title="Digits")
plt.title("t-SNE of our custom dataset")
plt.show()
```

t-SNE of our custom dataset



Display the class distribution, which, as expected, all classes are near 50 samples.

```
In [ ]: plt.figure(figsize=(15, 5))
graph = sns.countplot(x="Digit", data=df)
graph.bar_label(graph.containers[0])
plt.title("Distribution of target class(es)")
plt.xlabel("Class")
plt.ylabel("Count of samples")
plt.show()
```



```
In [ ]: df.unique()
```

```
Out[ ]: Digit      10
euler     191
Hu1       494
Hu2       494
Hu3       494
Hu4       494
Hu5       494
Hu6       494
Hu7       494
dtype: int64
```

Configuration of the folds (to perform K-Fold cross validation) and definition of variables to hold the results of the model training.

```
In [ ]: kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

modelsResults = {
```

```
"SVC" : {
    "Accuracy" : [],
    "AUC" : [],
    "Recall" : [],
    "Precision" : [],
    "F1Score" : []
},
"DT" : {
    "Accuracy" : [],
    "AUC" : [],
    "Recall" : [],
    "Precision" : [],
    "F1Score" : []
},
"RF" : {
    "Accuracy" : [],
    "AUC" : [],
    "Recall" : [],
    "Precision" : [],
    "F1Score" : []
}
}

X = df.drop(columns=["Digit"])
y = df["Digit"]
```

Hyperparameter Tuning

This block is to test multiple different model configurations (i.e. hyperparameters) in an attempt to maximize the performance of the models before actually defining the final model that will be used.

SUPPORT VECTOR CLASSIFIER

```
In [ ]: # Perform k-Fold Cross Validation on Train and Test (i.e. Validation) sets
for i, (train_index, validation_index) in enumerate(kf.split(X, y)):

    print(f"K FOLD #{i}")

    X_train, X_validation = X.iloc[train_index], X.iloc[validation_index]
    y_train, y_validation = y.iloc[train_index], y.iloc[validation_index]

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_validation = scaler.transform(X_validation)

    accuraciesTrain = []
```

```

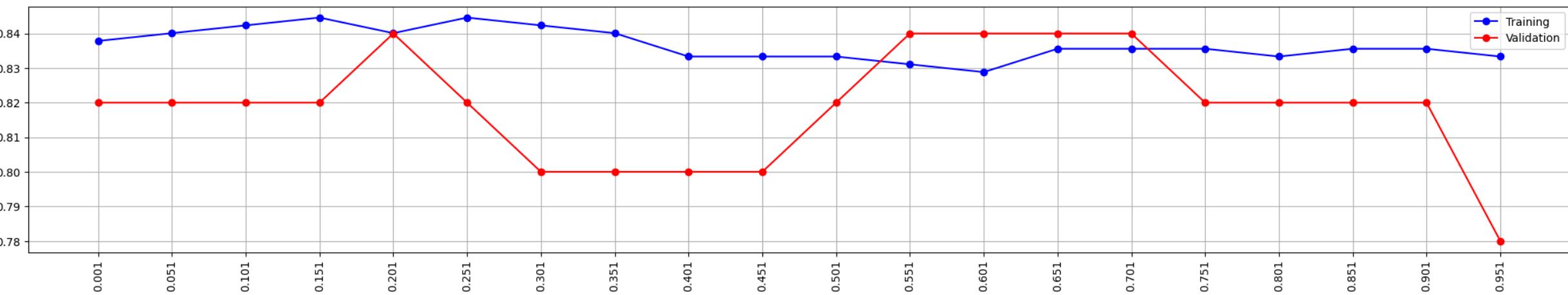
accuraciesTest = []
paramRange = np.arange(0.001, 1.0, 0.05)
for test in paramRange:
    Cls = SVC(
        random_state = 42,
        kernel = "linear", # linear, sigmoid, poly, rbf, precomputed
        probability = True,
        tol = test
    )
    Cls.fit(X_train, y_train)

    y_pred_train = Cls.predict(X_train)
    y_pred_validation = Cls.predict(X_validation)
    accuraciesTrain.append(accuracy_score(y_train, y_pred_train))
    accuraciesTest.append(accuracy_score(y_validation, y_pred_validation))

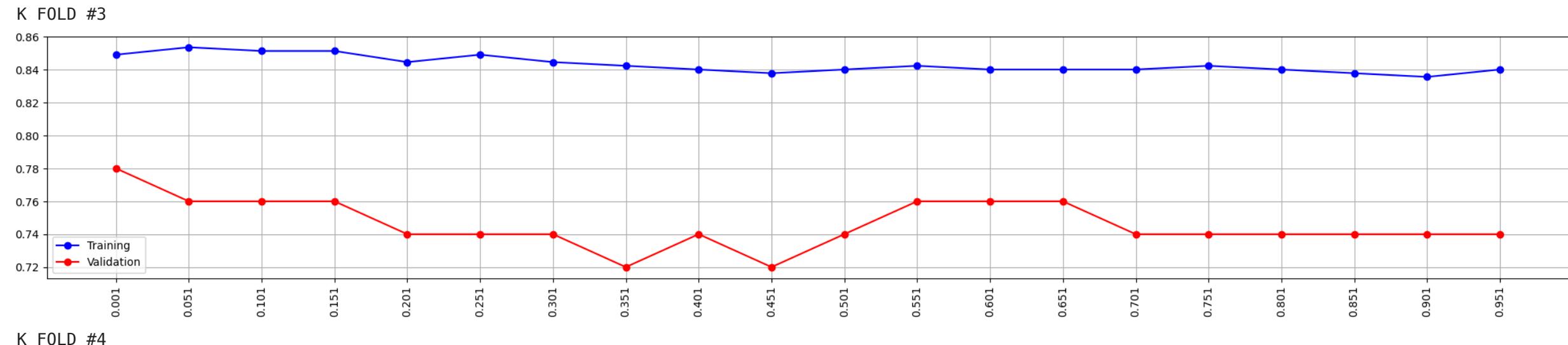
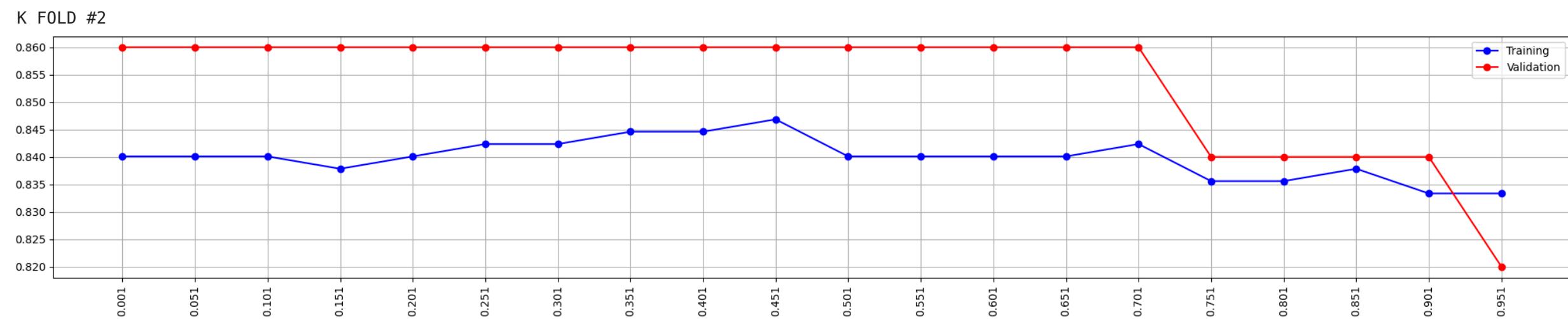
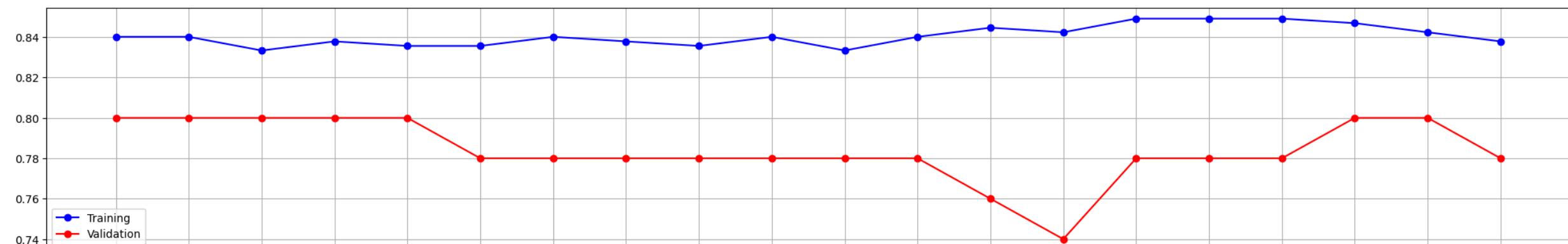
plt.figure(figsize=(25, 4))
plt.plot(paramRange, accuraciesTrain, "-o", label = "Training", color = "blue")
plt.plot(paramRange, accuraciesTest, "-o", label = "Validation", color = "red")
plt.xticks(paramRange, rotation=90)
plt.grid(True)
plt.legend()
plt.show()

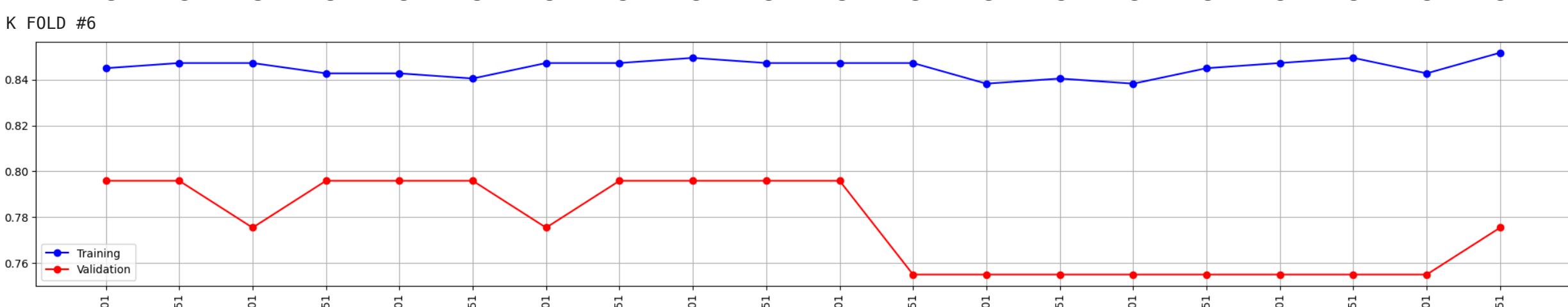
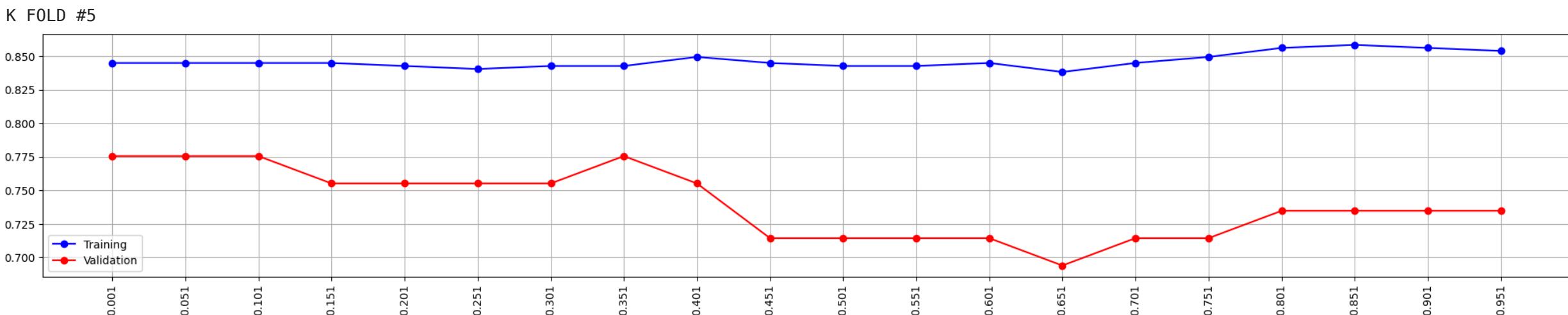
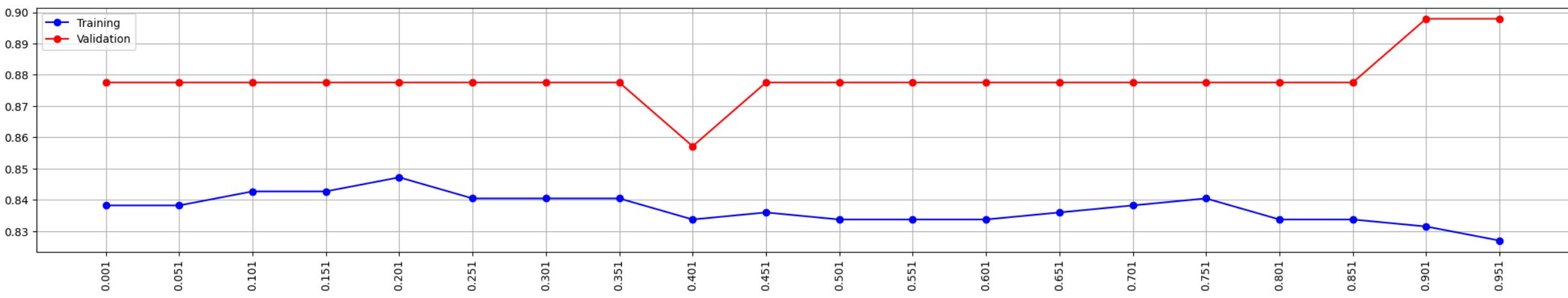
```

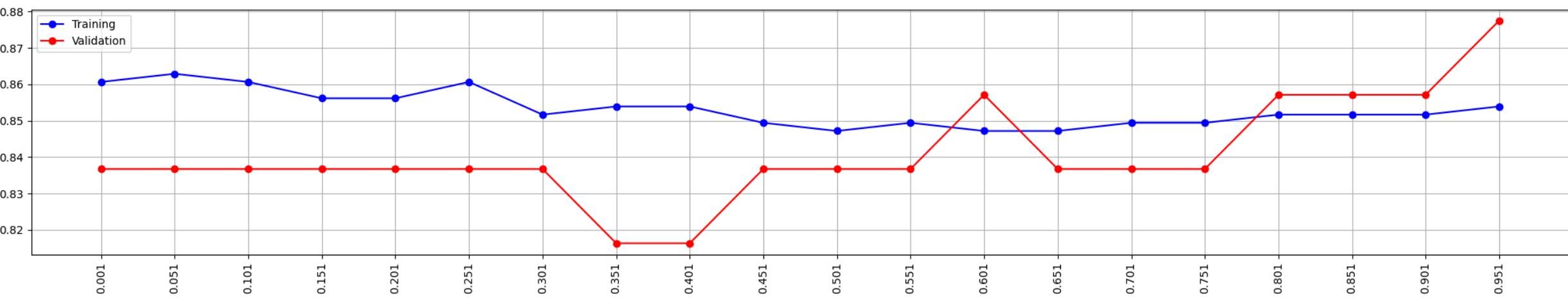
K FOLD #0



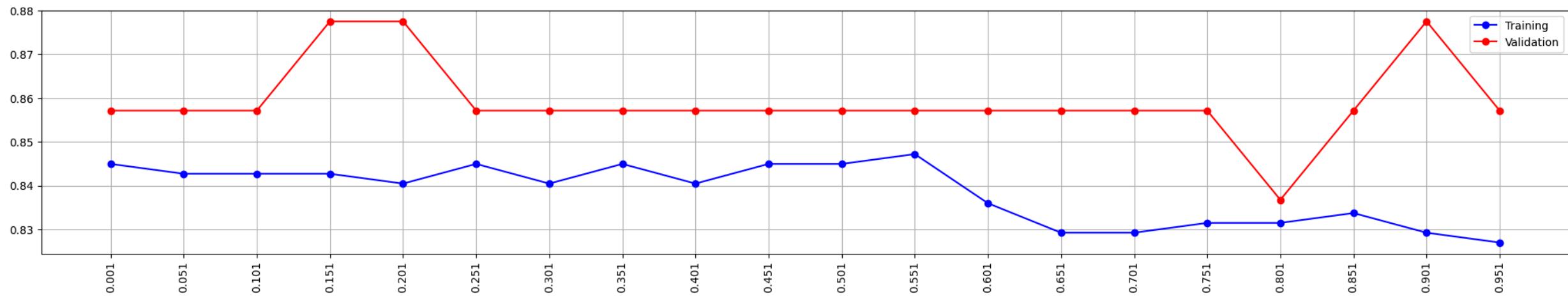
K FOLD #1



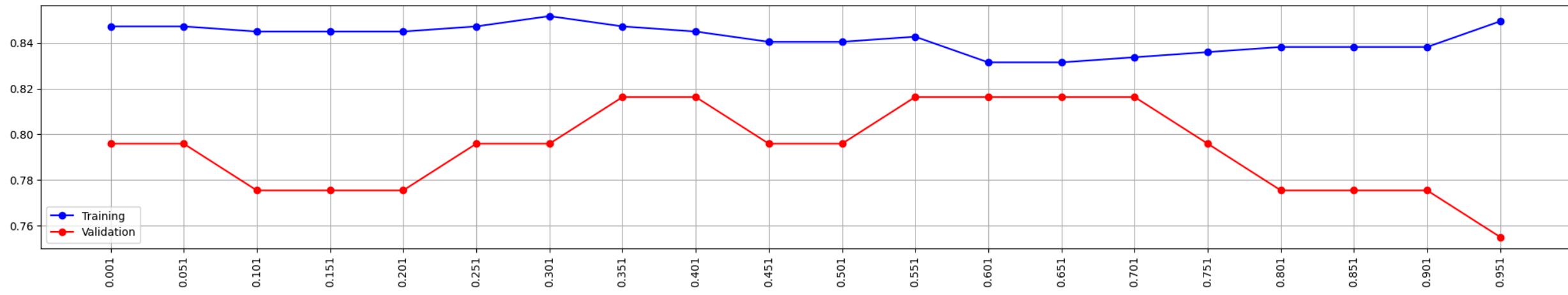




K FOLD #8



K FOLD #9



```
In [ ]: # Perform k-Fold Cross Validation on Train and Test (i.e. Validation) sets
for i, (train_index, validation_index) in enumerate(kf.split(X, y)):

