UNIVERSITÀ DEGLI STUDI DI PERUGIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

Quinta esercitazione (opzionale):

CALCOLO DISTRIBUITO E PARALLELO

INTELLIGENZA ARTIFICIALE, ALLENAMENTO DI UNA MACCHINA A RICONOSCERE IMMAGINI IN CUI SONO PRESENTI CANI E GATTI

Studente: Manuel Severi

INDICE

1 INTRODUZIONE

Con la seguente relazione si va a presentare come ho realizzato un software di intelligenza artificiale che riconosce i cani e i gatti presenti in un'immagine.

INTELLIGENZA ARTIFICIALE

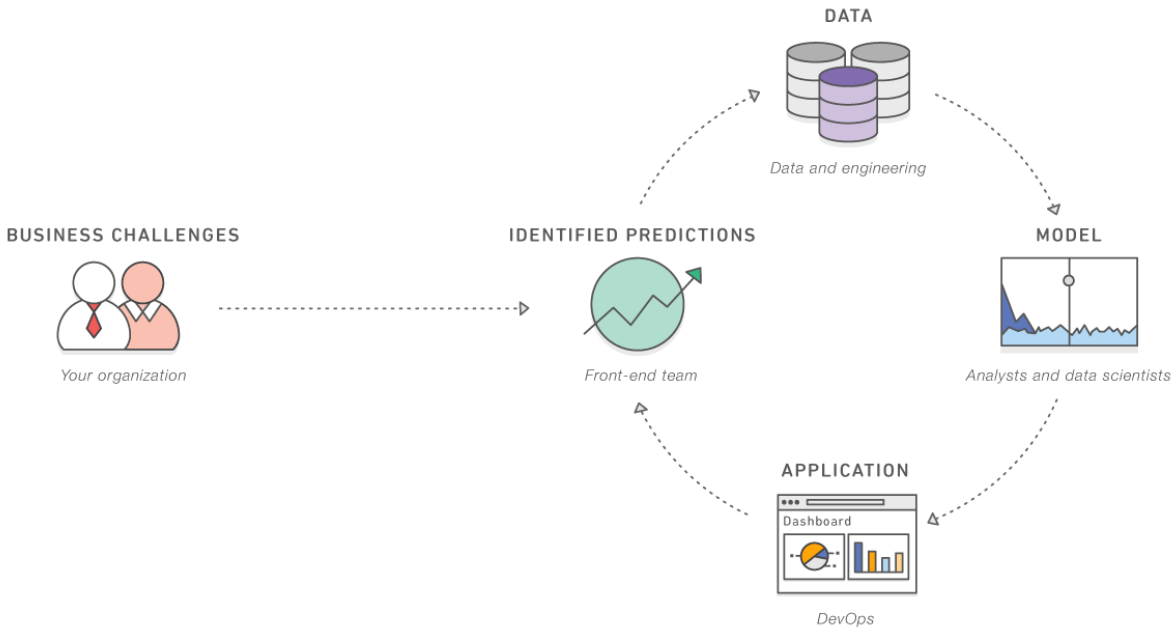
L'intelligenza artificiale è un campo delle scienze informatiche dedicato alla soluzione di problemi cognitivi comunemente associati all'intelligenza umana, ad esempio l'apprendimento, la risoluzione di problemi e il riconoscimento di pattern. L'intelligenza artificiale, a volte abbreviata in IA, spesso richiama scenari futuristici, ma non si applica solamente agli automi dei romanzi fantascientifici ed è già oggi una scienza avanzata. Il professor Pedro Domingos, un importante ricercatore in questo campo, descrive le "cinque tribù" del machine learning: i simbolisti, che traggono ispirazione dalla logica e dalla filosofia; i connettivisti, che si basano sulle neuroscienze; gli evoluzionisti, che applicano la biologia evolutiva; i bayesiani, che si occupano di statistica e di probabilità; gli analogisti, che fondano i loro studi sulla psicologia. Di recente, il miglioramento dell'efficienza del calcolo statistico ha portato i bayesiani a ottenere successi e avanzamenti in diverse aree, con il cosiddetto "machine learning", apprendimento automatico. In modo analogo, i miglioramenti nei calcoli di rete hanno permesso ai connettivisti di ottenere grandi avanzamenti in un campo secondario denominato "apprendimento approfondito". Il machine learning o apprendimento automatico e il deep learning o apprendimento approfondito sono due campi delle scienze informatiche derivati dalle discipline dell'intelligenza artificiale. In senso generale, queste tecniche si dividono in "supervisionate" o "non supervisionate"; nella prima categoria, sono utilizzati dati di addestramento che includono l'output desiderato; nella seconda categoria l'output non è incluso. Una IA diventa più "intelligente" e impara più rapidamente maggiore è la quantità di dati; ogni giorno le

aziende alimentano con enormi flussi di dati le loro soluzioni di apprendimento automatico e apprendimento approfondito raccogliendo ed estraendo dati da data warehouse come Amazon Redshift, analisi sul campo tramite crowdsourcing con Mechanical Turk o data mining dinamico tramite Kinesis Streams. Inoltre, con la comparsa dell'IoT, i dati provenienti dai sensori aumentano esponenzialmente la quantità di dati da analizzare, provenienti da sorgenti, percorsi, oggetti ed eventi che prima non erano considerati.

