Università degli Studi di Ferrara

Università degli Studi di Ferrara

DE: Department of Engineering Ferrara

Damiano Bressanin 138075

DEEP LEARNING

PRESENTAZIONE DEL PROGETTO

Descrizione del Progetto

Si vogliono confrontare diversi approcci ed architetture di *Convolutional Neural Network* con lo scopo di eseguire classificazione binaria (Hard Prediction) di immagini di cani e gatti con Dataset bilanciato.

APPROCCIO RISOLUTIVO

Inizialmente, dopo aver studiato il Dataset, ho risolto il problema in molti modi diversi andando a salvare i risultati ottenuti in modo da poter confrontare i diversi approcci.

Per i primi test ho definito un'architettura «base» di CNN andando ad analizzare il suo comportamento ed i risultati ottenuti al variare dei dati in ingresso (ripartizioni dataset, Data Agumentation e colori).

Successivamente ho provato ad utilizzare KerasTuner per la ricerca automatica degli iperparametri.

Poi ho creato due varianti dell'architettura base andando ad modificare la rete in modi differenti.

Infine ho utilizzato il Transfer Learning e Fine Tuning con delle reti preaddestrate.

TECNOLOGIE UTILIZZATE

- **Hardware (Laptop):**
  - ❑ CPU: Intel i7-7700HQ
  - ❑ GPU: Nvidia GeForce GTX 1070 Mobile 8 GB DDR5
  - ❑ RAM: 32 GB DDR4 2400 MHz
  - ❑ Storage: Samsung SSD 980 1 TB NVMe
    Samsung SSD 870 EVO 1 TB SATA

- **Software e librerie principali**:
  - ❑ Windows 10
  - ❑ Docker Desktop
  - ❑ JupyterLab
  - ❑ Python 3.8.10
  - ❑ Tensorflow 2.13.0
  - ❑ Keras
  - ❑ Matplotlib
  - ❑ Scikit-learn
  - ❑ Pandas
  - ❑ Numpy
  - ❑ Os
  - ❑ Shutil

PERCHÉ DOCKER?

Inizialmente avevo utilizzato la distribuzione di Python “Anaconda” ed un ambiente virtuale ma, per problemi di configurazione, non rileva in maniera corretta la GPU costringendomi ad utilizzare la CPU.

Per risolvere questo problema ho utilizzato un’Immagine Docker che contiene Tensorflow con supporto a GPU e ho creato un Container seguendo questa guida:

“*Tensorflow with GPU on Windows WSL using Docker*”

https://youtu.be/YozfiLI1ogY?si=BHqM7918FQKOX9bw

In questo modo tramite WSL avvio il Container e JupyterLab e posso procedere avendo il supporto alla GPU.

Uno svantaggio di questa soluzione è l’elevato consumo di risorse da parte della virtualizzazione (WSL e Docker).
Dopo alcuni test si è rivelata comunque una soluzione nettamente migliore rispetto all’utilizzo della CPU per l’addestramento delle reti.

ARCHITETTURA «BASE»

Ora verranno illustrate le scelte progettuali eseguite per la costruzione della prima CNN.

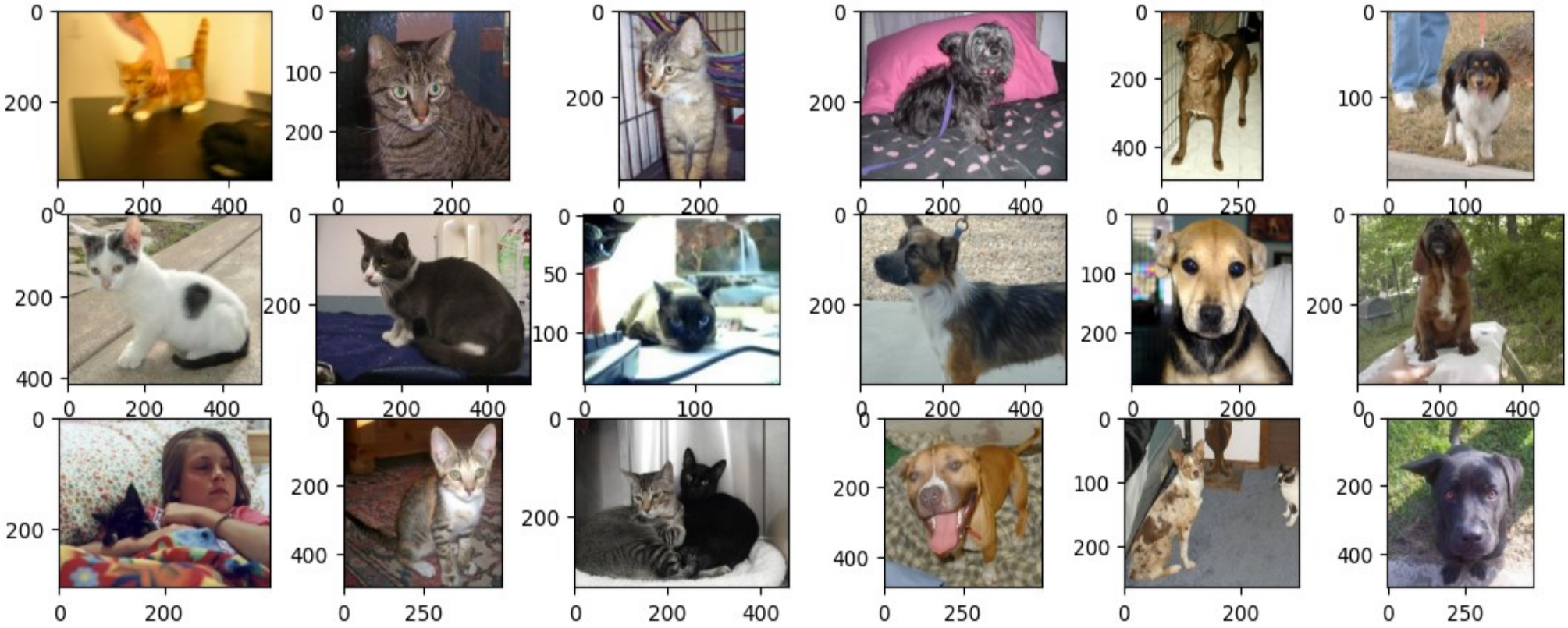
Gli elementi considerati sono stati i seguenti:

- Dataset
- Preprocessing e Data Augmentation
- Metriche
- Funzioni d'attivazione
- Loss
- Batch size
- Regolarizzazione
- Ottimizzatori
- Inizializzazione dei pesi
- Iperparametri

Ho preso spunto da "*How to Classify Photos of Dogs and Cats*" di Jason Brownlee.
Fonte: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/

STUDIO DEL DATASET (1/3)

Vediamo alcuni gatti ed alcuni cani presi dal Dataset



STUDIO DEL DATASET (2/3)

- Il dataset è **bilanciato** ed è composto da 12.500 foto di cani e da 12.500 foto di gatti per un totale di 25.000 foto etichettate;
- Ho abbastanza foto a disposizione per poter utilizzare un **Validation Set**;
- Il nome delle foto è un numero progressivo, aggiungerò il nome della classe per renderle univoche;
- Le foto sono **a colori** e di dimensione diversa, farò un **resize** per uniformarle ad una grandezza di 200x200 pixel;
- Farò **normalizzazione** per portare il valore dei pixel tra 0 e 1;
- **Rimuovo 3 foto corrotte**: «dog.11702.jpg», «cat.10404.jpg» e «cat.666.jpg» (La rimozione di 3 immagini su 25.000 non influenza le scelte progettuali. Continuo a considerare il Dataset bilanciato);
- A volte il soggetto è al centro, a volte è vicino al bordo della foto e a volte ci sono più animali;
- Per la **Data Augmentation** di sicuro non dovrò fare vertical_flip, rotazioni eccessive e non posso traslare molto la foto perché perdo informazioni importanti;

STUDIO DEL DATASET (3/3)

Con lo scopo di testare il comportamento ed i risultati ottenuti dalle CNN vado a creare due copie del Dataset di partenza andando ad effettuare partizionamenti diversi:

Primo Dataset:

- Training Set 64% (16.000 foto)
- Validation Set 16% (4.000 foto)
- Test Set 20% (5.000 foto)

Secondo Dataset:

- Training Set 80% (20.000 foto)
- Validation Set 10% (2.500 foto)
- Test Set 10% (2.500 foto)

PREPROCESSING E DATA AUGMENTATION

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
#    zoom_range=0.3,
    horizontal_flip=True,
    vertical_flip=False,
    brightness_range=[0.8,1.2],
#    rotation_range=20,
#    width_shift_range=0.15,
#    height_shift_range=0.15,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

val_datagen = ImageDataGenerator(rescale=1./255)