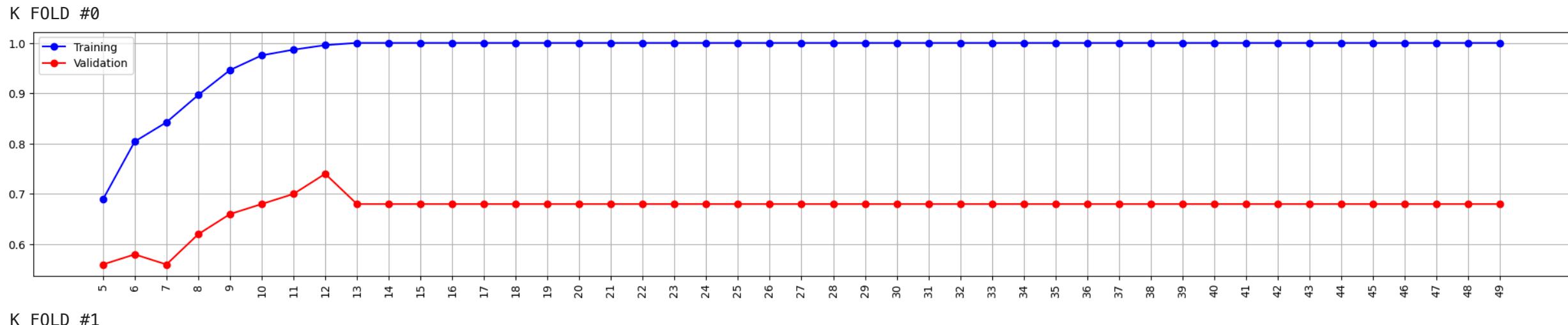
    print(f"K FOLD #{i}")

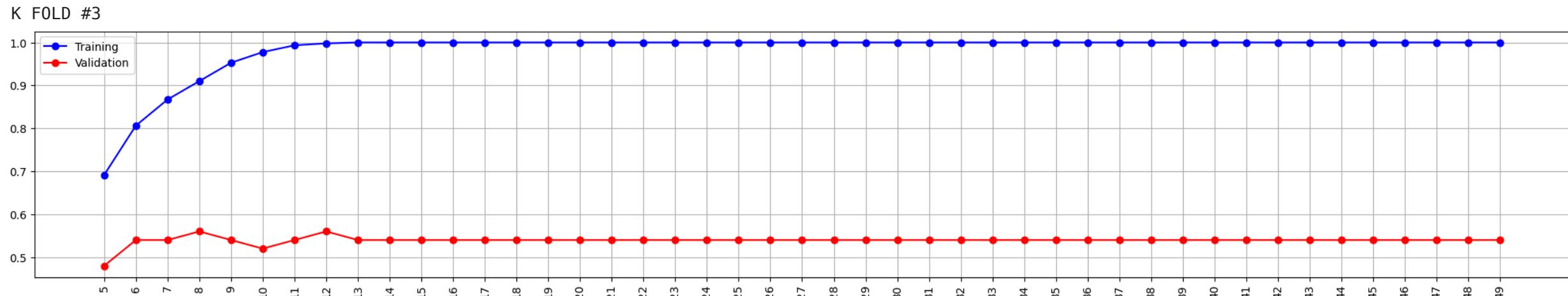
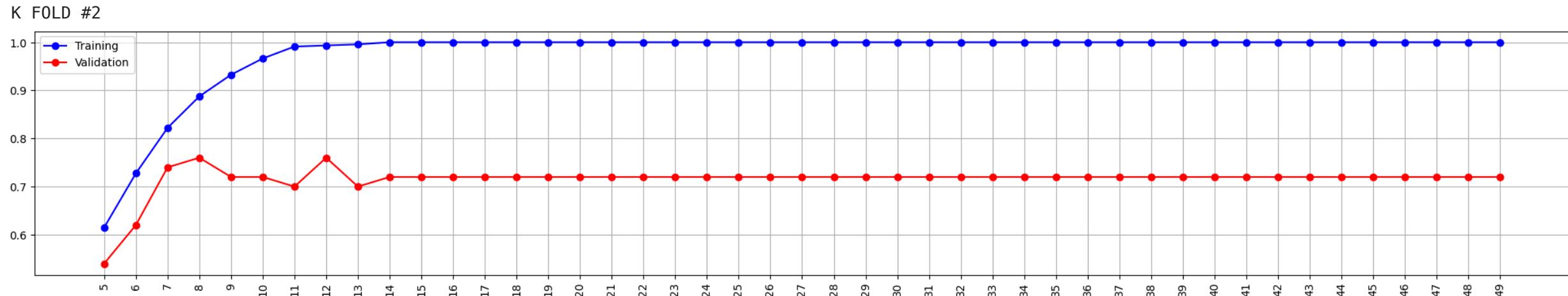
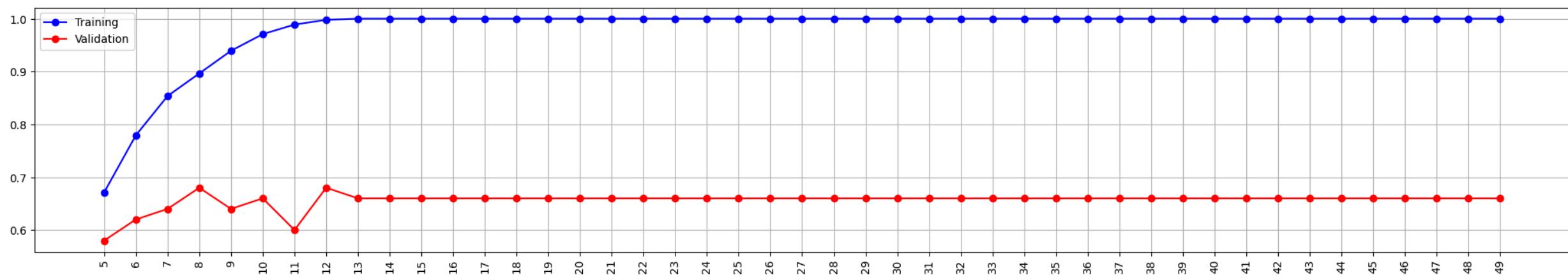
    X_train, X_validation = X.iloc[train_index], X.iloc[validation_index]
    y_train, y_validation = y.iloc[train_index], y.iloc[validation_index]

    accuraciesTrain = []
    accuraciesTest = []
    paramRange = np.arange(5, 50, 1)
    for test in paramRange:
        Cls = DecisionTreeClassifier(
            random_state = 42,
            criterion = "gini",
            max_depth = test
        )
        Cls.fit(X_train, y_train)

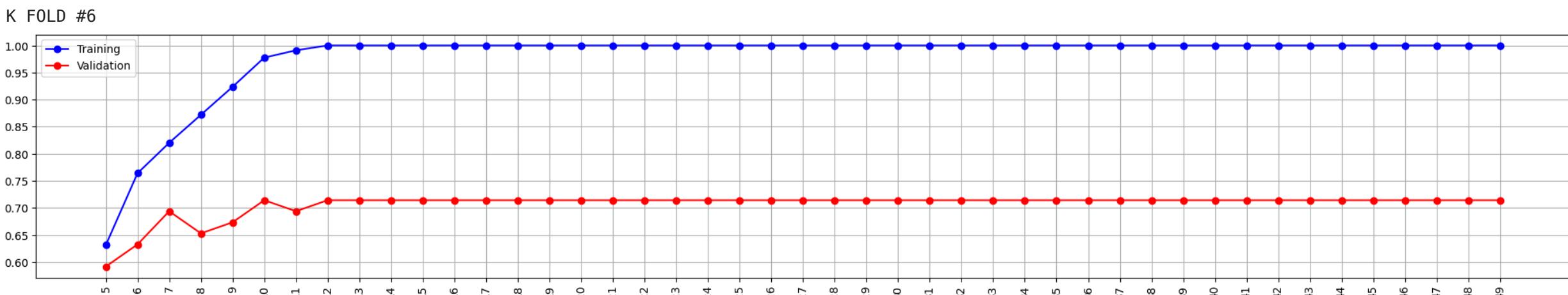
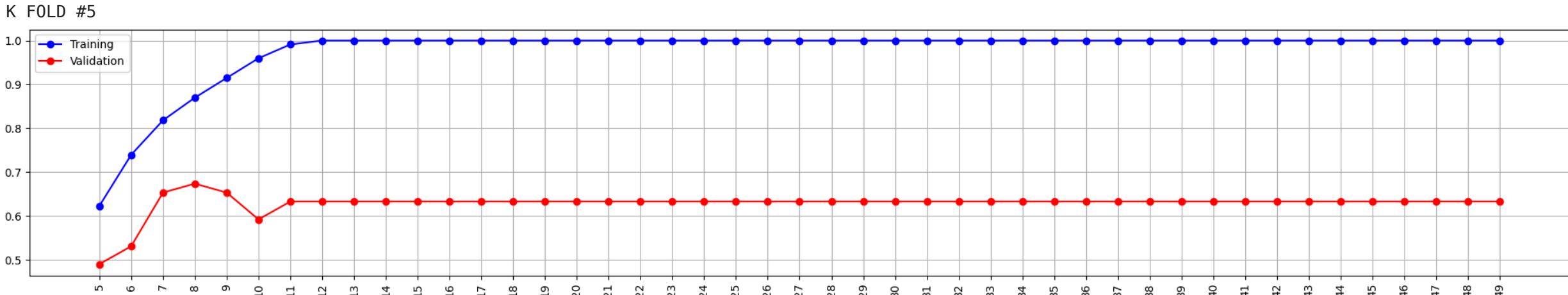
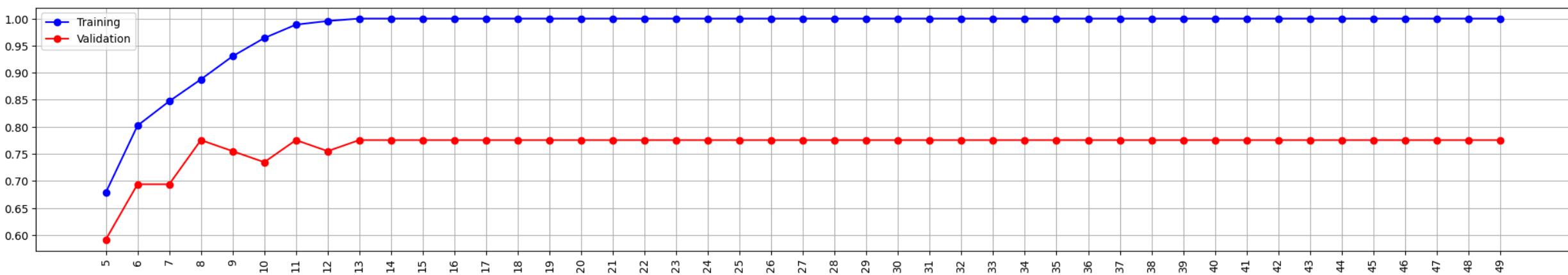
        y_pred_train = Cls.predict(X_train)
        y_pred_validation = Cls.predict(X_validation)
        accuraciesTrain.append(accuracy_score(y_train, y_pred_train))
        accuraciesTest.append(accuracy_score(y_validation, y_pred_validation))

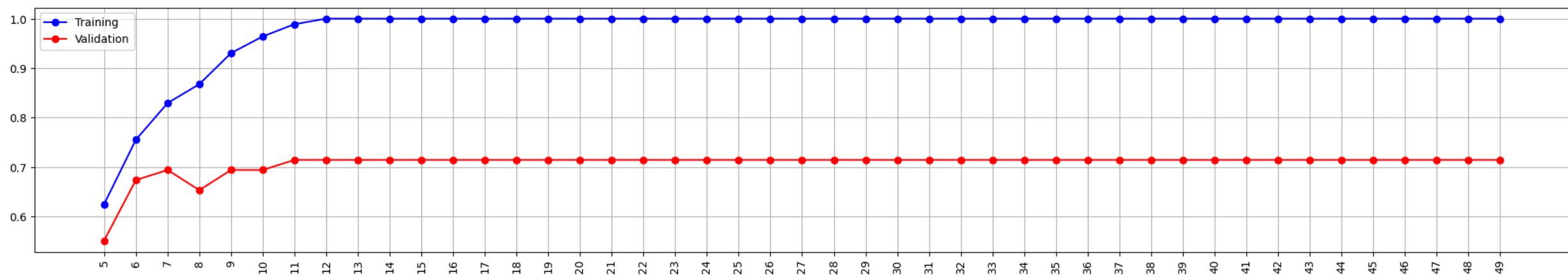
plt.figure(figsize=(25, 4))
plt.plot(paramRange, accuraciesTrain, "-o", label = "Training", color = "blue")
plt.plot(paramRange, accuraciesTest, "-o", label = "Validation", color = "red")
plt.xticks(paramRange, rotation=90)
plt.grid(True)
plt.legend()
plt.show()
```



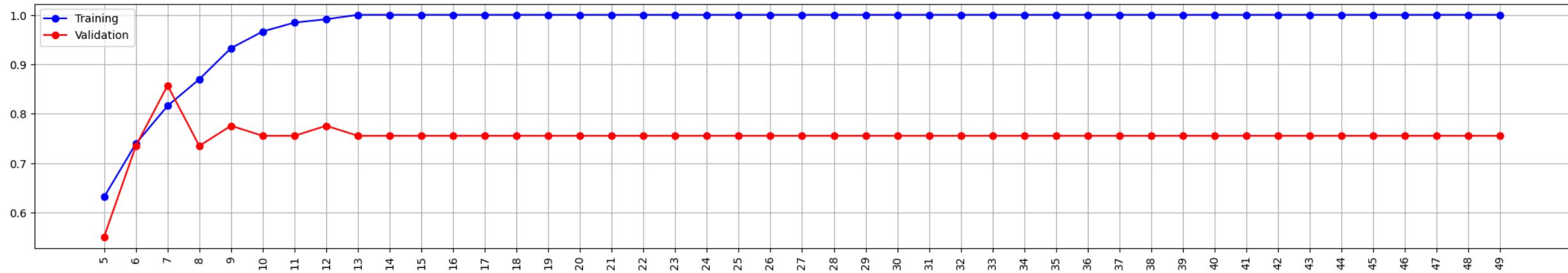


K FOLD #4

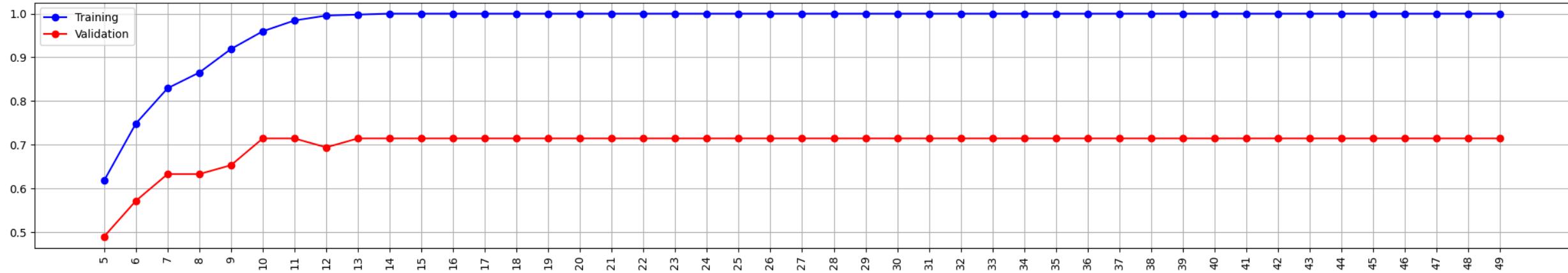




K FOLD #8



K FOLD #9



RANDOM FOREST CLASSIFIER

```
In [ ]: # Perform k-Fold Cross Validation on Train and Test (i.e. Validation) sets
for i, (train_index, validation_index) in enumerate(kf.split(X, y)):
```

```

print(f"K FOLD #{i}")

X_train, X_validation = X.iloc[train_index], X.iloc[validation_index]
y_train, y_validation = y.iloc[train_index], y.iloc[validation_index]

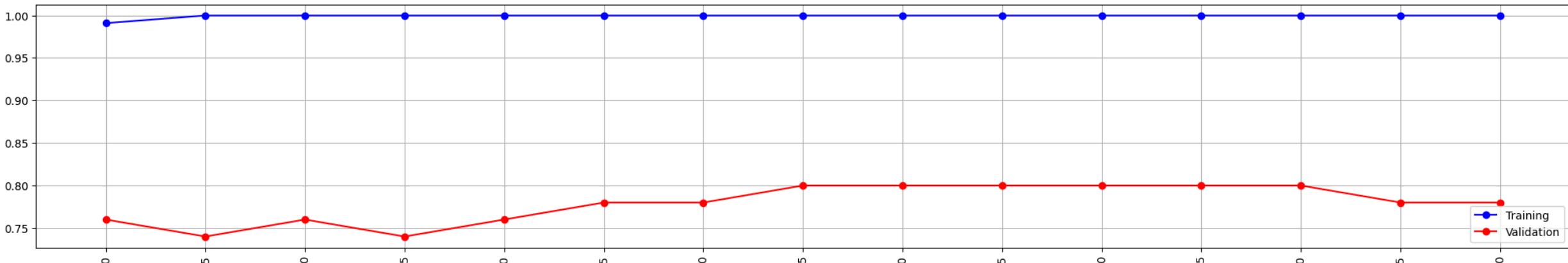
accuraciesTrain = []
accuraciesTest = []
paramRange = np.arange(10, 81, 5)
for test in paramRange:
    Cls = RandomForestClassifier(
        random_state = 42,
        n_estimators = test,
        criterion = "log_loss",
    )
    Cls.fit(X_train, y_train)

    y_pred_train = Cls.predict(X_train)
    y_pred_validation = Cls.predict(X_validation)
    accuraciesTrain.append(accuracy_score(y_train, y_pred_train))
    accuraciesTest.append(accuracy_score(y_validation, y_pred_validation))

