Progetto corso Image Data Analysis aa. 2022-2023

PathMNIST Multi-class classification using ConvNets

Fratus Edoardo

Indice

1. Introduzione

Il dataset usato per il seguente lavoro è un dataset facente parte del progetto MedMNIST, una collezione di dataset contenenti immagini di carattere bio-medico in formato standard, suddivise in 12 dataset 2D e 6 dataset 3D.

Tale progetto è stato creato con lo scopo di supportare numerose ricerche e progetti in ambiti come la biomedical image analysis, la computer vision e il deep learning. L'intero progetto MedMNIST è pubblicamente disponibile alla pagina https://medmnist.com/.

1.1 Presentazione del progetto

L'obiettivo del progetto è quello di testare diversi modelli di reti convoluzionali, analizzandoli secondo diverse metriche, al fine di effettuare una classificazione multi-class. I risultati verranno poi confrontati per stabilire quale sia il modello più adatto a tale task e a tale dataset. Il progetto inizierà con un'analisi esplorativa del dataset utilizzato per l'addestramento e la valutazione dei modelli. Successivamente, discuteremo le diverse architetture di rete convoluzionale utilizzate e le tecniche di ottimizzazione adottate per addestrare i modelli. Infine, valuteremo i modelli utilizzando diverse metriche di performance, come l'accuratezza, la precisione e il recall.

Infine, verrà effettuato un tentativo di interpretazione dei risultati della rete scelta, sulla base dei kernel utilizzati.

1.2 PathMNIST

Nello specifico, per questo lavoro, si è deciso di utilizzare il dataset PathMNIST. Tale dataset è basato su un precedente studio per la predizione della sopravvivenza al cancro al colon-retto a partire da slide istologiche ed è composto da un train-set di 100'000 porzioni uniche di immagini di tessuti colorati con ematossilina ed eosina (NCT-CRC-HE-100K) e da un test-set di 7'180 porzioni di immagini uniche provenienti da un centro clinico diverso (CRC-VAL-HE-7K).

Le immagini contengono 9 diversi tipi di tessuti, rendendo tale dataset ideale per un task di classificazione multi-class. Le immagini sono disponibili in formato standardizzato e

con risoluzione di 28x28x3, risultando così delle immagini a colori. Il train-set è ulteriormente diviso in train-set vero e proprio e validation-set secondo un rapporto 9:1.

Le classi sono descritte di seguito:

0 – adipose

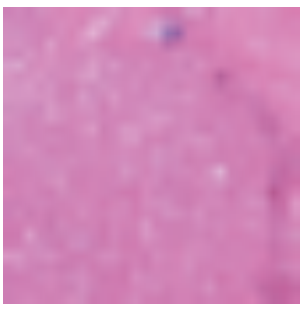
Tessuto adiposo.

1 – background

Immagine di background non utile ai fini dell'identificazione di cellule cancerogene.

2 – debris

Immagine di scarto ottenuta da una fase di preparazione o di acquisizione dell'immagine non corretta. Non è quindi utile ai fini dell'identificazione di cellule cancerogene.



3 – lymphocytes

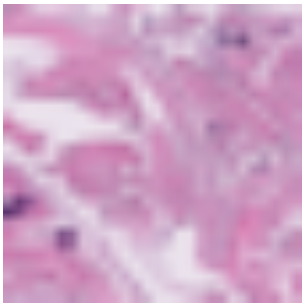
Tessuto contenente linfociti. La loro presenza può indicare un'infezione o una malattia auto-immune.

4 – mucus

Tessuto con presenza di muco. La sua presenza potrebbe indicare la presenza di malattia del tessuto in esame.

5 – smooth muscles

Normale tessuto muscolare liscio.



6 – normal colon mucosa

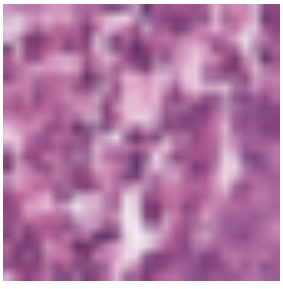
Tessuto della mucosa del colon identificato come normale.

7 – cancer associated stroma

Tessuto che circonda un tumore del colon-retto. La sua presenza è utilizzata per determinare la prognosi e la previsione del tumore.

8 – colorectal adenocarcinoma epithelium

Tessuto tumorale.



Ad un'analisi effettuata tramite le librerie Pandas e Matplotlib di Python, il dataset appare sbilanciato a favore di immagini di classe 5 e 8. Nonostante questo leggero sbilanciamento, si è deciso di non intervenire e di lasciare il dataset inalterato, per tre motivi principali:

- Il fine ultimo della classificazione è individuare le immagini di tessuti tumorali, di conseguenza l'abbondanza di dati di tale classe non è stata ritenuta un problema.
- La distribuzione delle immagini potrebbe rappresentare una distribuzione di probabilità di incontrare tali tipi di tessuto all'interno degli esami.
- Lo sbilanciamento non aveva una magnitudine tale da inficiare l'allenamento delle reti e quindi i risultati della classificazione.

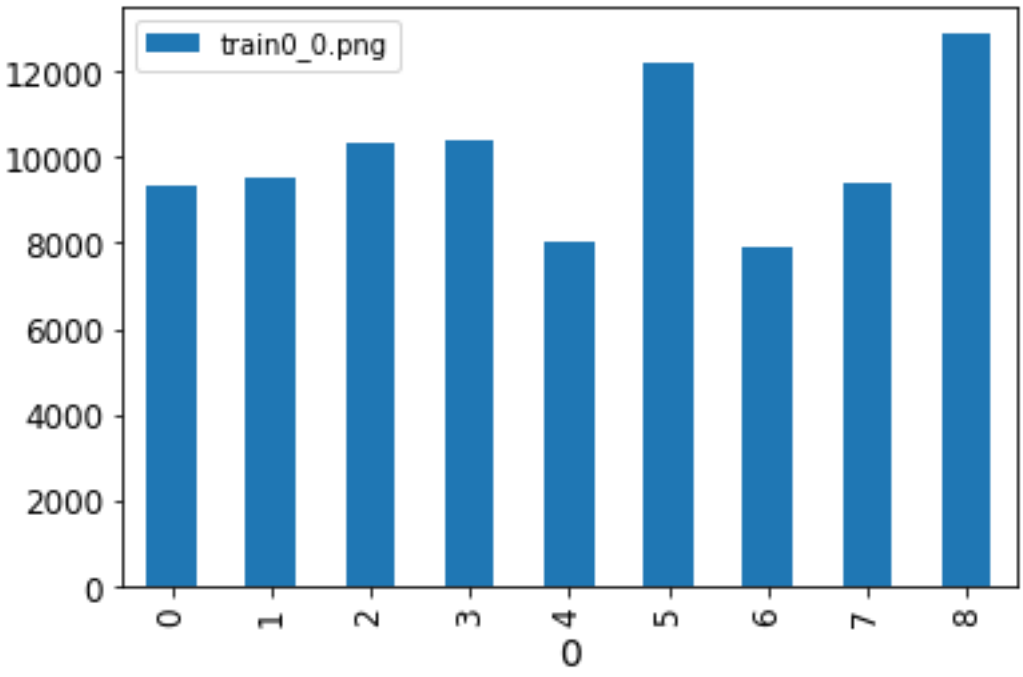


Figura 1: barplot delle occorrenze delle classi nel train-set

Essendo le immagini già state preparate da chi ha curato il dataset, è stato deciso di non effettuare alcun tipo di pre-processing o di data curation su di esse, e di allenare direttamente i modelli sulle immagini fornite.

2. Definizione ed allenamento delle reti

Ogni rete testata in questo progetto è stata allenata utilizzando l'intero train-set e facendo uso del validation-set per prevenire l'overfitting. Nello specifico, avendo molte istanze di training, sono stati utilizzati dei generatori, come visibile nella Figura 2.

Per il training è stato deciso di utilizzare un batch_size di 32 immagini. Inoltre, ogni immagine è stata ridimensionata per adeguarsi alle specifiche in input delle reti create.

```
from keras_preprocessing.image import ImageDataGenerator
import pandas as pd

traindf=pd.read_csv('train/pathmnist.csv',dtype=str)
datagen=ImageDataGenerator(rescale=1./255.)

train_generator=datagen.flow_from_dataframe(
  dataframe=traindf,
  directory="train/pathmnist",
  x_col="train0_0.png",
  y_col="0",
  subset="training",
  batch_size=32,
  seed=42,
  shuffle=True,
  class_mode="categorical",
  target_size=(64,64))

valdf = pd.read_csv('validation/pathmnist.csv',dtype=str)
val_generator=datagen.flow_from_dataframe(
  dataframe=valdf,
  directory="validation/pathmnist",
  x_col="val0_5.png",
  y_col="5",
  batch_size=32,
  seed=42,
  shuffle=True,
  class_mode="categorical",
  target_size=(64,64))
```

```
Found 89995 validated image filenames belonging to 9 classes.
Found 10003 validated image filenames belonging to 9 classes.
```

Figura 2: generator per train e validation set

Essendo in un problema medico molto importante valutare non solo l'accuratezza della rete, ma anche la precisione nell'identificazione di un caso positivo e la percentuale di casi positivi individuata, si è deciso di tenere traccia anche della **precision** e del **recall** delle reti neurali durante l'allenamento.

Inoltre, sempre per evitare l'overfitting, è stato deciso di applicare un metodo di **early termination** alla rete, in modo da stoppare l'addestramento in caso di overspecializzazione e conseguente stallo della validation_loss.

