AoM: Classificatore per la detection di attacchi di morphing

Alessandro Aquino, *Alberto* Montefusco e *Simone* Tartaglia

Abstract. Lo scopo del progetto è stato quello di sviluppare con successo soluzioni per la rilevazione degli attacchi di morphing (MAD) basate su dati sintetici. A tal fine, questo lavoro è stato sviluppato usando un dataset per la rilevazione degli attacchi di morphing basato su dati sintetici, chiamato Synthetic Morphing Attack Detection Development dataset (SMDD). Questo dataset viene utilizzato per addestrare cinque modelli di base per la rilevazione degli attacchi di morphing. Inoltre, un aspetto fondamentale di questo lavoro è l'analisi dettagliata delle prestazioni ottenute dai vari modelli addestrati.

1 Introduzione

1.1 Problema

I sistemi di riconoscimento facciale (FR), nonostante la loro elevata precisione e accettabilità, sono vulnerabili a molteplici attacchi, uno dei quali è l'attacco di morphing del volto (MA). Gli attacchi di morphing analizzati dimostrano che un'immagine del volto può essere associata, sia automaticamente che da esperti umani, a più di una persona. Se tali attacchi prendessero di mira documenti di viaggio o di identità, ciò consentirebbe a più soggetti di essere verificati con un unico documento. Questo collegamento errato, tra un documento e un'identità, può portare ad attività illegali, tra cui tratta di esseri umani, immigrazione illegale e transazioni finanziarie discutibili.

Tali attacchi possono essere in parte riscontrati utilizzando soluzioni di rilevamento di attacchi al volto (MAD - Morphing Attack Detection). Solitamente, le soluzioni MAD richiedono un addestramento per differenziare due classi: morphing attack e immagini del volto autentiche (BF, bonafide). Questa formazione richiede dati contenenti campioni di queste classi da utilizzare ed eventualmente condividere per la ricerca e lo sviluppo di MAD da parte di diverse persone e istituzioni.

1.2 Workflow: Attack on Morphing (AoM)

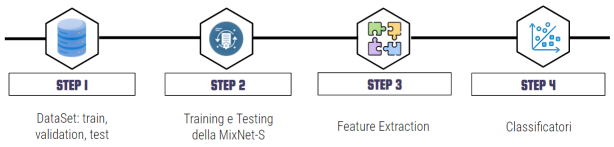

Figure 1. Workflow AoM

Data la problematica degli attacchi di morphing, abbiamo deciso di riproporre il progetto del paper. Sviluppare un classificatore capace di riconoscere diverse tipologie dei suddetti attacchi, basati su cinque tecniche differenti: OpenCV (OCV), FaceMorpher (FM), StyleGAN 2 (SG), WebMorpher (WM), e AMSL. Il Modello (MixNets) è stato, invece, addestrato su SMDD e validato su una porzione presa in maniera randomica del test. Il dataset è stato suddiviso come mostrato nelle tabella sottostante. La scelta della MixNet nella sua versione "s" è dovuta ad un carico computazionale inferiore rispetto alla versione "m" ed "l". L'Addestramento della MixNets è stato riprodotto cinque volte, considerando le validation generate dai test (OpenCV, FaceMorpher, StyleGAN, WebMorpher, e AMSL). Ogni modello validato è stato poi testato su: OpenCV, FaceMorpher, StyleGAN, WebMorpher, e AMSL. I risultati ottenuti si sono discostati da quelli riportati dal paper, soprattutto l'APCER in funzione di diverse tolleranze del BPCER. Pertanto, abbiamo deciso di considerare, nel confronto con il paper, unicamente l'ERR, il quale si discosta di poco da quello previsto. Supponi-

Train	
Bonafide	Attack
25000	15000

Validation				
Attack				
facemorpher	stylegan	amsl	opencv	webmorph
45	45	75	45	45

Test						
Attack					Bonafide	
facemorpher	stylegan	amsl	opencv	webmorph	smile	no smile
1222	1222	2175	1229	1221	205	103

Figure 2. Suddivisione Dataset

amo che la differenza ottenuta è data dalla validation, non presentando le stesse immagini (non abbiamo una cartella contenente le immagini di validation ma avendole estratte randomicamente dal test) abbia portato delle metriche leggermente diverse. Una volta replicata la sperimentazione proposta dal paper[1], abbiamo effettuato ulteriori veri-

fiche. Nello specifico abbiamo confrontato la MixNets con l'approccio geometrico, basato sulla geometria dei volti. Questo confronto nasce con l'obiettivo di verificare quale dei due approcci si comporta meglio. Inoltre, il nostro obiettivo è stato anche quello di verificare se l'unione dei due approcci avrebbe prodotto delle metriche migliori. L'interesse risiede anche nel verificare se la MixNets è capace di migliorare le criticità dell'approccio geometrico nel riconoscere i morph sorridenti, poiché soffre sui dettagli più complessi, come il sorriso.

