Proiect TIA
Clasificarea emoțiilor

- Drăgan Pavel 331AA -

• Obiectivul proiectului

Scopul principal al acestui proiect este de a dezvolta și implementa un sistem de recunoaștere a emoțiilor umane din imagini, utilizând tehnici de prelucrare a imaginilor și algoritmi de învățare automată. Prin utilizarea unui set de date divers și reprezentativ, proiectul vizează antrenarea unui model capabil să identifice și să clasifice corect expresiile faciale asociate diferitelor emoții umane. În paralel, se urmărește explorarea eficienței și a limitărilor algoritmului K-Means, utilizat pentru a organiza datele în clustere și interpretarea corespunzătoare a diverselor imagini.

Astfel, obiectivul final constă în dezvoltarea unui sistem robust și precis, capabil să interpreteze în mod adecvat emoțiile umane din imagini, precum și aprofundarea cunoștințelor asupra tehnicilor de învățare automată și a inteligenței artificiale în general.

• Obținerea și organizarea setului de date

Setul de date:

Înainte de a începe măcar să caut imagini pentru setul de date a trebuit să hotărăsc pentru ce emoții voi începe să adun datele. Inițial am început prin a căuta imagini pe internet doar pentru emoțiile de fericire și tristețe deoarece am considerat că sunt cele mai contrastante emoții și aș putea obține rezultate bune folosindu-le. Am descoperit apoi mai multe dataset-uri deja organizate precum: AffectNet, Cohn-Kanade Dataset (CK+), EMOTIC, Google Facial Expression Comparison Dataset. După recomandările primite în timpul laboratorului am decis să folosesc în preponderență dataset-ul FER2013 găsit pe kaggle.

Setul de date ales cuprinde o multitudine de imagini ce sunt împărțite în emoțiile: furie, dezgust, frică, fericire, tristețe, surprindere și neutru. De asemenea datele sunt organizate și în secțiuni de antrenare și testare. Imaginile au o dimensiune uniformă de 48x48 pixeli, cuprind doar fețe care sunt aproximativ centrate și sunt deja în gray-scale. Uniformizarea dimensiunii pozelor și aducerea lor în gray-scale erau deja etape pe care plănuiam să le parcurg deci faptul că acest dataset era deja organizat în acest mod îl face cu atât mai potrivit pentru acest proiect.

Prelucrarea datelor și problemele întâmpinate:

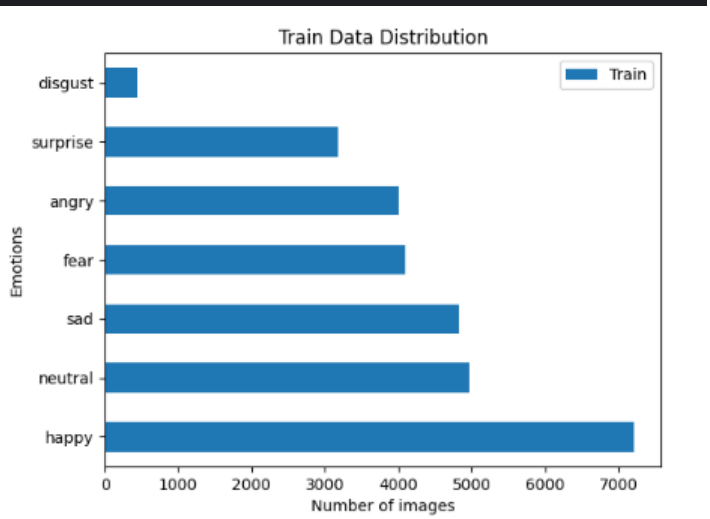
O primă problemă întâmpinată totuși la acest set de date era distribuția imaginilor pe diferitele emoții. Fericirea era cea mai răspândită emoție, fiind un punct de referință pentru ghicirea aleatorie de aproximativ 24.4%.

Astfel că primele linii de cod scrise au fost folosite pentru a încerca uniformizarea setului de date.

Programul de mai jos afiseaza numarul pozelor din fiecare fisier de antrenare (am atașat și un grafic care prezintă și mai bine problema).

```
# Print number of images for each class
folder_path = 'data/*
for cls in classes:
    path = os.path.join(folder_path, 'train', cls)
    lst = os.listdir(path)
    number_files = len(lst)
    print(cls, ': ', number_files)
```

```
angry :  3995
disgust :  436
fear :  4097
happy :  7215
neutral :  4965
sad :  4830
surprise :  3171
```



Și pentru folderele de antrenare situația era similară așa am luat ca punct de referință folderul cu cele mai puține imagini și am selectat din celelalte foldere imagini aleatoare care să egaleze numărul de imagini ce reprezentau emoția „dezgust”, folosind următoarea secvență de cod:

```python

# Create a empty folder
os.makedirs('zew_data')
# Create "train" folder inside new_data folder
os.makedirs('zew_data/train')
# Crate sub-folders
for cls in classes:
    os.makedirs('zew_data/train/'+cls)

Copy 436 files to new folder
import shutil
import random

num_files = 436

for cls in classes:
    # Downloaded original training image folder path for face emotion recognition
    src_path = os.path.join('D:\Proiecte TIA\cod emotii incepput\data', 'train', cls)
    # Sub folder path
    dst_path = os.path.join('D:\Proiecte TIA\cod emotii incepput\zew_data', 'train', cls)
    src_files = os.listdir(src_path)
    # Select random 436 images from source directory
    src_select_files = random.sample(src_files, num_files)

    # Copy selected images to destination folder
    for file_name in src_select_files:
        full_file_name = os.path.join(src_path, file_name)
        if os.path.isfile(full_file_name):
            shutil.copy(full_file_name, dst_path)

```

Rezultatul a fost următorul:

```
D:\Python\python.exe "D:\Proiecte TIA\cod emotii incepput\main.py"
angry :  436
disgust :  436
fear :  436
happy :  436
neutral :  436
sad :  436
surprise :  436
```

Acest lucru nu a ajutat doar la uniformizarea setului de date ci și la depășirea limitărilor de hardware pe care le-am întâmpinat. Nu dispun de un laptop foarte performant și antrenarea pe un număr mai redus ca acesta durează destul de mult timp, deci dacă aș fi fost nevoit să antrenez de

fiecare dată când testam o nouă modificare pe un set de 4000 de imagini per emoție timpul de așteptare ar fi fost cu siguranță mult mai ridicat.

