DIGITAL SIGNAL and IMAGE MANAGEMENT

Brought to you by

Anna Maria Teodori (889901), Michele Gazzola (825267)

CMU ARCTIC - Audio Classification

Agenda

01 Project Goals

02 Data Collection

03 Data Exploration

04 Data Preprocessing

05 Feature Extraction

06 Model Development and Evaluation

Project Goals

Data loading, exploration, preprocessing

Analysis of the audio files available to understand how to standardize them and extract features that can be used within the deep learning model

Accurate classification

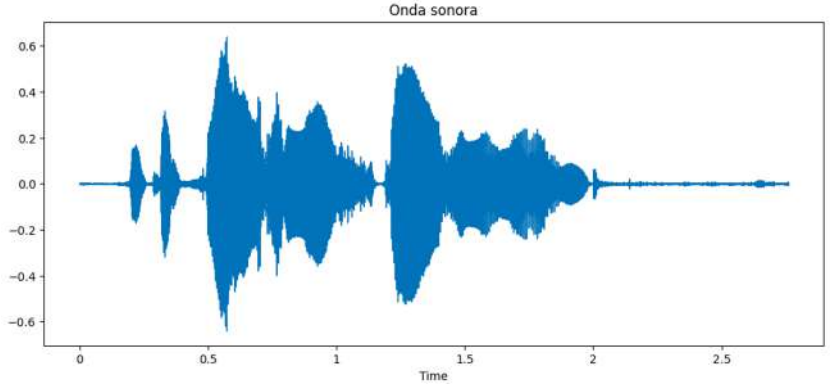
Consideration on applicability

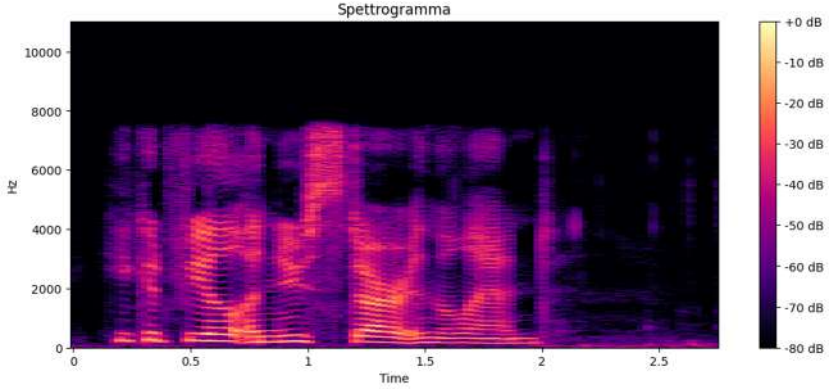
Dataset

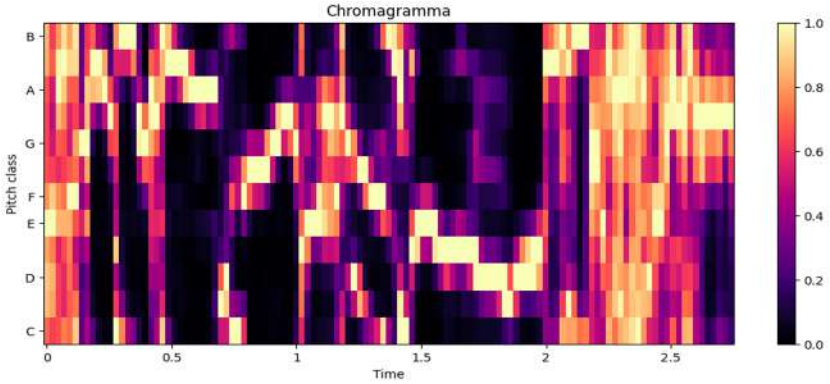

train
12501
test
3117
aup
awb
axb
bdl
clb
eey
fem
gka
arctic_a0006.wav
arctic_a0007.wav
arctic_a0008.wav
arctic_a0010.wav
arctic_a0011.wav
arctic_a0012.wav
arctic_a0013.wav
arctic_a0015.wav

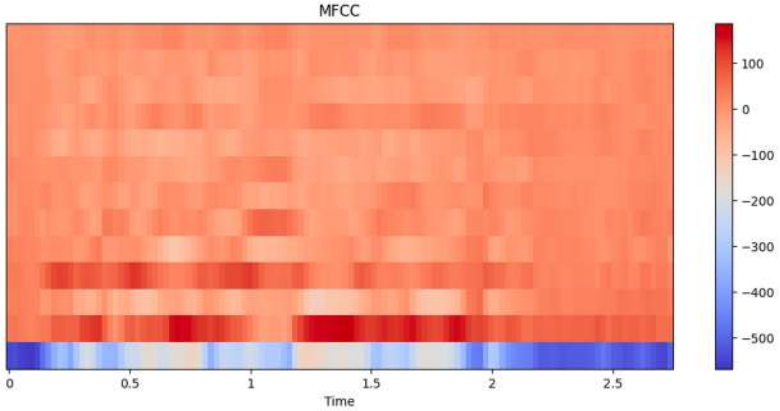
Data Exploration

multidimensional sound analysis



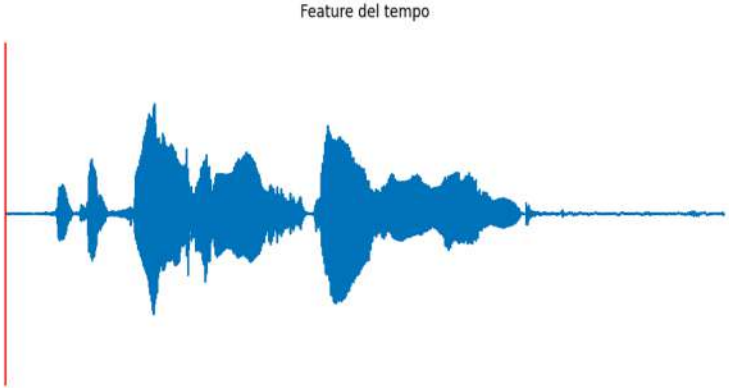


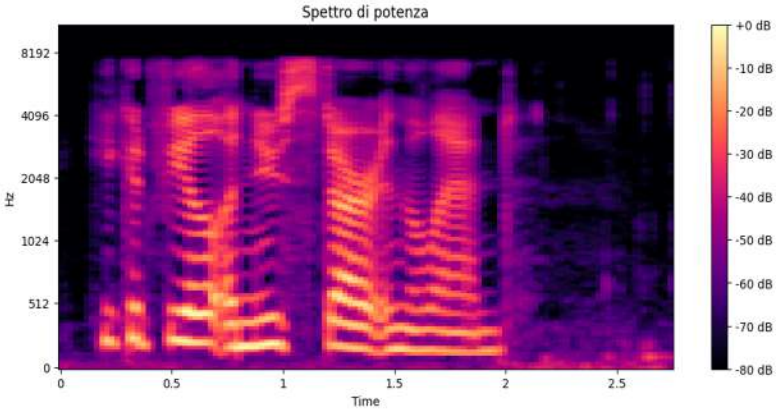


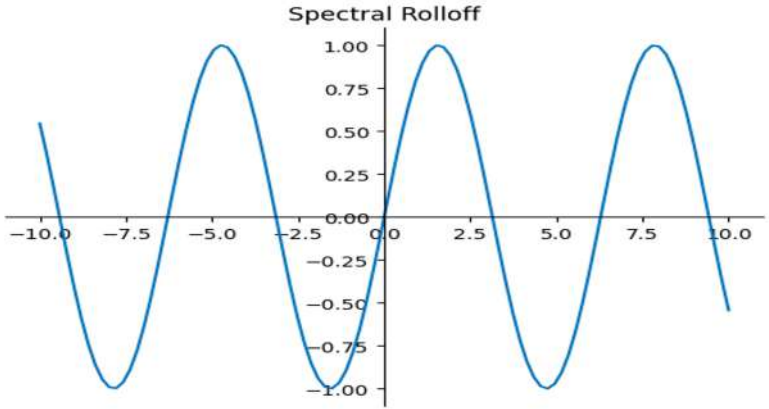


Data Exploration

multidimensional sound analysis







Data Exploration

analysis of the main properties

```python
def get_wav_info(wav_path):
    with wave.open(wav_path, 'rb') as wav_file:
        num_channels = wav_file.getnchannels()
        sample_rate = wav_file.getframerate()
        num_frames = wav_file.getnframes()
        duration = num_frames / float(sample_rate)
        bit_depth = wav_file.getsampwidth() * 8
        return num_channels, sample_rate, duration, bit_depth



wav_files = []
for root, dirs, files in os.walk(path_to_train_folder):
    for file in files:
        if file.endswith('.wav'):
            wav_files.append(os.path.join(root, file))

num_channels, sample_rate, duration, bit_depth = get_wav_info(wav_files[0])

print(f"Numero di canali: {num_channels}")
print(f"Frequenza di campionamento: {sample_rate} Hz")
print(f"Durata: {duration:.2f} secondi")
print(f"Profondità in bit dei campioni: {bit_depth} bit")
```

```
Numero di canali: 1
Frequenza di campionamento: 16000 Hz
Durata: 4.64 secondi
Profondità in bit dei campioni: 16 bit
```

```python
# lista dei file .wav
wav_files = train_wav_files

# dizionario per contare le frequenze di campionamento
sampling_freq_count = {}

# iterazione su ogni file .wav nella lista
for file in wav_files:
    # apertura del file
    with wave.open(file, 'r') as wav:
        # ottieni la frequenza di campionamento del file
        sampling_freq = wav.getframerate()

        # aggiorna il conteggio per la frequenza di campionamento corrente
        if sampling_freq in sampling_freq_count:
            sampling_freq_count[sampling_freq] += 1
        else:
            sampling_freq_count[sampling_freq] = 1

# stampa i risultati
print("Frequenze di campionamento e conteggio dei file che utilizzano quella frequenza:")
for freq, count in sampling_freq_count.items():
    print(f"{freq} Hz: {count} file")
```

```
Frequenze di campionamento e conteggio dei file che utilizzano quella frequenza:
16000 Hz: 12495 file
44100 Hz: 6 file
```

Data Exploration

analysis of the main properties

wave library
using this method: wav.getnchannels()

```python
# lista dei file .wav
wav_files = train_wav_files

# contatori per file mono e stereo
mono_count = 0
stereo_count = 0

# iterazione su ogni file .wav nella lista
for file in wav_files:
    # apertura del file
    with wave.open(file, 'r') as wav:
        # ottieni il numero di canali del file
        num_channels = wav.getnchannels()

        # controlla se il file è mono o stereo e aggiorna i contatori di conseguenza
        if num_channels == 1:
            mono_count += 1
        elif num_channels == 2:
            stereo_count += 1

# stampa i risultati
print(f"Numero di file mono: {mono_count}")
print(f"Numero di file stereo: {stereo_count}")
```

```
Numero di file mono: 12501
Numero di file stereo: 0
```

scipify library
using these methods: scipy.io.wavfile / data.ndim

```python
# lista dei file .wav
wav_files = train_wav_files

# contatori per file mono e stereo
mono_count = 0
stereo_count = 0

# iterazione su ogni file .wav nella lista
for file in wav_files:
    # leggi il file .wav
    rate, data = wavfile.read(file)

    # controlla se il file è mono o stereo e aggiorna i contatori di conseguenza
    if data.ndim == 1:
        mono_count += 1
    elif data.ndim == 2:
        stereo_count += 1

# stampa i risultati
print(f"Numero di file mono: {mono_count}")
print(f"Numero di file stereo: {stereo_count}")
```

```
Numero di file mono: 12501
Numero di file stereo: 0
```

