Assignment 4

Marana Nicolò

November 30, 2018

Abstract

L'assignment 4 consiste nell'effettuare fine tuning di una CNN pre-trainata sul dataset IMAGENET.

1 Dataset

Il dataset creato contiene immagini di animali appartenenti a quattro classi differenti.

Vengono di seguito elencate le caratteristiche del dataset:

- Numero Classi: 4
  - Classe 1: Orsi - Bears
  - Classe 2: Uccelli - Birds
  - Classe 3: Gatti - Cats
  - Classe 4: Cani - Dogs
- Training Set: contiene 80 immagini per ogni classe
- Validation Set: contiene 20 immagini per ogni classe
- Testing Set: contiene 25 immagini per ogni classe

Contiene in totale 500 immagini, ed è possibile visionarlo al seguente link:

`https://drive.google.com/drive/folders/1HQ1n01kG-OtfuMc_fsY9TMIJKP17Gzyp?usp=sharing`

Il task è quindi quello di riconoscere data un'immagine a quale classe appartiene.

2 Librarie Utilizzate

Di seguito vengono illustrate le principali librerie utilizzate:

1. **Keras**: Permette di creare modelli di NN in modo semplice, funzionando da wrapper e basandosi su Tensorflow o Theano.
2. **Matplotlib**: Permette di creare dei plot dei dati in modo da analizzarne i risultati.
3. **Datetime**: Permette di stimare i tempi di esecuzione.

3 Lavoro Svolto

Di seguito viene illustrato il lavoro svolto spiegando brevemente il contenuto dei file della cartella codice

3.1 seed.py

Contiene la funzione setup() che serve ad impostare i seed dell'ambiente in modo da ottenere la riproducibilità degli esperimenti evitando così di ottere risultati differenti ad ogni esecuzione.

3.2 preprocessing.py

Contiene la funzione data_generator() che serve per importare il dataset di immagini convertendolo in formato adatto per l'esecuzione delle CNN.

```
def data_generator(path, shuffle=True):
    generator = ImageDataGenerator(rescale=1. / 255)
    data = generator.flow_from_directory(path, target_size
        =(224, 224), batch_size=32, class_mode='categorical',
        shuffle=shuffle)
    return data
```

3.3 model.py

Contiene le seguenti funzioni:

La funzione vgg16() permette di importare la struttura della vgg16 e successivamente importare i pesi della rete trainata sul dataset IMAGENET.

```
def vgg16(weights_path=None):
    model = Sequential()
    model.add(ZeroPadding2D((1, 1), input_shape=(224, 224, 3)))
    model.add(Convolution2D(64, (3, 3), activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(128, (3, 3), activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(256, (3, 3), activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(256, (3, 3), activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(256, (3, 3), activation='relu'))
```

```
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, (3, 3), activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, (3, 3), activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, (3, 3), activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, (3, 3), activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1000, activation='softmax'))

    if weights_path:
        model.load_weights(weights_path)

    return model
```

La funzione vgg16_edit() che congela i pesi della vgg16, successivamente cancella alcuni layer e ne aggiunge di nuovi come da richiesta dell'assigment.

```
def vgg16_edit(path):
    model = vgg16(path)

    # Freezing Vgg16 layers
    for layer in model.layers[:37]:
        layer.trainable = False

    # number of element to delete
    number_delete_layer = 5
    for i in range(number_delete_layer):
        model.pop()

    # Add new layers
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4, activation='softmax'))
```

```
    return model
```

La funzione training() compila il modello ed effettua il fit, successivamente plotta i risultati.

```
def training(model, loss, optimizer, train, dev, epoch):

    # Compile model
    model.compile(loss=loss,
                  optimizer=optimizer,
                  metrics=['accuracy'])

    # Start timer
    time_start = datetime.datetime.now()

    # Fit model
    result = model.fit_generator(train, steps_per_epoch=5,
        epochs=epoch, validation_data=dev, validation_steps=5)

    # Stop timer
    time_stop = datetime.datetime.now()

    # Print time
    print("Execution time: ", (time_stop - time_start).
        total_seconds())

    # Show Plot
    show_history(result, 'acc', 'val_acc', 'accuracy', 'epoch',
         'train', 'validation', acc=1)
    show_history(result, 'loss', 'val_loss', 'loss', 'epoch', '
        train', 'validation')
    return model
```

La funzione show_history() una volta impostato il valore degli argomenti plotta un grafico contenente i risultati ottenuti.

```
def show_history(result, measure1='', measure2='', metrics='',
    unit='', set1='', set2='', acc=None):
    plt.plot(result.history[measure1])
    plt.plot(result.history[measure2])
    axes = plt.gca()
    axes.set_xlim([0, epochs])
    axes.set_ylim([0, acc])


    plt.ylabel(metrics)
    plt.xlabel(unit)
    plt.legend([set1, set2], loc='upper left')
    plt.show()
```