Acum că setul de date era organizat cât mai uptim urmează antrenarea propriu zisă a modelului.

• Algoritmul utilizat

Etapa de cercetare:

În timp ce căutam cel mai bun algoritm pe care să îl implementez în proiectul meu am dat întâlnit multe programe care foloseau modele pre-antrenate cum ar fi librăria DeepFace pentru Python care este un proiect open-source sub licența MIT. Am întâlnit de asemenea și multe modele care făceau recunoaștere de emoții în timp real având ca input un semnal video.

Deși folosirea acestor modele și algoritmi ar fi dus poate la un proiect mai complez și o rată mai mare de succes pentru predicțiile pe care urma să le fac am decis să mă bazez în principal pe algoritmul K-Means prezentat la curs și laborator care a fost folosit pentru clustering-ul imaginilor în funcție de caracteristicile lor.

Biblioteci Python utilizate:

- os: Manipulează fișiere și directoare în sistemul de operare.

- cv2 (OpenCV): Procesează și manipulează imagini și videoclipuri.

- numpy (np): Este o bibliotecă fundamentală pentru calculul științific în Python.

- sklearn.cluster.KMeans: Implementează algoritmul K-Means, care este utilizat pentru clusterizarea datelor. Algoritmul împarte datele în grupuri (clustere) astfel încât punctele din aceeași cluster să fie similare între ele și diferite față de celelalte clustere.

- sklearn.preprocessing.StandardScaler: Scalează și standardizează datele.

- matplotlib.pyplot as plt: Creează vizualizări și grafice pentru analiza datelor.

Programul dezvoltat:

O mare parte din programul final este ocupată de încărcarea imaginilor din foldere, prelucrarea lor, împărțirea în clustere. Programul începe cu o funcție care încarcă imaginile de antrenament dintr-un folder și le atribuie etichete:

```python
# Function to load train images from a folder and assign labels
def load_train_images_from_folder(folder, target_shape=None):
    images = []
    labels = []
    for emotion_folder in os.listdir(folder):
        emotion_folder_path = os.path.join(folder, emotion_folder)
        if os.path.isdir(emotion_folder_path):
            for filename in os.listdir(emotion_folder_path):
                if filename.endswith('.jpg'):  # Assuming images are in JPEG format
                    img = cv2.imread(os.path.join(emotion_folder_path, filename))
                    if img is not None:
                        if target_shape is not None:
                            img = cv2.resize(img, target_shape)
                        images.append(img)
                        labels.append(emotion_folder)  # Assign label based on folder name
                    else:
                        print(f"Warning: Unable to load {filename}")
    return images, labels


2 usages
def load_train_images_from_folder(folder, target_shape=None):
    images = []
    labels = []
    for emotion_folder in os.listdir(folder):
        emotion_folder_path = os.path.join(folder, emotion_folder)
        if os.path.isdir(emotion_folder_path):
            for filename in os.listdir(emotion_folder_path):
                if filename.endswith('.jpg'):  # Assuming images are in JPEG format
                    img = cv2.imread(os.path.join(emotion_folder_path, filename))
                    if img is not None:
                        if target_shape is not None:
                            img = cv2.resize(img, target_shape)
                        images.append(img)
                        labels.append(emotion_folder)  # Assign label based on folder name (e.g., 'happy' or 'sad')
                    else:
                        print(f"Warning: Unable to load {filename}")
    return images, labels

```

Procesul de antrenare începe apoi prin alegerea unui număr predefinit de clustere (k). Apoi, algoritmul încearcă să atribuie fiecare punct de date la unul dintre cele k clustere, astfel încât să minimizeze variația din interiorul fiecărui cluster. Acest lucru se realizează iterativ, prin recalcularea centrului fiecărui cluster și realocarea punctelor de date până când pozițiile centrelor clusterelor nu se mai schimbă semnificativ sau numărul maxim de iterații este atins, K-Means fiind eficient pentru seturi mari de date și fiind utilizat frecvent în probleme de clusterizare.

```python
# Function to plot images grouped by clusters
1 usage
def plot_cluster_images(images, cluster_assignments, titles):
    n_clusters = len(images)
    plt.figure(figsize=(15, 5))
    for cluster in range(n_clusters):
        cluster_images = images[cluster]
        cluster_title = f"Cluster {cluster} ({len(cluster_images)} images)"
        for i, image in enumerate(cluster_images):
            plt.subplot( *args: n_clusters, len(cluster_images), i + 1 + cluster * len(cluster_images))
            plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
            image_idx = np.where(cluster_assignments == cluster)[0][i] # Find the indices for the current cluster
            plt.title(f"{titles[image_idx]}")
            plt.axis('off')
    plt.show()

# Folder paths
data_folder = './emotions_dataset' # Folder with training data
test_folder = './test_faces' # Folder with test data

# Load images and labels from the 'emotions_dataset' folder and resize them to (200, 200)
images, labels = load_train_images_from_folder(data_folder, target_shape=(200, 200))

# Apply K-Means clustering
n_clusters = len(set(labels))  # Automatically detect the number of clusters based on unique labels
kmeans = KMeans(n_clusters=n_clusters, random_state=0)

# Reshape the images and convert them to grayscale
image_data = [cv2.cvtColor(image, cv2.COLOR_BGR2GRAY).flatten() for image in images]

# Convert the list of 1D arrays to a 2D numpy array
image_data = np.array(image_data)

# Scale the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(image_data)

# Train the K-Means model
kmeans.fit(scaled_data)
```

```python

# Train the K-Means model
kmeans.fit(scaled_data)

# Get cluster assignments
cluster_assignments = kmeans.labels_

# Group images by clusters
cluster_images = [[] for _ in range(n_clusters)]
for i, label in enumerate(labels):
    cluster = cluster_assignments[i]
    cluster_images[cluster].append(images[i])

# Assign titles for images in each cluster
titles = labels

# Plot images grouped by clusters
plot_cluster_images(cluster_images, cluster_assignments, titles)
```

