

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA -

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

ANNO ACCADEMICO 2020/2021

**- RELAZIONE FINALE -
TRASCRIZIONE AUTOMATICA
DI TABLATURE PER CHITARRA
TRAMITE L'USO DI
SMARTPHONE**

CORSO DI SISTEMI DIGITALI M

GRUPPO:
DE NARDI-TORNATORE

Indice

Introduzione	1
1 Chitarra	3
1.0.1 Corde	4
1.0.2 Tasti	4
1.0.3 Tab	5
2 Trasformata a Q Costante	7
3 Architettura e addestramento del modello	9
3.1 Set dati GuitarSet	9
3.1.1 Compressione con NumPy	10
3.1.2 Trasformata a Q Costante in Librosa	10
3.1.3 Pre-elaborare i dati	12
3.2 Modello della rete	14
3.2.1 Uso di Keras	14
3.2.2 Compilazione del modello	15
3.2.3 Addestramento del modello	16
3.3 Accuratezza del modello	16
4 Implementazione su dispositivo Embedded	19
4.1 Sviluppo applicazione iOS	19
4.1.1 Conversione del modello da Keras a CoreML	19
4.1.2 Conversione del modello da Keras a Tensorflow Lite	23
4.1.3 Sviluppo applicazione con Swift 5 e Xcode 12	23
4.1.4 Uso di TensorFlow Lite su iOS	25
4.1.5 Uso di un server per l'uso della libreria Librosa	25
4.2 Sviluppo applicazione Android	26
4.2.1 Conversione del modello da Keras a Tensorflow Lite	26
4.2.2 Sviluppo applicazione con Java 1.8 e Android Studio 4.1.2	27

Introduzione

La musica ha assunto un ruolo di primo piano nella storia dell'uomo. Forse è la voglia di esprimere i propri sentimenti e le proprie emozioni che hanno obbligato l'essere umano a comporre sempre nuovi brani. Senza alcun'ombra di dubbio, la chitarra è uno degli strumenti più usati. Per esempio, quando si va in campeggio e alla sera ci si siede davanti al falò, l'aria si riempie delle sue note.

Il tema libero lasciato dai professori ha permesso di approfondire ma soprattutto conoscere meglio il vastissimo campo della musica.

Il progetto ha l'obiettivo di trascrivere in modo automatico le tablature per chitarra in modo che anche chi non è in grado di suonare questo strumento lo possa fare. Il tutto funziona tramite l'ausilio di un semplice *smartphone*.

La relazione è articolata come segue:

- nel **capitolo 1** viene descritta brevemente come è fatta la chitarra;
- nel **capitolo 2** viene introdotto il dominio in cui lavoreremo;
- nel **capitolo 3** viene descritto il modello della nostra rete, l'addestramento che è stato eseguito e la sua accuratezza;
- nel **capitolo 4** viene realizzata l'implementazione sul dispositivo *embedded*; nel nostro caso sono gli *smartphone*;
- il **capitolo 5** conclude la relazione presentando i risultati ottenuti e gli obiettivi raggiunti.

La relazione è stata scritta come un *diario* mettendo in risalto tutti i passaggi, i tentativi e i problemi che si sono verificati.

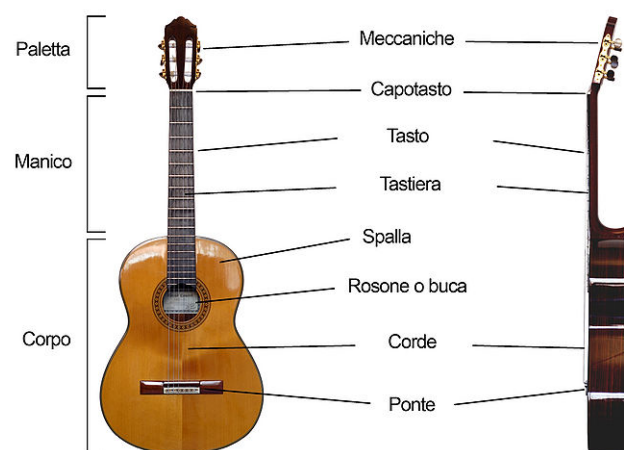
Capitolo 1

Chitarra

La chitarra ha una lunga tradizione che affonda le sue radici addirittura al tempo degli arabi. I primi esemplari risalgono al tredicesimo secolo. Inizialmente dotata di quattro corde, si è accresciuta nel Rinascimento di un'altra corda, arrivando poi nel periodo Barocco all'attuale numero di sei.

La chitarra è composta da due parti principali:

- il manico, su cui si trova la tastiera e che termina con la paletta la quale ospita le meccaniche per l'accordatura;
- la cassa di risonanza o tavola armonica con una cavità centrale, che serve ad amplificare il suono prodotto dalle corde.



1.0.1 Corde

Le corde delle chitarre moderne sono sei e sono ordinate dall'alto verso il basso nel seguente modo:

- la prima corda corrisponde alla nota Mi cantino (e);
- la seconda corrisponde alla nota Si (B);
- la terza corrisponde alla nota Sol (G);
- la quarta corrisponde alla nota Re (D);
- la quinta corrisponde alla nota La (A);
- la sesta corrisponde alla nota Mi basso (E).

L'ultima corda dell'elenco è quella più spessa, mentre la prima è la più sottile.

