VISIUM

November 10, 2021

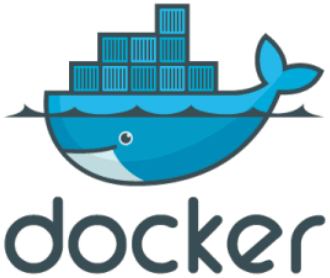
Use case Task

Visium presentation

Deploy a speech emotion recognition Web app using Flask and Docker

Data Scientist: Pier Luigi Segatto

Contact: pier.segatto@gmail.com



Outline

- **Part 1:** Data Setup
- **Part 2:** Features extraction
- **Part 3:** CNN Model
- **Part 4:** API structure
- **Part 5**: Results
- **Part 6:** Conclusions

Data setup

Emo-DB Dataset: 535 emotional utterances spoken by actors

- {'W': 0, 'L': 1, 'E': 2, 'A': 3, 'F': 4, 'T': 5, 'N': 6}
- {0: 'angry', 1: 'boredom', 2: 'disgust', 3: 'fear', 4: 'happy', 5: 'sad', 6: 'neutral'}

Data download: at image creation

Data handling: at first call of \train endpoint:

- Emo-DB data reading and label extraction
  - Emo-DB features extraction
- Data augmentation (if requested, default = True)
  - Augmented data feature extraction
- One-hot encode labels (7 classes)
- Creation of the training and test sets
- Dump to .csv

Data setup - Augmentation

```python
def augment_sample(file, n_modifications=3, dump=True,
                   save_path='data/data_augment/'):
    """Produce n_modified versions of the input audio file. The sample is shifted,
    stretched and white noise is added. Audio files can also be saved.

    As in the reference paper:
        'Our data augmentation techniques include alterations like moving the
        beginning of the sound file by some small amount,speeding up and
        slowing down the file by 1.23% and 0.81% of its normal speed,
        respectively, and adding random noise to the 25% of its length.
    """
    sample, sr = librosa.load(file, sr=None)
    base = os.path.basename(file)[:-4]
    code_to_label = get_maps(from_='code', to_='label')
    label = code_to_label[base[5]]

    new_samples = []

    for i in range(n_modifications):
        # white noise addition
        sample_noise = sample + 0.05*np.random.normal(0, 1, len(sample))

        # time shifting: delay the beginning of the sound
        delay = random.randint(0, 6200)
        sample_shift = np.pad(sample_noise, (delay, 0),
                              mode='constant', constant_values=0)

        # time stretching
        choice = random.random()
        if choice > 0.5:
            factor = 1.23  # fast
        else:
            factor = 0.81  # slow
        sample_stetched = librosa.effects.time_stretch(sample_shift, factor)

        new_samples.append(sample_stetched)

        if dump:
            soundfile.write(
                save_path+base+'aug'+str(i)+'.wav', sample_stetched, sr)

    return new_samples, label
```

Issa et al 2020: Speech emotion recognition with deep convolutional neural networks

Features extraction

```python
def get_features(file):
    """Takes a filename and returns a concatenated array of the features extracted
    using the librosa library"""
    # load audio file
    sample, sr = librosa.load(file, sr=None)
    # calculate the Mel-frequency cepstral coefficients (MFCCs)
    mfcc = np.mean(librosa.feature.mfcc(y=sample, sr=sr, n_mfcc=50).T, axis=0)

    # mel-frequency cepstrum (MFC): short-term power spectrum of a sound
    melspectrogram = np.mean(librosa.feature.melspectrogram(
        y=sample, sr=sr, n_mels=50).T, axis=0)

    # Spectral contrast
    stft = np.abs(librosa.stft(sample))
    spectral_contrast = np.mean(librosa.feature.spectral_contrast(
        S=stft, sr=sr, n_fft=100).T, axis=0)

    # Chromagram
    chroma = librosa.feature.chroma_stft(
        S=stft, sr=sr, n_chroma=80)

    # Tonnetz (tonal centroid features)
    tonnetz = np.mean(librosa.feature.tonnetz(
        y=librosa.effects.harmonic(sample), sr=sr, chroma=chroma).T, axis=0)

    # Averaged chromatogram
    chroma = np.mean(chroma.T, axis=0)

    # concatenate all features to get a 193 long vector
    features = np.concatenate(
        [mfcc, chroma, melspectrogram, spectral_contrast, tonnetz], axis=0)
    features = np.expand_dims(features, axis=0).astype(np.float32)
    return features
```