Il primo passo per permettere tale confronto è stato di estrarre le caratteristiche dal penultimo layer della rete (ossia prima della classificazione) ed utilizzare tali caratteriste con gli stessi classificatori (Decision tree, Random Forest, GuassianDB) dell'approccio geometrico. Estratte le features, queste passano per la fase di pre-prossessing utilizzando la PCA. La PCA() accetta diversi parametri, tra cui n_components (è impostato su "mle", il quale stima il numero di componenti in base alle proprietà statistiche dei dati. Questo approccio determina automaticamente il numero ottimale di componenti). L'output prodotto viene dato in input ai classificatori sia per il train che per il test ottenendo i primi risultati. Questo processo è stato ripetuto sia per i test contenenti i sorrisi che no. Successivamente, si è passati al merge con i file .csv contenenti le feautures dei due approcci. I merge risultanti sono due: "con smile" e "senza".

2 Implementazione AoM

In questa sezione verrà descritto il workflow di AoM, il quale riproporrà la sperimentazione del paper "Privacy-friendly Synthetic Data for the Development of Face Morphing Attack Detectors"[1]. Verrà utilizzato il dataset proposto dal paper con la stessa suddivisione del train, validation e test. Riprodotta la sperimentazione ed ottenuti i risultati simili, siamo passati alla features extraction.

Questa fase è utile per permettere una classificazione mediante ulteriori algoritmi di Machine Learning (Decison Tree, Random Forest e GaussianNB). Prima di dare in input le caratteristiche ai classificatori abbiamo effettuato una fase di pre-prossessing sulle features (PCA). Ottenute le prime metriche, le abbiamo confrontate con quelle dell'approccio geometrico. Oltre a verificare quale delle due soluzioni possegga metriche migliori, abbiamo effettuato il merge delle features, verificando se uniti potessero migliorare . Il merge è stato fatto considerando due casi specifici: "smile" e "no smile". Di seguito è riportata la sperimentazione completa.

2.1 Datasets: train, validation e test

Per il corretto training e successivo testing di AoM, è stato necessario effettuare un'adeguata suddivisione del dataset in tre diverse parti: un training set, un validation set, e un test set.

Per l'addestramento del modello, è stato utilizzato il dataset Synthetic Morphing Attack Detection Development dataset (SMDD). Si tratta di uno dei primi dataset pensati per la rilevazione di attacchi biometrici ad essere reso pubblicamente disponibile per fini di ricerca. Le immagini contenute in esso sono generate casualmente basandosi su una Gaussian noise Z, estraendole senza ripetizioni 500.000 volte da una distribuzione normale. Da ogni vettore, il generatore produce poi un campione di dati sintetici di volti, i quali sono stati poi suddivisi in un campione da 40.000 immagini destinate al training. Per la validation, invece, il numero di immagini è suddiviso nel modo seguente:

- 75 immagini per amsl;
- 45 immagini per facemorpher;
- 45 immagini per openCV;
- 45 immagini per stylegan;
- 45 immagini per webmorph.

Ogni coppia di immagini morphed è stata generata sfruttando l'algoritmo OpenCV/dlib [2], il quale ha dimostrato di essere uno dei generatori di MA migliori tra quelli attualmente disponibili [3]. Riguardo al testing e al validation, è stato utilizzato il dataset FRLL_test [3]. Questo dataset è stato generato partendo dal dataset pubblicamente disponibile Face Research London Lab. Esso include due sottocartelle: FRLL_bonafide contiene le immagini bonafide (205 immagini con "smile" e 103 immagini senza "smile"), mentre, FRLL_Morphs ha al suo interno le immagini generate dai vari tools di morphing (7069 immagini). Analizzando quest'ultimo, possiamo notare che al suo interno sono presenti le sottocartelle:

1. morph_amsl con 2175 immagini.
2. morph_facemorpher con 1222 immagini;
3. morph_opencv con 1229 immagini;
4. morph_stylegan con 1222 immagini;
5. morph_webmorph con 1221 immagini.

2.2 Modello MixNet

Per utilizzare SMDD ai fini del training del nostro MAD, abbiamo impiegato come backbone **MixNet-S** [4]. MixNet è, difatti, una famiglia di modelli basati su reti convoluzionali basata, a sua volta, sul modello MixConv. L'idea alla base di tale modello è quella di unire molteplici kernel, ognuno di dimensioni differenti, in un'unica operazione convoluzionale, così da ottenere facilmente diversi tipi di pattern dalle immagini ricevute in input. MixNet sfrutterà molteplici iterazioni del modello MixConv, ognuna di esse caratterizzata da una group size incrementale rispetto alla precedente. Esso è disponibile in tre versioni, ognuna caratterizzata da un'accuracy maggiore delle precedente, ma al contempo più esose dal punto di vista computazionale: MixNet-S, MixNet-M e MixNet-L. A causa di esigenze tecniche, è stato necessario limitarsi all'utilizzo di MixNet-S.

```
class MixNet(nn.Module):
    def forward(self, x):
        x = self.features(x)
        x=self.tail(x)
        x=self.feautre_layer(x)

        # adding the following classification layer for morph attack detection
        x = self.pool1(x)
        x = x.view(x.size(0), -1)
        x=self.features_norm(x)
        x = self.linear2(x.view(x.shape[0], -1))

        return x
```

