



Relazione progetto ML

Studente:

Fiorito Aldo 1000038099

Professore:

Giovanni Farinella

Corso di laurea magistrale LM-18

Machine Learning

Sommario

Introduzione	3
Problemi affrontati	3
Dataset	3
Requisiti e librerie	6
Metodi e soluzioni	6
Processing	7
Confusion matrix	20
Tabella riepilogo esperimenti	21
Demo	21
Struttura codice	22
Conclusioni	25

Introduzione

La seguente relazione mira a spiegare il lavoro svolto nel progetto del corso di ML, in particolar modo, si vuole evidenziare e risolvere la necessità di dover catalogare, in maniera automatica, un particolare brano secondo il suo genere musicale.

Al giorno d'oggi la musica e le piattaforme di streaming musicale occupano una grande rilevanza nel mondo dei dati. Riuscire a identificarne il genere per una corretta categorizzazione, sponsorizzazione di generi e/o una sistemazione di playlist è un compito da tenere in considerazione. Quando l'autore non rilascia nessuna informazione in merito, è possibile affidarsi al machine learning per risolvere il problema.

Problemi affrontati

Il problema evidenziato dunque è quello di categorizzazione automatica di genere di un brano musicale. Attraverso un sample di una traccia audio possiamo risalire alle sue informazioni intrinseche e provare a indentificarne la tipologia di brano, secondo un pool di classi definite.

Ogni brano, oltre ad essere una traccia sonora, contiene anche una traccia visiva, il suo spettro, ed è proprio da questa sua caratteristica intrinseca su cui verrà basata la risoluzione del problema

Questo tipo di problema è già noto e prende il nome di MGR (music genre recognition)

Dataset

Il dataset usato per addestrare la rete neurale è quello relativo al GTZAN. Esso contiene un sample di 30 secondi estratte da diverse tracce audio collezionate tra il 2000-2001 da varie fonti: personal CDs, radio, registrazioni di microfoni.

Le tracce sono tutte in formato .wav con audio Mono a 16 bit, con una frequenza di campionamento a 22050Hz

Esso contiene, nel totale, 1000 tracce audio, divise in 10 generi per 100 sample ciascuno

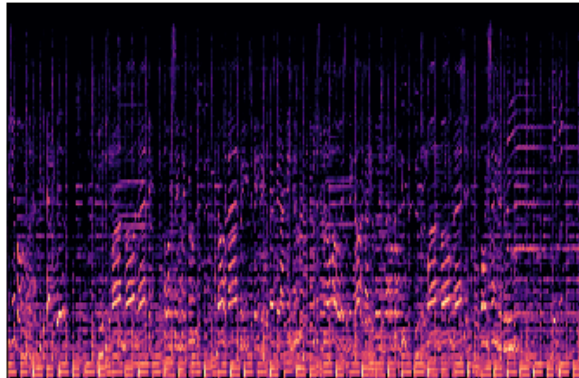
I generi possibili sono :

- blues
- classical
- country
- disco
- hiphop
- jazz
- metal
- pop
- reggae
- rock

Da queste tracce audio viene ottenuto lo spettro di esso, attraverso la libreria “librosa”. In particolar modo, lo spettro ottenuto è della tipologia “Mel Spectrograms”, ovvero l’unione di due parole

- Mel Scale
- Spectrogram

Un sample di immagine è possibile vederlo qui



La dimensione standard di ogni immagine è di 432x288 pixel, 3 canali (RGB)

Ognuna di queste immagini è organizzata all'interno della struttura "images_original", in cui è presente una rappresentazione *classe/immagine.png*

Volume (D:) > ALDO > Studio > Uni > Magistrale > MusicAi > resources > archive > Data > images_original >				
Nome	Ultima modifica	Tipo	Dimensione	
blues	14/06/2023 18:26	Cartella di file		
classical	14/06/2023 18:26	Cartella di file		
country	14/06/2023 18:26	Cartella di file		
disco	14/06/2023 18:26	Cartella di file		
hiphop	14/06/2023 18:26	Cartella di file		
jazz	14/06/2023 18:26	Cartella di file		
metal	14/06/2023 18:26	Cartella di file		
pop	14/06/2023 18:26	Cartella di file		
reggae	14/06/2023 18:26	Cartella di file		
rock	14/06/2023 18:26	Cartella di file		

Una volta ottenuto il dataset completo, pensiamo ad una fase di preprocessing e di trasformazione del dataset che vedremo nella parte di Metodi

Requisiti e librerie

- Python3.9.8
- Pickle (data serialization)
- Sklearn standard
- matplotlib
- Sklearn extra
- Pands
- Numpy
- Scipy
- Plotly
- Treys evaluator poker

Metodi e soluzioni

Prima di tutto, per poter eseguire tutti i passi necessari, è necessario scaricare la cartella Data, contenente sia i file audio per classe, sia le immagini pre elaborate e/o modificate.

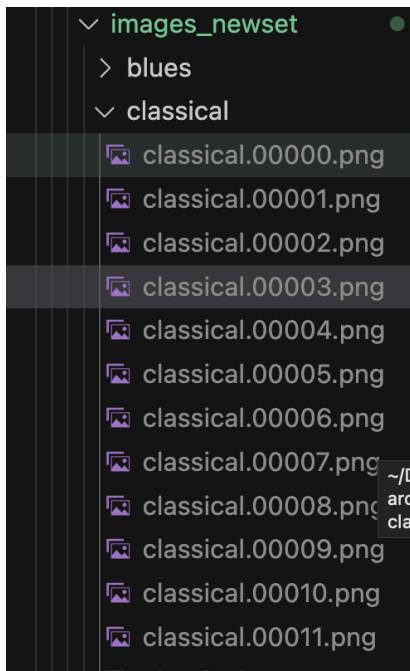
https://drive.google.com/file/d/1oaoP5ahdcPS_e3zoK8UR5JiB_DMTsXEI/view?usp=sharing

Le subfolder, all'interno di questo zip, devono essere estratte all'interno della folder Data, presente dentro "resources/archive"

Nel dataset messo a disposizione da GZTAN, è possibile usare delle immagini pre-elaborate, queste sono ottenute usando librosa e analizzando lo spettrogramma.

Al fine della corretta valutazione e del corretto training si è scelto di processare nuovamente i file audio e ottenere lo spettrogramma del file audio come immagine.

Attraverso lo script python, "*create_image_set*", creiamo la folder images_newset.



Qui invece sono disponibili i modelli pre-allenati uscenti dalla fase di learning

<https://drive.google.com/file/d/1lo6IX0dJzHPylxa6ArZN9mh-xe5CA4xl/view?usp=sharing>

Essi, da qui è possibile estrarre i file “.pth” relativi ai due modelli discussi in questa relazione.

I file pth dovranno essere inseriti su “archive/stored/models”

▼ models	oggi, 11:18
leNet.pth	16 luglio 2023, 16:51
miniAlex.pth	16 luglio 2023, 15:29

Processing

E' possibile passare alla fase di training della CNN attraverso il flag “doPreprocess”.