În partea de testare și afișare a programului avem o secvență de cod care va încărca imaginile de test din folderul "test_faces", le va prezice emoțiile folosind modelul KMeans și va afișa fiecare imagine de test împreună cu emoțiile prezise și cele reale.

```
### TESTING ###
# Emotion labels corresponding to numerical predictions
emotion_labels = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

# Load test images from the 'test_faces' folder
test_images, test_labels = load_train_images_from_folder(test_folder, target_shape=(200, 200))

# Predict emotions for test images and visualize results
for i, test_image in enumerate(test_images):
    test_image_data = cv2.cvtColor(test_image, cv2.COLOR_BGR2GRAY).flatten()
    test_image_data = np.array(test_image_data).reshape(1, -1)  # Reshape for a single sample
    scaled_test_data = scaler.transform(test_image_data)
    test_prediction = kmeans.predict(scaled_test_data)  # Predict classes

    predicted_emotion = emotion_labels[test_prediction[0]]
    actual_emotion = test_labels[i]

    print(f"Test Image {i + 1} - Predicted Emotion: {predicted_emotion}, Actual Emotion: {actual_emotion}")

    # Visualize the test image with its predicted emotion
    plt.figure()
    plt.imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
    plt.title(f"Test Image {i + 1} - Predicted Emotion: {predicted_emotion}, Actual Emotion: {actual_emotion}")
    plt.axis('off')
    plt.show()

```

Programul dezvoltat:

În contextul algoritmului K-Means, există anumiți parametri critici care influențează procesul de grupare:

- n_clusters: Numărul de clustere sau grupuri în care se împart datele.

- random_state: Parametru utilizat pentru controlul reproducibilității. Acesta asigură că rezultatele sunt consistente în diferite rulări.

- StandardScaler: Parametrii precum mean_ și scale_ sunt folosiți pentru scalarea datelor. Acesta asigură că datele sunt standardizate pentru a fi pe aceeași scară numerică.

Acești parametri sunt esențiali în determinarea modului în care algoritmul K-Means grupează datele și influențează calitatea rezultatelor obținute.

Procesul de antrenare:

După feedback-ul primit am mai schimbat puțin programul prezentat în prima parte a acestui document, urmând să detaliez noile funcționalități mai jos.

1. Încărcarea Datelor de Antrenare:

Funcțiile load_train_images_from_folder și load_test_images_from_folder sunt responsabile pentru încărcarea imaginilor de antrenare și testare, respectiv, din folderele specificate.

2. Preprocesare Imagini:

Imaginile sunt redimensionate la dimensiunea specificată ((200, 200)) în cadrul funcției load_train_images_from_folder și load_test_images_from_folder.

3. Extragerea Caracteristicilor:

Imaginile sunt convertite în imagini alb-negru (cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)) și aplatizate (flatten()), astfel încât să poată fi utilizate pentru antrenarea modelului.

4. Standardizarea Datelor:

Datele sunt standardizate cu ajutorul StandardScaler pentru a le aduce la o scară comună, asigurându-se că toate caracteristicile au o influență egală în procesul de clustering.

5. Antrenarea Modelului K-Means:

Modelul K-Means este creat folosind KMeans din sklearn.cluster. Numărul de clustere (n_clusters) este ales automat, fiind egal cu numărul unic de etichete (emoții) din datele de antrenare.

6. Ajustarea și Predictia:

Modelul este antrenat pe datele de intrare standardizate și obține atributele clusterelor. Apoi, aceste atribute sunt utilizate pentru a face predicții pe setul de date de testare.

Parametrii Utilizați:

- n_clusters (Numărul de Clustere):

Este stabilit automat în acest caz, fiind egal cu numărul unic de etichete (emoții) din datele de antrenare (n_clusters = len(set(labels))).

Această alegere este potrivită pentru problema noastră, deoarece se presupune că fiecare emoție reprezintă un cluster distinct.

- random_state:

Parametrul random_state este setat la 0 pentru a asigura reproducibilitatea rezultatelor. Acesta asigură că, în ciuda alegerilor aleatoare făcute de algoritm, rezultatele rămân constante între diferite rulări.

Comentarii:

Alegerea automată a numărului de clustere (n_clusters) este potrivită în acest caz, deoarece există un set clar de emoții și se dorește ca fiecare emoție să fie asociată cu un cluster.

Standardizarea (StandardScaler) este importantă pentru a asigura că variabilitatea diferitelor caracteristici nu influențează în mod disproporționat rezultatele.

Procesul de aplatizare și conversie la imagine alb-negru este simplist, dar poate funcționa bine pentru anumite sarcini. Cu toate acestea, în cazuri mai complexe, o extragere mai sofisticată a caracteristicilor poate fi necesară.

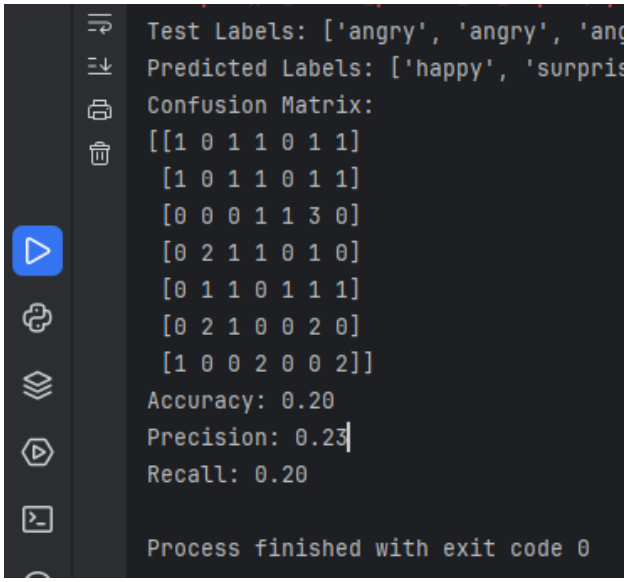
Metricile de evaluare adăugate (accuracy, precision, recall) oferă o înțelegere mai detaliată a performanței modelului pe setul de date de testare.