1.0.2 Tasti

Sul manico della chitarra c'è la tastiera. Si chiama tastiera proprio perchè ci sono i tasti. Quest'ultimi sono delimitati da delle barrette di metallo e ognuno di essi corrisponde a una nota. Dunque, se abbiamo una chitarra a 19 tasti, possiamo fare 19 note diverse per ogni corda.

La distanza tra due tasti della stessa corda prende il nome di **semitono**. Ad esempio, se premiamo la sesta corda in corrispondenza del La, poi premendo la corda al tasto adiacente più vicino alla cassa di risonanza (un semitono più alto) ascolteremo un La#.

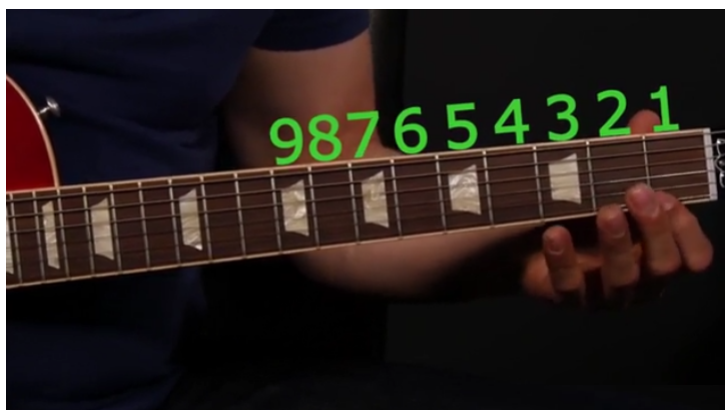
Se non premiamo nessun tasto la corda si dice che è suonata a vuoto. Le sei corde suonate a vuoto devono emettere dei suoni ben precisi: la chitarra deve essere quindi accordata. L'accordatura classica delle sei corde, ovvero la nota che devono suonare le corde a vuoto (dal basso all'alto), è la seguente: Mi, La, Re, Sol, Si, Mi. Conoscendo il suono prodotto dalle sei corde suonate a vuoto e sapendo che ogni nota suonata ad un tasto dista di un semitono dalla nota suonata al tasto adiacente possiamo mappare tutta la tastiera della chitarra.

MI	FA	F#	SOL	SOL#	LA	LA#	SI	DO	DO#	RE	RE#	MI		
SI	DO	DO#	RE	RE#	MI	FA	F#	SOL	SOL#	LA	LA#	SI		
SOL	SOL#	LA	LA#	SI	DO	DO#	RE	RE#	MI	FA	F#	SOL		
RE	RE#	MI	FA	F#	SOL	SOL#	LA	LA#	SI	DO	DO#	RE		
LA	LA#	SI	DO	DO#	RE	RE#	MI	FA	F#	SOL	SOL#	LA		
MI	FA	F#	SOL	SOL#	LA	LA#	SI	DO	DO#	RE	RE#	MI		

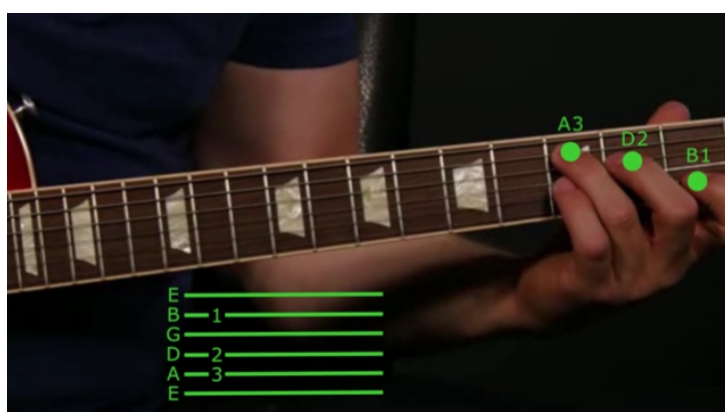
1.0.3 Tab

La *tab* è una rappresentazione delle corde della chitarra. Una tablatura è solitamente scritta usando sei linee orizzontali, ognuna corrispondente a una corda.

Al contrario dei normali spartiti, su una tablatura non ci sono le note da suonare ma si trovano le indicazioni su dove mettere le dita. I numeri sulle linee corrispondono ai tasti della tastiera. Ad esempio, un "1" sulla prima corda, indica di suonare il Mi cantino tenendo premuto il primo tasto. Se il numero è superiore a zero, bisogna premere il tasto corrispondente quando si suonerà quella corda. Se troviamo uno **zero** allora si suona la corda a vuoto.



Spesso leggendo una tablatura si trovano dei numeri che sono allineati verticalmente. In questo caso si premono i tasti indicati e poi si suonano le note allo stesso tempo. Le tab vanno lette come libri cioè da sinistra a destra.



