UNIVERSITÀ DEGLI STUDI DI CATANIA

Dipartimento di Matematica e Informatica

Corso di Laurea Magistrale in Informatica

Condorelli Giuseppe, Mezzina Alessio

Music Instruments Recognition

Progetto di fine corso

Prof. Giovanni Maria Farinella

Anno Accademico 2022 - 2023

Contents

List of Figures

List of Tables

Chapter 1

Problema

1.1 Introduzione

Il presente progetto si propone di affrontare la sfida del riconoscimento di strumenti musicali tramite immagini utilizzando una Convolutional Neural Network (CNN), una classe di modelli di machine learning profondo particolarmente adatta per l'analisi di dati visivi complessi [1].
L'obiettivo primario è sviluppare un sistema automatico e preciso in grado di identificare e classificare correttamente gli strumenti musicali in base alle immagini fornite.

La scelta di utilizzare una CNN deriva dalla sua natura intrinsecamente adatta al trattamento di dati di tipo immagine. Le reti neurali convoluzionali presentano una notevole capacità di apprendimento delle caratteristiche visive dai dati forniti, consentendo di catturare in modo efficiente le informazioni salienti relative agli strumenti musicali. Questa architettura è stata ampiamente impiegata in una vasta gamma di applicazioni di visione artificiale con risultati notevolmente promettenti, dimostrando la sua versatilità e robustezza [2, 3]. Ad esempio già nel 2012 si dimostrava un tasso di errore solo del 0.23% sul MNIST database [4], Successivamente, una CNN simile chiamata AlexNet [5] ha vinto il *"ImageNet Large Scale Visual Recognition Challenge 2012"*

Un altro aspetto fondamentale di questo progetto riguarda l'acquisizione e la preparazione del dataset. La fase di acquisizione dei dati ha richiesto una metodologia accurata per garantire la diversità e la rappresentatività delle immagini relative agli strumenti musicali. Sono stati adottati approcci di web scraping, utilizzando appositi script modificati, per raccogliere un vasto insieme di immagini provenienti da fonti attendibili e diversificate come siti di e-commerce specializzati in strumenti

musicali. Il dataset è stato successivamente sottoposto a operazioni di pulizia, includendo il filtraggio delle immagini di scarsa qualità e l'eliminazione di duplicati e placeholder, al fine di ottenere un dataset finale ottimizzato e bilanciato.

Dopo aver stabilito l'architettura della CNN e completata la fase di esperimento la nostra attenzione si è spostata sullo sviluppare diversi esperimenti e le valutazioni delle prestazione del modello, questo argomento è infatti abbondatemente approfondito nei capitoli 3, 4 e 5. Infine una spiegazione sommaria dei codici ausiliari utilizzati durante il progetto è presente nel capitolo 7, insieme ad una spiegazione dettagliata dei codici riguardanti gli obiettivi del corso.

1.1.1 Reperibilità del materiale

Tutto il materiale sviluppato, incluso il dataset è repribile alla seguente repo [6] di GitHub.

Chapter 2

Dataset

In questo capitolo, descriveremo in dettaglio il dataset utilizzato per addestrare e valutare il modello di machine learning per il riconoscimento di strumenti musicali tramite immagini. Esamineremo:

1. la fase di acquisizione dei dati
2. Operazioni preliminari sui dati
3. Esempio visivo dei dati
4. Dati di training e dati di test
5. Organizzazione cartelle del dataset

Inoltre, forniremo esempi visivi dei dati, discuteremo l'organizzazione delle cartelle e la suddivisione tra dati di training e di test.

2.1 Fase di acquisizione dei dati

La fase di acquisione rappresenta il passo fondamentale nella creazione di un dataset, l'obiettivo principale è stato quello di raccogliere un numero significativo di immagini rappresntative di diverse categorie di sturmenti musciali, per raggiungere questo obiettivo sono state utilizzate inizialmente diverse tecniche, infine abbiamo continuato con script di web scraping sulla base di quelli realizzati durante le lezioni di social media managment tenute dal Professore Furnari [7]. Gli script in questione sono stati apiamente modificati al fine di poter gestire il download delle immagini, (nel caso delle chitarre più di 7000 immagini) con un unica run dello script. I siti

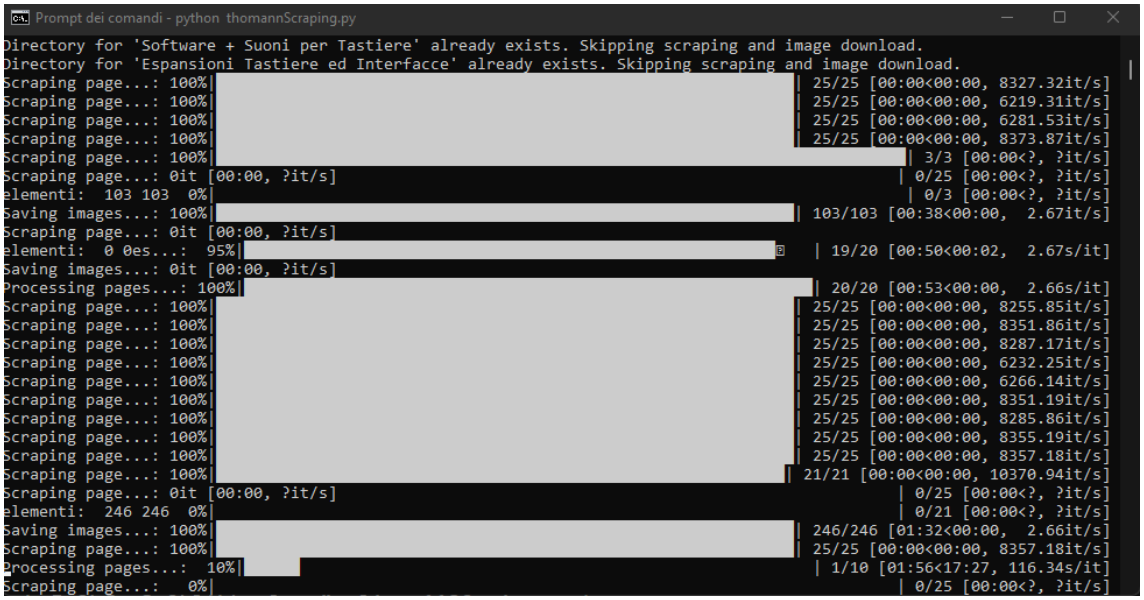


Figure 2.1: Esempio output script di web scraping.

utilizzati sono stati diversi come *guitar center* [8], *thomann* [9]. Inoltre è stata anche valutata la possibilità di utilizzare la libreria python *bing-image-downloader* [10] per scaricare le immagini dal motore di ricerca bing, successivamente questa libreria è stata poco utilizzata nel nostro progetto in quanto non forniva immagini ottimali per la creazione del dataset dal momento che le prendeva direttamente da internet.

Durante il processo di web scraping, abbiamo utilizzato la libreria Python "BeautifulSoup" per analizzare il contenuto HTML delle pagine web. Attraverso richieste HTTP (HTTP requests), abbiamo ottenuto gli URL delle pagine dei prodotti relativi a ciascuna categoria di strumenti musicali. Successivamente, abbiamo estratto le immagini degli strumenti musicali presenti in queste pagine utilizzando gli attributi HTML appropriati. In segutio a questa fase di webscraping abbiamo ottenuto diverse cartelle con categorie, dal momento che erano tante abbiamo deciso di unirle in 4 macrocategorie. Complessivvamente, la fase di acquisizione dei dati ha fornito un dataset ampio ed eterogeneo, che ha rappresentato una solida base per la creazione della nostra CNN, nei prossimi paragrafi inoltre vedremo quali azioni sono state effettuate sul dataset

2.2 Operazioni preliminari sui dati

Dopo aver completato il processo di web scraping, ci siamo dedicati a garantire la qualità delle immagini raccolte. Un'attenzione particolare è stata data al filtraggio e alla rimozione di immagini di scarsa qualità o duplicati. Questo ci ha permesso di ottenere un dataset contenente solo immagini nitide e rappresentative degli strumenti musicali. Durante il web scraping, ci siamo imbattuti in diverse immagini che fungono da placeholder, poiché non erano disponibili immagini complete per alcuni strumenti musicali specifici. I codici riguardanti le operazioni di pulizia del dataset sono affrontate nel dettaglio nel capitolo 7.

Tutte le operazioni di web scraping hanno portato ad avere dataset con dimensioni indicate nella tabella 2.1, le operazioni effettuate sul dataset, descritte nel dettaglio successivamente, hanno invece portato il dataset finale ad avere le dimensionalità presentate nella tabella 2.2

Classe	**Numero di immagini**
Percussioni	3484
Strumenti a corda	7238
Strumenti a tastiera	2748
Strumenti a fiato	3268

Table 2.1: Numero di elementi ottenuti in seguito alle operazioni di raccolta dei dati

Classe	**Numero di immagini**
Eletric guitar	872
Handpan	190
Piano	200
Drum	252
Keyboard	275
Trumpet	177
Tuba	130
Wind Gong	121

Table 2.2: Numero di elementi dopo aver effettuato operazioni di pulizia

2.3 Operazioni sul dataset

Come precedentemente anticipato, sono state apportate diverse modifiche al dataset al fine di garantire una maggiore eterogeneità e completezza dei dati. In questo paragrafo, analizzeremo tutte le operazioni di modifica effettuate per passare dalla tabella 2.1 alla tabella 2.2.

Iniziamo esaminando la figura 2.2, che mostra un notevole sbilanciamento nel dataset iniziale verso la classe "Strumenti a corda" (circa 7240 campioni), rispetto alla classe con strumenti a tastiera (circa 2750 campioni).

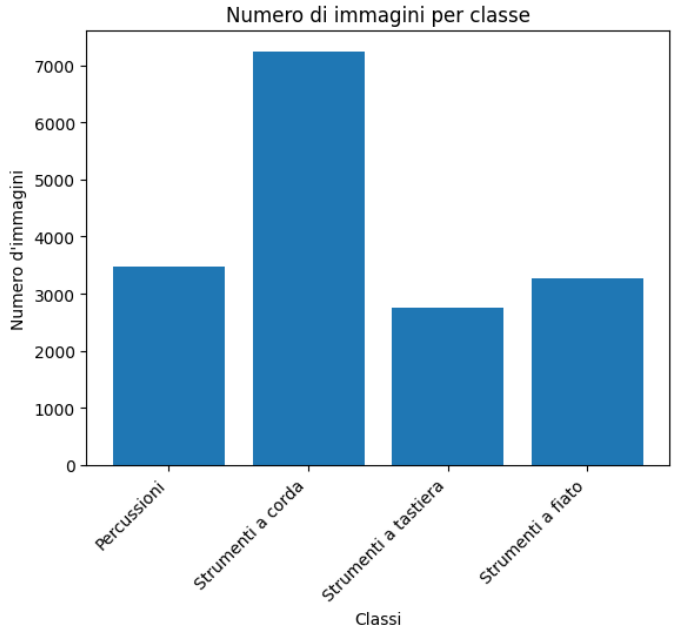


Figure 2.2: Istogramma delle classi

La presenza di una tale discrepanza nel numero di campioni tra le diverse classi potrebbe influenzare negativamente le prestazioni di un modello CNN, poiché il modello potrebbe apprendere in modo più accurato la classe dominante (avendo un alto bias) e trascurare le classi minori.

Per mitigare questo problema, sono state applicate diverse tecniche di pulizia e bilanciamento delle classi. Nella figura 2.3, è possibile osservare l'istogramma delle classi dopo l'esecuzione di tali operazioni.

Come si può notare, il dataset è stato ridotto da un totale di 16756 a circa 2217 immagini, ed inoltre è stato aggiunto un nuovo livello di differenziazione degli

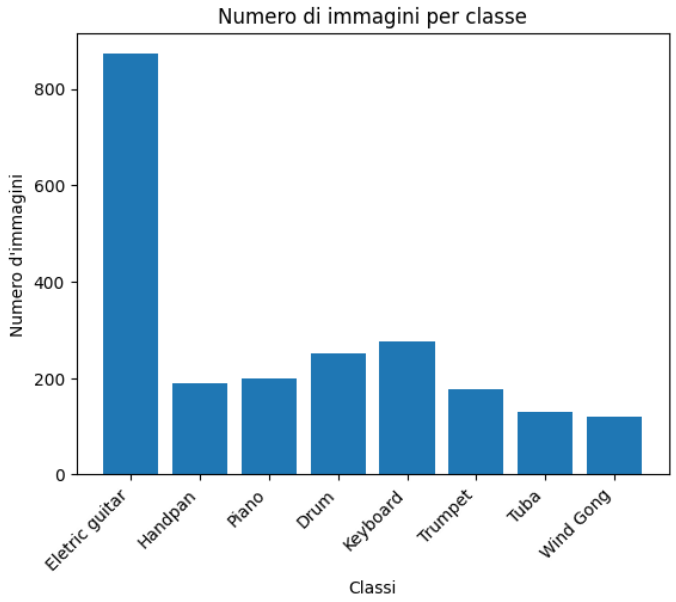


Figure 2.3: Istogramma delle classi dopo aver effettuato le operazioni di pulizia

strumenti. La riduzione è stata del circa 86.79%. Si può notare come il dataset sia ancora sbilanciato, infatti durante l'esecuzione dei vari esperimenti la classe "Eletric guitar" è stata ulteriormente diminuita a 219 elementi, quindi l'istogramma del dataset utilizzato durante gli esperimenti si può vedere in figura 2.4.

Questo aspetto verrà approfondito nel capitolo 8, in cui discuteremo l'impatto di questa riduzione sulla prestazione del modello di machine learning e di cosa è stato appreso nella creazione di un dataset.

Le operazioni di pulizia hanno coinvolto diverse tecniche, tra cui l'analisi della risoluzione delle immagini, il controllo delle dimensioni e il rilevamento di immagini duplicate. Inoltre, abbiamo affrontato il problema dei placeholder nel dataset. Poiché queste immagini vuote non rappresentavano gli strumenti musicali, abbiamo deciso di eliminarle per evitare di introdurre informazioni fuorvianti nel nostro dataset.

A tal fine, è stato creato uno script denominato *RemovePlaceholde.py*, che ha permesso di individuare e rimuovere in modo efficiente i placeholder. Un esempio dell'output dello script è mostrato nella figura 2.5.