Secvențele de cod implementate pentru calcularea metricilor menționate mai sus cât și a matricei de confuzie:

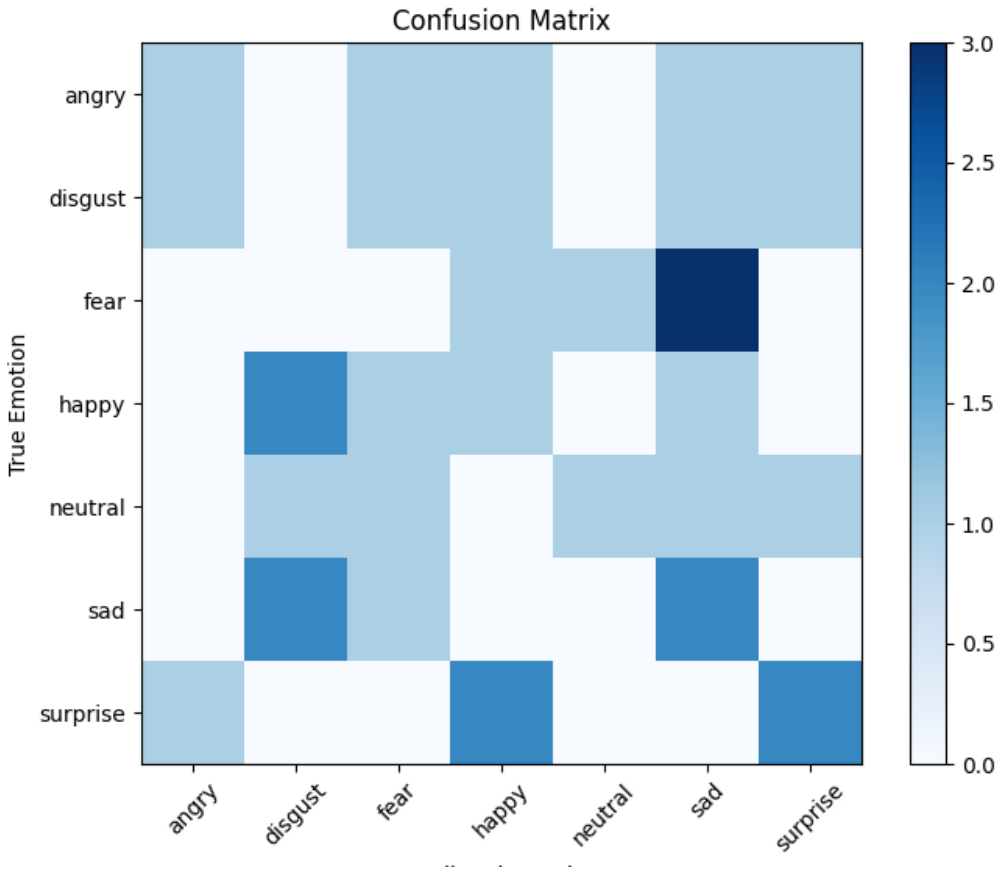
```
# Function to calculate confusion matrix
def calculate_confusion_matrix(test_labels, predicted_labels, labels):
    return confusion_matrix(test_labels, predicted_labels, labels=labels)
```

```python
# Calculate confusion matrix
confusion_mat = confusion_matrix(test_labels, predicted_labels, labels=emotion_labels)
print("Confusion Matrix:")
print(confusion_mat)

# Calculate accuracy, precision, and recall
accuracy = accuracy_score(test_labels, predicted_labels)
precision = precision_score(test_labels, predicted_labels, average='weighted')
recall = recall_score(test_labels, predicted_labels, average='weighted')

# Print evaluation metrics
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(confusion_mat, interpolation='nearest', cmap=plt.get_cmap('Blues'))
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(emotion_labels))
plt.xticks(tick_marks, emotion_labels, rotation=45)
plt.yticks(tick_marks, emotion_labels)
plt.xlabel('Predicted Emotion')
plt.ylabel('True Emotion')
plt.show()
```

Output-ul acestor secvențe de cod pentru problema noastră:



Am afișat și în consolă label-urile pentru a avea toate informațiile în același loc (nu se văd foarte bine în screenshot și am decis să le tai). Avem de asemenea matricea de confuzie în această formă cât și plot-ată mai intuitiv aici:



Cum putem interpreta aceste noi rezultate?

- Matricea de cofuzie:

Rândurile reprezintă etichetele reale, iar coloanele reprezintă etichetele prezise.

De exemplu, în primul rând (corespunzător 'angry'), modelul a prezis corect 1 instanță ca fiind 'angry', a clasificat greșit 1 instanță ca fiind 'fear' și 1 instanță ca fiind 'happy'.

În al doilea rând (corespunzător 'disgust'), modelul a prezis corect 1 instanță ca fiind 'disgust', a clasificat greșit 1 instanță ca fiind 'fear', 1 instanță ca fiind 'happy' și 1 instanță ca fiind 'surprise'.

- Interpretare:

Modelul pare să întâmpine dificultăți în distingerea între anumite emoții, ceea ce duce la clasificări greșite.

Anumite clase (de exemplu, 'angry', 'disgust', 'happy') arată predicții corecte, dar altele ('fear', 'sad', 'surprise') au mai multe clasificări greșite, lucru subliniat și mai jos în partea de expresii asemănătoare.

Precizia generală este relativ scăzută, confirmând evaluarea anterioară.

Pentru a îmbunătăți performanța modelului, am putea explora caracteristici suplimentare, ajusta parametrii de clusterizare sau lua în considerare algoritmi de clusterizare mai avansați. În plus, creșterea diversității și a cantității de date de antrenament ar putea îmbunătăți capacitatea modelului de a generaliza la diferite expresii faciale.

- Performanța:

- Exactitate (Accuracy):

Rata generală de clasificare corectă a modelului. În acest caz, avem o acuratețe de 20%, ceea ce înseamnă că 20% din imaginile au fost clasificate corect.

- Precizie (Precision):

Precizia este raportul dintre numărul de pozitive adevărate și totalul prezis pozitive. Pentru clasificarea emoțiilor, precizia este de 23%, indicând că dintre imaginile prezise ca fiind o anumită emoție, doar 23% sunt corecte.

