StandfordCar Car Classifier

Esame: Deep Learning and Generative Models

Alberto Coffrini - LM Scienze Informatiche **- UniPR**

Car Classifier

Stanford Car dataset :

URL: https://www.kaggle.com/datasets/jutrera/stanford-car-dataset-by-classes-folder

- 196 classes.
  - Train: 8144 images,
    - avg: 41.5 images per class.
  - Test: 8041 images,
    - avg: 41.0 images per class.

> **Ogni foto corrisponde a una label contenente: Marca, Modello, Anno Prod.**
>
> - > train -> 2012 Tesla Model S
> - > 2012 BMW M3 coupe
>
>   …
> - > test -> 2012 Tesla Model S
> - > 2012 BMW M3 coupe
>
>   …

Utilizzo di modelli Pytorch: **Resnet 18, Resnet34, VGG19**

Ogni modello è pretrainato, con rimpiazzo dell'ultimo FCLayer con un layer per le 196 classi.

Codice eseguito su Kaggle con GPU P100 attiva.

Vengono allegati al report:

- l'ultima versione del codice utilizzato con varie parti commentate dato che sono state utilizzate diverse configurazioni.
- modelli scaricabili dal link sulla pagina github del progetto ottenuti durante i test effettuati.

Import

Import delle librerie necessarie.

Alcune parti di codice nel file python avranno altri import, in qualche caso ripetuti, questo perchè sono parti di codice che a volte ho eseguito singolarmente per ottenere dati, informazioni o esecuzioni singole del modello.

```
import matplotlib.pyplot as plt
import numpy as np

import time
import os
import PIL.Image as Image
from IPython.display import display

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.models as models
import torchvision.transforms as transform
```

Calcolo Mean e Std del dataset

Calcolo della media e dev.std delle immagini del dataset.

Così da utilizzarle poi in seguito.

```
# Directory
# OPTION TODO: CHANGE IT ON OTHER DEVICE
dataset_dir = "/kaggle/input/stanford-car-dataset-by-classes-folder/car_data/car_data/"

transform_for_stats = transforms.Compose([
    transforms.Resize((300, 300)),
    transforms.ToTensor()
])

dataset_for_stats = ImageFolder(root=dataset_dir + "train", transform=transform_for_stats)
data_loader_for_stats = torch.utils.data.DataLoader(dataset_for_stats, batch_size=32, shuffle=False, num_workers=2)

# mean e std
mean = 0.0
std = 0.0
total_samples = 0

for images, _ in data_loader_for_stats:
    batch_size = images.size(0)
    images = images.view(batch_size, images.size(1), -1)
    mean += images.mean(2).sum(0)
    std += images.std(2).sum(0)
    total_samples += batch_size

mean /= total_samples
std /= total_samples

print("Mean:", mean)
print("Std:", std)
```

Il risultato è :

```
Mean: tensor([0.4708, 0.4602, 0.4550])
Std: tensor([0.2612, 0.2603, 0.2653])
```

Dataset e Transform

```
#dataset URL on Kaggle.
# OPTION TODO: CHANGE IT ON OTHER DEVICE
dataset_dir = "/kaggle/input/stanford-car-dataset-by-classes-folder/car_data/car_data/"

#Transform
transformation_train = transforms.Compose(
                               [transforms.Resize((300, 300)),
                                transforms.RandomHorizontalFlip(),
                                transforms.RandomRotation(15),
                                transforms.ToTensor(),
                                transforms.Normalize((0.4708, 0.4602, 0.4550), (0.2612, 0.2603, 0.2653))])
transformation_test = transforms.Compose([transforms.Resize((300, 300)),
                               transforms.ToTensor(),
                               transforms.Normalize((0.4708, 0.4602, 0.4550), (0.2612, 0.2603, 0.2653))])

#dataset for training and testing phase
dataset_train = torchvision.datasets.ImageFolder(root=dataset_dir+"train", transform = transformation_train)
trainloader = torch.utils.data.DataLoader(dataset_train, batch_size = 32, shuffle=True, num_workers = 2)

dataset_test = torchvision.datasets.ImageFolder(root=dataset_dir+"test", transform = transformation_test)
testloader = torch.utils.data.DataLoader(dataset_test, batch_size = 32, shuffle=False, num_workers = 2)
```

Funzione training

Definizione della funzione di training per il modello scelto e passato alla funzione.

Il numero di epoche di default è 5 ma ne sono state utilizzate il doppio per ogni test effettuato.

