Surgical Mask Detection Documentation

Mihai Stoian

Feature Extraction

The first step in my research was to find out what features are useful for this task.
To extarct the features I used a Python3 module called librosa frequently used for audio feature extraction.

I tested the features from librosa using two models.

Feature	Validation CNN accuracy	Validation FC accuracy
MFCC	0.623	0.64
Spectral bandwidth	0.509	0.527
Spectral centroids		0.526
Spectral contrast	0.583	0.563
Tonnetz	0.527	0.533
Melspectrogram	0.517	0.467
Fourier Tempogram		0.528

As we can see the results are not good. The models learn only from MFCC and Spectral contrast but the accuracy score is low. After reading this article "Sound classification using Images, fastai" and this paper "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification" I found out that they use spectogram images from melspectogram.
The melspectogram is a visual representation of the frequencies that are converted to the Mel scale.

I tested the melspectogram images on a CNN and the results were impressive.

Feature	Validation CNN accuracy
melspectogram images	0.74

After that, I tried more CNN architectures to maximize the test score.

Data	Model	Validation accuracy	Test accuracy
melspectogram images	CNN1	**0.763**	**0.663**
melspectogram images	ResNN1	**0.701**	**0.614**
melspectogram images	CNN1 - AverragPool	**0.678**	
melspectogram images	CNN2 - 3 layers cnn	**0.716**	**0.623**
melspectogram images	CNN3 - cnn1 pool size (3,3)	**0.724**	**0.613**
melspectogram images with more data on validation - 2k validation 7k train	CNN1	**0.702**	**0.581**

I tried to use other spectrogram images like CRF spectrogram and MFCC spectrogram but the results were poor.

Data	Obs
melspectrogram images	good results
Crf spectrogram images	max 0.6 acc on validation
Mfcc spectrogram images	max 0.6 acc on validation
Mfcc raw data	max 0.6 acc on validation
Crf raw data	max 0.6 acc on validation
RGB images on Fold/CRF/MFCC spectograms on R/G/B	poor results
Melspectogram images + Mfcc double input	poor results

Data augmentation

Data augmentation is a great way to add more data to the training set and make the model generalize better and achieve better results as shown in this paper "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification".

I did a bit of research on what augmentation is proper to be used on our data. And I found out that speed change and shift change achieve good results. But pitch change and noise removal lower the model accuracy a lot.
I used the code from this Kaggle Notebook "Sound Augmentation Librosa"

Augmentation techniques	Results in accuracy
Base on non-augmented data	**0.763**
Only random pitch change	0.614
Only random speed change	0.74
Only random shift change	0.765
Preprocessed and removed noise	0.6
Combined audio in 2-sec clips and speed and shift augmentation	0.66

The best architecture

The model

The chosen CNN architecture for the final submission is CNN1 with cross-validation and augmented data.
The CNN1 model is composed of 3 CNN blocks, each block followed by a max-pooling operation.
The Input shape is (221, 223, 1) because we have a grayscale image.
The first CNN block is composed of three CNN layers with 32 filters.
The second CNN block is composed of three CNN layers with 64 filters.
The third CNN block is composed of three CNN layers with 128 filters.

Every Conv layer is followed by a Batch Normalization layer and then by an Activation layer using Relu function.

After the CNN blocks I used a Flatten layer followed by two Dense layers.
The chosen optimizer is Adam.
And because we have binary classification I used binary_crossentropy loss function and Sigmoid activation on the last layer.

The Data

The training data is combined with the validation data resulting in 9000 samples. Then every sample is augmented with random speed change between 0.7 and 1.3, proceeding in 18000 samples. After that every sample is augmented with a random shift proceeding in 36000 samples.

During the training of the folds, I calculated the class weight for each of the fold, because the randomly split data was not evenly balanced every time.
Then the samples are split into 10 folds resulting in 10 models. After that, the predictions are averaged.

The Results

Results on validation

Accuracy recall and confusion matrix scores on every fold.

Fold ID	Accuracy	Recall	True Negative	False Positive	False Negative	True Positive
1	**0.998**	0.980	1764	9	36	1791
2	**0.997**	0.999	1775	9	1	1815
3	**0.993**	0.987	1745	3	24	1828
4	**0.996**	0.996	1767	9	7	1817
5	**0.996**	0.997	1706	7	6	1881
6	**0.970**	0.969	1667	49	58	1826
7	**0.966**	0.969	1736	67	56	1741
8	**0.967**	0.974	1698	73	47	1782
9	**0.967**	0.971	1712	66	53	1769
10	**0.975**	0.981	1666	56	35	1843

Results on test

Results on the joined models with average between all the predictions on test on Kaggle.