- Recall (Recall):

Recall reprezintă raportul dintre numărul de pozitive adevărate și totalul real pozitive. În cazul nostru, recall-ul este de 20%, indicând că doar 20% din imaginile reale cu o anumită emoție au fost identificate corect de către model.

În general, aceste rezultate sugerează că modelul ar putea necesita îmbunătățiri pentru a avea o performanță mai bună la clasificarea emoțiilor din imaginile date.

Rezultate

Algoritmul a dat și predicții satisfăcătoare, mai ales când a venit vorba de expresii mai exagerate care s-ar fi încadrat cu ușurință în categoriile alcătuite de noi însă pentru anumite expresii mai subtile cum ar fi o expresie dezamăgită care pare tristă au mai existat și predicții mai puțin satisfăcătoare. Pentru fiecare emoție folosită la testare am folosit cinci imagini pentru a nu încărca prea mult programul și a face imposibilă analizarea datelor (sunt deja 35 de imagini peste care trebuie să ne uităm de fiecare dată). Am atașat mai jos și câteva exemple de predicții obținute după rularea programului.

Exemple de predicții reușite:

Test Image 28 - Predicted Emotion: sad, Actual Emotion: sad

Test Image 24 - Predicted Emotion: neutral, Actual Emotion: neutral

Test Image 11 - Predicted Emotion: fear, Actual Emotion: fear

Test Image 4 - Predicted Emotion: angry, Actual Emotion: angry

Test Image 13 - Predicted Emotion: fear, Actual Emotion: fear

Test Image 9 - Predicted Emotion: disgust, Actual Emotion: disgust

Exemple de predicții nereușite / emoții ambigue:

Test Image 20 - Predicted Emotion: surprise, Actual Emotion: happy

Test Image 15 - Predicted Emotion: sad, Actual Emotion: fear

Test Image 7 - Predicted Emotion: angry, Actual Emotion: disgust

Test Image 14 - Predicted Emotion: disgust, Actual Emotion: fear

Test Image 1 - Predicted Emotion: disgust, Actual Emotion: angry

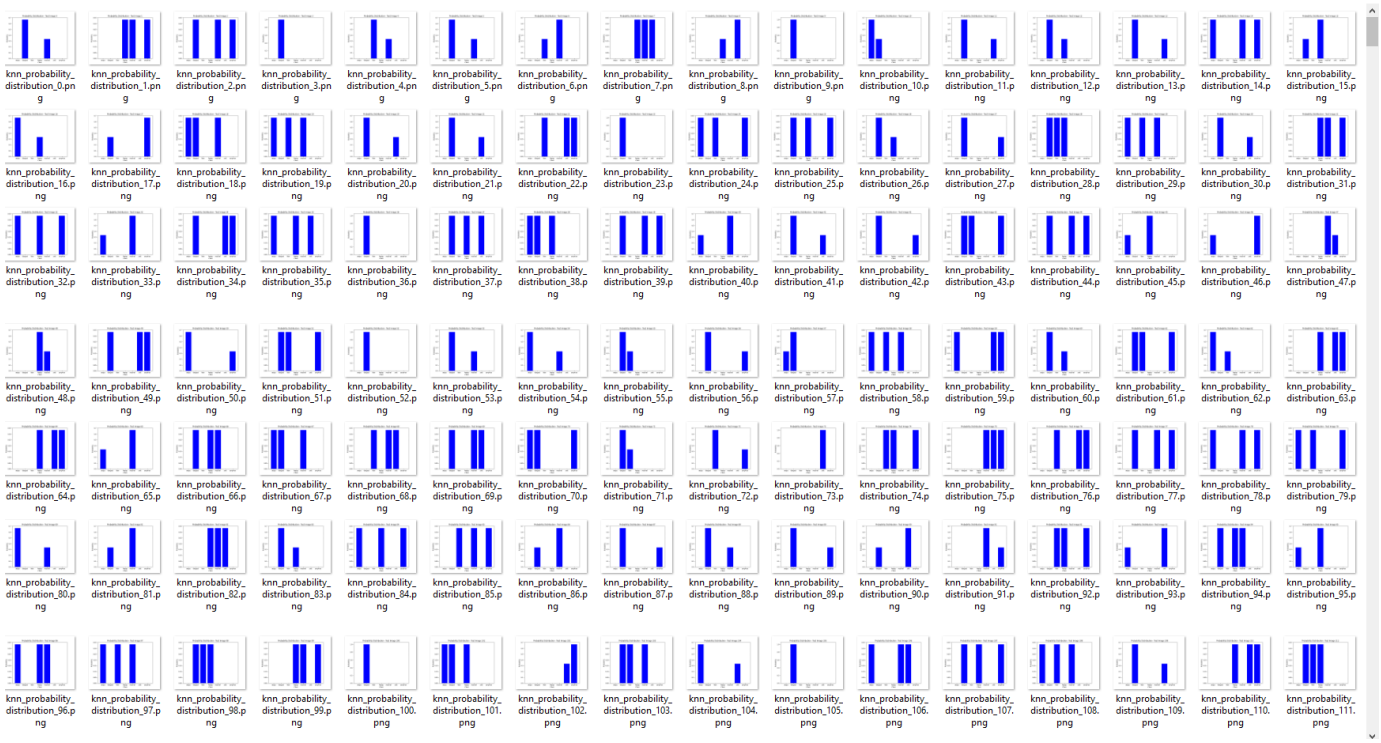
Test Image 6 - Predicted Emotion: sad, Actual Emotion: disgust

După aceste observații am testat modelul având ca date de antrenare doar emoția de fericire și tristețe (am încercat să aleg emoții cât mai diferite) pentru a elimina confuzia dintre emoțiile mai apropiate și am avut rezultate și cu 8 predicții corecte din 10, existând totuși fluctuații și în aceste rezultate.