APPRENDIMENTO AUTOMATICO

Il machine learning, o apprendimento automatico, è il nome che viene in genere assegnato ad alcune tecniche baynesiane utilizzate per il riconoscimento di pattern e l'apprendimento. Di base, l'apprendimento automatico consiste in una serie di algoritmi che imparano e creano previsioni a partire dai dati ricevuti, ottimizzano funzioni di utilità in condizioni di incertezza, estraggono strutture non immediatamente visibili nei dati e classificano le informazioni secondo descrizioni coincise. L'apprendimento automatico viene spesso impiegato quando la programmazione esplicita sarebbe troppo poco flessibile o pratica. A differenza del codice informatico generalmente distribuito dagli sviluppatori di software per generare output specifici in un programma in base all'input, l'apprendimento automatico genera codice di tipo statistico (o modello di apprendimento automatico) che restituirà il "risultato corretto" secondo un pattern riconosciuto sulla base di esempi precedenti di input (e di output, nel caso di tecniche supervisionate). La precisione di un modello di apprendimento automatico si basa soprattutto sulla qualità e sulla quantità di dati storici. Con dati di buona qualità, un modello di apprendimento automatico è in grado di analizzare problemi a più dimensioni con miliardi di esempi e di individuare la funzione ottimale che permette di prevedere quale risultato deriverà dall'input immesso. I modelli di apprendimento automatico generalmente garantiscono una buona affidabilità statistica sulle previsioni e sulle

prestazioni generali. L'affidabilità è importante quando si tratta di decidere se utilizzare un modello di apprendimento automatico o di procedere con stime individuali. L'apprendimento automatico viene spesso utilizzato per fornire previsioni basate su dati storici. Ad esempio, molte aziende lo impiegano per prevedere la quantità di prodotti che verranno venduti nei trimestri fiscali successivi in base a dati demografici specifici, oppure per stimare il profilo del cliente tipo che mostreranno più probabilmente maggiore o minore fedeltà al marchio. Queste previsioni permettono di prendere decisioni informate, creare esperienze utente più personali e ridurre i costi di fidelizzazione dei clienti. L'apprendimento automatico è complementare alla business intelligence: questa raccoglie dati aziendali riferiti al passato, mentre il primo fornisce previsioni sul futuro basate sulle tendenze e le transazioni passate. L'implementazione dell'apprendimento automatico in azienda prevede diverse fasi. Prima di tutto, è necessario identificare i problemi a cui applicarla, ovvero l'area in cui le previsioni consentirebbero maggiori vantaggi. Quindi è importante raccogliere dati in base a specifici parametri aziendali (transazioni, vendite, tassi di abbandono e così via). Con questi dati aggregati è così possibile creare un modello di apprendimento automatico. Il modello fornirà previsioni che potranno essere riutilizzate nei sistemi aziendali per prendere decisioni informate.



I casi principali in cui si usa l'apprendimento automatico sono:

- **Rilevamento delle anomalie** → Identifica elementi, eventi e osservazioni che non sono conformi a un pattern predeterminato o ad altri oggetti in un set di dati.
- **Rilevamento di attività fraudolente** → Crea modelli predittivi che facilitano l'identificazione di transazioni potenzialmente fraudolente e di recensioni malevoli o inappropriate.
- **Tasso di abbandono dei clienti** → Trova i clienti a rischio di abbandono, per consentirti di coinvolgerli anticipatamente con promozioni e offerte di servizio clienti.
- **Personalizzazione dei contenuti** → Con Amazon Machine Learning il tuo sito Web può offrire maggiore personalizzazione tramite modelli di analisi predittiva che permettono di consigliare articoli e ottimizzare il flusso del sito Web in base alle azioni precedenti dei clienti.

APPRENDIMENTO APPROFONDITO

L'apprendimento approfondito è una branca dell'apprendimento automatico che impiega algoritmi su più livelli per ottenere una comprensione più chiara dei dati. Gli algoritmi non sono più vincolati esclusivamente alla creazione di un set di relazioni comprensibili come in regressioni di base. Con l'apprendimento approfondito, questi diversi livelli di algoritmi non lineari permettono di creare rappresentazioni distribuite che interagiscono tra loro in base a diversi fattori. Con grandi volumi di dati su cui lavorare, gli algoritmi di apprendimento approfondito iniziano a identificare le relazioni tra elementi. Queste relazioni possono essere, ad esempio, forme, colori, parole o altro. A partire da questi elementi, il sistema può essere impiegato per creare previsioni. Con l'apprendimento automatico e l'intelligenza artificiale, l'apprendimento approfondito è in grado di identificare un numero maggiore di relazioni rispetto a quelle codificabili manualmente tramite software, nonché di

rilevare relazioni che sfuggirebbero al giudizio umano. Grazie a una fase di addestramento ottimale, la rete di algoritmi sarà in grado di fornire previsioni e interpretazioni a partire da dati estremamente complessi.