Tutte le reti sono state quindi allenate secondo i metodi presentati in Figura 3, utilizzando le metriche sopra indicate, "categorical cross-entropy" come loss-function e "adam" come optimizer.

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=[recall,precision,'accuracy'])
```

```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

```
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=val_generator.n//val_generator.batch_size
history = model.fit(train_generator,steps_per_epoch=STEP_SIZE_TRAIN,validation_data=val_generator,
                    validation_steps=STEP_SIZE_VALID,callbacks = callback, epochs=20)
```

Figura 3: metodi per allenamento delle reti

Infine, la history del training di ogni rete è stata plottata attraverso i metodi riportati in figura 4, al fine di favorire la comprensione dell'andamento dell'allenamento,

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

prec = history.history['precision']
val_prec = history.history['val_precision']

recall = history.history['recall']
val_recall = history.history['val_recall']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.figure()

plt.plot(epochs, prec, 'bo', label='Training precision')
plt.plot(epochs, val_prec, 'b', label='Validation precision')
plt.title('Training and validation precision')
plt.legend()

plt.figure()

plt.plot(epochs, recall, 'bo', label='Training recall')
plt.plot(epochs, val_recall, 'b', label='Validation recall')
plt.title('Training and validation recall')
plt.legend()
```

Figura 4: metodi per plotting dei grafici che descrivono l'allenamento

2.1 A simple CNN (Baseline Model)

La prima rete che si è deciso di testare è una Convolutional Neural Network la cui struttura, per scelta, è stata mantenuta abbastanza semplice, al fine di poter effettuare un training in tempi brevi e poterlo ripetere svariate volte.

Tale rete prende in input immagini con dimensioni 64x64px x3 canali e lavora con una sequenza di kernel 5x5 e 3x3 e layer di MaxPooling fino ad ottenere immagini di 4x4px x 256 canali, sfruttando quindi il classico modo di procedere con le reti convoluzionali.

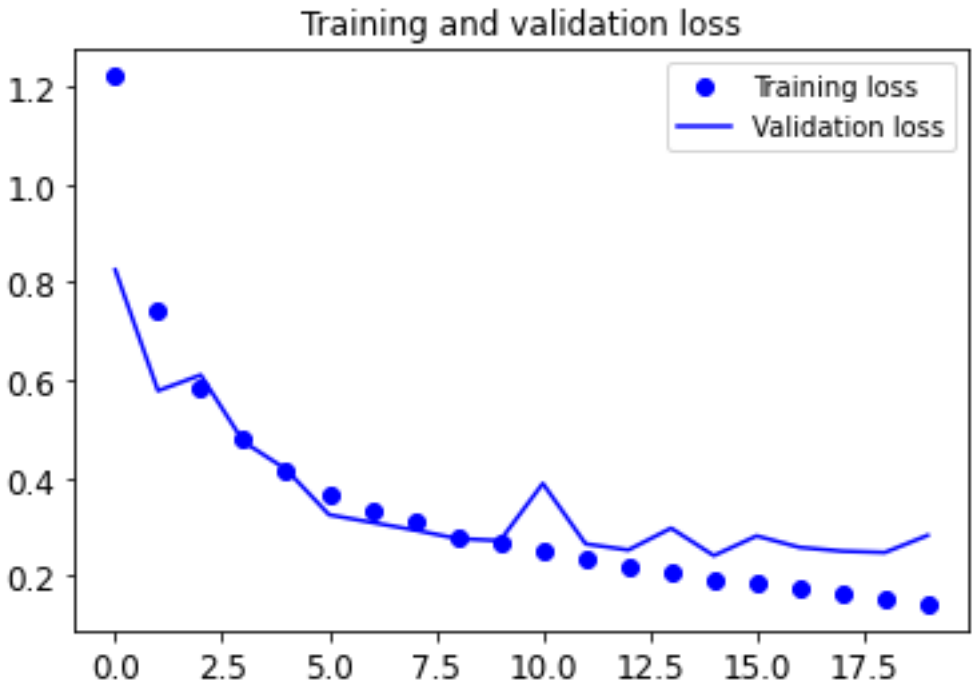
Al termine dei layer di convoluzione, dopo alcuni test, si è deciso di utilizzare un layer di GlobalAveragePooling al fine di riassumere i dati e trasformarli in formato sequenziale, piuttosto che un flatten come nelle architetture proposte a lezione.

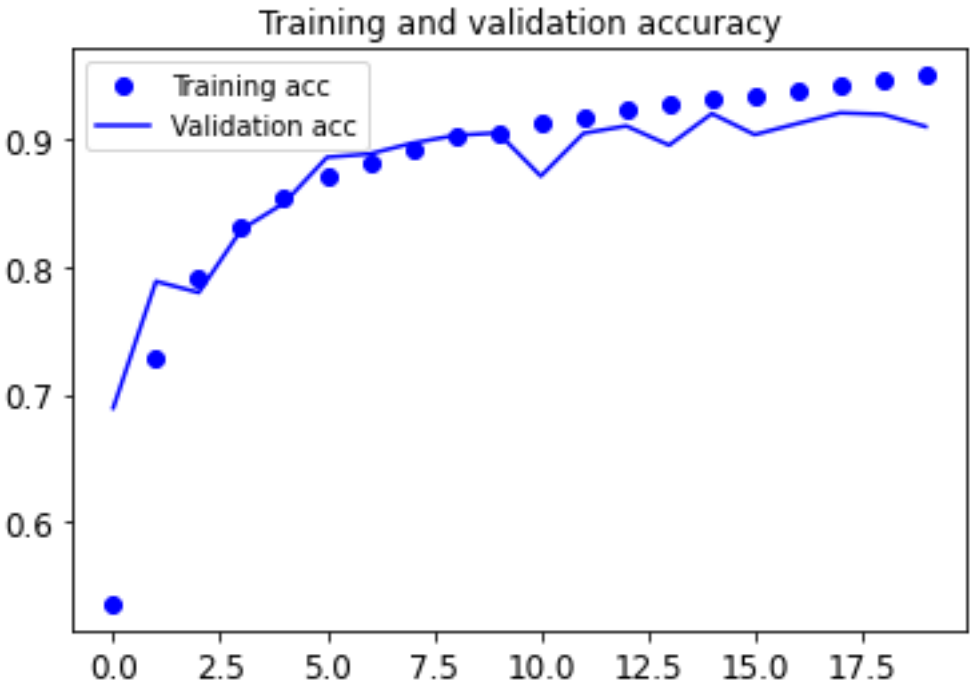
A questo punto sono stati aggiunti due Dense Layer, un layer di Dropout, ed un layer finale di output con funzione di attivazione Softmax al fine di effettuare la classificazione sulla base delle informazioni ottenute dai layer precedenti. Si è deciso di utilizzare il Dropout con un rate di 0.2 poiché è stato notato che tale scelta consentiva di ottimizzare le performance in allenamento e diminuire ulteriormente l'overfitting, oltre che limitare il numero di parametri della rete.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 60, 60, 32)        2432

 max_pooling2d_3 (MaxPooling  (None, 30, 30, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 28, 28, 64)        18496

 max_pooling2d_4 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 12, 12, 128)       73856

 max_pooling2d_5 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 conv2d_7 (Conv2D)           (None, 4, 4, 256)         295168

 global_average_pooling2d_1   (None, 256)              0
 (GlobalAveragePooling2D)

 dense_3 (Dense)             (None, 128)               32896

 dropout_1 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 9)                 585

=================================================================
Total params: 431,689
Trainable params: 431,689
Non-trainable params: 0
_________________________________________________________________
```

Figura 5: struttura del baseline model

Con gli accorgimenti descritti in precedenza si è riusciti quindi a tenere basso il numero di parametri, consentendo un allenamento snello, che ha portato ai risultati espressi nelle Figure 6 e 7.