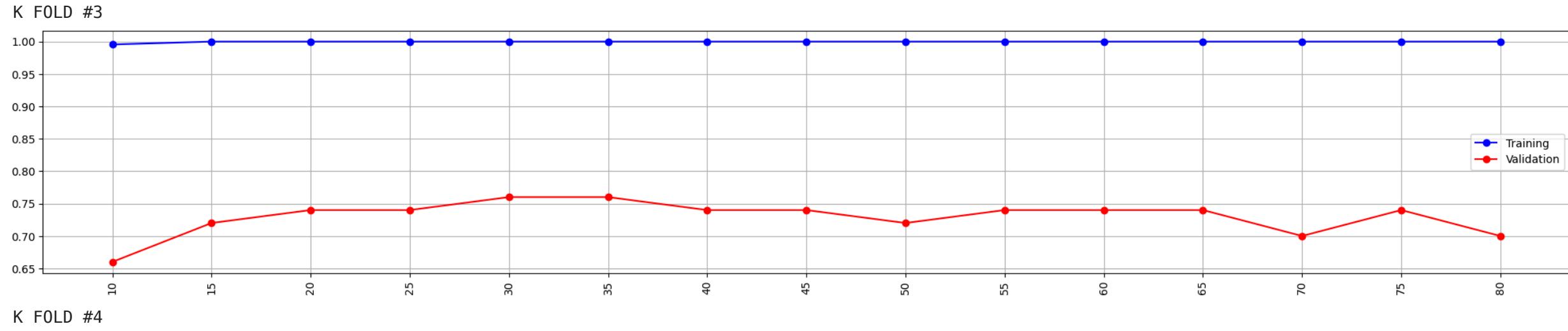
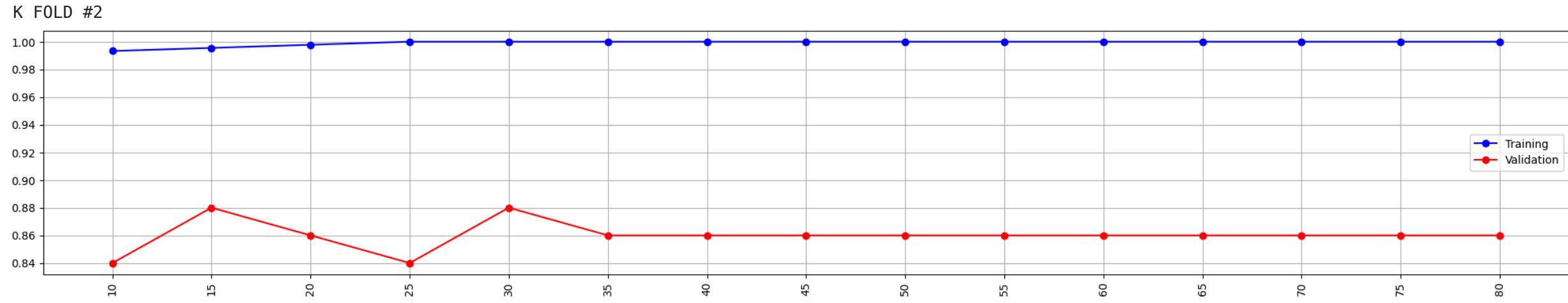
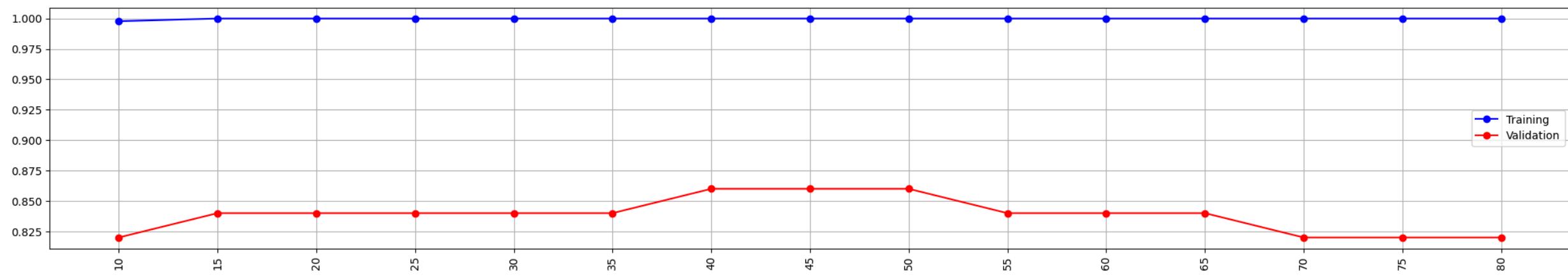
plt.figure(figsize=(25, 4))
plt.plot(paramRange, accuraciesTrain, "-o", label = "Training", color = "blue")
plt.plot(paramRange, accuraciesTest, "-o", label = "Validation", color = "red")
plt.xticks(paramRange, rotation=90)
plt.grid(True)
plt.legend()
plt.show()

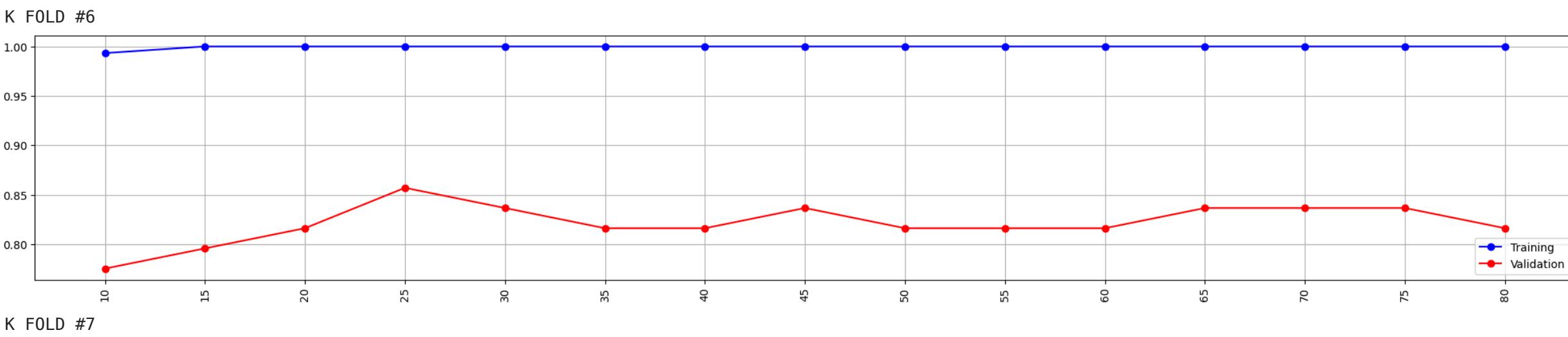
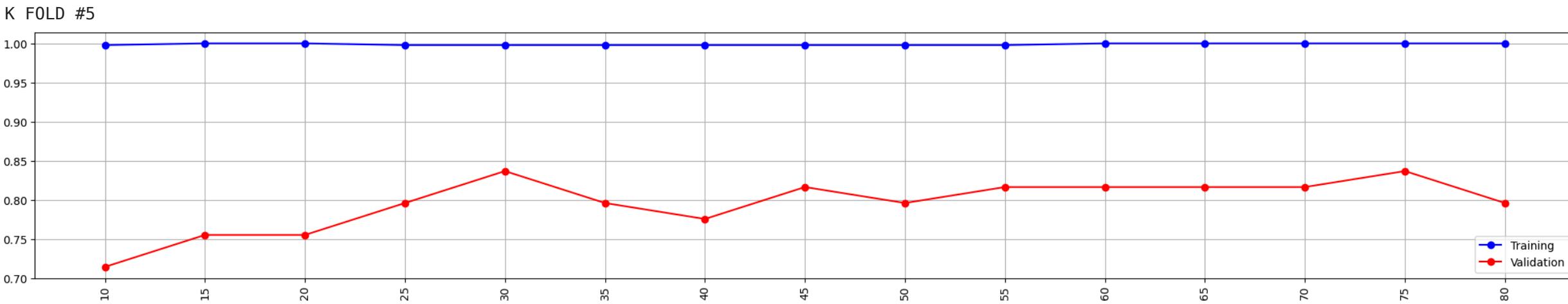
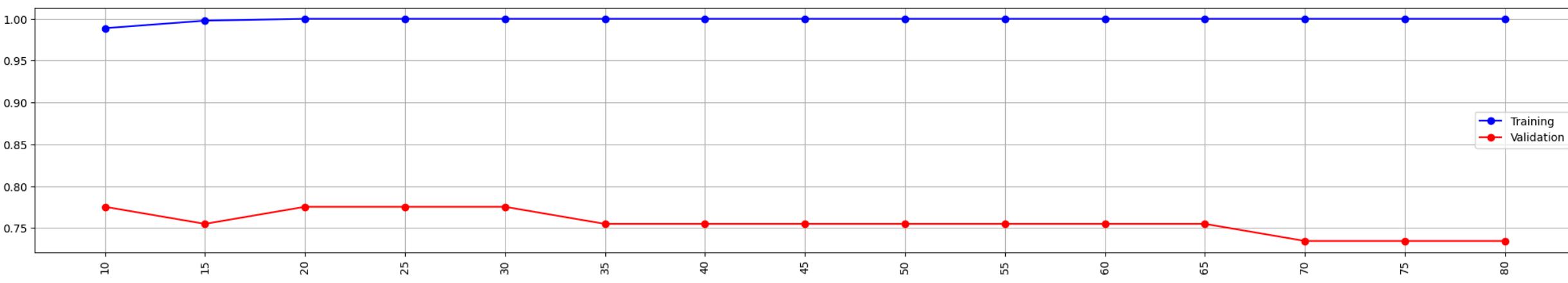
```

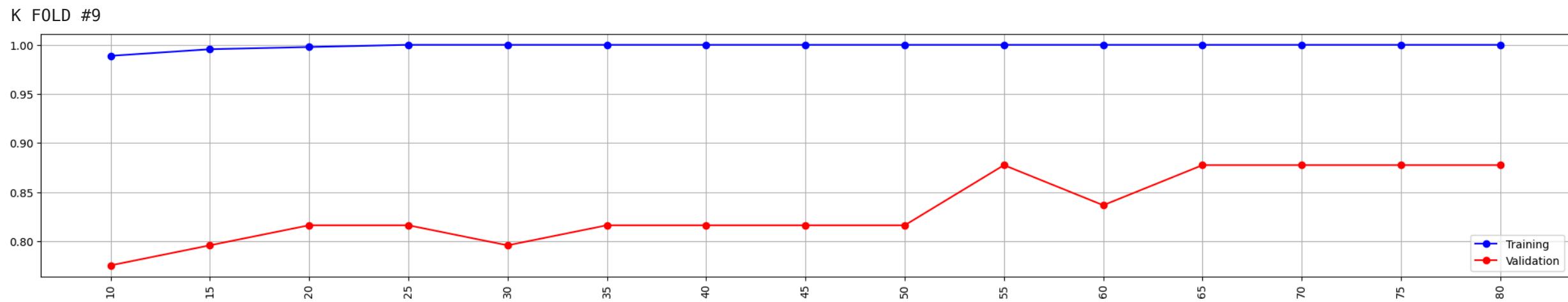
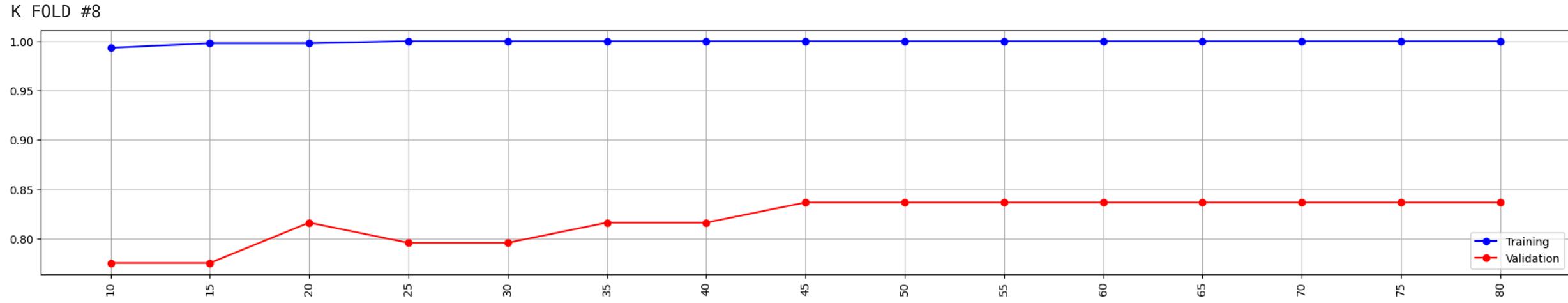
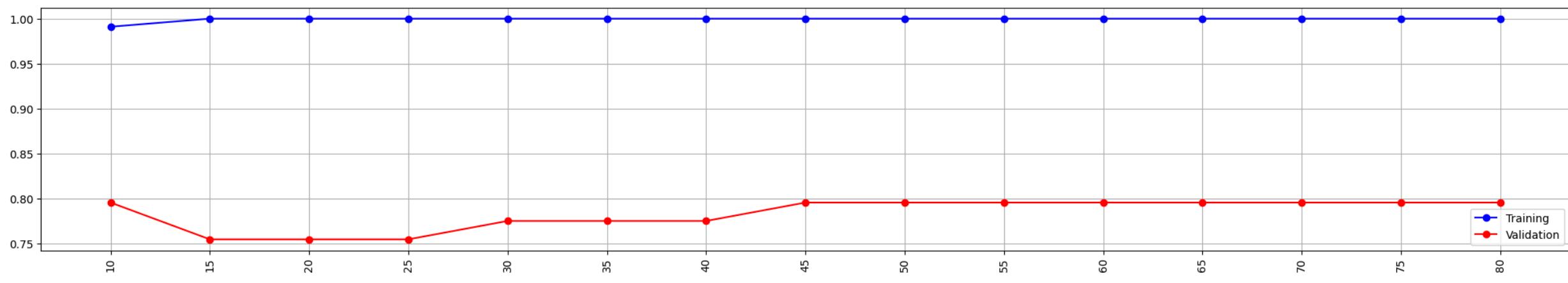
K FOLD #0



K FOLD #1







Model Creation with the Hyperparameters Selected

Now that we have found the (seemingly) best hyperparameters, in this section we train the models with them.

```
In [ ]: # Perform k-Fold Cross Validation on Train and Test (i.e. Validation) sets
for i, (train_index, validation_index) in enumerate(kf.split(X, y)):

    X_train, X_validation = X.iloc[train_index], X.iloc[validation_index]
    y_train, y_validation = y.iloc[train_index], y.iloc[validation_index]

    # Scaling -----
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_validation_scaled = scaler.transform(X_validation)

    # Model Validation -----
    svcCls = SVC(
        random_state = 42,
        kernel = "linear", # linear, sigmoid, poly, rbf, precomputed
        probability = True,
        tol = 0.201
    )
    dtCls = DecisionTreeClassifier(
        random_state = 42,
        criterion = "gini",
        max_depth = 12
    )
    rfCls = RandomForestClassifier(
        random_state = 42,
        #n_estimators = 65,
        #criterion = "gini",
        n_estimators = 20,
        criterion = "log_loss",
    )
    svcCls.fit(X_train_scaled, y_train)
    dtCls.fit(X_train, y_train)
    rfCls.fit(X_train, y_train)

    y_predSVC = svcCls.predict(X_validation_scaled)
    y_predDT = dtCls.predict(X_validation)
    y_predRF = rfCls.predict(X_validation)

    modelsResults["SVC"]["Accuracy"].append(accuracy_score(y_validation, y_predSVC))
    modelsResults["DT"]["Accuracy"].append(accuracy_score(y_validation, y_predDT))
    modelsResults["RF"]["Accuracy"].append(accuracy_score(y_validation, y_predRF))

    modelsResults["SVC"]["Recall"].append(recall_score(y_validation, y_predSVC, average="macro"))
    modelsResults["DT"]["Recall"].append(recall_score(y_validation, y_predDT, average="macro"))
    modelsResults["RF"]["Recall"].append(recall_score(y_validation, y_predRF, average="macro"))

    modelsResults["SVC"]["Precision"].append(precision_score(y_validation, y_predSVC, average="macro"))
```

```

modelsResults["DT"]["Precision"].append(precision_score(y_validation, y_predDT, average="macro"))
modelsResults["RF"]["Precision"].append(precision_score(y_validation, y_predRF, average="macro"))

modelsResults["SVC"]["F1Score"].append(f1_score(y_validation, y_predSVC, average="macro"))
modelsResults["DT"]["F1Score"].append(f1_score(y_validation, y_predDT, average="macro"))
modelsResults["RF"]["F1Score"].append(f1_score(y_validation, y_predRF, average="macro"))

y_probSVC = svcCls.predict_proba(X_validation_scaled)
y_probDT = dtCls.predict_proba(X_validation)
y_probRF = rfCls.predict_proba(X_validation)

modelsResults["SVC"]["AUC"].append(roc_auc_score(y_validation, y_probSVC, multi_class="ovr"))
modelsResults["DT"]["AUC"].append(roc_auc_score(y_validation, y_probDT, multi_class="ovr"))
modelsResults["RF"]["AUC"].append(roc_auc_score(y_validation, y_probRF, multi_class="ovr"))

```

Display the average and individual results of each Fold

```
In [ ]: for model in modelsResults.keys():

    print("\n# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #")
    print(model)

    print(" ===== ACCURACY")
    print("Per fold:", *(accuracyList := modelsResults[model]["Accuracy"]), sep="   ")
    print(f"Average: {sum(accuracyList)/len(accuracyList)}")

    print(" ===== RECALL")
    print("Per fold:", *(sensitivityList := modelsResults[model]["Recall"]), sep="   ")
    print(f"Average: {sum(sensitivityList)/len(sensitivityList)}")

    print(" ===== PRECISION")
    print("Per fold:", *(precisionList := modelsResults[model]["Precision"]), sep="   ")
    print(f"Average: {sum(precisionList)/len(precisionList)}")

    print(" ===== F1 SCORE")
    print("Per fold:", *(f1ScoreList := modelsResults[model]["F1Score"]), sep="   ")
    print(f"Average: {sum(f1ScoreList)/len(f1ScoreList)}")