I casi principali in cui si usa l'apprendimento automatico sono:

- **Classificazione e categorizzazione di immagini e video** → Le reti neurali convoluzionali hanno prestazioni molto migliori degli umani in molte attività visuali, tra cui la classificazione di oggetti. Dati milioni di immagini etichettate, il sistema di algoritmi è in grado di identificare il soggetto delle immagini. Molti servizi di archiviazione di immagini includono tecnologie di riconoscimento facciale, basate sull'apprendimento approfondito. Si tratta di una funzione centrale in Amazon Rekognition, Amazon Prime Photos e nel servizio Firefly di Amazon.
- **Riconoscimento vocale** → Google home e altri assistenti virtuali sono stati progettati per riconoscere richieste e offrire risposte. Il riconoscimento di una voce è un'attività che gli umani sviluppato fin dalla giovane età; ma solo di recente i computer sono diventati in grado di ascoltare e rispondere a stimoli vocali umani. I diversi accenti e forme verbali rendono queste attività estremamente complicate per una macchina che elabora dati matematici e informatici. Con l'apprendimento approfondito, un sistema di algoritmi può determinare con maggiore semplicità gli stimoli verbali e le richieste.
- **Riconoscimento del linguaggio naturale** → L'elaborazione del linguaggio naturale cerca di insegnare ai sistemi come comprendere linguaggio, tonalità e contesto. In questo modo, l'algoritmo sarebbe in grado di discernere concetti complessi come l'emozione e il sarcasmo. Si tratta di un campo in grande crescita, perché molte aziende puntano ad automatizzare il servizio clienti con bot vocali o di testo, ad esempio Amazon Lex.
- **Motori di raccomandazione** → Lo shopping online spesso richiede la presenza di suggerimenti per contenuti personalizzati correlati

agli articoli che un utente è più propenso ad acquistare, film che può desiderare di guardare o notizie a cui potrebbe essere interessato. In passato, questi sistemi venivano messi in modo manualmente da persone che creavano associazioni tra i diversi articoli. Tuttavia, con l'avvento dei Big Data e dell'apprendimento approfondito, non è più necessario l'intervento umano, perché gli algoritmi sono in grado di identificare gli articoli che possono attirare l'interesse di un utente in base alle visite e agli acquisti passati e al confronto con le attività di altri utenti

2 | STRUMENTI E LINGUAGGI UTILIZZATI

PYTHON

Python è un linguaggio di programmazione moderno, dalla sintassi semplice e potente che ne facilita l'apprendimento. Gli ambiti di applicazione di questo linguaggio di programmazione sono svariati: sviluppo di siti o applicazioni Web e desktop, realizzazione di interfacce grafiche, amministrazione di sistema, calcolo scientifico e numerico, database, giochi, grafica 3D, eccetera. La storia di questo linguaggio di programmazione inizia nei primi anni ottanta. In quegli anni, al National Research Institute for Mathematics and Computer Science (CWI) di Amsterdam, alcuni ricercatori tra cui Guido Van Rossum hanno sviluppato un linguaggio di nome ABC, molto potente ed elegante, che era diventato popolare nel mondo Unix. Qualche anno dopo (fine anni ottanta) Guido Van Rossum ha avuto una serie di idee mirate al miglioramento di ABC, e pertanto si mise a lavorare allo sviluppo di un nuovo linguaggio: Python, appunto. Oggi esistono numerosi linguaggi di programmazione, ma cosa spinge ad usare Python, cosa c'è di particolare in questo linguaggio? Per rispondere alla domanda esaminiamo alcuni dei punti di forza di Python ed alcune delle sue applicazioni principali.

- È GRATUITO → Python è completamente gratuito ed è possibile usarlo e distribuirlo senza restrizioni di copyright. Nonostante sia free, da oltre 25 anni Python ha una comunità molto attiva, e riceve costantemente miglioramenti che lo mantengono aggiornato e al passo coi tempi.
- È PORTABILE → si tratta di un linguaggio interpretato, quindi lo stesso codice può essere eseguito su qualsiasi piattaforma purché abbia l'interprete Python installato.

- È PERFORMANTE → i programmi vengono automaticamente compilati in un formato chiamato bytecode prima di essere eseguiti. Questo formato è compatto ed efficiente, e garantisce quindi prestazione elevate. Inoltre, diverse strutture dati, funzioni, e moduli di Python sono implementati internamente in C per essere ancora più performanti.
- Inoltre Python gestisce autonomamente la memoria grazie ad un meccanismo che prende il nome di Gabarage Cllection inoltre può interagire senza alcun problema con altri linguaggi.

Per capire che potenza ha questo linguaggio basta pensare che viene scelto da grandi realtà come:

- la NASA che lo usa per lo sviluppo di sistemi di controllo;
- Yahoo! ha sviluppato in Python alcuni servizi internet;
- Google, Youtube e RedHat usano Python.

GOOGLE COLAB

Google Colab è una piattaforma online gratuita che offre un servizio di cloud hosting di Jupyter Notebooks, con il supporto a GPU. Possiamo usare differenti librerie come PyTorch, Tensorflow, Keras e OpenCV giusto per citarne alcune. Google Colab ha dei limiti, come evidenziato dalla pagina FAQ, ma la versatilità che il servizio offre è in grado di offuscarli facilmente: quando raggiungi questi limiti basta cercare delle soluzioni diversi per aggirarli. Per muovere i primi passi con Tensorflow, è il posto giusto. Puoi creare il tuo primo neurone artificiale per la risoluzione di un problema reale direttamente attraverso Tensorflow 2. Lo stesso codice eseguito in un classico personal computer andrebbe sicuramente incontro a delle limitazioni legate all'hardware, utilizzando questo servizio, avendo una connessione internet, non bisogna più preoccuparsi dell'hardware.