Queste operazioni di pulizia e bilanciamento hanno contribuito a migliorare la

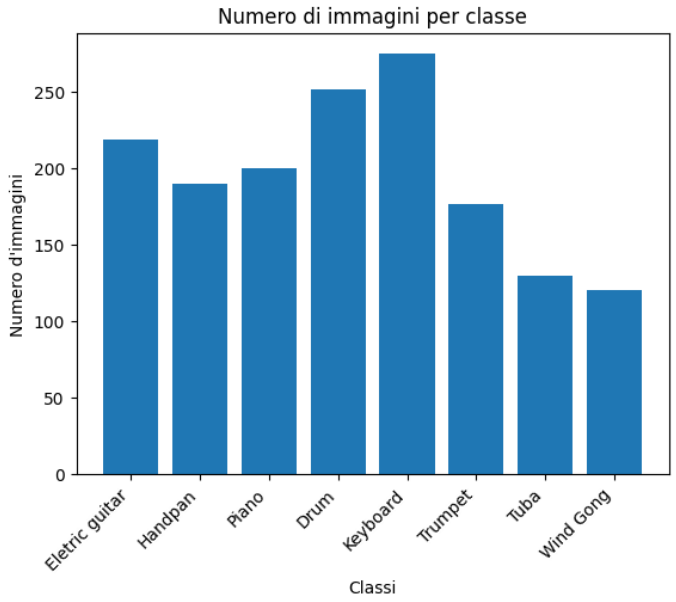


Figure 2.4: Istogramma delle classi dopo aver effettuato le operazioni di pulizia

```
C:\Users\Alessio\Desktop\Data>python  RemovePlaceholder.py
Processing images: 100%|                                                  | 7188/7188 [00:18<00
Removed 250 duplicate images.

C:\Users\Alessio\Desktop\Data>
```

Figure 2.5: Esempio output script di rimozione dei placeholder

qualità del dataset e, di conseguenza, hanno avuto un impatto significativo sul processo di addestramento e valutazione del nostro modello di machine learning. Ne discuteremo ulteriormente nei capitoli successivi, evidenziando le prestazioni del modello e le considerazioni sull'uso di dati bilanciati e puliti per il nostro progetto di machine learning.

2.4 Esempio visivo dei dati

In questa sezione, forniremo un esempio visivo dei dati contenuti nel nostro dataset. Gli esempi visivi ci permetteranno di avere un'idea più chiara delle immagini rappresentative delle diverse classi di strumenti musicali presenti nel dataset.

Il dataset finale è stato suddiviso in otto classi principali: Electric guitar, Handpan, Piano, Drum, Keyboard, Trumpet, Tuba, Wind Gong. Ciascuna classe rappresenta una categoria specifica di strumenti musicali. Di seguito, mostriamo alcuni

esempi visivi rappresentativi per ciascuna classe.

2.4.1 Electric guitar

Le chitarre elettriche sono uno degli strumenti musicali più iconici nel mondo della musica moderna. Questi strumenti producono suoni attraverso l'amplificazione delle vibrazioni delle corde.

(a) (b)

Figure 2.6: Electric guitar

2.4.2 Handpan

L'handpan è uno strumento a percussione relativamente nuovo, con un suono caldo e etereo. Viene suonato con le mani e produce note melodiche.

(a) (b)

Figure 2.7: Handpan

2.4.3 Piano

Il pianoforte è uno degli strumenti a tastiera più diffusi ed è ampiamente utilizzato nella musica classica e contemporanea. Produce suoni premendo i tasti associati alle diverse note musicali.

(a) (b)

Figure 2.8: Piano

2.4.4 Drum

Le percussioni sono uno degli elementi fondamentali di qualsiasi ensemble musicale. Esse includono tamburi, piatti, batterie e molti altri strumenti che vengono suonati colpendoli o scuotendoli. Di seguito sono riportati alcuni esempi visivi di strumenti percussivi presenti nel nostro dataset (i tamburi) :

(a) (b)

Figure 2.9: Tamburi

2.4.5 Keyboard

Gli strumenti a tastiera includono tastiere elettroniche e sintetizzatori, e producono suoni premendo i tasti associati alle diverse note musicali.

(a) (b)

Figure 2.10: Keyboard

2.4.6 Trumpet

La tromba è uno strumento a fiato noto per il suo suono brillante e potente. Produce suoni facendo vibrare il flusso d'aria attraverso le labbra del suonatore.

(a) (b)

Figure 2.11: Trumpet

2.4.7 Tuba

La tuba è uno strumento a fiato a bassa frequenza che produce suoni profondi e ricchi. Come gli altri strumenti a fiato, produce suoni facendo vibrare il flusso d'aria attraverso le labbra del suonatore.

(a) (b)

Figure 2.12: Tuba

2.4.8 Wind Gong

Il wind gong è uno strumento a percussione dalla forma piatta e produce suoni vibrando quando viene colpito.

(a) (b)

Figure 2.13: Wind Gong

2.5 Dati di training, dati di validation e dati di test

Il dataset è stato diviso con l'obiettivo di facilitare la fase di sperimentazione con modelli di CNN di deep learning. A tal fine, il set di immagini è stato suddiviso in tre partizioni:

- **Train:** Questa partizione contiene le immagini utilizzate dal modello per la fase di training, durante la quale il modello apprende le differenze tra le diverse tipologie di strumenti musicali proposti.
- **Validation:** Questa partizione verrà utilizzata in combinazione con il set di training al fine di valutare l'apprendimento del modello e prevenire problemi di overfitting.
- **Test:** Questa ulteriore partizione sarà utilizzata per valutare il modello allenato e trarre conclusioni sulla base delle metriche ottenute.

Partizione	**Numero di immagini**
Train	1174
Validation	240
Test	160

Table 2.3: Numero di immagini presenti in ogni partizione

2.6 Organizzazione cartelle del dataset

Ogni partizione del dataset (train, validation, test) è stata collocata all'interno della cartella omonima al fine di poter essere facilmente visionata ed utilizzata. Inoltre, all'interno di ciascuna partizione, sono presenti 8 cartelle, suddivise per classe di strumento musicale (Drum, Electric guitar, Handpan, Keyboard, Piano, Trumpet, Tuba e Wind gong). Ciascuna di queste cartelle contiene il set di immagini corrispondente alla classe di strumento musicale specifica.

La struttura del dataset è la seguente:

- **Train:** contiene le immagini utilizzate per il training del modello.
  - **Drum:** immagini della classe di strumento "Drum".
  - **Electric guitar:** immagini della classe di strumento "Electric guitar".
  - **Handpan:** immagini della classe di strumento "Handpan".
  - **Keyboard:** immagini della classe di strumento "Keyboard".
  - **Piano:** immagini della classe di strumento "Piano".
  - **Trumpet:** immagini della classe di strumento "Trumpet".
  - **Tuba:** immagini della classe di strumento "Tuba".
  - **Wind gong:** immagini della classe di strumento "Wind gong".
- **Validation:** contiene le immagini utilizzate per la validazione dell'apprendimento del modello, al fine di prevenire il problema di overfitting. La struttura delle sottocartelle è identica a quella di "Train".
- **Test:** contiene le immagini utilizzate per la valutazione finale del modello. La struttura delle sottocartelle è identica a quella di "Train" e "Validation".

Questo setup permette di organizzare in modo chiaro e sistematico le immagini per ciascuna classe di strumento musicale, semplificando così l'accesso e l'utilizzo del dataset durante la fase di sperimentazione e addestramento dei modelli di CNN di deep learning.

Chapter 3

Metodi

3.1 ImageDataGenerator

Per caricare il set di immagini su python per poter essere utilizzati da un modello CNN abbiamo utilizzato ImageDataGenerator fornito dalla libreria Keras [11].

ImageDataGenerator [12] e' un ottimo tool che permette, specificato il path del set di immagini ed altri parametri discussi piu' avanti, di fare il load delle immagini in memoria ed applicarne alcune trasformate al fine di adattare i dati al modello nonche' farne Data Augmentation utile per migliorarne ulteriormente le prestazioni.

Il costruttore è così definito:

```
ImageDataGenerator(
rescale=1./255,
rotation_range=20,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
vertical_flip=True,)
```

Spiegazione dei parametri usati:

- **rescale:** Questo parametro consente di normalizzare i valori contenuti nelle

immagini (che variano da 0 a 255) portandoli nel range [0,1]. Questa operazione è di fondamentale importanza poiché favorisce ulteriormente la convergenza di un modello di Deep Learning.

- **rotation_range:** Questo parametro permette di selezionare l'intervallo in gradi entro cui le immagini vengono ruotate, al fine di eseguire l'aumento dei dati (data augmentation) e consentire al modello di generalizzare meglio. Ruotando leggermente le immagini di addestramento, si possono ottenere nuove varianti dei dati, rendendo il modello più robusto e meno incline al sovra-addestramento (overfitting).

- **width_shift_range** e **height_shift_range**: Questi parametri permettono di specificare intervalli relativi alle traslazioni orizzontali e verticali delle immagini. In questo caso, `width_shift_range=0.2` significa che le immagini possono essere traslate orizzontalmente fino a un massimo del 20% rispetto alla loro larghezza, mentre `height_shift_range=0.2` permette traslazioni verticali fino al 20% dell'altezza dell'immagine.

- **shear_range**: Questo parametro consente di specificare un intervallo di taglio per le immagini. La trasformazione di taglio applica un'effetto di distorsione all'immagine lungo una direzione specificata. `shear_range=0.2` significa che le immagini possono subire una distorsione fino al 20%.

- **zoom_range**: Questo parametro consente di specificare un intervallo per lo zoom delle immagini. Ad esempio, `zoom_range=0.2` significa che le immagini possono essere ingrandite fino al 20% o rimpicciolite fino al 20%.

- **horizontal_flip** e **vertical_flip**: Questi parametri abilitano o disabilitano la possibilità di eseguire uno specchio orizzontale o verticale delle immagini, rispettivamente.

I parametri utilizzati permettono di aiutare ulteriormente il modello a essere più robusto rispetto alle variazioni nei dati di test.

3.2 Convolutional Neural Network

Per affrontare il problema proposto, abbiamo utilizzato l'architettura CNN, la più adatta per ottenere una rappresentazione funzionale delle immagini presenti nel nostro dataset.

Al fine di costruire la migliore architettura per la classificazione degli strumenti musicali, abbiamo adottato una strategia bottom-up, partendo da un modello molto semplice contenente pochi layer e progressivamente arrivando al modello più completo. Questa approccio ci ha permesso di calibrare la complessità del modello in base ai singoli risultati ottenuti.

Nelle architetture CNN, ogni modello presenta almeno un layer di Convoluzione con funzione di attivazione Relu, seguito da layers di Pooling. Inoltre, considerando le 8 classi presenti nel dataset, ogni modello dispone di un layer finale con 8 neuroni e funzione di attivazione Softmax per la classificazione.

Per l'apprendimento del modello, abbiamo utilizzato l'algoritmo di ottimizzazione Adam, che si basa sulla discesa del gradiente, insieme alla funzione di loss Cross Entropy per problemi multiclasse e la metrica Accuracy per valutare le prestazioni del modello.

Inoltre, per favorire una più completa valutazione e corretta sulle reali performance del modello, sono state usate ulteriori metriche e grafici presentati e spiegati nella sezione succcessiva.

Chapter 4

Valutazione

4.1 Metriche e grafici

Le metriche e grafici utilizzati sono:

- **Training and Validation Accuracy:** Questo grafico permette di avere una visuale completa sull'andamento dell'accuracy con lo scorrere delle epoche al fine di comprendere il comportamento del modello in fase di training.
- **Training and Validation Loss:** Questo grafico permette di capire se il modello sta riuscendo a convergere col passare delle epoche.
- **ROC:** Questo grafico si basa sulle metriche True Positive Rate e False Positive Rate e ci permette di comprendere quanto il nostro modello sta andando bene sul test set.
- **Confusion Matrix:** Una importante matrice che mostra le predizioni del modello rispetto alle classi reali al fine di stimarne altre metriche utili per l'analisi.
- **F1 score:** La metrica più utilizzata che ci permette di avere un valore reale sull'accuratezza del nostro modello sul test set.

Durante gli esperimenti, osserveremo attentamente tutti i grafici e le metriche citate al fine di valutarne correttamente le capacità e dunque adottare strategie migliori per il miglioramento del modello.

Chapter 5

Esperimenti

Table 5.1: Risultati degli Esperimenti

N° Esperimento	**Accuracy**	**Val_Accuracy**	**F1 macro avg**	**Note**
1	0.8773	0.7958	0.16	
2	0.8935	0.8167	0.14	Aggiunti layers di Conv e MaxPool
3	0.9412	0.7667	0.10	Aggiunto Batch Normalization
4	0.9608	0.3958	0.10	Aggiunti layers di Conv e MaxPool
5	0.8637	0.7708	0.15	Aggiunti layers di Dropout
6	0.9063	0.8417	0.10	Aumentati numero di Kernels e Neuroni
7	0.9302	0.7833	0.13	Diminuiti numero di Kernels e Neuroni
8	0.9796	0.3292	0.09	Fine tuning con EfficientNetB0
9	0.8603	0.8083	0.08	Primo modello semplificato

5.1 Primo esperimento

Table 5.2: Modello primo esperimento

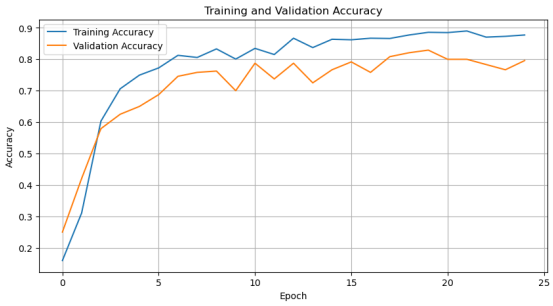
Layer (type)	**Output Shape**	**Param #**
conv2d_41 (Conv2D)	(None, 126, 126, 128)	3584
max_pooling2d_41 (MaxPooling2D)	(None, 63, 63, 128)	0
flatten_20 (Flatten)	(None, 508032)	0
dense_46 (Dense)	(None, 64)	32514112
dense_47 (Dense)	(None, 8)	520
Total params		32,518,216
Trainable params		32,518,216
Non-trainable params		0

Il primo modello risulta essere molto semplice poiché presenta solo 1 layer di Convolution seguito da 1 layer di max pooling, per poi intrecciarsi con due layer densi, di cui l'ultimo è utile per la classificazione multiclasse mediante la funzione di attivazione Softmax. Viene inoltre utilizzato l'ottimizzatore Adam per la discesa del gradiente e la funzione di loss categorical_crossentropy per misurare la discrepanza tra le etichette reali e le previsioni del modello durante l'addestramento.