- Mel-frequency Cepstral Coefficients (MFCCs) -> (50,)
- Mel-scaled spectrogram -> (50,)
- Chromagram -> (80,)
- Spectral contrast feature -> (7,)
- Tonnetz representation -> (6,)

- Concatenation -> INPUT SAMPLE (193,)

CNN Model

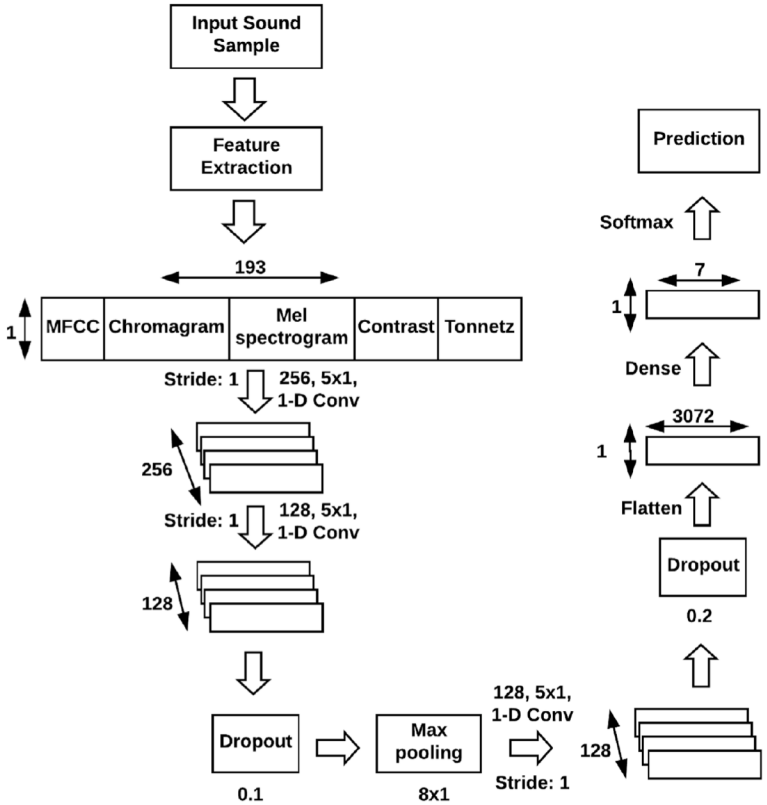


Fig. 4. The topology of the model with 7 classes (Model A).

```python
def network_model():
    """
    CNN model implemented in Dias Issa et al 2020 Speech emotion recognition
    with deep convolutional neural networks (Model A: model with 7 classes).
    """

    # Define architectural params
    n_classes = 7
    stride = 1
    kernel_size = 5

    # New model
    model = Sequential()
    # The first layer of our CNN receives 193 × 1 number arrays as input data
    # The initial layer iscomposed of 256 filters with the kernel size of 5 × 5 and stride 1.
    model.add(Conv1D(filters=256, kernel_size=kernel_size, strides=stride,
                     padding='same', input_shape=(193, 1)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))

    model.add(Conv1D(filters=128, kernel_size=kernel_size, strides=stride,
                     padding='same'))
    model.add(Activation('relu'))

    model.add(Dropout(0.1))
    # model.add(BatchNormalization())

    model.add(MaxPooling1D(pool_size=(8)))

    # model.add(Conv1D(filters=128, kernel_size=kernel_size, strides=stride,
    #                  padding='same'))
    # model.add(Activation('relu'))

    # model.add(Conv1D(filters=128, kernel_size=kernel_size, strides=stride,
    #                  padding='same'))
    # model.add(Activation('relu'))

    model.add(Conv1D(filters=128, kernel_size=kernel_size, strides=stride,
                     padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.2))

    # model.add(Conv1D(filters=128, kernel_size=kernel_size, strides=stride,
    #                  padding='same'))
    # model.add(Activation('relu'))

    model.add(Flatten())
    # model.add(Dropout(0.2))

    # Edit according to target class no.
    model.add(Dense(n_classes))
    # model.add(BatchNormalization())
    model.add(Activation('softmax'))

    model.summary()

    return model
```