TENSORFLOW

TensorFlow è una piattaforma open source end-to-end per l'apprendimento automatico. Dispone di un ecosistema completo e flessibile di strumenti, librerie e risorse della comunità che consente ai ricercatori di spingere lo stato dell'arte in ML (MACHINE LEARNING) e gli sviluppatori di creare e distribuire facilmente applicazioni basate su ML. I vantaggi che abbiamo nell'usare TensorFlow sono:

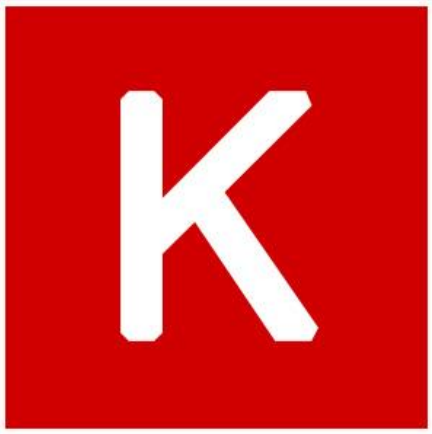
- **Easy model building** → costruire e addestrare modelli ML facilmente usando API intuitive di alto livello come Keras, il che rende l'iterazione immediata del modello e il debug semplice. Per attività di formazione ML di grandi dimensioni, utilizzare l'API di strategia di distribuzione per la formazione distribuita su diverse

configurazioni hardware senza modificare la definizione del modello.

- **Robust ML production anywhere** → Formare e distribuire facilmente modelli nel cloud, nel browser o sul dispositivo, indipendentemente dalla lingua utilizzata. Utilizzare TensorFlow Extended (TFX) se è necessaria una pipeline ML di produzione completa. Per eseguire l'inferenza su dispositivi mobili e periferici, utilizzare TensorFlow Lite. Formare e distribuire modelli in ambienti JavaScript utilizzando TensorFlow.js.
- **Powerful experimentation for research** → Un'architettura semplice e flessibile per portare nuove idee (dall'idea al codice), a modelli all'avanguardia e alle pubblicazioni sempre più veloci. TensorFlow supporta anche un ecosistema di potenti librerie e modelli di componenti aggiuntivi con cui sperimentare, tra cui Ragged Tensor, TensorFlow Probability, Tensor2Tensor e BERT.

Per comprendere quanto è vantaggioso questa piattaforma basta andare nel sito ufficiale di TensorFlow e vedere quali grandi compagnie la utilizzano. Citandone alcune come CocaCola, AirBnb, Google, Intel, Lenovo, Paypal, Twitter e molte altre della stessa caratura.

KERAS



Keras è una libreria open source per l'apprendimento automatico e le reti neurali, scritta in Python. È progettata come un'interfaccia a un livello di astrazione superiore di altre librerie simili di più basso livello, e supporta come back-end le librerie TensorFlow e Microsoft Cognitive Toolkit (CNTK). Progettata per permettere una rapida prototipazione di reti neurali profonde, si concentra sulla facilità d'uso, la modularità e l'estensibilità. È stata sviluppata come parte del progetto di ricerca ONEIROS, e il suo autore principale è François Chollet, di Google. Nel 2017 il team di TensorFlow ha deciso di supportare Keras ufficialmente. Chollet ha spiegato che Keras è stata pensata come un'interfaccia e non come una libreria stand-alone. Offre una serie di moduli che permettono di sviluppare reti neurali profonde indipendentemente dal back-end utilizzato, con un linguaggio comune e intuitivo.
È stato sviluppato con l'obiettivo di consentire una rapida sperimentazione. Essere in grado di passare dall'idea al risultato con il minor ritardo possibile è la chiave per fare una buona ricerca.

È consigliato usare Keras se hai bisogno di una libreria di deep learning che:

- Consente la prototipazione semplice e veloce (attraverso facilità d'uso, modularità ed estensibilità).
- Supporta sia reti convoluzionali che reti ricorrenti, nonché combinazioni delle due.
- Funziona perfettamente su CPU e GPU.

Keras è compatibile con: Python 2.7 e 3.6.

3 CODICE DEL PROGETTO

Come precedentemente detto, è stato usato Google Colab per eseguire il codice che andrò a spiegare, così da non dovermi preoccupare delle caratteristiche hardware del mio pc, purtroppo insufficienti per realizzare una rete neurale convoluzione in grado di riconoscere cani e gatti.
È stata seguita la guida fornita da Damiano Perri. Inizialmente bisogna scaricare un dataset dove sono presenti circa 25000 foto di cani e gatti, poi da caricare direttamente sul Google Drive personale così da poterlo utilizzare ogni volta che si apre la pagina del progetto.
Una volta che ho caricato il file sul drive ripasso alla pagina di colab e come si dice in gergo “monto” il drive nel mio progetto per poter utilizzare i file con il seguente codice

```
from google.colab import drive
drive.mount('/content/drive')
!unzip "/content/drive/My Drive/Appunti/dogandcat/dogs-vs-cats.zip"
```