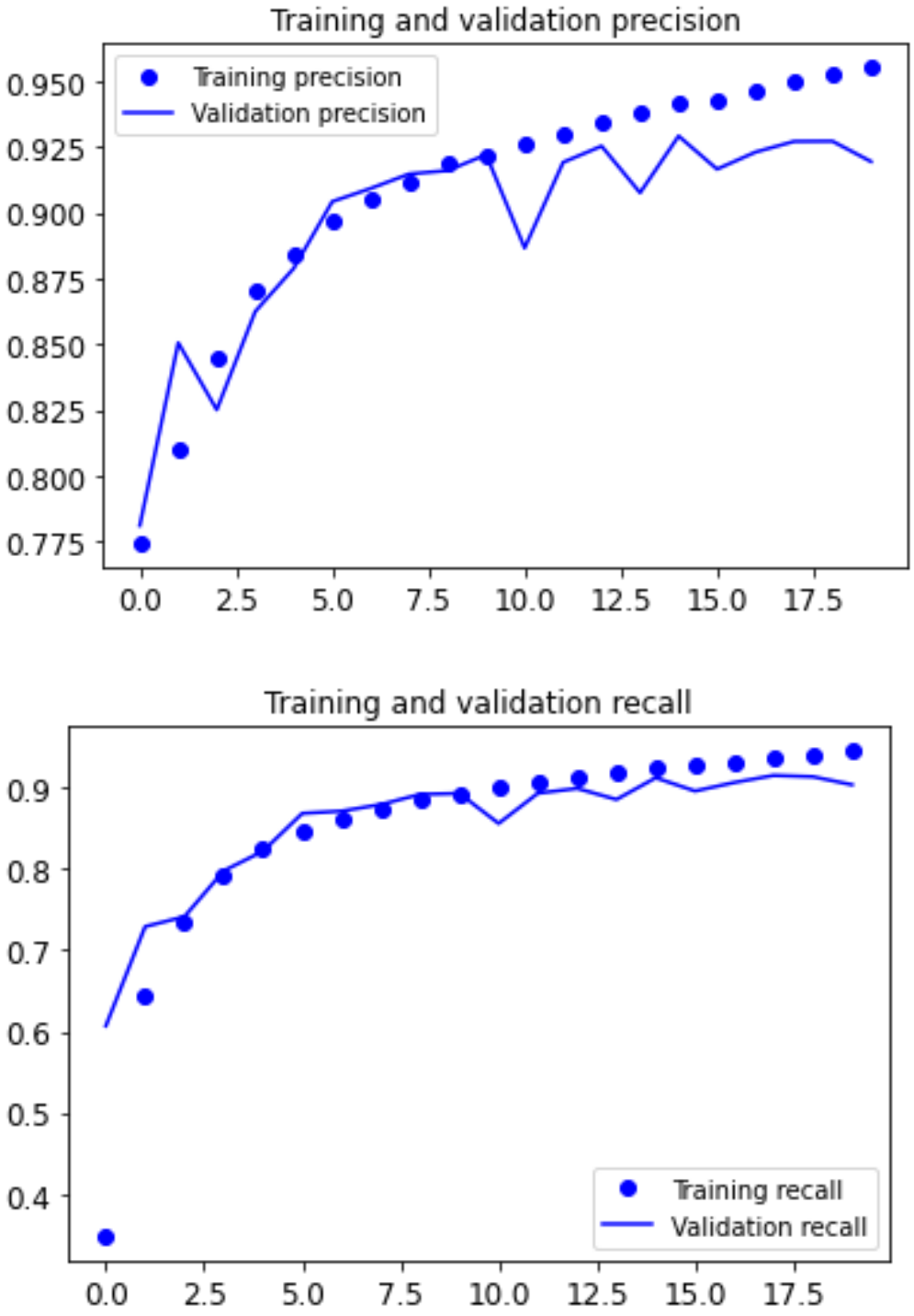


Figura 6: Analisi dell'addestramento del baseline model

	Train-set	**Validation-set**	**Test-set**
Accuracy	0.950	0.910	0.849
Precision	0.955	0.919	0.870
Recall	0.945	0.903	0.859

Figura 6: Risultati del baseline model

Considerando che il test set, come riportato nel paragrafo 1.2, è ottenuto da immagini appartenenti ad un centro clinico diverso, questa prima rete ha ottenuto dei risultati che sono stati giudicati buoni, dimostrando anche buone capacità di generalizzazione, anche su immagini che differivano da quelle usate per il training e la validazione. Di conseguenza, si è deciso di utilizzare tale modello come **baseline** per la valutazione dei modelli successivi.

L'addestramento di tale modello ha richiesto circa 20 minuti, dimostrando così un perfetto bilanciamento fra risultati e potenza di calcolo necessaria / complessità del modello.

2.2 VGG_16

La seconda rete utilizzata è la rete VGG_16. Tale rete utilizza una serie di VGG block, composti da operazioni di convoluzione, padding e pooling. In particolare, in una struttura VGG_16, tale blocco viene ripetuto 16 volte.

Si è importato quindi tale modello da Keras, a cui sono stati aggiunti 2 Dense Layer, un layer di Dropout e il layer finale di output.

La rete è stata addestrata per 20 epoche con i livelli convoluzionali bloccati, in modo da consentire un iniziale allenamento dei livelli densi. Successivamente è stato sbloccato l'allenamento della rete VGG ed è stato effettuato un **fine tuning** del modello per altre 10 epoche con un learning rate di 1e-5. Tale fine tuning ha consentito la specializzazione dell'intera rete, che era stata precedentemente allenata su ImageNet.

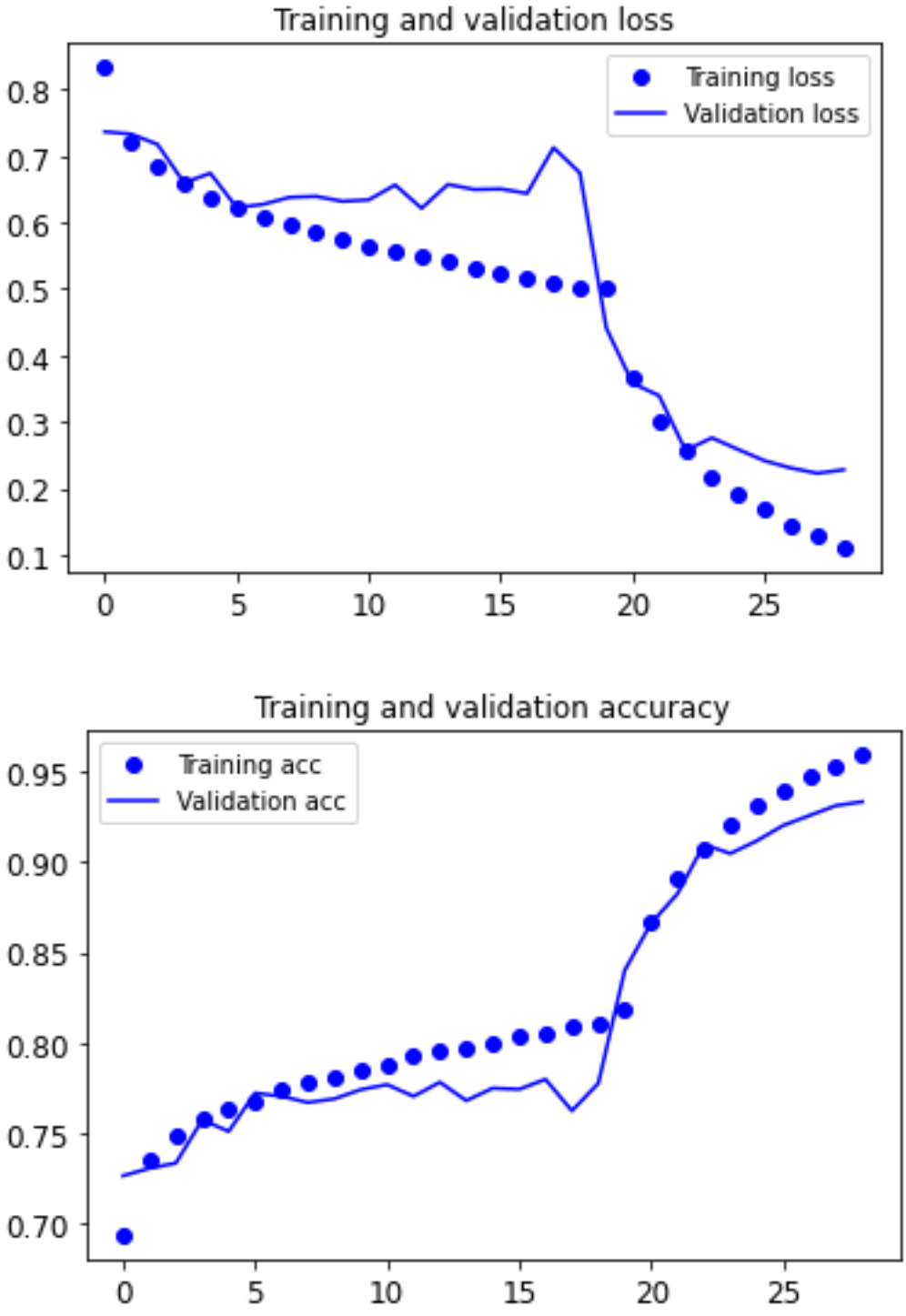
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0
block5_conv1 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv2 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv3 (Conv2D)	(None, 2, 2, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0

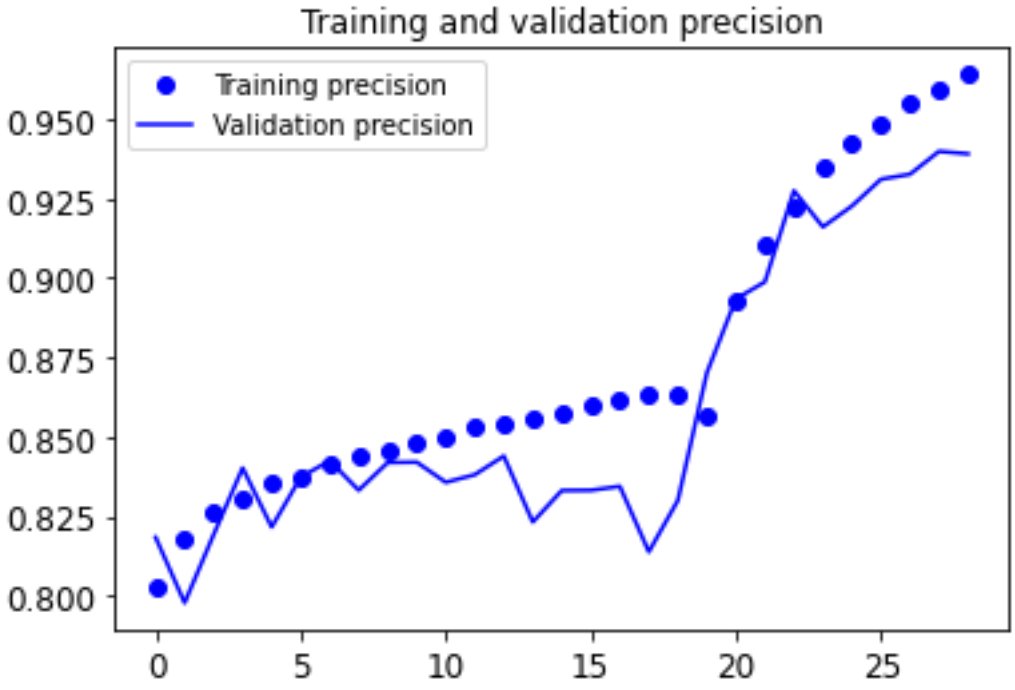
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

Figura 8: struttura di VGG_16

Come è possibile vedere nella figura 8, tale rete risulta ben più complessa del modello usato come baseline, avendo ben 14 milioni di parametri addstrabili.