Capitolo 2

Trasformata a Q Costante

Le note musicali si possono classificare nel seguente modo:

MUSICAL NOTE FREQUENCY CHART																	
NOTE	FREQ (Hz)	NOTE	FREQ (Hz)	NOTE	FREQ (Hz)	NOTE	FREQ (Hz)	NOTE	FREQ (Hz)	NOTE	FREQ (Hz)	NOTE	FREQ (Hz)	NOTE	FREQ (Hz)	NOTE	FREQ (Hz)
C0	16.35	C1	32.70	C2	65.41	C3	130.81	>C4<	>261.63<	C5	523.25	C6	1046.50	C7	2093.00	C8	4186.01
C#0	17.32	C#1	34.65	C#2	69.30	C#3	138.59	C#4	277.18	C#5	554.37	C#6	1108.73	C#7	2217.46		
D0	18.35	D1	36.71	D2	73.42	D3	146.83	D4	293.67	D5	587.33	D6	1174.66	D7	2349.32		
D#0	19.45	D#1	38.89	D#2	77.78	D#3	155.56	D#4	311.13	D#5	622.25	D#6	1244.51	D#7	2489.02		
E0	20.60	E1	41.20	E2	82.41	E3	164.81	E4	329.63	E5	659.26	E6	1318.51	E7	2637.02		
F0	21.83	F1	43.65	F2	87.31	F3	174.61	F4	349.23	F5	698.46	F6	1396.91	F7	2793.83		
F#0	23.12	F#1	46.25	F#2	92.50	F#3	185.00	F#4	369.99	F#5	739.99	F#6	1479.98	F#7	2959.96		
G0	24.50	G1	49.00	G2	98.00	G3	196.00	G4	392.00	G5	783.99	G6	1567.98	G7	3135.96		
G#0	25.96	G#1	51.91	G#2	103.83	G#3	207.65	G#4	415.31	G#5	830.61	G#6	1661.22	G#7	3322.44		
A0	27.50	A1	55.00	A2	110.00	A3	220.00	A4	440.00	A5	880.00	A6	1760.00	A7	3520.00		
A#0	29.14	A#1	58.27	A#2	116.54	A#3	233.08	A#4	466.16	A#5	932.33	A#6	1864.66	A#7	3729.31		
B0	30.87	B1	61.74	B2	123.47	B3	246.94	B4	493.88	B5	987.77	B6	1975.53	B7	3951.07		

La lettera a sinistra identifica la nota musicale mentre il numero a destra rappresenta la sua frequenza.

In musica, un'**ottava** è l'intervallo di 8 note posizionate a frequenza diversa nella scala musicale. Le frequenze intermedie sono altre sei note. Per esempio, il *la3* (A4) ha frequenza di 440 Hz, il *la* posto un'ottava sopra ha frequenza 880 Hz, quello un'ottava sotto ha frequenza 220 Hz.

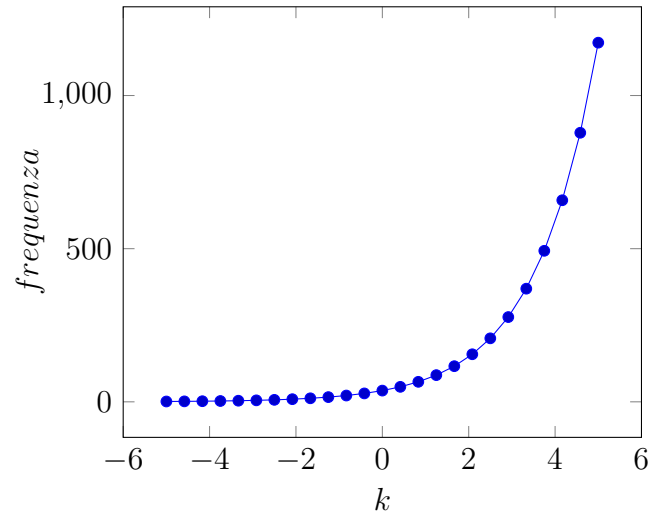
Se si rappresentassero le prime sei ottave della nota C potremmo vedere che la sua frequenza raddoppia ad ogni ottava.

I tasti che intercorrono fra gli estremi della stessa ottava (esempio *do3-do4*) sono dodici note per cui la frequenza deve raddoppiare ogni dodici note. Si può rappresentare quanto detto dalla seguente formula:

$$F_k = 440Hz \cdot 2^{\frac{k}{12}}$$

Nel campo della musica viene usata la trasformata a Q costante proprio per la sua natura esponenziale. Inoltre, l'accuratezza della trasformata a Q costante è

analoga alla scala logaritmica e imita l'orecchio umano, avendo una risoluzione di frequenza più alta a quelle più basse e una risoluzione più bassa alle frequenze più alte. Infatti, dal seguente grafico si può notare la natura esponenziale della funzione:



Capitolo 3

Architettura e addestramento del modello

3.1 Set dati GuitarSet

tempo di lavoro:

Fortunatamente, su Internet abbiamo trovato un *data set* di file audio di chitarra già pronto su cui lavorare. Il *GuitarSet*, chiamato così dal suo creatore, è costituito dai file audio e dai suoi **tab**.

Questo *data set* contiene 360 estratti di canzoni della durata di circa 30 secondi. Essi sono il risultato delle seguenti combinazioni:

- 6 persone suonano ciascuno gli stessi 30 fogli
- Vengono registrate 2 versioni diverse: comping e soloing

I 30 fogli sono generati da una combinazione di

- **5 stili**: Rock, Cantautore, Bossa Nova, Jazz e Funk
- **3 progressioni**: 12 Bar Blues, Autumn Leaves e Pachelbel Canon.
- **2 Tempi**: lento e veloce.