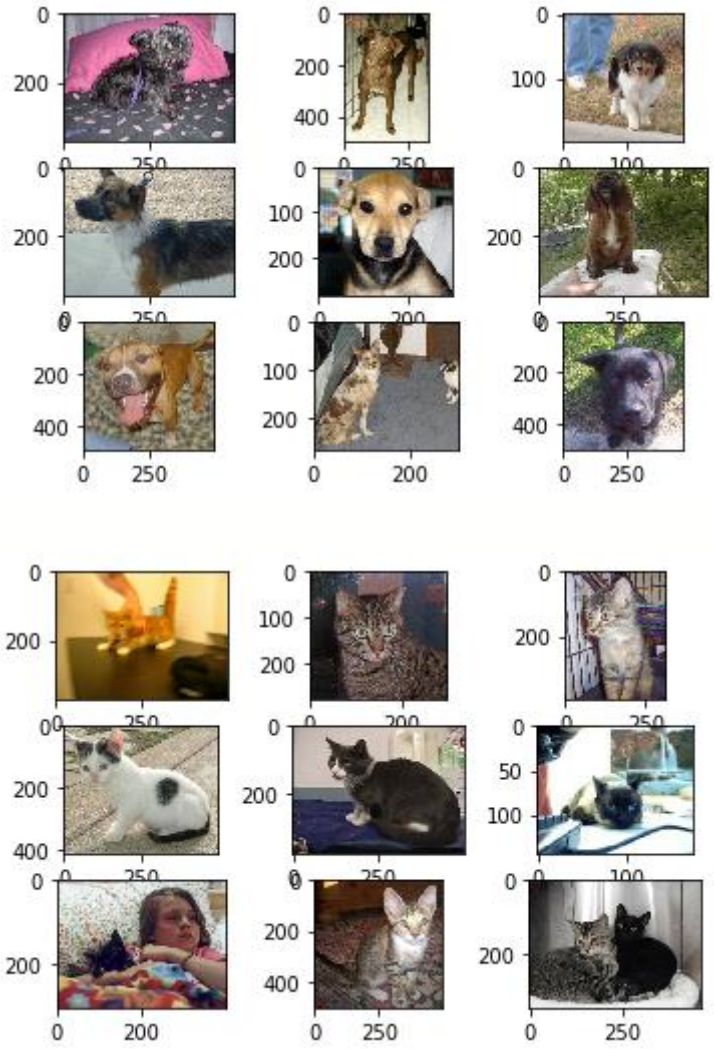
Continuo unzippando quello presente nello zip principale. Per fare ciò è importante ricordarsi di creare os e zipfile ed eseguire in seguito questo codice.

```
import os
import zipfile
local_zip = '/content/train.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/content')
zip_ref.close()
```

Ora andiamo a controllare che forma e che dimensioni hanno alcune immagine casuali del nostro dataset, in questo caso vado a controllare le prime nove immagine di cani

```
# plot dog photos from the dogs vs cats dataset
from matplotlib import pyplot
from matplotlib.image import imread
# define location of dataset
folder = '/content/train/'
# plot first few images
for i in range(9):
  # define subplot
  pyplot.subplot(330 + 1 + i)
  # define filename
  filename = folder + 'dog.' + str(i) + '.jpg'
  # load image pixels
  image = imread(filename)
  # plot raw pixel data
  pyplot.imshow(image)
# show the figure
pyplot.show()
```

controllando in seguito anche le immagini dei gatti andando a modificare parte del filename cambiando, appunto la stringa dog con cat. Otterremo un risultato molto simile alle due immagini successive.



Come possiamo vedere le immagini sono tra le più disparate sia per colori che dimensione.
Le foto dovranno essere rimodellate in modo che tutte le immagini abbiano la stessa forma. La scelta è caduta su una piccola immagine quadrata. Ci sono molti modi per raggiungere questo obiettivo, anche se il più comune è una semplice operazione di ridimensionamento che allungherà e deformerà le proporzioni di ogni immagine e la forzerà nella nuova forma. Potremmo caricare tutte le foto e guardare la distribuzione delle larghezze e altezze delle foto, quindi progettare una nuova dimensione della foto che rifletta al meglio ciò che è più probabile vedere

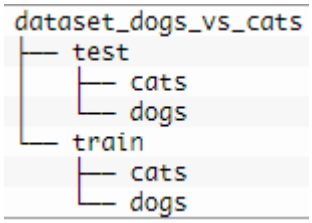
in pratica. In questo caso, scelgo questo approccio e scelgo una dimensione fissa di 200 × 200 pixel.
Sono 25.000 immagini con 200x200x3 pixel ciascuna o 3.000.000.000 di valori di pixel a 32 bit perciò servono all'incirca 12 gigabyte di RAM. Potremmo caricare tutte le immagini, rimodellarle e memorizzarle come un singolo array NumPy. Possiamo scrivere codice personalizzato per caricare le immagini in memoria e ridimensionarle come parte del processo di caricamento, quindi salvarle pronte per la modellazione. L'esempio seguente utilizza l'API di elaborazione delle immagini di Keras per caricare tutte le 25.000 foto nel set di dati di addestramento e le ridisegna in foto quadrate 200 × 200. L'etichetta è anche determinata per ogni foto in base ai nomi dei file. Viene quindi salvata una tupla di foto ed etichette.

```
# load dogs vs cats dataset, reshape and save to a new file
from os import listdir
from numpy import asarray
from numpy import save
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
# define location of dataset
folder = 'train/'
photos, labels = list(), list()
# enumerate files in the directory
for file in listdir(folder):
        # determine class
        output = 0.0
        if file.startswith('cat'):
                output = 1.0
        # load image
        photo = load_img(folder + file, target_size=(200, 200))
        # convert to numpy array
        photo = img_to_array(photo)
        # store
        photos.append(photo)
        labels.append(output)
# convert to a numpy arrays
photos = asarray(photos)
labels = asarray(labels)
print(photos.shape, labels.shape)
# save the reshaped photos
save('dogs_vs_cats_photos.npy', photos)
save('dogs_vs_cats_labels.npy', labels)
```