Le performance di VGG_16 sono riportate nelle immagini seguenti.





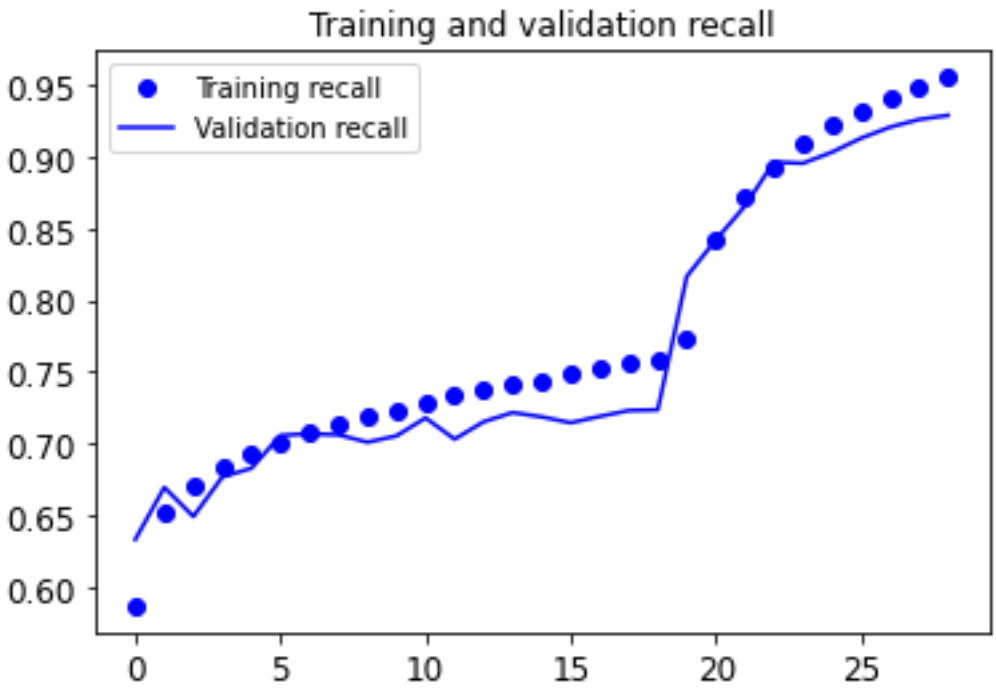


Figura 9: Performance di training di VGG_16

Dalle immagini è possibile notare un accenno di overfitting nelle fasi finali del primo training. Inoltre è possibile notare che il fine tuning ha avuto un'importanza notevole in quanto tutti i grafici impennano in prossimità dell'inizio del seconod training.

	Train-set	**Validation-set**	**Test-set**
Accuracy	0.959	0.933	0.861
Precision	0.964	0.939	0.870
Recall	0.954	0.928	0.856

Figura 10: risultati ottenuti da VGG_16

Dalla tabella dei risultati possiamo osservare come le performance siano nettamente migliori del modello usato come baseline, guadagnando circa 10 punti percentuali in tutte le metriche sul test set.

Questo però a discapito delle prestazioni, infatti l'addestramento di tale modello ha richiesto in totale più di 120 minuti.

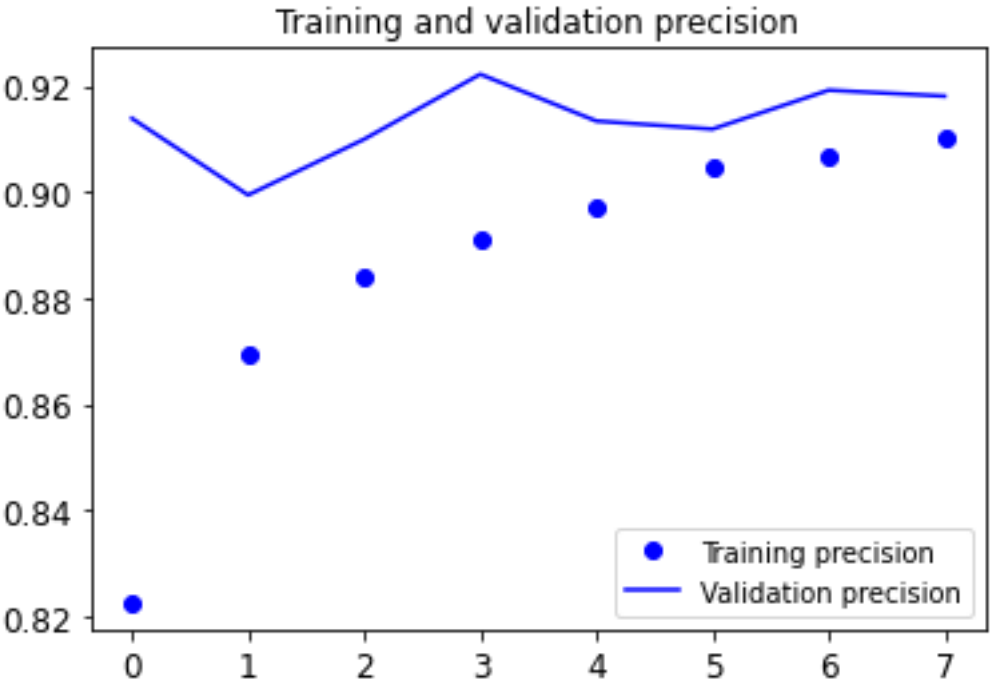
2.3 Xception

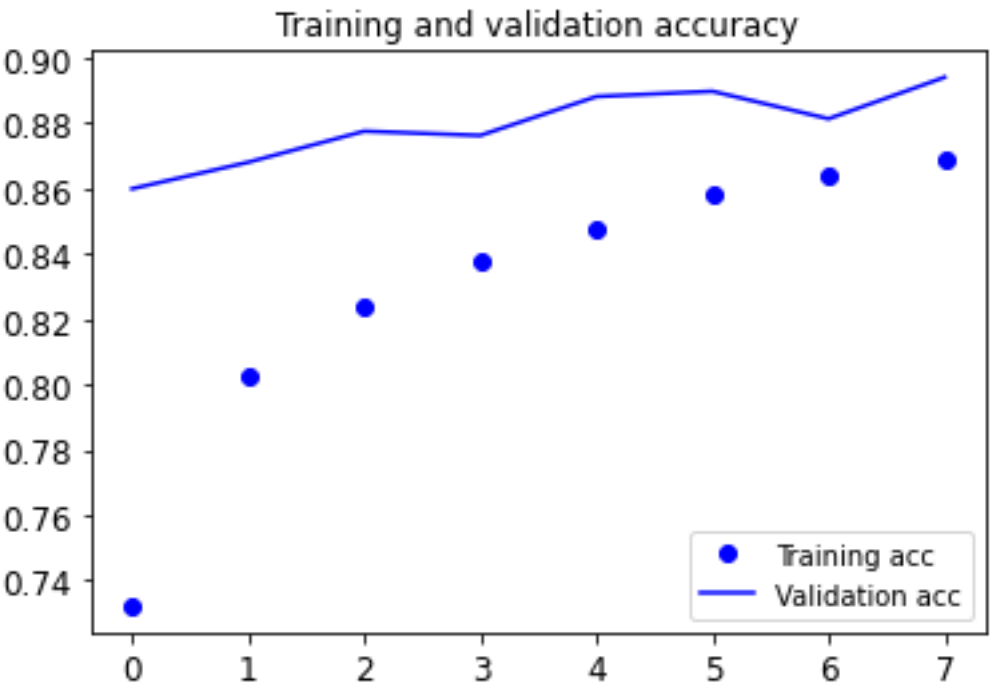
La terza rete utilizzata è la rete Xception, ottenuta tramite evoluzione della famosa architettura Inception di proprietà di Google. Tale architettura evolve la precedente sostituendo gli Inception Modules con una depthwise separable convolution. Questo modello ha over-performato rispetto ad Inception V3 sul dataset ImageNet, pur mantenendo lo stesos numero di parametri.