2.3 Features extraction

Per quanto riguarda il processo di features extraction, dapprima andiamo a caricare il modello sfruttando le funzionalità messe a disposizione dalla libreria torch. Dopodiché, andiamo a rimuovere l'ultimo layer del classificatore, così da poter separare le caratteristiche. Quindi, definiamo la trasformazione da applicare ai dati ricevuti in input, ridimensionando l'immagine ricevuta ad una risoluzione di 224*224 pixel, ed infine normalizzando il tutto. Ad ogni singola immagine verrà quindi applicata la trasformazione e, una ad una, verrà inviata in input al modello così da poter effettivamente estrarne le caratteristiche.

```
def feature_extraction(model, model_path, path_dataset_csv):
    # Load the pre-trained weights
    model.load_state_dict(torch.load(model_path))

    # Set the model to evaluation mode
    model.eval()

    # Remove the final classification layer
    model = nn.Sequential(*list(model.children())[:-1])

    # Define the transformation to be applied to your input data
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])

    path_feature_out_csv = 'output/feature_extraction/' + path_dataset_csv.split('/')[2]
    header = ['image_path', 'label']

    with open(path_feature_out_csv, 'w', newline='') as features_csv:
        writer = csv.writer(features_csv)
        writer.writerow(header)

    with open(path_dataset_csv, 'r') as file:
        reader = csv.reader(file)
        rows_origin = list(reader)

        for row_origin in rows_origin[1:]:
            path_image = row_origin[0]
            input_data = Image.open(path_image)

            # Apply the transformation to your input data
            input_data = transform(input_data)

            # Add a batch dimension to the input data
            input_data = input_data.unsqueeze(0)

            # Pass the input data through your model to extract features
            with torch.no_grad():
                features = model(input_data)

            data_row = [path_image, row_origin[1]]
            for feature in features:
                prova = feature.numpy()

                for elem in prova.flatten():
                    data_row.append(elem)

            with open(path_feature_out_csv, 'a', newline='') as features_csv:
                writer = csv.writer(features_csv)
                writer.writerow(data_row)
```

2.4 Pre-prossessing: PCA

La PCA (Principal Component Analysis) è una tecnica di riduzione della dimensionalità utilizzata per estrarre le componenti principali da un insieme di dati. Nel contesto della libreria scikit-learn in Python, il costruttore `PCA()` accetta diversi parametri, tra cui `n_components` (trova il numero di componenti che offre la migliore spiegazione dei dati senza sovradattamento, quindi, l'algoritmo cerca di trovare il numero ottimale di componenti che rappresentano la varianza dei dati nel modo più accurato possibile senza aggiungere componenti ridondanti o inutili.) e `copy` (se creare una copia dei dati di input o modificarli direttamente). Se `copy=True` (valore predefinito), viene creata una copia dei dati per evitare modifiche indesiderate, altrimenti i dati originali vengono modificati direttamente.

Nel nostro caso:

- **n_components**: è impostato su "mle", il quale stima il numero di componenti in base alle proprietà statistiche dei dati. Questo approccio determina automaticamente il numero ottimale di componenti, che può essere utile quando non si è sicuri del numero di componenti da scegliere.
- **copy**: nel nostro caso è True.

```
from sklearn.decomposition import PCA
    pca = PCA(n_components='mle', copy=True)
    pca_values = pca.fit_transform(x)
```

2.5 Modelli di machine learning

La fase di test viene effettuata su ogni tipo di attacco di morph. I tre classificatori vengono forniti sempre dalla libreria scikit-learn e prendono in input l'output della PCA e la mappa dei valori (label).

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
```

```
# select the GaussianNB algorithm
model = GaussianNB()
model.fit(pca_values, y)

# select the RabdomForest algorithm
model = RabdomForest()
model.fit(pca_values, y)

# select the DecisionTree algorithm
model = DecisionTreeClassifier()
model.fit(pca_values, y)
```

2.6 Visualizzazione delle metriche

Per quanto concerne la visualizzazione delle metriche, le prestazioni di AoM sono valutate in base all'Attack Presentation Classification Error Rate (APCER), ovvero la proporzione di immagini morphed classificate come bonafide, e il Bonafide Presentation Classification Error Rate (BPCER) che, inversamente, indica le immagini bonafide classificate come morphed. Per avere una visione dell'insieme quanto più completa, andiamo ad evidenziare l'APCER quando la BPCER raggiunge tre differenti valori (0.1%, 1.0%, 10% e 20%). Viene riportato anche l'Equal Error Rate (EER), ovvero la soglia in cui BPCER e APCER sono equivalenti.

La funzione utilizzata è la `get_bpcer_op`: calcola la differenza tra i valori di BPCER dati e lo specifico operating point ("`op`"):

```
 temp = abs(bpcer - op)
```

Trova il valore più piccolo (e il corrispondente indice) che contiene l'array '`temp`', che rappresenta il valore BPCER più vicino all'operating point:

```
min_val = np.min(temp)
index = np.where(temp == min_val)[0][-1]
```

Infine, dalla funzione siamo interessati al valore APCER corrispondente a quell'indice ('`apcer[index]`').