La prima fase del nostro processing parte da una serie di trasformazioni alle immagini di input per poterle usare nella nostra rete neurale CNN custom.

I passi necessari sono:

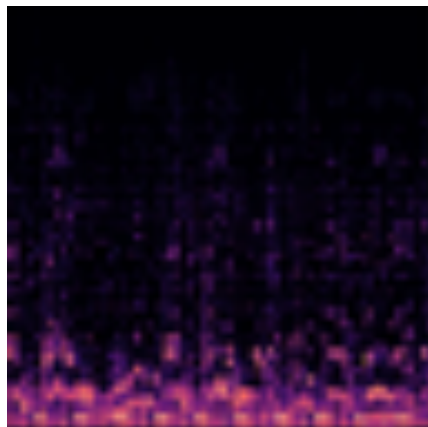
- Trim (per eliminare il bordo bianco nelle foto)
- Resize (per ridurre la dimensione della foto per diminuire la complessità dei parametri)

```
im_trimmed = trim(img)
new_image = im_trimmed.resize((size, size))
new_image.save(transformedData+"/"+category+"/"+file)
im_ts = torch.from_numpy(np.array(new_image))
```

Fatto ciò, abbiamo ottenuto la nostra nuova folder, dove le immagini avranno subito le nostre trasformazioni

me (D:) > ALDO > Studio > Uni > Magistrale > MusicAi > resources > archive > Data > images_transformed_64 >

Nome	Data	Tipo	Dimensione	Tag
blues	18/06/2023 12:26	Cartella di file		
classical	18/06/2023 12:26	Cartella di file		
country	18/06/2023 12:26	Cartella di file		
disco	18/06/2023 12:26	Cartella di file		
hiphop	18/06/2023 12:26	Cartella di file		
jazz	18/06/2023 12:26	Cartella di file		
metal	18/06/2023 12:26	Cartella di file		
pop	18/06/2023 12:26	Cartella di file		
reggae	18/06/2023 12:26	Cartella di file		
rock	18/06/2023 12:26	Cartella di file		



Prima di procedere allo splitting del dataset, ci assicuriamo che tutte le immagini siano normalizzate.

Per fare questo, collezioniamo le immagini in un dataset

```
dataset =  
ScenesDataset('./resources/archive/Data/images_transformed_'+str(size), './resources  
/archive/Data/songs_train_test/fullDs.txt', transform=transforms.ToTensor())
```

```
m = np.zeros(3)  
print(m)  
for sample in dataset:  
    m+=sample['image'].sum(1).sum(1).numpy()  
  
m=m/(len(dataset)*size*size)  
  
s = np.zeros(3)  
for sample in dataset:  
    s+=((sample['image']-torch.Tensor(m).view(3,1,1))*2).sum(1).sum(1).numpy()  
  
s=np.sqrt(s/(len(dataset)*size*size))
```

```
Medie [0.1489925  0.05465949 0.17116481]  
Dev.Std. [0.20587373 0.08150823 0.17132577]
```

Queste due misure verranno inserite successivamente nelle trasformazioni da compiere, prima di computare le immagini.

```
transform = transforms.Compose([transforms.ToTensor(),  
                                transforms.Normalize(m,s),  
                                ])
```

A questo punto, siamo pronti per suddividere il dataset in training_set e test_set.

Questo viene fatto, secondo le percentuali di 0.8% e 0.1%

```
splitfolders.ratio("./resources/archive/Data/images_transformed_"+str(size),  
output="./resources/archive/Data/",seed=1337, ratio=(.8, .1, .1),  
group_prefix=None, move=False) # default values
```

```
dataset_train =
ImagesDataset('./resources/archive/Data/train','./resources/archive/Data/songs_train_test/train_loader.txt',transform=transform)

dataset_test =
ImagesDataset('./resources/archive/Data/val','./resources/archive/Data/songs_train_test/val_loader.txt',transform=transform)
```

All'interno, quando verranno caricate le immagini, esse verranno trasformate in tensori pytorch, poiché seguiranno il processo di transform dei dati.

Siamo pronti adesso a costruire le nostre CNN.

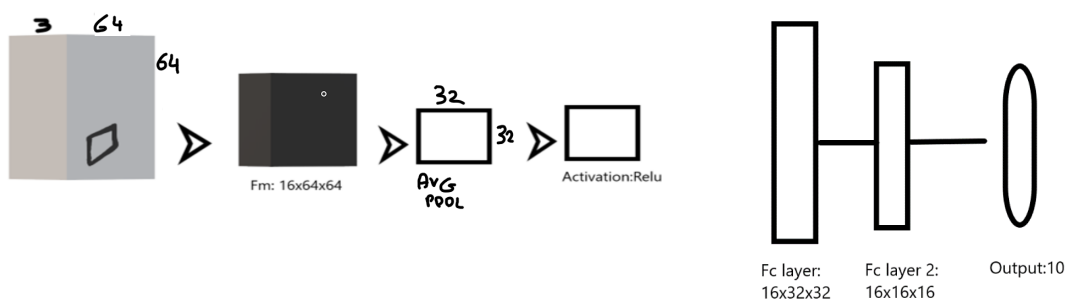
Prima di procedere con questa rete profonda, cerchiamo di costruire qualcosa di semplice e immediato.

Innanzitutto, carichiamo i nostri dati in un'altra struttura di PyTorch, in cui ci permette di definire i batch size da utilizzare durante la fase di training

```
train_dataset = DataLoader(dataset_train,batch_size=32,num_workers=0,shuffle=True)
test_dataset = DataLoader(dataset_test,batch_size=32,num_workers=0,shuffle=2)
```

Iniziamo con un batch size piccolo di 32 samples per epoca

La nostra prima rete è così composta



```
lenetModel = LeNetColor(sizeInput=size,outChannels=16)
```

Possiamo è possibile la nostra rete neurale. A causa di mancanza di GPU, verrà usata la modalità device=CPU. Come funzione di Loss useremo la funzione legata al softmax, usata per la classificazione e, per aggiornare i parametri, andremo ad aggiornarli attraverso la discesa del gradiente stocastica, adatta ai mini batch.

```
criterion = nn.CrossEntropyLoss()  
optimizer = SGD(model.parameters(), lr, momentum=momentum)
```

Andiamo a vedere le curve di training e test, a paragone di loss e accuracy.