Gli estratti sono registrati sia con il pickup esafonico che con un microfono a condensatore Neumann U-87. Ci sono tre registrazioni audio per ogni estratto:

- **hex**: file wav originale a 6 canali dal pickup esafonico;
- **hex_cln**: file wav esadecimale con rimozione delle interferenze applicata;

- **mic**: registrazione monofonica dal microfono di riferimento

Noi abbiamo usato registrazioni di tipo **mic** perchè sono quelle che la nostra rete avrebbe usato.

Ciascuno dei 360 estratti ha anche un file `.jams` che memorizza 16 annotazioni da cui prenderemo le tab:

- Intonazione:
 - 6 annotazioni *pitch_contour* (1 per stringa);
 - 6 annotazioni *midi_note* (1 per stringa);
- Beat e tempo:
 - 1 annotazione *beat_position*;
 - 1 annotazione del tempo;
- Accordi:
 - 2 annotazioni di accordi (istruite ed eseguite).

Noi useremo le annotazioni *midi_note*

3.1.1 Compressione con NumPy

NumPy è una libreria che aggiunge supporto a grandi matrici e array multidimensionali insieme a una vasta collezione di funzioni matematiche di alto livello per poter operare efficientemente su queste strutture dati.

I dati sono stati compressi in *numpy array* (`.npz`) per organizzarli meglio altrimenti avremmo avuto un'immagine e una label per ogni file audio.

3.1.2 Trasformata a Q Costante in Librosa

Librosa è una libreria per la musica e l'analisi audio. Fornisce gli elementi costitutivi necessari per creare sistemi di recupero delle informazioni musicali.

La trasformazione a Q Costante può essere facilmente applicata ai file audio utilizzando la libreria *librosa*.

```

# parameters
self.sr_downs = 22050
self.hop_length = 512
self.n_bins = 192
self.bins_per_octave = 24

self.frameDuration = self.hop_length / self.sr_downs

# save file path
self.save_path = self.path_data + "dataset1/"

def audio_CQT(self, file_num, start, dur): # start and dur in seconds
    path = os.path.join(self.path_audio,
                        os.listdir(self.path_audio)[file_num])

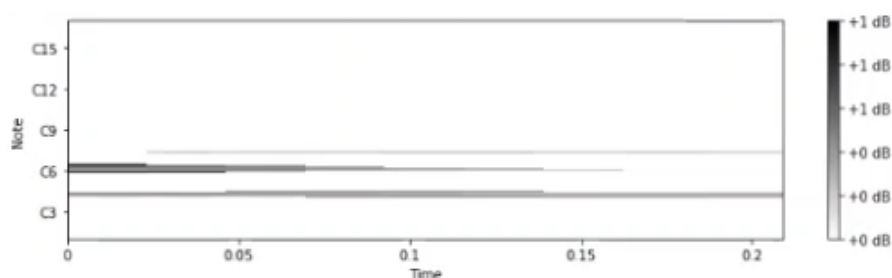
    # Function for removing noise
    def cqt_lim(CQT):
        new_CQT = np.copy(CQT)
        new_CQT[new_CQT < -60] = -120
        return new_CQT

    # Perform the Constant-Q Transform
    data, sr = librosa.load(path, sr = self.sr_downs,
                            mono = True, offset = start, duration = dur)
    data = librosa.util.normalize(data)
    data = librosa.cqt(data, sr = self.sr_downs,
                        hop_length = self.hop_length,
                        fmin = None,
                        n_bins = self.n_bins,
                        bins_per_octave = self.bins_per_octave)
    data = librosa.magphase(data)[0]**4
    data = librosa.core.amplitude_to_db(data, ref = np.amax)
    cqt = cqt_lim(data)
    return cqt

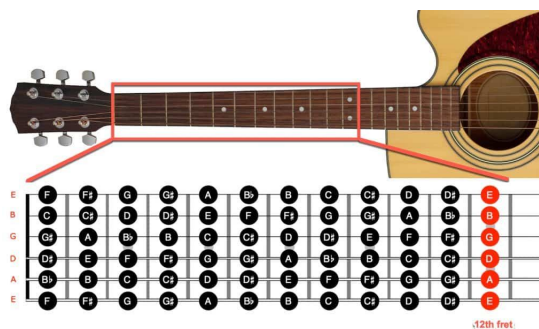
```

Ogni file audio è stato diviso ogni tot secondi in modo da stampare le note che sono state suonate sulla chitarra sempre tramite la libreria *librosa*. L'obiettivo è quello di prendere l'immagine appena ottenuta e di darla in input alla rete CNN. Sperimentalmente, abbiamo visto che per ottenere un buon valore di accuratezza nella rete, le immagini devono essere stampate ogni 0.2 secondi.

12 CAPITOLO 3. ARCHITETTURA E ADDESTRAMENTO DEL MODELLO



Se il tempo fosse stato inferiore avremmo avuto immagini di file audio con solo note singole e quando la rete avrebbe dovuto riconoscere più note non sarebbe stata in grado di farlo. Nell'immagine seguente è possibile vedere che le lettere sulle note si ripetono. Ad esempio, la lettera F si trova sulla corda F e D. Se la mano è posizionata sulla corda A, è impossibile che si riesca a suonare la F della nota G. Dunque, un istante di tempo troppo corto non aiutava la rete a riconoscere la nota giusta.



L'immagine risultante da *librosa* ha dimensione 192x9.