Alla fine di ogni epoca eseguo il print dei dati ottenuti .

```
def train_model(model, criterion, optimizer, scheduler, n_epochs = 5):
    #keep in memory data analysis
    losses = []
    accuracies = []
    test_accuracies = []

    # model in train mode
    model.train()

    for epoch in range(n_epochs):
        since = time.time()
        running_loss = 0.0
        running_correct = 0.0
        for i, data in enumerate(trainloader, 0):

            # obtain the inputs
            inputs, labels = data
            # assign inputs to cuda
            inputs = inputs.to(device)
            labels = labels.to(device)
            # optimizer
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = model(inputs)
```

```
            _, predicted = torch.max(outputs.data, 1)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            running_correct += (labels==predicted).sum().item()

        # data each epoch
        epoch_duration = time.time()-since
        epoch_loss = running_loss/len(trainloader)
        epoch_acc = 100/32*running_correct/len(trainloader)
        print("Epoch %s, duration: %d s, loss: %.4f, acc: %.4f" % (epoch+1, epoch_duration, epoch_loss, epoch_acc)) #printing on terminal

        losses.append(epoch_loss)
        accuracies.append(epoch_acc)

        # model in eval mode
        model.eval()
        test_acc = eval_model(model)
        test_accuracies.append(test_acc)

        # model in train mode
        model.train()
        scheduler.step(test_acc)
        since = time.time()

    print('Finished Training') #ending of training phase

    return model, losses, accuracies, test_accuracies
```

Funzione valutazione su Test Set

Definizione della funzione di valutazione del modello sul test set.

```
def eval_model(model):
    correct = 0.0
    total = 0.0
    with torch.no_grad():
        for i, data in enumerate(testloader, 0):
            images, labels = data
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)

            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    test_acc = 100.0 * correct / total
    print('Accuracy of the network on the test images: %d %%' % (test_acc))
    return test_acc
```

Modello pytorch

Scelta del modello da utilizzare. Come sottolineato in precedenza vengono testati principalmente tre modelli:

Resnet18, Resnet34, VGG19

Utilizzo di CrossEntropyLoss e discesa del gradiente

```
# pytorch model here
#model_ft = models.resnet18(pretrained=True)
model_ft = models.resnet34(pretrained=True)
num_ftrs = model_ft.fc.in_features

# last fc layer replaced qwith an untrained one
model_ft.fc = nn.Linear(num_ftrs, 196) #(with grads's data)
model_ft = model_ft.to(device)

#  CrossEntropyLoss and stochastic gradient descent
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model_ft.parameters(), lr=0.01, momentum=0.9)

lrscheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='max', patience=3, threshold = 0.9)
```

Una volta eseguito il training viene salvato il modello in output

```
#save model on output file
#OPTION TODO: change name for different model
torch.save(model_ft.state_dict(), "./trainedModelFinal_NameModel.pt")
```

Plotting

Plotting di accuracy su test, training set e loss delle singole epoche

```
# Plotting

f, axarr = plt.subplots(2,2, figsize = (12, 8))

axarr[0, 0].plot(training_losses)
axarr[0, 0].set_title("Training loss")

axarr[0, 1].plot(training_accs)
axarr[0, 1].set_title("Training acc")

axarr[1, 0].plot(test_accs)
axarr[1, 0].set_title("Test acc")
```