Accuracy Public Test	Accuracy Private Test
0.71555	**0.69047**

Models:

CNN1

▼ Click to expand!

```
main_input = Input(shape = (217, 223, 1))
x = Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same')(main_input)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPooling2D((2,2))(x)
```

```
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPooling2D((3,3))(x)

x = Conv2D(128, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(128, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(128, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPooling2D((2,2))(x)
x = Flatten()(x)

x = Dense(128, activation='relu', kernel_initializer='he_uniform')(x)
x = Dense(64, activation='relu', kernel_initializer='he_uniform')(x)
output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=main_input, outputs=output)
model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
```

ResNN1

▼ Click to expand!

```
main_input = Input(shape = (217, 223, 1))
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(main_input)
x = BatchNormalization()(x)
x = MaxPooling2D((3,3))(x)
skip1 = x
x = Activation("relu")(x)

x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
skip2 = concatenate([skip1, x])
x = skip2
x = Activation("relu")(x)

x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
skip3 = concatenate([skip2, x])
x = skip3
x = Activation("relu")(x)

x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
skip4 = concatenate([skip3, x])
x = skip4
```

```python
x = Activation("relu")(x)

x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
skip5 = concatenate([skip4, x])
x = skip5
x = Activation("relu")(x)
x = MaxPooling2D((3,3))(x)
x = Flatten()(x)


x = Dense(128, activation='relu', kernel_initializer='he_uniform')(x)
x = Dense(64, activation='relu', kernel_initializer='he_uniform')(x)
output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=main_input, outputs=output)
model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
```

CNN1 - AverragPool

▼ Click to expand!

```python
main_input = Input(shape = (217, 223, 1))
x = Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same')(main_input)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = AveragePooling2D((2,2))(x)

x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = AveragePooling2D((3,3))(x)

x = Conv2D(128, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(128, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(128, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = AveragePooling2D((2,2))(x)
x = Flatten()(x)

x = Dense(128, activation='relu', kernel_initializer='he_uniform')(x)
x = Dense(64, activation='relu', kernel_initializer='he_uniform')(x)
output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=main_input, outputs=output)
model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
```

CNN2 - 3 layers cnn

▼ Click to expand!

```
main_input = Input(shape = (217, 223, 1))
x = Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same')(main_input)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPooling2D((2,2))
x = Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPooling2D((2,2))
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPooling2D((2,2))
x = Flatten()(x)

x = Dense(128, activation='relu', kernel_initializer='he_uniform')(x)
output = Dense(1, activation='sigmoid')(x)
model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
```

CNN3 - cnn1 pool size (3,3)

▼ Click to expand!

```
main_input = Input(shape = (217, 223, 1))
x = Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same')(main_input)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(32, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPooling2D((3,3))(x)

x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(64, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPooling2D((3,3))(x)

x = Conv2D(128, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(128, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = Conv2D(128, (3, 3), kernel_initializer='he_uniform', padding='same')(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPooling2D((3,3))(x)
x = Flatten()(x)

x = Dense(128, activation='relu', kernel_initializer='he_uniform')(x)
x = Dense(64, activation='relu', kernel_initializer='he_uniform')(x)
output = Dense(1, activation='sigmoid')(x)
```

```
model = Model(inputs=main_input, outputs=output)
model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
```

CNN4

▼ Click to expand!

```
main_input = Input(shape = (217, 223, 1))
x = main_input
# l1
x = BatchNormalization(axis=1)(x)
x = Activation("relu")(x)
x = Conv2D(24, (5, 5), kernel_initializer='he_normal', padding='same')(x)
x = BatchNormalization(axis=1)(x)
x = Activation("relu")(x)
x = MaxPooling2D(pool_size=(4, 2))(x)

# l2
x = BatchNormalization(axis=1)(x)
x = Activation("relu")(x)
x = Conv2D(48, (5, 5), kernel_initializer='he_normal', padding='same')(x)
x = BatchNormalization(axis=1)(x)
x = Activation("relu")(x)
x = MaxPooling2D(pool_size=(4, 2))(x)

# l3
x = BatchNormalization(axis=1)(x)
x = Activation("relu")(x)
x = Conv2D(48, (5, 5), kernel_initializer='he_normal', padding='same')(x)
x = BatchNormalization(axis=1)(x)
x = Activation("relu")(x)

x = Flatten()(x)
#x = Dropout(0.5)(x)
x = Dense(64, kernel_initializer='he_normal', kernel_regularizer=l2(1e-3), activation="relu")(x)
#x = Dropout(0.5)(x)

output = Dense(1, kernel_initializer='he_normal', kernel_regularizer=l2(1e-3), activation="sigmoid")(x)
model = Model(inputs=main_input, outputs=output)
model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
```

CNN4 source: "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification" paper.