ΕΠ08 Pattern Recognition – Machine Learning

3rd Assignment

You must submit **only one IPython notebook file (Jupyter notebook)** named: ***Surname_IdentificationNumber.ipynb***. You can use caption cells to further organize your document. You must make sure that the IPython notebook that you submitted can open and run in the google colab environment. If you read and / or use any material and / or source code that is available on the internet, you have to cite the source and / or the link to the website from which you obtained information correctly. In all cases, copying part of the project or the whole project is unacceptable and in case of suspecting copying, all involved parties will fail the course.

In this assignment, you will deal with predicting music genres from music signals using neural networks. More specifically, the purpose of the project is to classify 1 second of music signal into the following genres: classical music, pop, rock and blues. For each 1 second you are given two sound signal representations: (i) MFCCs and (ii) mel-spectrograms.

The MFCCs are the coefficients of the power spectrum transformed according to the Mel scale, a scale that's close to the way that the human perceives sound signals through hearing. In our case, we use 13 coefficients which are calculated for every 50 msec and therefore a sequence of 20 feature vectors of size 13 is obtained for each piece of music in the dataset. In order to represent this information with a static vector, which is easier to use, we calculate the mean and standard deviation of each of the 13 coefficients from the sequence of the 20 time points. We end up with a 26 feature vector for each piece of music in the dataset.

A spectogram, which is the second way of representation that we are going to use, is a two-dimensional representation that shows the time evolution of the frequency spectrum. If we apply the Mel scale to the spectrogram, the mel-spectrogram or melgram is obtained, which we are going to work with in this project.

Calculating and reversing the time and frequency axes, a 21 (time) x 128 (frequency) matrix is obtained for each element of the dataset.

The data you will use is here and is divided in training (3200 samples), validation (800 samples) and test (1376 samples) sets, which will be used for training, hyperparameter tuning and evaluation of the generalization ability.

Follow the instructions of the questions below and prepare your answers by running your code in Google Colab. The **only** framework that must be used for programming the neural networks is Pytorch.

[Question 1: Feedforward Neural Network] [50 points]

Step 1: Load data (mfccs)

We begin by loaading the mfcc data for training, validation and testing via the corresponding numpy files X.npy and labels.npy. Then, we convert the labels from strings (classical, blues etc.) to integers from 0 to 3, keeping the respective mapping from class names to integers. Finally we load our data into 3 Pytorch dataloaders (one for each dataset) with batch size 16, so that they can be used in our models. Give the argument shuffle = True in the train and validation dataloaders as well.

Step 2: Definition of the Neural Network

Define a class of a fully connected neural network which consists of 4 layers with 26, 128, 32 and 4 neurons respectively, where 26 is the input dimension and 4 is the number of classes to predict.

Step 3: Definition of a training procedure

Define a function that will be responsible for training the network. More specifically, given a number of epochs, an optimizer, a dataloader, a cost function and a neural network, it will pass each batch through the neural network, calculate and print the loss, update the weights and terminate by returning the neural network, when the number of epochs is reached.

Step 4: Definition of a testing procedure

Define a testing function, which will pass all batches of a dataloader through the model and take its predictions without updating the weights. Using the predictions, it will calculate and return (i) the loss, (ii) the f1 macro average, (iii) the accuracy and (iv) the confusion matrix.

Step 5: Neural Network training

Train the neural network in the training set using the following:

- optimizer: stochastic gradient descent
- learning rate: 0.002
- loss function: cross-entropy loss
- number of epochs: 30

Then use the testing function of the previous question to calculate the performance of the trained model in the test set. What is the performance of your model?

Step 6: Neural Network training with GPU

Repeat step 5, but this time start by loading your data and initialized neural network to the GPU provided by Colab. Make sure that the training process is executed on the GPU and print the difference of execution times in GPU and CPU. Make sure that your Colab session includes the use of a GPU – which is free.

Step 7: Model selection

During training (30 epochs) different snapshots of our model are obtained, i.e. models that have different weights. During optimization, we don't know which snapshot of our model has the best generalization ability. This is the reason why we will use the validation set at the end of each epoch to evaluate the model snapshots. Save the model that has the best performance in the validation set according to the f1 metric and use it to measure the performance on the test set. Comment on the results.

For the next questions of the project you must work in the same way using the validation set to find the best snapshot.

[Question 2: Convolutional Neural Network] [50 points]

In this question we will use the mel-spectrograms as an input in Convulutional Neural Networks in order to classify music genres.

Step 1: Load data (spectrograms)

Follow the steps of question 1.1, but for melgrams.
Visualize a random melgram from each class.

Step 2: Definition of the Neural Networks

Define a Convolutional Neural Network which consists of

- A sequence of four conolutional layers with kernel size 5, so that the following channel sequence is achieved: 1, 16, 32, 64, 128
- The output of the last convolutional network is the input of a fully connected neural network of 5 levels with: x (convolutional neural network output dimension), 1024, 256, 32, out_dim neurons.

Step 3: Training of the networks

Make changes where it's is needed and execute the training and testing process, so that the new neural network is trained.

What do you notice? Can the network be trained?

Step 4: Pooling and padding

Embed max pooling with kernel size 2 and padding of size 2 in the convolutional layers. Comment on the use of these two elements. What is the performance of your model?