    print(" ===== AUC - ROC")
    print("Per fold:", *(aucList := modelsResults[model]["AUC"]), sep="   ")
    print(f"Average: {sum(aucList)/len(aucList)}")
```

#

SVC

===== ACCURACY

Per fold: 0.82 0.8 0.86 0.76 0.8775510204081632 0.7755102040816326 0.7755102040816326 0.8367346938775511 0.8571428571428571 0.7755102040816326 0.84 0.8 0.86 0.74 0.8775510204081632 0.7551020408163265 0.7959183673469388 0.8367346938775511 0.8775510204081632 0.7755102040816326 0.84 0.8 0.86 0.74 0.8775510204081632 0.7551020408163265 0.7959183673469388 0.8367346938775511 0.8775510204081632 0.7755102040816326
Average: 0.8151564625850339

===== RECALL

Per fold: 0.805 0.7999999999999999 0.86 0.76 0.875 0.775 0.78 0.8350000000000002 0.86 0.77 0.825 0.7999999999999999 0.86 0.74 0.875 0.75 0.8 0.8350000000000002 0.8800000000000001 0.77

Average: 0.8130000000000001

===== PRECISION

Per fold: 0.8061904761904763 0.8383333333333333 0.8838095238095238 0.7808333333333334 0.8930952380952382 0.7841666666666666 0.8166666666666667 0.8633333333333333 0.8835714285714286 0.8319444444444445 0.8561904761904764 0.8383333333333333 0.8838095238095238 0.7605952380952381 0.8930952380952382 0.7508333333333334 0.8316666666666667 0.8633333333333333 0.8954761904761905 0.8275 0.8561904761904764 0.8383333333333333 0.8838095238095238 0.7605952380952381 0.8930952380952382
Average: 0.8394537037037039

===== F1 SCORE

Per fold: 0.7895959595959596 0.7959090909090908 0.8640404040404042 0.7486113886113885 0.872929292929293 0.7665190365190365 0.78002331002331 0.8407808857808858 0.8555050505050505 0.7745421245421247 0.8073737373737373 0.7959090909090908 0.8640404040404042 0.7243689643689644 0.872929293 0.7403285603285604 0.7978010878010877 0.8407808857808858 0.87696969696969697 0.7733821733821735 0.8073737373737373 0.7959090909090908 0.8640404040404042 0.7243689643689644 0.872929292929293 0.7403285603285604 0.7978010878010877 0.8407808857808858 0.87696969696969697 0.7733821735
Average: 0.8092074777074778

===== AUC - ROC

Per fold: 0.9395990338164252 0.9764444444444444 0.9857777777777776 0.9515555555555556 0.988030303030303 0.9591414141414141 0.9668181818181818 0.9644444444444445 0.9866161616161616 0.9654545454545456 0.9400434782608696 0.9768888888888888 0.9866666666666667 0.9528888888888888 0.987020202020 0.9613131313131313 0.9668181818181818 0.9662626262626264 0.9860606060606061 0.9654545454545456 0.9400434782608696 0.9768888888888888 0.9866666666666667 0.9528888888888888 0.9870202020202020 0.9613131313131313 0.9668181818181818 0.9662626262626264 0.9860606060606061 0.9654545454545456
Average: 0.9687572097789487

#

DT

===== ACCURACY

Per fold: 0.68 0.66 0.7 0.54 0.7755102040816326 0.6326530612244898 0.7142857142857143 0.7142857142857143 0.7551020408163265 0.7142857142857143 0.857143 0.74 0.68 0.76 0.56 0.7551020408163265 0.6326530612244898 0.7142857142857143 0.7142857142857143 0.755102040816326 0.6938775510204082 0.74 0.68 0.76 0.56 0.7551020408163265 0.6326530612244898 0.7142857142857143 0.7142857142857143 0.7755102040816326 0.6938775510204082
Average: 0.6979183673469388

===== RECALL

Per fold: 0.6733333333333332 0.6599999999999999 0.7 0.54 0.775 0.635 0.72 0.72 0.7549999999999999 0.72 0.7333333333333333 0.6799999999999999 0.76 0.5599999999999999 0.76 0.5599999999999999 0.755 0.635 0.72 0.72 0.7749999999999999 0.7 0.7333333333333333 0.6799999999999999 0.76 0.5599999999999999
Average: 0.6991666666666665

===== PRECISION
Per fold: 0.7026190476190475 0.6957142857142857 0.6814285714285715 0.55916666666666667 0.8147619047619049 0.6171428571428572 0.7519047619047619
0.75444444444444445 0.7943650793650793 0.7772222222222223 0.7564285714285715 0.6950000000000001 0.7464285714285714 0.5713095238095237 0.79499999
999999999 0.6171428571428572 0.7519047619047619 0.7544444444444445 0.8263095238095237 0.7357142857142855 0.7564285714285715 0.6950000000000001
0.7464285714285714 0.5713095238095237 0.7949999999999999 0.6171428571428572 0.7519047619047619 0.7544444444444445 0.8263095238095237 0.7357142
857142855
Average: 0.7216044973544974
===== F1 SCORE
Per fold: 0.6761111111111111 0.6653030303030303 0.6799855699855699 0.5326029526029527 0.7695904095904096 0.6148212898212898 0.7235858585858587
0.7194155844155844 0.7582539682539682 0.701933621933622 0.7343939393939394 0.67883838383838 0.7422077922077922 0.55005439005439 0.7506510156
510157 0.6148212898212898 0.7235858585858587 0.7194155844155844 0.7769519369519369 0.67828282828282 0.7343939393939394 0.67883838383838
0.7422077922077922 0.55005439005439 0.7506510156510157 0.6148212898212898 0.7235858585858587 0.7194155844155844 0.7769519369519369 0.67828282
282822
Average: 0.6926669811669812
===== AUC - ROC
Per fold: 0.8189361001317523 0.8111111111111111 0.8435555555555556 0.7444444444444445 0.8750252525252524 0.7970959595959597 0.8440909090909091
0.8440909090909091 0.8638636363636365 0.8551010101010101 0.8622694334650858 0.8222222222222222 0.8762222222222222 0.7542222222222222 0.8639141
414141414 0.7970959595959597 0.8440909090909091 0.8440909090909091 0.8756313131313131 0.8437121212121212 0.8622694334650858 0.8222222222222222
0.8762222222222222 0.7542222222222222 0.8639141414141414 0.7970959595959597 0.8440909090909091 0.8440909090909091 0.8756313131313131 0.8437121
212121212
Average: 0.8354752598448251

RF
===== ACCURACY
Per fold: 0.8 0.86 0.86 0.74 0.7551020408163265 0.8163265306122449 0.8367346938775511 0.7959183673469388 0.8367346938775511 0.8163265306
122449 0.78 0.9 0.86 0.78 0.8163265306122449 0.7551020408163265 0.8571428571428571 0.7959183673469388 0.8163265306122449 0.7959183673469
388 0.76 0.84 0.86 0.74 0.7755102040816326 0.7551020408163265 0.8163265306122449 0.7551020408163265 0.8163265306122449 0.816326530612244
9
Average: 0.806952380952381
===== RECALL
Per fold: 0.7933333333333333 0.86 0.8600000000000001 0.74 0.755 0.8200000000000001 0.8400000000000001 0.8 0.835 0.82 0.773333333333333
3 0.9 0.8600000000000001 0.78 0.8150000000000001 0.75 0.8600000000000001 0.8 0.8150000000000001 0.8 0.753333333333334 0.840000000000000
001 0.86 0.74 0.775 0.755 0.8200000000000001 0.76 0.8150000000000001 0.8149999999999998
Average: 0.8069999999999999
===== PRECISION
Per fold: 0.8155555555555555 0.8891666666666668 0.883333333333332 0.8 0.7966666666666667 0.8474999999999999 0.8695238095238096 0.815833333
333333 0.8497619047619047 0.8178571428571428 0.7766666666666666 0.9147619047619047 0.883333333333332 0.8175000000000001 0.8547619047619047
0.7789285714285714 0.8811904761904762 0.805833333333334 0.8408333333333333 0.8139285714285714 0.7625 0.86 0.8847619047619049 0.803333333333
333 0.768333333333333 0.7736507936507937 0.8480952380952381 0.7616666666666666 0.840833333333333 0.825833333333333
Average: 0.8293981481482
===== F1 SCORE
Per fold: 0.7926118326118325 0.8604079254079254 0.8646464646464647 0.7542207792207792 0.7406060606060605 0.7985392385392386 0.8410606060606061
0.7924281274281275 0.8314646464646465 0.8040404040404041 0.7681529581529581 0.89813131313131 0.8646464646464647 0.7897902097902099 0.7951010
101010101 0.7418437118437119 0.8547979797979798 0.792983682983683 0.8071972471972473 0.7894988344988344 0.7458624708624708 0.84005050505052
0.85878787878788 0.751010101010101 0.7606926406926406 0.7411832611832612 0.8211616161616162 0.7561616161616163 0.813943278943279 0.808943278
943279

```
Average: 0.8026655381655381
=====
AUC - ROC
Per fold: 0.9691316425120771 0.9908888888888889 0.9848888888888888 0.9475555555555554 0.9796212121212122 0.9745707070707071 0.9736363636363636
0.9586363636363637 0.9895454545454546 0.9897727272727274 0.9735934343434343 0.9913333333333334 0.9782222222222222 0.9635555555555557 0.9823989
898989899 0.967449494949495 0.988409090909091 0.9636363636363638 0.9865151515151516 0.9920454545454547 0.9526606280193237 0.986 0.9762222222
22222 0.9426666666666665 0.9769949494949495 0.9752020202020202 0.9663636363636364 0.9606818181818182 0.978459595959596 0.989090909090909
Average: 0.974991644707949
```

Model Testing

In this section, we use the test images from the folder *digitsTest*, which contain the handwritten Student ID of the members of our team. We measure the accuracy of the model and display on the image the digit that our model detected for each one of the digits in the ID.

```
In [ ]: colnames = ["euler", "Hu1", "Hu2", "Hu3", "Hu4", "Hu5", "Hu6", "Hu7"]
X_test = pd.DataFrame(columns = colnames)
boundariesPerDigit = []

y_test = {
    "Den" : pd.Series([0, 1, 5, 6, 2, 0, 7, 7]),
    "Luis" : pd.Series([0, 1, 3, 7, 8, 2, 4, 8]),
    "Rivers" : pd.Series([0, 1, 3, 7, 8, 9, 6, 5])
}

TEST = "Rivers"          # ----- Change this line if you want to test with someone else's ID
```

Segment the test image and get the features per digit

```
In [ ]: frame = cv.imread(os.path.join("digitsTest", f"{TEST}.jpeg"))
rawImg = frame.copy()

kernel = np.ones((5,5),np.uint8)
bkgnd = cv.dilate(frame,kernel,iterations = 1)
bkgnd = cv.dilate(bkgnd,kernel,iterations = 1)

frame = (bkgnd - frame)
frgnd = frame.copy()

kernel = np.ones((3,3),np.uint8)
frame = cv.dilate(frame,kernel,iterations = 1)

green = frame[:, :, 2]
ret, img = cv.threshold(green, 32, 100, 0)
```

```
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(cv.cvtColor(bkgnd, cv.COLOR_BGR2RGB))
ax[0].set_title("Background")

ax[1].imshow(cv.cvtColor(frgnd, cv.COLOR_BGR2RGB))
ax[1].set_title("Foreground (Just Numbers)")

ax[2].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
ax[2].set_title("Refined Numbers")
plt.show()

# #####
labeled, regions, EdgeImage = SegmentationRegions(img, frame)

labeled = labeled % 21

img = cv.normalize(labeled, None, 0, 255, cv.NORM_MINMAX, cv.CV_8U)
im_color = cv.applyColorMap(img, cv.COLORMAP_HSV)
im_color = cv.medianBlur(im_color,3)
RGB_im = cv.cvtColor(im_color, cv.COLOR_BGR2RGB)

# #####
fig, ax = plt.subplots(1,2, figsize=(10, 5))
for n in range(len(regions)-2):
    idx = n+2
    if regions[idx].Perimetro > PERIMETERS[digit][0]:
        if regions[idx].Perimetro < PERIMETERS[digit][1]:

            pointlist = orderPointSet(regions[idx].edgePoints, EdgeImage)
            smoothpoints = np.array(smoothPointSet(pointlist[0],3))
            euler, hu1, hu2, hu3, hu4, hu5, hu6, hu7, xMin, yMin, xMax, yMax = getInvariantFeatures(pointlist, displayIndividual=False)
            digitFeatures = pd.DataFrame([[euler, hu1, hu2, hu3, hu4, hu5, hu6, hu7]], columns = colnames)
            X_test = pd.concat([X_test, digitFeatures], ignore_index = True)

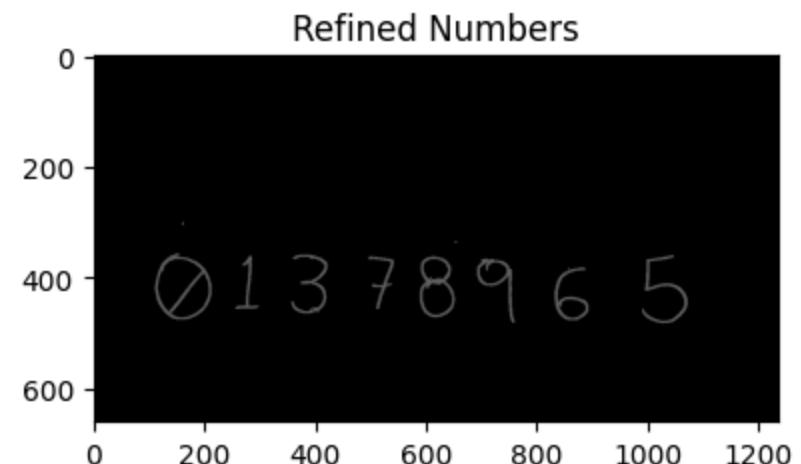
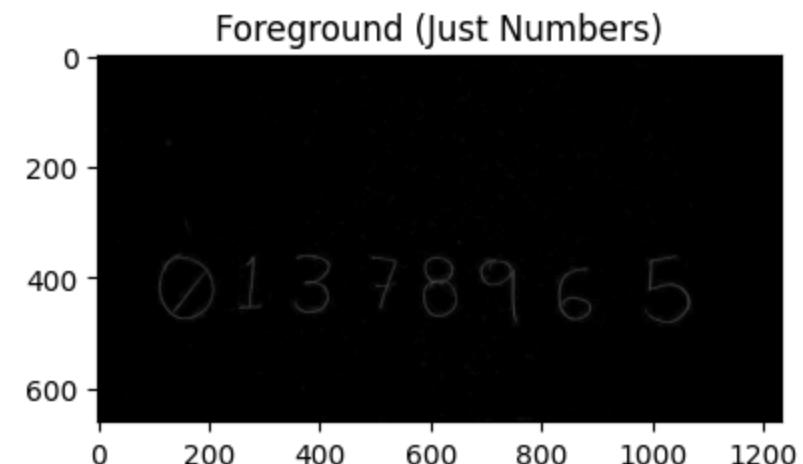
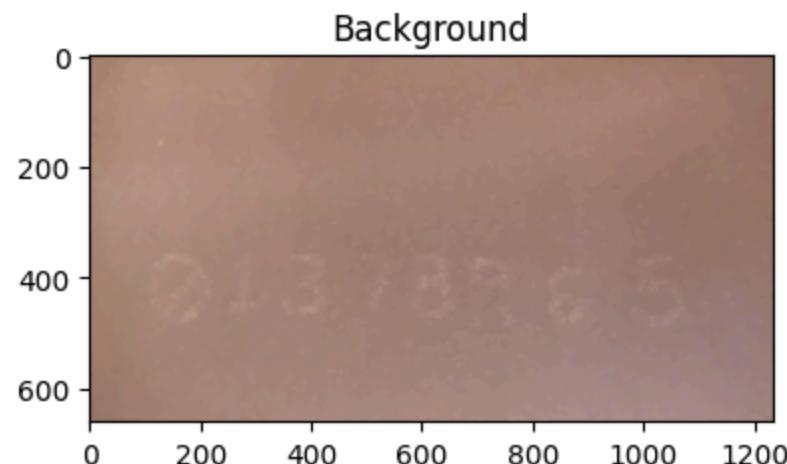
            boundariesPerDigit.append(((xMin, yMin), (xMax, yMax)))

            ax[0].plot(smoothpoints[:,0],smoothpoints[:,1])

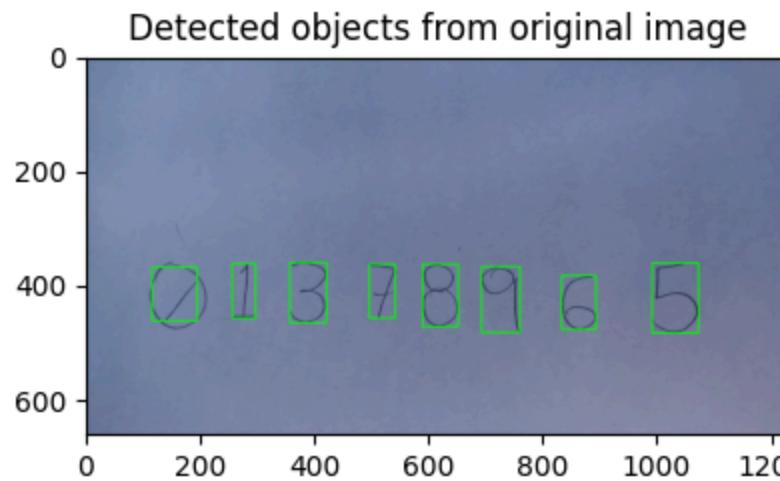
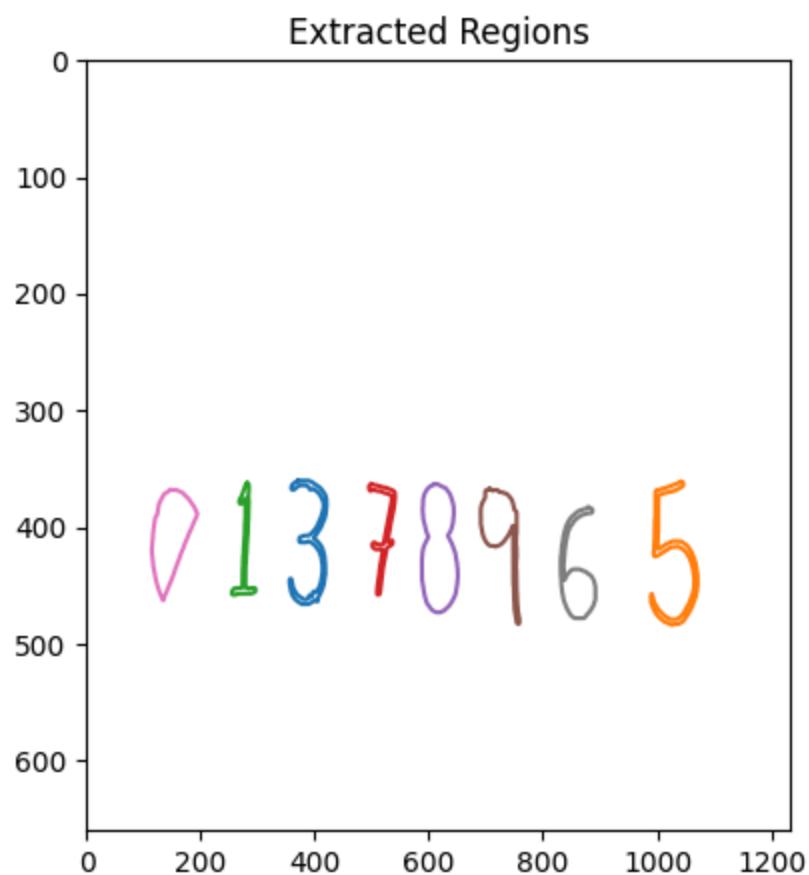
    ax[0].set_ylim(frame.shape[0], 0)
    ax[0].set_xlim(0, frame.shape[1])
    ax[0].set_title("Extracted Regions")

    boundedImg = rawImg.copy()
    for startPoint, endPoint in boundariesPerDigit:
        cv.rectangle(boundedImg, startPoint, endPoint, (0,255,0), 2)
    ax[1].imshow(boundedImg)
    ax[1].set_title("Detected objects from original image")
```

```
plt.show()
```



```
768  
745  
16  
0  
22
```



Sort the detected elements based on their appearance in the page (from left to right)

```
In [ ]: # Pair the DataFrame rows (only values) and boundary tuples
pairedData = zip(X_test.itertuples(index=False, name=None), boundariesPerDigit)

# Sort the pairs by the xMin value of each tuple
sortedPairs = sorted(pairedData, key=lambda pair: pair[1][0][0])

X_test = pd.DataFrame([row for row, _ in sortedPairs], columns=X_test.columns)
boundariesPerDigit = [bound for _, bound in sortedPairs]
```

```
In [ ]: X_test
```

	euler	Hu1	Hu2	Hu3	Hu4	Hu5	Hu6	Hu7
0	-348	6.773842	6.970468	26.018195	1.079363	-4.896685	-2.849619	-2.956335
1	-326	3.885601	12.070505	7.139027	2.090368	7.960518	7.085440	1.356146
2	-636	3.715669	4.874793	4.204810	1.057521	-2.179013	-2.277989	0.474182
3	-400	3.086699	5.135050	6.828121	0.729274	-0.761329	-0.337087	-1.438305
4	-388	6.864169	8.883627	1.172738	0.202881	-0.086722	0.604686	0.047671
5	-412	4.767428	9.909504	27.881572	2.209706	-16.756491	-6.530058	4.477708
6	-432	4.062084	2.969372	11.598192	3.069221	-9.049546	-4.606719	15.919722
7	-718	3.924083	3.646848	3.141065	0.020356	-0.002614	0.028781	-0.004434

In []: boundariesPerDigit

