Assignment 4

Transfer Learning using VGG16 and CIFAR10

Introduzione

Il progetto è stato realizzato su google **colab** in **python**, servendosi delle librerie Keras, sklearn, pandas, mathplotlib (è già presente il comando pip per installarla) e **Pandas**.

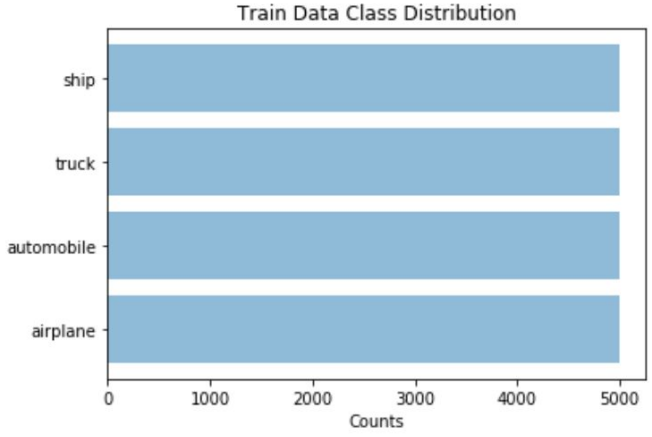
Dataset scelto

Andava scelto un dataset di immagini contenente **dalle 3 alle 10 classi** e ho trovato tramite google "**CIFAR-10**": https://www.cs.toronto.edu/~kriz/cifar.html

Il dataset CIFAR-10 consiste in **60.000** immagini **32x32** a colori divise in **10 classi** con 6000 immagini per classe. Il test set è formato da 10.000 test immagini scelte tra le 60.000.

```
plt.title('Train Data Class Distribution')
plt.show()
```

```
Training data available in x classes
[5000 5000 5000 5000]
```

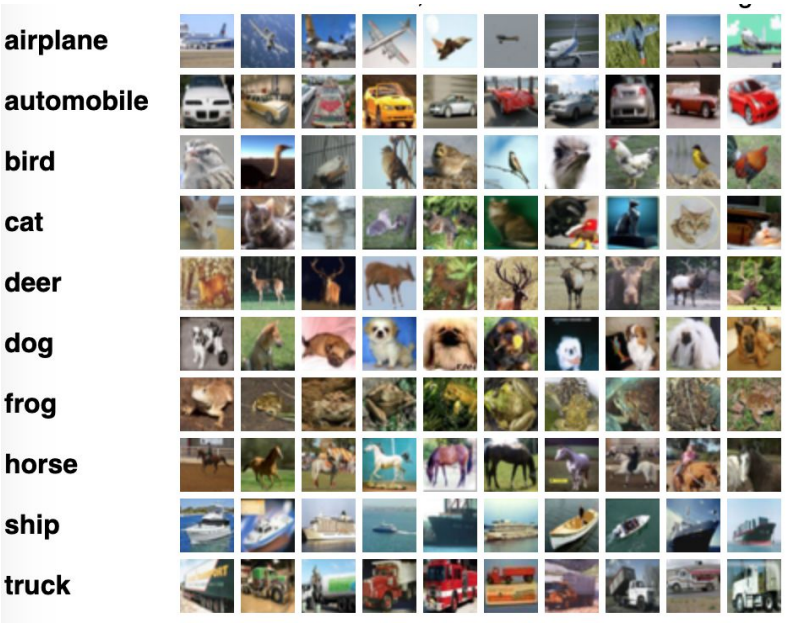


Le classi sono perfettamente bilanciate.

Ho scelto questo dataset poichè rispetta le caratteristiche richieste e contiene immagini 32x32 che sono quindi abbastanza "piccole" per garantire un tempo di computazione non esagerato.

Nuovo task - Selezione delle classi

Le classi originarie del dataset CIFAR10 sono le seguenti



Inizialmente l'assignment è stato svolto cercando di considerare tutte le 10 classi, ovvero tutte le 50.000 immagini presenti nel dataset. Così facendo però la fase di traning della SVM (approfondita dopo) era davvero **troppo complessa per le risorse a disposizione.**

Ho quindi selezionato 4 delle 10 classi di questo dataset, creando un nuovo dataset contenente solo le immagini relative a:

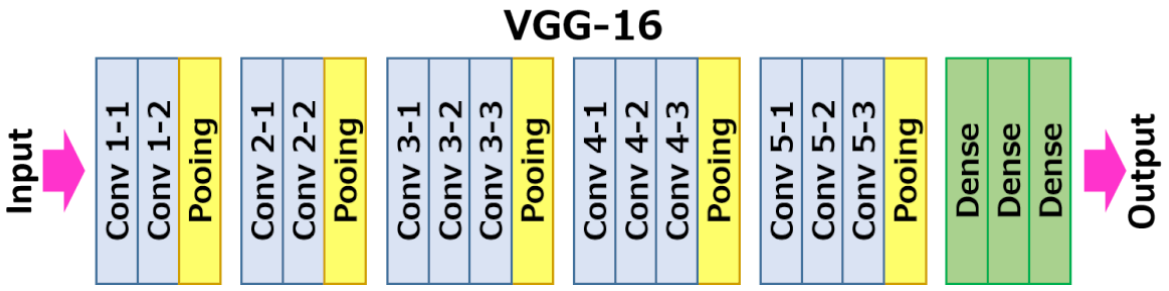
- Aeroplani (0)
- Automobili (1)
- Navi (8)
- Camion (9)

Da notare che la scelta è stata volutamente fatta all'interno della categoria "**mezzi di trasporto**". Infatti se avessi messo 2 classi di animali e 2 classi di mezzi di trasporto probabilmente il modello sarebbe stato "facilitato" nel distinguere queste categorie.

Il nuovo dataset quindi è costituito da 20.000 immagini di training (meno della metà di quello di partenza) **e 4.000 immagini di test**:

```
› Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
  170500096/170498071 [==============================] - 2s 0us/step

  X Train shape: (20000, 32, 32, 3)

  Y Train shape: (20000, 1)

  X Test shape: (4000, 32, 32, 3)

  Y Test shape: (4000, 1)
```

VGG16 - Importazione e layer di taglio



La VGG-16 è stata importata tramite le funzioni già predisposte da keras escludendo i layer dense finali dell'immagine qui sopra.

```
model = VGG16(weights="imagenet", include_top=False, input_shape=(32,32,3))

model.summary()

model2 = Model(model.input, model.layers[-5].output)
model2.summary()

model3 = Model(model2.input, model2.layers[-5].output)
model3.summary()

model4 = Model(model3.input, model3.layers[-5].output)
model4.summary()
```

Ho quindi ricavato 4 modelli, così composti:

- Model1: dall'input layer fino a **block5_pool** (prima dei dense) con **512** parametri
- Model2: dall'input layer fino a **block4_pool** con **2048** parametri
- Model3: dall'input layer fino a **block3_pool** con **4096** parametri
- Model4: dall'input layer fino a **block2_pool** con **8192** parametri

All'interno del source code sono presenti ulteriori dettagli riguardanti i layer (visualizzati tramite la funzione summary)

Feature extraction sui dati di training e di test

E' stata poi eseguita la funzione **predict** sui dati di **test** e di **train** per estrarre, da ogni immagine, **le feature** che i modelli precedenti hanno ricavato dalle stesse.

```
print("Predicting on model 1")
train_features_model1 = model.predict(x_train)
print("Predicting on model 2")
train_features_model2 = model2.predict(x_train)
print("Predicting on model 3")
train_features_model3 = model3.predict(x_train)
print("Predicting on model 4")
train_features_model4 = model4.predict(x_train)
```

```
print("Predicting on model 1")
test_features_model1 = model.predict(x_test)
print("Predicting on model 2")
test_features_model2 = model2.predict(x_test)
print("Predicting on model 3")
test_features_model3 = model3.predict(x_test)
print("Predicting on model 4")
test_features_model4 = model4.predict(x_test)
print("END")
```

Successivamente **le matrici tridimensionali contenenti le feature estratte sono state "appiattite"** ottenendo così le seguenti shape, corrispondenti al modello1, modello2, modello3, modello4 sopradescritti.

```
(20000, 1, 1, 512)
(20000, 2, 2, 512)
(20000, 4, 4, 256)
(20000, 8, 8, 128)
---
(20000, 512)
(20000, 2048)
(20000, 4096)
(20000, 8192)
```

Original Shape

After flattering

E' possibile notare come **le feature aumentino avvicinandosi al layer di input** e diventino quindi più complesse da elaborare nei passaggi successivi (durante la SVM)

Traning della SVM e prediction sui dati di test

Per questo progetto è stato richiesto di utilizzare un **classificatore lineare**, ho quindi optato per una **SVM**.

Quindi, il passaggio successivo, è stato quello di **addestrare la SVM con le feature estratte dalla VGG-16** tagliata nei 4 punti**.** Il processo è stato ripetuto sui 4 modelli visti prima per poter valutare quale taglio della VGG-16 fornisse feature "migliori" per poter effettuare la classificazione.

```
print("SVM su model 1 in corso...")
svm_model_linear1 = SVC(kernel = 'linear', C = 1, verbose=True).fit(train_features_model1_flatten, y_train)
print("\nSVM su model 1 terminato\nPredizione sui dati di test...\n")
y_pred1 = svm_model_linear1.predict(test_features_model1_flatten)
acc1=accuracy_score(y_test, y_pred1)
print(classification_report(y_test, y_pred1, target_names=image_classess, digits=3))
```

Dopo il **fit** della SVM sui dati di **training** è stato realizzata direttamente la **predizione** sui dati di **test** con una SVM, stampando a video le performance dello stesso.
La SVM è stata lasciata con i parametri di default, specificando l'utilizzo di un kernel lineare.