```
return index, apcer[index], threshold[index]
```

Per quanto riguarda la stampa dell'Equal Error Rate (EER) la funziond di interesse è la `get_eer_threshold`, la quale calcola la differenza tra TPR (True Positive Rate) e FPR (False Positive Rate) sottraendo 1.0. Questo calcolo è finalizzato a trovare il punto in cui la differenza tra TPR e FPR è più vicina a zero, poiché l'EER è il punto in cui TPR e FPR si incontrano. In seguito, trova l'indice corrispondente al valore minimo della differenza tra TPR e FPR ottenendo il valore FPR corrispondente, che rappresenterà l'EER.

```
differ_tpr_fpr_1=tpr+fpr-1.0
index = np.nanargmin(np.abs(differ_tpr_fpr_1))
eer = fpr[index]

return eer, index, threshold[index]
```

Per calcolare l'accuratezza, abbiamo usato la funzione `accuracy_score` di scikit-learn, la quale calcola l'accuratezza del modello di classificazione misurando la percentuale di previsioni corrette rispetto alle etichette corrette dei dati di test.

3 Analisi dei risultati

In questa sezione andremo a mostrare e a discutere dei risultati ottenuti dai diversi modelli stabilendo quale caso ha prodotto metriche migliori. Non verranno trattati tutti i risultati ottenuti ma solo quelli utili al confronto. Per una visione completa andare al GitHub del progetto.

3.1 Validation AoM

I risultati che verranno presentati in questa sezione andranno a mostrare come varia l'EER per ogni validation. Quindi rappresenteremo come nei cinque test l'EER varia in base al validation utilizzato.

Smile

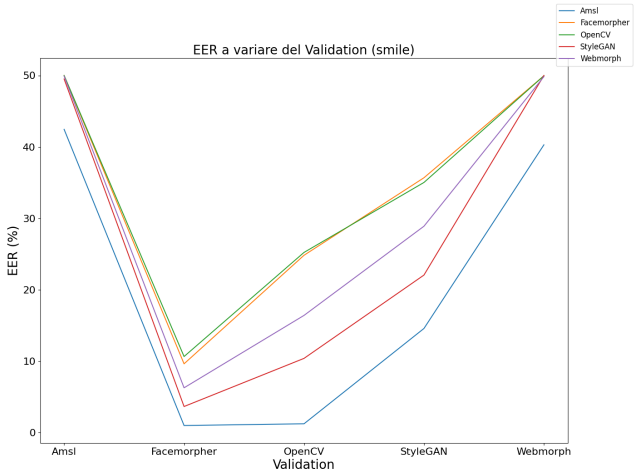

Figure 3. Validation AoM (smile)

No Smile

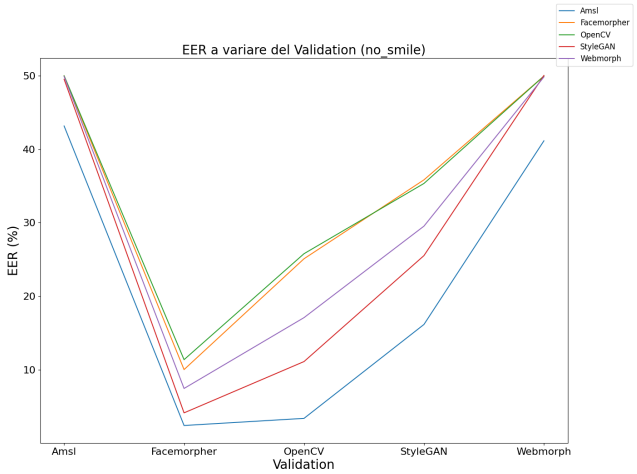

Figure 4. Validation AoM (no smile)

Da questi due grafici si evince che i validations che hanno portato un EER minore sono stati: facemorpher e OpenCV.

3.2 Confronto AoM - paper

3.2.1 EER

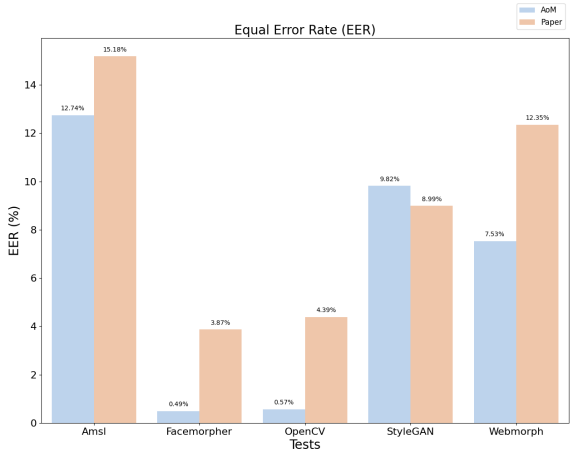

Figure 5. Confronto ERR

Da questo grafico si evince che in media l'EER del paper è più alto rispetto a quello di AoM.

3.3 Confronto AoM - Approccio Geometrico