Data Preprocessing

Standardization

```python
# definisci la nuova frequenza di campionamento (Hz)
new_sr = 16000

# definisci la nuova lunghezza dei file audio (in campioni)
new_length = int(new_sr * 4) # 4 secondi di durata

# definisci il percorso delle cartelle di input e output
input_folder = '/content/drive/MyDrive/PROGETTO_DSIM/DATASET/audio/train'
output_folder = '/content/train_norm'

# itera attraverso tutte le sottocartelle della cartella di input
for root, dirs, files in os.walk(input_folder):
    for file in files:
        # verifica che il file sia un file .wav
        if file.endswith('.wav'):
            # carica il file audio con Librosa
            filepath = os.path.join(root, file)
            signal, sr = librosa.load(filepath, sr=None)
            # standardizza la frequenza di campionamento
            signal_resampled = librosa.resample(signal, orig_sr=sr, target_sr=new_sr)
            # uniforma la lunghezza del segnale audio
            if len(signal_resampled) < new_length:
                signal_resampled = librosa.util.pad_center(data=signal_resampled, size=new_length)
            else:
                signal_resampled = signal_resampled[:new_length]
            # crea il percorso di output e salva il nuovo file audio
            output_dir = os.path.join(output_folder, os.path.relpath(root, input_folder))
            output_path = os.path.join(output_dir, file)
            os.makedirs(output_dir, exist_ok=True)
            sf.write(output_path, signal_resampled, new_sr)
```

Data Preprocessing

Standardization

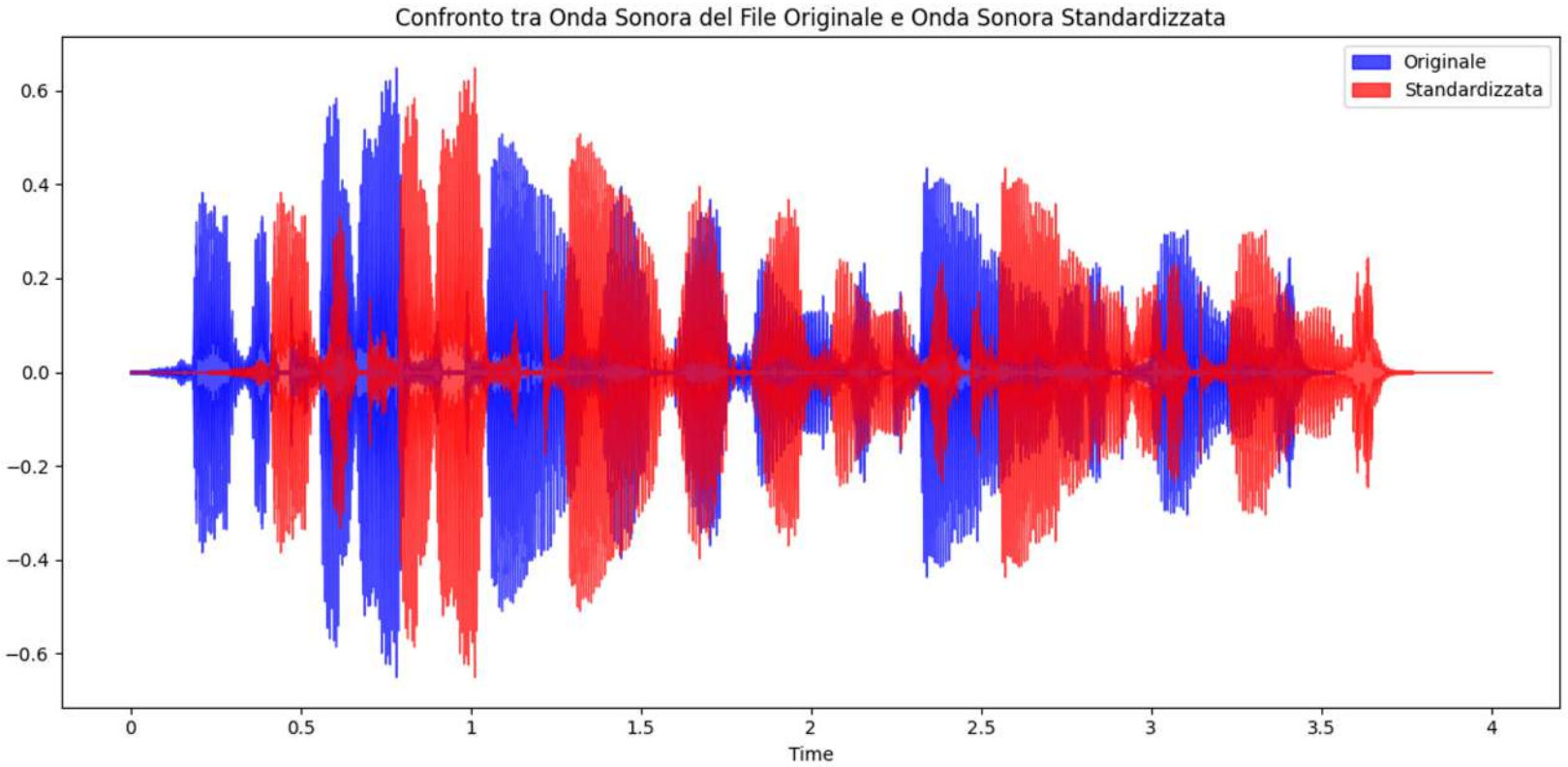
```python
train_dir = '/content/drive/MyDrive/PROGETTO_DSIM/DATASET/audio/train'
train_norm_dir = '/content/train_norm'

same_files = 0
different_files = 0

for subdir, _, files in os.walk(train_dir):
    # Create the corresponding subdirectory in train_norm
    norm_subdir = subdir.replace(train_dir, train_norm_dir)
    os.makedirs(norm_subdir, exist_ok=True)

    # Check each file in the current subdirectory
    for file in files:
        # Check if the file exists in train_norm
        norm_file = os.path.join(norm_subdir, file)
        if os.path.exists(norm_file):
            # Compare the files and update the counters accordingly
            if filecmp.cmp(os.path.join(subdir, file), norm_file, shallow=False):
                same_files += 1
                print(f"{file} is the same in train and train_norm")
            else:
                different_files += 1
        else:
            different_files += 1

print(f"Number of files that are the same: {same_files}")
print(f"Number of files that are different: {different_files}")
```

output:

```
arctic_a0435.wav is the same in train and train_norm
arctic_b0534.wav is the same in train and train_norm
arctic_a0582.wav is the same in train and train_norm
Number of files that are the same: 3
Number of files that are different: 12498
```