Output epochs:

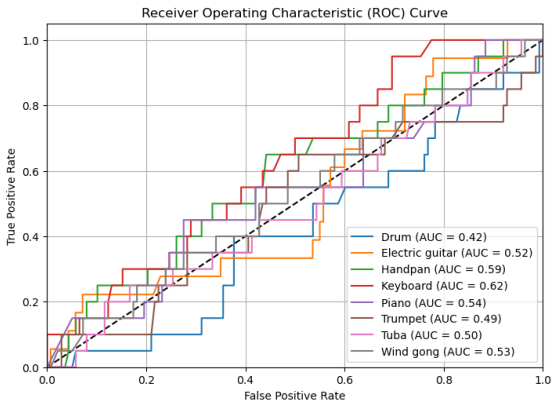
```
Epoch 1/25
19/19 [==============================] - 30s 2s/step - loss: 12.2884 - accuracy: 0.1593 - val_loss: 1.9285 -
     val_accuracy: 0.2500
Epoch 2/25
19/19 [==============================] - 24s 1s/step - loss: 1.6737 - accuracy: 0.3109 - val_loss: 1.6716 -
     val_accuracy: 0.4208
Epoch 3/25
19/19 [==============================] - 22s 1s/step - loss: 1.1791 - accuracy: 0.6031 - val_loss: 1.2575 -
     val_accuracy: 0.5792
Epoch 4/25
19/19 [==============================] - 21s 1s/step - loss: 0.9009 - accuracy: 0.7061 - val_loss: 1.1305 -
     val_accuracy: 0.6250
Epoch 5/25
19/19 [==============================] - 22s 1s/step - loss: 0.7933 - accuracy: 0.7496 - val_loss: 1.0575 -
     val_accuracy: 0.6500
Epoch 6/25
19/19 [==============================] - 21s 1s/step - loss: 0.7314 - accuracy: 0.7726 - val_loss: 0.9463 -
     val_accuracy: 0.6875
Epoch 7/25
19/19 [==============================] - 22s 1s/step - loss: 0.6179 - accuracy: 0.8126 - val_loss: 0.7839 -
     val_accuracy: 0.7458
Epoch 8/25
19/19 [==============================] - 21s 1s/step - loss: 0.6127 - accuracy: 0.8058 - val_loss: 0.8283 -
     val_accuracy: 0.7583
```

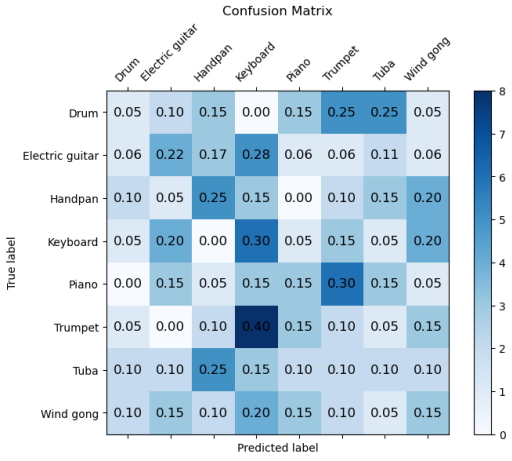
```
Epoch 9/25
19/19 [==============================] - 22s 1s/step - loss: 0.5408 - accuracy: 0.8330 - val_loss: 0.8284 -
    val_accuracy: 0.7625
Epoch 10/25
19/19 [==============================] - 21s 1s/step - loss: 0.5855 - accuracy: 0.8007 - val_loss: 0.9201 -
    val_accuracy: 0.7000
Epoch 11/25
19/19 [==============================] - 22s 1s/step - loss: 0.5376 - accuracy: 0.8348 - val_loss: 0.7365 -
    val_accuracy: 0.7875
Epoch 12/25
19/19 [==============================] - 22s 1s/step - loss: 0.5800 - accuracy: 0.8152 - val_loss: 0.8509 -
    val_accuracy: 0.7375
Epoch 13/25
...
Epoch 24/25
19/19 [==============================] - 22s 1s/step - loss: 0.3941 - accuracy: 0.8731 - val_loss: 0.6349 -
    val_accuracy: 0.7667
Epoch 25/25
19/19 [==============================] - 22s 1s/step - loss: 0.3973 - accuracy: 0.8773 - val_loss: 0.6035 -
    val_accuracy: 0.7958
```



(a)

(b)





(c)

Figure 5.1: Grafici primo esperimento

Come è possibile analizzare dai grafici proposti in Fig. 5.1(a) e Fig. 5.1(b) e dall'output riguardo il training nel corso dell'epoch, è possibile notare come il modello riesca perfettamente a convergere riuscendo addirittura a raggiungere un'accuracy nel train del 87.73%, e nel validation del 79.58%.

Tuttavia, nonostante i promettenti risultati in fase di training, analizzando attentamente la curva ROC e la Confusion Matrix, mostrata in Fig. 5.1(c) siamo chiaramente in grado di notare come il modello predica molto spesso sbagliato nel test set, traducendosi dunque in un pessimo risultato che è confermato dal F1 Macro avg di solo 16%. (Tab.5.3)

Class	Precision	Recall	F1-Score	Support
Drum	0.10	0.05	0.07	20
Electric guitar	0.21	0.22	0.22	18
Handpan	0.24	0.25	0.24	20
Keyboard	0.19	0.30	0.23	20
Piano	0.19	0.15	0.17	20
Trumpet	0.09	0.10	0.09	20
Tuba	0.11	0.10	0.11	20
Wind gong	0.16	0.15	0.15	20
Accuracy			0.16	158
Macro Avg	0.16	0.17	0.16	158
Weighted Avg	0.16	0.16	0.16	158

Table 5.3: Classification Report Primo Esperimento

5.2 Secondo esperimento

Table 5.4: Modello secondo esperimento

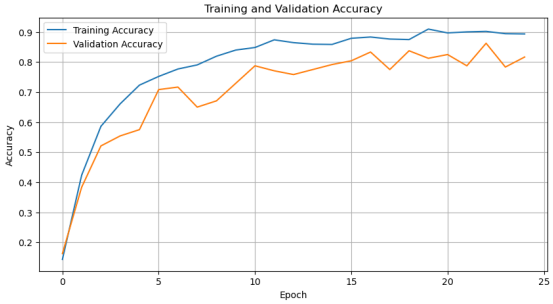
Layer (type)	**Output Shape**	**Param #**
conv2d_42 (Conv2D)	(None, 126, 126, 128)	3584
max_pooling2d_42 (MaxPooling2D)	(None, 63, 63, 128)	0
conv2d_43 (Conv2D)	(None, 61, 61, 64)	73792
max_pooling2d_43 (MaxPooling2D)	(None, 30, 30, 64)	0
flatten_21 (Flatten)	(None, 57600)	0
dense_48 (Dense)	(None, 64)	3686464
dense_49 (Dense)	(None, 8)	520
Total params		3,764,360
Trainable params		3,764,360
Non-trainable params		0

In questo esperimento abbiamo deciso di complicare leggermente di più il modello aumentandone i livelli di layer.
In particolare abbiamo aggiunto un layer di convoluzione con 64 kernels ed un ulteriore layer di MaxPooling.

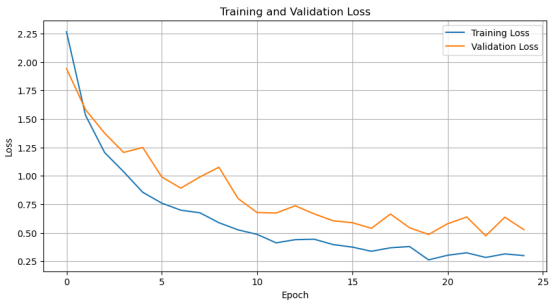
Output epochs:

```
Epoch 1/25
19/19 [==============================] - 31s 2s/step - loss: 2.2662 - accuracy: 0.1431 - val_loss: 1.9425 -
     val_accuracy: 0.1625
Epoch 2/25
19/19 [==============================] - 29s 2s/step - loss: 1.5320 - accuracy: 0.4233 - val_loss: 1.5820 -
     val_accuracy: 0.3833
Epoch 3/25
19/19 [==============================] - 29s 2s/step - loss: 1.2052 - accuracy: 0.5860 - val_loss: 1.3755 -
     val_accuracy: 0.5208
Epoch 4/25
19/19 [==============================] - 29s 1s/step - loss: 1.0361 - accuracy: 0.6610 - val_loss: 1.2058 -
     val_accuracy: 0.5542
Epoch 5/25
19/19 [==============================] - 29s 2s/step - loss: 0.8561 - accuracy: 0.7232 - val_loss: 1.2498 -
     val_accuracy: 0.5750
Epoch 6/25
19/19 [==============================] - 27s 1s/step - loss: 0.7604 - accuracy: 0.7521 - val_loss: 0.9902 -
     val_accuracy: 0.7083
Epoch 7/25
19/19 [==============================] - 27s 1s/step - loss: 0.6985 - accuracy: 0.7768 - val_loss: 0.8925 -
     val_accuracy: 0.7167
Epoch 8/25
19/19 [==============================] - 27s 1s/step - loss: 0.6755 - accuracy: 0.7905 - val_loss: 0.9904 -
     val_accuracy: 0.6500
Epoch 9/25
```

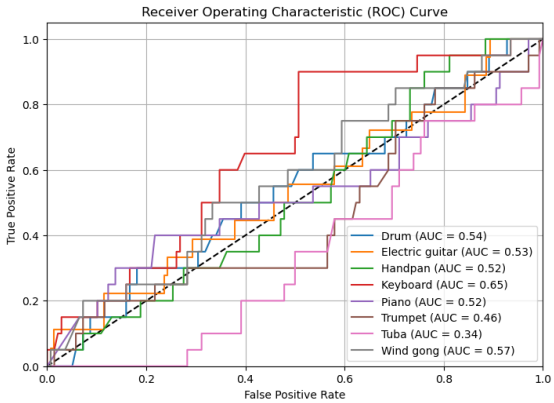
```
19/19 [==============================] - 27s 1s/step - loss: 0.5885 - accuracy: 0.8194 - val_loss: 1.0761 -
    val_accuracy: 0.6708
Epoch 10/25
19/19 [==============================] - 27s 1s/step - loss: 0.5256 - accuracy: 0.8399 - val_loss: 0.8015 -
    val_accuracy: 0.7292
Epoch 11/25
19/19 [==============================] - 27s 1s/step - loss: 0.4863 - accuracy: 0.8484 - val_loss: 0.6778 -
    val_accuracy: 0.7875
Epoch 12/25
19/19 [==============================] - 27s 1s/step - loss: 0.4114 - accuracy: 0.8739 - val_loss: 0.6736 -
    val_accuracy: 0.7708
Epoch 13/25
...
Epoch 24/25
19/19 [==============================] - 28s 1s/step - loss: 0.3140 - accuracy: 0.8944 - val_loss: 0.6372 -
    val_accuracy: 0.7833
Epoch 25/25
19/19 [==============================] - 28s 1s/step - loss: 0.2997 - accuracy: 0.8935 - val_loss: 0.5272 -
    val_accuracy: 0.8167
```

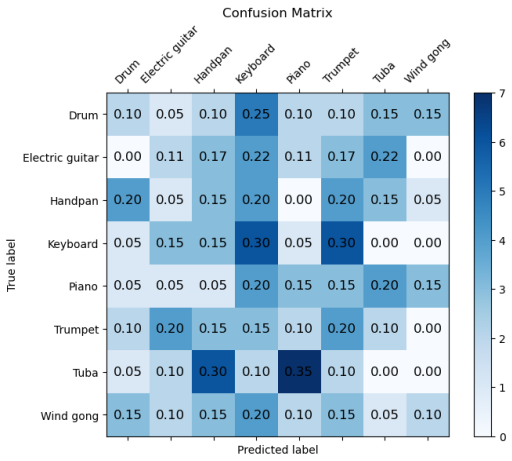


(a)



(b)





(c)

Figure 5.2: Grafici modello secondo esperimento

Questa scelta ha portato il modello ad avere un accuracy in fase di training

ancora più elevata riuscendo ad ottenere un 89.35% nel train ed un 81.67% nel validation.

Tuttavia anche qui, come nel primo caso, ci troviamo in una situazione di overfitting dato che il modello continua a caversela benissimo in fase di training ma malissimo in fase di testing come dimostrato dalle curve ROC che si mantengono vicine alla diagonale principale ed alla matrice di confusione che non ha una diagonale più marcata.

Class	Precision	Recall	F1-Score	Support
Drum	0.14	0.10	0.12	20
Electric guitar	0.12	0.11	0.12	18
Handpan	0.12	0.15	0.14	20
Keyboard	0.19	0.30	0.23	20
Piano	0.16	0.15	0.15	20
Trumpet	0.15	0.20	0.17	20
Tuba	0.00	0.00	0.00	20
Wind gong	0.22	0.10	0.14	20
Accuracy			0.14	158
Macro Avg	0.14	0.14	0.13	158
Weighted Avg	0.14	0.14	0.13	158

Table 5.5: Classification Report

5.3 Terzo esperimento

Table 5.6: Modello terzo esperimento

Layer (type)	**Output Shape**	**Param #**
conv2d_44 (Conv2D)	(None, 126, 126, 128)	3584
max_pooling2d_44 (MaxPooling2D)	(None, 63, 63, 128)	0
conv2d_45 (Conv2D)	(None, 61, 61, 64)	73792
max_pooling2d_45 (MaxPooling2D)	(None, 30, 30, 64)	0
batch_normalization_3 (BatchNormalization)	(None, 30, 30, 64)	256
flatten_22 (Flatten)	(None, 57600)	0
dense_50 (Dense)	(None, 64)	3686464
dense_51 (Dense)	(None, 8)	520
Total params		3,764,616
Trainable params		3,764,488
Non-trainable params		128

In questo esperimento abbiamo deciso di aggiungere un layer di Batch normalization per cercare di ridurre l'overfitting del modello permettendogli di generalizzare meglio cosiì da ottenere risultati migliori in fase di testing.