I seguenti risultati verranno rappresentati come comparazione tra Approccio Geometrico - AoM - Approccio Geometrico & AoM per l'accuracy. Nel confronto con l'EER consideriamo anche il confronto con il paper. In modo tale da rendere il più intuitivo possibile quale modello è stato condizionato in meglio o in peggio.

3.3.1 Smile

In questa sezione si andranno a mostrare i confronti considerando anche le immagini con persone sorridenti, ossia dove l'approccio geometrico soffre maggiormente.

Accuracy. In questo grafico vengono riportate le accuracy per ogni test che si è comportato meglio con un dato train (il train migliore è stato facemorpher oppure amsl, mentre StylGAN di poco rispetto agli altri). Consapevoli del fatto che il dato delle accuracy non è indice di qualità nel nostro caso, avendo un dataset sbilancianto. La scelta è stata ponderata anche sul f1-score della macro avg che considera la precision e la recall che sono metriche più adatte alla valutazione dei nostri dati. Si può notare che il merge dei due approcci è migliore rispetto agli approcci presi singolarmente.

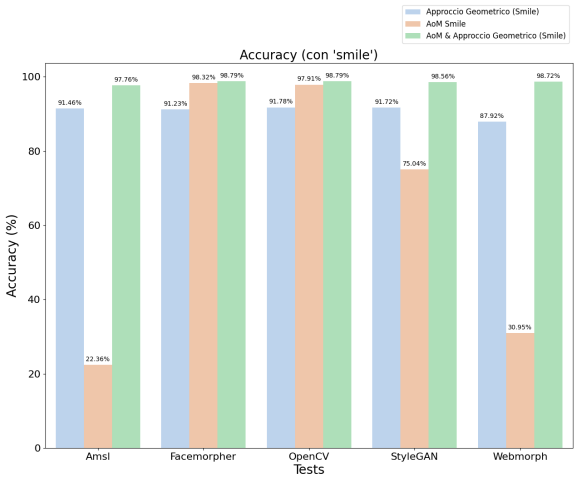

Figure 6. Confronto accuracy AoM - AG

Accuracy generali AoM. Di seguito vengono mostrate le accuracy di AoM per ogni classificatore evidenziando quali test hanno portato le metriche migliori.

	Smile		
Test	DecisionTreeClassifier()	RandomForestClassifier()	GaussianNB()
AMSL	16,90%	11,43%	22,36%
FaceMorpher	93,27%	96,84%	98,32%
OpenCV	90,37%	93,02%	97,91%
StyleGAN	65,64%	62,27%	75,04%
WebMorph	22,39%	16,21%	30,95%
MEDIA	57,71%	55,95%	64,92%

Figure 7. Accuracy AoM smile

	Not Smile		
Test	DecisionTreeClassifier()	RandomForestClassifier()	GaussianNB()
AMSL	13,90%	7,60%	17,52%
FaceMorpher	92,22%	95,39%	95,54%
OpenCV	87,30%	90,53%	93,76%
StyleGAN	63,44%	60,05%	70,17%
WebMorph	16,48%	9,83%	24,04%
MEDIA	54,67%	63,95%	60,21%

Figure 8. Accuracy AoM no smile

Matrice di confusione. Di seguito vengono mostrate le matrici di confusione di AoM per ogni classificatore evidenziando quali test hanno portato le metriche migliori.

Smile						
Confusion Matrix	DecisionTreeClassifier()		RandomForestClassifier()		GaussianNB()	