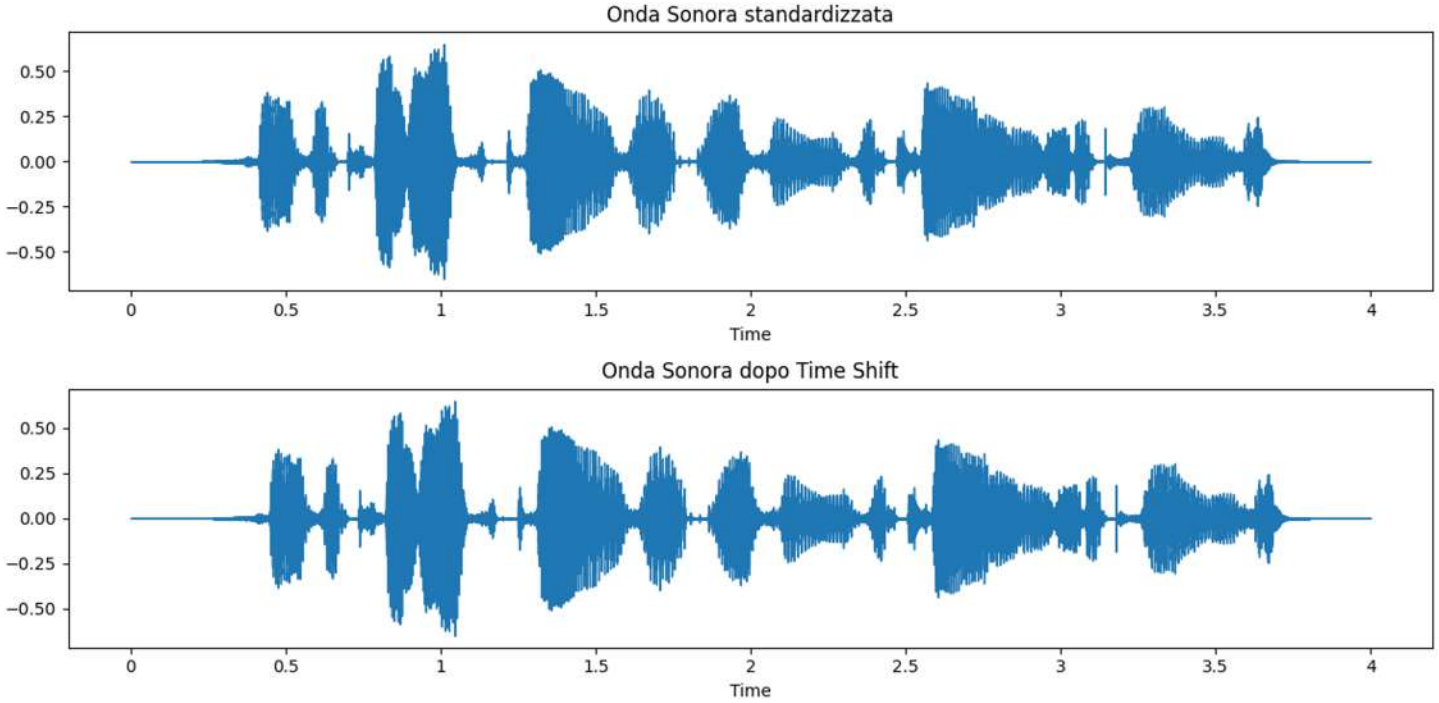
Data Preprocessing

Standardization



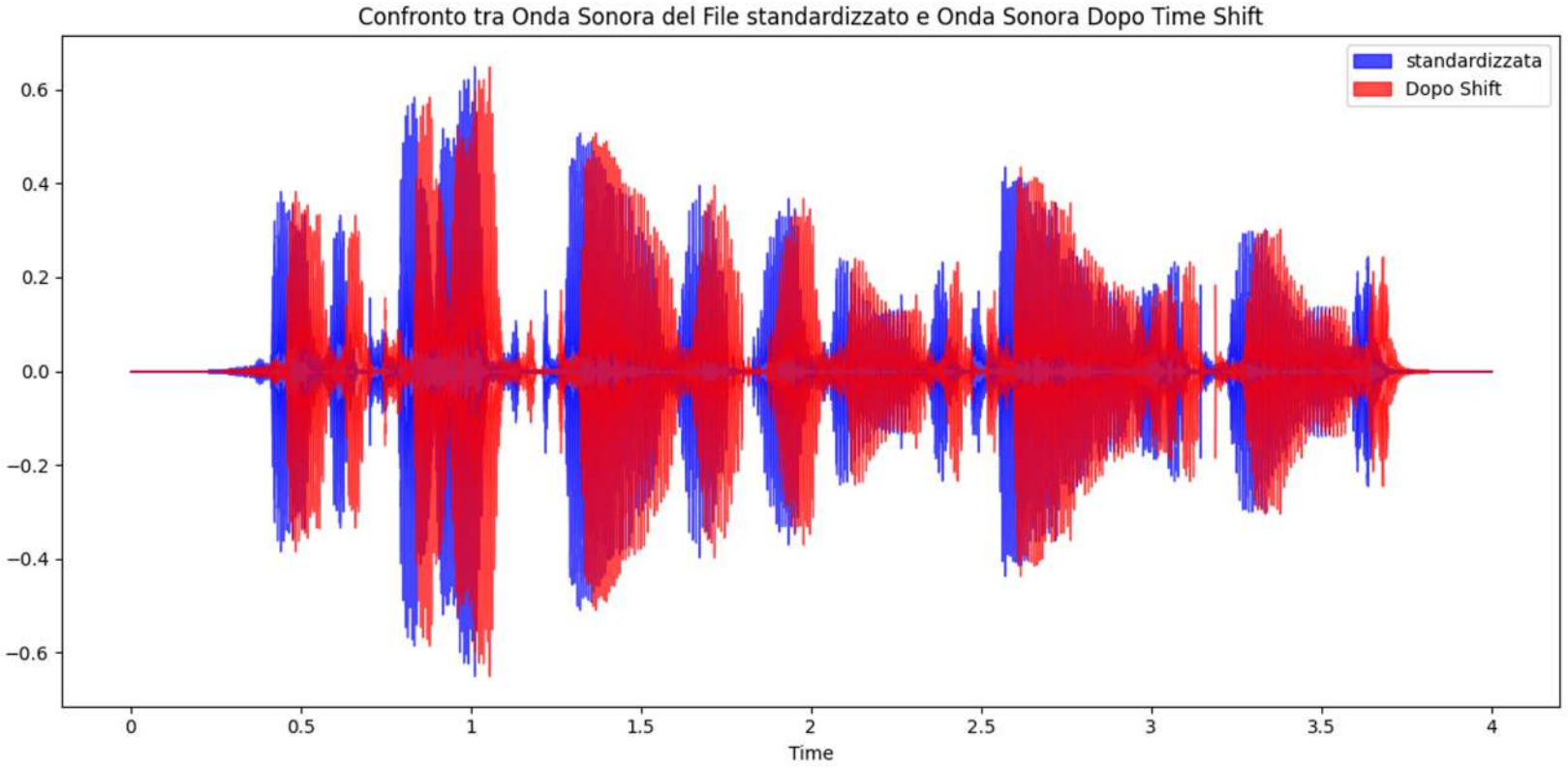
Data Preprocessing

Time-Shift



Data Preprocessing

Time-Shift



Data Preprocessing

CSV Creation

kaggle csv:

```
with open('/content/drive/MyDrive/PROGETTO_DSIM/DATASET/audio/train_kaggle.csv', 'r') as file:
    reader = csv.reader(file)
    row_count = sum(1 for row in reader)

print(f"Il file ha {row_count} righe.")
```

```
Il file ha 12467 righe.
```

```
# Creazione di un dataframe vuoto con due colonne
df = pd.DataFrame(columns=['file_name', 'speakers'])

# Percorso della cartella principale
root_path = "/content/drive/MyDrive/PROGETTO_DSIM/DATASET/audio/train"

# Itera attraverso le sottocartelle della cartella principale
for subdir, dirs, files in os.walk(root_path):
  # Itera attraverso i file nella sottocartella corrente
  for file in files:
    # Verifica se il file ha estensione .wav
    if file.endswith('.wav'):
      # Aggiunge il nome del file e della sottocartella al dataframe
      df = df.append({'file_name': file, 'speakers': os.path.basename(subdir)}, ignore_index=True)

# Salvataggio del dataframe in formato CSV
df.to_csv('/content/train_creato.csv', index=False)
```

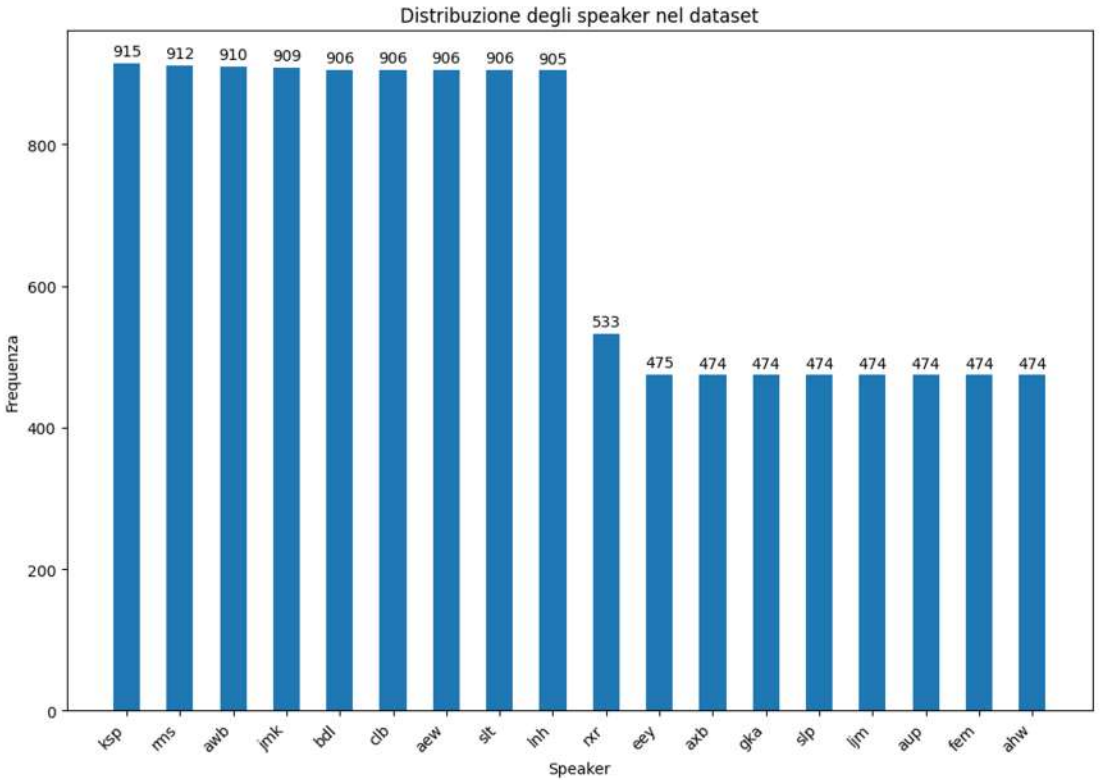
file_name	speakers
arctic_a0116.wav	ahw
arctic_a0108.wav	ahw
arctic_a0081.wav	ahw
arctic_a0088.wav	ahw
arctic_a0012.wav	ahw
arctic_a0106.wav	ahw
arctic_a0033.wav	ahw

⋮

File.wav Distribution for Every Speaker

```
#controllo se la variabile "speaker" è sbilanciata
speaker_counts = df["speakers"].value_counts()
speaker_counts
```

```
ksp    915
rms    912
awb    910
jmk    909
bdl    906
clb    906
aew    906
slt    906
lnh    905
rxr    533
eey    475
axb    474
gka    474
slp    474
ljm    474
aup    474
fem    474
ahw    474
Name: speakers, dtype: int64
```



Feature Extraction

```python
# Funzione per estrarre le features MFCC
def extract_mfcc_features(file):
    # Carica il file audio
    audio, sample_rate = librosa.load(file)
    # Estrae gli MFCC
    mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
    # Calcola le caratteristiche ridimensionate
    mfccs_scaled_features = np.mean(mfccs_features.T, axis=0)
    return mfccs_scaled_features

# Funzione per estrarre l'energia
def extract_energy(file):
    # Carica il file audio
    audio, sample_rate = librosa.load(file)
    # Calcola l'energia
    energy = np.sum(audio**2)
    return energy

# Funzione per estrarre lo ZCR
def extract_zcr(file):
    # Carica il file audio
    audio, sample_rate = librosa.load(file)
    # Calcola lo ZCR
    zcr = np.mean(librosa.feature.zero_crossing_rate(audio))
    return zcr
```

```python
# Funzione per estrarre la durata
def extract_duration(file):
    # Carica il file audio
    audio, sample_rate = librosa.load(file)
    # Calcola la durata
    duration = librosa.get_duration(y=audio, sr=sample_rate)
    return duration

# Percorso della cartella principale contenente le sottocartelle di file audio
main_folder = "/content/train_norm"

# Inizializzazione del dataframe
df_norm = pd.DataFrame(columns=['file_name', 'speakers', 'mfcc_features', 'energy', 'zcr', 'duration'])
```

Feature Extraction

```python
# Iterazione attraverso le sottocartelle della cartella principale
for subfolder in os.listdir(main_folder):
    # Percorso della sottocartella corrente
    subfolder_path = os.path.join(main_folder, subfolder)
    # Iterazione attraverso i file .wav della sottocartella corrente
    for file_name in os.listdir(subfolder_path):
        if file_name.endswith('.wav'):
            # Percorso del file corrente
            file_path = os.path.join(subfolder_path, file_name)
            # Estrazione delle features
            mfcc_features = extract_mfcc_features(file_path)
            energy = extract_energy(file_path)
            zcr = extract_zcr(file_path)
            duration = extract_duration(file_path)
            # Aggiunta delle informazioni al dataframe
            df_norm = df_norm.append({'file_name': file_name,
                                      'speakers': subfolder,
                                      'mfcc_features': mfcc_features,
                                      'energy': energy,
                                      'zcr': zcr,
                                      'duration': duration}, ignore_index=True)

# Salvataggio del dataframe in un file csv
df_norm.to_csv('train_norm.csv', index=False)
```

file_name	speakers	mfcc_features	energy	zcr	duration
arctic_b0030.wav	clb	[-3.09226715e+02 8 -1.13130283e+01 7 -4.32095146e+00 -1 2.74854159e+00 -9 -4.79842472e+00 -7 -7.77592421e+00 -8 -6.05466366e+00 6 8.83970857e-01 -1 2.58629870e+00 1 9.37820339e+00 8	1367.5433	0.1154771044	4
arctic_b0082.wav	clb	[-3.0937003e+02 7. -1.3407845e+01 1. -2.7064645e+00 7. 4.9939814e+00 -8. -2.0424159e+00 -6. -6.9192557e+00 -5. -5.5551467e+00 -5. -1.4648350e+00 -9. 3.5468097e+00 -1. 1.5616586e+00 2.	880.4661	0.1532638638	4
arctic_a0200.wav	clb	[-3.6385513e+02 9. -8.1757412e+00 2. -4.0098605e+00 -1. -5.7402551e-01 -8.6 -6.8769059e+00 -5. -6.5126171e+00 -7. -5.5096745e+00 -2. -2.8515637e+00 -3. 6.1866891e-01 -2.2 5.2597256e+00 7.	1224.9895	0.06028438855	4
⋮					