RISULTATI

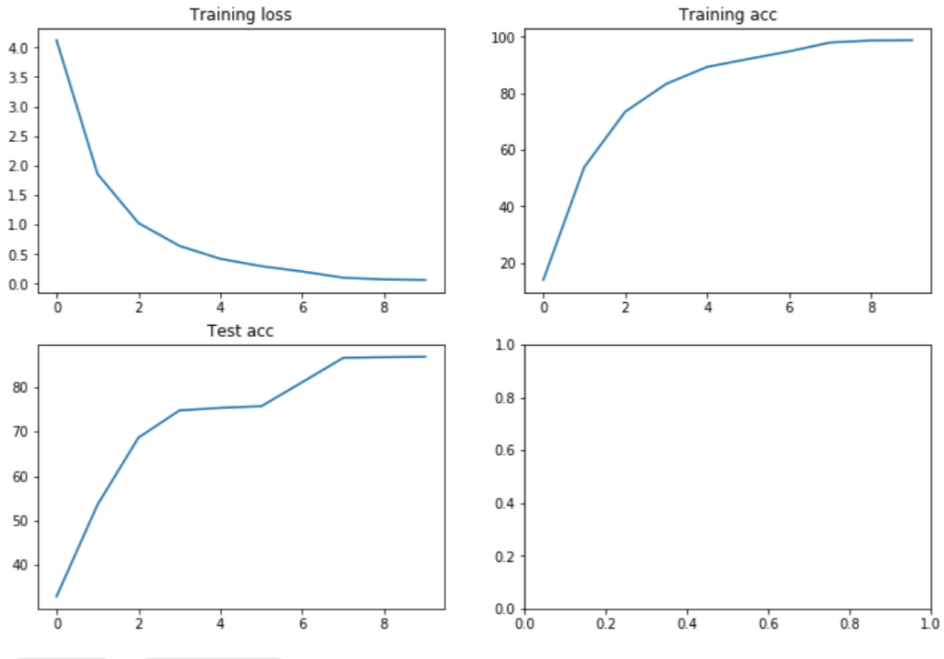
Resnet 18

```
# pytorch model here
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features

# last fc layer replaced qwith an untrained one
model_ft.fc = nn.Linear(num_ftrs, 196) #(with grads)
model_ft = model_ft.to(device)

#  CrossEntropyLoss and stochastic gradient descent
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model_ft.parameters(), lr=0.01, momentum=0.9)
```

```
lrscheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='max', patience=3, threshold = 0.9)
```

```
#Training con:
#Resnet18
#train_tfms = transforms.Compose([transforms.Resize((300, 300)),
#                                  transforms.RandomHorizontalFlip(),
#                                  transforms.RandomRotation(15),
#                                  transforms.ToTensor(),
#                                  transforms.Normalize((0.4708, 0.4602, 0.4550), (0.2612, 0.2603, 0.2653))])

Training
Epoch number 1, duration: 110 s, loss: 4.1222, acc: 13.8113
Accuracy on the test images: 32 %
Epoch number 2, duration: 110 s, loss: 1.8596, acc: 53.8235
Accuracy on the test images: 53 %
Epoch number 3, duration: 109 s, loss: 1.0282, acc: 73.4804
Accuracy on the test images: 68 %
Epoch number 4, duration: 109 s, loss: 0.6419, acc: 83.3578
Accuracy on the test images: 74 %
Epoch number 5, duration: 109 s, loss: 0.4231, acc: 89.3750
Accuracy on the test images: 75 %
Epoch number 6, duration: 111 s, loss: 0.2986, acc: 92.1569
Accuracy on the test images: 75 %
Epoch number 7, duration: 109 s, loss: 0.2080, acc: 94.8652
Accuracy on the test images: 81 %
Epoch number 8, duration: 107 s, loss: 0.1035, acc: 98.0147
Accuracy on the test images: 86 %
Epoch number 9, duration: 109 s, loss: 0.0743, acc: 98.7255
Accuracy on the test images: 86 %
Epoch number 10, duration: 107 s, loss: 0.0650, acc: 98.8358
Accuracy on the test images: 86 %
Finished Training
```

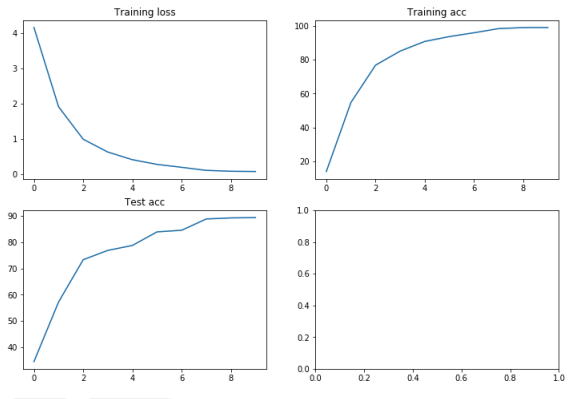
```
Text(0.5, 1.0, 'Test acc')
```



```
# Training con:
# Resnet18
# train_tfms = transforms.Compose([transforms.Resize((400, 400)),
#                                  transforms.RandomHorizontalFlip(),
#                                  transforms.RandomRotation(15),
#                                  transforms.ToTensor(),
#                                  transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])


Training
Epoch 1, duration: 145 s, loss: 4.1637, acc: 14.0686
Accuracy of the network on the test images: 34 %
Epoch 2, duration: 141 s, loss: 1.9088, acc: 54.7794
Accuracy of the network on the test images: 57 %
Epoch 3, duration: 140 s, loss: 0.9851, acc: 76.7525
Accuracy of the network on the test images: 73 %
Epoch 4, duration: 140 s, loss: 0.6203, acc: 85.0613
Accuracy of the network on the test images: 76 %
Epoch 5, duration: 141 s, loss: 0.4020, acc: 90.7475
Accuracy of the network on the test images: 78 %
Epoch 6, duration: 141 s, loss: 0.2685, acc: 93.6152
Accuracy of the network on the test images: 83 %
Epoch 7, duration: 139 s, loss: 0.1853, acc: 95.8578
Accuracy of the network on the test images: 84 %
Epoch 8, duration: 144 s, loss: 0.0979, acc: 98.3088
Accuracy of the network on the test images: 88 %
Epoch 9, duration: 142 s, loss: 0.0732, acc: 98.9093
Accuracy of the network on the test images: 89 %
Epoch 10, duration: 143 s, loss: 0.0662, acc: 98.9583
Accuracy of the network on the test images: 89 %
Finished Training
```

```
# plot the stats

f, axarr = plt.subplots(2,2, figsize = (12, 8))
axarr[0, 0].plot(training_losses)
axarr[0, 0].set_title("Training loss")
axarr[0, 1].plot(training_accs)
axarr[0, 1].set_title("Training acc")
axarr[1, 0].plot(test_accs)

axarr[1, 0].set_title("Test acc")
```