ricordo che la seguente operazione potrebbe richiedere diversi minuti ma creerà due file che ridurranno i tempi in seguito. I file avranno come nome dogs_vs_cats_photos.npy e dogs_vs_cats_labels.npy se come nel mio caso si utilizza Colab e consigliabile scaricare i seguenti file che poi in seguito saranno ricaricati ed aperti con il seguente codice

```
from numpy import load
photos = load('dogs_vs_cats_photos.npy')
labels = load('dogs_vs_cats_labels.npy')
print(photos.shape, labels.shape)
```

se invece si vogliono creare delle directory standard di pre-elaborazione utilizzando la classe Keras ImageDataGenerator e l'API flow_from_directory (). L'esecuzione sarà più lenta ma verrà eseguita su più macchine se si dispone di calcolo distribuito. Visto i permessi che ci concede Colab la directory va creata manualmente e deve avere la seguente struttura



Ora creaimo uno script in python per trattenere il 25% delle immagini nel set di test. Questo viene fatto costantemente fissando il seme per il generatore di numeri pseudocasuali in modo da ottenere la stessa suddivisione dei dati ogni volta che il codice viene eseguito.

```
import random
import shutil, sys

dataset_home = 'dataset_dogs_vs_cats/'
subdirs = ['train/', 'test/']

# seed random number generator
random.seed(1)
# define ratio of pictures to use for validation
val_ratio = 0.25
# copy training dataset images into subdirectories
src_directory = 'train/'
for file in listdir(src_directory):
  src = src_directory + '/' + file
  dst_dir = 'train/'
  if random.random() < val_ratio:
    dst_dir = 'test/'
  if file.startswith('cat'):
    dst = dataset_home + dst_dir + 'cats/'  + file
    shutil.copyfile(src, dst)
  elif file.startswith('dog'):
    dst = dataset_home + dst_dir + 'dogs/'  + file
    shutil.copyfile(src, dst)
```

MODELLO CNN DI BASE

Ora andiamo a realizzare un modello di rete neurale convoluzione di base per il set di dati “cani e gatti”. Un modello di base stabilirà una prestazione minima del modello a cui tutti gli altri modelli possono essere confrontati, nonché un'architettura di modello che possiamo usare come base di studio e miglioramento. Andremo a definire un modello che ci permette di definire un modello di rete neurale convoluzionale per il problema dei cani e gatti con un blocco in stile vgg. Per poi andare a ridimensionare i valori dei pixel nell’intervallo 0-1. Il modello funzionerà per 20 epoche vista la sua semplicità e una volta adattato, il modello finale può essere valutato direttamente sul set di dati del test e riportato l'accuratezza della classificazione. Tutto quello appena descritto è svolto dalla seguente porzione di codice.

```
# baseline model for the dogs vs cats dataset
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator

# define cnn model
def define_model():
	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(Flatten())
	model.add(Dense(128, activation='relu',
kernel_initializer='he_uniform'))
	model.add(Dense(1, activation='sigmoid'))
	# compile model
	opt = SGD(lr=0.001, momentum=0.9)
	model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])
	return model

# plot diagnostic learning curves
def summarize_diagnostics(history):
	# plot loss
	pyplot.subplot(211)
	pyplot.title('Cross Entropy Loss')
	pyplot.plot(history.history['loss'], color='blue', label='train')
	pyplot.plot(history.history['val_loss'], color='orange', label='test')
	# plot accuracy
```

```
        pyplot.subplot(212)
        pyplot.title('Classification Accuracy')
        pyplot.plot(history.history['accuracy'], color='blue', label='train')
        pyplot.plot(history.history['val_accuracy'], color='orange',
label='test')
        # save plot to file
        filename = sys.argv[0].split('/')[-1]
        pyplot.savefig(filename + '_plot.png')
        pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
        # define model
        model = define_model()
        # create data generator
        datagen = ImageDataGenerator(rescale=1.0/255.0)
        # prepare iterators
        train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
                class_mode='binary', batch_size=64, target_size=(200, 200))
        # fit model
        history = model.fit_generator(train_it, steps_per_epoch=len(train_it),
                validation_data=test_it, validation_steps=len(test_it),
epochs=20, verbose=0)
        # evaluate model
        _, acc = model.evaluate_generator(test_it, steps=len(test_it),
verbose=0)
        print('> %.3f' % (acc * 100.0))
        # learning curves
        summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()
```

ora per ottenere un risultato andiamo a realizzare un vgg a blocco con singolo livello convoluzionale con 32 filtri seguito da un livello di pooling massimo in questo caso la funzione define_model() per questo modello è così strutturata.

```
# define cnn model
def define_model():
        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(Flatten())
        model.add(Dense(128, activation='relu',
kernel_initializer='he_uniform'))
        model.add(Dense(1, activation='sigmoid'))
        # compile model
        opt = SGD(lr=0.001, momentum=0.9)
        model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])
        return model
```

l’esecuzione di questo primo esempio restituirà il seguente output

```
Found 18696 images belonging to 2 classes.
Found 6304 images belonging to 2 classes.
> 72.313
```

I risultati possono differire in questo caso abbiamo ottenuto un’accuratezza all’incirca del 72%. Migliorando la capacità del modello ed il numero di blocchi migliora anche l’accuratezza.