batch_size = 64

train_it = train_datagen.flow_from_directory("dataset_diviso/train/",
                                             class_mode='binary',
                                             batch_size=batch_size,
                                             target_size=(200, 200),
                                             color_mode="rgb",
                                             shuffle=True,
                                             seed=42
                                            )
```

[. . .]

Per caricare le foto in batch e per il preprocessing faccio uso di oggetti ImageDataGenerator e di flow_from_directory andando a creare un iteratore per le immagini di Train, Test e Validation.

- Data Augmentation «on-the-fly» solo alle foto di Training (non anche su Validation e Test Set);
- Normalizzazione «on-the-fly»;
- **Seed fissato** per garantire replicabilità dei risultati e delle trasformazioni;

Pro:
- Non mantengo salvate su disco le foto modificate e normalizzate;
- Utile per testare i parametri delle foto (rgb, grayscale e size);
- Utile per testare al meglio i parametri per la Data Augmentation e la loro intensità;

Contro:
- Consumo più risorse durante l'addestramento perché deve generare le nuove immagini a runtime e tenerle in RAM;
- Non posso modificare manualmente le probabilità di eseguire le trasformazioni;

METRICHE

- Classificazione Binaria
- Dataset Bilanciato
- Hard Prediction

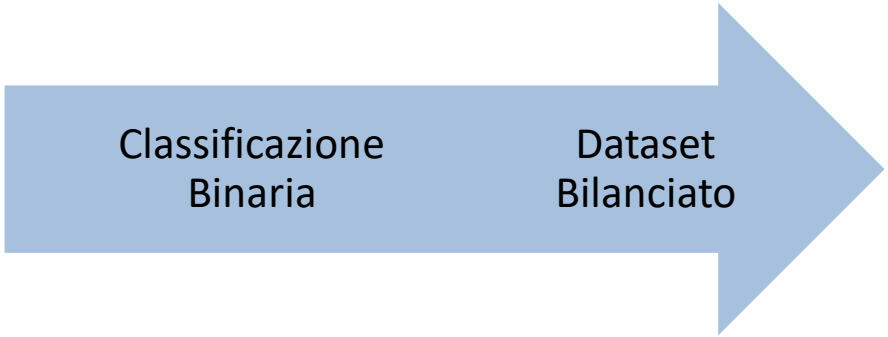
Accuracy + Confusion Matrix

- **Accuracy**
  - The number of correct prediction made by the model.

Predicted \ Actual	Positives	Negatives
Positives	**TP**	**FP**
Negatives	**FN**	**TN**

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Activation e Loss Functions



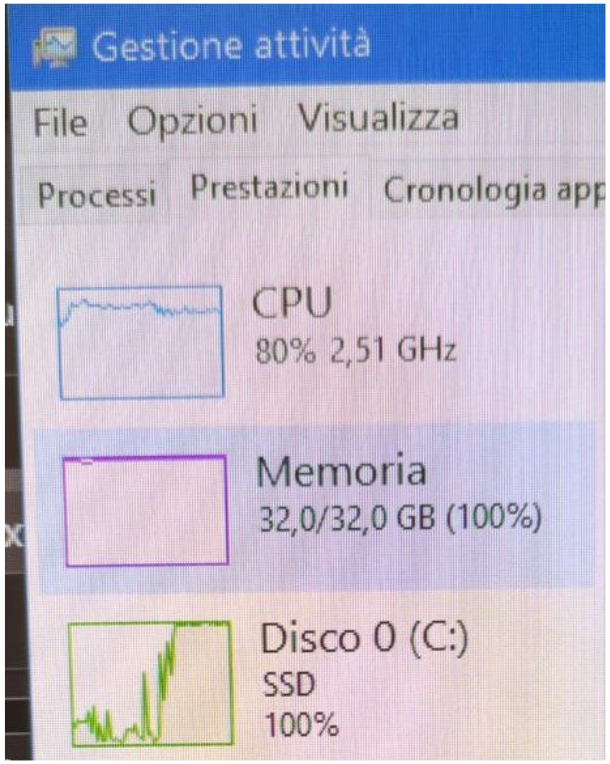
Loss Function = Binary Cross Entropy

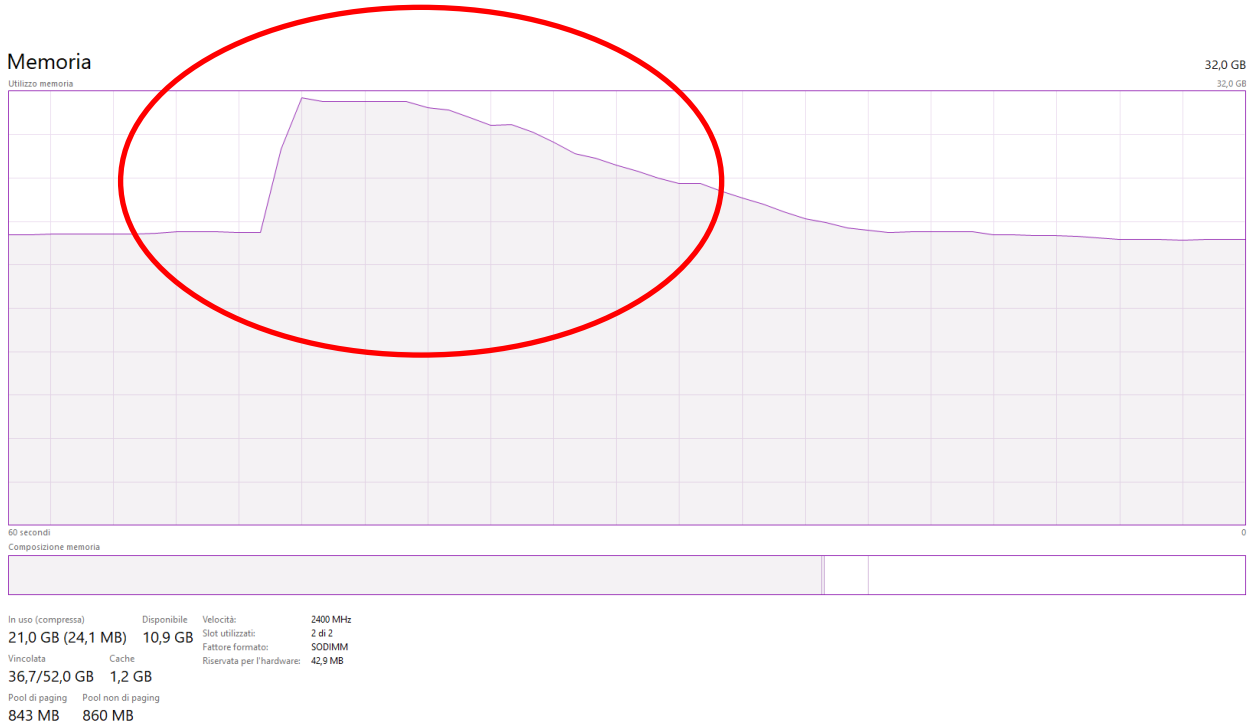
Hidden Layers = ReLU

Output Layer = Sigmoide

BATCH SIZE

Per la scelta del **batch size**, iperparametro che influisce sull'addestramento, ho dovuto tenere in considerazione anche i limiti del mio hardware:





Con una batch size troppo grande si causa un blocco del sistema

Con una batch size di dimensioni appropriata non ci sono problemi di memoria.
Durante il progetto ho testato diverse grandezze comprese tra 16 e 256 foto per batch.

REGOLARIZZAZIONE

La regolarizzazione serve per migliorare la stabilità della rete e per migliorare l'apprendimento evitando di andare in overfitting.

Le tecniche di regolarizzazione utilizzate che hanno portato ai risultati migliori sono state le seguenti:

- **Dropout** (*parameter sharing*);
- **Early Stopping** (equivalente a L^2);
- **Data Augmentation** (*noise injection*);

Le prime due sono tecniche che richiedono un Validation Set, che ho a disposizione.

Bisogna bilanciare in maniera corretta la regolarizzazione perché toglie capacità alla rete ed è necessario fare attenzione alla compatibilità dei vari tipi.

Ho anche provato ad utilizzare la Batch Normalization ma nel mio caso il Dropout funziona meglio.

OTTIMIZZATORI

Gli ottimizzatori testati sono i seguenti:

- **SGD with momentum;**
- **Adam;**
- **RMSprop;**

Dipende molto dalla rete in analisi ma generalmente i primi due mi hanno portato a risultati migliori

INIZIALIZZAZIONE DEI PESI

L'inizializzazione dei pesi gioca un ruolo importante nelle NN e può influenzare di molto il risultato ottenuto.

Ci sono svariati modi per inizializzare i pesi ma ho scelto i seguenti in base alla funzione d'attivazione utilizzata:

- **he_uniform** per gli strati con la ReLU;
- **glorot_uniform** (default) per lo strato di output con la sigmoide;

He_uniform è stato creato e testato appositamente per gli strati che utilizzano la **ReLU** e PReLU.
Glorot_uniform è stato creato e testato appositamente per gli strati che hanno funzioni d'attivazione come la **sigmoide**, tanh e softsign.

Fonti:

Kaiming He et al. «*Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*» https://arxiv.org/pdf/1502.01852v1.pdf
Xavier Glorot et al. «*Understanding the difficulty of training deep feedforward neural networks*» https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

IPERPARAMETRI

Gli iperparametri sono stati selezionati manualmente per andare a cercare le combinazioni migliori procedendo per tentativi.

Successivamente illustrerò come effettuare Random Search utilizzando KerasTuner.

ARCHITETTURA «BASE» (1/3)


```python
def define_model(optimizer):
    model=Sequential()

    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    model.add(Flatten())
    model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dropout(0.5))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

- 3 blocchi: convoluzione, MaxPooling e poi Dropout;
- Kernel 3x3 e in numero crescente (stride=1 di default);
- Padding «same» per fare in modo che anche i pixel vicino ai bordi vengano utilizzati come quelli più interni e per non ridurre la grandezza dell'output;
- Max Pooling 2x2 per «rompere» l'invarianza e per fare downsampling senza perdere tutte le informazioni.