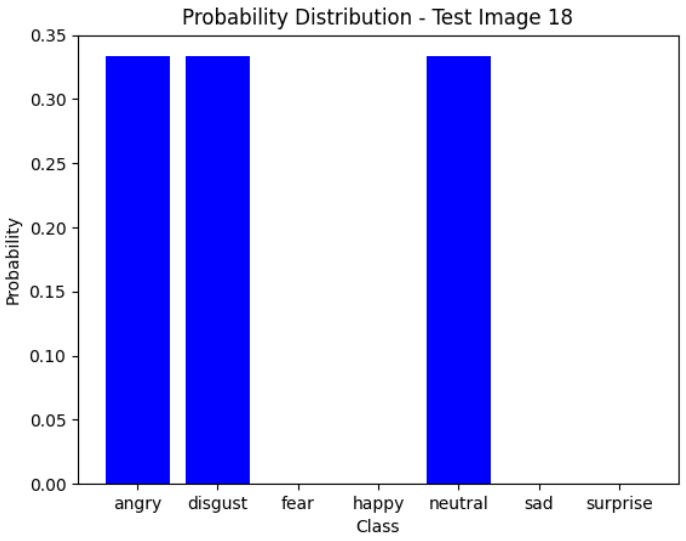
• Comparații cu alți algoritmi

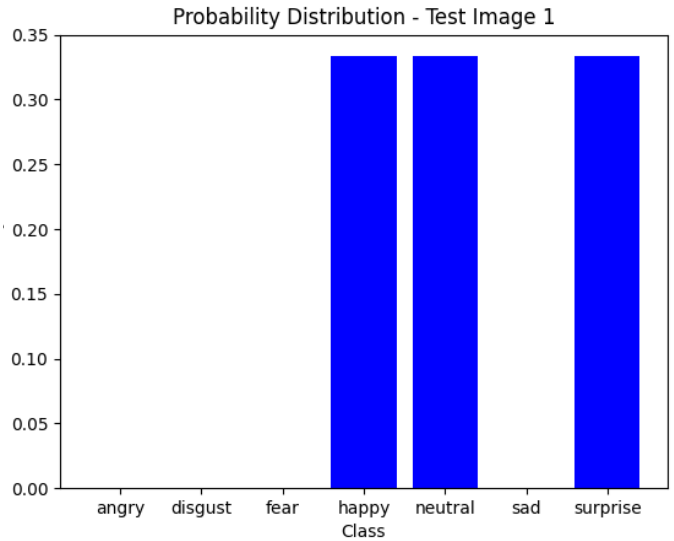
Având experiența temei doi, putem analiza performanța acestui set de date și când este folosit de alți algoritmi si anume K-Nearest Neighbors și Naïve Bayes.

Rezultatele obținute pentru k-NN:



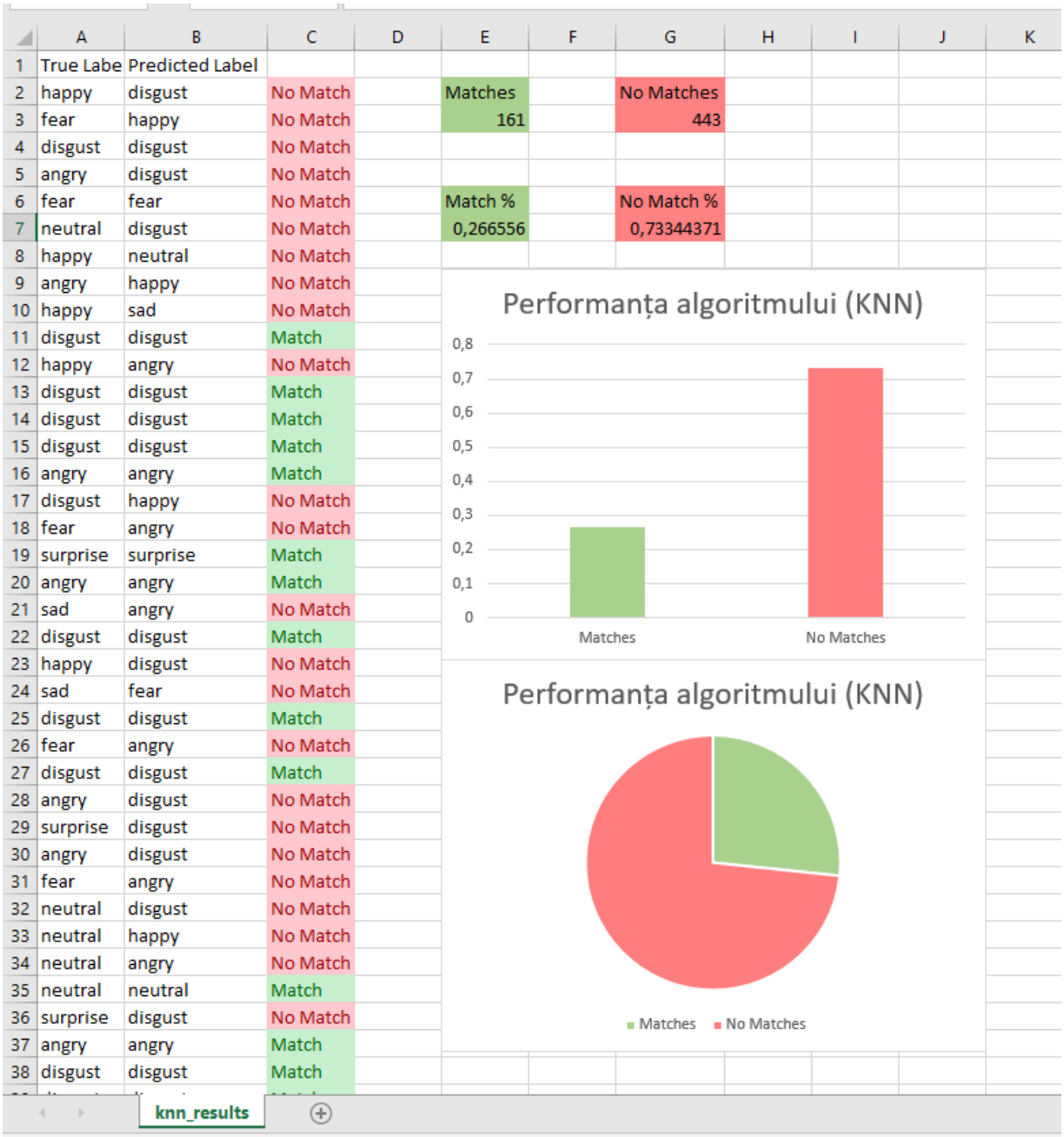
Pentru început observăm că algoritmul k-NN nu a reușit decât în puține cazuri să conveargă pe o singură predicție, existând multe cazuri în care 3 emoții aveau probabilități egale. Multe dintre aceste grupuri de emoții împart caracteristici similare când vine vorba de trăsături, cum ar fi furia, dezgustul și frica sau fericirea și surprinderea, problemă semnalată și în prima temă.