Step 5: Optimization algorithms

There are various optimization algorithms for a neural network. Try a set of optimizers that are mentioned here and create a table with the accuracy and f1 metric values for each algorithm. Display algorithm names in the columns and the metic values in the rows of the table. What is the difference when it comes to performance?

[Question 3: Improving Performance – optional question] [70 points]

In this question we will try to use Deep Learning techniques and tools to enhance the performance of the Convolutional Neural Network.

Step 1: Reproducibility

We should try multiple different techniques in order to improve the network's performance. To make sure that a technique improves the performance, we must train the network under the exact same conditions. This means that initialization of the weights and the order of the data should be the same every time, so that the performance of the network does not depend on a better initialization point or more favorable management of the data.

For that reason, you have to "seed" all the necessary libraries and algorithms. Consult the following article and make the appropriate changes to your code. Try to run the exact same training process twice. You should achieve the exact same loss in each training epoch and the exact same performances on the test set.

Step 2: Activation functions

The Neural Network that we have made so far, applies only linear transformations on the data. To make the network "learn" more complex relationships in the data, we have to introduce non-linear activation functions.

Try different activation functions from this list, applying them on each convolutional layer (after convolution and before pooling) and on each linear level input. Create a table containing the values of the f1 and accuracy metrics on each line for the activation functions that you tried.

Step 3: Learning rate scheduler

The learning rate is a crucial variable of the optimization algorithm that directly affects the snapshot in which the model is going to end up with. We have been using a static learning rate so far. However the learning rate that we have chosen is not the best choice for all execution steps, as it is probably small for the initial steps in which we wish for a lot of weight adjustments and large for the final steps in which we have approached a good snapshot of the model and don't want to deviate from it.

It is possible to use learning rate schedulers that dynamically adjust the learning rate according to the history of losses. Introduce different schedulers (from here) to your training procedure, giving the argument verbose=True, so that you can monitor the dynamic changes that occur. What is the performance of your model?

Step 4: Batch Normalization

The neural networks function better if the data have specific statistical properties.
We often normalize the input in order to obtain these properties. However, as we use more network levels and as the weights are updated, it is possible that the statistical properties of each layer's input differ from a batch to another. This doesn't help the training algorithm as it tries to learn from data whose the distribution changes between batches. One solution to this problem is to apply normalization to each layer in all elements of a batch so that we obtain the same properties.

Introduce BatchNorm2d layers to your architecture in each convolutional layer before each activation function.

Step 5: Regularization

Try to decrease the difference between train and validation loss by placing and trying different values: (i) weight_decay in the optimizer and (ii) dropout in the linear layers. Increase the number of epochs from 30 to 60 and try (i) and (ii) together and separately. What is the performance of your model on the test set?

Step 6: Training efficiency

Batch size
One training variable that can affect the final performance of the model as well as the training time is batch size. Try the 7 first powers of 2 as batch size and print both the performance and execution time. Comment on your results.

Early stopping
During the experiments of this project we often observe that the best snapshot of the model is obtained much earlier than the number of epochs we have defined. Therefore resuming the training until we reach the final number of epochs makes the process more time-consuming and also results in wasting resources, which is not desirable especially in servers in which lots of users want to have access to the GPUs. For that reason the number of epochs we define should be considered "the maximum number of epochs". Therefore, embed early stopping in your training procedure, namely an ending condition of the training in case the performance of the model on the validation set hasn't improved for a consecutive period (patience) of a number of epochs (e.g. 7). Do you see any difference in the execution time for different values of patience?

[Question 4: Testing – optional question] [30 points]

By evaluating the classifier on the test set we try to get a picture of its ability to generalize to data that hasn't be used during training. To see how close this picture is to reality, we will produce predictions based on data from the real world (YouTube videos).

Step 1: Inference

You will need to write a function that takes a dataset (dataloader with shuffle=False) and a trained convolutional neural network as input and returns a list with the model's predictions.

Step 2: Music download

Install youtube-dl in Colab using the following sequence of commands:

!sudo apt-get update

!sudo curl -L https://yt-dl.org/downloads/latest/youtube-dl -o /usr/local/bin/youtube-dl

!sudo chmod a+rx /usr/local/bin/youtube-dl

The youtube.py file that has been given to you with the data contains functions that, given a YouTube URL, download the sound and compute a sequence of mel spectrograms (one for each second). The youtube_to_melgram function saves the melgram sequence of a given URL to the file melgrams.npy. Use it with at least 1 URL from each music genre that is included to out dataset. You are free to use whichever URL you like. Some examples are given:

- classical music: https://www.youtube.com/watch?v=9E6b3swbnWg
- pop: https://www.youtube.com/watch?v=EDwb9jOVRtU
- rock: https://www.youtube.com/watch?v=OMaycNcPsHI
- blues: https://www.youtube.com/watch?v=l45f28PzfCI

Step 3: Predictions

For each music genre use the function of Step 1 to produce predictions. Print a diagram presenting the music genres on the vertical axis and the timestamps (corresponding to seconds) on the horizontal axis. Do your results make sense compared to the sound of the YouTube videos? Comment on how close the performance of your classifier is to the YouTube videos (in predictions of one second) in comparison to the performance on the test set.