La funzione testing() permette di valutare le performance del modello sui dati di testing.

```
def testing(model, test, batch):

    score = model.evaluate_generator(test, batch)

    print(score[0], 'loss')
    print(score[1], 'accuracy')

    return
```

3.4 main.py

Contiene i seguenti elementi:

Le directory dei dati:

```
''' Directory '''
path_train = 'dataset/training_set/'
path_dev = 'dataset/dev_set/'
path_test = 'dataset/test_set/'
path_weights = 'weights/
    vgg16_weights_tf_dim_ordering_tf_kernels.h5'
```

Gli hyper-parametri scelti:

```
''' Hyper-parameter '''
batch_size = 64
epochs = 30
loss = 'categorical_crossentropy'
lr = 0.0005
decay = lr/epochs
adam = Adam(lr=lr, beta_1=0.9, beta_2=0.999, epsilon=None,
    decay=decay)
optimizer = adam
```

Vengono generati i dati di train, dev e test:

```
''' Create Generator for train, dev, test '''
train_generator = data_generator(path_train, shuffle=True)
dev_generator = data_generator(path_dev, shuffle=True)
test_generator = data_generator(path_test, shuffle=False)
```

Vengono stampate le label del dataset:

```
''' Print statistics about data'''
class_dictionary = train_generator.class_indices
print("Class-Index:", class_dictionary)
```

Viene chiamata la funzione vgg16_edit per la creazione del nuovo modello

```
''' Create model based on vgg16 edited'''
model = vgg16_edit(path_weights)
```

Viene chiamata la funzione training() per effettuare il training del modello

```
''' Training model'''
training(model, loss, optimizer, train_generator, dev_generator
    , epochs)
```

Vengono infine valutate le prestazioni del dataset di testing.

```
''' Testing model'''
testing(model, test_generator, batch=100)
```

4 Esperimenti Effettuati

Di seguito sono elencati i due esperimenti effettuati durante questo assignment:

4.1 Hyper-parameters

In entrambi gli esperimenti sono stati utilizzati gli stessi ovvero:

- batch_size = 32
- epochs = 30
- loss = 'categorical_crossentropy'
- lr = 0.00005
- decay = lr/epochs
- adam = Adam(lr=lr, beta_1=0.9, beta_2=0.999, epsilon=None, decay=decay)

4.2 Fine Tuning Modello

Per questioni di computazione vengono effettuate modifiche alla rete solamente sull'ultima parte della rete non toccando i layer convoluzionali.

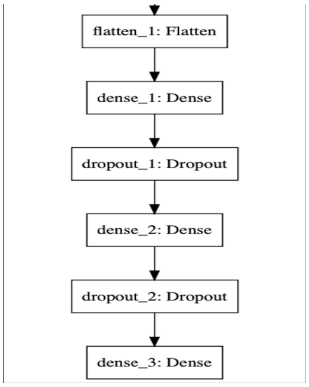
4.2.1 Modello Originale

Di seguito viene mostrato gli ultimi layer del modello vgg16 originale:

Nello specifico le dimensioni dei layer sono le seguenti:

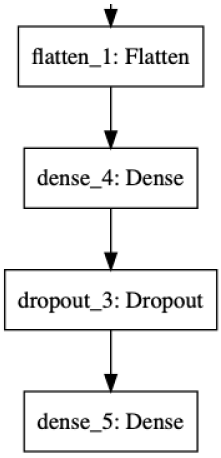
- Flatten()
- Dense(4096, activation='relu')

- Dropout(0.5)
- Dense(4096, activation='relu')
- Dropout(0.5)
- Dense(1000, activation='softmax')



4.2.2 Modello A

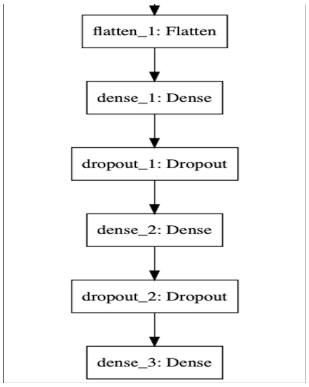
- Vengono Eliminati gli ultimi 5 layer della VGG16 originale
- Vengono frezzati i layer convoluzionali
- Vengono aggiunti 3 layer:
  - Dense(64, activation='relu')
  - Dropout(0.5)
  - Dense(3, activation='softmax')



4.2.3 Modello B

- Viene eliminato l'ultimo layer della VGG16 originale
- Vengono frezzati solamente i layer convoluzionali
- Viene aggiunto il layer finale
  - Dense(3, activation='softmax')

La struttura rimane uguale a quella del modello originale con l'unica differenza nel layer finale.