Feature Extraction

Check Datasets

```python
# Carica i file CSV in dataframe
df_norm = pd.read_csv('/content/drive/MyDrive/PROGETTO_DSIM/DATASET/audio/train_csv_da_usare/train_norm.csv')
df = pd.read_csv('/content/drive/MyDrive/PROGETTO_DSIM/DATASET/audio/train_csv_da_usare/train_creato_1.csv')

# Controlla se i due dataframe hanno lo stesso numero di righe
if len(df_norm) == len(df):
    print('I due CSV hanno lo stesso numero di righe')
else:
    print('I due CSV non hanno lo stesso numero di righe')

# Verifica se i nomi dei file nella colonna 'file_name' di "train_norm.csv"
# corrispondono ai nomi nella colonna 'file_name' di "train_creato.csv"
if set(df_norm['file_name']) == set(df['file_name']):
    print('I nomi dei file corrispondono in entrambi i CSV')
else:
    print('I nomi dei file non corrispondono in entrambi i CSV')
```

```
I due CSV hanno lo stesso numero di righe
I nomi dei file corrispondono in entrambi i CSV
```

```python
# Carica i due dataframe
df = pd.read_csv('/content/drive/MyDrive/PROGETTO_DSIM/DATASET/audio/train_csv_da_usare/train_creato_1.csv')
df_norm = pd.read_csv('/content/drive/MyDrive/PROGETTO_DSIM/DATASET/audio/train_csv_da_usare/train_norm.csv')

# Crea i set dei nomi dei file presenti in ogni dataframe
set1 = set(df['file_name'])
set2 = set(df_norm['file_name'])

# Trova i nomi di file presenti in uno ma non nell'altro
diff1 = set1 - set2
diff2 = set2 - set1

# Stampa i nomi di file presenti solo in train.csv e audio_features.csv
print("Nomi di file presenti solo in train.csv:")
print(diff1)
print("Nomi di file presenti solo in audio_features.csv:")
print(diff2)

# Conta il numero di file diversi tra i due file.csv
num_diff_files = len(diff1) + len(diff2)
print(f"Il numero di file diversi tra i due file.csv è: {num_diff_files}")
```

```
Nomi di file presenti solo in train.csv:
set()
Nomi di file presenti solo in audio_features.csv:
set()
Il numero di file diversi tra i due file.csv è: 0
```

Feature Extraction

Add ID Column

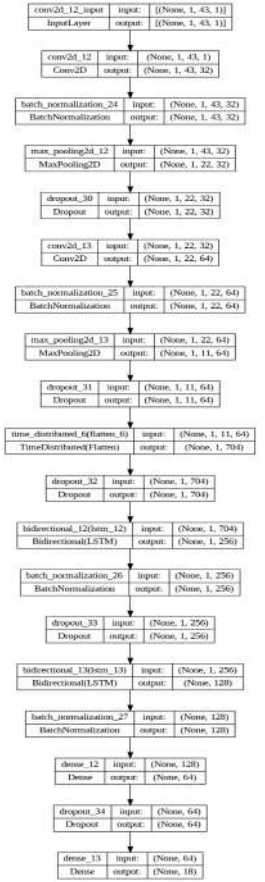
```python
# legge i file CSV
df = pd.read_csv('/content/drive/MyDrive/PROGETTO_DSIM/DATASET/audio/train_csv_da_usare/train_creato_1.csv')
df_norm = pd.read_csv('/content/drive/MyDrive/PROGETTO_DSIM/DATASET/audio/train_csv_da_usare/train_norm.csv')

# crea un dizionario di mapping tra (file_name, speakers) e id
id_map = {}
id_counter = 0
for idx, row in pd.concat([df, df_norm]).iterrows():
    key = (row['file_name'], row['speakers'])
    if key not in id_map:
        id_map[key] = hashlib.sha256(str(id_counter).encode()).hexdigest()
        id_counter += 1

# aggiungi la colonna "id" ai dataframe df1 e df2
df['id'] = df.apply(lambda row: id_map.get((row['file_name'], row['speakers'])), axis=1)
df_norm['id'] = df_norm.apply(lambda row: id_map.get((row['file_name'], row['speakers'])), axis=1)

# salva i file CSV aggiornati
df.to_csv('/content/TRAIN_CREATO_ID.csv', index=False)
df_norm.to_csv('/content/TRAIN_FINALE_NORM.csv', index=False)
```

file_name	speakers	mfcc_features	energy	zcr	duration	id
arctic_b0030.wav	clb	[-3.09226715e+02 8 -1.13130283e+01 7 -4.32095146e+00 -1 2.74854159e+00 -9 -4.79842472e+00 -7 -7.77592421e+00 -8 -6.05466366e+00 6 8.83970857e-01 -1 2.58629870e+00 1 9.37820339e+00 8	1367.5433	0.1154771044	4	995d6f3a9b46a039a
arctic_b0082.wav	clb	[-3.0937003e+02 7. -1.3407845e+01 1. -2.7064645e+00 7. 4.9939814e+00 -8. -2.0424159e+00 -6. -6.9192557e+00 -5. -5.5551467e+00 -5. -1.4648350e+00 -9. 3.5468097e+00 -1. 1.5616586e+00 2.	880.4661	0.1532638638	4	fd488ef7f15f0a3c0f4

