Project Documentation

This file contains the documentation for both models I decided to use for this competition, as well as feature extraction and running guides.

I have used various sources of research from the internet while working on this project which are linked either here or in some of the **.py** files right before the parts of code that are inspired from them.

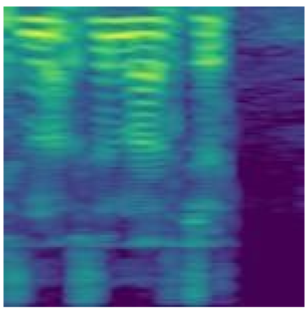
It was a fun competition from which I learned many useful things while trying to improve my models' prediction accuracy.

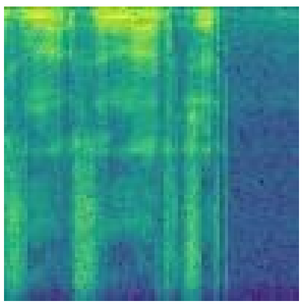
1. Feature extraction

Description

Right from the very beginning when I saw we have to classify audio files I tried to manipulate them in a way that would provide me with a 2d-shaped matrix of features, that I would be able to use a Convolutional Neural Network (CNN) on. I thought that with a good method of feature manipulation I could transform the audio classification problem into an image classification problem, with which I had much more experience.

The first idea that I tried was to extract the spectrograms from the .wav files using **scipy.signal**. However, to my disappointment, it ended up giving pretty poor results (around 64-65% on validation with a basic CNN model), so I kept on researching until I stumbled upon the concept of melspectrograms. Apparently, in a melspectrogram the lower frequencies have more detail added to them, which made the resulting images more textured than the ones obtained with spectrograms (the image on the left is a melspectrogram, while the one on the right is a basic spectrogram, both generated from the same audio sample).





With the same basic CNN model the melspectrogram classification scored around 68-69% on the validation set, so I decided to stick with the melspectrograms (computed using **librosa.feature.melspectrogram**) for the rest of the competition.

I ended up deciding on generating 150x150-sized images for each audio sample, because this way the pictures are square-shaped and they have high enough detail without occupying too much space (I have tried 200x200 but I do not have high enough specs on my computer to store the whole set of images as well as the whole model architecture at the same time on RAM/vRAM).

Relevant files

- **data_formatter.py**
  - **audio_to_images_librosa()** generates 150x150 melspectrograms for the train, validation and test sets to be used by the CNN model.
  - **audio_format_svm()** generates 50x56 melspectrograms for the train, validation and test sets to be used by the SVM model. Due to the SVM requiring single-dimensional features, the melspectrograms are flattened to an array of 2800 features. The melspectrograms used for the SVM model are smaller because using regular 150x150 images would make the model train for too many hours.

2. Convolutional Neural Network model

Description

The first model that I opted for and the one I have also spent most of the time trying to improve is the CNN model. The reason behind it is the experience I had builiding such networks using **PyTorch** at an internship last summer.

At the beginning the model was pretty basic and it had more linear layers than convolutional ones. It had 3 convolutional layers, 3 maxpool layers and 6 linear layers at the end. I played around a little bit with different optimizers and schedulers, but it seemed that no matter what parameters I chose, this model would not perform better than 72-73% on the validation set and 66-67.4% on the public test set.

I decided it was time for a change so I did some research on some popular CNN architectures used for image classification and I decided to try to tweak VGG-16 (officially implemented here in PyTorch) to this task's needs. I ended up deleting some of the convolutional layers because the network was too big, and the final model I ended up working with looks like this:

Number	Layer Applied	Resulting Image Shape
-	-	1 x 150 x 150
1.	**Conv2d**(in_channels = 1, out_channels = 16)	16 x 150 x 150
2.	**Conv2d**(in_channels = 16, out_channels = 64)	64 x 150 x 150
3.	**MaxPool2d**	64 x 75 x 75
4.	**Conv2d**(in_channels = 64, out_channels = 128)	128 x 75 x 75
5.	**Conv2d**(in_channels = 128, out_channels = 128)	128 x 75 x 75
6.	**MaxPool2d**	128 x 37 x 37
7.	**Conv2d**(in_channels = 128, out_channels = 256)	256 x 37 x 37
8.	**Conv2d**(in_channels = 256, out_channels = 256)	256 x 37 x 37
9.	**MaxPool2d**	256 x 18 x 18
10.	**Conv2d**(in_channels = 256, out_channels = 512)	512 x 18 x 18
11.	**Conv2d**(in_channels = 512, out_channels = 512)	512 x 18 x 18
12.	**MaxPool2d**	512 x 9 x 9
13.	**Conv2d**(in_channels = 512, out_channels = 512)	512 x 9 x 9
14.	**Conv2d**(in_channels = 512, out_channels = 512)	512 x 9 x 9
15.	**MaxPool2d** + Flatten the array	8192
16.	**Linear**(in_features = 8192, out_features = 4096)	4096
17.	**Linear**(in_features = 4096, out_features = 2048)	2048