ARCHITETTURA «BASE» (2/3)

$$\#Parametri\_conv = (larghezza\_kernel * altezza\_kernel) * \#canali\_input * \#kernel + \#kernel\ (bias)$$

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 200, 200, 32)      896

 max_pooling2d (MaxPooling2  (None, 100, 100, 32)      0
 D)

 dropout (Dropout)           (None, 100, 100, 32)      0

 conv2d_1 (Conv2D)           (None, 100, 100, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 50, 50, 64)        0
 g2D)

 dropout_1 (Dropout)         (None, 50, 50, 64)        0

 conv2d_2 (Conv2D)           (None, 50, 50, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 25, 25, 128)       0
 g2D)

 dropout_2 (Dropout)         (None, 25, 25, 128)       0

 flatten (Flatten)           (None, 80000)             0

 dense (Dense)               (None, 256)               20480256

 dropout_3 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 20573761 (78.48 MB)
Trainable params: 20573761 (78.48 MB)
Non-trainable params: 0 (0.00 Byte)
```

padding='same' e **input=rgb**

$(3 * 3) * 3 * 32 + 32 = 896$

Il MaxPooling dimezza la dimensioni e non ha pesi.
Il Dropout non modifica le dimensioni e non ha pesi.

$(3 * 3) * 32 * 64 + 64 = 18.496$

$(3 * 3) * 64 * 128 + 128 = 73.856$

$25 * 25 * 128 = 80.000$
$80000 * 256 + 256 = 20.480.256$

$256 * 1 + 1 = 257$

$Totale = 896 + 18496 + 73856 + 20480256 + 257 = 20.573.761$

ARCHITETTURA «BASE» (3/3)

$$\#Parametri\_conv = (larghezza\_kernel * altezza\_kernel) * \#canali\_input * \#kernel + \#kernel\ (bias)$$

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 200, 200, 32)      320

 max_pooling2d (MaxPooling2  (None, 100, 100, 32)      0
 D)

 dropout (Dropout)           (None, 100, 100, 32)      0

 conv2d_1 (Conv2D)           (None, 100, 100, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 50, 50, 64)        0
 g2D)

 dropout_1 (Dropout)         (None, 50, 50, 64)        0

 conv2d_2 (Conv2D)           (None, 50, 50, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 25, 25, 128)       0
 g2D)

 dropout_2 (Dropout)         (None, 25, 25, 128)       0

 flatten (Flatten)           (None, 80000)             0

 dense (Dense)               (None, 256)               20480256

 dropout_3 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 20573185 (78.48 MB)
Trainable params: 20573185 (78.48 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

padding='same' e **input=grayscale**

$$(3 * 3) * \cancel{3}^{1} * 32 + 32 = 320$$

$$(3 * 3) * 32 * 64 + 64 = 18.496$$

$$(3 * 3) * 64 * 128 + 128 = 73.856$$

$$25 * 25 * 128 = 80.000$$
$$80000 * 256 + 256 = 20.480.256$$

$$256 * 1 + 1 = 257$$

$$Totale = 20.573.185$$

Rispetto al caso **'rgb'** ho solo 576 pesi in meno ma il risparmio vero è computazionale: con **'grayscale'** faccio 1/3 dei calcoli nel primo strato.

TRAINING E TEST


```
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

```
model=define_model(SGD(learning_rate=0.001, momentum=0.9))
history = model.fit(train_it, steps_per_epoch=len(train_it), validation_data=val_it, validation_steps=len(val_it),
                    epochs=300, verbose=1, callbacks=[tensorboard_callback,early_stop],validation_split=0)
```

```
_, acc = model.evaluate(test_it, steps=len(test_it), verbose=1)
print('> %.3f' % (acc * 100.0))
```

Esempio di come viene effettuato il Training e il Test del modello.

L'uso di Early Stopping permette di addestrare per un numero molto elevato di epoche perché arresta automaticamente il training quando la Validation Loss inizia a peggiorare in maniera continuativa e restituisce i pesi migliori.

SALVATAGGIO MODELLO E GRAFICI


```
model_dir = "modelli/"+ unique_name +".keras"
model.save(model_dir)
```

```
def summarize_diagnostics(history):
    pyplot.figure(figsize=(20, 15))

    # plot loss
    pyplot.subplot(211)
    pyplot.title('Cross Entropy Loss')
    pyplot.plot(history.history['loss'], color='orange', label='train')
    pyplot.plot(history.history['val_loss'], color='blue', label='val')
    pyplot.legend(loc='upper right') # legenda
    pyplot.grid(which='both', linestyle='-', linewidth=1, color='black')  # griglia nera

    # plot accuracy
    pyplot.subplot(212)
    pyplot.title('Classification Accuracy')
    pyplot.plot(history.history['accuracy'], color='red', label='train')
    pyplot.plot(history.history['val_accuracy'], color='green', label='val')
    pyplot.legend(loc='lower right')
    pyplot.grid(which='both', linestyle='-', linewidth=1, color='black')

    # salvo i plot su file
    filename = 'plot/' + unique_name
    pyplot.savefig(filename + '_plot.png')
    pyplot.close()

summarize_diagnostics(history)
```

Salvo il modello su disco, estraggo le informazioni da *history* e le uso per creare i grafici

Università degli Studi di Ferrara

RISULTATI MODELLO «BASE»:

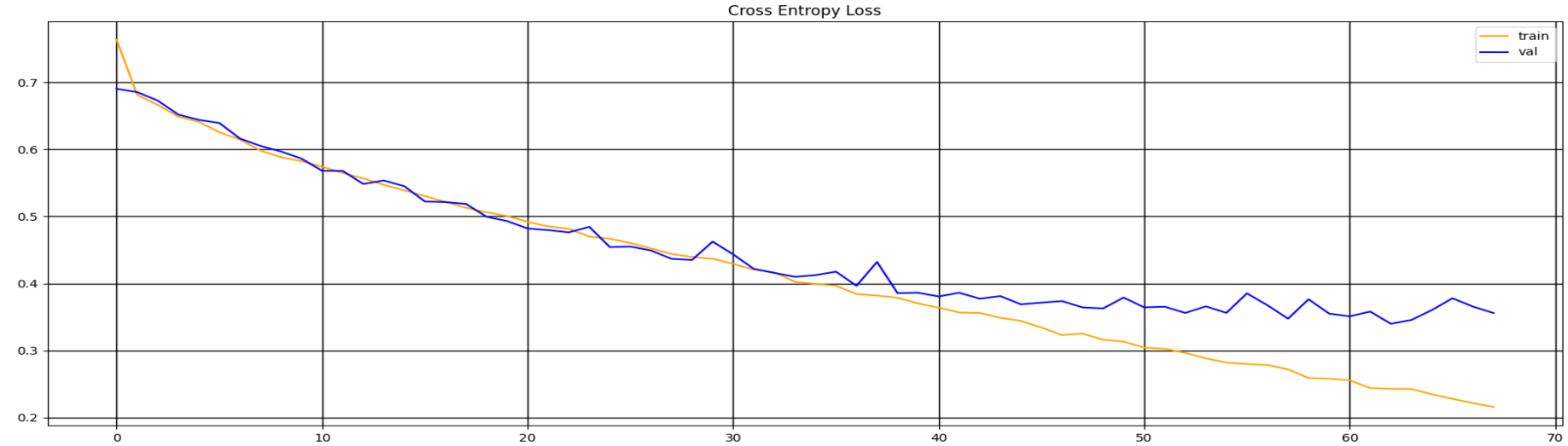
Department of Engineering Ferrara

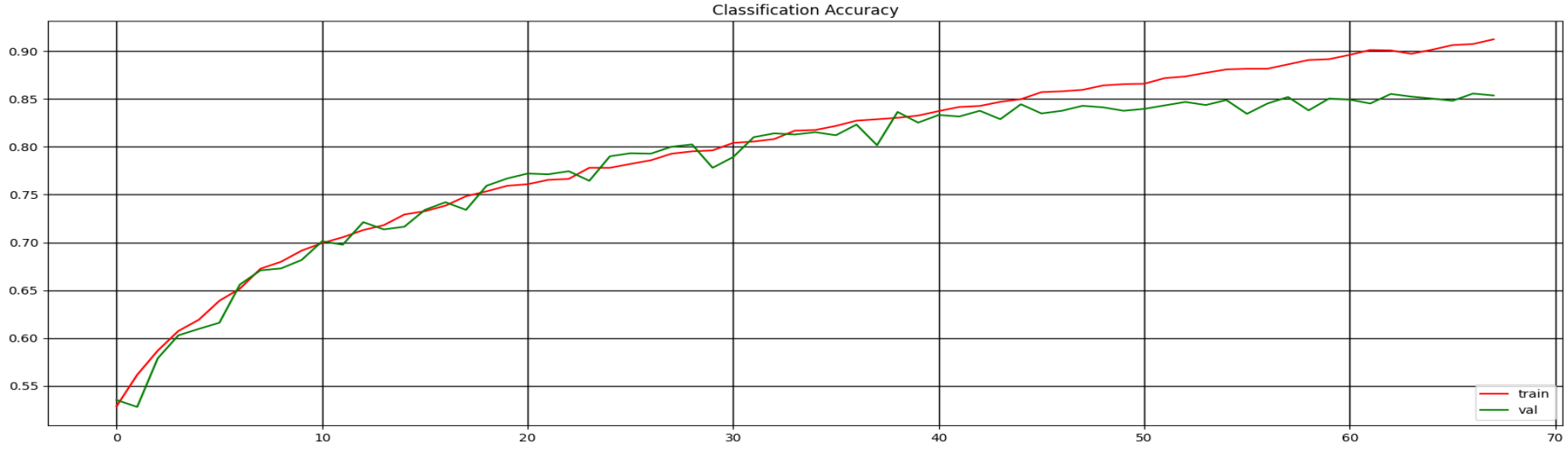
Dataset	input	Data Augmentation	Accuracy	Loss	#epoche (migliore)	(Circa) secondi epoca
64-16-20	grayscale	No	77,36%	0,4779	48	44
64-16-20	grayscale	Sì	80,88%	0,4255	67	93
64-16-20	RGB	No	79,88%	0,4426	41	46
64-16-20	RGB	Sì	83,22%	0,3969	70	160
80-10-10	grayscale	No	79,12%	0,4456	41	60
80-10-10	grayscale	Sì	82,44%	0,3963	66	115
80-10-10	RGB	No	80,44%	0,4154	37	54
80-10-10	RGB	Sì	84,64%	0,3668	63	215

ALCUNE CONSIDERAZIONI

- Ho utilizzato il PC mentre venivano eseguiti gli addestramenti, quindi i tempi di calcolo potrebbero essere stati influenzati;
- Più scende la Loss e più aumenta l'Accuracy;
- La Data Augmentation e le foto RGB hanno un notevole impatto sull'Accuracy;
- L'utilizzo del secondo dataset (80-10-10) ha fornito sempre un incremento delle prestazioni;
- Più dati si hanno a disposizione per il Training e più aumenta l'Accuracy;
- L'uso di Data Augmentation ha introdotto rumore rallentando l'entrata in overfitting;
- L'utilizzo di Data Augmentation a runtime ha un notevole impatto sulle tempistiche;

GRAFICI 801010 RGB DATA AGUMENTATION





(63 + 5) epoche * 215 secondi/epoca = circa 243 minuti = circa 4 ore

KERASTUNER (1/4)

KerasTuner è uno strumento che risolve il problema della ricerca degli iperparametri migliori e mette a disposizione diversi algoritmi.
Io ho utilizzato la Random Search per alcuni degli iperparametri del modello come: il numero di Kernel nella convoluzione, il valore del Dropout e l'ottimizzatore.
È possibile direzionare la ricerca limitandola a dei range specifici e si può fornire anche il valore per l'incremento.