I tab sono matrici 6x19 e hanno caratteristica di essere *one-shot* cioè si può trovare un uno solo in ogni riga. Ad ogni uno presente nella riga corrisponde a una nota. Le sei righe della matrice corrispondono alle sei note della chitarra mentre i tasti sono diciotto. La colonna in più, più precisamente la prima, indica che nella prima colonna la corda non è stata toccata.

3.1.3 Pre-elaborare i dati

3.2 Modello della rete

3.2.1 Uso di Keras

Keras consente di implementare algoritmi basati su reti neurali. Permette di sviluppare e prototipare in maniera semplice e veloce modelli nell'ambito del machine learning e del deep learning. **tempo di lavoro:**

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, kernel_size = (3, 3), activation=tf.nn.relu))
model.add(tf.keras.layers.Conv2D(64, kernel_size = (3, 3), activation=tf.nn.relu))
model.add(tf.keras.layers.Conv2D(64, kernel_size = (3, 3), activation=tf.nn.relu))
model.add(tf.keras.layers.MaxPooling2D(pool_size = (2,2)))
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(6*19))
model.add(tf.keras.layers.Reshape((6, 19)))
model.add(tf.keras.layers.Activation(activation=tf.nn.softmax))
model.summary()
```

```
epochs = 30
learning_rate = 0.01
momentum = 0.8
decay = learning_rate/epochs
sgd = SGD(lr = learning_rate, momentum = momentum, decay = decay, nesterov = True)

# Training (Functional Method)
model_in = Input(shape = input_shape)
conv1 = Conv2D(32, kernel_size = (3, 3), activation = 'relu')(model_in)
conv2 = Conv2D(64, kernel_size = (3, 3), activation = 'relu')(conv1)
conv3 = Conv2D(64, kernel_size = (3, 3), activation = 'relu')(conv2)
pool1 = MaxPooling2D(pool_size = (2, 2), strides = (2, 2))(conv3)
flat = Flatten()(pool1)

# Create fully connected model heads
y1 = Dense(152, activation = 'relu')(flat)
```

```

y1 = Dropout(0.5)(y1)
y1 = Dense(76)(y1)
y1 = Dropout(0.2)(y1)

y2 = Dense(152, activation = 'relu')(flat)
y2 = Dropout(0.5)(y2)
y2 = Dense(76)(y2)
y2 = Dropout(0.2)(y2)

y3 = Dense(152, activation = 'relu')(flat)
y3 = Dropout(0.5)(y3)
y3 = Dense(76)(y3)
y3 = Dropout(0.2)(y3)

y4 = Dense(152, activation = 'relu')(flat)
y4 = Dropout(0.5)(y4)
y4 = Dense(76)(y4)
y4 = Dropout(0.2)(y4)

y5 = Dense(152, activation = 'relu')(flat)
y5 = Dropout(0.5)(y5)
y5 = Dense(76)(y5)
y5 = Dropout(0.2)(y5)

y6 = Dense(152, activation = 'relu')(flat)
y6 = Dropout(0.5)(y6)
y6 = Dense(76)(y6)
y6 = Dropout(0.2)(y6)

# Connect heads to final output layer
out1 = Dense(19, activation = 'softmax', name = 'estring')(y1)
out2 = Dense(19, activation = 'softmax', name = 'Bstring')(y2)
out3 = Dense(19, activation = 'softmax', name = 'Gstring')(y3)
out4 = Dense(19, activation = 'softmax', name = 'Dstring')(y4)
out5 = Dense(19, activation = 'softmax', name = 'Astring')(y5)
out6 = Dense(19, activation = 'softmax', name = 'Estring')(y6)

```

3.2.2 Compilazione del modello

tempo di lavoro:

```
# Create model
model = Model(inputs = model_in,
               outputs = [out1, out2, out3, out4, out5, out6])
model.compile(optimizer = SGD,
              loss = ['categorical_crossentropy',
                     'categorical_crossentropy',
                     'categorical_crossentropy',
                     'categorical_crossentropy',
                     'categorical_crossentropy',
                     'categorical_crossentropy'],
              metrics = ['accuracy'])
```

3.2.3 Addestramento del modello

tempo di lavoro:

Dopo diverse prove sperimentali, abbiamo deciso di eseguire il modello per 30 epoche:

Questo modello raggiunge una precisione di circa 0,91 (o 91%) sui dati di addestramento.

```
Epoch 1/10
1875/1875 [=====] - 3s 1ms/step - loss: 0.4917 - accuracy: 0.8277
Epoch 2/10
1875/1875 [=====] - 3s 1ms/step - loss: 0.3702 - accuracy: 0.8674
Epoch 3/10
1875/1875 [=====] - 3s 2ms/step - loss: 0.3328 - accuracy: 0.8793
Epoch 4/10
1875/1875 [=====] - 3s 1ms/step - loss: 0.3106 - accuracy: 0.8859
Epoch 5/10
1875/1875 [=====] - 3s 1ms/step - loss: 0.2915 - accuracy: 0.8927
Epoch 6/10
1875/1875 [=====] - 3s 1ms/step - loss: 0.2771 - accuracy: 0.8968
Epoch 7/10
```

3.3 Accuratezza del modello

tempo di lavoro:

Abbiamo confrontato le prestazioni del modello sul set di dati di test:

```
val_loss, val_accuracy = model.evaluate(X_test, y_test)
```

```
313/313 - 1s - loss: 0.3637 - accuracy: 0.8693
```

```
Test accuracy: 0.8693000078201294
```

Capitolo 4

Implementazione su dispositivo Embedded

La scelta è caduta sui dispositivi mobili. Questa decisione è stata vincolante perchè disponevamo di solo queste risorse *hardware*.

4.1 Sviluppo applicazione iOS

4.1.1 Conversione del modello da Keras a CoreML

tempo di lavoro: 12 giorni

Per poter usare il modello pre-addestrato sul cellulare abbiamo dovuto convertirlo nel formato *.mlmodel* in modo da poter usare il *framework Core ML*.

Problemi riscontrati: durante la conversione del modello sono apparsi diversi errori che impedivano la conversione. Gli errori sono simili a quello riportato di seguito:

```
ValueError: Input 0 of node save/AssignVariableOp was passed float from conv2d/bias:0 incompatible with expected resource.
```

Soluzioni provate:

- Abbiamo preso spunto dal codice che si trova sul blocco di lucidi visti a lezione. Esso usa il package *tfcoreml*. Il focus dello *script* è il seguente:

```
import tfcoreml as tf_converter
tf_converter.convert(
    tf_model_path = './output/frozen_model.pb',
```

```
mlmodel_path = './output/frozen.mlmodel',
output_feature_names = ['Softmax:0'])
```

A questo punto serviva ottenere il file in formato *.pb* da usare come *input*. Questo file prende il nome di modello congelato. Prima di ottenerlo serve salvarci il modello che si ottiene con *Tensorflow*. Esso è formato da quattro file:

- **model-ckpt.meta**: contiene il grafico completo (flusso di dati, le annotazioni per le variabili, le *pipeline* di *input* e altre informazioni);
- **model-ckpt.data-0000-of-00001**: contiene tutti i valori delle variabili (pesi, segnaposto, gradienti, iperparametri, ecc.);
- **model-ckpt.index**: ci sono tutti i metadati. È una tabella immutabile in cui ogni chiave è un nome di un tensore e il suo valore descrive i metadati di un tensore;
- **checkpoint**: tutte le informazioni sul *checkpoint*.

```
output_directory_path = './output'

# the model_dir states where the graph and checkpoint files
# will be saved to
estimator_model = tf.keras.estimator.model_to_estimator(
    keras_model = model,
    model_dir = output_directory_path)

def input_function(features, labels=None, shuffle=False):
    input_fn = tf.estimator.inputs.numpy_input_fn(
        x={"conv2d_1_input": features},
        y=labels,
        shuffle=shuffle,
        batch_size = 128,
        num_epochs = 30
    )
    return input_fn

estimator_model.train(input_fn = input_function(X_train, y_train, True))
```

estimator_model.train serve a verificare che il modello esportato in precedenza sia effettivamente funzionante.

Il modello congelato ci consente di eliminare tutte le informazioni in più che vengono salvate perchè si potrebbe ricaricare quello appena salvato e l'addestramento continua da dove era stato interrotto.

```
def freeze_graph(model_dir, output_node_names):
    if not tf.gfile.Exists(model_dir):
        raise AssertionError(
            "Export directory doesn't exists. Please specify an export "
            "directory: %s" % model_dir)

    if not output_node_names:
        print("You need to supply the name of a node to output_node_names.")
        return -1

    checkpoint = tf.train.get_checkpoint_state(model_dir)
    input_checkpoint = checkpoint.model_checkpoint_path

    absolute_model_dir = "/".join(input_checkpoint.split('/')[:-1])
    output_graph = absolute_model_dir + "/frozen_model.pb"

    clear_devices = True

    with tf.Session(graph=tf.Graph()) as sess:
        saver = tf.train.import_meta_graph(
            input_checkpoint + '.meta', clear_devices=clear_devices)

        saver.restore(sess, input_checkpoint)

        output_graph_def = tf.graph_util.convert_variables_to_constants(
            sess,
            tf.get_default_graph().as_graph_def(),
            output_node_names.split(","))

    with tf.gfile.GFile(output_graph, "wb") as f:
        f.write(output_graph_def.SerializeToString())
        print("%d ops in the final graph." % len(output_graph_def.node))

    return output_graph_def

freeze_graph('./output/', "save/restore_all")
```

- Uno dei nuovi tentativi si è basato sul cambio di codice per salvare il modello: abbiamo usato il seguente codice:

```
output_directory_path = './output'

supervisor = tf.train.Supervisor(logdir=output_directory_path)

with supervisor.managed_session() as session:
    # train the model here
    supervisor.saver.save(session, output_directory_path)
```

Tuttavia, i risultati non sono stati quelli sperati.

- Cercando nella documentazione di *coreml*, abbiamo scoperto che non sono previsti più aggiornamenti e consigliavano di usare un nuovo *package*. Anche se fossimo stati in grado di convertirlo non avremmo potuto utilizzato su sistemi operativi *iOS* maggiori di 12. La nuova libreria che abbiamo usato si chiama *coremltools*.