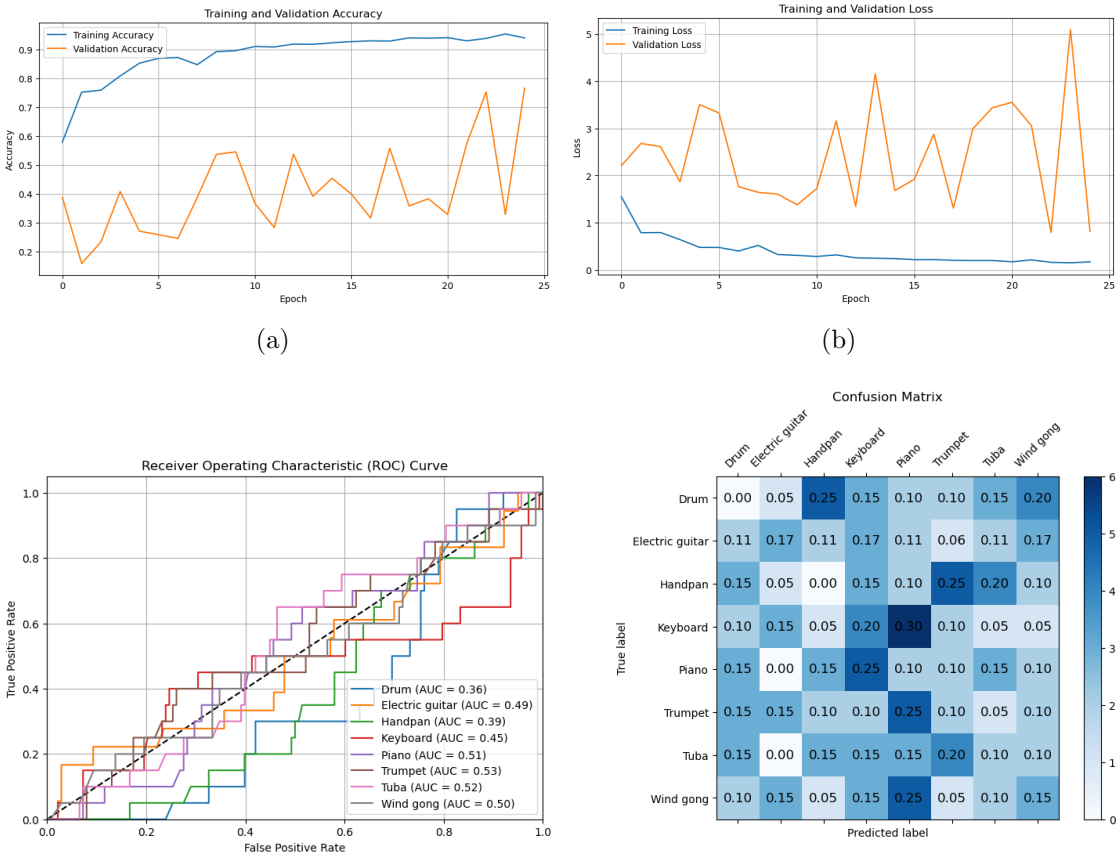
Output epochs:

```
Epoch 1/25
19/19 [==============================] - 42s 2s/step - loss: 1.5492 - accuracy: 0.5801 - val_loss: 2.2133 -
     val_accuracy: 0.3875
Epoch 2/25
19/19 [==============================] - 39s 2s/step - loss: 0.7902 - accuracy: 0.7530 - val_loss: 2.6838 -
     val_accuracy: 0.1583
Epoch 3/25
19/19 [==============================] - 39s 2s/step - loss: 0.7929 - accuracy: 0.7598 - val_loss: 2.6156 -
     val_accuracy: 0.2333
Epoch 4/25
19/19 [==============================] - 39s 2s/step - loss: 0.6436 - accuracy: 0.8092 - val_loss: 1.8718 -
     val_accuracy: 0.4083
Epoch 5/25
19/19 [==============================] - 40s 2s/step - loss: 0.4790 - accuracy: 0.8535 - val_loss: 3.5075 -
     val_accuracy: 0.2708
Epoch 6/25
19/19 [==============================] - 39s 2s/step - loss: 0.4766 - accuracy: 0.8705 - val_loss: 3.3293 -
     val_accuracy: 0.2583
Epoch 7/25
19/19 [==============================] - 38s 2s/step - loss: 0.4025 - accuracy: 0.8731 - val_loss: 1.7670 -
     val_accuracy: 0.2458
Epoch 8/25
19/19 [==============================] - 39s 2s/step - loss: 0.5203 - accuracy: 0.8484 - val_loss: 1.6466 -
     val_accuracy: 0.3875
Epoch 9/25
```

```
19/19 [==============================] - 42s 2s/step - loss: 0.3282 - accuracy: 0.8935 - val_loss: 1.6079 -
     val_accuracy: 0.5375
Epoch 10/25
19/19 [==============================] - 41s 2s/step - loss: 0.3085 - accuracy: 0.8969 - val_loss: 1.3834 -
     val_accuracy: 0.5458
Epoch 11/25
19/19 [==============================] - 41s 2s/step - loss: 0.2856 - accuracy: 0.9114 - val_loss: 1.7234 -
     val_accuracy: 0.3667
Epoch 12/25
19/19 [==============================] - 41s 2s/step - loss: 0.3192 - accuracy: 0.9097 - val_loss: 3.1606 -
     val_accuracy: 0.2833
Epoch 13/25
...
Epoch 24/25
19/19 [==============================] - 38s 2s/step - loss: 0.1513 - accuracy: 0.9549 - val_loss: 5.1007 -
     val_accuracy: 0.3292
Epoch 25/25
19/19 [==============================] - 38s 2s/step - loss: 0.1723 - accuracy: 0.9412 - val_loss: 0.8141 -
     val_accuracy: 0.7667
```

Così come nei precedenti due esperimenti non siamo riusciti ad ottenere un miglioramento determinanente, ottenendo sempre gli stessi valori per le metriche analizzate.

Osservando nel dettaglio la matrice di confusione risalta la difficoltà del modello di classificare correttamente strumenti musicali Handpan e Drum.



(a) (b)

(c)

Figure 5.3: Grafici modello terzo esperimento

Table 5.7: Classification Report terzo esperimento

Class	**Precision**	**Recall**	**F1-score**	**Support**
Drum	0.00	0.00	0.00	20
Electric guitar	0.21	0.17	0.19	18
Handpan	0.00	0.00	0.00	20
Keyboard	0.15	0.20	0.17	20
Piano	0.07	0.10	0.09	20
Trumpet	0.11	0.10	0.10	20
Tuba	0.11	0.10	0.11	20
Wind gong	0.16	0.15	0.15	20
accuracy	0.10			158
macro avg	0.10	0.10	0.10	158
weighted avg	0.10	0.10	0.10	158

5.4 Quarto esperimento

Table 5.8: Modello quarto esperimento

Layer (type)	**Output Shape**	**Param #**
conv2d_46 (Conv2D)	(None, 126, 126, 128)	3584
max_pooling2d_46 (MaxPooling2D)	(None, 63, 63, 128)	0
conv2d_47 (Conv2D)	(None, 61, 61, 64)	73792
max_pooling2d_47 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_48 (Conv2D)	(None, 28, 28, 64)	36928
max_pooling2d_48 (MaxPooling2D)	(None, 14, 14, 64)	0
batch_normalization_4 (BatchNormalization)	(None, 14, 14, 64)	256
flatten_23 (Flatten)	(None, 12544)	0
dense_52 (Dense)	(None, 64)	802880
dense_53 (Dense)	(None, 8)	520
Total params		917,960
Trainable params		917,832
Non-trainable params		128

Nel quarto esperimento sono stati aggiunti ulteriori layer di convoluzione e pooling mantenendo lo stesso un layer di Batch normalization.

Output epochs:

```
{
Epoch 1/25
19/19 [==============================] - 40s 2s/step - loss: 1.2236 - accuracy: 0.5920 - val_loss: 2.1955 -
     val_accuracy: 0.2667
Epoch 2/25
19/19 [==============================] - 38s 2s/step - loss: 0.6663 - accuracy: 0.7853 - val_loss: 2.3700 -
     val_accuracy: 0.2417
Epoch 3/25
19/19 [==============================] - 38s 2s/step - loss: 0.4958 - accuracy: 0.8458 - val_loss: 2.8546 -
     val_accuracy: 0.2375
Epoch 4/25
19/19 [==============================] - 38s 2s/step - loss: 0.4933 - accuracy: 0.8467 - val_loss: 2.5710 -
     val_accuracy: 0.2500
Epoch 5/25
19/19 [==============================] - 37s 2s/step - loss: 0.3797 - accuracy: 0.8790 - val_loss: 1.5777 -
     val_accuracy: 0.4292
Epoch 6/25
19/19 [==============================] - 38s 2s/step - loss: 0.3194 - accuracy: 0.8935 - val_loss: 4.3569 -
     val_accuracy: 0.2333
Epoch 7/25
19/19 [==============================] - 38s 2s/step - loss: 0.3700 - accuracy: 0.8952 - val_loss: 1.5563 -
     val_accuracy: 0.3542
Epoch 8/25
19/19 [==============================] - 38s 2s/step - loss: 0.3426 - accuracy: 0.8927 - val_loss: 1.5512 -
     val_accuracy: 0.4708
```

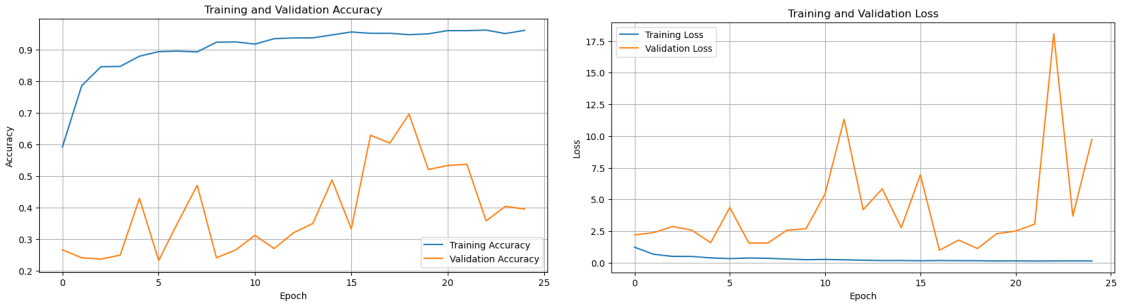
```
Epoch 9/25
19/19 [==============================] - 38s 2s/step - loss: 0.2882 - accuracy: 0.9233 - val_loss: 2.5636 -
     val_accuracy: 0.2417
Epoch 10/25
19/19 [==============================] - 37s 2s/step - loss: 0.2369 - accuracy: 0.9242 - val_loss: 2.6817 -
     val_accuracy: 0.2667
Epoch 11/25
19/19 [==============================] - 38s 2s/step - loss: 0.2575 - accuracy: 0.9174 - val_loss: 5.4313 -
     val_accuracy: 0.3125
Epoch 12/25
19/19 [==============================] - 38s 2s/step - loss: 0.2284 - accuracy: 0.9344 - val_loss: 11.3268 -
     val_accuracy: 0.2708
Epoch 13/25
...
Epoch 24/25
19/19 [==============================] - 38s 2s/step - loss: 0.1426 - accuracy: 0.9506 - val_loss: 3.6766 -
     val_accuracy: 0.4042
Epoch 25/25
19/19 [==============================] - 37s 2s/step - loss: 0.1348 - accuracy: 0.9608 - val_loss: 9.7281 -
     val_accuracy: 0.3958
}
```

L'aumento della complessità del modello ha portato ad avere ottimi risultati nel training ma pessimi nella validation. E' infatti possibile notare come il modello non sia in grado di generalizzare nella validation portandolo a non convergere ed ad avere un val_accuracy molto bassa.

Inoltre ciò è trasmesso anche nel testing dove manteniamo un F1 macro score sempre basso.

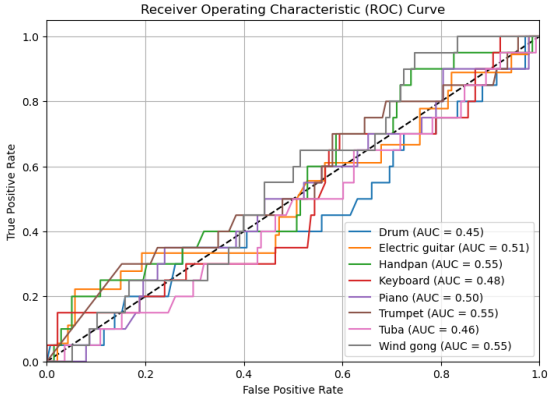
Table 5.9: Classification Report quarto esperimento

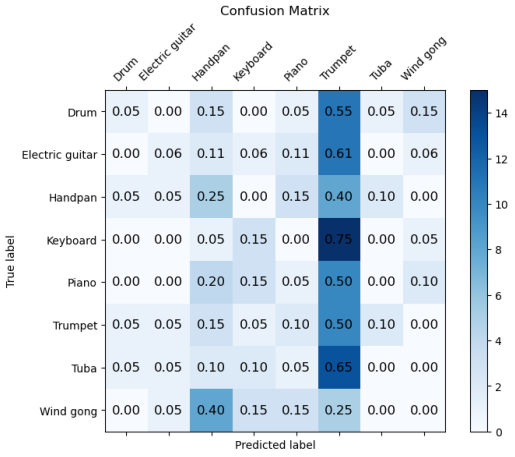
Class	**Precision**	**Recall**	**F1-Score**	**Support**
Drum	0.25	0.05	0.08	20
Electric guitar	0.20	0.06	0.09	18
Handpan	0.18	0.25	0.21	20
Keyboard	0.23	0.15	0.18	20
Piano	0.08	0.05	0.06	20
Trumpet	0.12	0.50	0.19	20
Tuba	0.00	0.00	0.00	20
Wind gong	0.00	0.00	0.00	20
Accuracy			0.13	158
Macro avg	0.13	0.13	0.10	158
Weighted avg	0.13	0.13	0.10	158



(a)

(b)





(c)

Figure 5.4: Grafici modello quarto esperimento

5.5 Quinto esperimento

Table 5.10: Modello quinto esperimento

Layer (type)	**Output Shape**	**Param #**
conv2d_22 (Conv2D)	(None, 126, 126, 128)	3584
max_pooling2d_22 (MaxPooling2D)	(None, 63, 63, 128)	0
dropout_4 (Dropout)	(None, 63, 63, 128)	0
conv2d_23 (Conv2D)	(None, 61, 61, 64)	73792
max_pooling2d_23 (MaxPooling2D)	(None, 30, 30, 64)	0
dropout_5 (Dropout)	(None, 30, 30, 64)	0
conv2d_24 (Conv2D)	(None, 28, 28, 64)	36928
max_pooling2d_24 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_6 (Dropout)	(None, 14, 14, 64)	0
flatten_11 (Flatten)	(None, 12544)	0
dense_22 (Dense)	(None, 64)	802880
dense_23 (Dense)	(None, 32)	2080
dense_24 (Dense)	(None, 8)	264
Total params		919,528
Trainable params		919,528
Non-trainable params		0

Considerando la presenza di overfitting nonostante l'integrazione di Batch normalization abbiamo deciso di affrontare il problema aggiungendo dei dropout tra gli strati di convoluzione al fine di riuscire a ridurlo.