Il primo esperimento che andremo a fare verrà effettuato con

```
epochs = 200,lr=0.001,momentum=0.5
```

Per la visualizzazione useremo il tool integrato di “tensorboard”

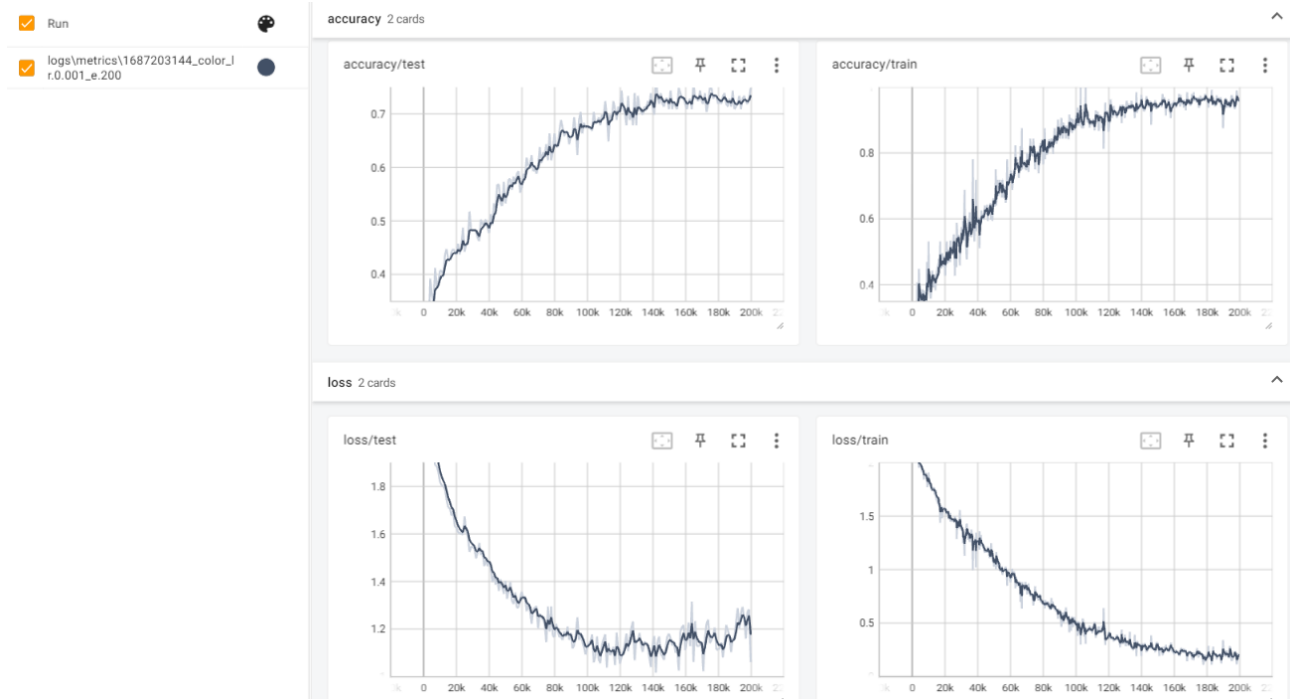
Il primo risultato ci mostra un accuracy non troppo malvagia sul test set, accettabile. Il 95% tuttavia ci fa presagire un pelo di overfitting.

Ma vediamo le curve di loss/train

La discesa del learning rate è abbastanza smooth, e segue un andamento mediamente buono.

Tuttavia, la loss nel test set, dopo un primo andamento in cui sembrava stabilizzarsi, è aumentata.

```
Accuracy train LeNetColor: 0.95  
Accuracy test LeNetColor: 0.73
```



Probabilmente è dovuto al fatto che è presente un leggero overfitting, vediamo quali tecniche possono aiutarci a migliorare i risultati.

Tra le possibili soluzioni all'overfitting possiamo trovare 3 utili metodi distinti.

JitterCasuale,

Dalla famiglia della Data Augmentation, cerchiamo di manipolare le immagini che daremo in pasto alla CNN.

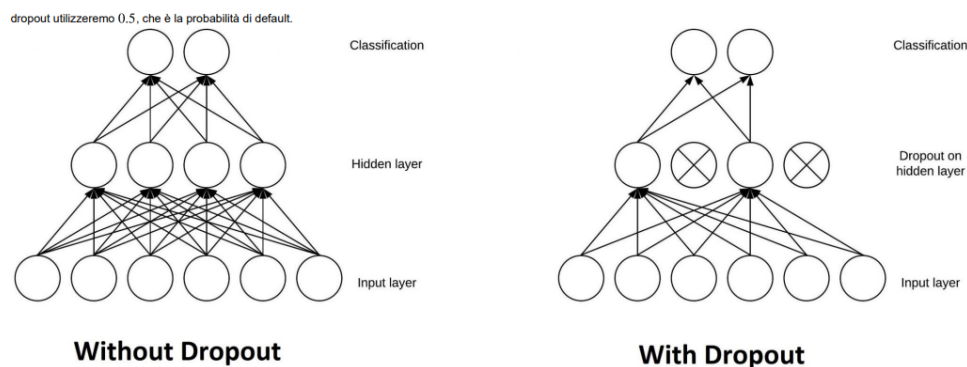
Questo è necessario affinché la CNN generalizzi maggiormente gli esempi durante la fase di training, a beneficio della classificazione di immagini mai viste e/o diverse maggiormente. Le manipolazioni occorreranno con una probabilità dello 0.33% e andranno a trasformare l'immagine nel suo complesso secondo la funzione *ColorJitter* fornita da pytorch

```
transform =  
transforms.ColorJitter(brightness=(0.5,1.5),contrast=(1),saturation=(0.5,1.5),hue=(  
-0.1,0.1))
```

All'interno bisogna specificare il range del cambiamento da effettuare e quali caratteristiche dell'immagine cambiare

Dropout, in cui rimuoviamo dei nodi durante la fase di training, con una probabilità dello 0.50% seguendo una distribuzione di bernoulli

`p: probability of an element to be zeroed. Default: 0.5`

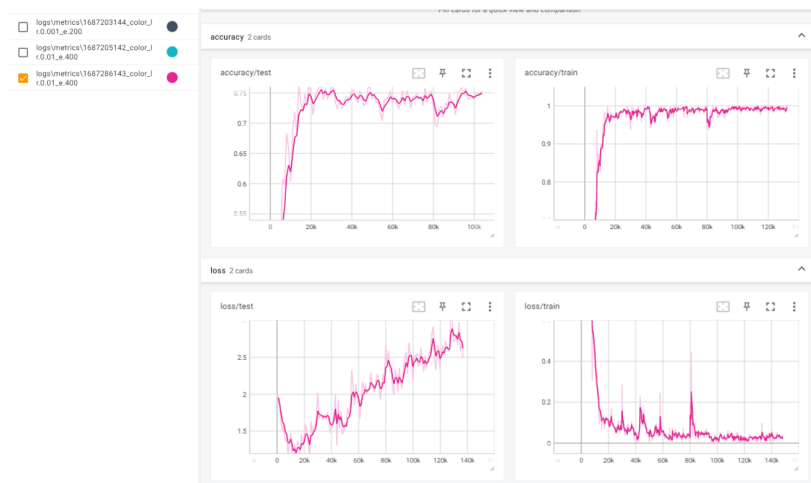


Introduciamo un dropout prima del nostro fully connected layer e vediamo come cambiano le loss e l'accuracy

```
self.classifier = nn.Sequential(  
    nn.Dropout(),  
    nn.Linear(outChannels * 32 * 32, outChannels * 16 * 16),  
    nn.ReLU(),  
    nn.Linear(outChannels * 16 * 16, 10),  
)
```

L'applicazione del Dropout con una probabilità del 0.50% si è rivelata un peggioramento, come è possibile vedere dalla loss e dall'aumento dell'accuracy nel training set.