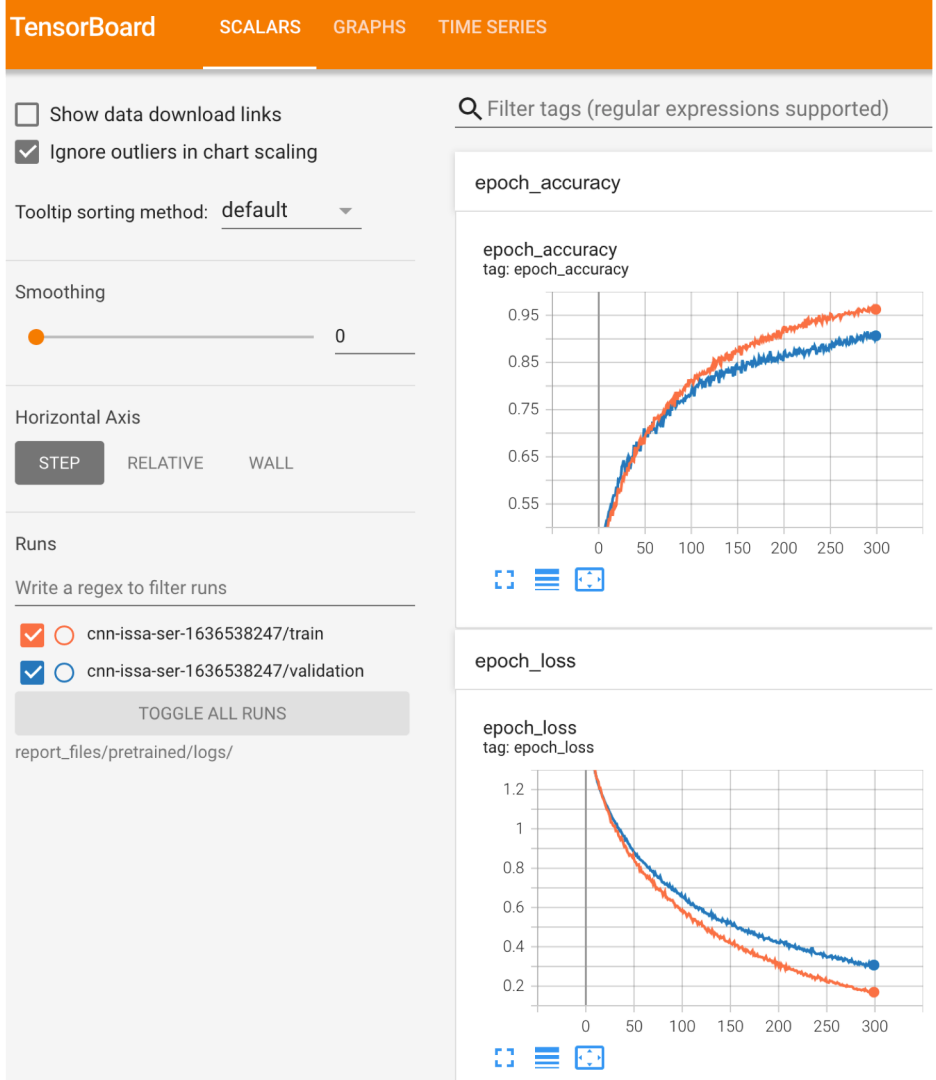
API Structure

!curl "http://localhost:5000/train"