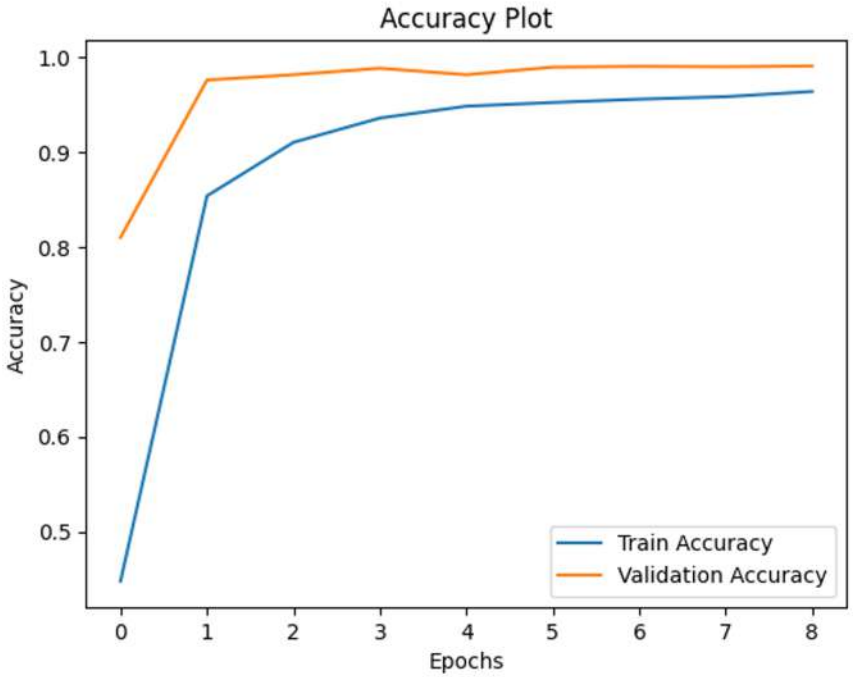
Model Development

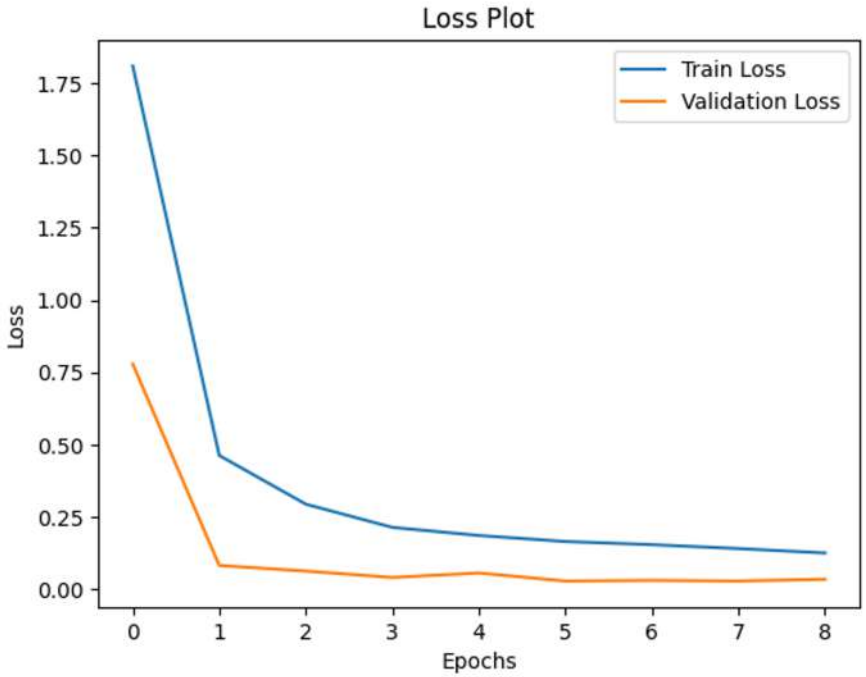
Train and Test split:

```
data = pd.read_csv("/content/drive/MyDrive/PROGETTO_DSIM/DATASET/audio/train_csv_da_usare/TRAIN_FINALE_NORM.csv")

# Estrazione delle feature
X = []
for row in data.itertuples():
    # Estrazione dei mfcc features
    mfcc_feature = np.fromstring(row.mfcc_features.replace('[', '').replace(']', ''), sep=' ')
    # Calcolo degli altri feature
    duration = row.duration
    energy = row.energy
    zcr = row.zcr
    # Concatenazione di tutte le feature
    features = np.concatenate([mfcc_feature, [duration, energy, zcr]])
    X.append(features)
X = np.array(X)

# Codifica delle etichette
le = LabelEncoder()
y = le.fit_transform(data['speakers'])


# Codifica delle etichette con one-hot
y_onehot = to_categorical(y)


# Divide il dataset in insieme di training e di test
X_train, X_test, y_train, y_test = train_test_split(X, y_onehot, test_size=0.2, random_state=42)