AMSL	195	9	204	0	204	0
	1968	207	2107	68	1847	328
FaceMorpher	195	9	204	0	204	0
	77	1145	45	1177	24	1198
OpenCV	195	9	204	0	204	0
	129	1100	100	1129	30	1199
StyleGAN	195	9	198	6	204	0
	481	741	532	690	356	866
WebMorph	195	9	204	0	204	0
	1097	124	1194	27	984	237

Figure 9. Confusion Matrix AoM smile

Not Smile						
Confusion Matrix	DecisionTreeClassifier()		RandomForestClassifier()		GaussianNB()	
AMSL	96	6	100	2	102	0
	1973	202	2102	73	1878	297
FaceMorpher	96	6	100	2	102	0
	97	1125	59	1163	59	1163
OpenCV	96	6	100	2	102	0
	163	1066	124	1105	83	1146
StyleGAN	96	6	100	2	102	0
	478	744	527	695	395	827
WebMorph	96	6	102	0	102	0
	1099	122	1193	28	1005	216

Figure 10. Confusion Matrix AoM no smile

Accuracy generali Merge. Di seguito vengono mostrate le accuracy del modello che possiede le features di AoM e dell'Approccio Geometrico, per ogni classificatore, evidenziando quali test hanno portato le metriche migliori.

	Smile		
Test	DecisionTreeClassifier()	RandomForestClassifier()	GaussianNB()
AMSL	19,46%	16,56%	97,76%
FaceMorpher	96,45%	97,89%	98,79%
OpenCV	92,77%	95,87%	98,79%
StyleGAN	70,02%	71,53%	98,56%
WebMorph	23,89%	20,26%	98,72%
MEDIA	60,52%	60,42%	98,52%

Figure 11. Accuracy merge smile

	Not Smile		
Test	DecisionTreeClassifier()	RandomForestClassifier()	GaussianNB()
AMSL	20,49%	17,76%	95,76%
FaceMorpher	97,44%	99,29%	98,72%
OpenCV	92,23%	96,38%	97,51%
StyleGAN	67,67%	69,34%	96,15%
WebMorph	19,12%	13,76%	95,46%
MEDIA	59,39%	59,31%	96,72%

Figure 12. Accuracy merge no smile

Matrice di confusione. Di seguito vengono mostrate le matrici di confusione degli approcci uniti (AoM e Approccio Geometrico), per ogni classificatore, evidenziando quali test hanno portato le metriche migliori.

Smile						
Confusion Matrix	DecisionTreeClassifier()		RandomForestClassifier()		GaussianNB()	
AMSL	199	5	190	14	188	16
	1829	244	1886	187	35	2038
FaceMorpher	199	5	190	14	188	16
	42	1078	29	1091	0	1120
OpenCV	199	5	190	14	188	16
	92	1027	1042	1077	0	1119
StyleGAN	186	18	190	14	188	16
	379	741	363	757	3	1117
WebMorph	186	18	190	14	188	16
	989	130	1041	78	1	1118