18.	**Linear**(in_features = 2048, out_features = 2)	2

All the convolutional layers have kernel_size = (3, 3), stride = (1, 1) and padding = (1, 1) and all the maxpool layers have kernel_size = (2, 2) and stride = (2, 2).

In addition to that, after every convolutional layer I apply batch normalization and the relu() activation function and after every linear layer I apply relu() as well. At the end, the softmax() function is applied.

I opted against using dropout, because it always provided worse results compared to the normal version on the validation set (0.5% - 1% accuracy loss) and test set (the submissions with dropout had 1% - 1.5% less accuracy).

Before starting training the network, I also normalized the melspectrograms using **torchvision.transforms.Normalize()**.

Hyperparameter choices

Alongside the model, I had to group the melspectrograms in batches and choose a good optimizer and in some cases a good scheduler to obtain good results.

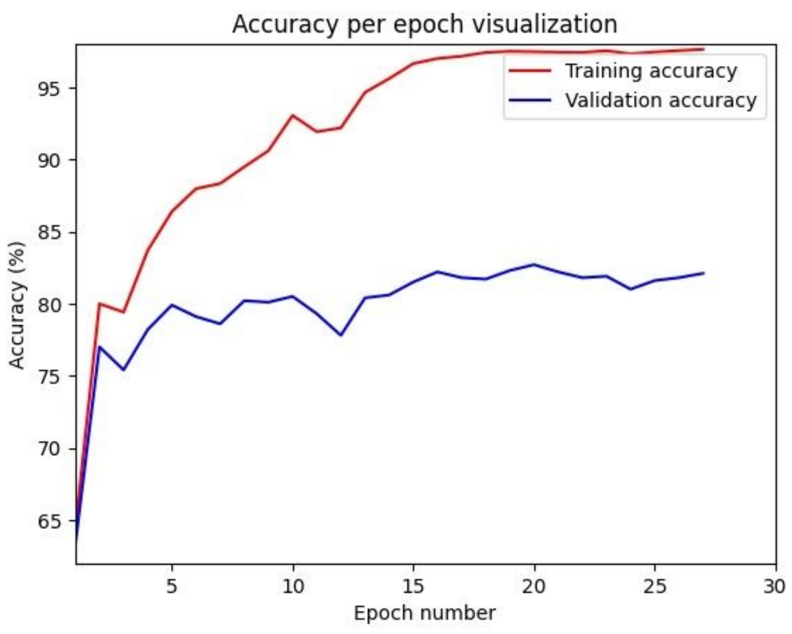
After quite a lot of testing, I have chosen the following hyperparameters as the best:

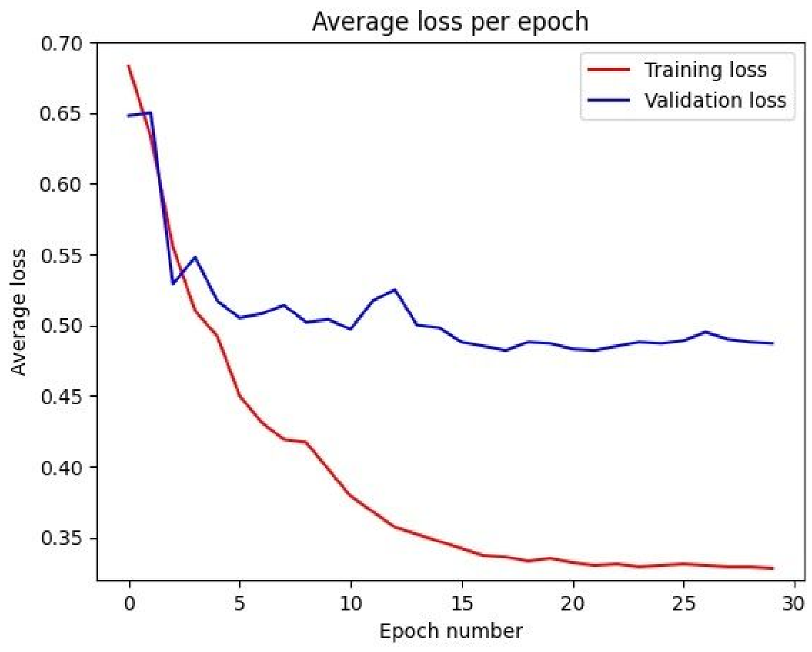
- optimizer : SGD(lr = 0.0008, momentum = 0.9, nesterov = True)
- scheduler : StepLR(step_size = 5, gamma = 0.6)
- loss function: CrossEntropyLoss()