Output epochs:

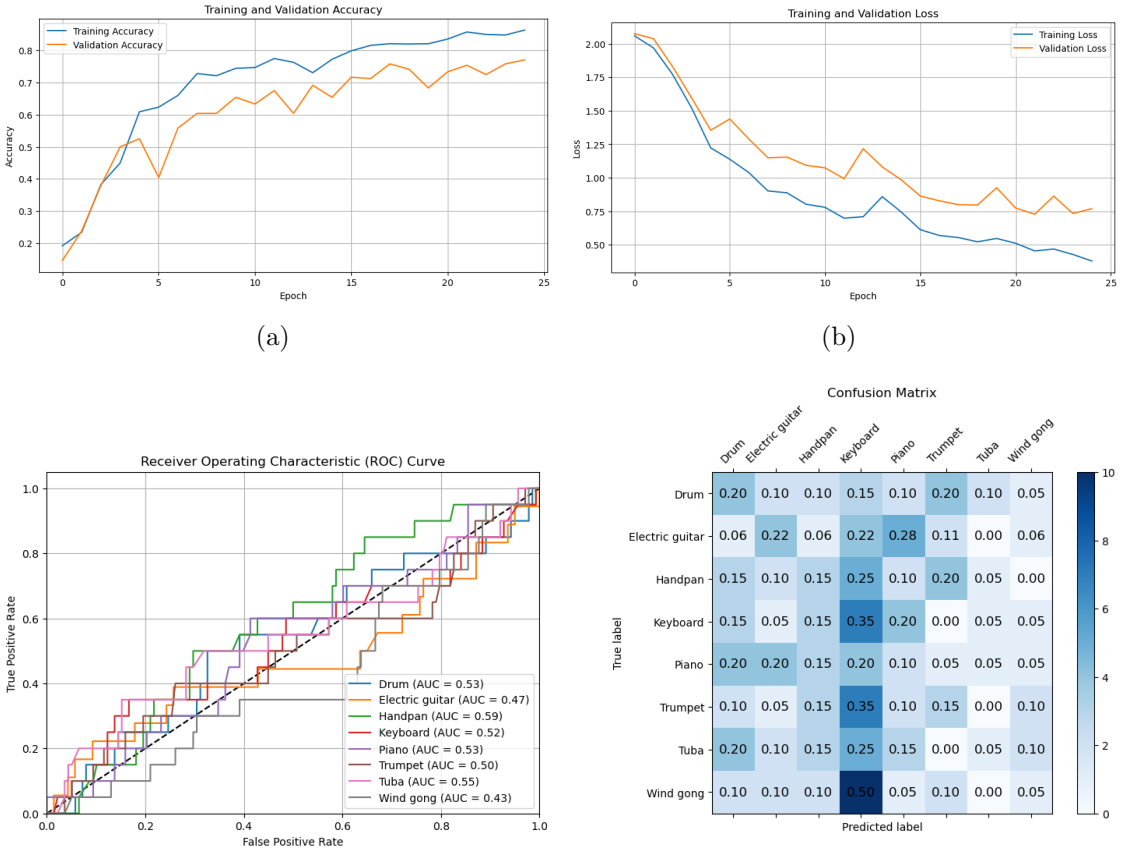
```
{
Epoch 1/25
19/19 [==============================] - 40s 2s/step - loss: 2.0593 - accuracy: 0.1917 - val_loss: 2.0752 -
     val_accuracy: 0.1458
Epoch 2/25
19/19 [==============================] - 33s 2s/step - loss: 1.9685 - accuracy: 0.2334 - val_loss: 2.0375 -
     val_accuracy: 0.2375
Epoch 3/25
19/19 [==============================] - 33s 2s/step - loss: 1.7711 - accuracy: 0.3825 - val_loss: 1.8294 -
     val_accuracy: 0.3792
Epoch 4/25
19/19 [==============================] - 34s 2s/step - loss: 1.5204 - accuracy: 0.4497 - val_loss: 1.5953 -
     val_accuracy: 0.5000
Epoch 5/25
19/19 [==============================] - 34s 2s/step - loss: 1.2228 - accuracy: 0.6090 - val_loss: 1.3548 -
     val_accuracy: 0.5250
Epoch 6/25
19/19 [==============================] - 34s 2s/step - loss: 1.1383 - accuracy: 0.6235 - val_loss: 1.4387 -
     val_accuracy: 0.4042
```

```
Epoch 7/25
19/19 [==============================] - 34s 2s/step - loss: 1.0387 - accuracy: 0.6601 - val_loss: 1.2892 -
     val_accuracy: 0.5583
Epoch 8/25
19/19 [==============================] - 34s 2s/step - loss: 0.9020 - accuracy: 0.7283 - val_loss: 1.1496 -
     val_accuracy: 0.6042
Epoch 9/25
19/19 [==============================] - 33s 2s/step - loss: 0.8879 - accuracy: 0.7215 - val_loss: 1.1540 -
     val_accuracy: 0.6042
Epoch 10/25
19/19 [==============================] - 30s 2s/step - loss: 0.8027 - accuracy: 0.7445 - val_loss: 1.0935 -
     val_accuracy: 0.6542
Epoch 11/25
19/19 [==============================] - 30s 2s/step - loss: 0.7794 - accuracy: 0.7470 - val_loss: 1.0741 -
     val_accuracy: 0.6333
Epoch 12/25
19/19 [==============================] - 30s 2s/step - loss: 0.6985 - accuracy: 0.7751 - val_loss: 0.9934 -
     val_accuracy: 0.6750
Epoch 13/25
...
Epoch 24/25
19/19 [==============================] - 35s 2s/step - loss: 0.4280 - accuracy: 0.8484 - val_loss: 0.7338 -
     val_accuracy: 0.7583
Epoch 25/25
19/19 [==============================] - 32s 2s/step - loss: 0.3790 - accuracy: 0.8637 - val_loss: 0.7696 -
     val_accuracy: 0.7708
}
```

L'utilizzo del dropout ha permesso di aumentare nuovamente la val_accuracy, tuttavia nella fase di testing abbiamo nuovamente ottenuto uno scarso risultato mostrato nel Classification Report.

Table 5.11: Classification Report quinto esperimento

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Drum	0.17	0.20	0.19	20
Electric guitar	0.22	0.22	0.22	18
Handpan	0.15	0.15	0.15	20
Keyboard	0.16	0.35	0.22	20
Piano	0.10	0.10	0.10	20
Trumpet	0.19	0.15	0.17	20
Tuba	0.17	0.05	0.08	20
Wind gong	0.11	0.05	0.07	20
Accuracy			0.16	158
Macro avg	0.16	0.16	0.15	158
Weighted avg	0.16	0.16	0.15	158



(a) (b)

(c)

Figure 5.5: Grafici modello quinto esperimento

5.6 Sesto esperimento

Table 5.12: Modello sesto esperimento

Layer (type)	**Output Shape**	**Param #**
conv2d_52 (Conv2D)	(None, 126, 126, 256)	7168
max_pooling2d_52 (MaxPooling2D)	(None, 63, 63, 256)	0
conv2d_53 (Conv2D)	(None, 61, 61, 128)	295040
max_pooling2d_53 (MaxPooling2D)	(None, 30, 30, 128)	0
conv2d_54 (Conv2D)	(None, 28, 28, 128)	147584
max_pooling2d_54 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_55 (Conv2D)	(None, 12, 12, 64)	73792
max_pooling2d_55 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten_25 (Flatten)	(None, 2304)	0
dense_57 (Dense)	(None, 128)	295040
dense_58 (Dense)	(None, 32)	4128
dense_59 (Dense)	(None, 8)	264
Total params		823,016
Trainable params		823,016
Non-trainable params		0

In questo esperimento abbiamo deciso di rimuovere i dropout ed, invece di toccare il numero di layer, abbiamo deciso di aumentare il numero di kernels e neuroni per osservare il comportamento del modello.

Output epochs:

```
{
Epoch 1/25
19/19 [==============================] - 76s 4s/step - loss: 2.0294 - accuracy: 0.1763 - val_loss: 2.0395 -
     val_accuracy: 0.2375
Epoch 2/25
19/19 [==============================] - 74s 4s/step - loss: 1.8329 - accuracy: 0.3092 - val_loss: 1.9677 -
     val_accuracy: 0.1500
Epoch 3/25
19/19 [==============================] - 73s 4s/step - loss: 1.6607 - accuracy: 0.3543 - val_loss: 1.6778 -
     val_accuracy: 0.2833
Epoch 4/25
19/19 [==============================] - 73s 4s/step - loss: 1.3708 - accuracy: 0.4847 - val_loss: 1.5260 -
     val_accuracy: 0.4750
Epoch 5/25
19/19 [==============================] - 74s 4s/step - loss: 1.2180 - accuracy: 0.5818 - val_loss: 1.4605 -
     val_accuracy: 0.4458
Epoch 6/25
19/19 [==============================] - 73s 4s/step - loss: 1.3167 - accuracy: 0.5213 - val_loss: 1.5203 -
     val_accuracy: 0.3833
```

```
Epoch 7/25
19/19 [==============================] - 73s 4s/step - loss: 1.1315 - accuracy: 0.5963 - val_loss: 1.3131 -
     val_accuracy: 0.5167
Epoch 8/25
19/19 [==============================] - 73s 4s/step - loss: 0.9647 - accuracy: 0.6806 - val_loss: 1.2872 -
     val_accuracy: 0.5667
Epoch 9/25
19/19 [==============================] - 73s 4s/step - loss: 0.8151 - accuracy: 0.7428 - val_loss: 1.1370 -
     val_accuracy: 0.5875
Epoch 10/25
19/19 [==============================] - 72s 4s/step - loss: 0.7661 - accuracy: 0.7717 - val_loss: 1.2227 -
     val_accuracy: 0.5792
Epoch 11/25
19/19 [==============================] - 73s 4s/step - loss: 0.7308 - accuracy: 0.7785 - val_loss: 1.0199 -
     val_accuracy: 0.7208
Epoch 12/25
19/19 [==============================] - 72s 4s/step - loss: 0.6820 - accuracy: 0.7956 - val_loss: 0.9363 -
     val_accuracy: 0.7000
Epoch 13/25
...
Epoch 24/25
19/19 [==============================] - 72s 4s/step - loss: 0.2662 - accuracy: 0.9208 - val_loss: 0.5700 -
     val_accuracy: 0.8167
Epoch 25/25
19/19 [==============================] - 71s 4s/step - loss: 0.3013 - accuracy: 0.9063 - val_loss: 0.6109 -
     val_accuracy: 0.8417
}
```

Anche qui siamo riusciti ad ottenere ottimi risultati in fase di training e validation, evidenziati dall'alto accuracy di 90.63% e $\text{val}_{a}ccuracydi84.17\%, mantenendoalcontempounpessi$

Table 5.13: Classification Report sesto esperimento

Class	**Precision**	**Recall**	**F1-score**	**Support**
Drum	0.11	0.07	0.08	30
Electric guitar	0.13	0.13	0.13	30
Handpan	0.11	0.13	0.12	30
Keyboard	0.09	0.10	0.10	30
Piano	0.08	0.07	0.07	30
Trumpet	0.05	0.07	0.06	30
Tuba	0.21	0.20	0.21	30
Wind gong	0.03	0.03	0.03	30
Accuracy			0.10	240
Macro avg	0.10	0.10	0.10	240
Weighted avg	0.10	0.10	0.10	240

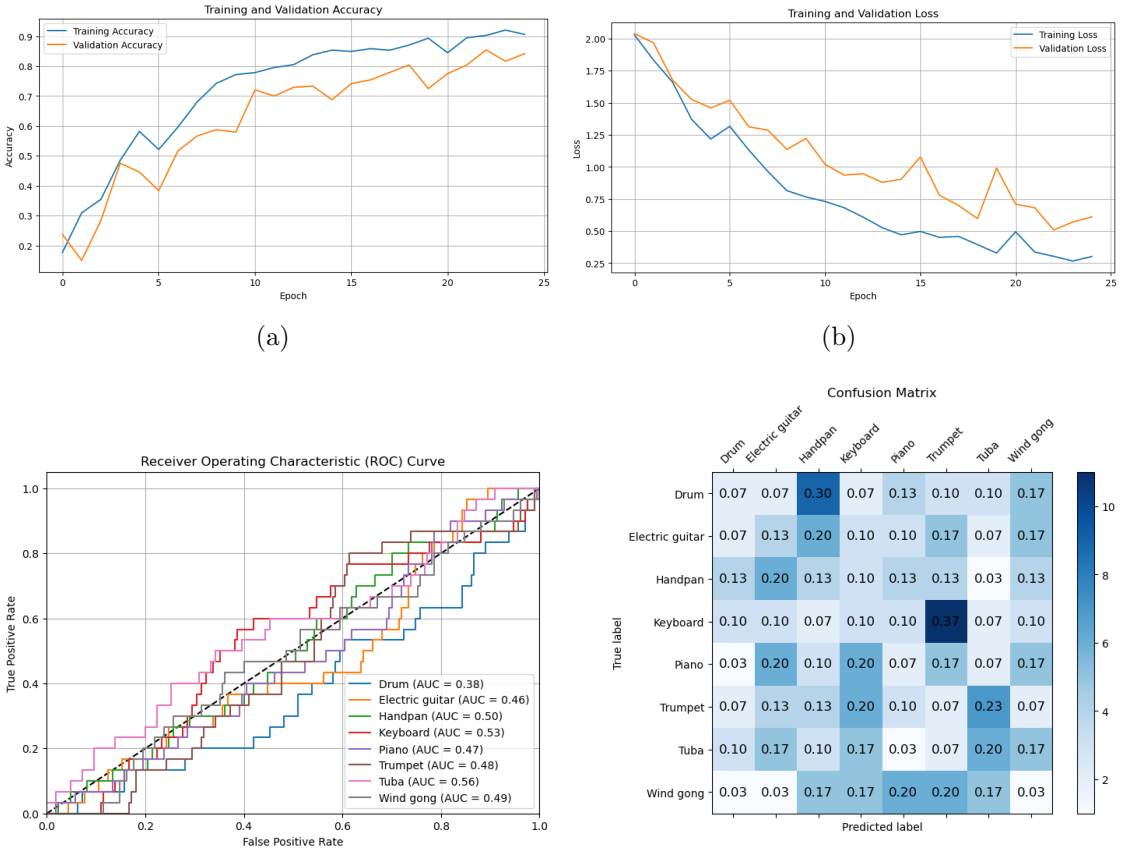


Figure 5.6: Grafici modello sesto esperimento

5.7 Settimo esperimento

Table 5.14: Modello settimo esperimento

Layer (type)	**Output Shape**	**Param #**
conv2d_56 (Conv2D)	(None, 126, 126, 256)	7168
max_pooling2d_56 (MaxPooling2D)	(None, 63, 63, 256)	0
conv2d_57 (Conv2D)	(None, 61, 61, 128)	295040
max_pooling2d_57 (MaxPooling2D)	(None, 30, 30, 128)	0
conv2d_58 (Conv2D)	(None, 28, 28, 64)	73792
max_pooling2d_58 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_59 (Conv2D)	(None, 12, 12, 64)	36928
max_pooling2d_59 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten_26 (Flatten)	(None, 2304)	0
dense_60 (Dense)	(None, 64)	147520
dense_61 (Dense)	(None, 32)	2080
dense_62 (Dense)	(None, 8)	264
Total params		562,792
Trainable params		562,792
Non-trainable params		0

Visto l'insuccesso di aumentare il numero di kernel e neuroni, qui abbiamo deciso di diminuire di quantità per notare il comportamento.