# Reshape delle feature in modo da poterle passare al modello CNN/LSTM
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```



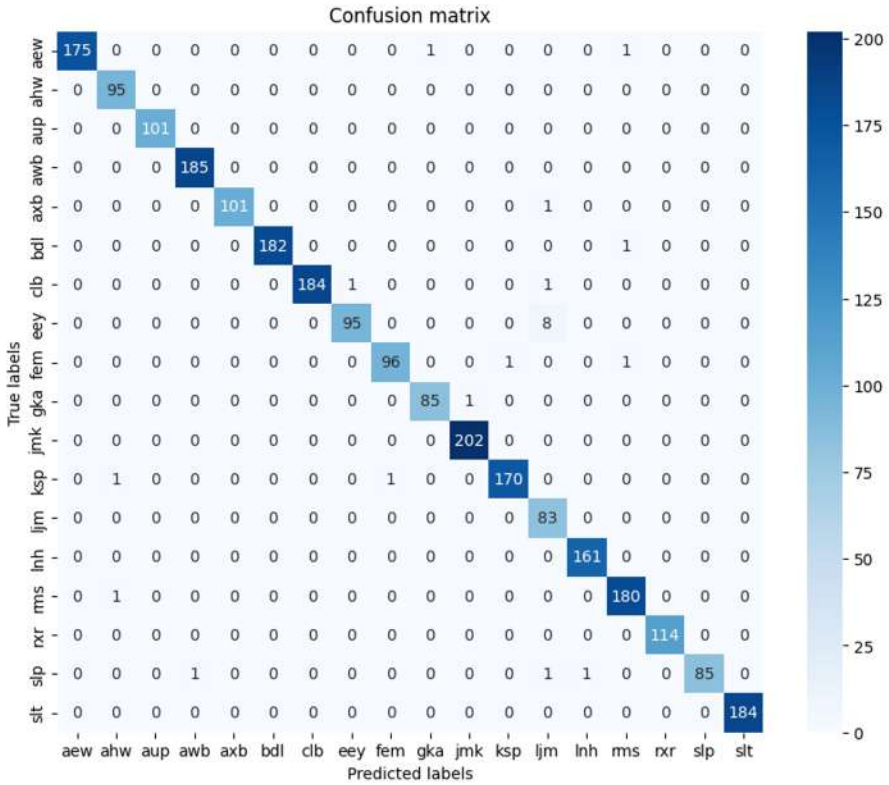
Accuracy & Loss





Model report

Confusion Matrix



Accuracy: 0.9912

Loss: 0.0318

Precision: 0.9914

Recall: 0.9912

F1 Score: 0.9911

Labelled Faces in the Wild - GAN

Agenda

01 Project Goals

02 Data Collection

03 Data Exploration

04 Data Preprocessing

05 Data Modeling

06 Model Evaluation

Project Goals

Exploration of data images

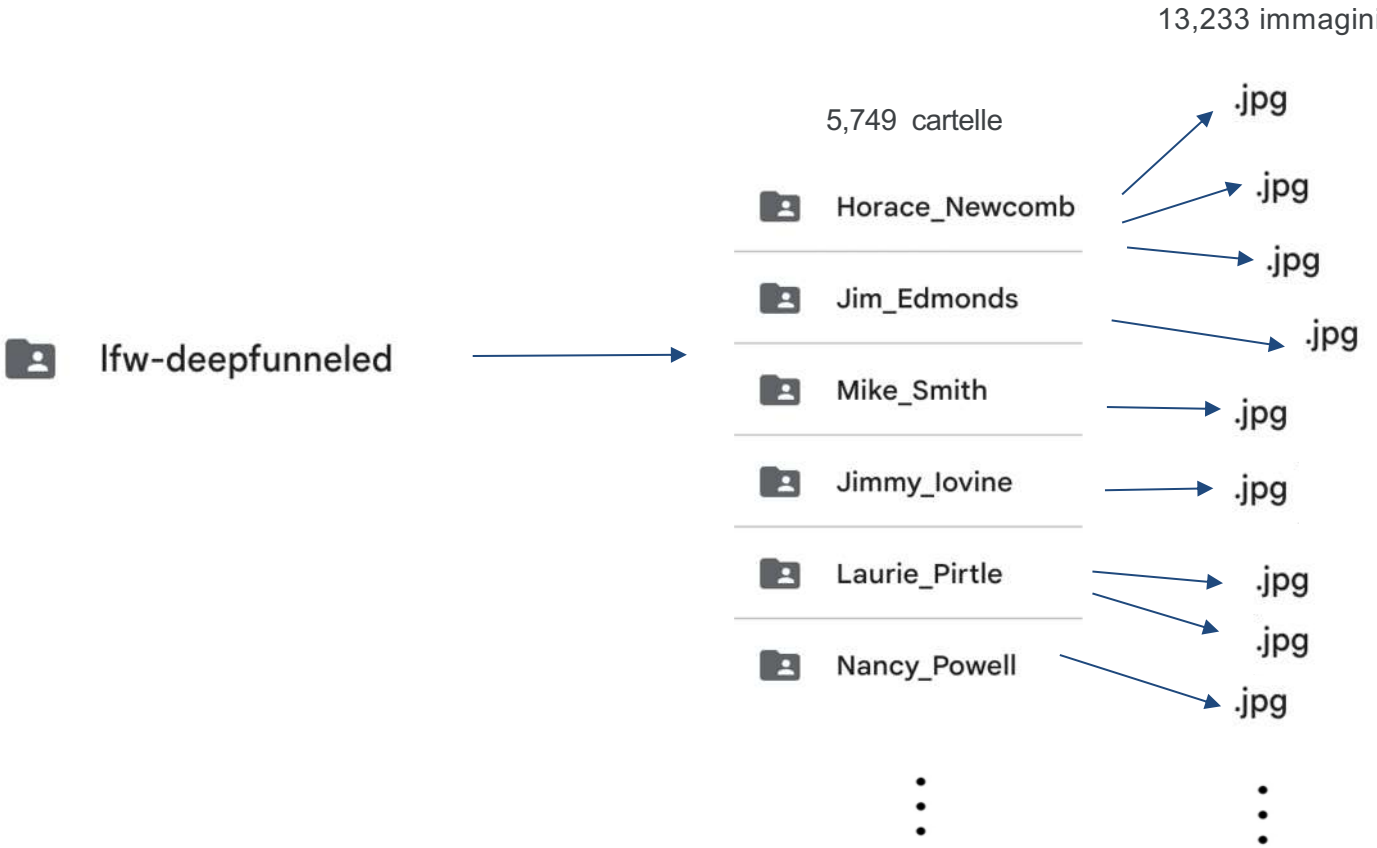
Experiments two types of preprocessing one by the face detection and one by cropping

Data loading

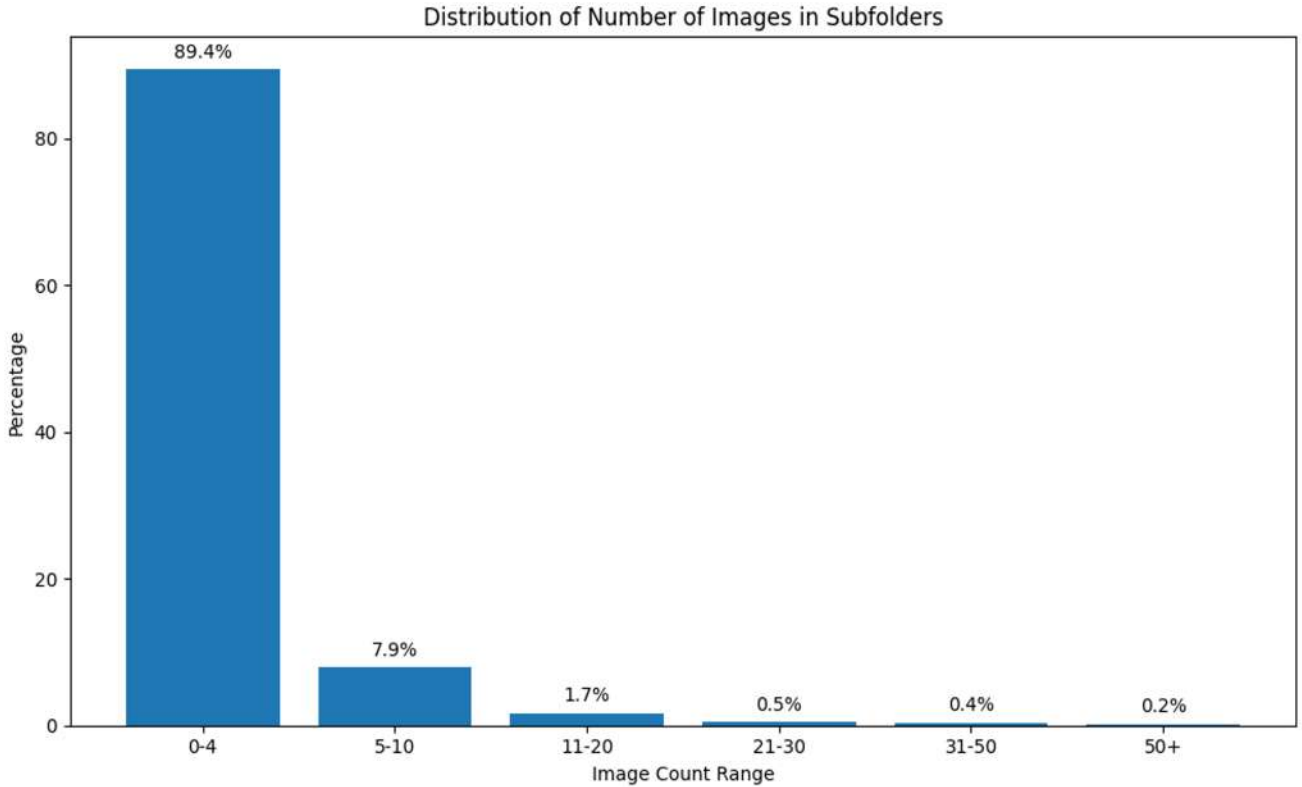
Test two types of GAN models

Evaluation about the results

Dataset


13,233 immagini
5,749 cartelle
lfw-deepfunneled
Horace_Newcomb
Jim_Edmonds
Mike_Smith
Jimmy_Iovine
Laurie_Pirtle
Nancy_Powell
.jpg
.jpg
.jpg
.jpg
.jpg
.jpg
.jpg
.jpg
.jpg

Dataset Exploration



Pre processing with face detection

13,233 immagini

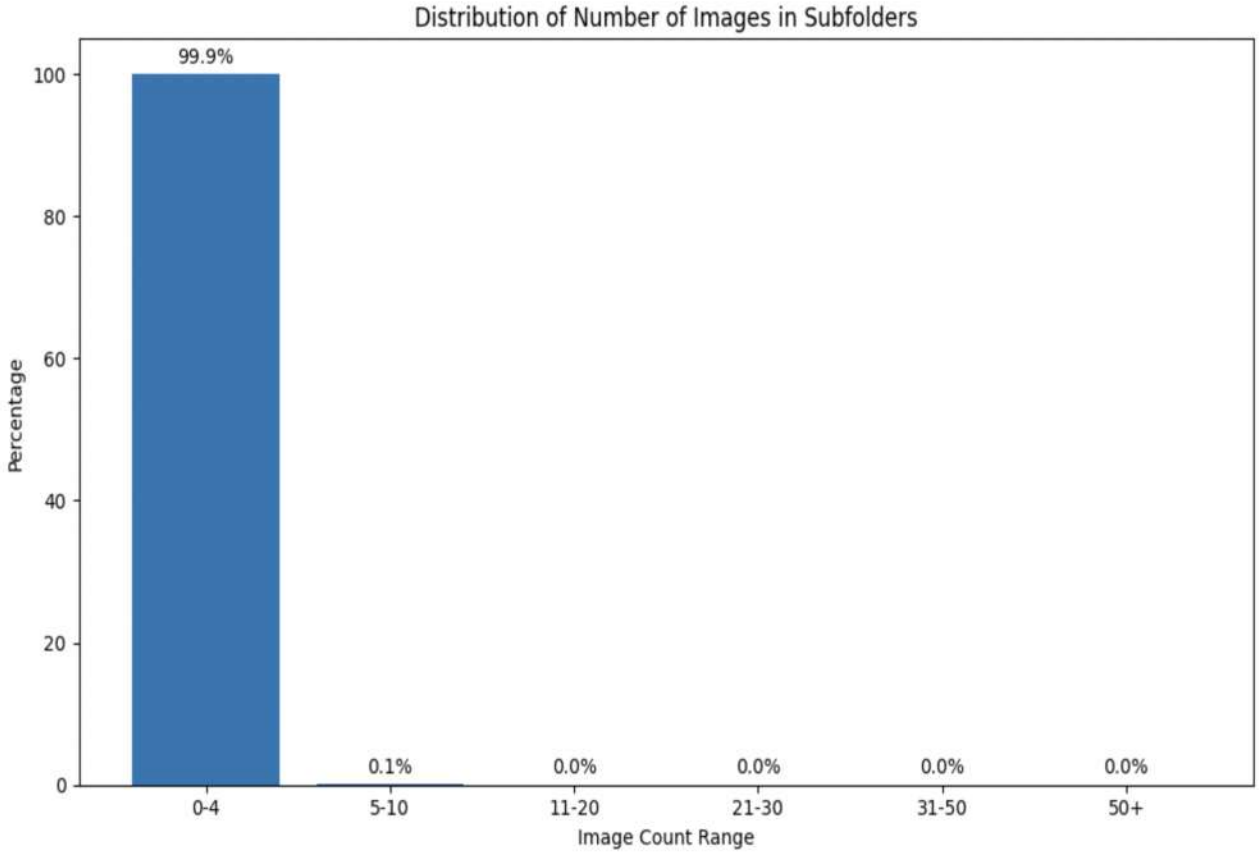
↓

lfw-deepfunneled

6,560 immagini

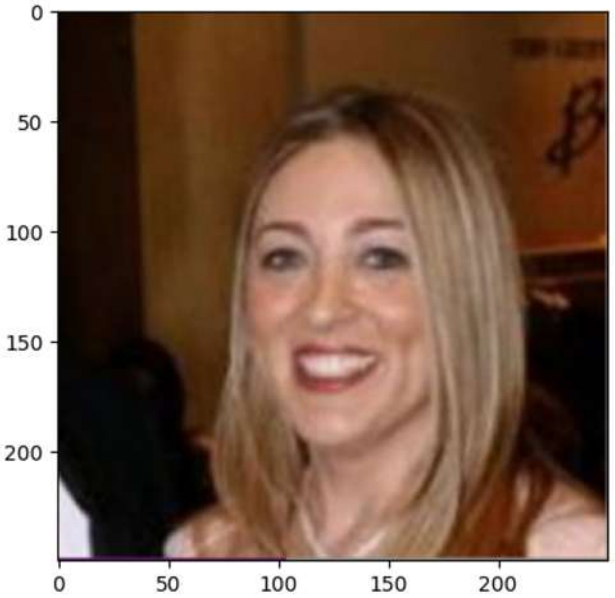