```
Out[ ]: [((np.int64(115), np.int64(368)), (np.int64(194), np.int64(462))),
 ((np.int64(255), np.int64(362)), (np.int64(296), np.int64(457))),
 ((np.int64(356), np.int64(360)), (np.int64(420), np.int64(465))),
 ((np.int64(495), np.int64(363)), (np.int64(540), np.int64(457))),
 ((np.int64(588), np.int64(363)), (np.int64(650), np.int64(472))),
 ((np.int64(691), np.int64(367)), (np.int64(758), np.int64(482))),
 ((np.int64(831), np.int64(383)), (np.int64(891), np.int64(477))),
 ((np.int64(989), np.int64(361)), (np.int64(1070), np.int64(483)))]
```

Perform the actual prediction on the test set

SUPPORT VECTOR CLASSIFIER

```
In [ ]: X_test = scaler.transform(X_test)

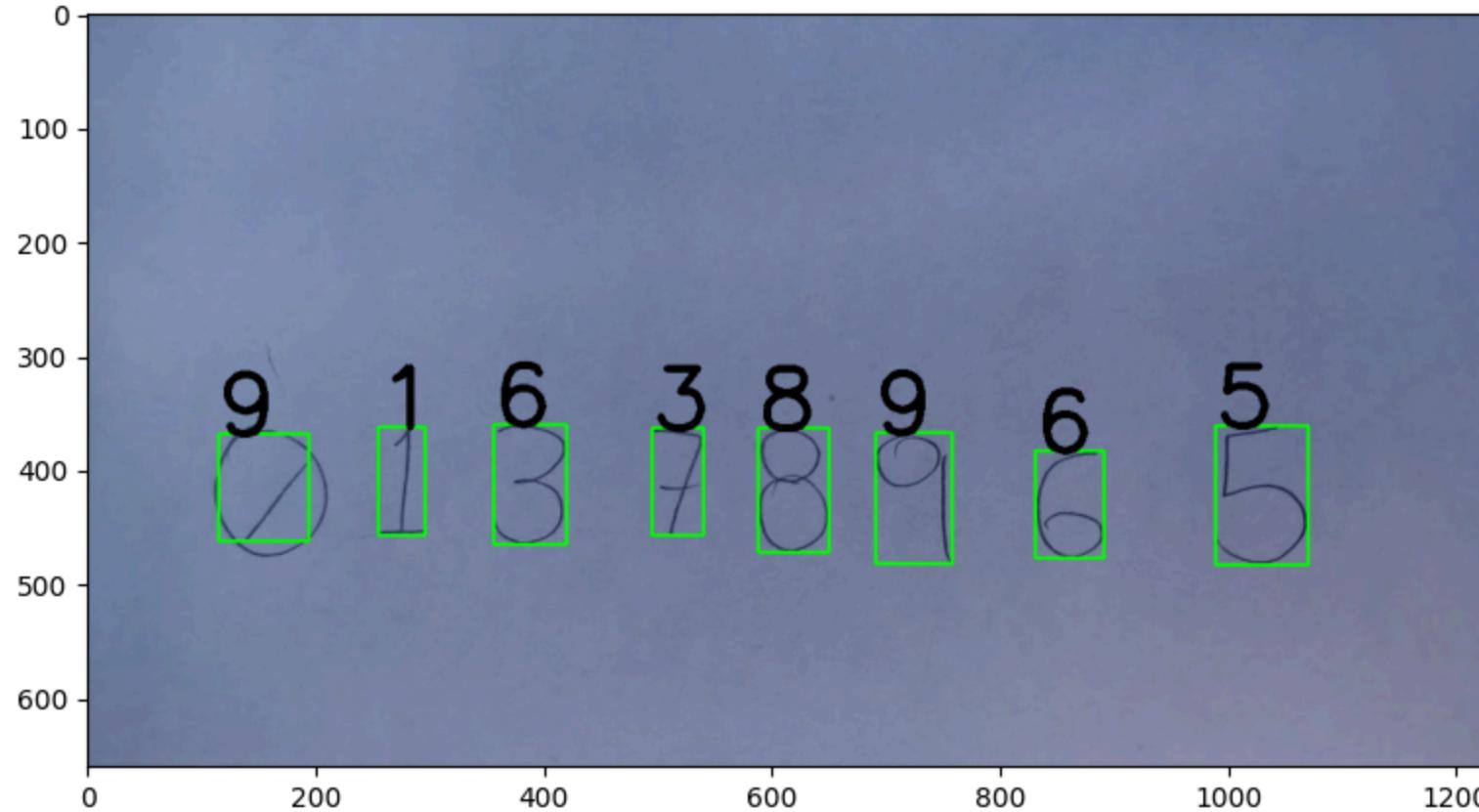
# Performance of the model in the test set
y_pred = svcCls.predict(X_test)

fig = plt.figure(figsize=(10, 5))
boundedImg = rawImg.copy()
for i, (startPoint, endPoint) in enumerate(boundariesPerDigit):
    cv.rectangle(boundedImg, startPoint, endPoint, (0,255,0), 2)
    cv.putText(boundedImg, str(y_pred[i]), startPoint, cv.FONT_HERSHEY_SIMPLEX, 2.5, (0,0,0), 5)

plt.imshow(boundedImg)
plt.title("Detection results with SVC")
plt.show()
```

```
print(f"ACCURACY: {accuracy_score(y_test[TEST], y_pred)}")
```

Detection results with SVC



ACCURACY: 0.625

DECISION TREE CLASSIFIER

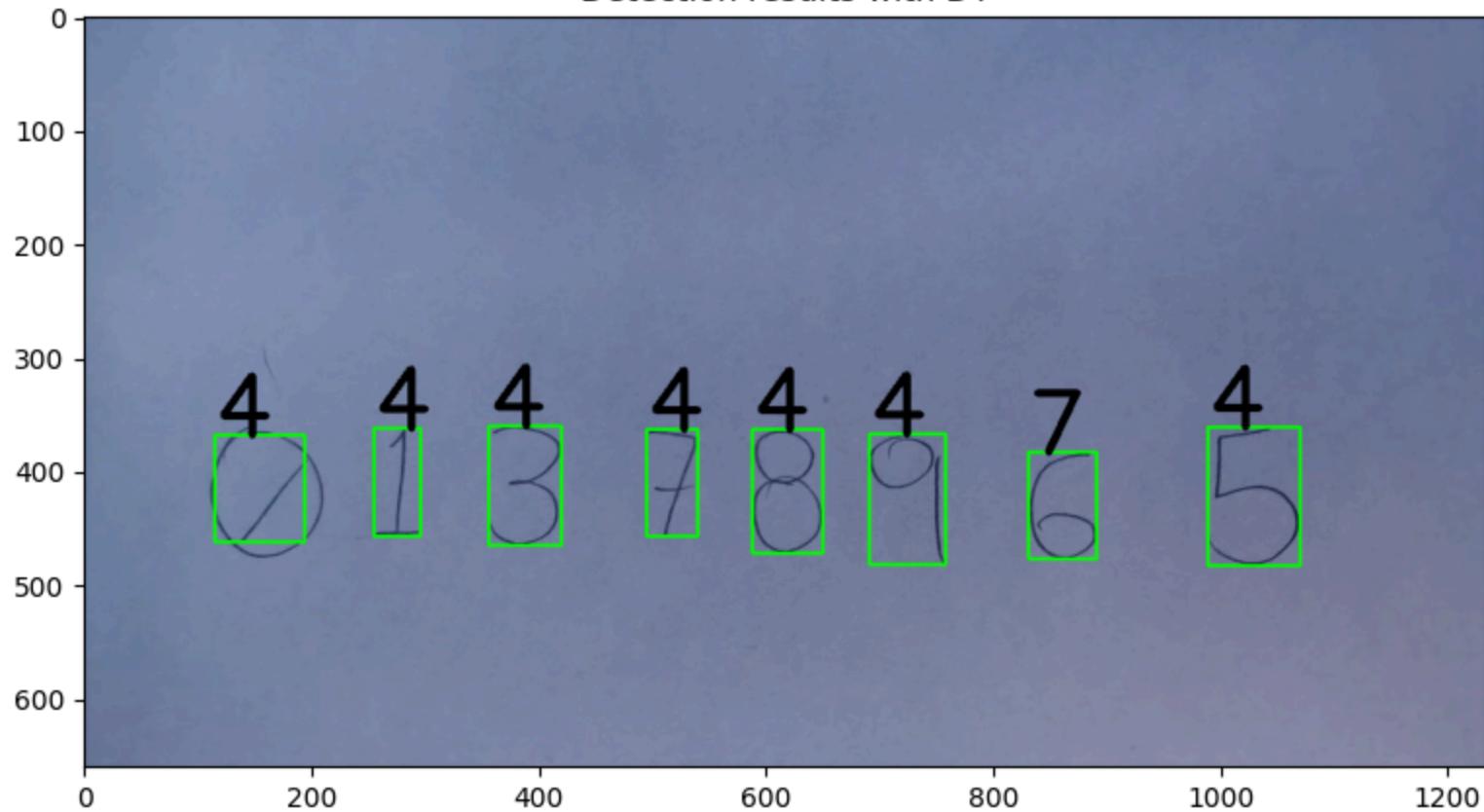
```
In [ ]: # Performance of the model in the test set
y_pred = dtCls.predict(X_test)

fig = plt.figure(figsize=(10, 5))
boundedImg = rawImg.copy()
for i, (startPoint, endPoint) in enumerate(boundariesPerDigit):
    cv.rectangle(boundedImg, startPoint, endPoint, (0,255,0), 2)
    cv.putText(boundedImg, str(y_pred[i]), startPoint, cv.FONT_HERSHEY_SIMPLEX, 2.5, (0,0,0), 5)

plt.imshow(boundedImg)
plt.title("Detection results with DT")
plt.show()

print(f"ACCURACY: {accuracy_score(y_test[TEST], y_pred)}")
```

Detection results with DT



ACCURACY: 0.0

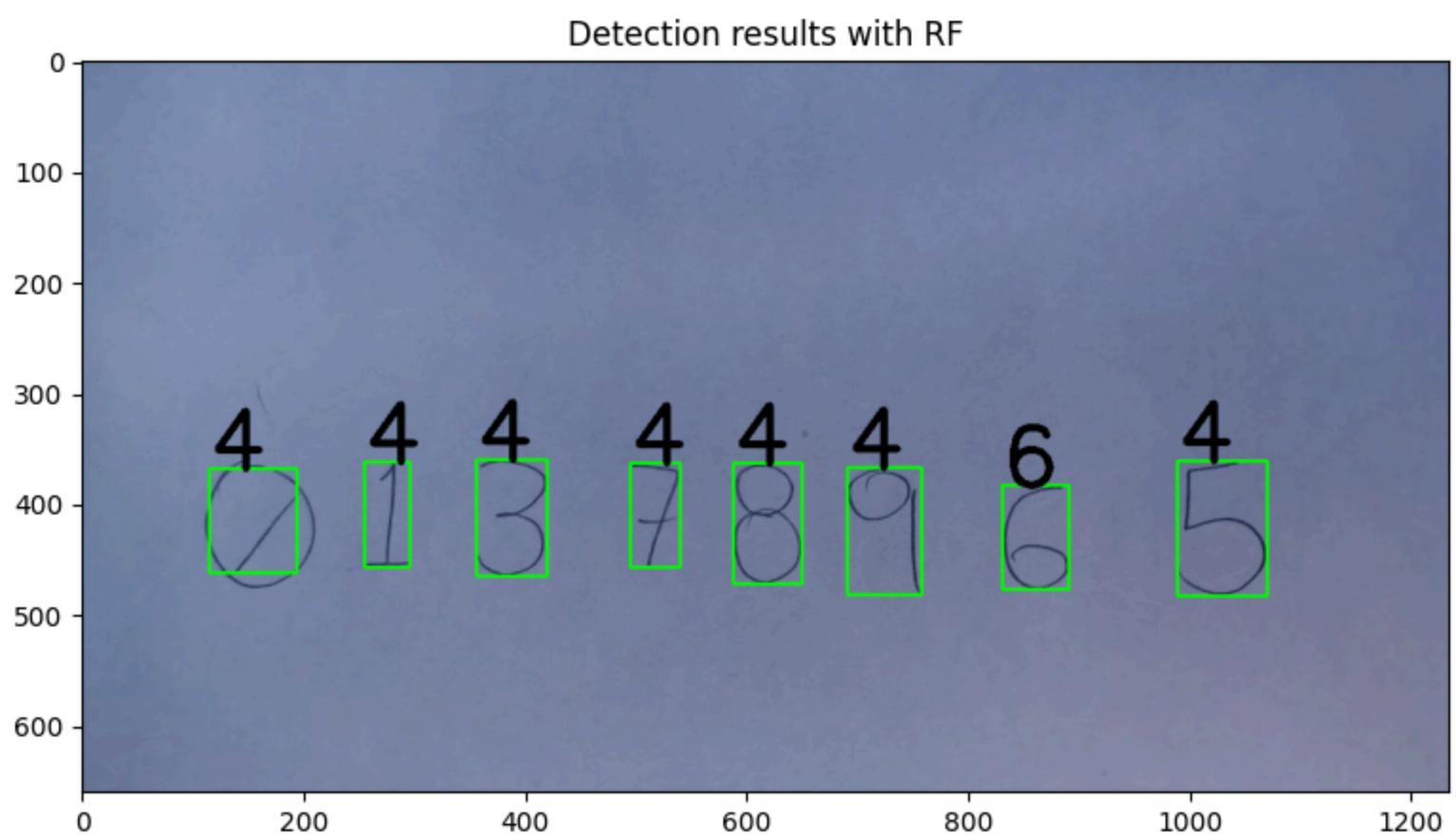
RANDOM FOREST CLASSIFIER

```
In [ ]: # Performance of the model in the test set
y_pred = rfCls.predict(X_test)

fig = plt.figure(figsize=(10, 5))
boundedImg = rawImg.copy()
for i, (startPoint, endPoint) in enumerate(boundariesPerDigit):
    cv.rectangle(boundedImg, startPoint, endPoint, (0,255,0), 2)
    cv.putText(boundedImg, str(y_pred[i]), startPoint, cv.FONT_HERSHEY_SIMPLEX, 2.5, (0,0,0), 5)

plt.imshow(boundedImg)
plt.title("Detection results with RF")
plt.show()

print(f"ACCURACY: {accuracy_score(y_test[TEST], y_pred)}")
```



ACCURACY: 0.125

CONCLUSIONS

All methods seemed to have good performance on training and validation sets. However, SVC is the only one that remained somewhat consistent when faced with the testing set. On the other hand, RFs seemed to have really good performance over the validation set, but when faced with testing samples, it performed poorly; in fact, it was the lowest of the three methods used.

Digit Recognition using Deep Learning and Convolutional Networks.

```
In [ ]: import math
import numpy as np
import cv2 as cv
import time
from scipy import stats
from matplotlib import pyplot as plt
# import skfuzzy as fuzz
import csv
import os
import pandas as pd
```

Generating the Test Set

```
In [ ]: class Region:
    def __init__(self):
        self.xm = 0
        self.ym = 0
        self.xx = 0
        self.yy = 0
        self.xy = 0
        self.xxx = 0
        self.yyy = 0
        self.xxy = 0
        self.xyy = 0
        self.Perimetro = 0
        self.Area = 0
        self.closed = False
        self.edgePoints = []
        self.red = 0
        self.green = 0
        self.blue = 0
    def addPoint(self,x,y,isedge,color):
        self.xm = self.xm + x
        self.ym = self.ym + y
        self.xx = self.xx + x*x
        self.yy = self.yy + y*y
        self.xy = self.xy + x*y
        self.xxx = self.xxx + x*x*x
        self.yyy = self.yyy + y*y*y
        self.xxy = self.xxy + x*x*y
```

```

    self.xyy = self.xyy + x*y*y
    self.Area = self.Area + 1
    self.red = self.red + color[0]
    self.green = self.green + color[1]
    self.blue = self.blue + color[2]

    if isedge > 0:
        self.Perimetro = self.Perimetro + 1
        if (isedge & 1) > 0: #up
            self.edgePoints.append((x+0.5,y+0.001))
        if (isedge & 2) > 0: #left
            self.edgePoints.append((x+0.001,y+0.5))
        if (isedge & 4) > 0: #down
            self.edgePoints.append((x+0.5,y+0.999))
        if (isedge & 8) > 0: #right
            self.edgePoints.append((x+0.999,y+0.5))

def addRegion(self,region):
    if (self.closed == False) and (region.closed == False) and (region.Area > 0):
        self.xm = self.xm + region.xm
        self.ym = self.ym + region.ym
        self.xx = self.xx + region.xx
        self.yy = self.yy + region.yy
        self.xy = self.xy + region.xy
        self.xxx = self.xxx + region.xxx
        self.yyy = self.yyy + region.yyy
        self.xxy = self.xxy + region.xxy
        self.xyy = self.xyy + region.xyy
        self.Area = self.Area + region.Area
        self.Perimetro = self.Perimetro + region.Perimetro
        self.red = self.red + region.red
        self.green = self.green + region.green
        self.blue = self.blue + region.blue
        for n in range(len(region.edgePoints)):
            self.edgePoints.append(region.edgePoints[n])

def closeRegion(self):
    if (self.Area > 0) and (self.closed == False):
        self.xm = self.xm/self.Area
        self.ym = self.ym/self.Area
        self.xx = self.xx/self.Area - self.xm*self.xm
        self.yy = self.yy/self.Area - self.ym*self.ym
        self.xy = self.xy/self.Area - self.xm*self.ym
        self.red = self.red/self.Area
        self.green = self.green/self.Area
        self.blue = self.blue/self.Area
        self.closed = True

```

```
In [ ]: def smoothPointSet(points,steps):
    lpts = len(points)
    for stp in range(steps):
        apt = np.array(points)
        for n in range(lpts):
            bidx = (n + lpts - 1) % lpts
            uidx = (n + lpts + 1) % lpts
            points[n] = ((apt[bidx][0]+apt[n][0]+apt[uidx][0])/3,(apt[bidx][1]+apt[n][1]+apt[uidx][1])/3)
    return points
```

```
In [ ]: def getInvariantFeatures(points, displayIndividual = True):
    outlinePoints = points

    # Convert the list of points to integer coordinates
    points = np.array([(int(pt[0]), int(pt[1])) for pt in outlinePoints[0]])

    xMin, yMin = np.min(points, axis=0)
    xMax, yMax = np.max(points, axis=0)

    # Create a binary image that fits the bounding box (a black BG image)
    binary_image = np.zeros((yMax - yMin + 1, xMax - xMin + 1), dtype=np.uint8)
    shifted_points = points - [xMin, yMin]