MODELLO COMPLESSO FINALE

Nella sezione precedente, abbiamo sviluppato un modello di base utilizzando blocchi in stile VGG e abbiamo scoperto una tendenza al miglioramento delle prestazioni con una maggiore capacità del modello. In questa sezione, inizieremo con il modello di base con tre blocchi VGG (ovvero VGG 3) ed esploreremo alcuni semplici miglioramenti del modello. Possiamo esplorare due approcci per tentare di affrontare questo overfitting: regolarizzazione del dropout e aumento dei dati. Entrambi questi approcci dovrebbero rallentare il tasso di miglioramento durante l'allenamento e, si spera, contrastare il sovradimensionamento del set di dati.

La regolarizzazione del dropout è un modo computazionalmente economico per regolarizzare una rete neurale profonda. Il dropout funziona rimuovendo probabilisticamente o " eliminando " gli input in un layer, che possono essere variabili di input nel campione di dati o attivazioni da un layer precedente. Ha l'effetto di simulare un gran numero di reti con strutture di rete molto diverse e, a sua volta, di rendere i nodi della rete generalmente più robusti per gli input.

L'aumento dei dati delle immagini è una tecnica che può essere utilizzata per espandere artificialmente le dimensioni di un set di dati di training creando versioni modificate delle immagini nel set di dati. La formazione di modelli di reti neurali di apprendimento profondo su più dati può comportare modelli più abili e le tecniche di aumento possono creare variazioni delle immagini che possono migliorare la capacità dei modelli adattati di generalizzare ciò che hanno appreso su nuove immagini. L'aumento dei dati può anche fungere da tecnica di regolarizzazione, aggiungendo rumore ai dati di allenamento e incoraggiando il modello ad apprendere le stesse caratteristiche, invariante rispetto alla loro posizione nell'input. Piccole modifiche alle foto di input di cani e gatti potrebbero essere utili per questo problema, come piccoli spostamenti e ribaltamenti orizzontali. Questi aumenti possono essere specificati come argomenti per ImageDataGenerator utilizzato per il set di dati di training.

Keras offre una gamma di modelli pre-addestrati che possono essere caricati e utilizzati in tutto o in parte tramite l' API delle applicazioni Keras. Un modello utile per l'apprendimento del trasferimento è uno dei modelli VGG, come VGG-16 con 16 strati che al momento dello sviluppo ha raggiunto i migliori risultati nella sfida della classificazione delle foto di ImageNet.

Il modello è composto da due parti principali, la parte di estrazione delle caratteristiche del modello costituita da blocchi VGG e la parte classificatore del modello costituita da livelli completamente collegati e il livello di output. È possibile utilizzare la parte di estrazione delle caratteristiche del modello e aggiungere una nuova parte di classificazione del modello su misura per il set di dati di cani e gatti. In particolare, possiamo tenere fissati i pesi di tutti gli strati convoluzionali durante l'allenamento e addestrare solo nuovi livelli completamente collegati che impareranno a interpretare le caratteristiche estratte dal modello e fare una classificazione binaria. Ciò può essere ottenuto caricando il modello VGG-16, rimuovendo i livelli completamente collegati dall'estremità di uscita del modello, quindi aggiungendo i nuovi livelli completamente collegati per interpretare l'output del modello ed effettuare una previsione. La parte classificatore del modello può essere rimossa automaticamente impostando l’argomento " include_top " su " False ", che richiede anche che la forma dell'input sia specificata anche per il modello, in questo caso (224, 224, 3). Ciò significa che il modello caricato termina all'ultimo livello di pooling massimo, dopo di che è possibile aggiungere manualmente un livello Flatten e i nuovi livelli di classificazione. La seguente funzione define_model () implementa questo e restituisce un nuovo modello pronto per l'allenamento. Una volta creato, possiamo addestrare il modello come prima nel set di dati di addestramento.