↓

lfw-deepfunneled-faces

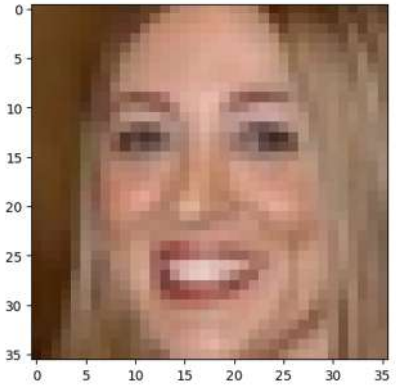


Pre processing with face detection

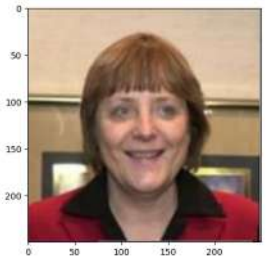
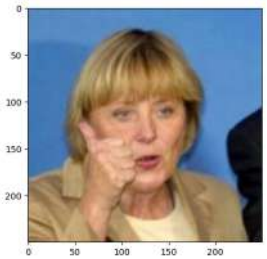
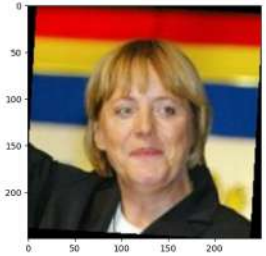
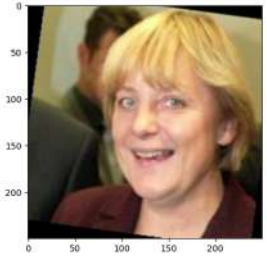
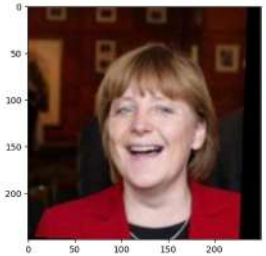
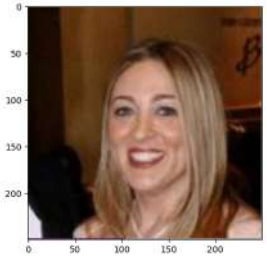
250x250

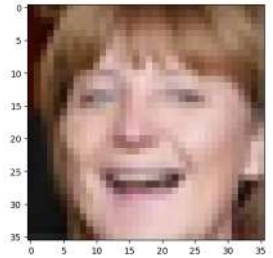
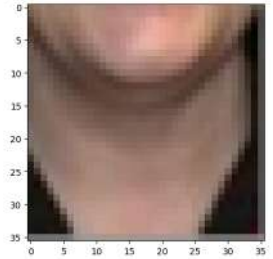


36x36



Pre processing with face detection

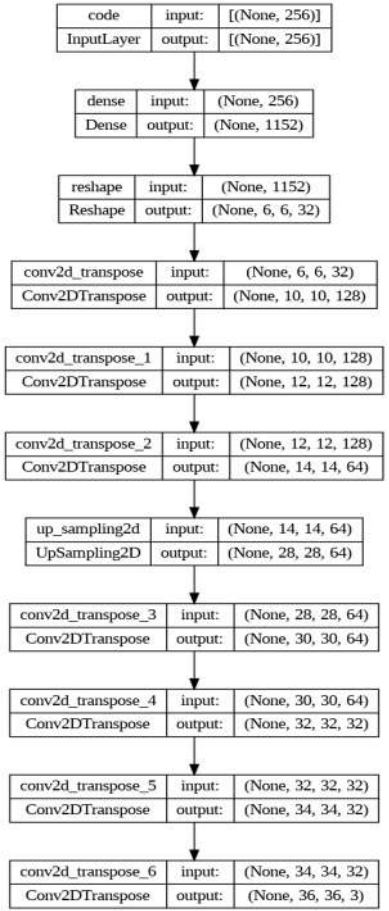







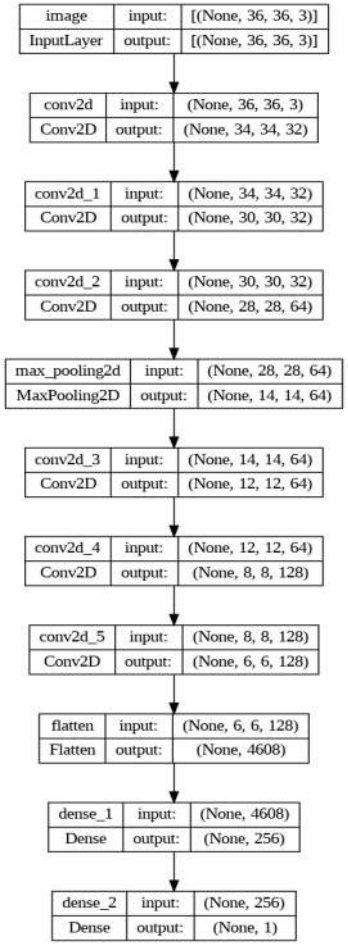
Model 1

Face Detection

Generatore



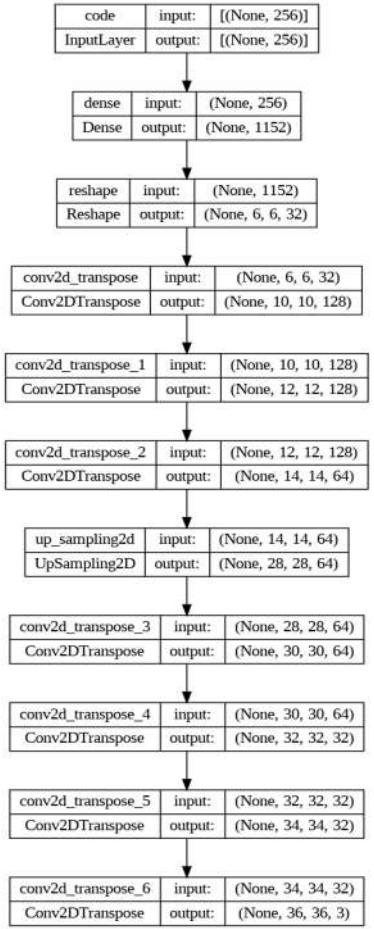
Discriminatore



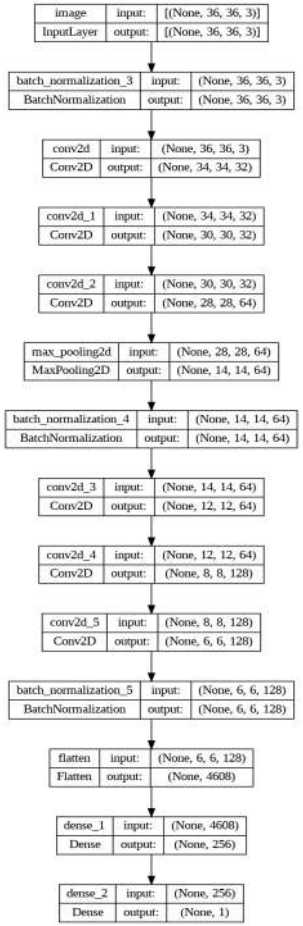
Model 2

Face Detection

Generatore



Discriminatore



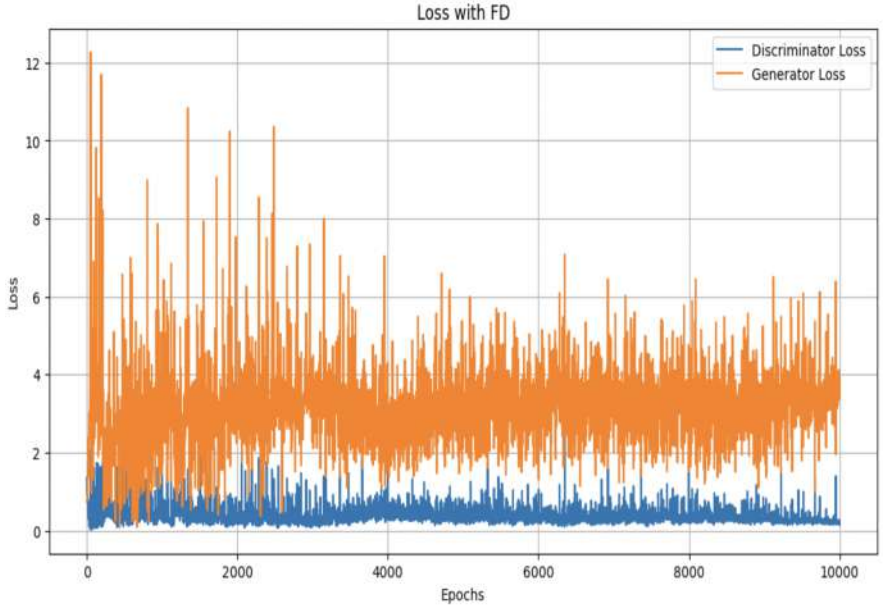
Quality of images

Model 1

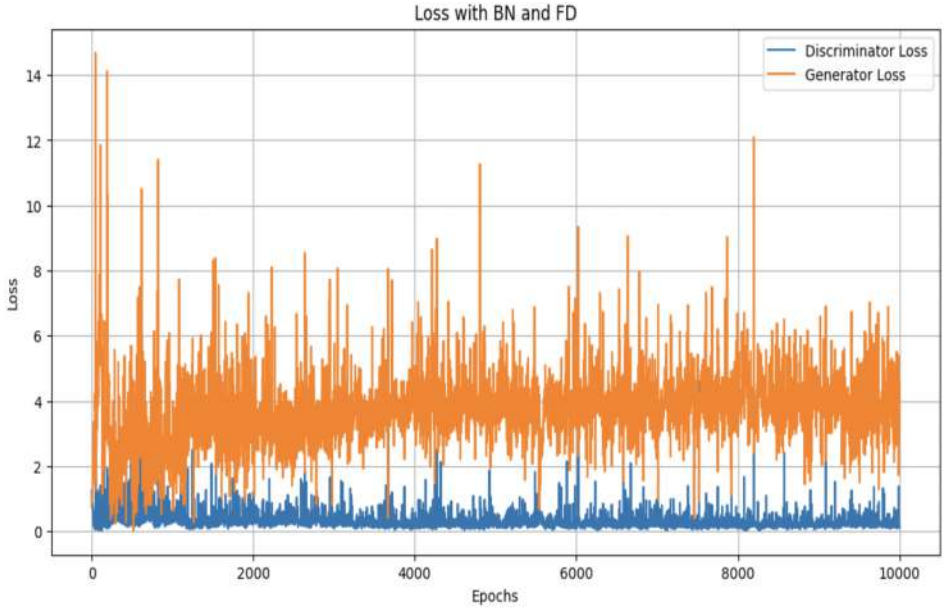
Model 2

Loss

Model 1

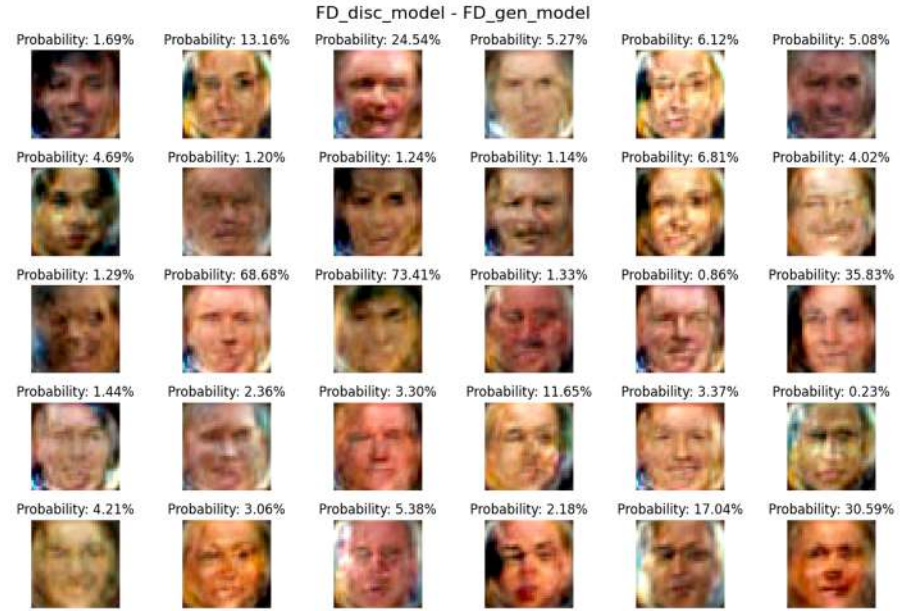


Model 2

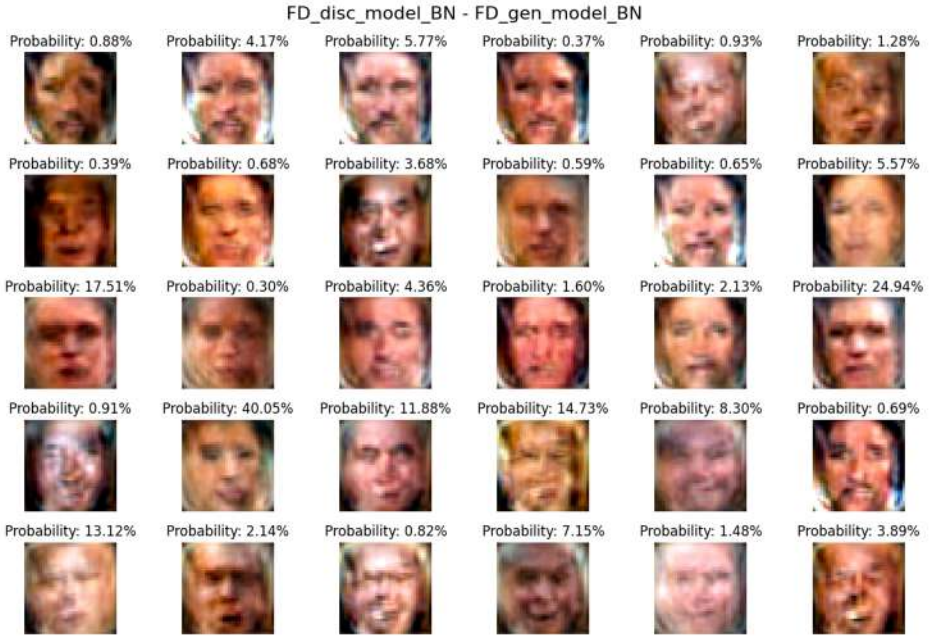


Some predicted images

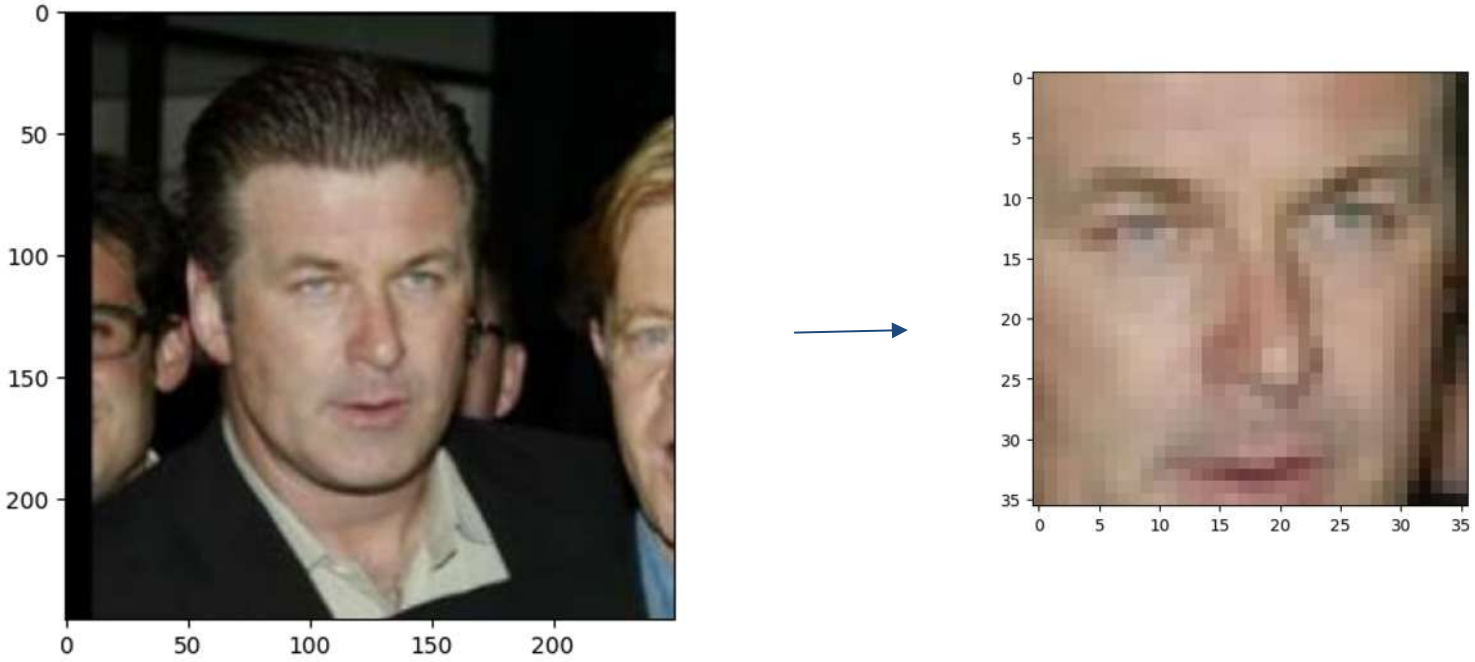