```
import coremltools
coreml_model = coremltools.converters.keras.convert(model)

coreml_model.author = 'De Nardi-Tornatore'
coreml_model.short_description = 'Recognition guitar music'

coreml_model.save("Stock.mlmodel")
```

- Per non avere altri errori che ci apparivano, abbiamo usato i livelli di convoluzione presi direttamente da *keras* e non da *tensorflow*. E' stato molto difficile superare tutti questi ostacoli perché i messaggi di errore non aiutavano a capire bene che cosa bisognasse modificare.
- Successivamente, per aumentare l'accuratezza, è stata inserita una funzione di attivazione personalizzata. Purtroppo, anche con *coremltools* non siamo stati in grado di convertirlo perché ci appariva un errore in corrispondenza del livello della nuova funzione:

```
TypeError: argument of type 'NoneType' is not iterable
```

Soluzione finale: abbiamo deciso di usare *Tensorflow Lite* perché siamo riusciti a convertire il modello subito senza nessun problema.

4.1.2 Conversione del modello da Keras a Tensorflow Lite

tempo di lavoro: 1h

Come accennato nel paragrafo precedente, abbiamo convertito il modello usando *Tensorflow Lite*.

```
model = tf.keras.models.load_model("saved/model.h5",
    custom_objects={'softmax_by_string': self.softmax_by_string,
    'avg_acc': self.avg_acc, 'catcross_by_string': self.catcross_by_string})
converter = lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
open("saved/model.tflite", "wb").write(tflite_model)
```

Il seguente codice consente di caricare il modello addestrato tramite *Tensorflow* e di convertirlo nel formato *.tflite* pronto per essere usato su un dispositivo mobile nel nostro caso.

4.1.3 Sviluppo applicazione con Swift 5 e Xcode 12

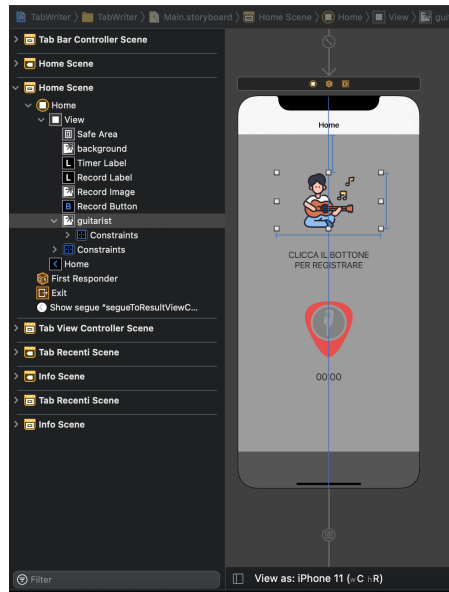
tempo di lavoro: 20 giorni

Swift è un linguaggio di programmazione *object-oriented* concepito per programmare sui sistemi operativi *Apple*.

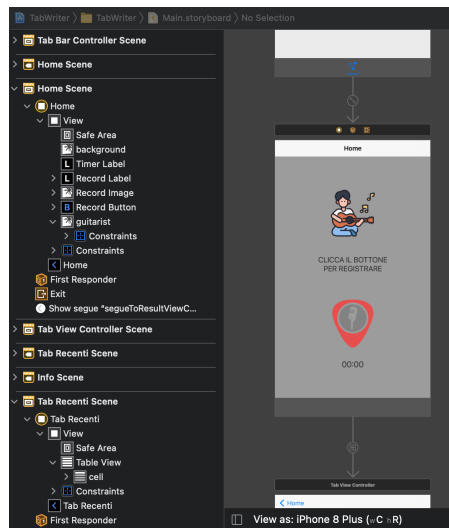
Xcode è un ambiente di sviluppo integrato completamente sviluppato e mantenuto da *Apple*, che consente di sviluppare *software* per i sistemi *macOS*, *iOS*, *watchOS* e *tvOS*.

Abbiamo dovuto prendere un pò di familiarità con il nuovo linguaggio e il nuovo *IDE* dato che non avevamo mai programmato nel mondo *Apple*. La documentazione messa a disposizione agli sviluppatori è molto vasta e le solide basi apprese a ingegneria hanno fatto il resto. La scelta è ricaduta direttamente sia all'ultima versione del linguaggio che dell'ambiente di sviluppo perché non avevamo vincoli sulla realizzazione dell'applicazione.

Le interfacce si realizzano in modo molto semplice perchè *Xcode* consente di spostare gli elementi grafici con il *mouse* e di posizionarli come si vuole. Tuttavia, è stata la parte che ha richiesto più tempo perchè li abbiamo dovuti configurare nel modo più adatto alle nostre esigenze.



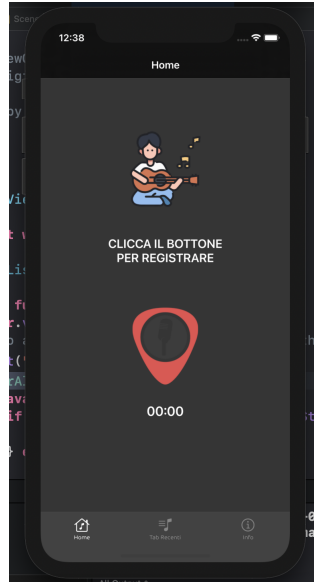
E' stata prestata anche molta attenzione a rendere compatibile l'applicazione su modelli diversi. La dimensione dello schermo influisce molto sul *layout* dell'applicazione. Senza le giuste modifiche è possibile che un elemento venga nascosto o spostato.



Le due immagini precedenti mostrano chiaramente quello appena descritto: i due dispositivi sono diversi e le proporzioni vengono rispettate in entrambi.

Inoltre, una particolare attenzione è stata dedicata anche sulla nuova modalità che sta riscontrando un grandissimo successo: la *dark mode*. Dunque, sono stati

presi tutti gli accorgimenti necessari per avere sia l'app compatibile con la versione chiara che con quella scura. L'immagine successiva mostra l'app in versione *dark*:



4.1.4 Uso di TensorFlow Lite su iOS

Abbiamo eseguito i seguenti passaggi:

- Registrato l'app sul sito *Firebase* perchè bisogna monitorarla (passaggio obbligatorio come scritto nella guida);
- Scaricato e aggiunto il file di configurazione che si ottiene dopo la registrazione su *Firebase*;
- Aggiunto *Firebase* all'app;
- Inizializzato *Firebase* nel progetto *iOS* e usate le sue *API* per usare il modello sullo *smartphone*.

4.1.5 Uso di un server per l'uso della libreria Librosa

tempo di lavoro: 5 giorni

Problemi riscontrati: purtroppo non sono state trovate librerie in grado di convertire il file audio nella trasformata a Q costante.

Soluzioni provate:

- Abbiamo provato ad usare la libreria *PythonKit* senza successo perchè sui dispositivi *iOS* manca l'interprete *Python*.

Soluzione finale: per questo motivo ci siamo serviti di un *server* che prende in ingresso la registrazione che è stata effettuata dallo *smartphone* e restituisce in uscita le immagini del file audio. Ovviamente la soluzione non è efficiente ma ai fini del progetto può andare più che bene. La predizione viene eseguita sul dispositivo e **non** sul *server*.

Il *server* è stato scritto grazie al framework di *Python* che si chiama *flask*