Accuracy train LeNetColor: 0.99
Accuracy test LeNetColor: 0.70



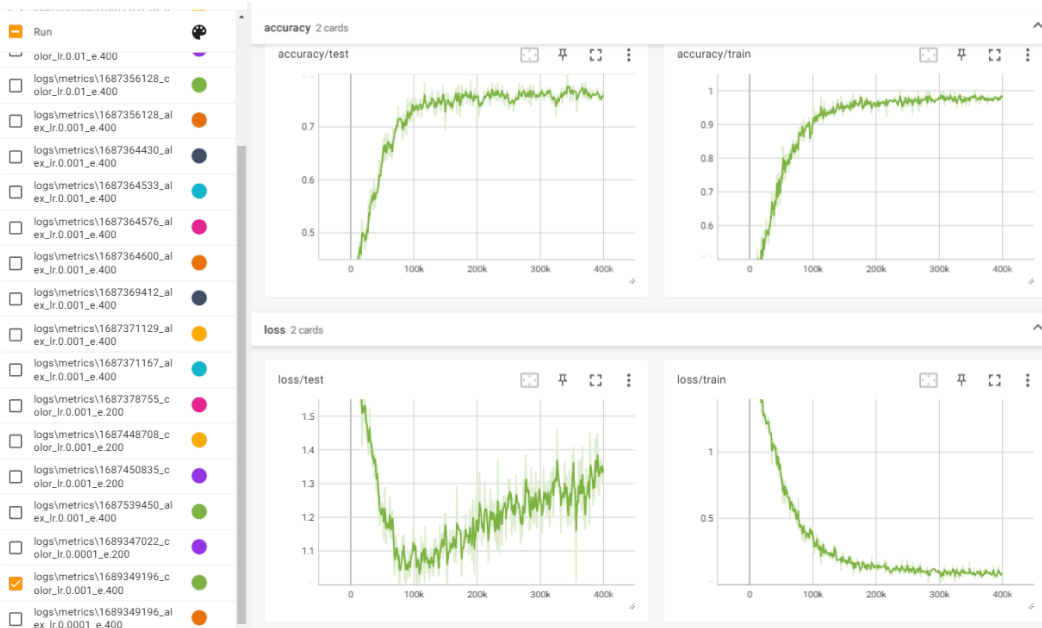
Con una dropout del 0.20%

```
Accuracy train LeNetColor: 1.00
Accuracy test LeNetColor: 0.75
```



Dividendo per 10 l'attuale lr, riportando il droprate al 0.5 non si ha avuto nessun miglioramento significativo.

Con un batch-rate raddoppiato e un lr riportato a 0.001..



BatchNormalization

La B.N è una tecnica per ridurre la distribuzione dei dati durante la fase di training e rendere meno variabile la covarianza di essi. In poche parole, usando la media e la deviazione standard. Nella nostra rete iniziale questa può essere piazzata prima di un blocco convoluzionale (tranne nel primo) e prima del blocco fully connected.

Dunque, applicando alla rete la funzione “*BatchNorm1d*”

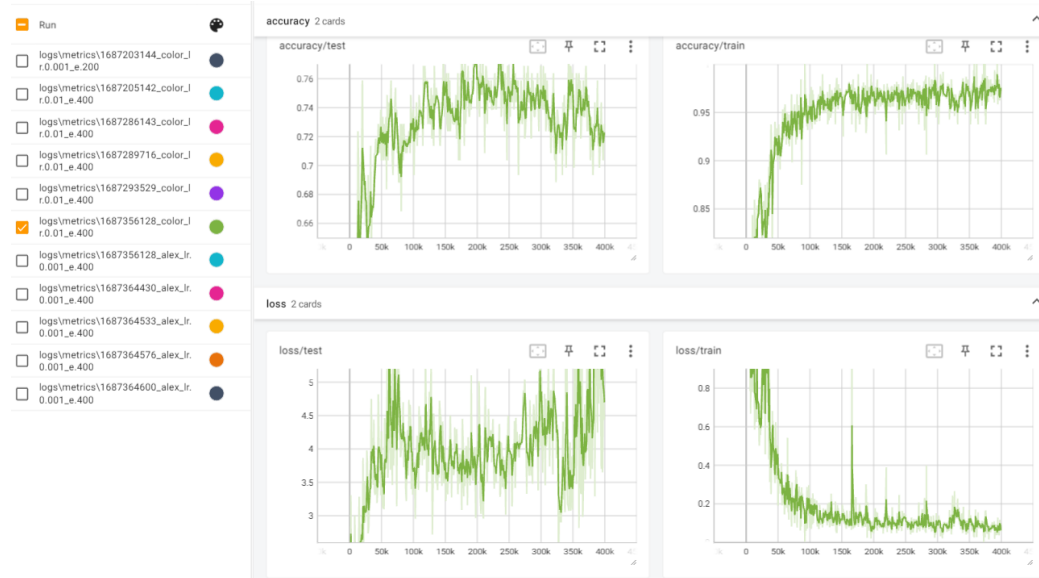
Piazziamo la normalization dopo il blocco del dropout nel fully connected 1

```
nn.Dropout(p=0.50),
nn.BatchNorm1d(outChannels * 32 * 32),
nn.Linear(outChannels * 32 * 32, outChannels * 16 * 16),
```

E vedendo l’accuracy, abbiamo qualche punto percentuale in meno nel training e sia nel dataset di test.

```
Accuracy train LeNetColor: 0.96
Accuracy test LeNetColor: 0.71
```

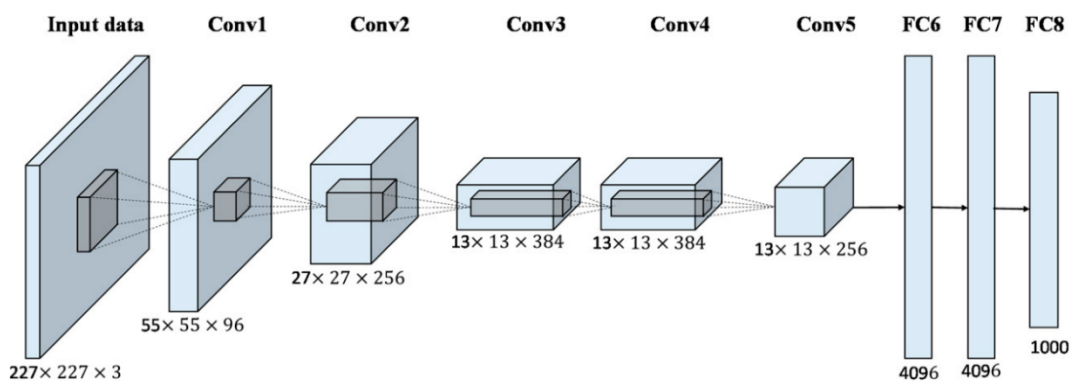
Tuttavia, il grafico della loss presenta degli spike non rassicuranti, presenti anche prima quando abbiamo applicato la fase di dropout con $p=0.20\%$