!curl "http://localhost:5000/predict?ID=5"

```python
app = Flask(__name__)

# and the api routes


@app.route("/")
def home():
    return "Visium project: visit /train and /predict endpoints to proceed"


@app.route("/train")
def train_model():
    # User modifiable parameters
    augment = True  # If the dataset has to be augmented
    n_modifications = 3  # how many times each audio will be augmented
    test_size = 0.2  # fraction of the dataset kept for test
    random_state = 1  # random state for reproducibility
    # set true to shuffle the dataset (according to the random state)
    shuffle = True
    force_train = False  # if true when issued a train request the previous model that has been queried will be improved with a new train
    num_epoch = 10  # num of epochs to train
    save_history = True  # set to true to log the loss and accuracy over each epoch
    pretrained = True  # set to true to use the pretrained model I provided

    # create the dataset and augment it if not already present or required
    X, y = get_dataset(augment=augment, n_modifications=n_modifications)
    X_train, X_test, y_train, y_test = create_train_test_sets(
        X, y, test_size=test_size, random_state=random_state, shuffle=shuffle)

    # log sizes
    train_shape = X_train.shape
    test_shape = X_test.shape

    # initialize the model
    model = network_model()
    # Train or get metrics for the requested model
    scores_train, scores_test, history = run_best_model_or_train_model(model, X_train, X_test, y_train, y_test,
                                                                       force_train=force_train, num_epoch=num_epoch, save_history=save_history,
                                                                       pretrained=pretrained)

    return jsonify({'Augmented dataset': augment,
                    'Number of times each audio has been augmented': n_modifications,
                    'Train size': train_shape,
                    'Test size': test_shape,
                    'Used the optimized best model': pretrained,
                    'Training has been forced': force_train,
                    'if training epochs just performed:': num_epoch,
                    f"Training {model.metrics_names[1]}": f"{scores_train[1]*100}%",
                    f"Test {model.metrics_names[1]}": f"{scores_test[1]*100}%"})


@app.route("/predict")
def predict_cnn():
    # User modifiable parameters
    best_pretrained = True

    # get the sample ID requested
    sample = request.args.get('ID')
    if sample is None:
        # display all available filenames and their ID
        files_and_indexes_available = {
            i: el for i, el in enumerate(os.listdir('data/wav'))}
        return jsonify(files_and_indexes_available)

    # get the prediction (or error message if input is wrong format)
    output = predict_from_sample(int(sample), best_pretrained=best_pretrained)

    # if len of output > 1 then no error msg has ben raised
    if len(output) > 1:
        # unpack the output
        file, label, predicted_class, class_probabilities, true_emotion, predicted_emotion = output
        # beautify the probabilities with the emotion labels
        emotions = list(get_maps(from_='label', to_='emotion').values())
        dict_class_probabilities = {i: j for i,
                                    j in zip(emotions, class_probabilities[0])}
        # create message
        message = {"Audio file": file,
                   "True emotion is": true_emotion,
                   "Predicted emotion is": predicted_emotion,
                   "Class probabilities": str(dict_class_probabilities),
                   'Used the optimized best model': best_pretrained}
        return jsonify(message)
    else:
        return jsonify({'Error:': output})
```

Results


TensorBoard
SCALARS
GRAPHS
TIME SERIES
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
cnn-issa-ser-1636538247/train
cnn-issa-ser-1636538247/validation
TOGGLE ALL RUNS
report_files/pretrained/logs/
Filter tags (regular expressions supported)
epoch_accuracy
epoch_accuracy
tag: epoch_accuracy
0.95
0.85
0.75
0.65
0.55
0
50
100
150
200
250
300
epoch_loss
epoch_loss
tag: epoch_loss
1.2
1
0.8
0.6
0.4
0.2
0
50
100
150
200
250
300