```
import flask

app = Flask(__name__)

@app.route('/upload/', methods = ['POST'])
def handle_request():
    #when the request arrives

if __name__=="__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```

4.2 Sviluppo applicazione Android

4.2.1 Conversione del modello da Keras a Tensorflow Lite

tempo di lavoro: 1h

Tutto ha funzionato al primo colpo senza nessun problema.

```
model = tf.keras.models.load_model("saved/model.h5",
    custom_objects={'softmax_by_string': self.softmax_by_string,
    'avg_acc': self.avg_acc, 'catcross_by_string': self.catcross_by_string})
    converter = lite.TFLiteConverter.from_keras_model(model)
    tflite_model = converter.convert()
    open("saved/model.tflite", "wb").write(tflite_model)
```

Il seguente codice consente di caricare il modello addestrato tramite *Tensorflow* e di convertirlo nel formato *.tflite* pronto per essere usato su un dispositivo mobile nel nostro caso.

4.2.2 Sviluppo applicazione con Java 1.8 e Android Studio

4.1.2

tempo di lavoro: 12 giorni

Java è una piattaforma che consente di eseguire i programmi scritti in questo linguaggio.

Android Studio è un ambiente di sviluppo integrato per lo sviluppo per la piattaforma *Android*.

L'applicazione che è stata realizzata da un punto di vista estetico è uguale a quella su *iOS*. Cambiano solo leggeri particolari che differenziano i due mondi.

Problemi riscontrati: purtroppo non sono state trovate librerie in grado di convertire il file audio nella trasformata a Q costante.

Soluzioni provate:

- Abbiamo usato la soluzione del *server* come in *iOS*.

Soluzione finale: grazie ai ricevimenti fatti con il professore, ci è stato consigliato di usare *chaquopy*.

Capitolo 5

Conclusioni

Sulle due applicazioni sono stati eseguiti diversi *test* e possiamo affermare che funziona tutto correttamente. Siamo molto soddisfatti del progetto che è stato realizzato. Siamo consapevoli che abbiamo esplorato solo una piccola parte di questa vastissima materia ma quello che abbiamo appreso sarà sicuramente usato come base di partenza in futuri progetti.

Bibliografia

- [1] *Chitarra classica*. (s.d.). Wikipedia, l'enciclopedia libera. Ultimo accesso: 28 febbraio 2021, <https://it.wikipedia.org/wiki/Chitarra>
- [2] *I tasti della chitarra*. (s.d.). TestoeAccordi. Ultimo accesso: 26 febbraio 2021, <https://www.testoeaccordi.it/menu/tasti.htm>
- [3] *Note manico chitarra* [Image]. (s.d.). Videocorsochitarra. Ultimo accesso: 27 febbraio 2021, <https://videocorsochitarra.it/note-manico-chitarra/>
- [4] Savage. N. (s.d.). *Come Leggere Tablature per Chitarra*. Wikihow. <https://www.wikihow.it/Leggere-Tablature-per-Chitarra>
- [5] Q. Xi, R. Bittner, J. Pauwels, X. Ye, and J. P. Bello, "Guitarset: A Dataset for Guitar Transcription", in *19th International Society for Music Information Retrieval Conference*, Paris, France, Sept. 2018.
- [6] Note Frequency Chart (Download) [Image], Soundonsound.com, Dec 03, 2018 11:32 am.
- [7] Drexel University, ExCITE Center, *Expressive and Creative Interaction Technologies*, NEMISIG 2019
- [8] *Utilizza un modello TensorFlow Lite per inferenza con ML Kit su iOS*. (s.d.). Firebase. Ultimo accesso: 03 marzo 2021, <https://firebase.google.com/docs/ml-kit/ios/use-custom-models>