Model 1



Model 2



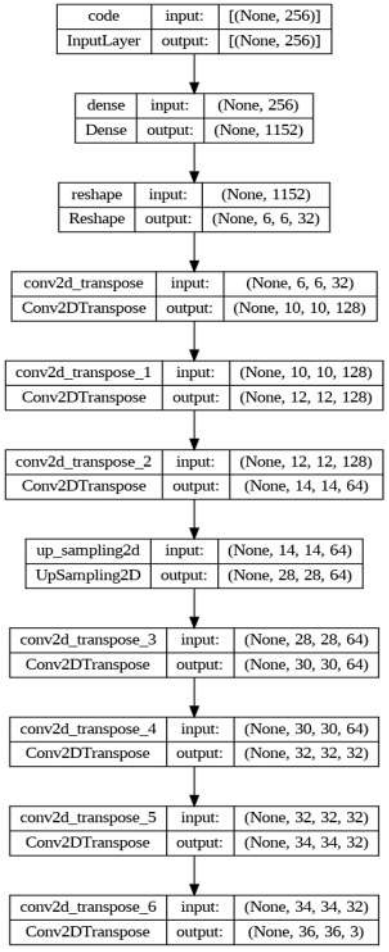
Pre processing with cropping



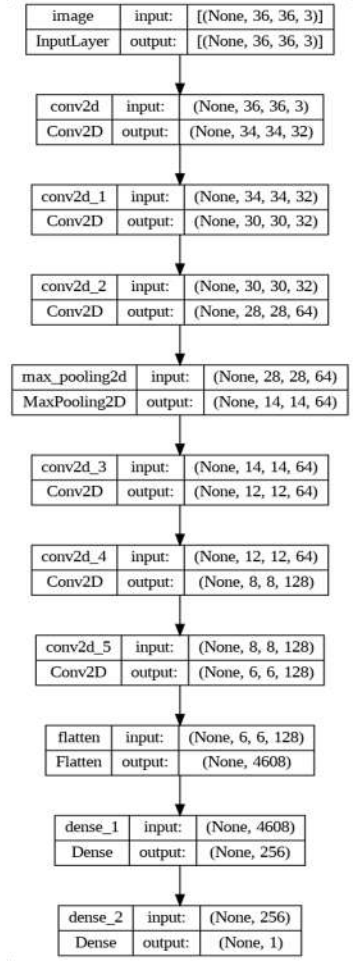
Model 1

Cropping - centered images

Generatore



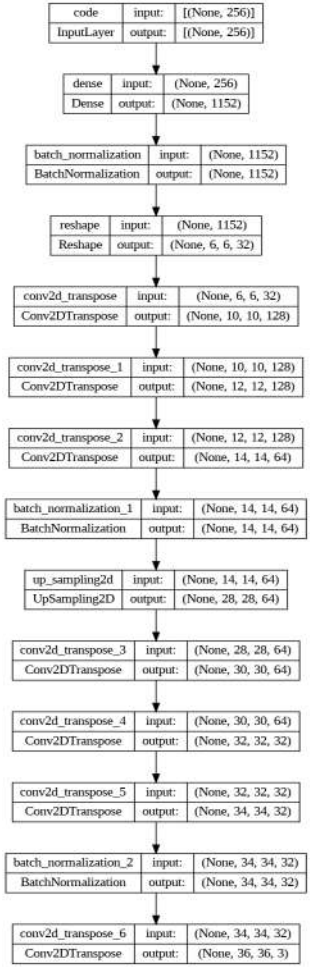
Discriminatore



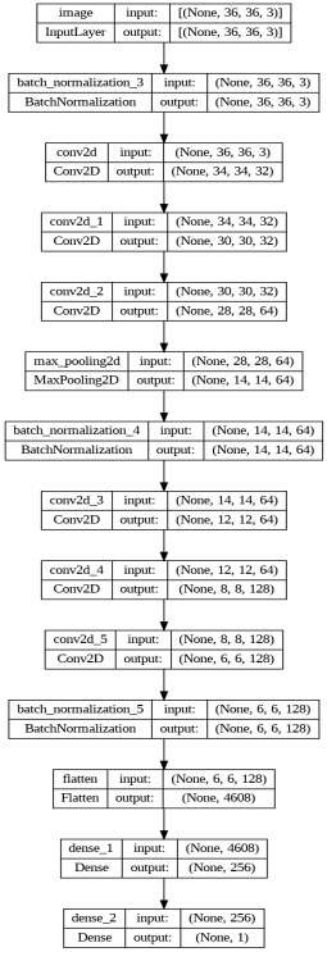
Model 2

Cropping - centered images

Generatore

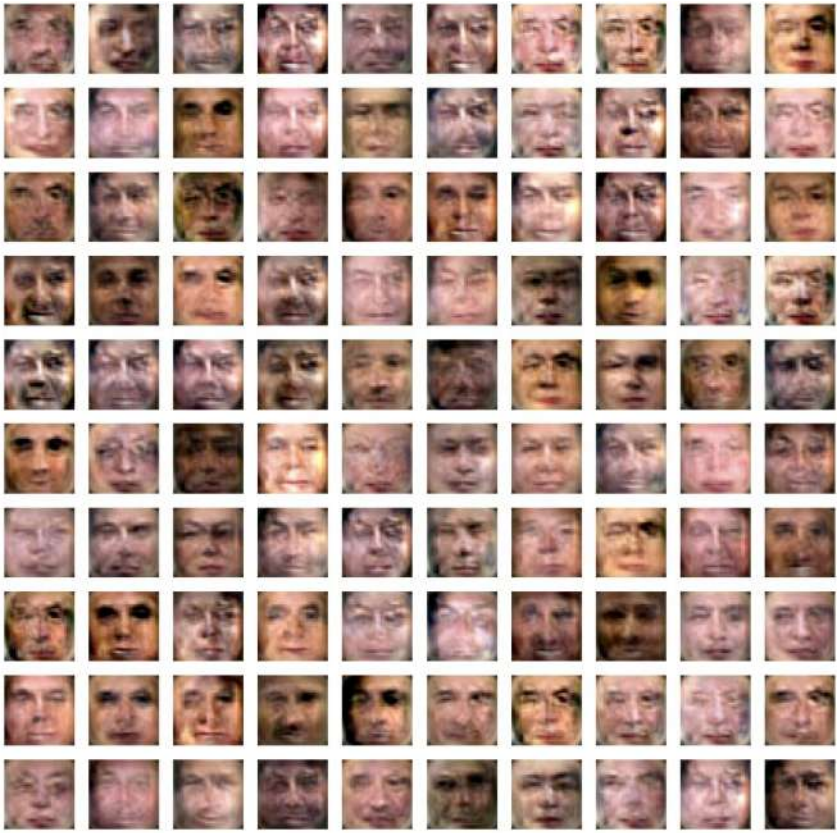


Discriminatore



Quality of images

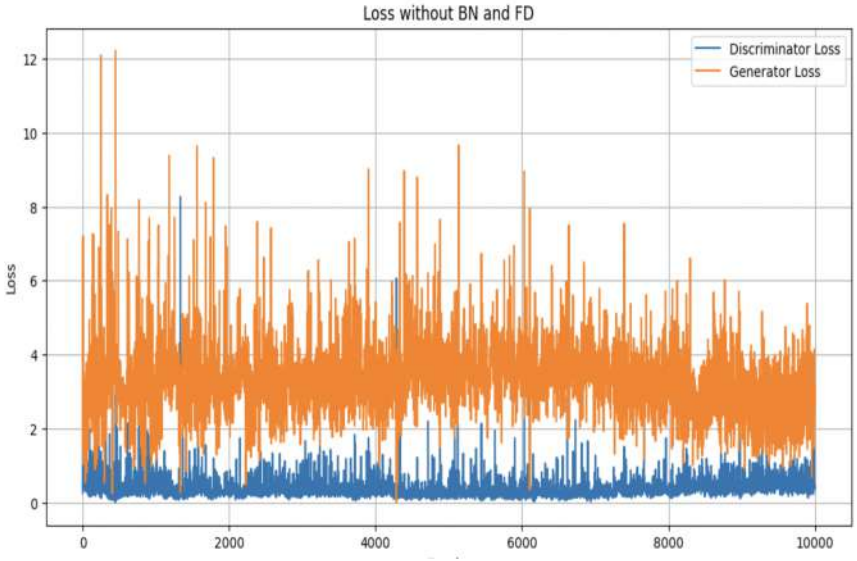
Model 1



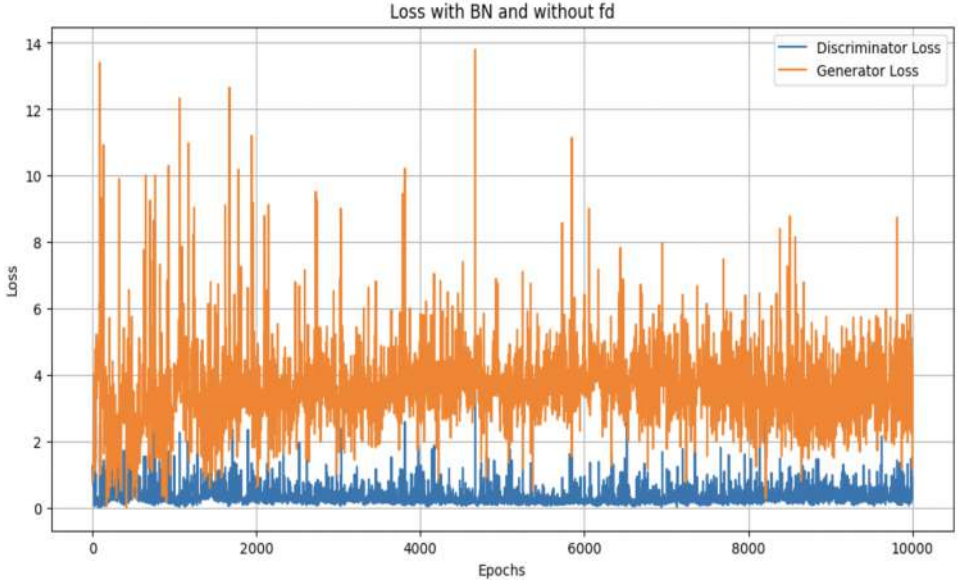
Model 2

Loss

Model 1

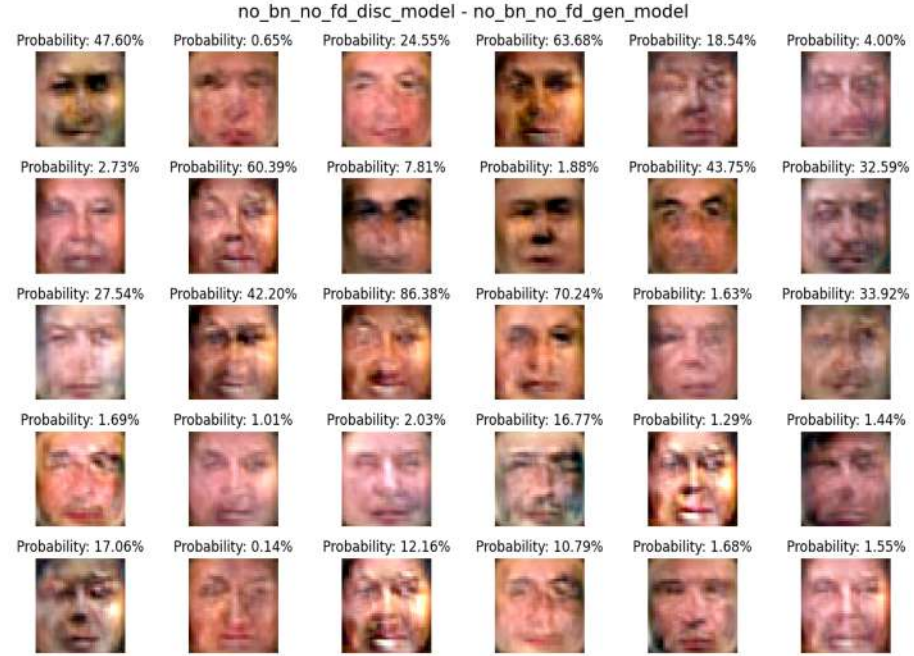


Model 2

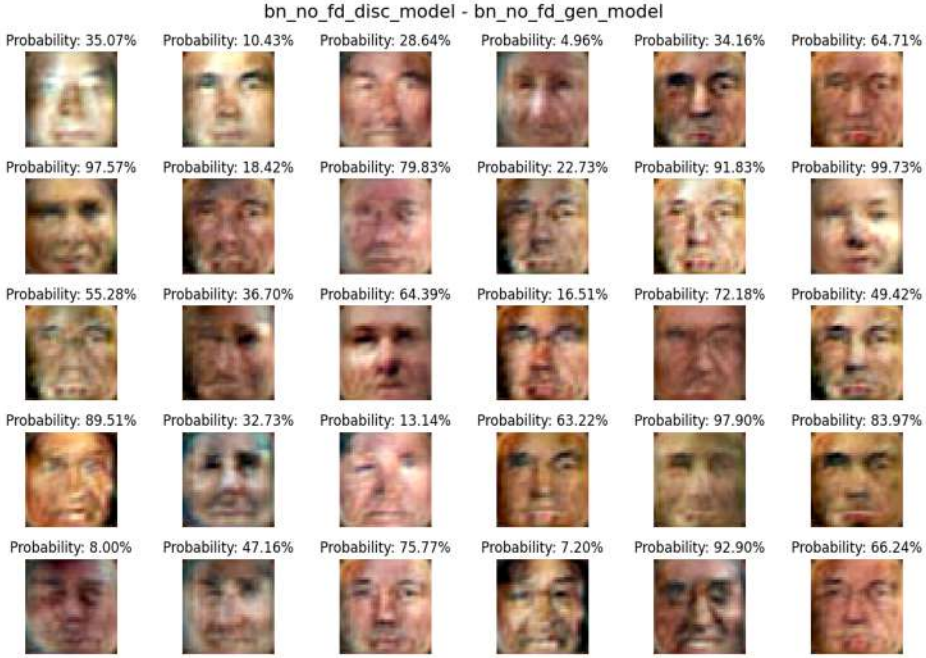


Some predicted images

Model 1



Model 2



Improvements and Next Steps about Audios

Implementation of other models and algorithms

Improving hyperparameters optimization

Applying cross-validation

Improving performances

Incorporating additional data sources

Improvements and Next Steps about Images

Increase the Number of Epochs

Applying Data Augmentation

Train and Test Split

Improving Performances

Thank you.