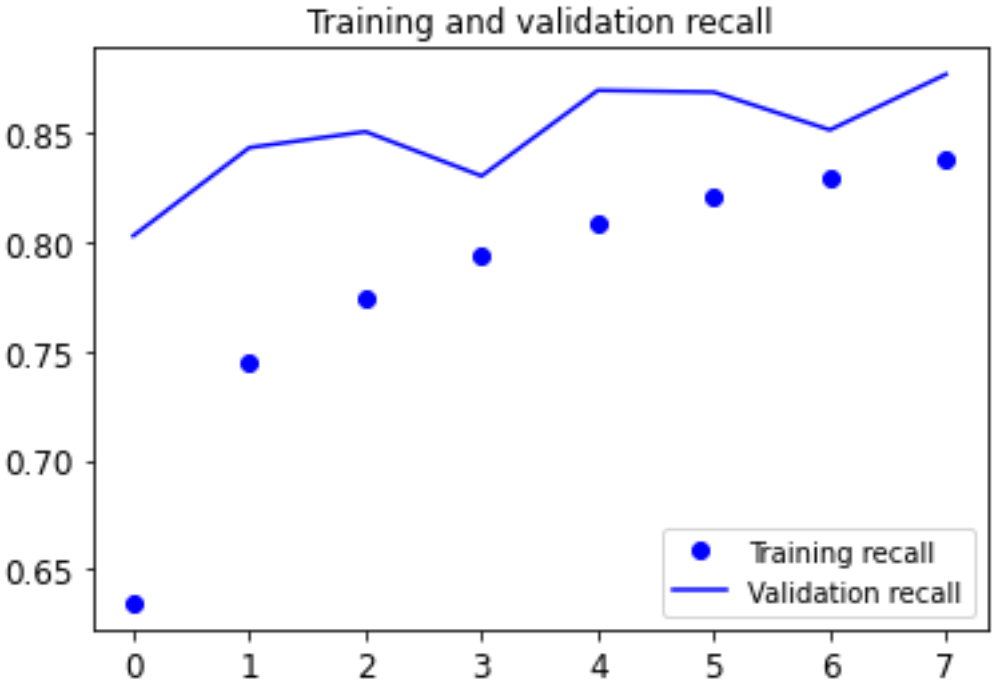
Essendo tale modello enorme, circa 73 milioni di parametri, si è deciso di non eseguire un fine tuning, in quanto sarebbe stato impossibile con la capacità di calcolo a disposizione. Di conseguenza la rete è stata addestrata solamente per quanto riguarda i livelli finali di classificazione, affidandosi totalmente al modello pre-addestrato per la parte di convoluzione.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 xception (Functional)        (None, 10, 10, 2048)      20861480

 flatten_1 (Flatten)          (None, 204800)            0

 dense_2 (Dense)              (None, 256)               52429056

 dropout_1 (Dropout)          (None, 256)               0

 dense_3 (Dense)              (None, 9)                 2313

=================================================================
Total params: 73,292,849
Trainable params: 52,431,369
Non-trainable params: 20,861,480
_________________________________________________________________
```

Figura 11: modello di Xception


Training and validation precision
0.92
0.90
0.88
0.86
0.84
0.82
0
1
2
3
4
5
6
7
Training precision
Validation precision


Training and validation accuracy
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0
1
2
3
4
5
6
7
Training acc
Validation acc



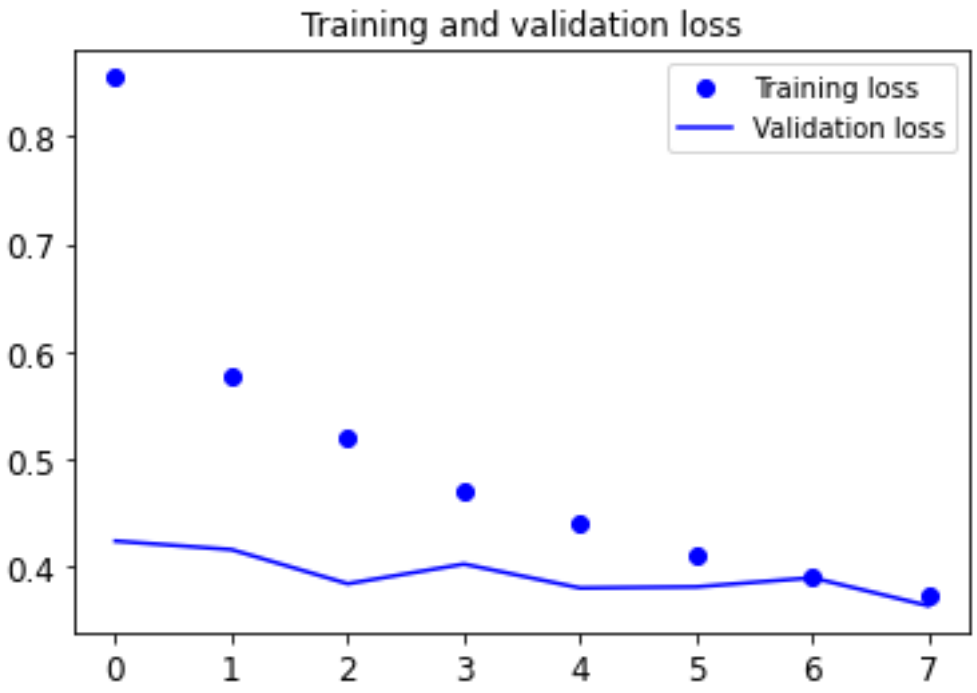


Figura 12: history allenamento Xception

	Train-set	Validation-set	Test-set
Accuracy	0.868	0.894	0.877
Precision	0.910	0.918	0.915
Recall	0.837	0.877	0.894

2.4 Conclusioni

Al termine dell'addestramento delle 3 reti appare evidente che l'architettura Xception, pur senza fine tuning, si sia rivelata la migliore per quanto riguarda le performance sul test set. Questo era prevedibile in quanto si tratta di una rete molto più elaborata e complessa rispetto alle precedenti, e di conseguenza dotata di maggior potere di generalizzazione, come è stato dimostrato.

Nonostante questo, si sono ritenuti positivi anche i risultati ottenuti con gli altri due modelli testati, che hanno ottenuto comunque delle buone performance, tenendo anche in considerazione la minor potenza computazionale necessaria per il loro allenamento.

Tale risultato lascia la porta aperta per eventuali evoluzioni future del progetto, avendo a disposizione una maggiore potenza computazionale o maggior tempo sarebbe infatti possibile migliorare ulteriormente i risultati ottenuti, specializzando Xception su immagini in campo medico tramite Fine Tuning.

Nell'ultima parte di questa relazione si proverà a dare un significato ed una spiegazione dei risultati ottenuti dalla rete baseline.

3. ConvNet Explainability

In quest'ultima sezione si proverà a dare una spiegazione ed un'interpretazione ai risultati ottenuti tramite la rete proposta come baseline model.

È stato scelto di effettuare tale analisi su questa rete a causa della sua semplicità, effettuare questo tipo di analisi su una delle due reti addestrate successivamente avrebbe comportato una complessità molto maggiore.

Innanzitutto, osservando la Confusion Matrix delle prediction della rete è possibile osservare che essa sta lavorando bene:

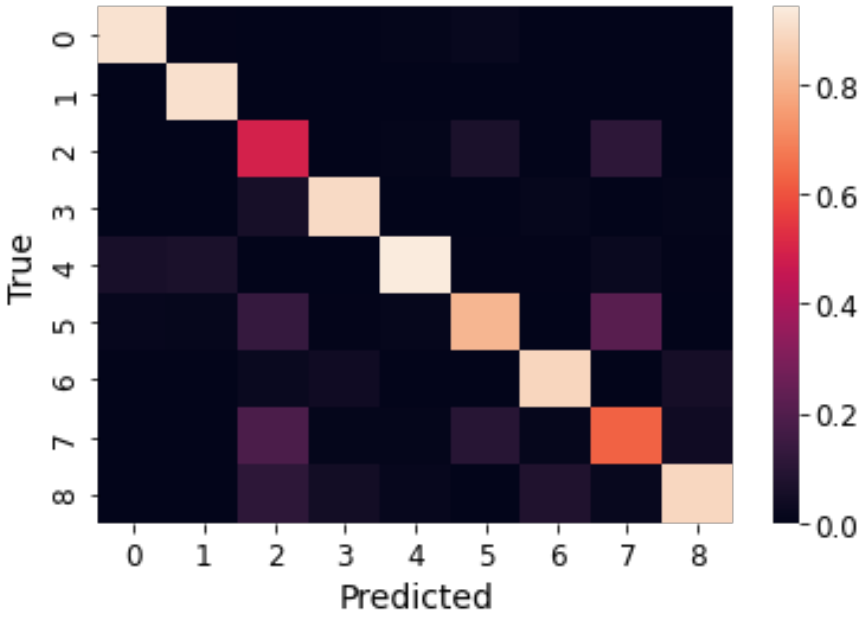


Figura 13: confusion matrix baseline model

Per effettuare tale tipo di analisi è necessario poter ottenere diversi output a partire dalla stessa rete, in particolare siamo interessati ad ottenere in output il risultato di ogni livello di convoluzione. Per farlo sono stati utilizzati i seguenti metodi e oggetti di Keras:

```
from keras import models

# uso una list comprehension per creare una lista in modo semplice prendendo solamente l'output di
# ogni livello della rete
layer_outputs = [layer.output for layer in model.layers[1:7]]