Results

```
!curl "http://localhost:5000/predict?ID=5"
```

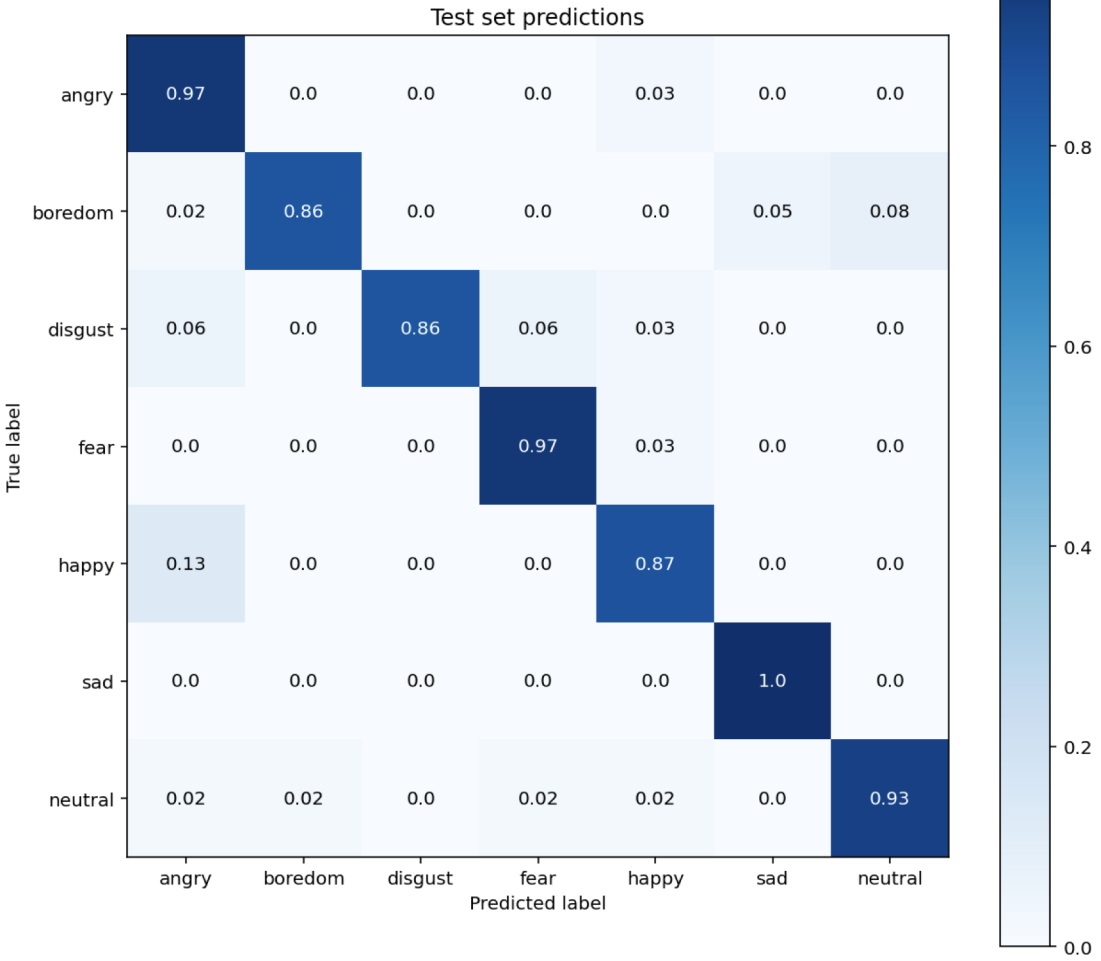

```
{
  "Audio file": "data/wav/14a07Na.wav",
  "Class probabilities": "{'angry': 5.3010273e-11, 'boredom': 5.616377e-05, 'disgust': 6.1022307e-09, 'fear': 3.94809
06e-08, 'happy': 1.3807703e-08, 'sad': 0.007886225, 'neutral': 0.9920575}",
  "Predicted emotion is": "neutral",
  "True emotion is": "neutral",
  "Used the optimized best model": true
}
```


Beware of using the correct argument:

```
!curl "http://localhost:5000/predict?ID=-5"
```


```
{
  "Error:": [
    "Sample must be in range(0,535)"
  ]
}
```

Results


Test set predictions
angry
boredom
disgust
fear
happy
sad
neutral
True label
Predicted label
0.97 0.0 0.0 0.0 0.03 0.0 0.0
0.02 0.86 0.0 0.0 0.0 0.05 0.08
0.06 0.0 0.86 0.06 0.03 0.0 0.0
0.0 0.0 0.0 0.97 0.03 0.0 0.0
0.13 0.0 0.0 0.0 0.87 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.02 0.02 0.0 0.02 0.02 0.0 0.93
1.0
0.8
0.6
0.4
0.2
0.0

Conclusions

- Test Model Accuracy: 91%
- CNN model fully integrated with the Flask web app and served at port 5000
- The Jupyter web server is served at port 8888
- Project dockerized and its container manges both services using supervisord
- Large room of improvement:
  - code (classes, cleanup code, externalize parameters)
  - Api (turning the train endpoint into a post method)

Thank you for the attention