Ho fatto una prova utilizzando il dataset 64-16-20 a colori e ho limitato il numero di tentativi perché è un procedimento estremamente lungo a causa dei numerosi addestramenti. Per questo ho utilizzato anche l'Early Stopping per terminare il training dei modelli che, a causa dell'inizializzazione randomica dei pesi, si trovano in una brutta posizione e non riescono a migliorare.
Inoltre viene salvata la cronologia del processo di ricerca rendendo possibile vedere i parametri testati e il risultato ottenuto.

Fonti:
https://keras.io/keras_tuner/
https://keras.io/guides/keras_tuner/getting_started/

KERASTUNER (2/4)

Definizione del modello


```python
def build_model(hp):
    model = Sequential()
    model.add(Conv2D(filters=hp.Int('conv_1_filter', min_value=32, max_value=64, step=16),
                     kernel_size=hp.Choice('conv_1_kernel', values=[3]), #3,5
                     activation='relu',
                     padding='same',
                     input_shape=(200, 200, 3)))
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(rate=hp.Float('dropout_1', min_value=0.1, max_value=0.4, step=0.1)))
    model.add(Conv2D(filters=hp.Int('conv_2_filter', min_value=64, max_value=128, step=32),
                     kernel_size=hp.Choice('conv_2_kernel', values=[3]),
                     activation='relu',
                     padding='same'))
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(rate=hp.Float('dropout_2', min_value=0.1, max_value=0.3, step=0.1)))
    model.add(Conv2D(filters=hp.Int('conv_3_filter', min_value=128, max_value=256, step=64),
                     kernel_size=hp.Choice('conv_3_kernel', values=[3]),
                     activation='relu',
                     padding='same'))
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(rate=hp.Float('dropout_3', min_value=0.1, max_value=0.3, step=0.1)))
    model.add(Flatten())
    model.add(Dense(units=hp.Int('dense_1_units', min_value=128, max_value=256, step=32),
                    activation='relu'))
    model.add(Dropout(rate=hp.Float('dropout_4', min_value=0.2, max_value=0.4, step=0.1)))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=hp.Choice('optimizer', values=['adam', 'sgd']),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model
```

KERASTUNER (3/4)

Avvio la ricerca e prendo il miglior modello

```
tuner = RandomSearch(build_model,
                     objective='val_accuracy',
                     max_trials=10,
                     executions_per_trial=3,
                     directory='3blockKerasTuner',
                     project_name='canigatti')

# Effettuo la ricerca degli iperparametri
tuner.search(train_it,
             epochs=50,
             validation_data=val_it,
             callbacks=[early_stop, tensorboard_callback])

# 10 combinazioni, ognuna addestrata 3 volte
# con i pesi inizializzati a valori casuali
# = 30 addestramenti.
# Ogni addestramento dura 50 epoche ma con early stopping.

# Prendo il miglior modello
best_model = tuner.get_best_models()[0]
```

KERASTUNER (4/4)

Risultato e Test

```
Trial 10 Complete [00h 22m 35s]
val_accuracy: 0.7056028048197428

Best val_accuracy So Far: 0.8161580761273702
Total elapsed time: 11h 50m 30s
INFO:tensorflow:Oracle triggered exit
```

Early Stopping

```
_, acc = best_model.evaluate(test_it, steps=len(test_it), verbose=1)
print('> %.3f' % (acc * 100.0))
```

```
79/79 [==============================] - 14s 167ms/step - loss: 0.4207 - accuracy: 0.8078
> 80.780
```

Gli iperparametri migliori sono stati salvati nei log e sono stati riportati a destra

- root
  - trial_id "06"
  - hyperparameters
    - space [] 12 items
    - values
      - conv_1_filter 64
      - conv_1_kernel 3
      - dropout_1 0.1
      - conv_2_filter 64
      - conv_2_kernel 3
      - dropout_2 0.1
      - conv_3_filter 256
      - conv_3_kernel 3
      - dropout_3 0.1
      - dense_1_units 160
      - dropout_4 0.2
      - optimizer "adam"
  - metrics
  - score 0.8161580761273702
  - best_step 5
  - status "COMPLETED"
  - message null

MODIFICA #1 AL MODELLO «BASE»

```python
def define_model(optimizer):

    model=Sequential()

    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    model.add(Flatten())
    model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dropout(0.4))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Nella prima modifica ho raddoppiato il numero di kernel in ogni strato convoluzionale

MODIFICA #2 AL MODELLO «BASE»

```
def define_model(optimizer):

    model=Sequential()

    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(200, 200, 3)))
    model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))

    model.add(Flatten())
    model.add(Dense(256, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dropout(0.4))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Nella seconda modifica ho aggiunto uno strato convoluzionale per ogni blocco

CONFRONTO MODIFICHE (1/2)

#1

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 200, 200, 64)	1792
max_pooling2d (MaxPooling2D)	(None, 100, 100, 64)	0
dropout (Dropout)	(None, 100, 100, 64)	0
conv2d_1 (Conv2D)	(None, 100, 100, 128)	73856
max_pooling2d_1 (MaxPooling2D)	(None, 50, 50, 128)	0
dropout_1 (Dropout)	(None, 50, 50, 128)	0
conv2d_2 (Conv2D)	(None, 50, 50, 256)	295168
max_pooling2d_2 (MaxPooling2D)	(None, 25, 25, 256)	0
dropout_2 (Dropout)	(None, 25, 25, 256)	0
flatten (Flatten)	(None, 160000)	0
dense (Dense)	(None, 256)	40960256
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 1)	257

Total params: 41331329 (157.67 MB)
Trainable params: 41331329 (157.67 MB)
Non-trainable params: 0 (0.00 Byte)

#2

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 200, 200, 32)	896
conv2d_1 (Conv2D)	(None, 200, 200, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 100, 100, 32)	0
dropout (Dropout)	(None, 100, 100, 32)	0
conv2d_2 (Conv2D)	(None, 100, 100, 64)	18496
conv2d_3 (Conv2D)	(None, 100, 100, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 50, 50, 64)	0
dropout_1 (Dropout)	(None, 50, 50, 64)	0
conv2d_4 (Conv2D)	(None, 50, 50, 128)	73856
conv2d_5 (Conv2D)	(None, 50, 50, 128)	147584
max_pooling2d_2 (MaxPooling2D)	(None, 25, 25, 128)	0
dropout_2 (Dropout)	(None, 25, 25, 128)	0
flatten (Flatten)	(None, 80000)	0
dense (Dense)	(None, 256)	20480256
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 1)	257

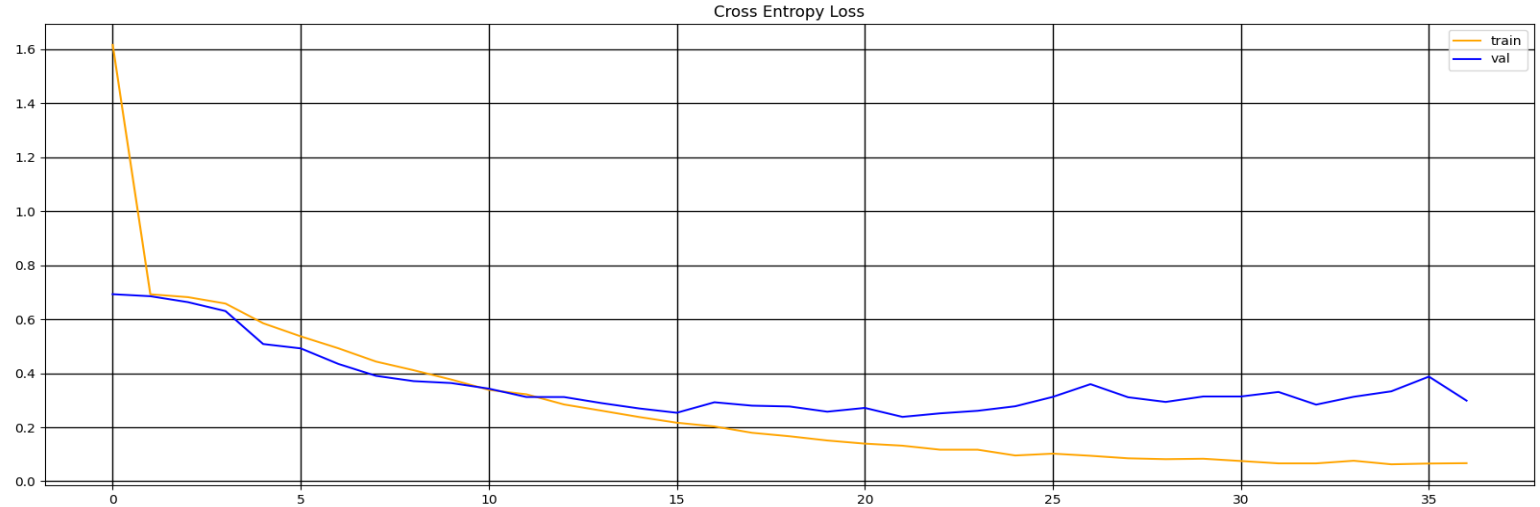
Total params: 20767521 (79.22 MB)
Trainable params: 20767521 (79.22 MB)
Non-trainable params: 0 (0.00 Byte)