Figure 13. Confusion Matrix merge smile

Not Smile						
Confusion Matrix	DecisionTreeClassifier()		RandomForestClassifier()		GaussianNB()	
AMSL	94	8	96	6	99	3
	892	138	925	105	45	985
FaceMorpher	98	4	102	0	99	3
	25	1005	8	1022	14	1200
OpenCV	98	4	102	0	99	3
	84	946	41	989	30	1191
StyleGAN	94	8	96	6	99	3
	420	802	400	822	48	1174
WebMorph	94	8	96	6	99	3
	1062	159	1135	86	57	1164

Figure 14. Confusion Matrix merge no smile

EER.
In questo grafico mostriamo sull'asse delle x l'EER dei modelli validati e testati sulle stesse immagini di morph attack, mentre, sull'asse delle y le corrispettive percentuali. Si evince un'analogia da quanto riportato dall'accuracy dato che il merge mostra ancora dei miglioramenti rispetto ai due approcci (tranne per facemorpher e OpenCV), sopratutto di quello geometrico.

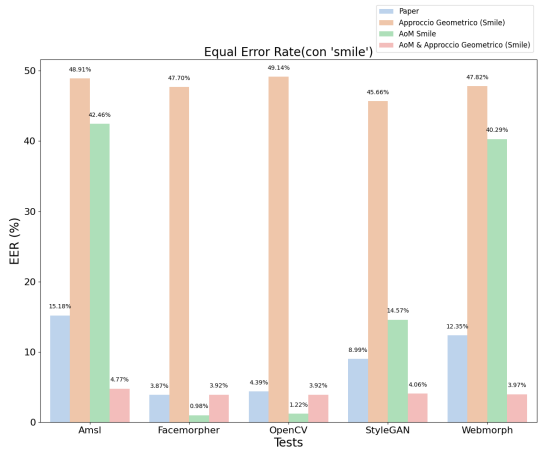

Figure 15. Confronto ERR Paper-AoM-AG

BPCER (%) @ APCER. Con una soglia di tolleranza così bassa solo l'approccio geometrico risulta prevalere rispetto al merge e all'AoM.

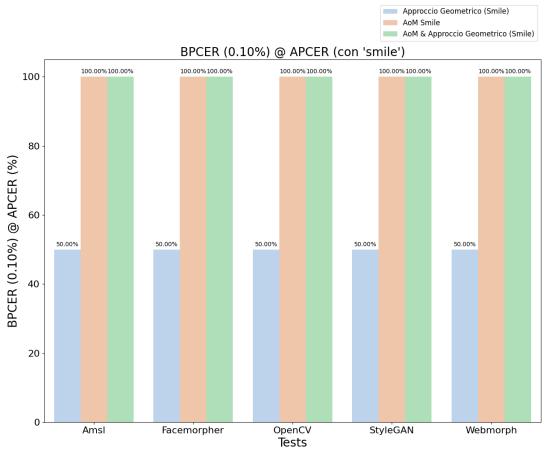

Figure 16. Soglia di tolleranza al 0,10% di errori sulla classificazione di bonafide

Con una tolleranza maggiore, AoM prevale (facemorpher, opencv e stylegan) per 1%, per 10% e 20% consideriamo anche amsl e webmorph.

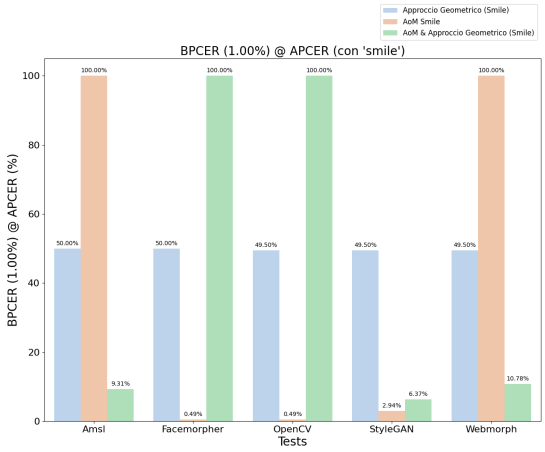

Figure 17. Soglia di tolleranza all'1% di errori sulla classificazione di bonafide

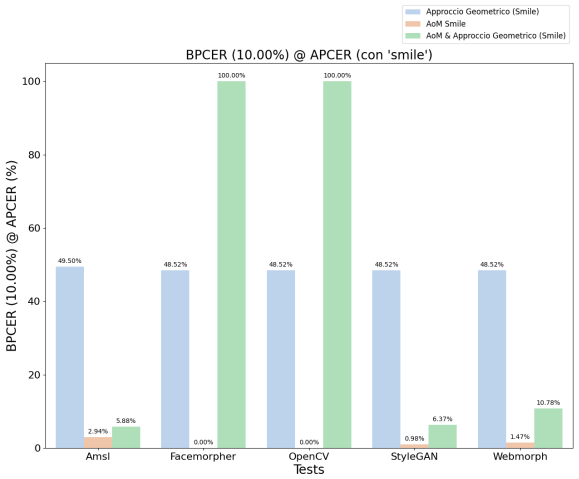

Figure 18. Soglia di tolleranza al 10% di errori sulla classificazione di bonafide

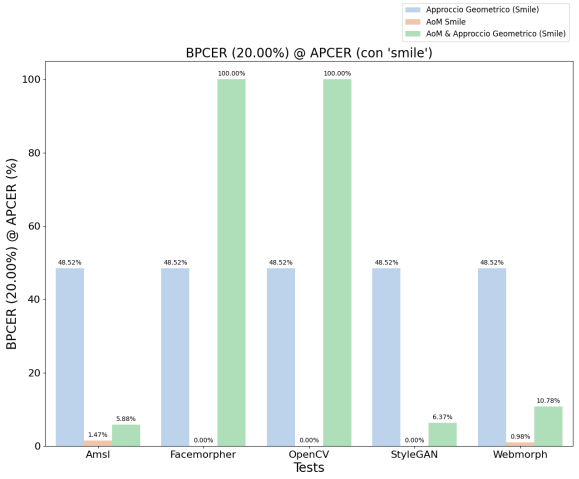

Figure 19. Soglia di tolleranza al 20% di errori sulla classificazione di bonafide

3.3.2 No Smile

In questa sezione si mostreremo i confronti fatti senza considerare le immagini con persone sorridenti.

Accuracy. In questo grafico vengono riportate le accuracy per ogni test che si è comportato meglio con un dato validation (il modello con il validation migliore è stato facemorpher, mentre amsl è ottimo di poco rispetto gli altri). Si può notare che il merge dei due approcci si è comportato, in media, meglio rispetto gli approcci presi singolarmente, migliorandoli entrambi. L'aspetto discordate compare in OpenCV e StyleGAN. Un altro aspetto interessante è come l'approccio geometrico in mancanza di immagini smile sia migliorato.