    # Draw the digit in the binary image (white FG)
    for x, y in shifted_points:
        binary_image[y, x] = 1

    return xMin, yMin, xMax, yMax
```

```
In [ ]: def orderPointSet(opoints,EdgeImage):
    orderpointsList = []
    if len(opoints) > 0:
        points = opoints.copy()
        neigbors = np.array([(1,0),(1,1),(0,1),(-1,1),(-1,0),(-1,-1),(0,-1),(1,-1),(0,0)])*0.5
        plen = len(points)
        n = 0
        last = 0
        inserted = 0
        orderpoints = []
        orderpoints.append(points[0])
        points.pop(0)
        lastidx = 0
        who = 0
        for inserted in range(plen-1):
            n = 0
            cpt = np.array(orderpoints[last])
```

```
edgetype = EdgeImage[int(cpt[1]),int(cpt[0])]
dis = 1
thepoints = np.array(points)
thesum = np.sum(abs(thepoints-cpt),axis=1)
lastdir = np.sum(np.square(neigbors-neigbors[lastidx]),axis=1)
dis = 10
idx = lastidx;
for n in range(len(points)):
    if thesum[n] <= 2:
        npoints = cpt + neigbors
        theNsum = np.sum(np.abs(npoints-thepoints[n]),axis=1) + 0.01*lastdir
        for nn in range(8):
            nedgetype = EdgeImage[int(npoints[nn,1]),int(npoints[nn,0])]
            dist = theNsum[nn] + 0.001*((edgetype ^ nedgetype) > 0)
            if dist < dis:
                dis = dist
                idx = nn
                who = n
    if dis < 1.0:
        lastidx = idx
        orderpoints.append(points[who])
        points.pop(who)
    else:
        if len(orderpoints) > 2:
            orderpointsList.append(orderpoints)
        orderpoints = []
        orderpoints.append(points[0])
        points.pop(0)
        last = 0
    last = len(orderpoints) - 1
if len(orderpoints) > 2:
    orderpointsList.append(orderpoints)
return orderpointsList
```

```
In [ ]: def SegmentationRegions(image,colorimg):
    regions = []
    si,sj = image.shape;
    fimage = image.astype(np.int16)
    EdgeImage = image.astype(np.int16)

    regionlabels = image[:, :];
    regionlabels = regionlabels.astype(np.int32)
    shortlabellut = np.zeros((int(si*sj/100+1),), dtype=np.int32)
    labellut = shortlabellut
    upKernel = np.array((
        [0, -1, 0],
        [0, 1, 0],
        [0, 0, 0],
```

```
[0, 0, 0]), dtype=np.int16)
upsameI = abs(cv.filter2D(fimage,-1,upKernel)) == 0
leftKernel = np.array([
[0, 0, 0],
[-1, 1, 0],
[0, 0, 0]), dtype=np.int16)
leftsameI = abs(cv.filter2D(fimage,-1,leftKernel)) == 0
downKernel = np.array([
[0, 0, 0],
[0, 1, 0],
[0, -1, 0]), dtype=np.int16)
downsameI = abs(cv.filter2D(fimage,-1,downKernel)) == 0
rightKernel = np.array([
[0, 0, 0],
[0, 1, -1],
[0, 0, 0]), dtype=np.int16)
rightsameI = abs(cv.filter2D(fimage,-1,rightKernel)) == 0
currentlabel=0
uplabelArray = regionlabels[0,:]
upsame = False
leftsame = False
thelabel = 0
uplabel = 0
leftlabel = 0
regions.append(Region())
for i in range(si):
    leftlabel = labellut[regionlabels[i,0]]
    for j in range(sj):
        color = colorimg[i,j,:]
        thelabel = leftlabel
        upsame = False
        leftsame = False
        EdgeImage[i,j] = 1*(not upsameI[i,j]) + 2*(not leftsameI[i,j]) + 4*(not downsameI[i,j]) + 8*(not rightsameI[i,j])
        if i>0:
            uplabel = uplabelArray[j]
            upsame = upsameI[i,j]
            if upsame:
                thelabel = uplabel
        if j>0:
            leftsame = leftsameI[i,j]
            if leftsame:
                if thelabel > leftlabel:
                    thelabel = leftlabel
            if upsame:
                thelabel = labellut[thelabel]
                labellut[leftlabel] = thelabel
                labellut[uplabel] = thelabel
        if (not(upsame) and not(leftsame)):
```

```

currentlabel = currentlabel + 1
regions.append(Region())
thelabel = currentlabel
if (currentlabel >= labellut.size):
    labellut = np.append(labellut, shortlabellut)
labellut[currentlabel] = currentlabel
else:
    thelabel = labellut[thelabel]
regionlabels[i,j] = thelabel
regions[thelabel].addPoint(j,i,EdgeImage[i,j],color)
leftlabel = thelabel
uplabelArray = regionlabels[i,:]
print(len(regions))
lreg = len(regions)
changes = 1
while changes > 0:
    changes = 0
    for idx in range(lreg):
        newlabel = labellut[idx]
        if regions[idx].Area > 0 and idx != newlabel:
            changes = changes + 1
            regions[newlabel].addRegion(regions[idx])
            regions[idx].__init__()
            labellut[idx] = labellut[newlabel]
    print(changes)
closed = 0
for n in range(lreg):
    if (n == labellut[n]) and (regions[n].Area > 0):
        closed = closed + 1
        regions[n].closeRegion()
print(closed)
regionlabels = labellut[regionlabels]
return regionlabels, regions, EdgeImage

```

In []:

```

PERIMETERS = [
    (300, 2000), # 0
    (100, 700),
    (150, 700), # 2
    (250, 500),
    (200, 700), # 4
    (200, 1000),
    (200, 700), # 6
    (150, 700),
    (400, 700), # 8
    (200, 700)
]

```

```
PERIMETERS_TEST = [250, 1000]
```

```
In [ ]: for digit in ['Den', 'Luis', 'Rivers']:
```

```
    print("#####")
    print()
    print(f"EXTRACTION OF ID {digit}")
    print()
    print("#####")

    frame = cv.imread(os.path.join("digitsTest", f"{digit}.jpeg"))
    rawImg = frame.copy()

    kernel = np.ones((5,5),np.uint8)
    bkgnd = cv.dilate(frame,kernel,iterations = 1)
    bkgnd = cv.dilate(bkgnd,kernel,iterations = 1)

    frame = (bkgnd - frame)
    frgnd = frame.copy()

    kernel = np.ones((3,3),np.uint8)
    frame = cv.dilate(frame,kernel,iterations = 1)

    green = frame[:, :, 2]
    ret, img = cv.threshold(green, 32, 100, 0)

    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    ax[0].imshow(cv.cvtColor(bkgnd, cv.COLOR_BGR2RGB))
    ax[0].set_title("Background")

    ax[1].imshow(cv.cvtColor(frgnd, cv.COLOR_BGR2RGB))
    ax[1].set_title("Foreground (Just Numbers)")

    ax[2].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
    img_to_save = cv.cvtColor(img, cv.COLOR_BGR2RGB)
    ax[2].set_title("Refined Numbers")
    plt.show()

    #####
    labeled, regions, EdgeImage = SegmentationRegions(img,frame)

    labeled = labeled % 21

    img = cv.normalize(labeled, None, 0, 255, cv.NORM_MINMAX, cv.CV_8U)
    im_color = cv.applyColorMap(img, cv.COLORMAP_HSV)
    im_color = cv.medianBlur(im_color,3)
```

```

RGB_im = cv.cvtColor(im_color, cv.COLOR_BGR2RGB)

# #####
fig, ax = plt.subplots(1,2, figsize=(10, 5))
boundaries = []
for n in range(len(regions)-2):
    idx = n+2
    if regions[idx].Perimetro > PERIMETERS_TEST[0]:
        if regions[idx].Perimetro < PERIMETERS_TEST[1]:

            pointlist = orderPointSet(regions[idx].edgePoints, EdgeImage)
            smoothpoints = np.array(smoothPointSet(pointlist[0],3))
            xMin, yMin, xMax, yMax = getInvariantFeatures(pointlist, displayIndividual=False)

            boundaries.append(((xMin, yMin), (xMax, yMax)))

            ax[0].plot(smoothpoints[:,0],smoothpoints[:,1])

    ax[0].set_ylim(frame.shape[0], 0)
    ax[0].set_xlim(0, frame.shape[1])
    ax[0].set_title("Extracted Regions")

    boundedImg = rawImg.copy()
    for ind, (startPoint, endPoint) in enumerate(boundaries):
        cv.rectangle(boundedImg, startPoint, endPoint, (0,255,0), 2)
        #print(rawImg.shape, 'raw image shape')
        xd = cv.cvtColor(img_to_save, cv.COLOR_BGR2RGB)[startPoint[1]:endPoint[1], startPoint[0]:endPoint[0]]
        cv.imwrite(f'TestSet/{digit}/{ind}.jpg', xd)

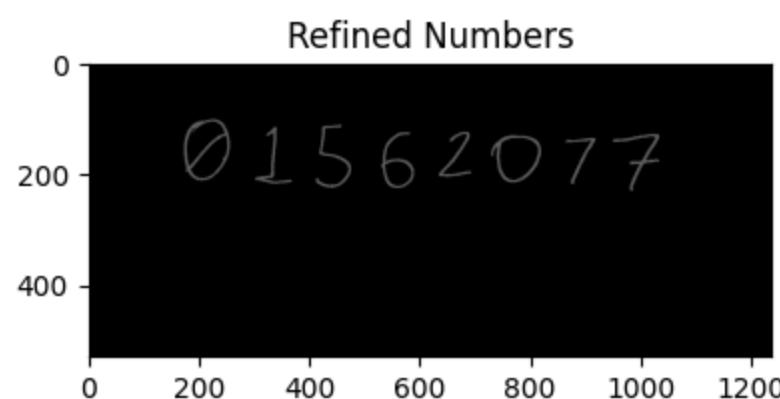
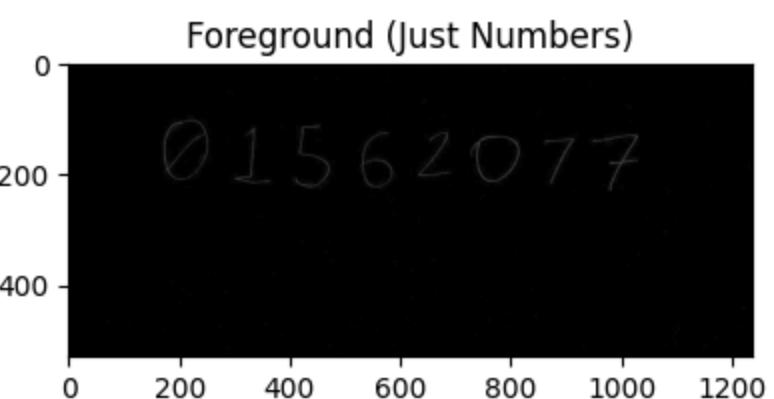
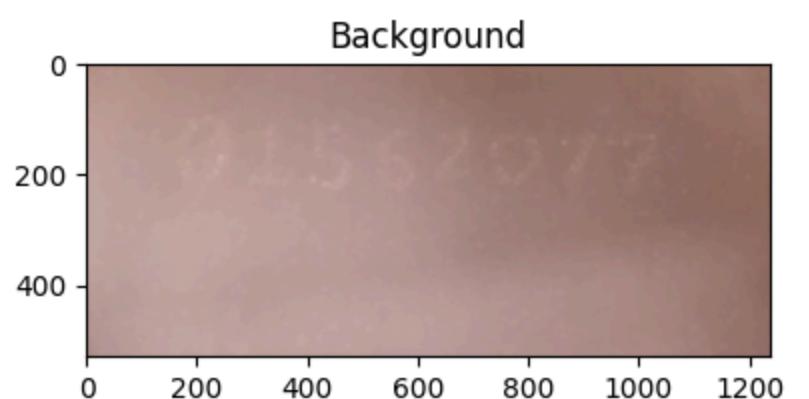
    ax[1].imshow(boundedImg)
    ax[1].set_title("Detected objects from original image")
    plt.show()

#####

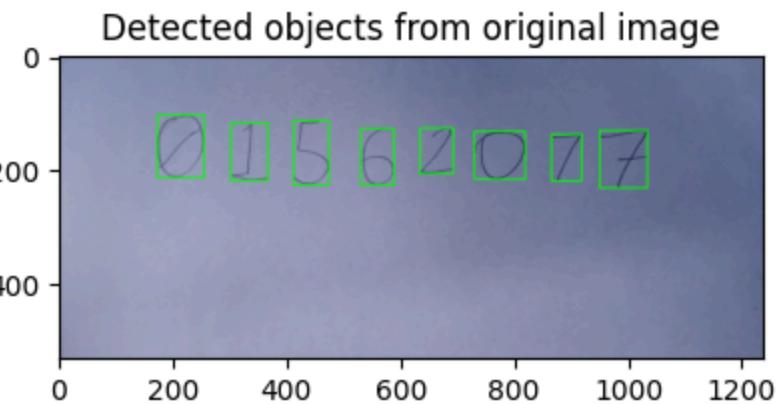
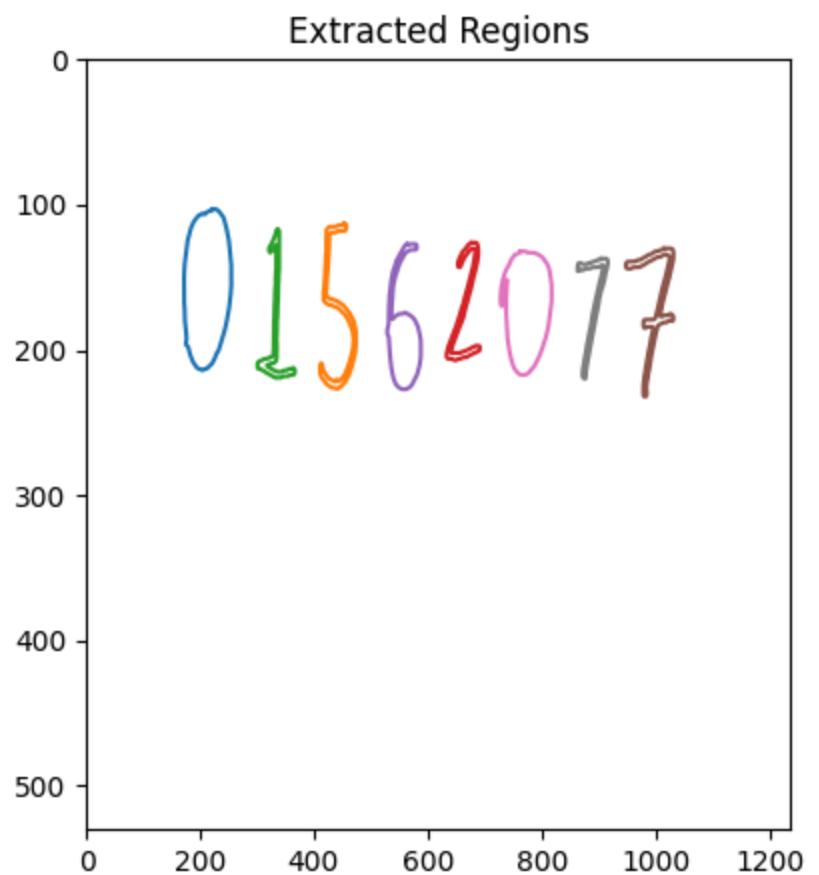
```

EXTRACTION OF ID Den

```
#####
```



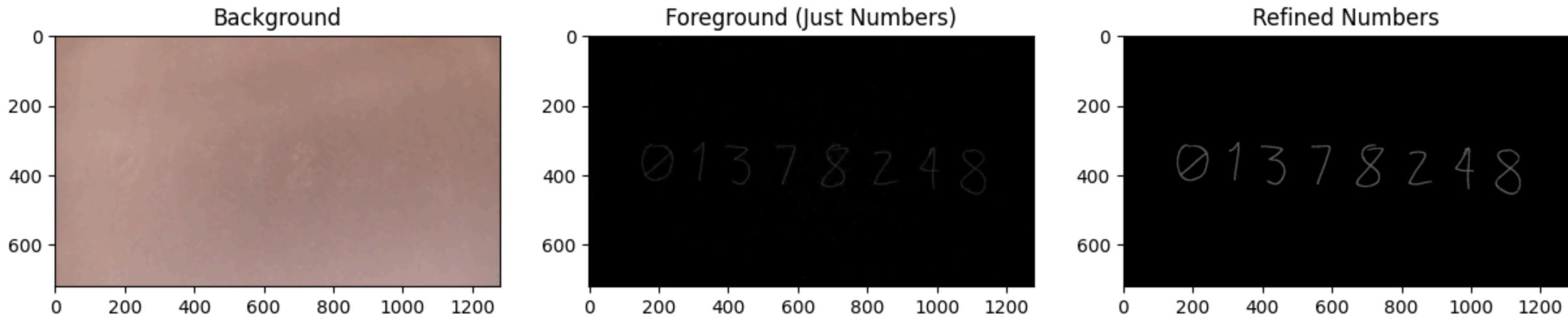
878
860
14
0
17



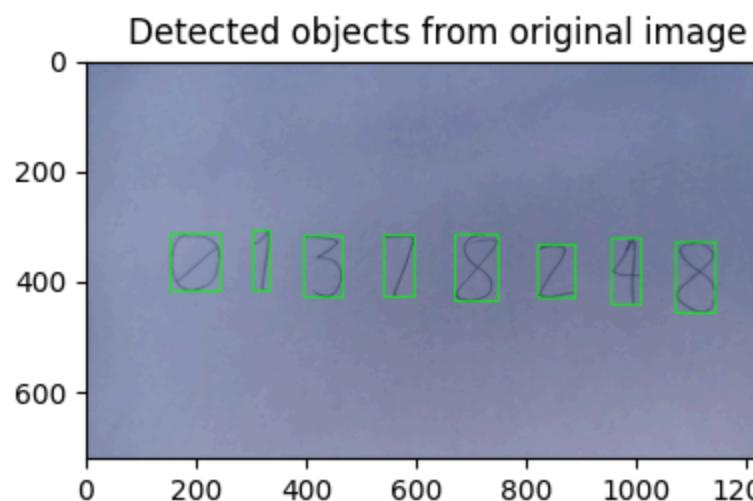
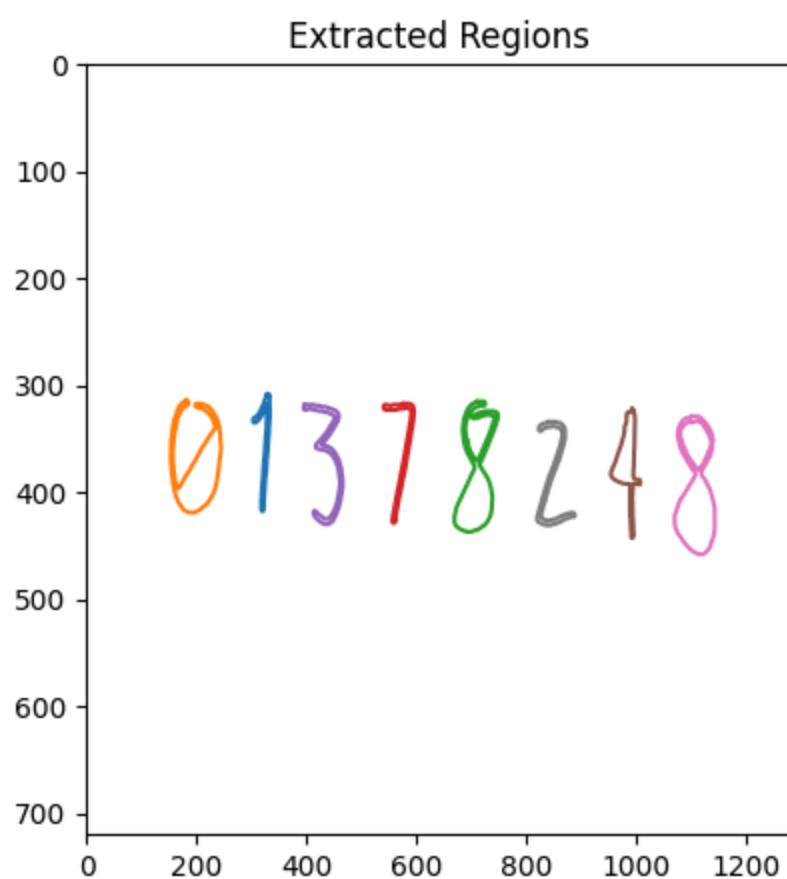
```
#####
#####
```

EXTRACTION OF ID Luis