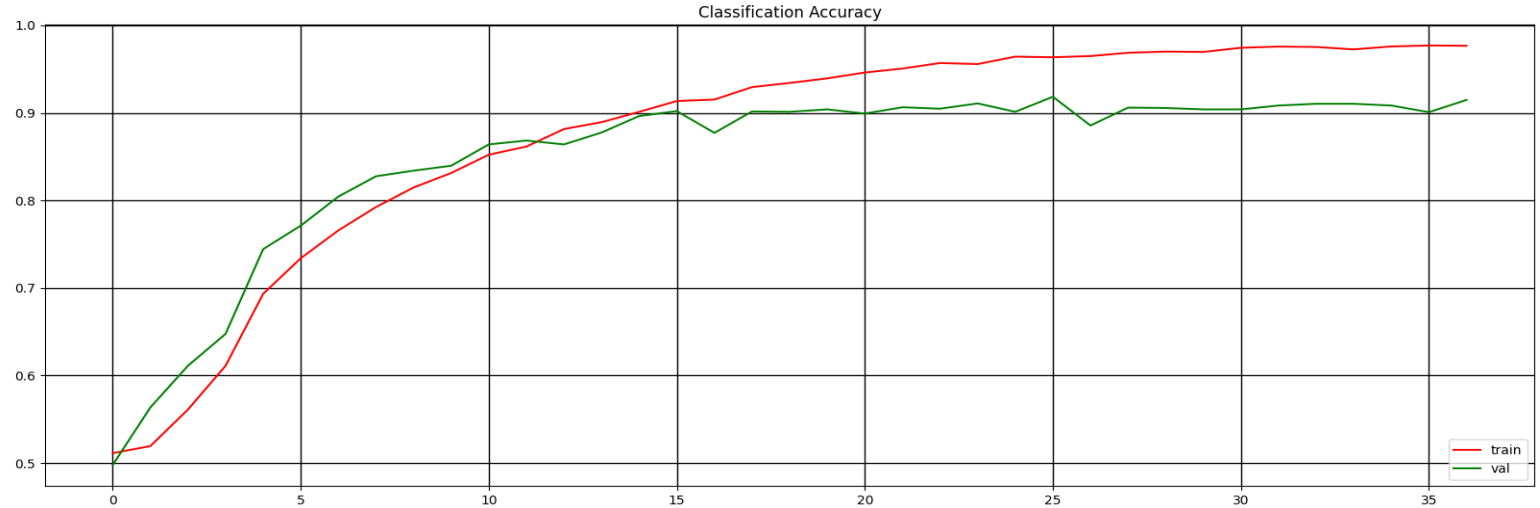
CONFRONTO MODIFICHE (2/2)

In tutti i test qui riportati ho usato Dataset 80-10-10, foto a colori e stessa Data Agumentation

Modifica	Accuracy	Loss	#epoche (migliore)	(Circa) secondi epoca	Pesi (MB)
#1	86,12%	0,3451	51	240-300	157,67
#2	90,40%	0,2490	22	230-250	79,22
Nessuna	84,64%	0,3668	63	215	78,48

GRAFICI MODIFICA #2





Early Stopping con pazienza = 15

22 epoche * 250 secondi/epoca (arrotondo al caso peggiore) = circa 92 minuti = circa 1 ora e 32 minuti

Molto meglio rispetto a prima!

TRANSFER LEARNING E FINE TUNING

Per il Transfer Learning e il Fine Tuning ho utilizzato Keras per prendere due modelli già preaddestrati su un dataset compatibile come ImageNet. Ho scelto VGG16 e MobileNetV2 perché entrambe hanno le stessa accuracy ma hanno forti differenze:

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8

Fonti utilizzate per questa parte:

https://keras.io/api/applications/
https://keras.io/guides/transfer_learning/
https://keras.io/api/applications/vgg/#vgg16-function
https://keras.io/api/applications/mobilenet/#mobilenetv2-function
https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/
"Very Deep Convolutional Networks for Large-Scale Image Recognition" https://arxiv.org/pdf/1409.1556.pdf
"MobileNetV2: Inverted Residuals and Linear Bottlenecks" https://arxiv.org/pdf/1801.04381.pdf

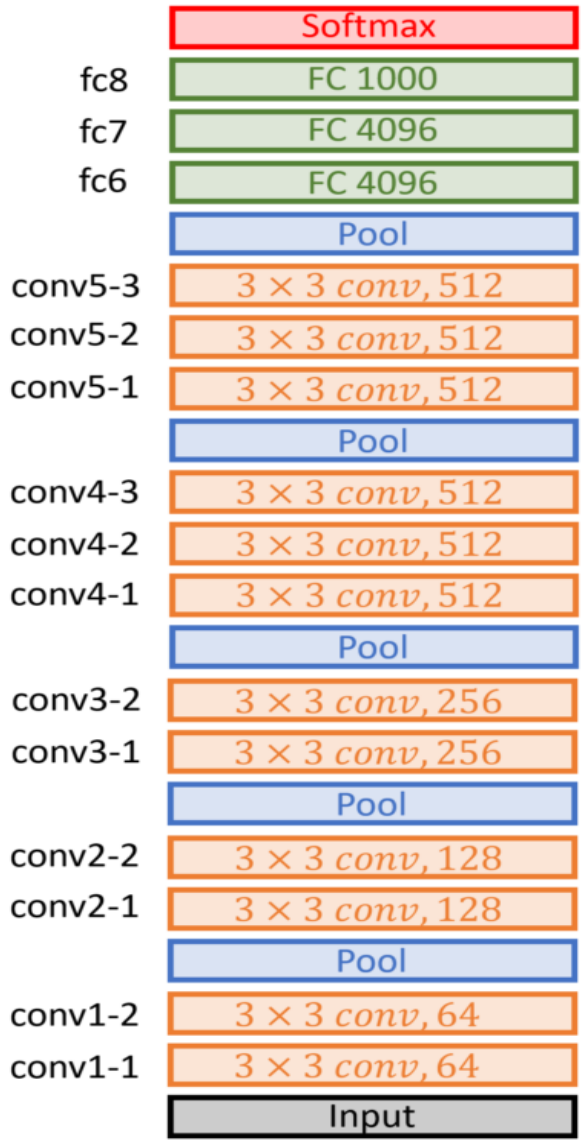
PROCEDIMENTO GENERALE:

- Carico il modello già fatto insieme ai suoi pesi senza però includere la parte finale;
- «Congelo» tutti gli strati del modello rendendoli non addestrabili;
- Aggiungo la parte finale densa e l'output;
- Addestro il modello con il mio dataset;
- «Scongelo» tutti gli strati, oppure solo una parte, e addestro ancora per qualche epoca utilizzando un learning rate molto più basso;

VGG16

Caratteristiche principali:

- 16 strati
- Molti filtri piccoli 3x3: campo ricettivo più grande ma meno pesi
- ReLU
- Max Pooling
- Modello molto pesante
- Utilizzabile per Transfer Learning



VGG16

Fonte immagine: https://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/

VGG16

Transfer Learning

```
from keras.applications.vgg16 import VGG16
from keras.models import Model
def define_model():
    # carico il modello
    model = VGG16(include_top=False, input_shape=(224, 224, 3))
    # congelo gli strati
    for layer in model.layers:
        layer.trainable = False
    # aggiungo parte finale
    flat1 = Flatten()(model.layers[-1].output)
    class1 = Dense(256, activation='relu', kernel_initializer='he_uniform')(flat1)
    class2 = Dense(256, activation='relu', kernel_initializer='he_uniform') (class1)
    output = Dense(1, activation='sigmoid')(class2)
    model = Model(inputs=model.inputs, outputs=output)
    opt = SGD(learning_rate=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

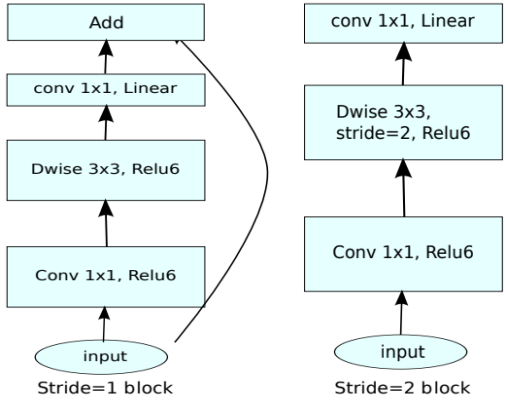
Fine Tuning

```
for layer in model.layers:
    layer.trainable = True

# learning rate più basso
opt=SGD(learning_rate=0.00005, momentum=0.9)

model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
```

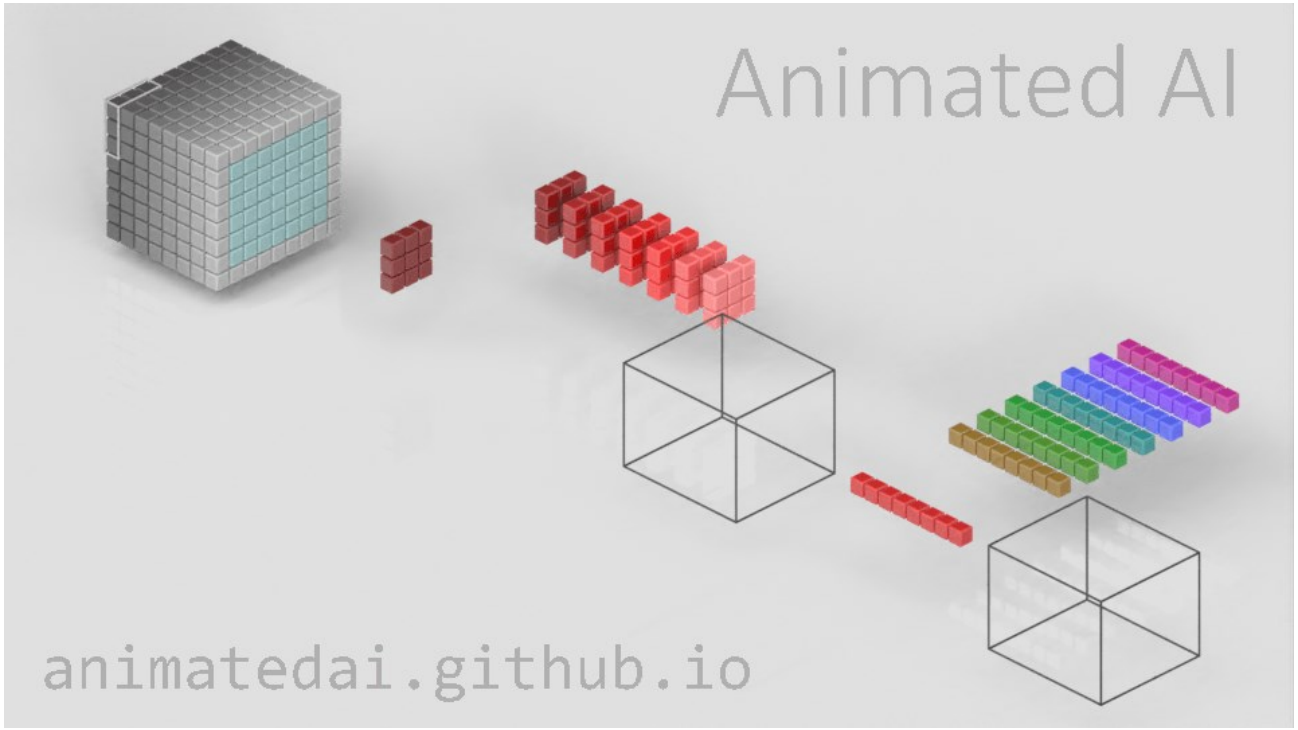
MobileNetV2

Caratteristiche principali:

- 105 strati;
- Molti filtri piccoli 3x3;
- ReLU6: ReLU che satura a 6;
- Modello molto leggero;
- Utilizzabile per Transfer Learning;



(d) Mobilenet V2



Fonte: https://animatedai.github.io/

Nuovo layer: «*inverted residual with linear bottleneck*»

1. **Espansione:** input sottoposto a convoluzione 1x1 che espande il numero dei canali + ReLU6;
2. «***Depthwise convolution***»**:** i canali espansi vengono sottoposti a convoluzione 3x3 che opera su ogni canale in modo separato + ReLU6;
3. «***Linear Bottleneck***»**:** dopo la *Depthwise Convolution* viene effettuata una convoluzione 1x1 per ridurre il numero di canali e poi non si utilizza una funzione d'attivazione non lineare;
4. «***Shortcut***»**:** se le dimensioni dell'input e dell'output sono identiche allora l'input originale viene sommato all'output del bottleneck.

MobileNetV2

Transfer Learning