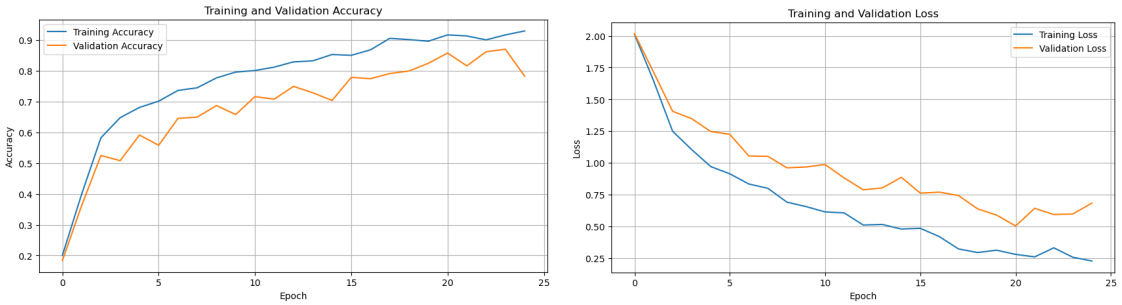
Output epochs:

```
{
Epoch 1/25
19/19 [==============================] - 61s 3s/step - loss: 2.0120 - accuracy: 0.2002 - val_loss: 2.0171 -
     val_accuracy: 0.1833
Epoch 2/25
19/19 [==============================] - 59s 3s/step - loss: 1.6464 - accuracy: 0.3995 - val_loss: 1.7148 -
     val_accuracy: 0.3625
Epoch 3/25
19/19 [==============================] - 59s 3s/step - loss: 1.2493 - accuracy: 0.5826 - val_loss: 1.4063 -
     val_accuracy: 0.5250
Epoch 4/25
19/19 [==============================] - 59s 3s/step - loss: 1.1051 - accuracy: 0.6482 - val_loss: 1.3485 -
     val_accuracy: 0.5083
Epoch 5/25
19/19 [==============================] - 59s 3s/step - loss: 0.9713 - accuracy: 0.6814 - val_loss: 1.2473 -
     val_accuracy: 0.5917
Epoch 6/25
19/19 [==============================] - 59s 3s/step - loss: 0.9145 - accuracy: 0.7019 - val_loss: 1.2252 -
     val_accuracy: 0.5583
Epoch 7/25
```

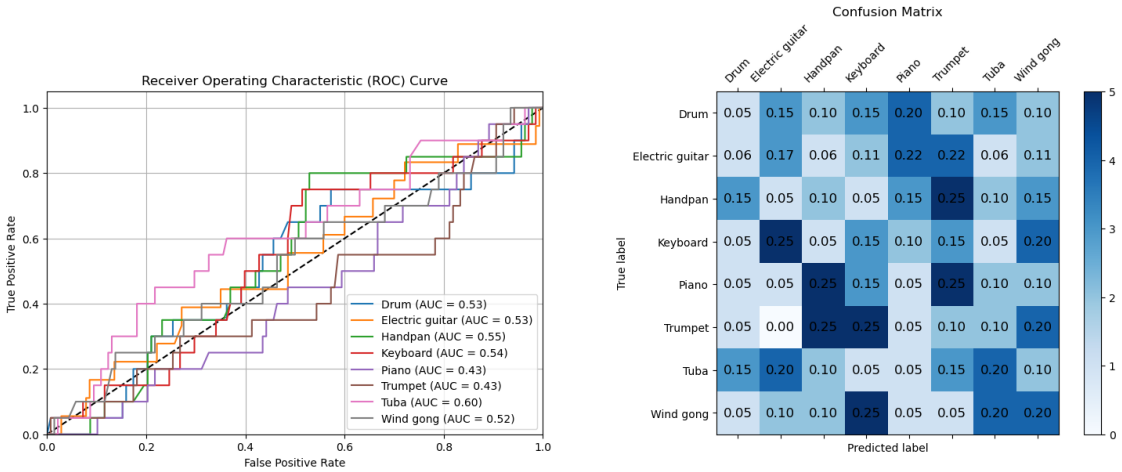
```
19/19 [==============================] - 59s 3s/step - loss: 0.8343 - accuracy: 0.7368 - val_loss: 1.0546 -
     val_accuracy: 0.6458
Epoch 8/25
19/19 [==============================] - 59s 3s/step - loss: 0.8003 - accuracy: 0.7453 - val_loss: 1.0516 -
     val_accuracy: 0.6500
Epoch 9/25
19/19 [==============================] - 59s 3s/step - loss: 0.6914 - accuracy: 0.7777 - val_loss: 0.9614 -
     val_accuracy: 0.6875
Epoch 10/25
19/19 [==============================] - 58s 3s/step - loss: 0.6565 - accuracy: 0.7964 - val_loss: 0.9679 -
     val_accuracy: 0.6583
Epoch 11/25
19/19 [==============================] - 59s 3s/step - loss: 0.6148 - accuracy: 0.8015 - val_loss: 0.9877 -
     val_accuracy: 0.7167
Epoch 12/25
19/19 [==============================] - 59s 3s/step - loss: 0.6069 - accuracy: 0.8126 - val_loss: 0.8822 -
     val_accuracy: 0.7083
Epoch 13/25
...
Epoch 24/25
19/19 [==============================] - 59s 3s/step - loss: 0.2579 - accuracy: 0.9174 - val_loss: 0.5983 -
     val_accuracy: 0.8708
Epoch 25/25
19/19 [==============================] - 59s 3s/step - loss: 0.2282 - accuracy: 0.9302 - val_loss: 0.6839 -
     val_accuracy: 0.7833
}
```

In questo modello abbiamo notato un leggero overfitting nel train mantenendo comunque buoni risultati nella validation.
Come già visto negli scorsi esperimenti tuttavia l'incapacità di generalizzare del modello resta lo stesso abbastanza marcata (come sempre visibile nei grafici e metriche proposti).

Quest'ultimo risultato, insieme agli altri esperimenti, ci ha portato ad abbandonare questa strada cercando altre strategie da poter adottare come mostrato nei prossimi esperimenti.



(a) (b)



(c)

Figure 5.7: Grafici modello settimo esperimento

Table 5.15: Classification Report settimo esperimento

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Drum	0.08	0.05	0.06	20
Electric guitar	0.16	0.17	0.16	18
Handpan	0.10	0.10	0.10	20
Keyboard	0.13	0.15	0.14	20
Piano	0.06	0.05	0.05	20
Trumpet	0.08	0.10	0.09	20
Tuba	0.21	0.20	0.21	20
Wind gong	0.17	0.20	0.19	20
Accuracy			0.13	158
Macro avg	0.12	0.13	0.12	158
Weighted avg	0.12	0.13	0.12	158

5.8 Ottavo esperimento

Table 5.16: Model ottavo esperimento

Layer (type)	**Output Shape**	**Param #**
efficientnetb3 (Functional)	(None, 1536)	10,783,535
dense$_6$3(*Dense*)	(None, 8)	12,296
Total params		10,795,831
Trainable params		10,708,528
Non-trainable params		87,303

Un'altra strada che abbiamo provato ad intraprendere è stata quella di prova un modello pre-addestrato, EfficientNetB0, facendone fine tuning.

EfficientNetB0 [13] è un'architettura leggera e altamente efficace di reti neurali convoluzionali, che ha guadagnato popolarità grazie al suo uso efficiente delle risorse e alle prestazioni competitive. Queste motivazioni ci hanno indotto a fare una prova adattandolo al problema proposto. Per fare ciò è stato semplicemente aggiunto un Batch normalization ed un layer finale con 8 neuroni e la funzione di attivazione SoftMax.

Inoltre abbiamo adottato, come nei precedenti esperimenti, la funzione di ottimizzazione adam e funzione loss Cross entropy.

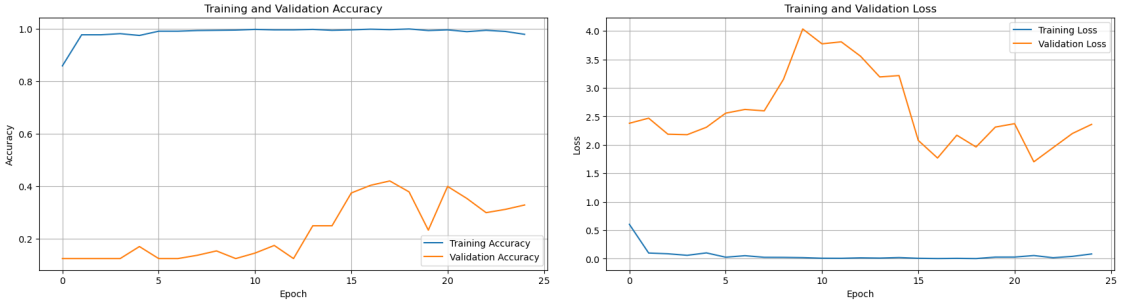
Output epochs:

```
{
Epoch 1/25
19/19 [==============================] - 130s 5s/step - loss: 0.6031 - accuracy: 0.8595 - val_loss: 2.3796 -
     val_accuracy: 0.1250
Epoch 2/25
19/19 [==============================] - 101s 5s/step - loss: 0.0998 - accuracy: 0.9779 - val_loss: 2.4688 -
     val_accuracy: 0.1250
Epoch 3/25
19/19 [==============================] - 102s 5s/step - loss: 0.0864 - accuracy: 0.9779 - val_loss: 2.1869 -
     val_accuracy: 0.1250
Epoch 4/25
19/19 [==============================] - 101s 5s/step - loss: 0.0596 - accuracy: 0.9821 - val_loss: 2.1786 -
     val_accuracy: 0.1250
Epoch 5/25
19/19 [==============================] - 101s 5s/step - loss: 0.1020 - accuracy: 0.9753 - val_loss: 2.3093 -
     val_accuracy: 0.1708
Epoch 6/25
```

```
19/19 [==============================] - 101s 5s/step - loss: 0.0262 - accuracy: 0.9915 - val_loss: 2.5571 -
     val_accuracy: 0.1250
Epoch 7/25
19/19 [==============================] - 101s 5s/step - loss: 0.0528 - accuracy: 0.9915 - val_loss: 2.6221 -
     val_accuracy: 0.1250
Epoch 8/25
19/19 [==============================] - 101s 5s/step - loss: 0.0247 - accuracy: 0.9940 - val_loss: 2.5975 -
     val_accuracy: 0.1375
Epoch 9/25
19/19 [==============================] - 101s 5s/step - loss: 0.0236 - accuracy: 0.9949 - val_loss: 3.1516 -
     val_accuracy: 0.1542
Epoch 10/25
19/19 [==============================] - 101s 5s/step - loss: 0.0190 - accuracy: 0.9957 - val_loss: 4.0358 -
     val_accuracy: 0.1250
Epoch 11/25
19/19 [==============================] - 101s 5s/step - loss: 0.0082 - accuracy: 0.9983 - val_loss: 3.7748 -
     val_accuracy: 0.1458
Epoch 12/25
19/19 [==============================] - 101s 5s/step - loss: 0.0070 - accuracy: 0.9966 - val_loss: 3.8102 -
     val_accuracy: 0.1750
Epoch 13/25
...
Epoch 24/25
19/19 [==============================] - 101s 5s/step - loss: 0.0407 - accuracy: 0.9906 - val_loss: 2.1994 -
     val_accuracy: 0.3125
Epoch 25/25
19/19 [==============================] - 101s 5s/step - loss: 0.0834 - accuracy: 0.9796 - val_loss: 2.3602 -
     val_accuracy: 0.3292
}
```

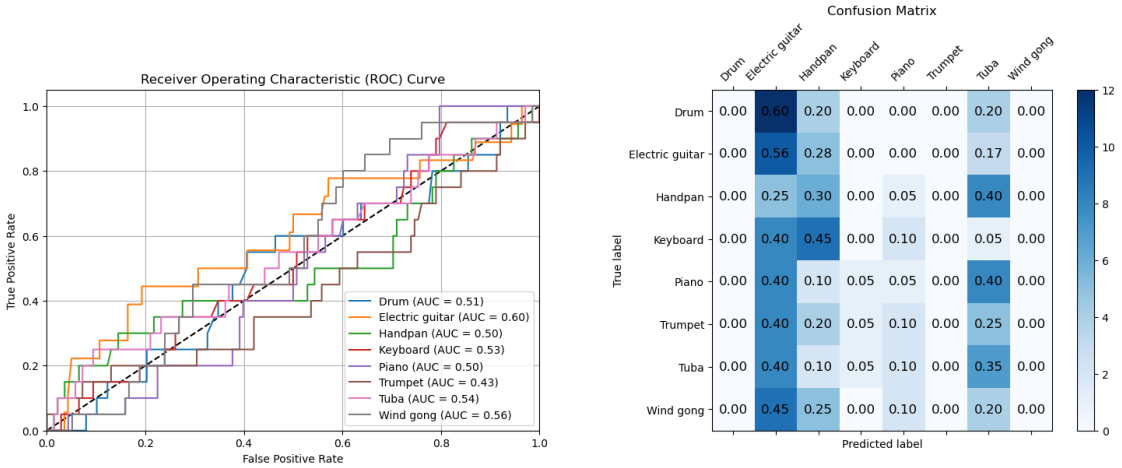
I risultati ci hanno fatto maggiormente capire, insieme anche ai precedenti esperimenti, che i modelli proposti fin'ora sono troppo complessi per il problema proposto, il che porta ad overfitting.

In questo particolare esperimento riusciamo infatti a notare come l'overfitting sia molto accentuato a causa dell'alta complessità del modello utilizzato.