# Creates a model that will return these outputs, given the model input:
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
```

Figura 14: utilizzo di Models in Keras

Successivamente, sono state prese 2 immagini di test, in modo da poter visualizzare gli output della rete sia per un'immagine classificata come tessuto sano, sia per un'immagine classificata come tessuto tumorale ed osservare le differenze.

Le immagini scelte sono le seguenti:

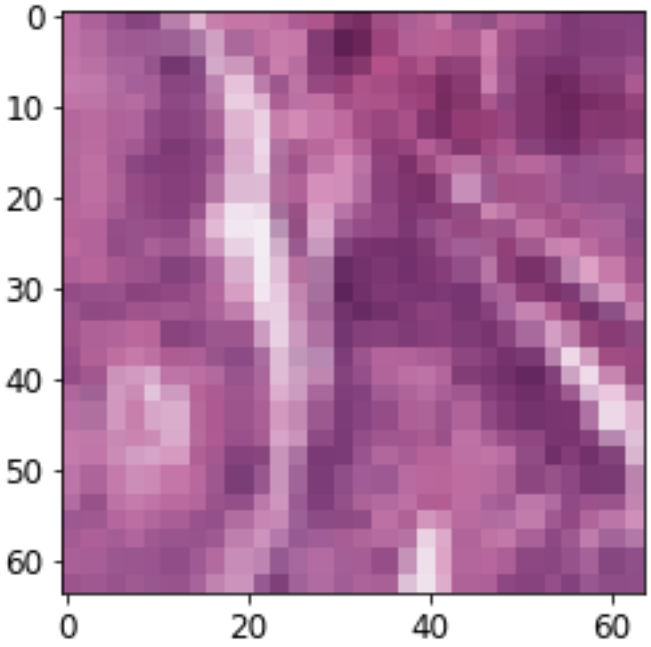


Figura 15: tessuto tumorale

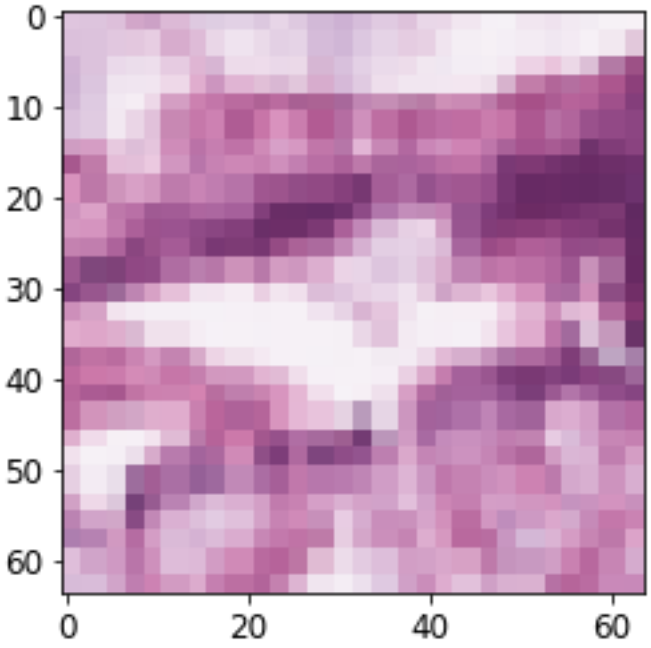
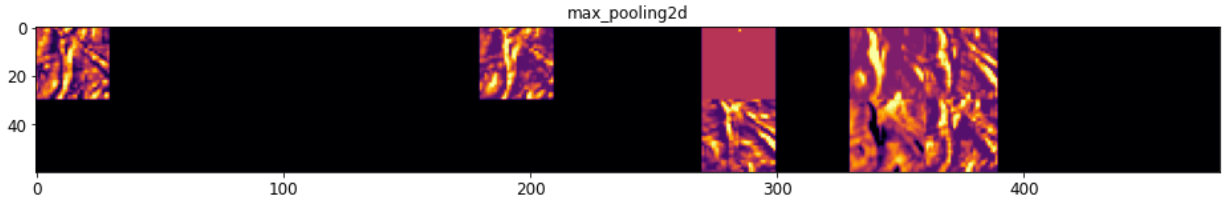
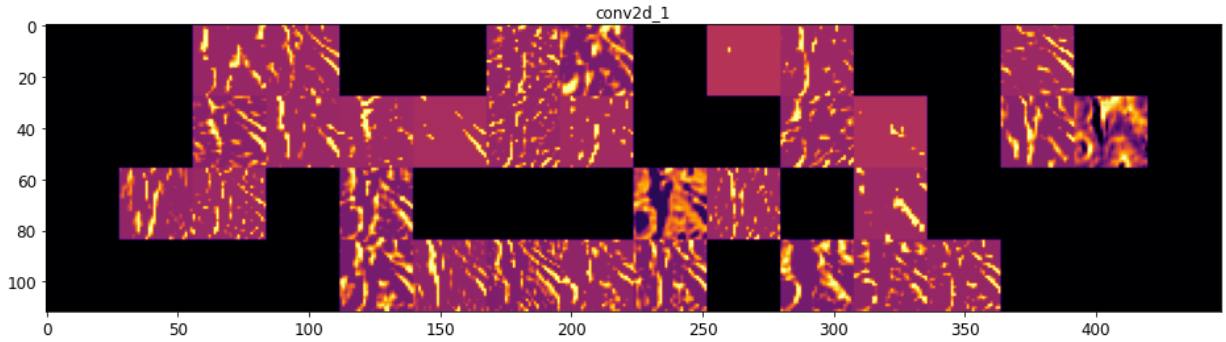


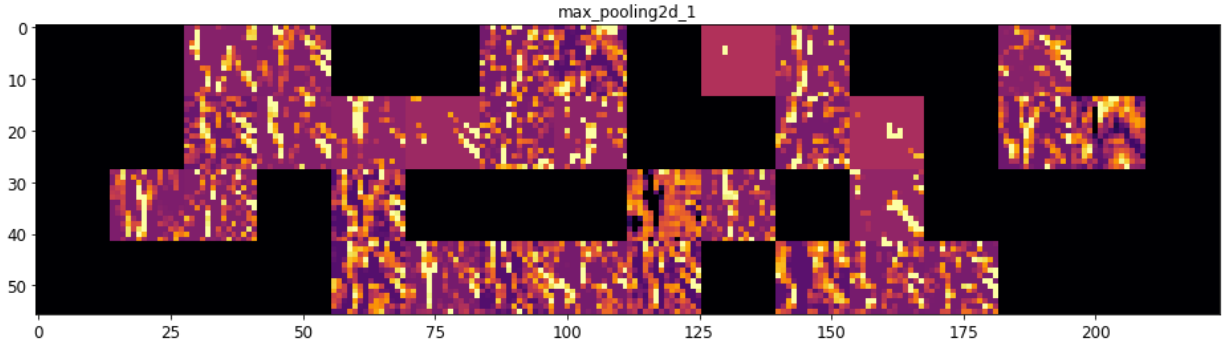
Figura 16: tessuto sano

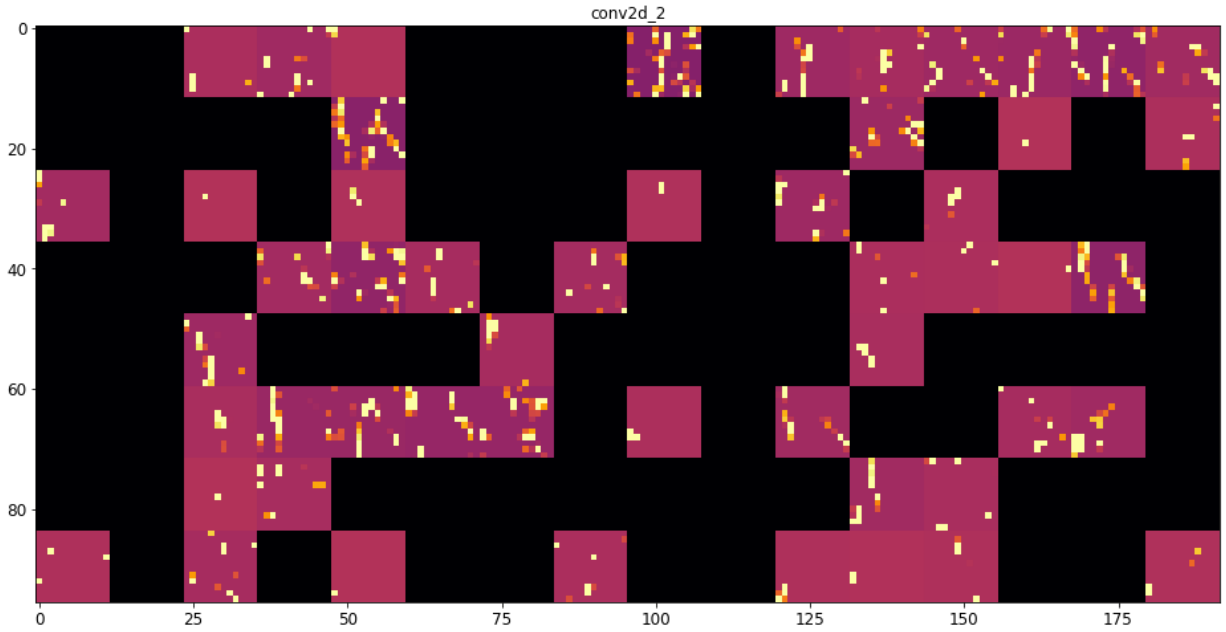