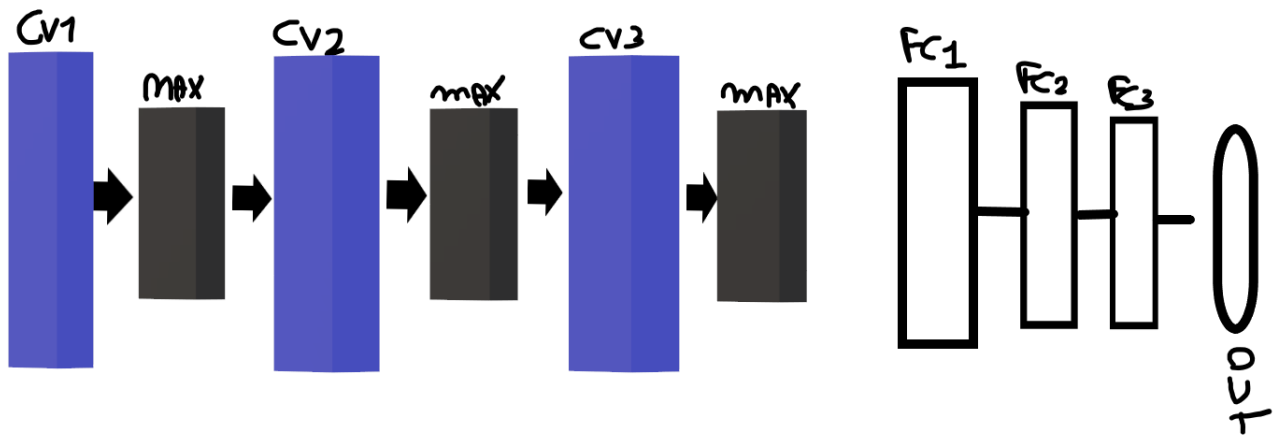


Forse, dunque, conviene validare dal 1° modello proposto in cui il grafico non presentava moltissimi spike

Cosa succede rendendo più profonda la nostra rete?

Per questo task ci ispireremo a una struttura nota, chiamata “AlexNet”, semplificando un po’ quest’ultima e creeremo una CNN da zero usando pytorch, cercando di combinare le tecniche viste fino adesso, cercando di minimizzare gli spike e migliorando nel totale la CNN.



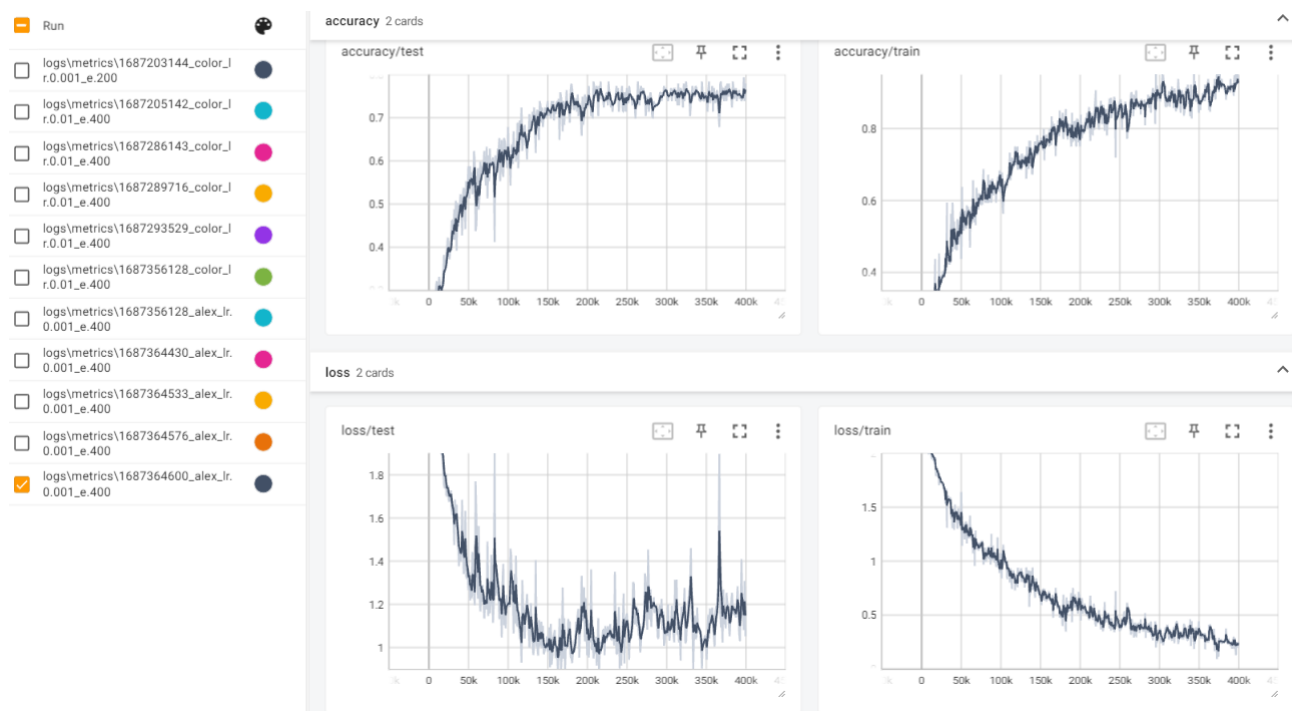


Applicheremo dunque 3 livelli convoluzionali con un maxPool piazzato prima della funzione relu. E per quanto riguarda i livelli full connected, applicheremo 3 volte il dropout e di seguito una batchNormalization1d.

La fase di training sarà composta da 400 epoche, con un lr pari a 0.001 e un momentum del 0.90

Allenando la nuova rete proposta i risultati vengono mostrati sotto.

```
Accuracy train MiniAlexNet: 0.96
Accuracy test MiniAlexNet: 0.76
```



La discesa della loss nel train è accettabile, andando a convergenza in maniera simile al primo modello, così come l'incremento dell'accuracy durante la fase di test e training.

Unica pecca la loss durante la fase di test, forse segno di un modello che su alcuni test