```
#####
#####
```

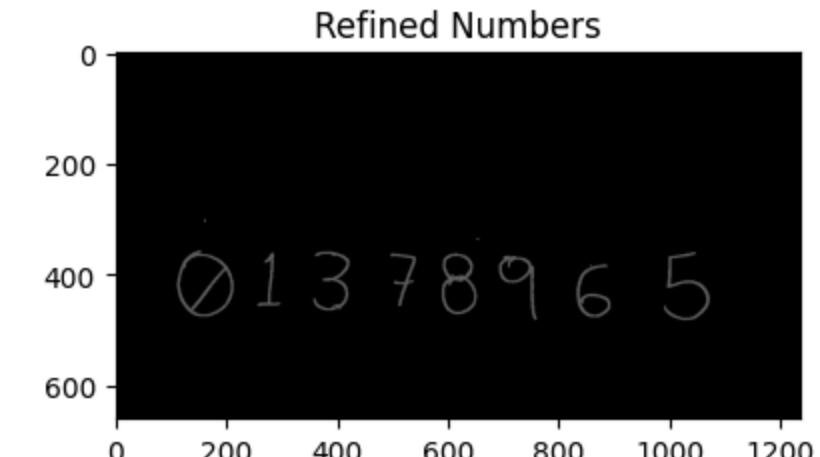
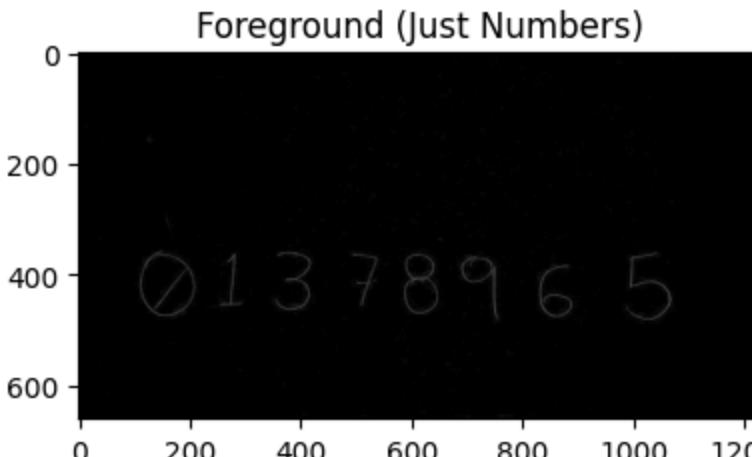
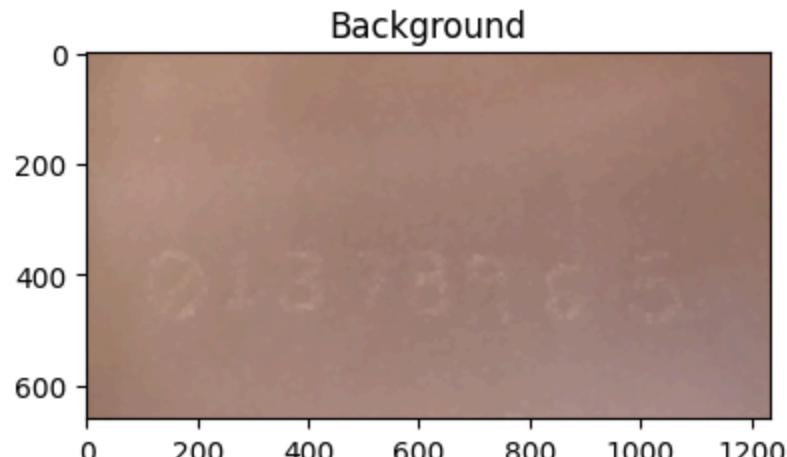


971
956
12
0
14



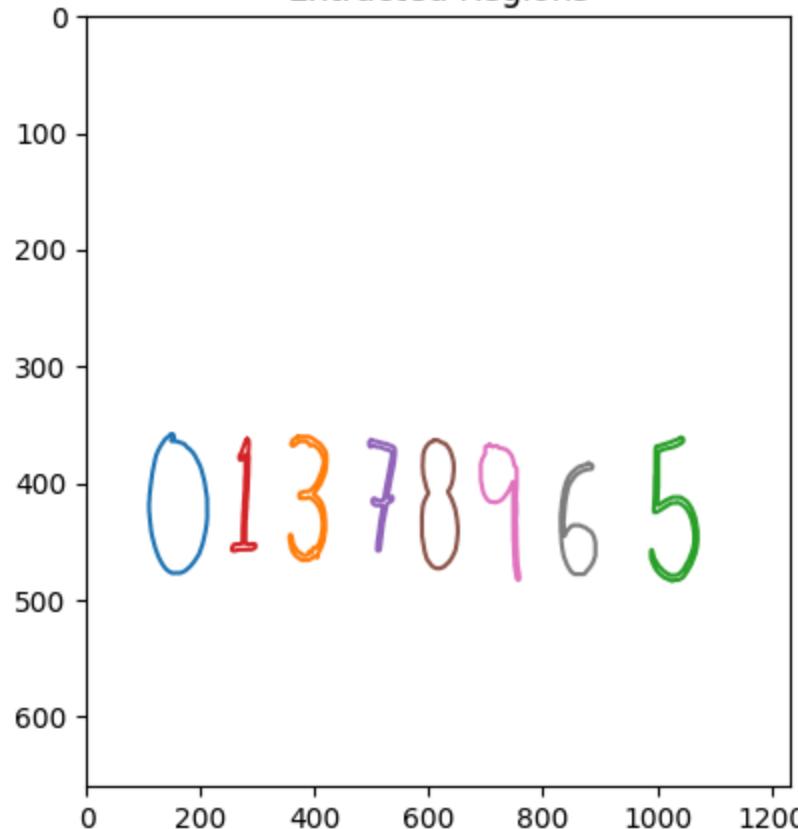
#####
EXTRACTION OF ID Rivers

#####
Background

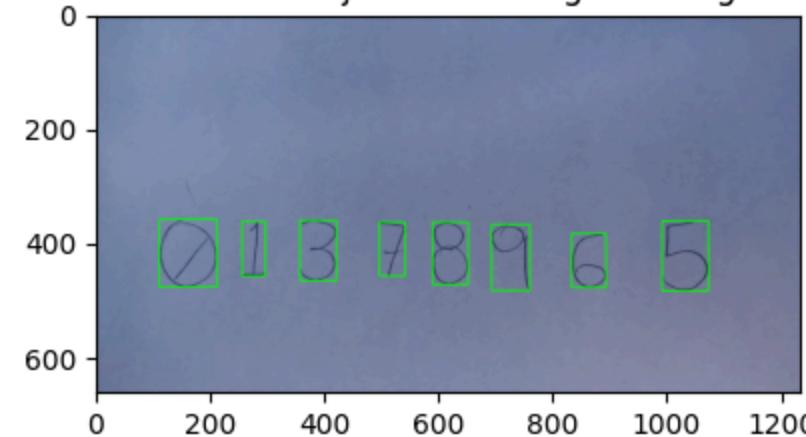


```
768  
745  
16  
0  
22
```

Extracted Regions



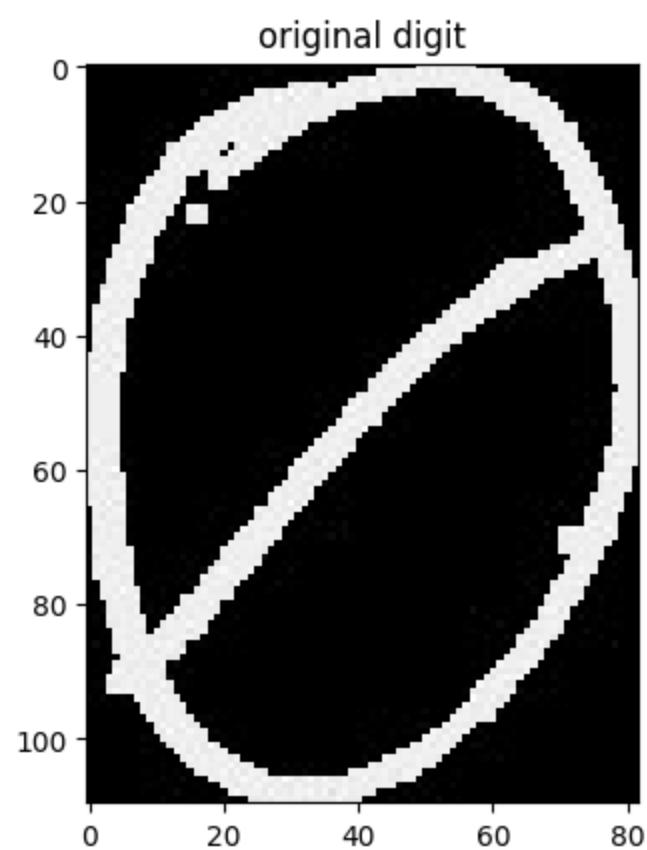
Detected objects from original image

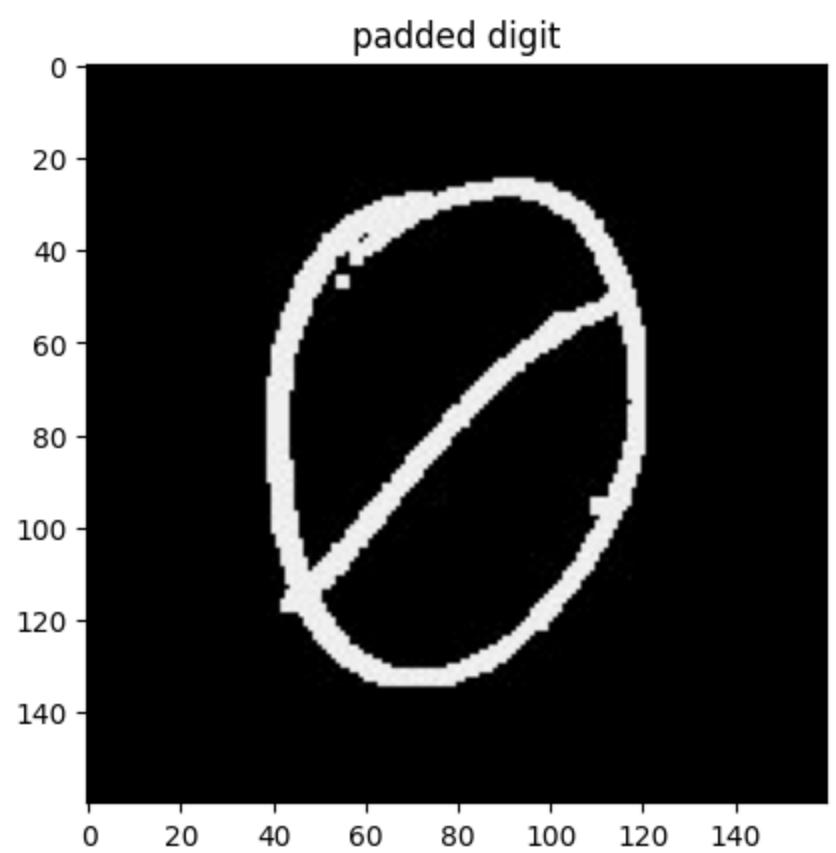


Formatting digits to match MNIST shape

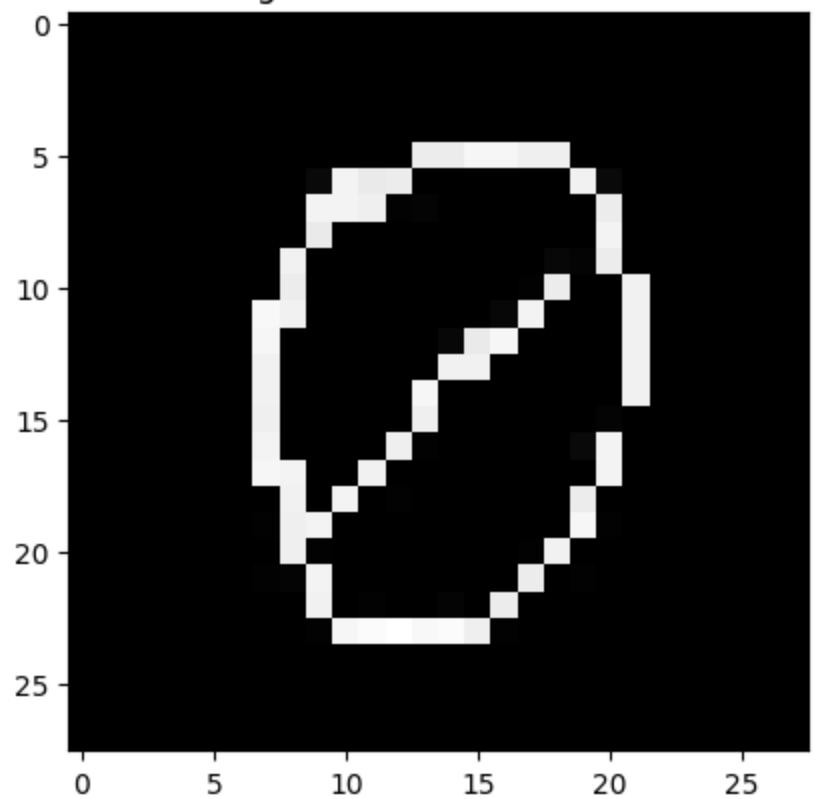
```
In [ ]: for person in ['Den', 'Luis', 'Rivers']:  
    files = sorted(os.listdir(f'TestSet/{person}'))  
    for file in files:  
        img = cv.imread(f'TestSet/{person}/{file}', cv.IMREAD_GRAYSCALE)  
        if person == 'Den' and file == '0.jpg':  
            plt.imshow(img, cmap='gray')  
            plt.title('original digit')  
            plt.show()  
  
        pad = 25  
        #large side
```

```
ls = max(img.shape)
#large index
li = img.shape.index(ls)
# pad index
pi = 1 - li
# pad width
pw = [[0, 0], [0, 0]]
# pad size
ps = abs(img.shape[0] - img.shape[1])
pw[pi] = [ps // 2 + pad, ps // 2 + pad]
pw[li] = [pad, pad]
xd = np.pad(img, pad_width=pw, mode='constant')
if person == 'Den' and file == '0.jpg':
    plt.imshow(xd, cmap='gray')
    plt.title('padded digit')
    plt.show()
mnist_size = cv.resize(xd, (28, 28),
                       interpolation = cv.INTER_NEAREST)
if person == 'Den' and file == '0.jpg':
    plt.imshow(mnist_size, cmap='gray')
    plt.title('digit resized to mnist size')
    plt.show()
cv.imwrite(f'TestSet/{person}/{file[:-4]}ms.jpg', mnist_size)
```





digit resized to mnist size



In []:

Digit Recognition using Deep Learning and Convolutional Networks.

```
In [ ]: import tensorflow as tf
import tensorflow_datasets as tfds
import cv2 as cv
import os
import matplotlib.pyplot as plt
import numpy as np
```

Training the model on MNIST

```
In [ ]: (ds_train, ds_test), ds_info = tfds.load(
    'mnist',
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,
)
```

```
In [ ]: def normalize_img(image, label):
    """Normalizes images: `uint8` -> `float32`."""
    return tf.cast(image, tf.float32) / 255., label

ds_train = ds_train.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_train = ds_train.cache()
ds_train = ds_train.shuffle(ds_info.splits['train'].num_examples)
ds_train = ds_train.batch(128)
ds_train = ds_train.prefetch(tf.data.AUTOTUNE)
```

```
In [ ]: ds_test = ds_test.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_test = ds_test.batch(128)
ds_test = ds_test.cache()
ds_test = ds_test.prefetch(tf.data.AUTOTUNE)
```

```
In [ ]: model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(10, (3, 3),#, input_shape=(28, 28)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10),
```

```
    tf.keras.layers.Softmax()
])
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),#from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

model.fit(
    ds_train,
    epochs=3,
    #validation_data=ds_test,
)
```

```
Epoch 1/3
469/469 35s 46ms/step - loss: 0.4017 - sparse_categorical_accuracy: 0.8829
Epoch 2/3
469/469 23s 49ms/step - loss: 0.1101 - sparse_categorical_accuracy: 0.9668
Epoch 3/3
469/469 41s 49ms/step - loss: 0.0698 - sparse_categorical_accuracy: 0.9787
```

```
Out[ ]: <keras.src.callbacks.history.History at 0x79cd454ff430>
```

Evaluate model on MNIST test set

```
In [ ]: score = model.evaluate(ds_test, verbose=0)
print('loss=', score[0])
print('accuracy=', score[1])

loss= 0.09519213438034058
accuracy= 0.9711999893188477
```

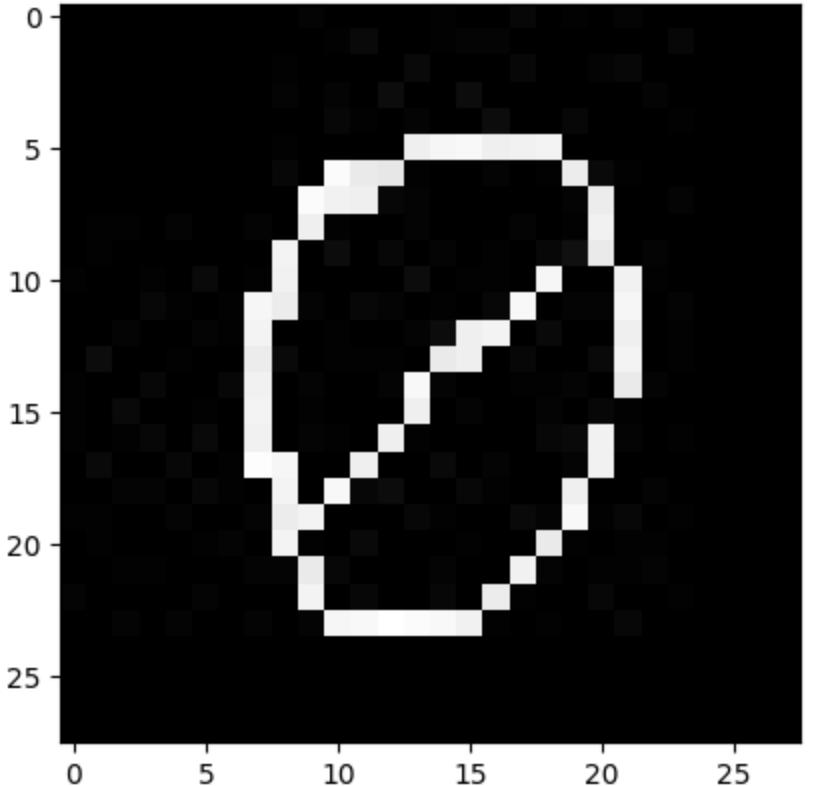
Predictions on our own student IDs