In questo caso non sarà richiesto molto addestramento, poiché solo il nuovo livello completamente connesso e di uscita ha pesi allenabili. Pertanto, fisseremo il numero di epoche di addestramento a 10. Il modello VGG16 è stato addestrato su uno specifico set di dati di sfida ImageNet. Come tale, è configurato per le immagini di input previste per avere la forma 224 × 224 pixel. Useremo questo come dimensione target quando si caricano foto dal set di dati di cani e gatti. Il modello prevede inoltre che le immagini siano centrate. Cioè, per avere i valori medi dei pixel di ciascun canale (rosso, verde e blu) calcolati sul set di dati di training di ImageNet sottratto dall'input. Keras fornisce una funzione per eseguire questa preparazione per singole foto tramite la funzione preprocess_input (). Tuttavia, possiamo ottenere lo stesso effetto con ImageDataGenerator impostando l'argomento " featurewise_center " su " True " e specificando manualmente i valori medi dei pixel da utilizzare durante il centraggio come valori medi dal set di dati di training di ImageNet: [123.68, 116.779, 103.939]. Di seguito è riportato l'elenco completo dei codici del modello VGG per l'apprendimento del trasferimento sul set di dati di cani e gatti.

```
import sys
from matplotlib import pyplot
from keras.utils import to_categorical
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
# define cnn model
def define_model():
  # load model
  model = VGG16(include_top=False, input_shape=(224, 224, 3))
  # mark loaded layers as not trainable
  for layer in model.layers:
    layer.trainable = False
  # add new classifier layers
  flat1 = Flatten()(model.layers[-1].output)
  class1 = Dense(128, activation='relu', kernel_initializer='he_uniform')(flat1)
  output = Dense(1, activation='sigmoid')(class1)
  # define new model
  model = Model(inputs=model.inputs, outputs=output)
  # compile model
  opt = SGD(lr=0.001, momentum=0.9)
  model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
  return model
```

```
# plot diagnostic learning curves
def summarize_diagnostics(history):
  # plot loss
  pyplot.subplot(211)
  pyplot.title('Cross Entropy Loss')
  pyplot.plot(history.history['loss'], color='blue', label='train')
  pyplot.plot(history.history['val_loss'], color='orange', label='test')
  # plot accuracy
  pyplot.subplot(212)
  pyplot.title('Classification Accuracy')
  pyplot.plot(history.history['accuracy'], color='blue', label='train')
  pyplot.plot(history.history['val_accuracy'], color='orange', label='test')
  # save plot to file
  filename = sys.argv[0].split('/')[-1]
  pyplot.savefig(filename + '_plot.png')
  pyplot.close()

# run the test harness for evaluating a model
def run_test_harness():
  # define model
  model = define_model()
  # create data generator
  datagen = ImageDataGenerator(featurewise_center=True)
  # specify imagenet mean values for centering
  datagen.mean = [123.68, 116.779, 103.939]
  # prepare iterator
  train_it = datagen.flow_from_directory('dataset_dogs_vs_cats/train/',
    class_mode='binary', batch_size=64, target_size=(224, 224))
  test_it = datagen.flow_from_directory('dataset_dogs_vs_cats/test/',
    class_mode='binary', batch_size=64, target_size=(224, 224))
  # fit model
  history = model.fit_generator(train_it, steps_per_epoch=len(train_it),
    validation_data=test_it, validation_steps=len(test_it), epochs=10, verbose=1
)
  # evaluate model
  _, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=0)
  print('> %.3f' % (acc * 100.0))
  # learning curves
  summarize_diagnostics(history)

# entry point, run the test harness
run_test_harness()
```

otteniamo questo output

```
Found 18696 images belonging to 2 classes.
Found 6304 images belonging to 2 classes.
> 97.366
```

Possiamo vedere come abbiamo raggiungo un risultato quasi strabiliante, con un’accuratezza superiore al 97%

4 RISULTATO DEL PROGETTO

Per controllare se la nostra rete svolge il lavoro che vogliamo andiamo a sottoporgli un'immagine dove è presente un cane e vediamo se effettivamente restituisce il valore 1. Se restituisce questo valore significa che l'IA ha riconosciuto un cane all'interno dell'immagine.

Il codice per fare ciò è il seguente:

```
# make a prediction for a new image.
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model

# load and prepare the image
def load_image(filename):
  # load the image
  img = load_img(filename, target_size=(224, 224))
  # convert to array
  img = img_to_array(img)
  # reshape into a single sample with 3 channels
  img = img.reshape(1, 224, 224, 3)
  # center pixel data
  img = img.astype('float32')
  img = img - [123.68, 116.779, 103.939]
  return img

# load an image and predict the class
def run_example():
  # load the image
  img = load_image('sample_image.jpg')
  # load model
  model = load_model('final_model.h5')
  # predict the class
  result = model.predict(img)
  print(result[0])

# entry point, run the example
run_example()
```

il modello a cui si fa riferimento è quello che abbiamo precedentemente salvato che otteneva un'accuratezza del 97%.

l'immagine che sono andato a sottoporre al codice da me sviluppato è la seguente e, se tutto è stato svolto nel modo coretto l'output del codice deve essere il valore 1.

Come possiamo facilmente notare noi umani, nell'immagine precedente è presente un cane, ma anche la nostra macchina è stata in grado di riconoscere l'animale infatti la risposta è stata [1.] come possiamo vedere dall'immagine sottostante

```
5
6 # load and prepare the image
7 def load_image(filename):
8   # load the image
9   img = load_img(filename, target_size=(224, 224))
10  # convert to array
11  img = img_to_array(img)
12  # reshape into a single sample with 3 channels
13  img = img.reshape(1, 224, 224, 3)
14  # center pixel data
15  img = img.astype('float32')
16  img = img - [123.68, 116.779, 103.939]
17  return img
18
19 # load an image and predict the class
20 def run_example():
21  # load the image
22  img = load_image('sample_image.jpg')
23  # load model
24  model = load_model('final_model.h5')
25  # predict the class
26  result = model.predict(img)
27  print(result[0])
28
29 # entry point, run the example
30 run_example()
```

```
[1.]
```

5 CONCLUSIONE

in questa relazione ho potuto mettermi a confronto con i principi base del machine learning per il riconoscimento di un'immagine. Ho realizzata una rete di convoluzione neurale di base utilizzando il modello VGG spiegando i vari passi ed i codici che sono andato ad utilizzare. Il risultato è stato strabiliante visto anche l'accuratezza che ho ottenuto nel mio modello nonostante il dataset non sia così ampio, infatti per fare il training dei machine learning si usano dati nell'ordine minimo consigliato dei terabyte.

Questo esempio basilare è stato estremamente importante , in grado di dimostrare quanto questa scienza possa essere potente e quanti milioni di utilizzi nella vita reale di ogni persona o durante un processo produttivo può ottenere.