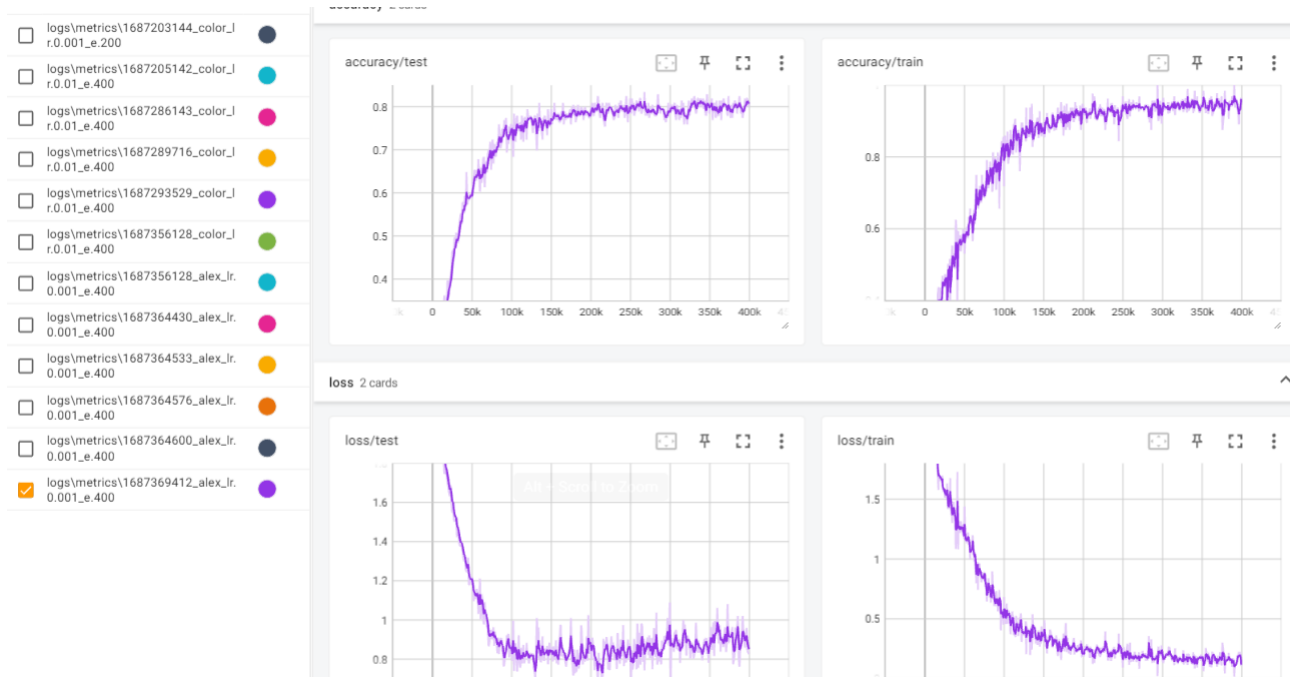
Proviamo ad inserire ulteriori step di normalization. Questa volta andremo a posizionarli durante la fase delle convoluzioni e dunque prima delle *Conv2d*. Il passo di normalization questa volta sarà 2d piuttosto che 1d

```
nn.BatchNorm2d(int(outChannels)),  
nn.Conv2d(int(outChannels),int(outChannels)*2,kernel_size=2,stride=1,padding=1),  
nn.MaxPool2d(2),  
nn.ReLU(),
```

Dopo la fase di training otteniamo un accuracy di test mai vista fino adesso e la loss sul test è abbastanza stabile

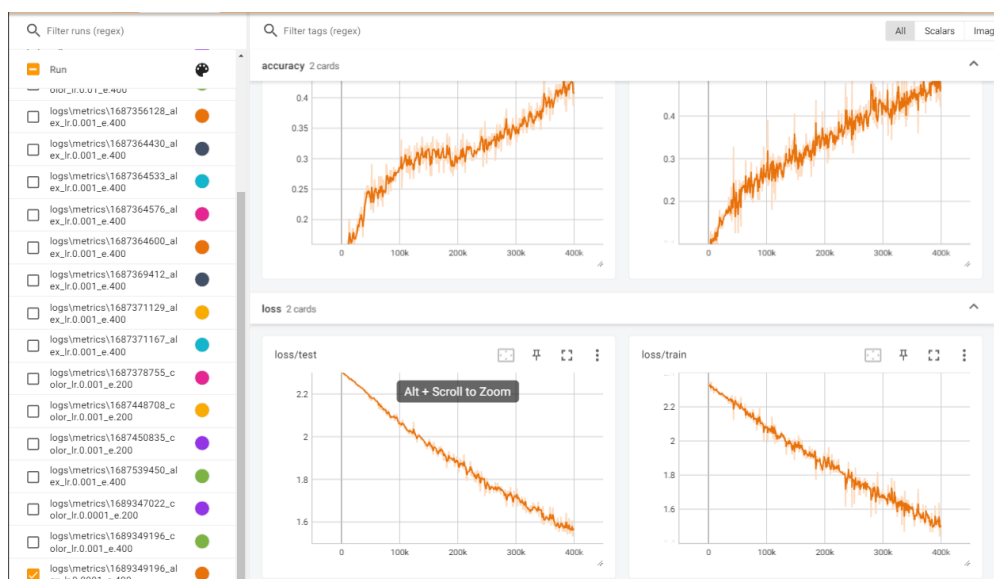
```
Accuracy train MiniAlexNet: 0.99  
Accuracy test MiniAlexNet: 0.81
```

Dunque, una rete maggiormente profonda a fatto in modo di cogliere maggiormente le particolarità di immagini mai viste.



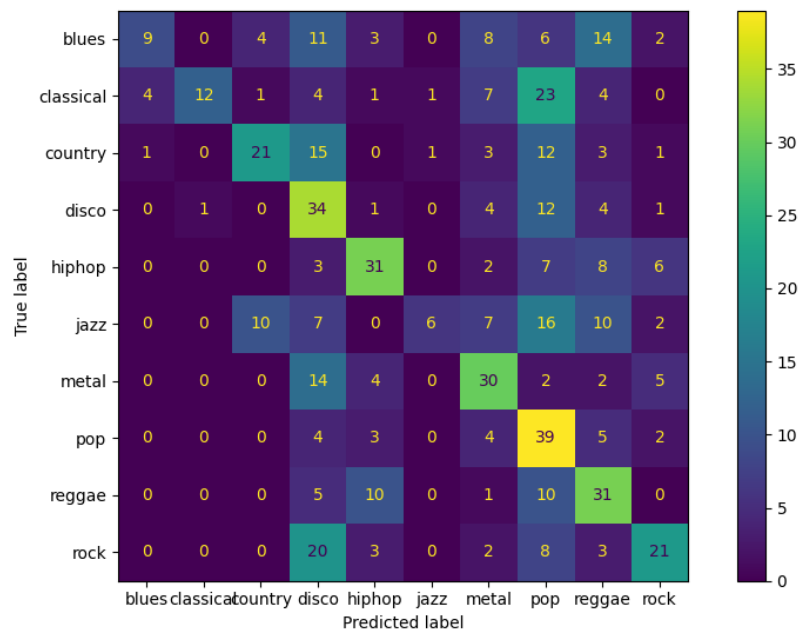
Un altro tentativo con questa CNN è stato provato, tentando di raddoppiare gli input channel, durante le convoluzioni, ma si è ottenuto nuovamente una situazione di overfitting.