```python
from tensorflow.keras.applications import MobileNetV2
from keras.models import Model


def define_model():
    # carico il modello
    model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    # congelo gli strati
    for layer in model.layers:
        layer.trainable = False
    # aggiungo parte finale
    flat1 = GlobalAveragePooling2D()(model.layers[-1].output)
    class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)
    class2 = Dense(64, activation='relu', kernel_initializer='he_uniform') (class1)
    output = Dense(1, activation='sigmoid')(class2)

    model = Model(inputs=model.inputs, outputs=output)

    #opt = SGD(learning_rate=0.0001, momentum=0.9)
    opt = Adam(learning_rate=0.0001)
    #opt = RMSprop(learning_rate=0.0001)

    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

MobileNetV2

Fine tuning

```
# voglio addestrare solo dal blocco X in poi
blockX_start_index = None
for index, layer in enumerate(model.layers):
    if layer.name == 'block_16_expand':
        blockX_start_index = index
        break

for layer in model.layers[:blockX_start_index]:
    layer.trainable = False

# lascio congelati i layer con la Batch Normalization
for layer in model.layers[blockX_start_index:]:
    if trova_sottostringa(layer.name)==False:
        #print(layer.name)
        layer.trainable = True
    else:
        #print(layer.name)
        layer.trainable = False
```

```
opt=Adam(learning_rate=0.00001)

model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
```

CONFRONTO VGG16 E MOBILENETV2

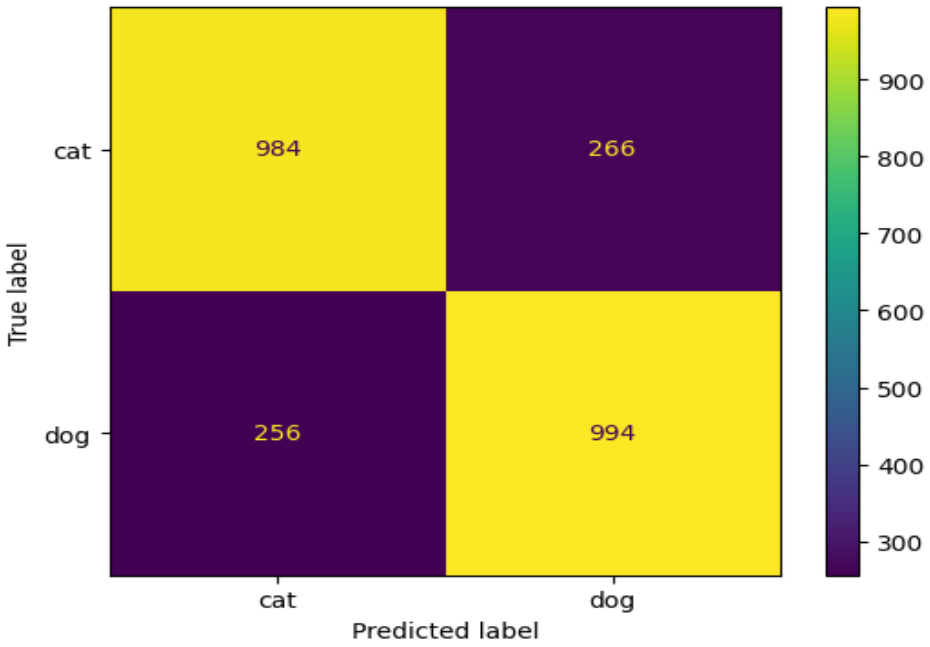
In tutti i test qui riportati ho usato Dataset 80-10-10, foto a colori e stessa Data Agumentation.

Riporto i dati rispettivamente di Transfer Learning e Fine Tuning

Rete	Accuracy	#epoche	(Circa) secondi epoca	Pesi (MB)
VGG16	98,28% e 98,80%	3 e 3	267 e 300	80,89
MobileNetV2	98,96% e 99,04%	4 e 9	267 e 254	9,27

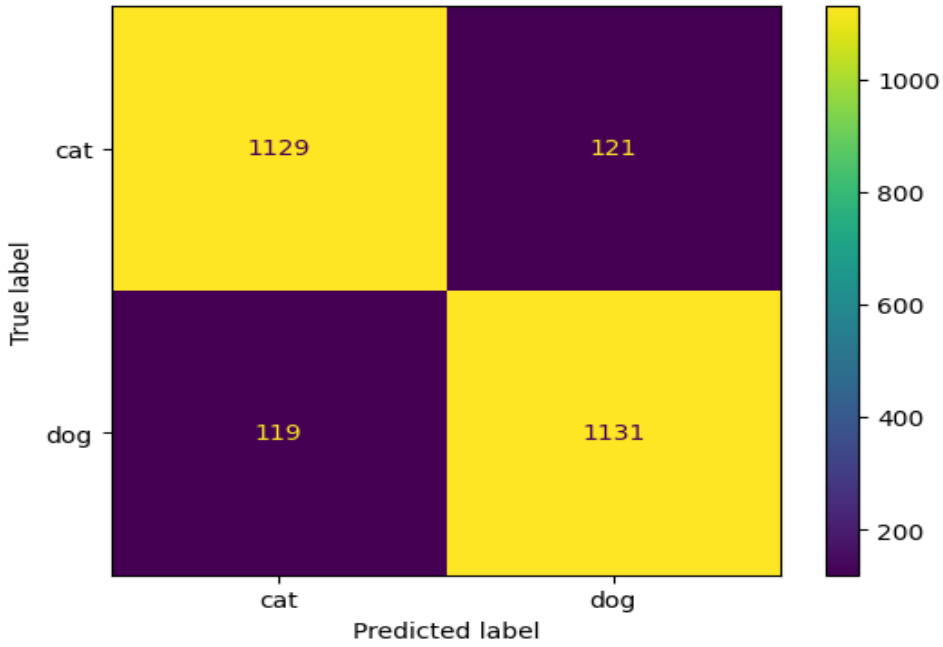
CONFUSION MATRIX (1/2)