Riporto i risultati del test nella pagina successiva:

Performance sui dati di test

Performance del **modello 1: accuracy 0,75**

```
SVM su model 1 in corso...
[LibSVM]
SVM su model 1 terminato
Predizione sui dati di test...

              precision    recall  f1-score   support

    airplane       0.760     0.775     0.767      1000
  automobile       0.735     0.743     0.739      1000
       truck       0.754     0.740     0.747      1000
        ship       0.733     0.723     0.728      1000

    accuracy                           0.745      4000
   macro avg       0.745     0.745     0.745      4000
weighted avg       0.745     0.745     0.745      4000
```

Performance del **modello 2: accuracy 0,81**

```
SVM su model 2 in corso...
[LibSVM]
SVM su model 2 terminato
Predizione sui dati di test...

              precision    recall  f1-score   support

    airplane       0.787     0.867     0.825      1000
  automobile       0.807     0.806     0.806      1000
       truck       0.859     0.826     0.842      1000
        ship       0.815     0.764     0.789      1000

    accuracy                           0.816      4000
   macro avg       0.817     0.816     0.816      4000
weighted avg       0.817     0.816     0.816      4000
```

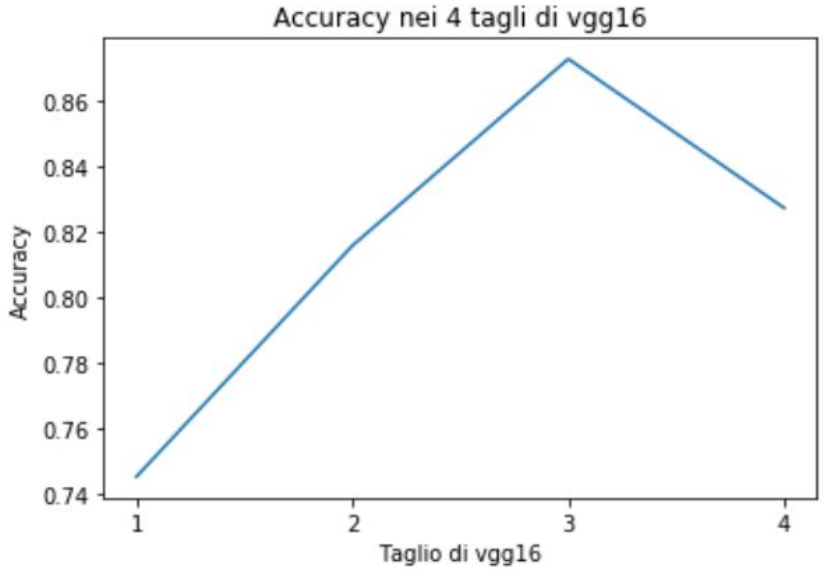
Performance del **modello 3: accuracy 0,87**

```
SVM su model 3 in corso...
[LibSVM]
SVM su model 3 terminato
Predizione sui dati di test...

              precision    recall  f1-score   support

    airplane       0.860     0.894     0.877      1000
  automobile       0.866     0.863     0.864      1000
       truck       0.892     0.883     0.887      1000
        ship       0.874     0.851     0.862      1000

    accuracy                           0.873      4000
   macro avg       0.873     0.873     0.873      4000
weighted avg       0.873     0.873     0.873      4000
```

Performance del **modello 4**: **accuracy 0,82**

```
SVM su model 4 in corso...
[LibSVM]
SVM su model 3 terminato
Predizione sui dati di test...

              precision    recall  f1-score   support

    airplane       0.798     0.860     0.828      1000
  automobile       0.826     0.826     0.826      1000
       truck       0.848     0.802     0.824      1000
        ship       0.841     0.821     0.831      1000

    accuracy                           0.827      4000
   macro avg       0.828     0.827     0.827      4000
weighted avg       0.828     0.827     0.827      4000
```

Riportando in un grafico l'andamento dell'accuracy ottengo questo risultato



Si nota facilmente come il **terzo** taglio della VGG-16 produca le **performance migliori** con la SVM applicata successivamente.

Come mi aspettavo **le performance migliori non si ottengono ne con tagli vicini all'input** (dove le feature sono tante e poco specifiche) **ne vicino all'output** (dove le feature sono più specifiche)

Viene ora riportata una breve analisi visiva sui risultati ottenuti

Analisi visiva sui dati di Test

“PRED: X” dove X si riferisce a cosa il modello ha predetto per l’immagine corrispettiva tra le seguenti categorie:

- **0** - Aeroplani
- **1**- Automobili
- **8** - Navi
- **9** - Camion