3.1 Tessuto tumorale

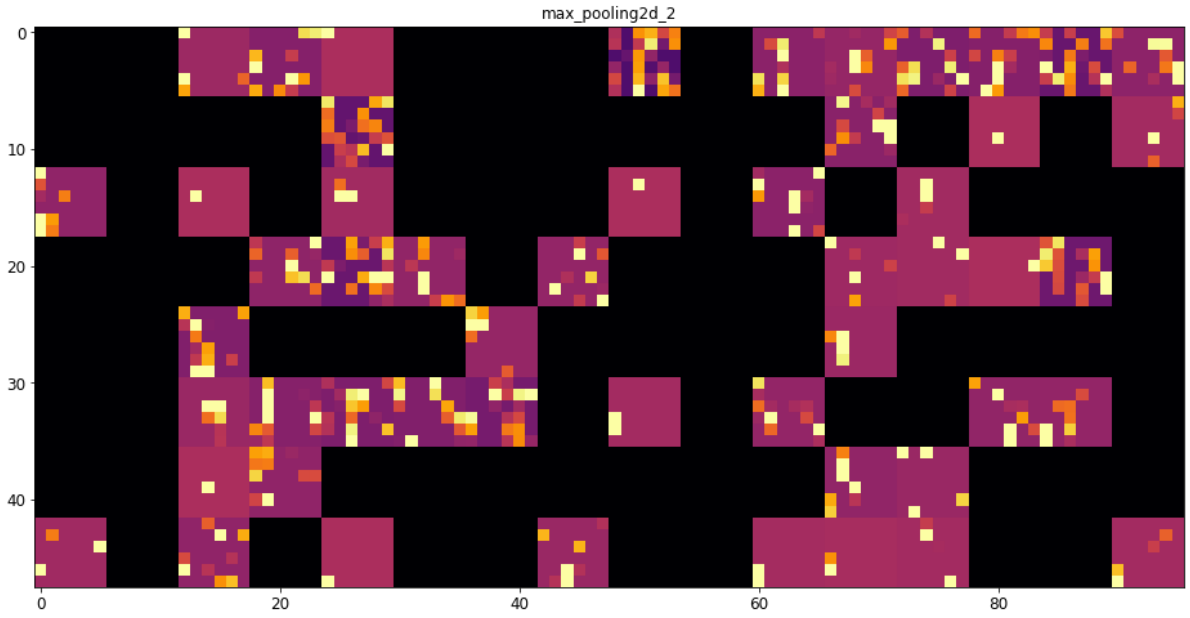
Di seguito è riportato l'output di ogni livello:

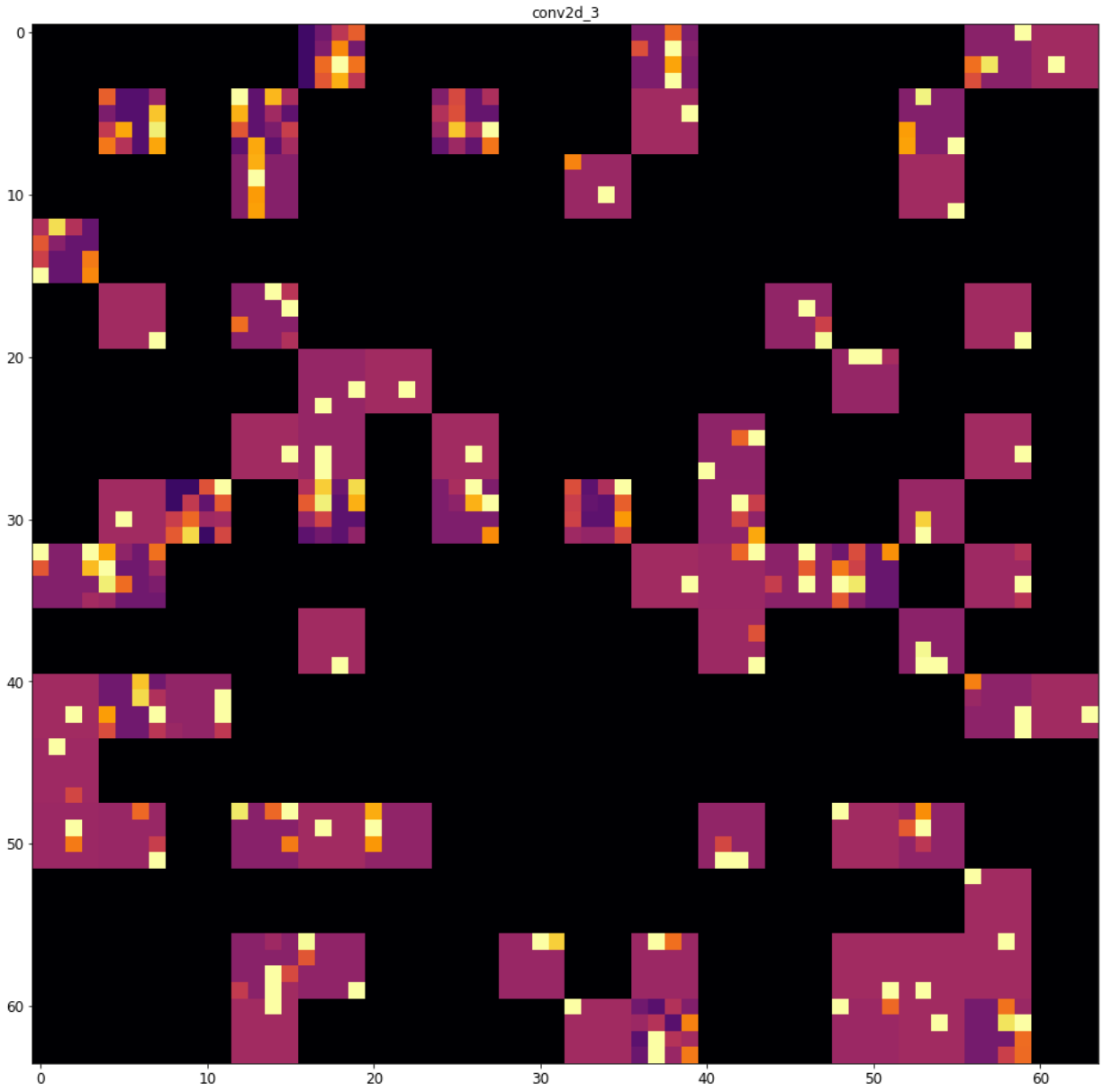






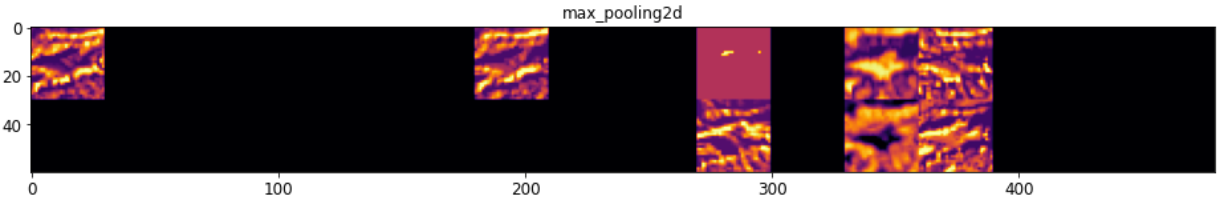


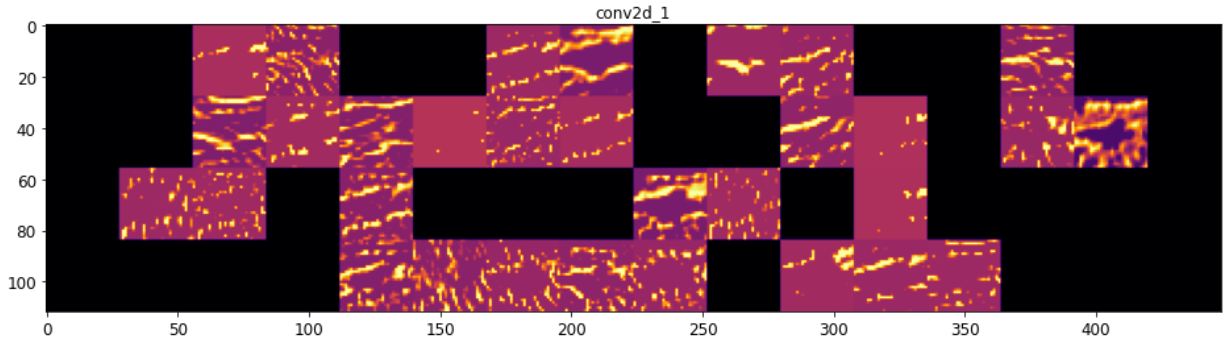

max_pooling2d_2
0
10
20
30
40
0
20
40
60
80

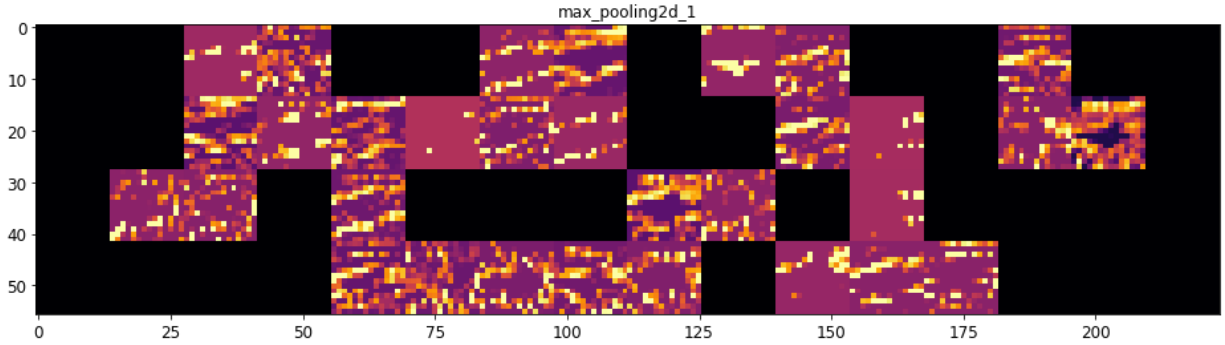

conv2d_3
0
10
20
30
40
50
60
0
10
20
30
40
50
60

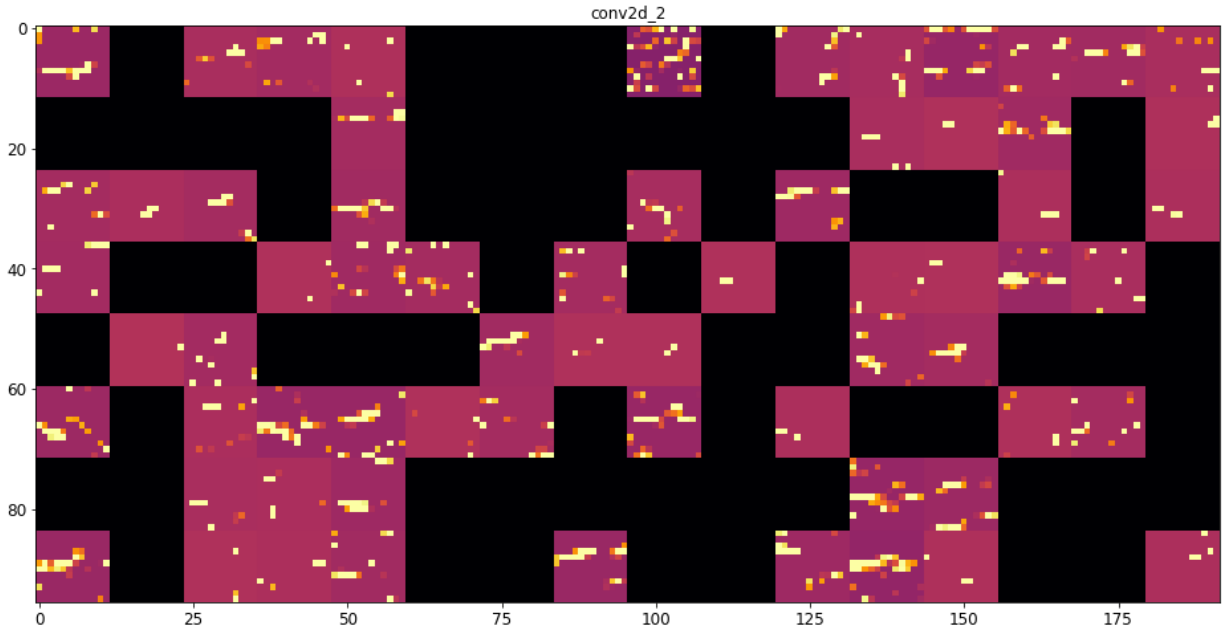
3.2 Tessuto sano

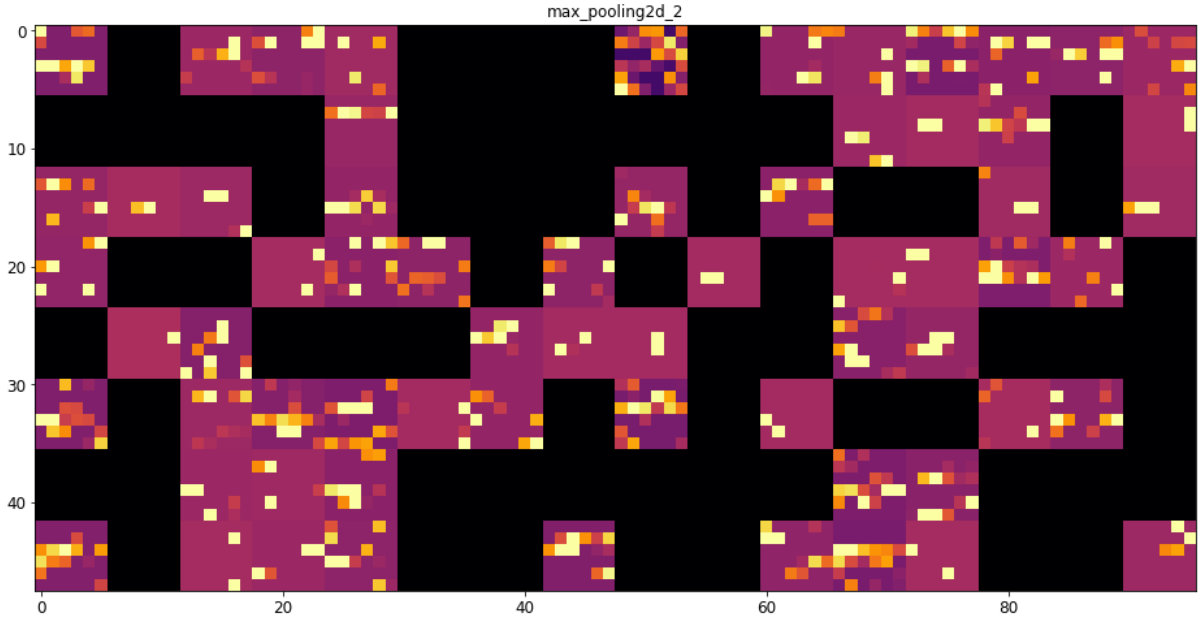
Di seguito è riportato l'output di ogni livello:

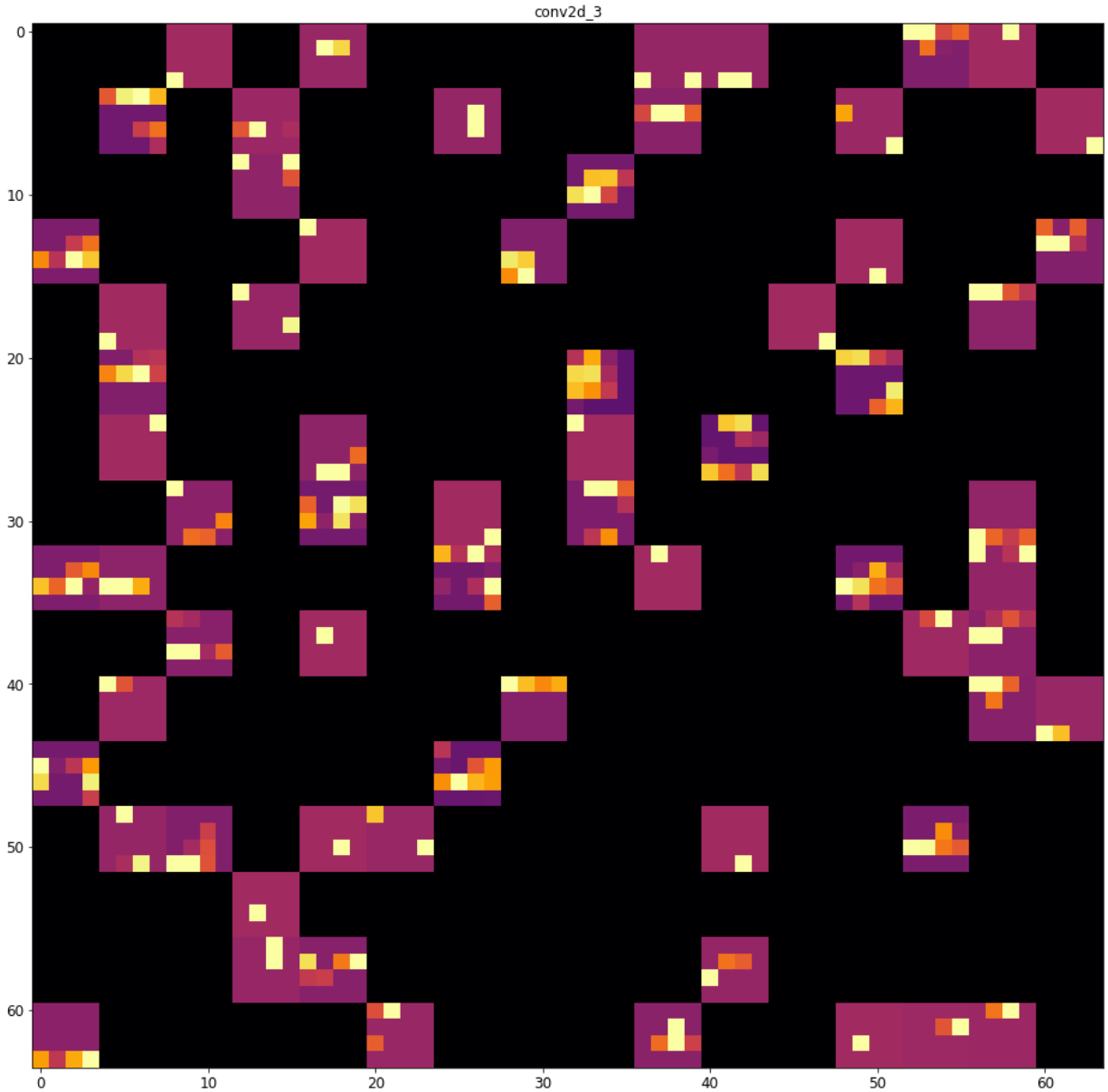









max_pooling2d_2
0
10
20
30
40
0
20
40
60
80


conv2d_3
0
10
20
30
40
50
60
0
10
20
30
40
50
60

3.3 Conclusioni

Innanzitutto, dalle immagini precedenti è possibile osservare come lavora la ConvNet:

- Inizialmente vengono usati diversi kernel per mettere in luce diverse caratteristiche dell'immagine, ad esempio linee orizzontali, verticali o oblique.
- Con l'aumentare della profondità della rete, il focus è sempre più su una porzione di immagine. La rete, quindi cerca di estrarre feature di più alto livello.
- Ad ogni livello di pooling la risoluzione viene dimezzata, e ciò consente di migliorare la pose invariance della rete.

Osservando solamente queste due immagini di test, dagli output analizzati è possibile osservare come la rete, per l'immagine contenente tessuto tumorale, abbia dato molta più importanza alle features verticali, cosa che non è successa invece per l'immagine del tessuto sano.