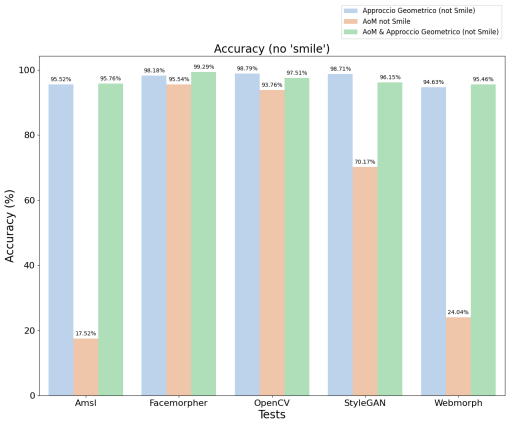

Figure 20. Confronto accuracy AoM-AG

EER. In questo grafico si evince un'analogia da quanto riportato dall'accuracy; questo perchè il merge mostra ancora dei miglioramenti rispetto ai due approcci, sopratutto di AoM.

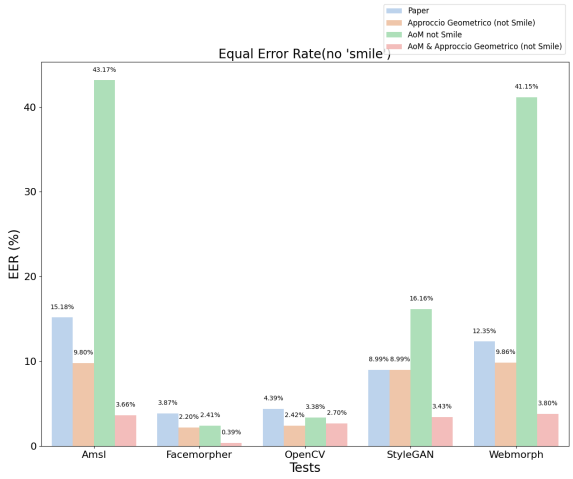

Figure 21. Confronto ERR Paper-AoM-AG

BPCER (%) @ APCER. Con una soglia di tolleranza cosi bassa (0,10% e 1%) solo l'approccio geometrico risulta prevalere rispetto al merge e ad AoM.

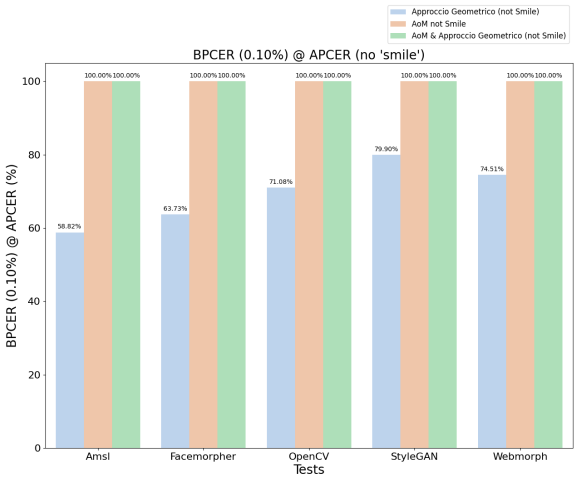

Figure 22. Soglia di tolleranza al 0,10% di errori sulla classificazione di bonafide

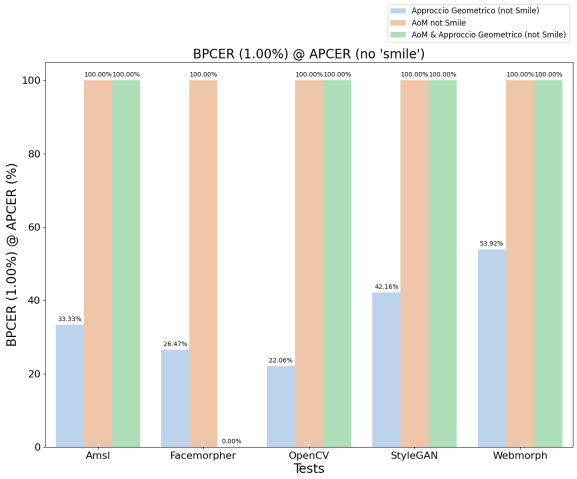

Figure 23. Soglia di tolleranza al 1% di errori sulla classificazione di bonafide

Con una tolleranza maggiore, il merge prevale andando a migliorare entrambi gli approcci, sia per il 10% che per il 20%. Si presentano delle anomalie nei casi di OpenCV e WebMorph.

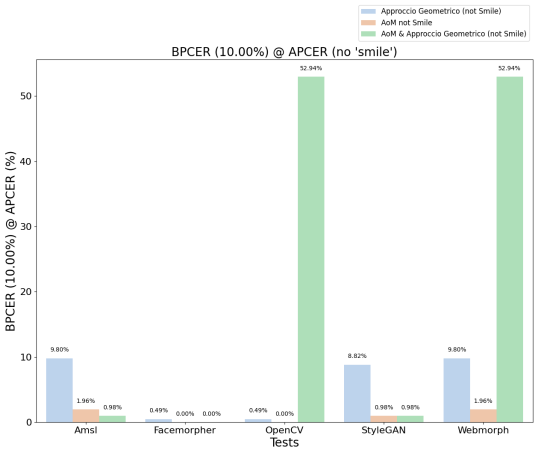

Figure 24. Soglia di tolleranza al 10% di errori sulla classificazione di bonafide

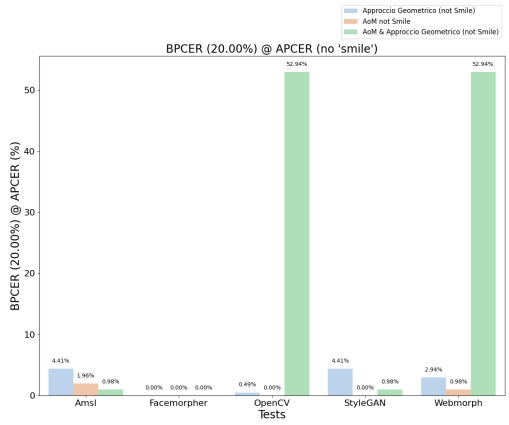

Figure 25. Soglia di tolleranza al 20% di errori sulla classificazione di bonafide

4 Data availability

Codice_Workflow:https://github.com/Alberto-00/Differential-MAD
Paper:openaccess.thecvf.com
Github_paper:https://github.com/naserdamer/SMDD-Synthetic-Face-Morphing-Attack-Detection-Development-dataset/tree/main
Datasets: https://drive.google.com/drive/folders

References

[1] N. Damer, C.A.F. López, M. Fang, N. Spiller, M.V. Pham, F. Boutros, *Privacy-friendly synthetic data for the development of face morphing attack detectors*, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 1606–1617

[2] S. Mallick, LearnOpenCV **1** (2016)

[3] E. Sarkar, P. Korshunov, L. Colbois, S. Marcel, arXiv preprint arXiv:2012.05344 (2020)

[4] M. Tan, Q.V. Le, CoRR (2019), 1907.09595