```
In [ ]: all_preds = []

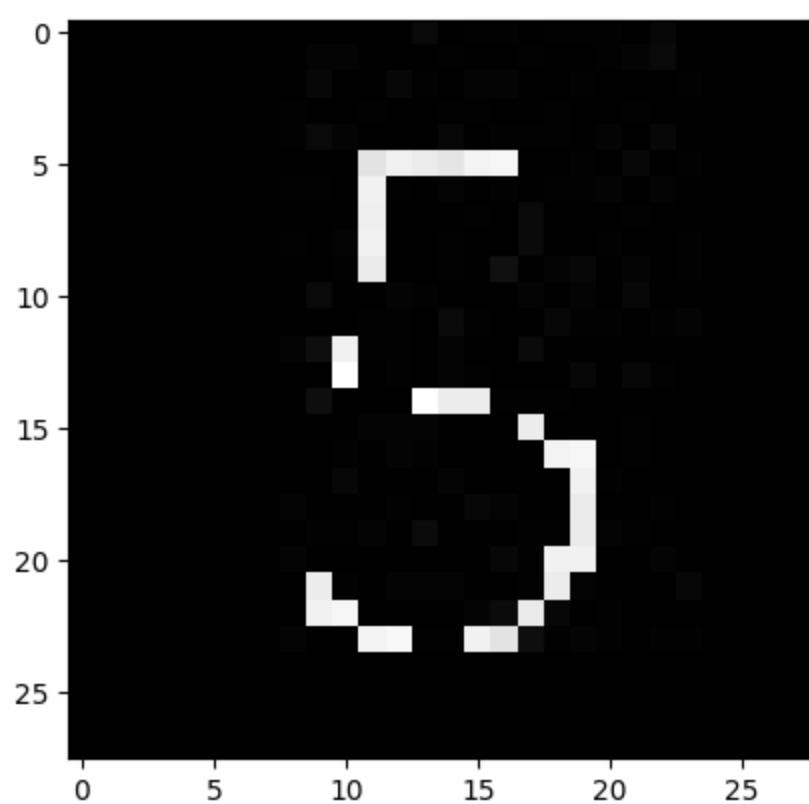
for person in ['Den', 'Luis', 'Rivers']:
    curr_preds = []
    for file in sorted(os.listdir(f'TestSet/{person}')):
        if file[-4:] != '.jpg':
            continue

        xd = cv.imread(f'TestSet/{person}/{file}', cv.IMREAD_GRAYSCALE)
        xd = xd.astype(np.float16) / 255.0
        if person == 'Den':
            plt.imshow(xd, cmap='gray')
```

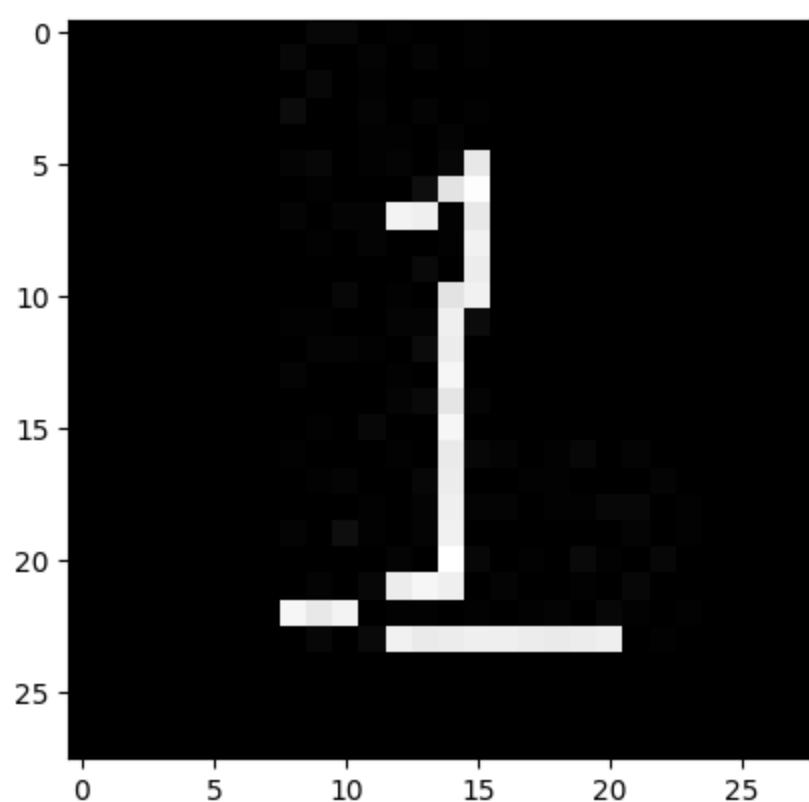
```
    plt.show()
    xdd = xd.reshape((1, 28, 28))
    pred = model.predict(xdd)[0].tolist()
    ind_pred = pred.index(max(pred))
    curr_preds += [ind_pred]
    if person == 'Den':
        print(f'Prediction for previous image {ind_pred}')
    all_preds += [curr_preds]
```



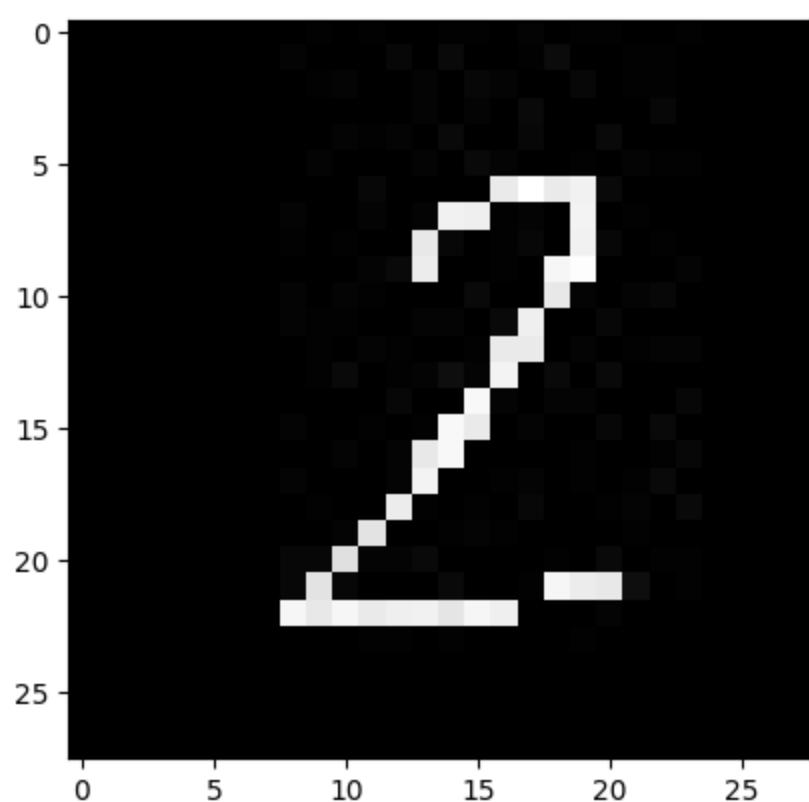
1/1 ————— 0s 41ms/step
Prediction for previous image 0



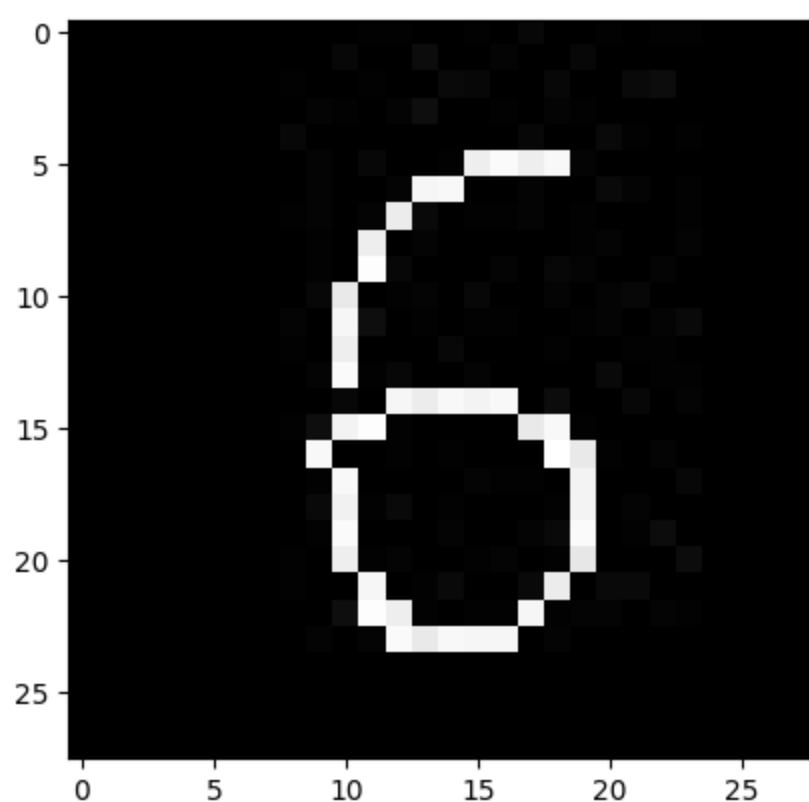
1/1  0s 25ms/step
Prediction for previous image 5



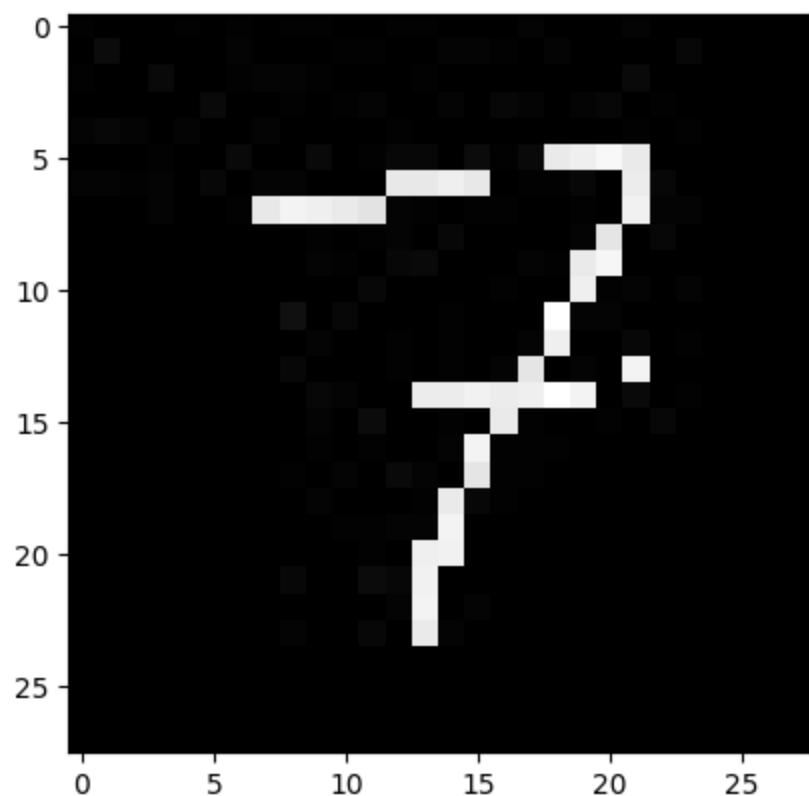
1/1 0s 23ms/step
Prediction for previous image 1



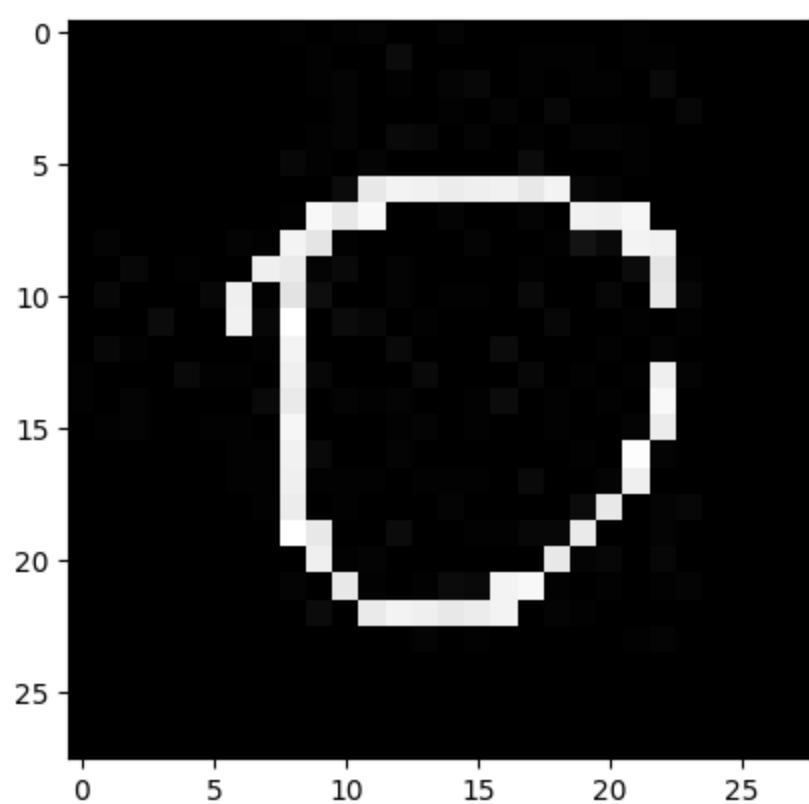
1/1  0s 25ms/step
Prediction for previous image 1



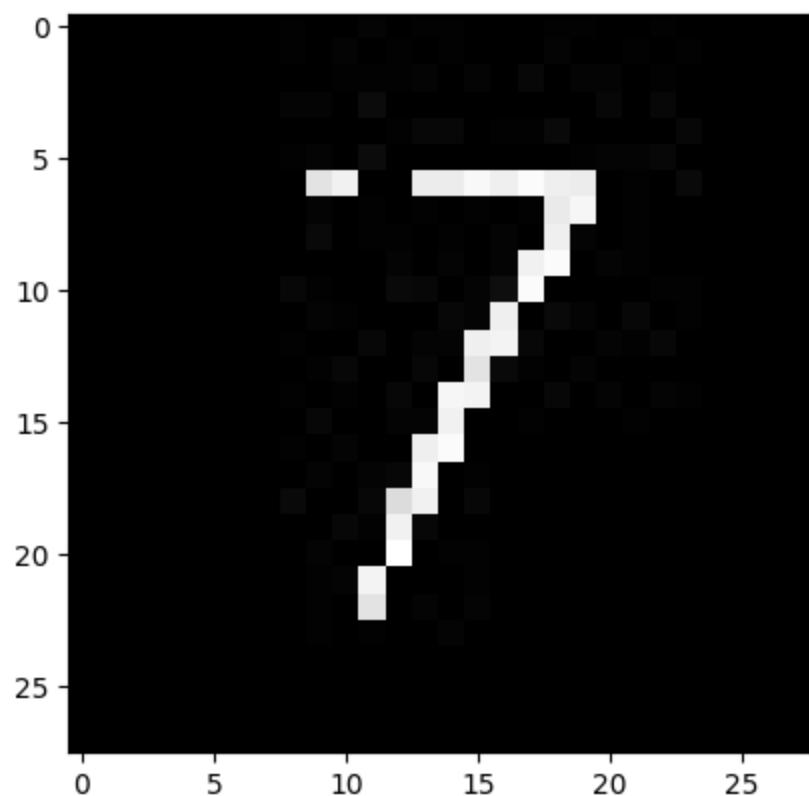
1/1 0s 22ms/step
Prediction for previous image 5



1/1 0s 24ms/step
Prediction for previous image 7



1/1 0s 24ms/step
Prediction for previous image 5



1/1 0s 23ms/step
Prediction for previous image 7
1/1 0s 23ms/step
1/1 0s 34ms/step
1/1 0s 31ms/step
1/1 0s 25ms/step
1/1 0s 24ms/step
1/1 0s 24ms/step
1/1 0s 26ms/step
1/1 0s 27ms/step
1/1 0s 29ms/step
1/1 0s 29ms/step
1/1 0s 40ms/step
1/1 0s 29ms/step
1/1 0s 31ms/step
1/1 0s 33ms/step
1/1 0s 23ms/step
1/1 0s 23ms/step

Evaluating the model with our student IDs

```
In [ ]: preds = np.array(all_preds).flatten()
print(preds, 'Predictions')
```

```
ground_truth = np.array([0, 5, 1, 2, 6, 7, 0, 7, 1, 0, 8, 7, 3, 4, 8, 2, 0, 3, 5, 1, 7, 8, 9, 6])
print(ground_truth, 'Ground truth')
from sklearn.metrics import classification_report
```

```
print(classification_report(ground_truth, preds))
```

```
[0 5 1 1 5 7 5 7 1 2 9 7 3 4 5 2 2 3 5 1 7 4 9 6] Predictions
[0 5 1 2 6 7 0 7 1 0 8 7 3 4 8 2 0 3 5 1 7 8 9 6] Ground truth
      precision    recall   f1-score   support

          0         1.00     0.25     0.40       4
          1         0.75     1.00     0.86       3
          2         0.33     0.50     0.40       2
          3         1.00     1.00     1.00       2
          4         0.50     1.00     0.67       1
          5         0.40     1.00     0.57       2
          6         1.00     0.50     0.67       2
          7         1.00     1.00     1.00       4
          8         0.00     0.00     0.00       3
          9         0.50     1.00     0.67       1

  accuracy                           0.67      24
  macro avg       0.65     0.72     0.62      24
weighted avg       0.70     0.67     0.62      24
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Discussion and Conclusion

Classes 0, 3, 6 and 7 had a perfect precision, which means that every time the model predicted these digits, it was correct. Classes 1, 3, 4, 5, 7 and 9 had a recall of 1, which means that the model managed to identify all the cases in our IDs where these digits were present. Digits 2 and eight were particularly challenging, our hypothesis is that the model confused 2 with 0 and several numbers with eight, like 4, 5, and 9, because of the similarities these have with eight.

So, out of all the digits in our three ids, the model manages to correctly identify 67% of them, which is better than a random prediction, because a random prediction would get a 10% accuracy, but nevertheless, there is a very clear room for improvement. To get better results, more experiments should be carried out with different neural network architectures, and more strategies that allow the digits to be as clear as possible should be explored.