(a)

(b)



(c)

Figure 5.8: Grafici modello ottavo esperimento

Table 5.17: Classification Report ottavo esperimento

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Drum	0.00	0.00	0.00	20
Electric guitar	0.15	0.56	0.23	18
Handpan	0.16	0.30	0.21	20
Keyboard	0.00	0.00	0.00	20
Piano	0.10	0.05	0.07	20
Trumpet	0.00	0.00	0.00	20
Tuba	0.17	0.35	0.23	20
Wind gong	0.00	0.00	0.00	20
Accuracy			0.15	158
Macro avg	0.07	0.16	0.09	158
Weighted avg	0.07	0.15	0.09	158

5.9 Nono esperimento

Table 5.18: Modello nono esperimento

Layer (type)	**Output Shape**	**Param #**
conv2d	(None, 125, 125, 64)	3136
max_pooling2d	(None, 62, 62, 64)	0
flatten	(None, 246016)	0
dense	(None, 32)	7872544
dense_1	(None, 8)	264
Total params		7,875,944
Trainable params		7,875,944
Non-trainable params		0

Data la conferma avuta nell'ultimo esperimento e dai precedenti della costante presenza in overfitting, in questo eperimento abbiamo deciso di fare marcia indietro riprendendo il primo esperimento e diminuendogli la complessità al fine di contrastare l'overfitting.

L'idea dietro è che i modelli proposti siano troppo elaborati per il problema di classificazione che stiamo affrontando.

A tal fine abbiamo diminuito il numero di kernel del primo layer aumentando al contempo le dimensioni della base. Inoltre abbiamo anche diminuito il numero di neuroni nell'ultimo strato.

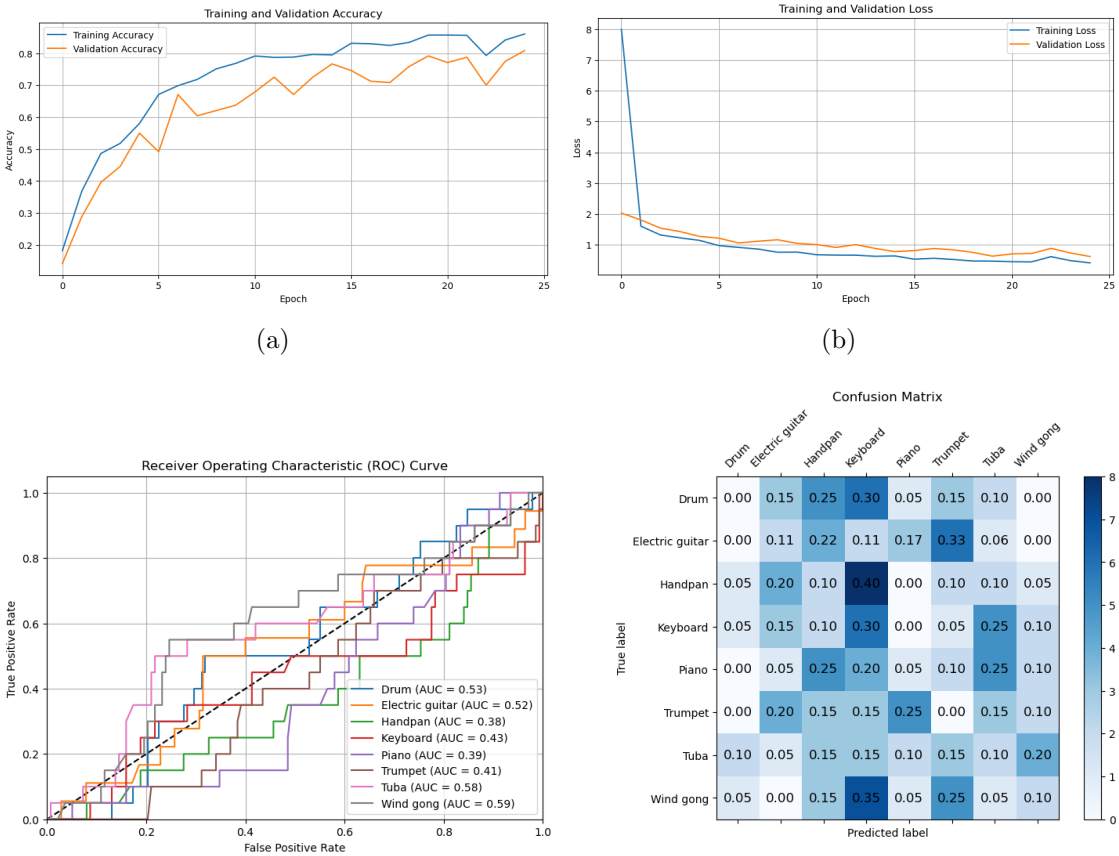
Output epochs:

```
{
Epoch 1/25
19/19 [==============================] - 20s 1s/step - loss: 7.9857 - accuracy: 0.1823 - val_loss: 2.0306 -
     val_accuracy: 0.1417
Epoch 2/25
19/19 [==============================] - 12s 618ms/step - loss: 1.6041 - accuracy: 0.3671 - val_loss: 1.8077 -
     val_accuracy: 0.2875
Epoch 3/25
19/19 [==============================] - 12s 618ms/step - loss: 1.3204 - accuracy: 0.4864 - val_loss: 1.5413 -
     val_accuracy: 0.3958
Epoch 4/25
19/19 [==============================] - 12s 616ms/step - loss: 1.2273 - accuracy: 0.5179 - val_loss: 1.4319 -
     val_accuracy: 0.4458
Epoch 5/25
```

```
19/19 [==============================] - 12s 617ms/step - loss: 1.1451 - accuracy: 0.5801 - val_loss: 1.2744 -
     val_accuracy: 0.5500
Epoch 6/25
19/19 [==============================] - 12s 640ms/step - loss: 0.9741 - accuracy: 0.6712 - val_loss: 1.2152 -
     val_accuracy: 0.4917
Epoch 7/25
19/19 [==============================] - 13s 665ms/step - loss: 0.9184 - accuracy: 0.6985 - val_loss: 1.0616 -
     val_accuracy: 0.6708
Epoch 8/25
19/19 [==============================] - 13s 690ms/step - loss: 0.8625 - accuracy: 0.7181 - val_loss: 1.1184 -
     val_accuracy: 0.6042
Epoch 9/25
19/19 [==============================] - 13s 656ms/step - loss: 0.7607 - accuracy: 0.7513 - val_loss: 1.1653 -
     val_accuracy: 0.6208
Epoch 10/25
19/19 [==============================] - 13s 673ms/step - loss: 0.7654 - accuracy: 0.7683 - val_loss: 1.0469 -
     val_accuracy: 0.6375
Epoch 11/25
19/19 [==============================] - 12s 640ms/step - loss: 0.6801 - accuracy: 0.7913 - val_loss: 1.0099 -
     val_accuracy: 0.6792
Epoch 12/25
19/19 [==============================] - 12s 649ms/step - loss: 0.6674 - accuracy: 0.7871 - val_loss: 0.9170 -
     val_accuracy: 0.7250
Epoch 13/25
...
Epoch 24/25
19/19 [==============================] - 12s 642ms/step - loss: 0.4876 - accuracy: 0.8416 - val_loss: 0.7367 -
     val_accuracy: 0.7750
Epoch 25/25
19/19 [==============================] - 12s 656ms/step - loss: 0.4176 - accuracy: 0.8603 - val_loss: 0.6220 -
     val_accuracy: 0.8083
}
```

Nonostante abbiamo diminuito la complessità del modello siamo lo stesso giunti ad overfitting, mostrato dagli ottimi risultati nella fase di training e validation e bassi risultati ottenuti nella fase di testing accentuati dalle metriche e grafici proposti.

In particolare, è possibile notare un peggioramento del F1 macro score rispetto al primo esperimento, motivo che ci ha fatto arrivare alla conclusione che il problema principale non risiede nel modello ma nella scarsità di samples presenti nel dataset.



(a) (b)

(c)

Figure 5.9: Grafici modello nono esperimento

Table 5.19: Classification Report nono esperimento

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Drum	0.00	0.00	0.00	20
Electric guitar	0.11	0.11	0.11	18
Handpan	0.07	0.10	0.09	20
Keyboard	0.15	0.30	0.20	20
Piano	0.08	0.05	0.06	20
Trumpet	0.00	0.00	0.00	20
Tuba	0.10	0.10	0.10	20
Wind gong	0.15	0.10	0.12	20
Accuracy			0.09	158
Macro Avg	0.08	0.10	0.08	158
Weighted Avg	0.08	0.09	0.08	158

Chapter 6

Demo

La demo proposta si compone di un notebook e di un modello salvato in formato .h5.

All'interno del notebook viene caricato in memoria il modello *modello_primo_esperimento.h5* che in fase di sperimentazione ha dato i risultati migliori grazie alla funzione *load_model()* della libreria Keras [12].

Successivamente è definita la seguente funzione:

```
def predict_instrument(input_image_path):
# Imposta le classi delle etichette (assicurati che
   siano nello stesso ordine utilizzato durante
   l'addestramento)
class_labels = ["Drum", "Electric guitar", "Handpan",
   "Keyboard", "Piano", "Trumpet", "Tuba", "Wind gong"]

image = Image.open(input_image_path)
# Ridimensiona l'immagine alla dimensione di input del
   modello
image = image.resize((128, 128))
# Converti l'immagine in un array numpy

image_array = np.array(image)
image_array = np.expand_dims(image_array, axis=0)
image_array = image_array / 255.0

prediction = model.predict(image_array)[0]
```

```
predicted_class_index = np.argmax(prediction)
predicted_class_label =
   class_labels[predicted_class_index]

# Mostra l'immagine e la classe predetta
image = load_img(input_image_path)
plt.figure()
plt.imshow(image)
plt.title(f"Predicted Class: {predicted_class_label}")
plt.axis('off')
plt.show()
```

La funzione permette di classificare un'immagine data il percorso del file tramite il suo unico parametro "input_image_path". Prima di effettuare la predizione, l'immagine viene sottoposta a un preprocessing necessario, che include il ridimensionamento alla dimensione 128x128 pixel e il rescaling dei valori dei pixel per normalizzarli nel range [0, 1].

Una volta completato il preprocessing, il modello effettua la predizione sull'immagine e restituisce come output l'immagine con la classe predetta.

L'utilizzo della funzione e del notebook è molto semplice. Un esempio comune è scaricare o ottenere un'immagine contenente uno strumento musicale da classificare e specificarne il percorso come input alla funzione. Tuttavia, è importante notare che la funzione di predizione ha una condizione fondamentale: può classificare solo immagini che siano uguali o superiori alla dimensione di 128x128 pixel.

Per riassumere, la funzione fornisce un'interfaccia facile da usare per la classificazione di immagini, ma è necessario assicurarsi che l'immagine in input rispetti la dimensione minima richiesta per il corretto funzionamento della funzione di predizione.

Chapter 7

Codice

In questo capitolo verrano affrontati le parti principali dei codici consegnati, tuttavia, per evitare di allungare troppo la relazione ed evitare ripetizioni non verranno scritti qui i codici completi, essi sono reperibili alla seguente repository[6]

7.1 Notebook CNN

Il notebook CNN è il cuore principale del progetto. Esso infatti contiene tutti gli esperimenti fatti, con relativi modelli e risultati, e alcune funzioni e procedure utili per il calcolo e plot di metriche nonchè loading del dataset in memoria per essere utilizzato dai modelli.

7.1.1 Funzione di plot e calcolo metriche

Nel notebook viene subito definita la seguente funzione:

```
def evaluator(history, model, test_generator):
```

Questa è una funzione molto importante poiché permette di avere una panoramica completa delle performance del modello.

I suoi parametri sono:

- **history:** lo storico del training del modello. Viene utilizzato dalla funzione per fare il plot dell'andamento dell'accuracy e della funzione loss sia per il train set che per il validation set;
- **model:** il modello allenato utile per essere testato sul test set;

- **test_generator:** un oggetto di ImageDataGenerator che carica il set di immagini presenti all'interno del test set.

La funzione, grazie all'utilizzo dei parametri, è dunque in grado di fornirci:

- **Training e validation accuracy:** mostra l'andamento dell'accuracy lungo lo scorrere delle epoche.
- **Training e validation loss:** mostra l'andamento della funzione loss lungo lo scorrere delle epoche.
- **Curve ROC:** è un grafico con coordinate (True Positive Rate, False Positive Rate) che ci permette di vedere graficamente come il modello si sta comportando nella classificazione per ogni classe.
- **Matrice di confusione:** è una matrice utile a vedere quali classi il modello predice correttamente e quali no.
- **Classification report:** è una tabella che fornisce una serie di metriche (Accuracy, F1-score, Precision, Recall) per le classi, utili per effettuare una valutazione delle prestazioni del modello.

7.1.2 Fase di load delle immagini

Un'altra importante parte del notebook è la sezione di caricamento delle immagini con l'utilizzo di ImageDataGenerator.

In questa sezione vengono definiti i path riguardanti le partizioni train, validation e test del dataset, che vengono successivamente utilizzate da ImageDataGenerator per creare:

- **Train_generator:** viene utilizzata dai modelli per ottenere il training set di immagini durante la fase di addestramento.
- **Validation_generator:** viene utilizzata dai modelli per ottenere il validation set durante la fase di addestramento.
- **Test_generator:** viene utilizzata dalla funzione *evaluator* per valutare il modello utilizzando il test set.

Inoltre grazie ad ImageDataGenerator vengono applicate delle trasformate alle immagini al fine di farne Data augmentation e migliorare la generalizzazione del modello.

7.1.3 Esperimenti

Il resto del notebook è composto da 9 esperimenti in cui in ognuno di esso viene allenato un modello CNN con architettura diversa al fine di trovare il migliore.

Ogni fase di training, della durata di 25 epoche, è seguita da una chiamata ad Evaluator che fornisce le metriche e grafici necessari per valutare la precisione del modello.

7.2 Codici Web Scraping

I codici presentati in questa sezione sono i codici realizzati al fine di fare web scraping per acquisire un dataset

7.2.1 thomannScraping.py

Questo script è progettato per effettuare web scraping delle immagini dei prodotti da pagine web specifiche all'interno di una categoria. Esiste anche una versione semplificata dello script per effettuare lo scraping di pagine singole.

Lo script si compone di due funzioni principali:

```
def scrape_page(page_url, category, page_number,
   _records=None):
    if _records is None:
        _records = []
    page_html = uRequest(page_url).read()
    page_soup = soup(page_html, features="html.parser")
    containers = page_soup.findAll('div', {'class':
       'fx-product-list-entry'})
    for container in tqdm(containers, desc=f'Estrazione
       pagina {page_number}...', total=len(containers)):
```

```
            img_url = container.findAll('a', {'class':
                'product__image'})[0].picture.source.get('data-srcset',
                None)
            _records.append(img_url)
    return _records, category
```

La funzione "scrape_page" è progettata per estrarre le immagini dei prodotti da una specifica pagina web e restituire una lista di URL delle immagini. Richiede quattro parametri di input: "page_url," "category," "page_number," e "_records."