Inoltre, raddoppiando la batch size e dividendo anche qui il learning rate i risultati non sono andati nel verso giusto



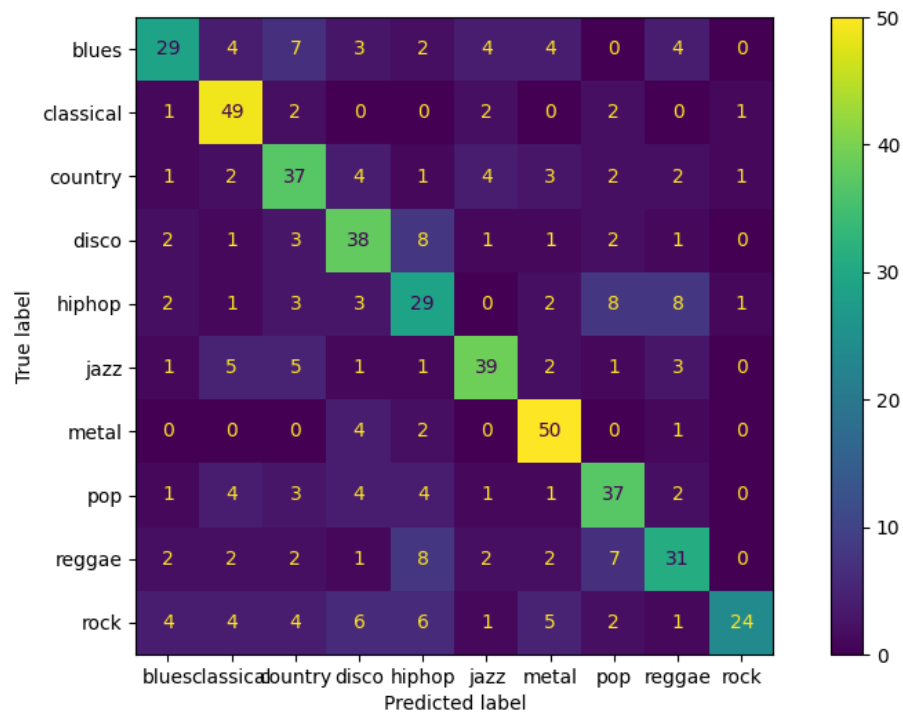
Confusion matrix

Plottando la CM del modello alex ottenuto, notiamo uno sbilanciamento verso la classe 'pop'



Proviamo ad allenare la CNN andando a ridurre i dati di TR dalla classe pop.

Tutti i valori della colonna pop, classificati erroneamente come tale si sono distribuiti nella diagonale principale.



La cm è stata valutata usando il modello LeNetColor

Tabella riepilogo esperimenti

Network	BatchSize	epochs	learning rate	momentus	BatchN1d	BatchN2d	jitter	Dropout	AC Train	AC Test
LeNetColor	32	200	0.001	0.5	x	x	x	0.0	0.95	0.73
LeNetColor	32	200	0.001	0.5	x	x	v	0.5	0.99	0.70
LeNetColor	32	200	0.001	0.5	x	x	v	0.2	1.00	0.75
LeNetColor	32	200	0.0001	0.9	x	x	v	0.5	0.84	0.66
LeNetColor	32	200	0.001	0.5	v	x	v	0.5	0.96	0.71
LeNetColor	64	400	0.001	0.5	v	x	v	0.5	0.99	0.76
MiniAlexNet	32	400	0.001	0.9	v	x	v	0.5	0.96	0.76
MiniAlexNet	32	400	0.001	0.9	v	v	v	0.5	0.99	0.81
MiniAlexNet	64	400	0.0001	0.9	v	v	v	0.5	0.43	0.42




Demo

Un video demo della interfaccia è possibile trovarlo al seguente link drive

<https://drive.google.com/file/d/13UZX0Jf31kZ1ZvI7sDCJv8ub-ns5W-vH/view?usp=sharing>

Struttura codice

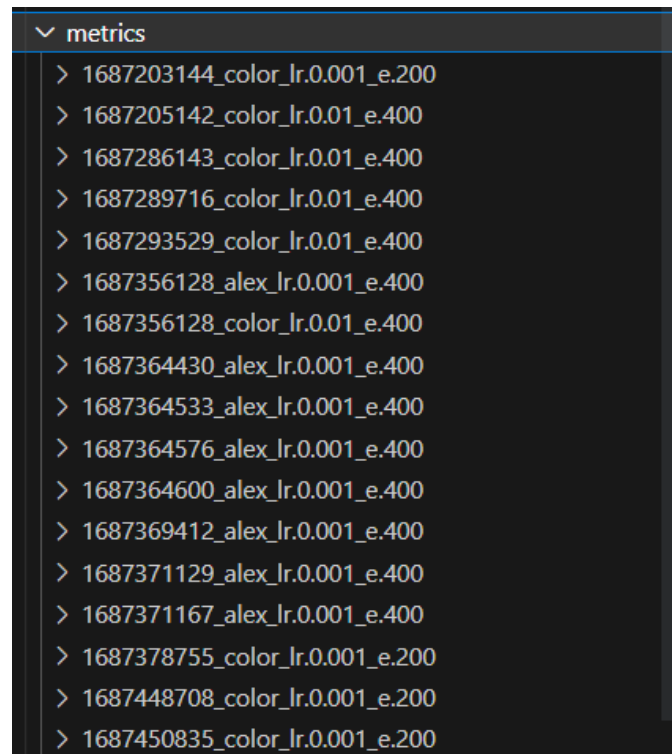
Il progetto, chiamato *MusicAi* è strutturato in 3 cartelle principali più 2 main, uno da cui far partire la creazione del modello insieme al preprocessing delle immagini e un altro *main_interface* legato all'interfaccia utente.

 bin	14/06/2023 18:34	Cartella di file
 logs	15/06/2023 14:25	Cartella di file
 resources	18/06/2023 16:15	Cartella di file



Dentro *bin* possiamo trovare

1. *Loaders.py*, dentro vi è definita la classe che si occuperà di contenere il nostro dataset di immagini. Questa classe è ereditata dalla classe *dataset* di *pytorch*. La classe implementa i metodi privati di *getItem* e *len*. Il metodo *getItem* si occupa di applicare le trasformazioni definite quando viene caricata un'immagine all'interno di esso.
2. *Metrics_eval.py*, all'interno vengono definite classi e metodi utili alla valutazione e training del modello, sia a livello di training che a livello di testing. In particolare, il metodo più importante è la parte riferita al training, in cui vengono usate le funzioni di ottimizzazione del gradiente, oltre che alla definizione della funzione di loss (SGD, CrossEntropyLoss). Inoltre, vengono cumulate le metriche utili al plotting dei grafici. Queste si troveranno dentro *logs/metrics* e poi si potranno trovare i tag *loss/train*, *accuracy/train*, *loss/*, *accuracy/*. Questa cartella verrà consultata da *tensorboard* durante la visualizzazione della schermata principale.
3. *Models.py*, qui vengono collezionati e implementati le varie reti neurali proposte, le CNN in questione. Tutte le classi ereditano la superclasse *nn.Module*.