801010-grayscale-noDA-3blocks



$$Accuracy = \frac{984 + 994}{2500} = 0{,}7912 = 79{,}12\%$$

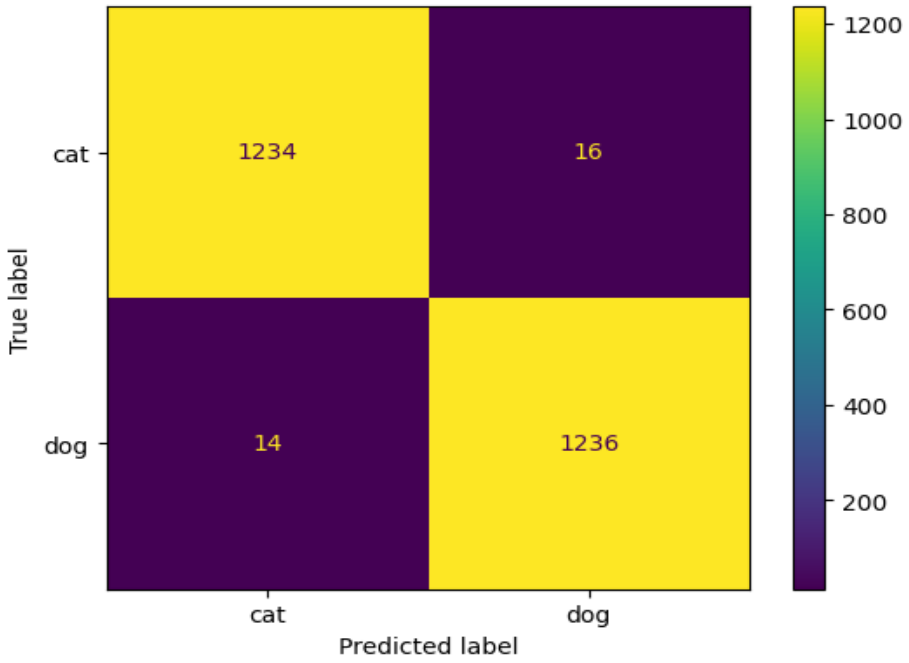
«base» con modifica #2



$$Accuracy = \frac{1129 + 1131}{2500} = 0{,}9040 = 90{,}40\%$$

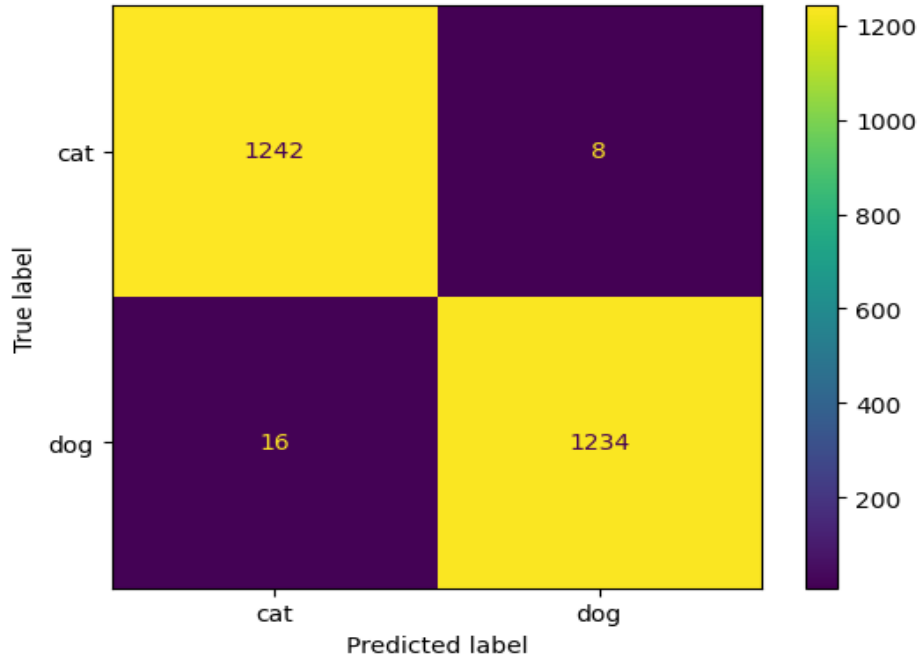
CONFUSION MATRIX (2/2)

VGG16



$$Accuracy = \frac{1234 + 1236}{2500} = 0{,}9880 = 98{,}80\%$$

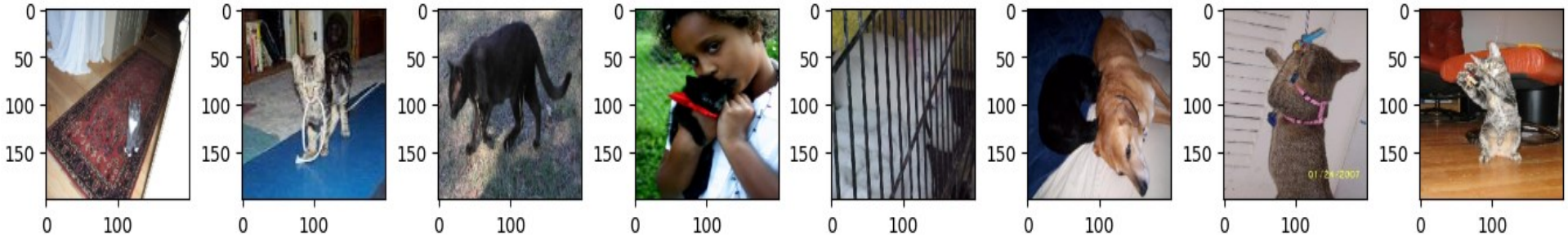
MobileNetV2



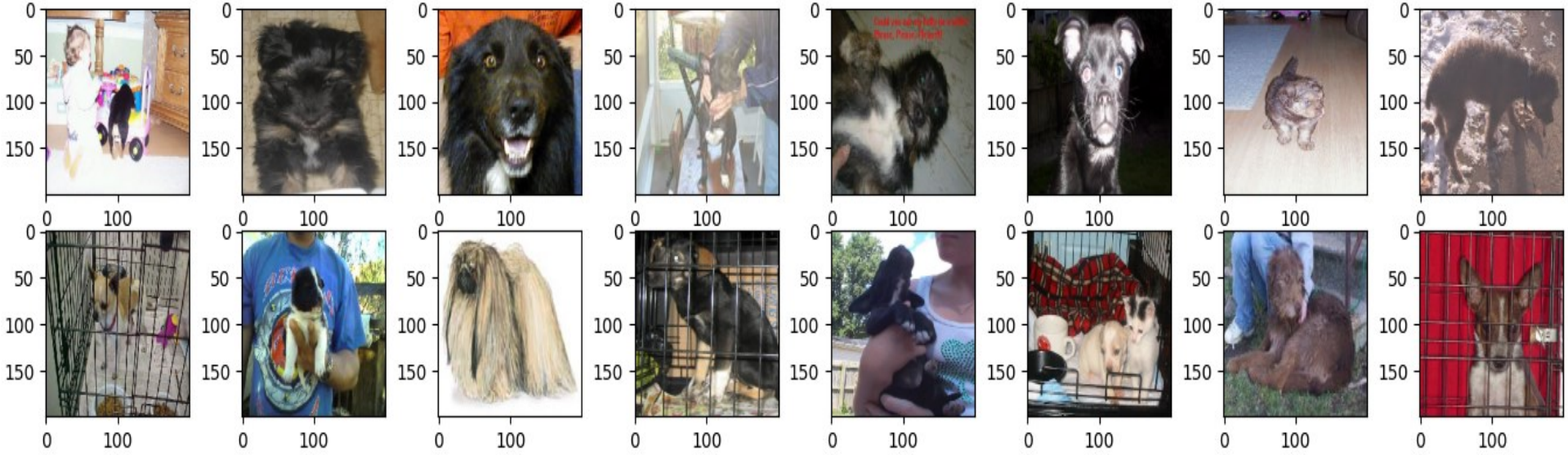
$$Accuracy = \frac{1242 + 1234}{2500} = 0{,}9904 = 99{,}04\%$$

TUTTE LE FOTO CLASSIFICATE IN MANIERA ERRATA
MOBILENETV2

gatti classificati come cani



cani classificati come gatti



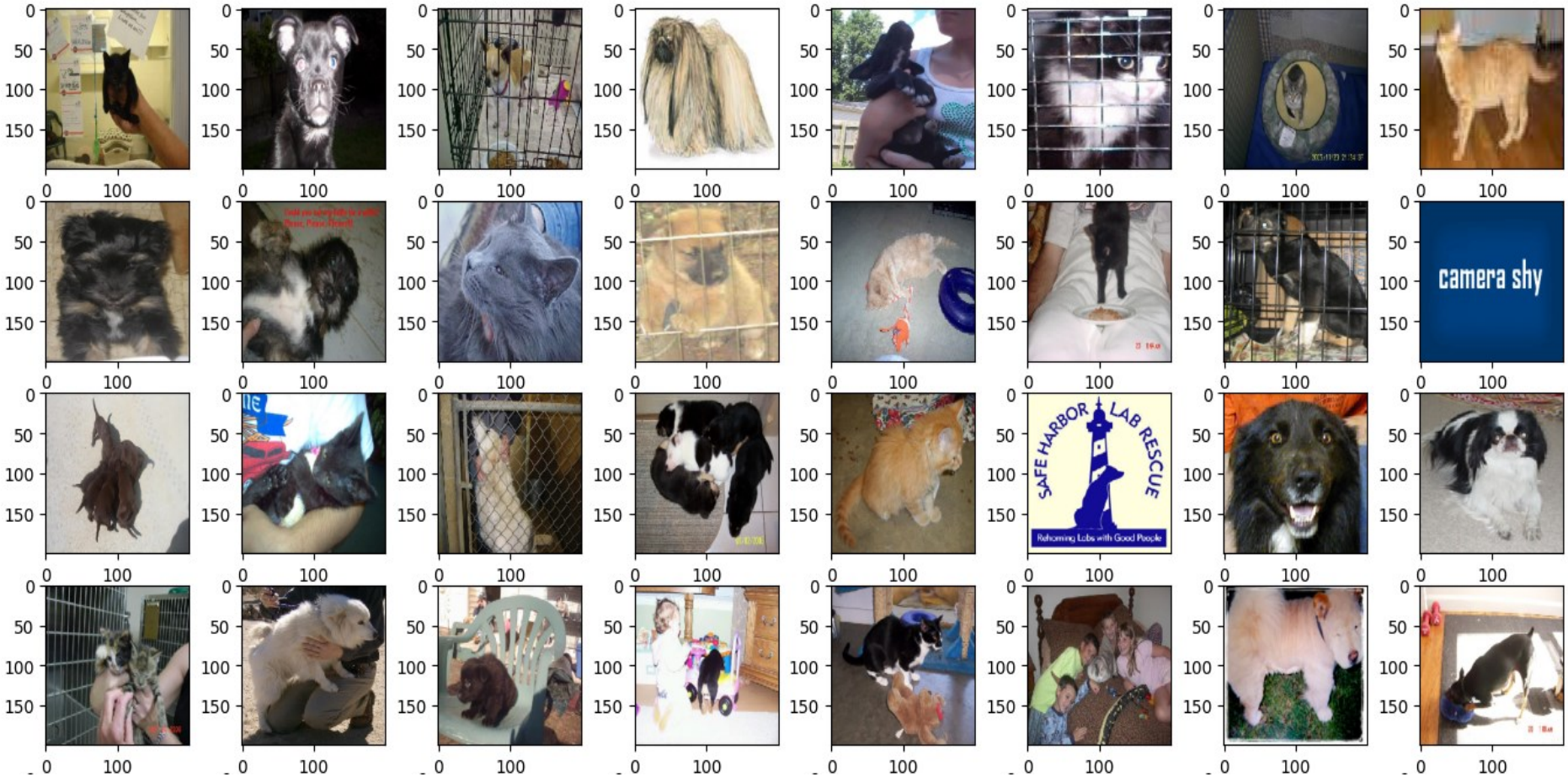
Ho preso ispirazione da: https://github.com/girishkuniyal/Cat-Dog-CNN-Classifier