I could not perform all the possible combinations of batch sizes, learning rates, momentums, step sizes and gammas because the network took about 2 hours to train each time.

However, I will provide the results of some testing with different hyperparameters below.

Batch Size	Optimizer (learning rate, momentum, nesterov)	Scheduler (step_size, learning rate)	Validation Accuracy	Validation Average Loss	Test Accuracy (where possible)
4	**SGD(0.0008, 0.9, True)**	**(5, 0.6)**	**82.7%**	**0.482**	**72.6%**
2	SGD(0.0008, 0.9, True)	(5, 0.6)	82.8%	0.475	-
5	SGD(0.0008, 0.9, True)	(5, 0.6)	83.1%	0.472	71%
4	SGD(0.0008, 0, False)	(5, 0.5)	81.6%	0.496	70.4%
4	Adam(0.00001)	-	81.6%	0.491	71.2%
4	SGD(0.0009, 0.8, True)	(5, 0.5)	82.9%	0.482	71.3%
4	SGD(0.0006, 0.9, True)	(5, 0.3)	82.3%	0.486	70.2%
4	SGD(0.0005, 0.9, True)	(5, 0.6)	83.6%	0.476	71.2%
5	Adam(0.0001)	-	78.9%	0.531	69.3%
4	SGD(0.001, 0, False)	(5, 0.4)	81.6%	0.494	70.4%





Confusion Matrix, precision and recall

Confusion Matrix	0 (predicted label)	1 (predicted label)
0 (true label)	374	98
1 (true label)	75	453

	Precision	Recall
0	0.83	0.79
1	0.82	0.85

Running instructions

1. make 3 empty directories called **train_mels_150**, **test_mels_150**, **validation_mels_150** in the same directory as **neural.py**. After extracting the features from the audio files, the data is not only given to the model, but it is also saved locally.
2. run the python file: **python3 neural.py**

3. SVM model

Description

The second model that I opted for is a SVM. Although it doesn't achieve the same result as the other model, I tried to improve it as much as I could.

I decided to implement it using **sklearn.svm.SVC()**.

The data that is being fed to this model is flattened, because the model requires an unidimensional array of features. Therefore, I generated 50x56 images and flattened the array, which resulted in arrays with 2800 features. These arrays are normalized using **sklearn.preprocessing.Normalizer()** before being fed into the SVM model.

Hyperparameter choices

I varied the normalizer norm, the kernel type and the regularization parameter, which led to different results. The best results I have obtained are with norm = 'l2' and an 'rbf' kernel with regularization parameter C = 1.

Normalizer norm	Kernel Type	Regularization Parameter (C)	Validation accuracy
l1	linear	1	53.2%
l2	linear	1	63.3%
l2	**rbf**	**1**	**65.4%**
l2	rbf	2	65.9%
l2	rbf	3	66.9%
l2	rbf	4	66.6%
l2	rbf	5	67.1%
l2	rbf	0.5	64.8%

Even though it might seem that a bigger value of C gives better results, the model actually overfits and behaves worse on the test set.

Confusion Matrix, precision and recall

Confusion Matrix	0 (predicted label)	1 (predicted label)
0 (true label)	304	168
1 (true label)	178	350

	Precision	Recall
0	0.63	0.64
1	0.68	0.66

Running instructions

1. make 3 empty directories called **train_svm**, **test_svm**, **validation_svm** in the same directory as **svm.py**. After extracting the features from the audio files, the data is not only given to the model, but it is also saved locally.
2. run the python file: **python3 svm.py**