Performanța algoritmului k-NN:

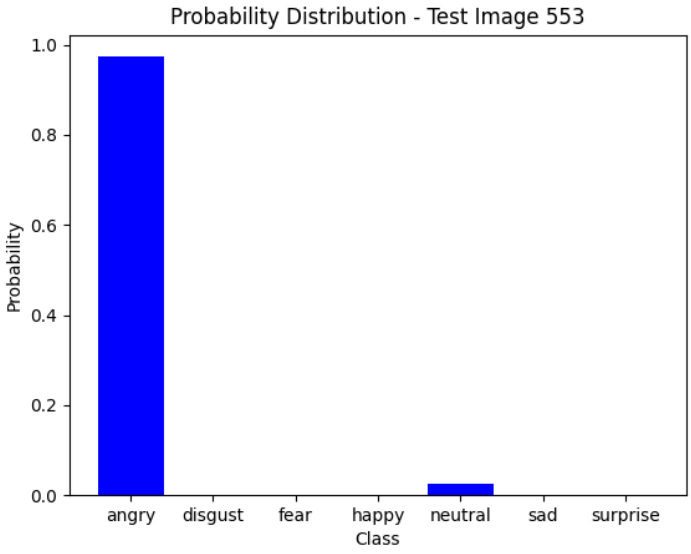
Observăm că performanța nu este una foarte mulțumitoare ceea ce era totuși de așteptat ținând cont de setul nostru de date și de algoritmul folosit. O alegere mai bună pentru clasificarea emoțiilor ar fi fost poate algoritmul HOG cu HAAR Cascade Classifier pe care am și încercat să îl implementez. Să vedem totuși înainte cum se compară acest rezultat cu rezultatul obținut pentru Naive Byes.

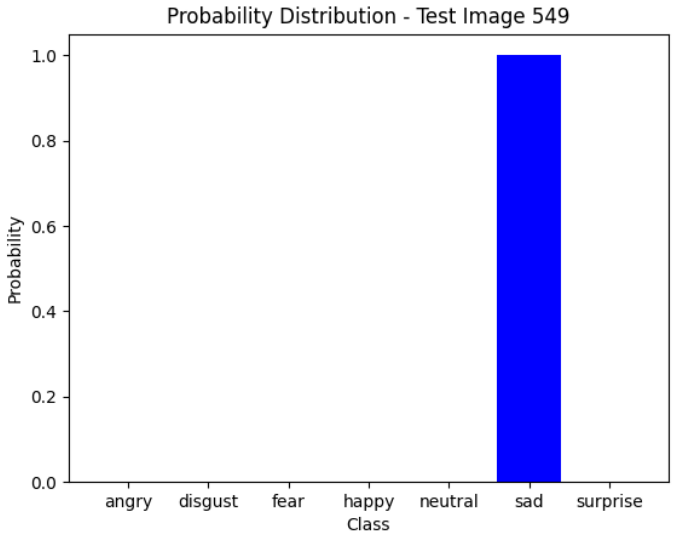


	A	B	C	D	E	F	G	H	I	J	K
1	True Labe	Predicted Label									
2	happy	disgust	No Match		Matches		No Matches				
3	fear	happy	No Match		161		443				
4	disgust	disgust	No Match								
5	angry	disgust	No Match								
6	fear	fear	No Match		Match %		No Match %				
7	neutral	disgust	No Match		0,266556		0,73344371				
8	happy	neutral	No Match								
9	angry	happy	No Match								
10	happy	sad	No Match								
11	disgust	disgust	Match								
12	happy	angry	No Match								
13	disgust	disgust	Match								
14	disgust	disgust	Match								
15	disgust	disgust	Match								
16	angry	angry	Match								
17	disgust	happy	No Match								
18	fear	angry	No Match								
19	surprise	surprise	Match								
20	angry	angry	Match								
21	sad	angry	No Match								
22	disgust	disgust	Match								
23	happy	disgust	No Match								
24	sad	fear	No Match								
25	disgust	disgust	Match								
26	fear	angry	No Match								
27	disgust	disgust	Match								
28	angry	disgust	No Match								
29	surprise	disgust	No Match								
30	angry	disgust	No Match								
31	fear	angry	No Match								
32	neutral	disgust	No Match								
33	neutral	happy	No Match								
34	neutral	angry	No Match								
35	neutral	neutral	Match								
36	surprise	disgust	No Match								
37	angry	angry	Match								
38	disgust	disgust	Match								

Rezultatele obținute pentru Naive Bayes:

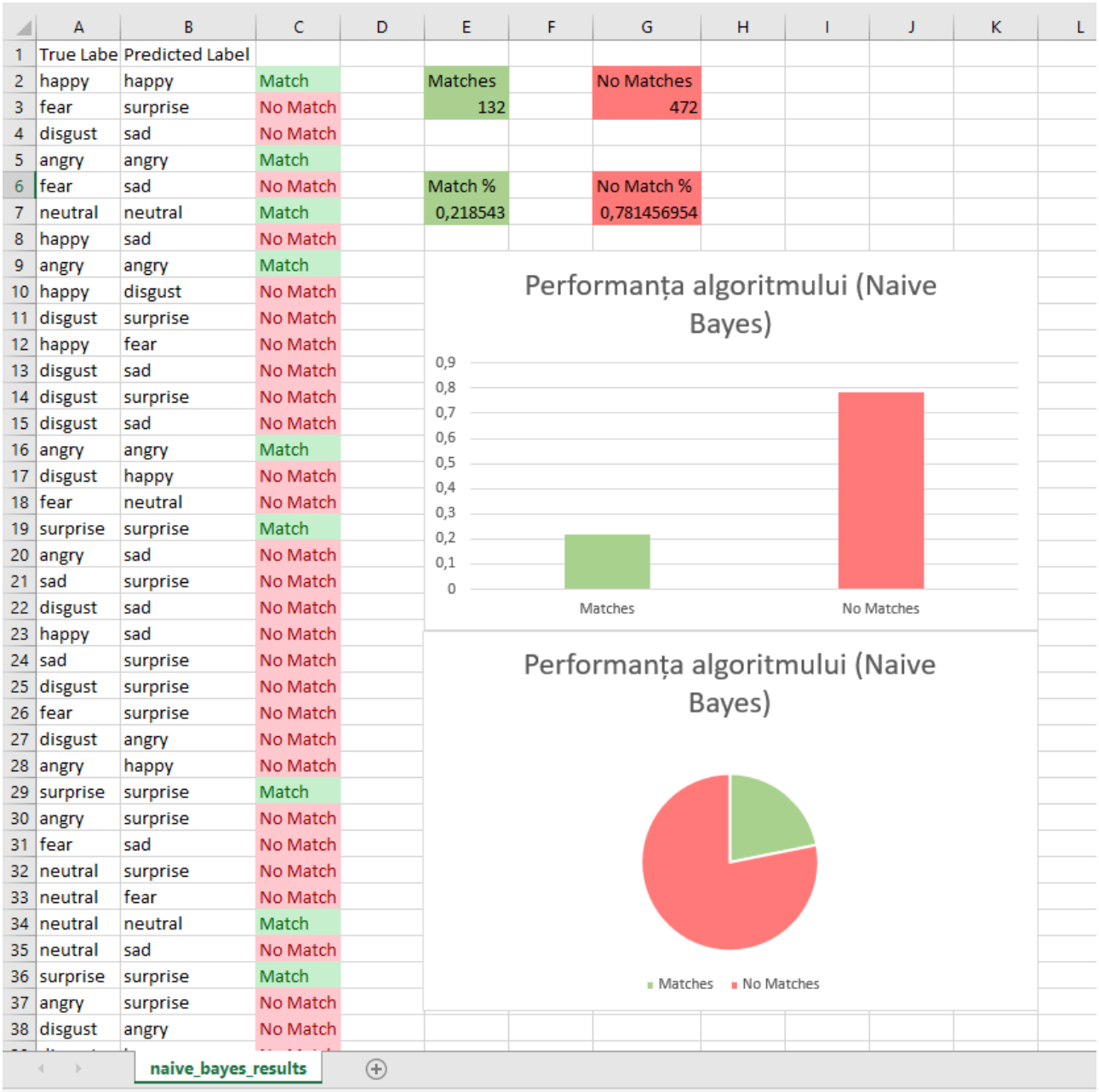
Observăm că spre deosebire de k-NN acest algoritm a dat rezultate mult mai „definitive”,cazurile în care o imagine să aibă probabilitatea de a conține două sau mai multe emoții fiind mult mai rare.





Performanța algoritmului Naive Bayes:

Cu toatea acestea procentajul predicțiilor reușite a fost foarte similar cu cel al algoritmului precedent.



	A	B	C	D	E	F	G	H	I	J	K	L
1	True Labe	Predicted Label										
2	happy	happy	Match		Matches		No Matches					
3	fear	surprise	No Match		132		472					
4	disgust	sad	No Match									
5	angry	angry	Match									
6	fear	sad	No Match		Match %		No Match %					
7	neutral	neutral	Match		0,218543		0,781456954					
8	happy	sad	No Match									
9	angry	angry	Match									
10	happy	disgust	No Match									
11	disgust	surprise	No Match									
12	happy	fear	No Match									
13	disgust	sad	No Match									
14	disgust	surprise	No Match									
15	disgust	sad	No Match									
16	angry	angry	Match									
17	disgust	happy	No Match									
18	fear	neutral	No Match									
19	surprise	surprise	Match									
20	angry	sad	No Match									
21	sad	surprise	No Match									
22	disgust	sad	No Match									
23	happy	sad	No Match									
24	sad	surprise	No Match									
25	disgust	surprise	No Match									
26	fear	surprise	No Match									
27	disgust	angry	No Match									
28	angry	happy	No Match									
29	surprise	surprise	Match									
30	angry	surprise	No Match									
31	fear	sad	No Match									
32	neutral	surprise	No Match									
33	neutral	fear	No Match									
34	neutral	neutral	Match									
35	neutral	sad	No Match									
36	surprise	surprise	Match									
37	angry	surprise	No Match									
38	disgust	angry	No Match									

Putem observa că procentul de predicții corecte e foarte asemănător cu cel obținut și la metodele explorate anterior. Având în minte setul de date extins, cu emoții asemănătoare și expresii uneori ambigue e posibil ca această performanță scăzută să fie datorată și setului de date ales, pe lângă nepotrivirea algoritmilor pentru task-ul dat.

• Concluzii și observații

În concluzie performanța obținută de modelul nostru a fost satisfăcătoare însă nu se poate spune că e un model extrem de consecvent. Am observat totuși că reducerea setului la doar două emoții din șapte care au și expresii destul de diferite aduce o creștere a performanței.

Un lucru pe care am putea să îl îmbunătățim la setul nostru de date ar fi să folosim expresii mai categorice și bine definite pentru o antrenare mai bună a modelului.

• Bibliografie

https://www.kaggle.com/datasets/msambare/fer2013

https://www.edlitera.com/blog/posts/emotion-detection-in-images

https://medium.com/analytics-vidhya/facial-expression-detection-using-machine-learning-in-python-c6a188ac765f

https://www.kaggle.com/code/shakil19/emotion-detection-cnn

https://thinkinfi.com/real-time-emotion-recognition-using-facial-expressions-with-deep-learning-cnn/

https://www.hackersrealm.net/post/facial-emotion-recognition-using-python

https://towardsdatascience.com/train-image-recognition-ai-with-5-lines-of-code-8ed0bdd8d9ba

https://analyticsindiamag.com/top-8-datasets-available-for-emotion-detection/