Il parametro "page_url" rappresenta l'URL della pagina web da cui estrarre le immagini dei prodotti. "Category" rappresenta la categoria di prodotti a cui appartengono le immagini. "Page_number" indica il numero di pagina corrente. Infine, il parametro "_records" è opzionale e rappresenta una lista di URL delle immagini già estratte dalle pagine precedenti.

La funzione inizia controllando se il parametro "_records" è vuoto. Se lo è, lo inizializza come una lista vuota. Successivamente, utilizza la libreria "uRequest" per aprire l'URL della pagina web specificata e leggere il contenuto HTML della pagina. Il contenuto HTML viene quindi analizzato utilizzando la libreria "BeautifulSoup" per trovare l'elemento HTML specifico contenente le informazioni sui prodotti.

Una volta trovati gli elementi HTML pertinenti, la funzione li attraversa con un ciclo "for" per estrarre gli URL delle immagini dei prodotti. Questi URL vengono quindi aggiunti alla lista "_records."

Infine, la funzione restituisce la lista "_records" e la categoria di prodotti corrente. Questa funzione può essere utilizzata per estrarre le immagini dei prodotti da più pagine web e categorizzarle in base alla categoria specificata.

```
def navigate_and_scrape(base_url, category, _records=None):
    if _records is None:
        _records = []
    all_records = _records
    page = 1
    while True:
        url = f"{base_url}?pg={page:d}&ls=25"
        records, _ = scrape_page(url, category, page)
        all_records.extend(records)
        if len(records) == 0:
```

```
            break
        next_url = f"{base_url}?pg={page + 1:d}&ls=25"
        next_records, _ = scrape_page(next_url, category,
            page + 1)
        if len(next_records) == 0:
            break
        page += 1
        if page > 20:
            break
    return all_records
```

La funzione "navigate_and_scrape" è progettata per navigare attraverso le pagine web di una specifica categoria di prodotti e utilizzare la funzione "scrape_page" per estrarre le immagini dei prodotti da ciascuna pagina. Richiede tre parametri di input: "base_url," "category," e "_records."

Il parametro "base_url" rappresenta l'URL di base della categoria di prodotti. "Category" rappresenta la categoria di prodotti. Infine, il parametro "_records" è opzionale e rappresenta una lista di URL delle immagini già estratte dalle pagine precedenti.

La funzione inizia controllando se il parametro "_records" è vuoto. Se lo è, lo inizializza come una lista vuota. La funzione quindi utilizza un ciclo "while" per navigare attraverso le pagine web della specifica categoria di prodotti.

Per ogni pagina web, la funzione utilizza la funzione "scrape_page" per estrarre le immagini dei prodotti e le aggiunge alla lista "_records." Se la funzione "scrape_page" non trova immagini di prodotto sulla pagina corrente, il ciclo "while" viene interrotto.

Se la funzione "scrape_page" trova immagini di prodotto sulla pagina corrente, la funzione utilizza l'URL della pagina successiva per estrarre ulteriori immagini di prodotto. Se la funzione "scrape_page" non trova immagini di prodotto sulla pagina successiva, il ciclo "while" viene interrotto.

La funzione continua a navigare attraverso le pagine web della categoria di prodotti fino a quando tutte le immagini di prodotto disponibili sono estratte o fino a quando raggiunge la ventesima pagina. Se la funzione raggiunge la ventesima pagina, si interrompe per evitare possibili problemi di prestazioni.

Infine, la funzione restituisce la lista completa di URL delle immagini dei prodotti estratte da tutte le pagine web della categoria specificata. Questa funzione può essere utilizzata per estrarre tutte le immagini di prodotto disponibili per una specifica categoria e salvarle.

```
def listofurls = []

for url in tqdm(listofurls, desc='Elaborazione degli
   URL...', total=len(listofurls)):
    page_html = uRequest(url).read()
    page_soup = soup(page_html, features="html.parser")
    category_grid = page_soup.findAll('div', {'class':
       'fx-category-grid'})
    if not category_grid:  # Verifica se category_grid
       vuoto
        print(f"Nessuna griglia di categorie trovata
           nell'URL: {url}")
        continue
    page_links = [x['href'] for x in
       category_grid[0].findAll('a')]
    categories = [x.text.replace('\n', '').strip() for x in
       category_grid[0].findAll('a')]
    for page_link, category in tqdm(zip(page_links,
       categories), desc='Elaborazione delle pagine...',
       total=len(page_links)):
        dest_dir = 'thomannFiato/'
        dest_dir = os.path
```

7.2.2 Bing_Downloader.py

Il codice utilizza la libreria "bing_image_downloader" [10] per scaricare immagini da Bing Images relative alla parola chiave inserita.

```
from bing_image_downloader import downloader
downloader.download("Keyword", limit=200,
   output_dir='Kewword_Folder', adult_filter_off=True,
   force_replace=False, timeout=120)
```

La funzione "download" accetta diversi parametri [10], tra cui

- Il limite massimo di immagini da scaricare
- La directory di output in cui salvare le immagini
- L'opzione per disattivare il filtro per adulti
- L'opzione per forzare la sostituzione di eventuali immagini già presenti nella directory di output
- Il timeout massimo per il download di ogni immagine

Come già accenato precedentemente la seguente funzione non è stato particolarmente usato in quanto la qualità delle immagini non era sufficente a creare un dataset, tuttavia può essere usata per il download di qualche di immagine a fine valutativo una volta che il modello è stato allenato e validato

7.3 Codici ausiliari

I seguenti codici sono stati utili al fine di creare un dataset "pulito" ed eterogeneo

7.3.1 Count_elements.py

E' uno script Python con una funzione che conta il numero di file, cartelle e cartelle vuote all'interno di una cartella specificata dall'utente. La funzione utilizza il modulo "os" [14] per accedere alle informazioni sulla cartella e la funzione è ricorsiva per contare anche gli elementi all'interno delle sottocartelle. Se la cartella specificata non esiste, viene stampato un messaggio di errore. Alla fine, i risultati del conteggio vengono stampati a schermo.

Il codice richiede all'utente di inserire il percorso della cartella da esaminare e chiama la funzione "count_elements_in_folders" con il percorso inserito come argomento.

7.3.2 DeleteEmptyFolder.py

E' una funzione Python che elimina tutte le cartelle vuote all'interno di una cartella specificata dall'utente. La funzione utilizza la libreria "os" [14] per accedere alle informazioni sulla cartella e la funzione è ricorsiva per eliminare anche le cartelle vuote all'interno delle sottocartelle. Se la cartella specificata non esiste, viene stampato un messaggio di errore. Alla fine, viene stampato un messaggio per ogni cartella vuota eliminata.

Il codice richiede all'utente di inserire il percorso della cartella da esaminare e chiama la funzione "delete_empty_folders" con il percorso inserito come argomento.

7.3.3 RemovePlaceholder.py

Il codice è una funzione Python che cerca e rimuove le immagini duplicate all'interno di una cartella specificata dall'utente. La funzione utilizza la libreria "os" [14] per accedere alle informazioni sulla cartella e la libreria "PIL" [15] per calcolare l'hash MD5 [16] di ogni immagine.

```
def calculate_md5(image_path):
with open(image_path, 'rb') as f:
    return hashlib.md5(f.read()).hexdigest()
```

Inoltre la funzione utilizza anche il modulo "tqdm" [17] per mostrare una barra di avanzamento durante l'elaborazione delle immagini.

```
with tqdm(total=total_files, desc="Processing images")
   as pbar:
    ... #codice
    pbar.update(1)
```

La funzione cerca le immagini duplicate rispetto all'immagine di input specificata dall'utente e le rimuove, infine, viene stampato il numero di immagini duplicate rimosse.

Il codice specifica il percorso della cartella contenente le immagini e l'immagine di input e chiama la funzione "find_and_remove_duplicates" con i percorsi inseriti come argomenti.

Chapter 8

Conclusioni

Il nostro progetto per il riconoscimento di strumenti musicali tramite immagini ha comportato diverse fasi, dalle operazioni di acquisizione e pulizia dei dati fino all'addestramento e valutazione del modello. Nonostante i risultati ottenuti non siano stati ottimali, il processo di sviluppo del progetto ci ha fornito importanti lezioni.

8.1 Considerazioni rigurardanti il dataset

Il dataset è stato acquisito attraverso web scraping di siti web di strumenti musicali. Questa fase ha portato a un dataset contenente diverse categorie di strumenti, e Successivamente, sono state effettuate operazioni di pulizia per rimuovere immagini di scarsa qualità, placeholder e duplicati. Inoltre, è stata affrontata l'eterogeneità tra le classi, cercando di bilanciarle per mitigare il problema dello sbilanciamento dei dati. Nonostante questi sforzi, il dataset rimane comunque limitato in termini di dimensione, il che può ha influito negativamente sulle prestazioni del modello.

8.2 Considerazioni riguardanti gli esperimenti

Durante lo sviluppo del progetto, sono stati creati e testati diversi modelli di deep learning utilizzando reti neurali convoluzionali. Sono stati effettuati diversi esperimenti con architetture diverse, introducendo strati di convoluzione, pooling, batch normalization, dropout e dense layer. Tuttavia, nonostante gli sforzi per migliorare i modelli, i risultati non sono stati estremamente soddisfacenti. Questo può essere attribuito alla complessità del problema e alle limitazioni del dataset. Sono stati

utilizzati diversi strumenti per la valutazione dei modelli, come le curve ROC, la matrice di confusione e le metriche di precisione, recall e F1-score. Questi strumenti hanno aiutato a capire come il modello si comporta sul test set, evidenziando la difficoltà di predire correttamente le diverse classi.

8.3 Lezioni apprese

Lo sviluppo del progetto e la conseguente stesura della relazione ci ha fornito diverse lezioni:

1. **Importanza di un dataset di dimensioni adeguate**: È cruciale disporre di un dataset abbastanza ampio e bilanciato per addestrare un modello in modo efficace. Dati limitati possono portare a problemi di overfitting o underfitting.

2. **Regolazione dei modelli**: Abbiamo avuto l'opportunità di sperimentare con diversi modelli di CNN, modificandone la complessità e aggiungendo o rimuovendo strati. Tuttavia, abbiamo compreso che un modello più complesso non è sempre sinonimo di migliori prestazioni, poiché può aumentare il rischio di overfitting.

3. **Importanza delle metriche**: Abbiamo apprezzato l'importanza delle metriche per valutare correttamente le prestazioni del modello. Oltre all'accuracy, abbiamo utilizzato la matrice di confusione, l'F1-score e le curve ROC per ottenere una visione più completa delle capacità del modello. Infatti guardando solo l'accuracy il nostro modello dava risultati ottimi.

4. **Necessità di esperimenti**: Abbiamo compreso che l'apprendimento nel campo del deep learning spesso richiede sperimentazione e iterazione. È fondamentale testare diversi modelli, iperparametri e tecniche di elaborazione dei dati per individuare la combinazione ottimale.

8.4 Sviluppi futuri

Inoltre vogliamo sottolineare come abbiamo individuato diverse possibilità di migliorare le prestazioni del modello e sviluppare ulteriormente questo progetto in futuro:

1. **Aumentare la quantità di dati**: Un dataset più grande e vario potrebbe aiutare il modello a generalizzare meglio e migliorare le prestazioni.

2. **Tuning degli iperparametri**: Esplorare una gamma più ampia di iperparametri potrebbe aiutare a trovare una configurazione migliore per il modello.

3. **Approccio multiclasse o binario**: Potrebbe essere utile considerare un approccio binario (ad esempio, keyboard vs piano) o combinazioni di problemi multiclasse e binario per affrontare il problema in modo più efficace.

Bibliography

[1] M. V. Valueva, N. N. Nagornov, P. A. Lyakhov, G. V. Valuev, and N. I. Chervyakov. "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation". In: *Mathematics and Computers in Simulation* 177 (2020), pp. 232–243. DOI: 10.1016/j.matcom.2020.04.031.

[2] D. Ciresan, U. Meier, and J. Schmidhuber. "Multi-column deep neural networks for image classification". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. New York, NY: Institute of Electrical and Electronics Engineers (IEEE), 2012, pp. 3642–3649. ISBN: 978-1-4673-1226-4. DOI: 10.1109/CVPR.2012.6248110. URL: https://ieeexplore.ieee.org/abstract/document/6248110.

[3] D. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. "Flexible, High Performance Convolutional Neural Networks for Image Classification". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence-Volume Volume Two*. Vol. 2. 2011, pp. 1237–1242. URL: https://www.ijcai.org/Proceedings/13/Papers/517.pdf.

[4] Y. LeCun, C. Cortes, and C. J. Burges. "THE MNIST DATABASE of handwritten digits". In: *http://yann. lecun. com/exdb/mnist/* (1998).

[5] D. Gershgorn. "The inside story of how AI got good enough to dominate Silicon Valley". In: *Quartz* (). URL: https://qz.com/1304013/the-revolution-inside-the-story-of-how-artificial-intelligence-got-good-enough-to-dominate-big-tech/.

[6] G. C. Alessio Mezzina. *Machine Learning Project Repository 2022-2023*. https://github.com/timeassassinRG/Progetto-Machine-Learning-2022-2023.

[7] A. Furnari. *Web Scraping with Python for Data Science*. https://www.antoninofurnari.it/lecture-notes/it/data-science-python/web-scraping/.

[8] *Guitar Center*. https://www.guitarcenter.com/.

[9] *Thomann*. https://www.thomann.de/it/.

[10] *Bing Image Downloader*. https://pypi.org/project/bing-image-downloader/.

[11] Wikipedia. *Keras — Wikipedia, L'enciclopedia libera*. [Online; in data 24-luglio-2023]. 2021. URL: \url{//it.wikipedia.org/w/index.php?title=Keras&oldid=121484014}.

[12] TensorFlow. *ImageDataGenerator*. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator. 2021.

[13] TensorFlow Authors. *EfficientNetB0 - TensorFlow API*. https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet/EfficientNetB0. anno di accesso.

[14] P. software foundation. *Python 3 - os module*. https://docs.python.org/3/library/os.html.

[15] P. pillow contributors. *Pillow Documentation*. https://pillow.readthedocs.io/en/stable/.

[16] Wikipedia contributors. *MD5 — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=MD5&oldid=1163481387. [Online; accessed 24-July-2023]. 2023.

[17] tqdm contributors. *tqdm: A Fast, Extensible Progress Bar for Python and CLI*. https://pypi.org/project/tqdm/.