Dentro la folder *logs*, come detto prima, vengono salvate le metriche delle funzioni da plottare



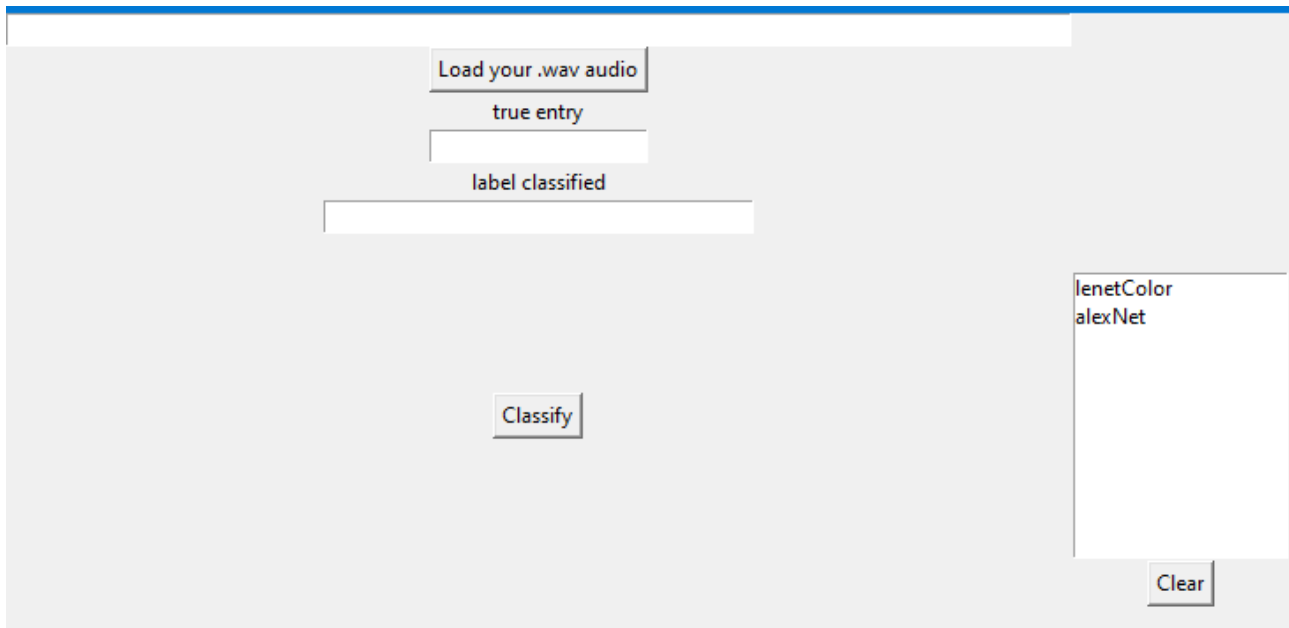
Ultima folder *resources/archive*, ci saranno

 Data	18/06/2023 13:04	Cartella di file
 stored	18/06/2023 10:38	Cartella di file

Dentro *Data* troveremo i dataset originali e quelli trasformati dove sono contenute le immagini in cui sono applicate i resize e la cancellazione del bordo. Inoltre, è possibile trovare le cartelle in cui la funzione *splitfolders* ha generato le cartelle a partire dalla folder trasformata.

Stored conterrà i pesi uscenti dal training dei modelli. Questi saranno essenziali quando dovremmo fare restore del modello, nella classe di interface, per poter utilizzare la rete neurale nella classificazione della singola istanza.

Main.interface



Dalla schermata principale è possibile cliccare su *load audio* per selezionare il file .wav da classificare.

Da questo, attraverso una fase di preprocessing, offerto da librosa, possiamo ottenere il nostro mel-spectrogram.

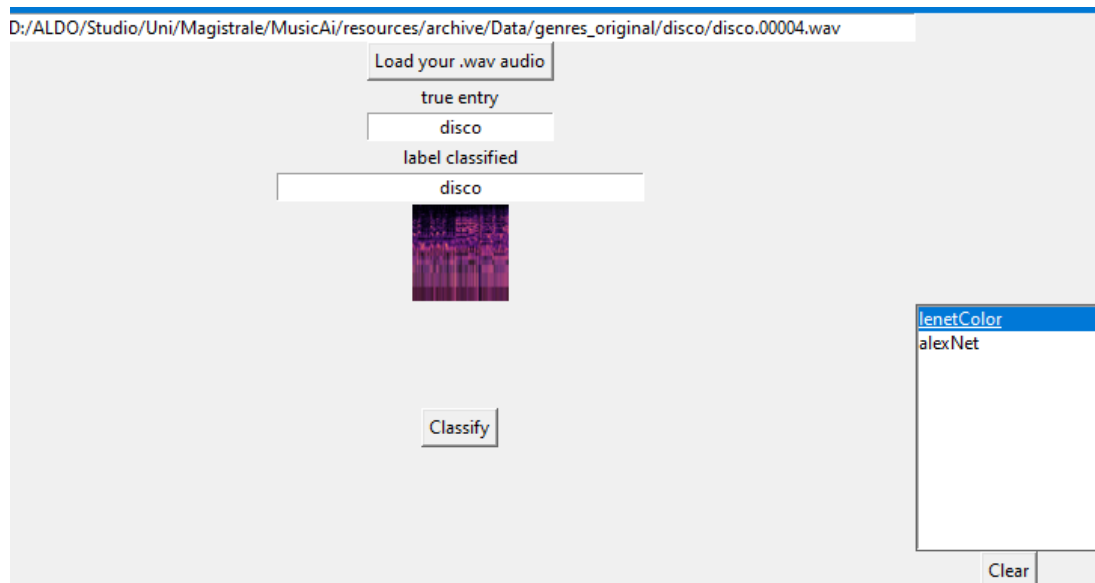
```
S = librosa.feature.melspectrogram(y=data, sr=sr)
S_DB = librosa.amplitude_to_db(S, ref=np.min)
librosa.display.specshow(S_DB, sr = sr, hop_length = hop_length, x_axis = 'time',
y_axis = 'log')
```

Esso, verrà dato in pasto al modello trainato in precedenza, di cui avremmo fatto il restore, attraverso il file dei pesi salvato.

```
model = MiniAlexNet(outChannels=16).to("cpu")
model.load_state_dict(torch.load("./resources/archive/stored/models/miniAlex.pth"))
model.eval()
```

La funzione `.eval()` è necessaria per disabilitare quei moduli usati in fase di training, come dropout e batchnorm.

Dalla destra potremmo selezionare il tipo di modello da usare per classificare l'immagine



Conclusioni

Si è visto e trattata l'applicazione di una rete neurale di tipo CNN ad un problema di classificazione odierno. Sebbene le classi, in questo esempio, siano ben discrete, la musica oggi possiede un vario e notevole numero di generi. Inoltre, il dataset usato potrebbe essere ormai datato se consideriamo come la musica si è evoluta nel tempo, sia in termini di produzioni di suoni sia in termine di stile.

Per migliorare il risultato attuale si potrebbe pensare di espandere l'attuale dataset, magari partendo proprio dall'inclusione di nuovi brani e/o di nuovi parti estratte da un audio e non limitarsi ai 30 secondi proposti nell'esempio.

Un'altra strada per migliorare l'approccio al problema è quello di usare una rete pre-trained e più complessa rispetto a queste proposte. Alla fine, il compito di image classification è piuttosto noto al mondo della ricerca. Si potrebbe tentare di fare un transfer learning, qualora i risultati non fossero soddisfacenti, del nuovo modello, aggiungendo layer nella parte finale della rete o all'inizio, in base alla propria dimensione del dataset e a quella usata per allenare il modello pre-trained.