ALCUNE FOTO CHE METTONO IN DIFFICOLTÀ **MOBILENETV2**


```
df_insicuri = pd.DataFrame({
    'filename':test_it.filenames,
    'predict':ytesthat[:,0],
    'y':test_it.classes
})
df_insicuri['y_pred'] = df_insicuri['predict']>0.5
df_insicuri.y_pred = df_insicuri.y_pred.astype(int)

df_insicuri['sicurezza'] =abs(0.5- df_insicuri['predict'])
df_insicuri= df_insicuri.sort_values(by='sicurezza', ascending=True)
df_insicuri.head(10)
```

	filename	predict	y	y_pred	sicurezza
245	cat/cat.1277.jpg	0.48996	0	0	0.01004
1563	dog/dog.2458.jpg	0.48725	1	0	0.01275
1991	dog/dog.5785.jpg	0.48481	1	0	0.01519
2177	dog/dog.7458.jpg	0.46886	1	0	0.03114
2338	dog/dog.8421.jpg	0.45370	1	0	0.04630
720	cat/cat.5842.jpg	0.45204	0	0	0.04796
1023	cat/cat.8383.jpg	0.45010	0	0	0.04990
259	cat/cat.1423.jpg	0.44465	0	0	0.05535
1454	dog/dog.1924.jpg	0.44404	1	0	0.05596
1546	dog/dog.2362.jpg	0.43850	1	0	0.06150

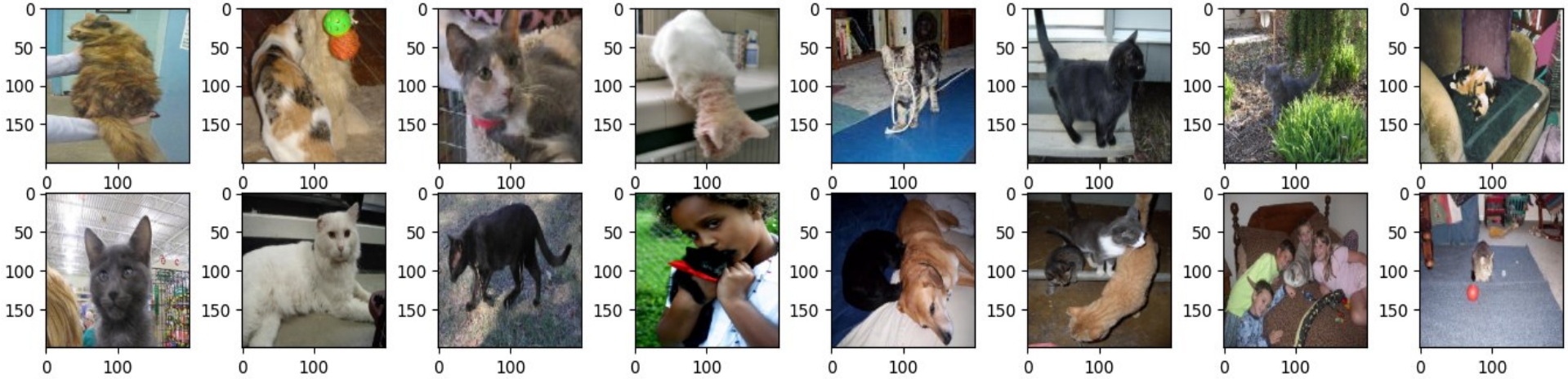
ALCUNE FOTO CHE METTONO IN DIFFICOLTÀ **MOBILENETV2**



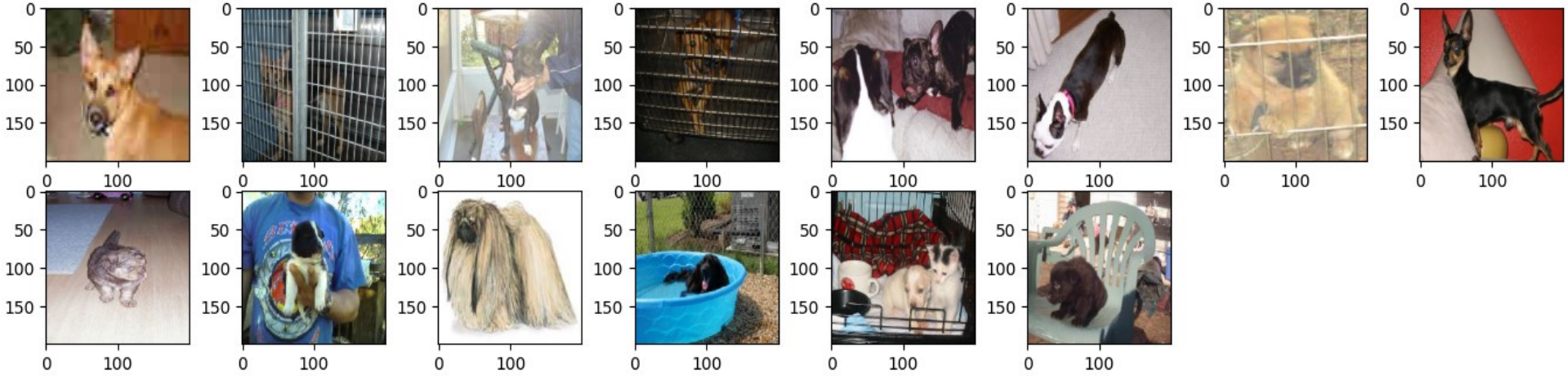
TUTTE LE FOTO CLASSIFICATE IN MANIERA ERRATA

VGG16

gatti classificati come cani



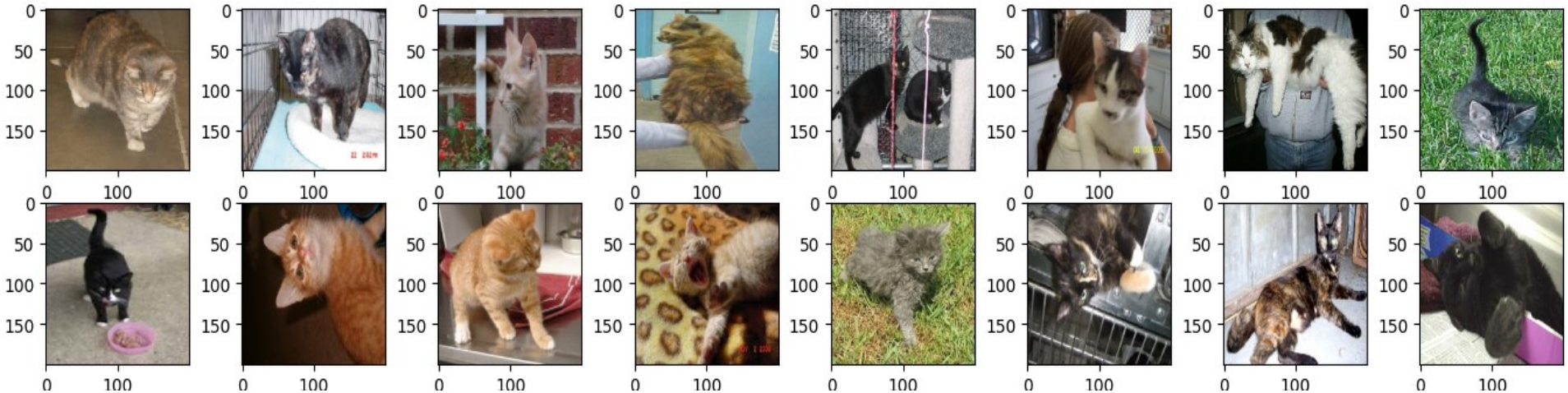
cani classificati come gatti



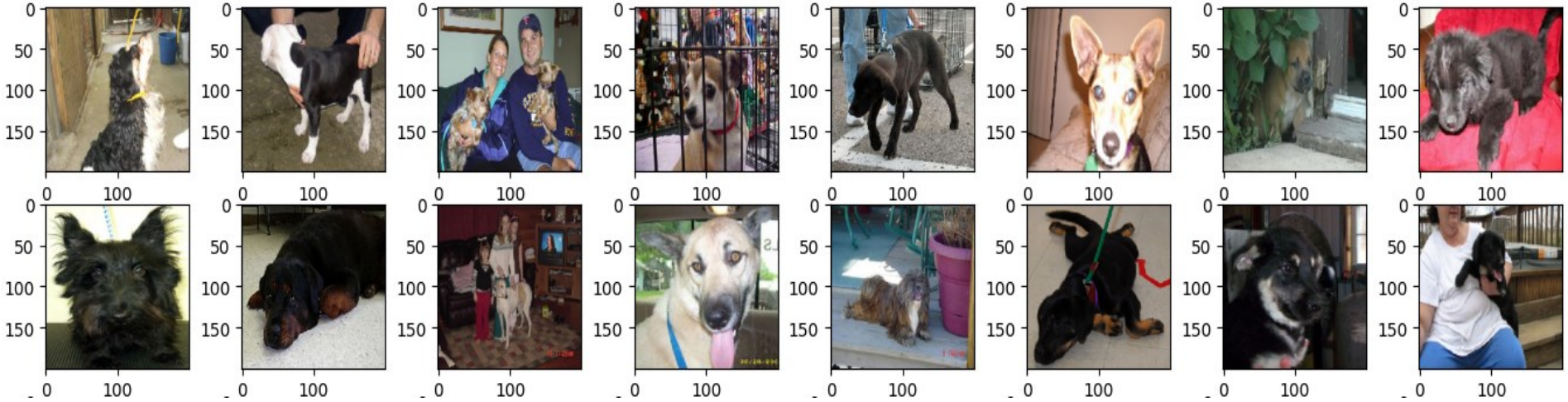
ALCUNE FOTO CHE METTONO IN DIFFICOLTÀ **VGG16**

ALCUNE FOTO CLASSIFICATE IN MANIERA ERRATA «BASE» CON MODIFICA #2

gatti classificati come cani



cani classificati come gatti



ALCUNE FOTO CHE METTONO IN DIFFICOLTÀ «BASE» CON MODIFICA #2

CONCLUSIONI

E’ stato molto interessante poter combinare insieme molti argomenti teorici e valutarne i risultati pratici, confrontando le varie soluzioni.

ALTRO MATERIALE UTILIZZATO

- Slide del corso

Università degli Studi di Ferrara

Department of Engineering Ferrara

GRAZIE PER L'ATTENZIONE!

Università degli Studi di Ferrara