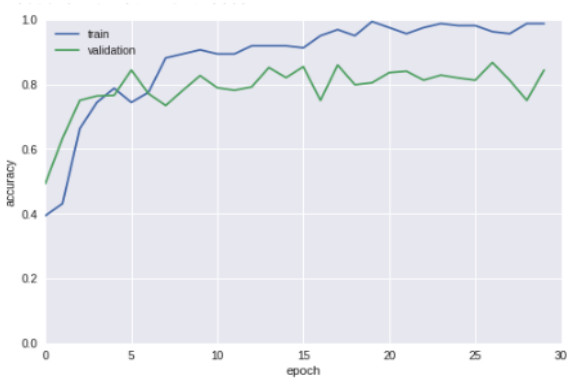
5 Risultati

Come anticipato nella sezione 1, il dataset è composta da un set di immagini di training, di validation e testing.

Di seguito sono elencati i risultati ottenuti per il modello A e B.

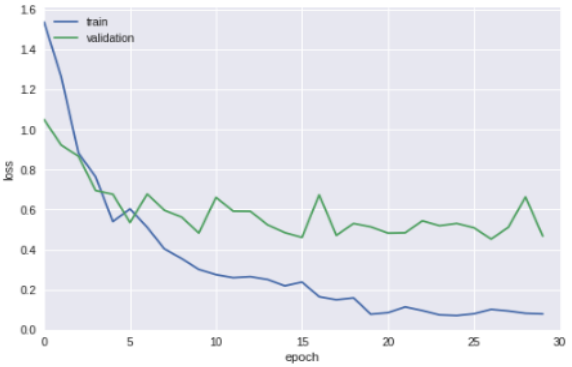
5.1 Modello A

5.1.1 Accuracy on Training e Validation



RESULT		
	Training	Validation
Accuracy	0.9875	0.8438

5.1.2 Loss on Training e Validation



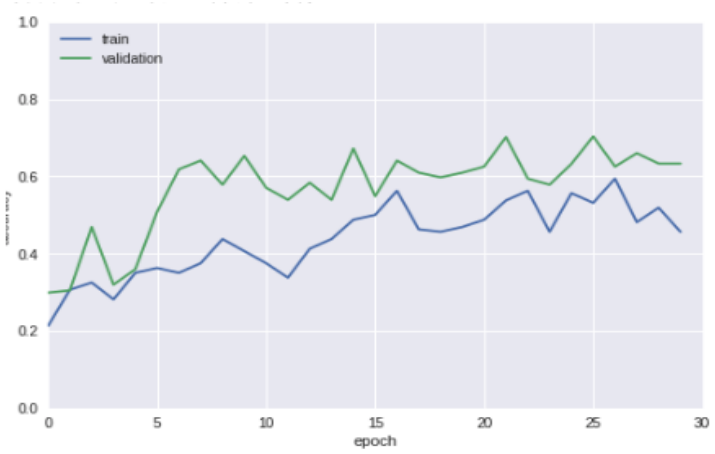
RESULT		
	Training	Validation
Loss	0.0782	0.4673

5.1.3 Accuracy & Loss on Test

RESULT	
	Testing
Loss	0.5598
Accuracy	0.82

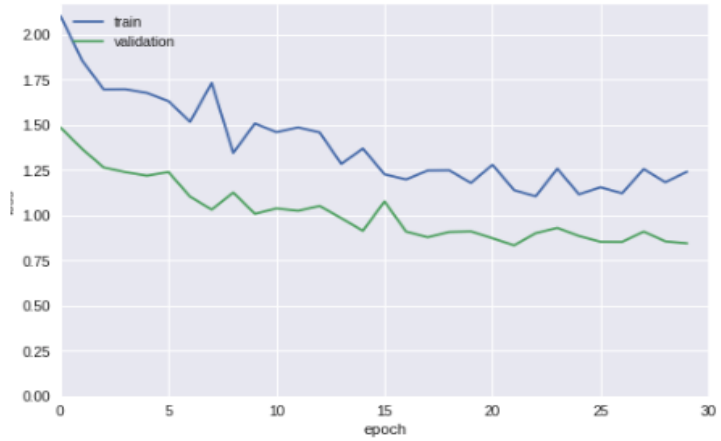
5.2 Modello B

5.2.1 Accuracy on Training e Validation



RESULT		
	Training	Validation
Accuracy	0.4562	0.6328

5.2.2 Loss on Training e Validation



RESULT		
	Training	Validation
Loss	1.2399	0.8439

5.2.3 Accuracy & Loss on Test

RESULT	
	Testing
Loss	0.8814
Accuracy	0.64

6 Conclusioni

Confrontando i due modelli scegliamo il **Modello A**.

Attraverso il Finetuning sfruttando i pesi già trainati della vgg16 riusciamo a creare un classificatore che riesce ad predire la classe corretta di 82 immagini su 100 date in testing.