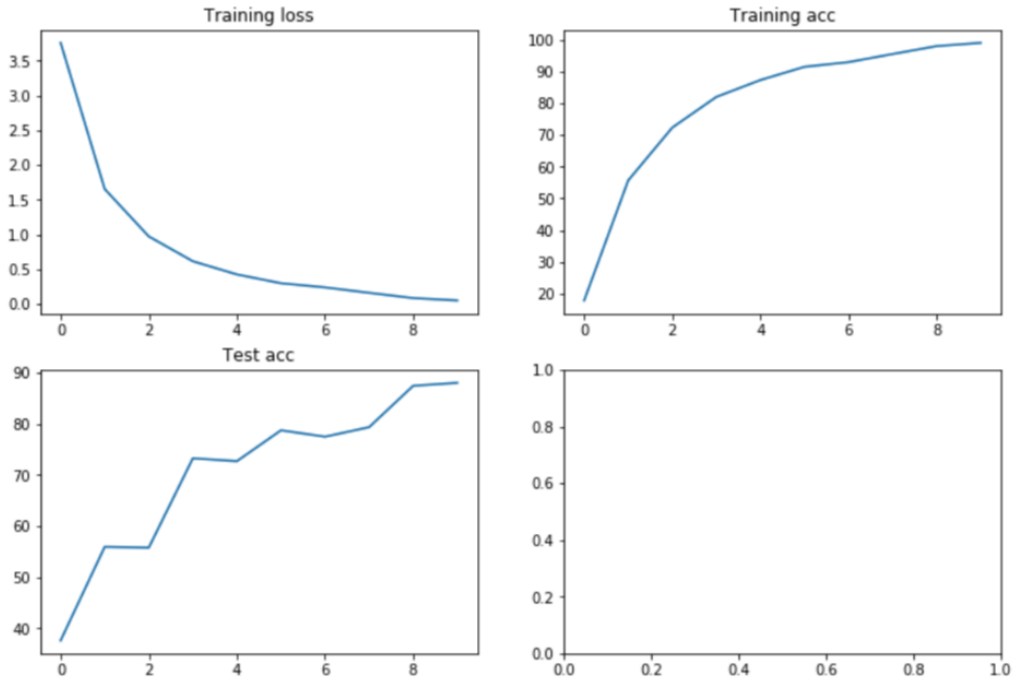
```
Text(0.5, 1.0, 'Test acc')
```



Resnet34

```
# Training con:
# Resnet34
# transformation_train = transforms.Compose(
#                               [transforms.Resize((300, 300)),
#                                transforms.RandomHorizontalFlip(),
#                                transforms.RandomRotation(15),
#                                transforms.ToTensor(),
#                                transforms.Normalize((0.4708, 0.4602, 0.4550), (0.2612, 0.2603, 0.2653))])

Training
Epoch number 1, duration: 117 s, loss: 3.7584, acc: 17.8922
Accuracy on the test images: 37 %
Epoch number 2, duration: 118 s, loss: 1.6535, acc: 55.6618
Accuracy on the test images: 55 %
Epoch number 3, duration: 119 s, loss: 0.9710, acc: 72.2794
Accuracy on the test images: 55 %
Epoch number 4, duration: 118 s, loss: 0.6135, acc: 81.9363
Accuracy on the test images: 73 %
Epoch number 5, duration: 117 s, loss: 0.4237, acc: 87.2426
Accuracy on the test images: 72 %
Epoch number 6, duration: 117 s, loss: 0.2970, acc: 91.4093
Accuracy on the test images: 78 %
Epoch number 7, duration: 119 s, loss: 0.2372, acc: 92.8799
Accuracy on the test images: 77 %
Epoch number 8, duration: 125 s, loss: 0.1589, acc: 95.4289
Accuracy on the test images: 79 %
Epoch number 9, duration: 121 s, loss: 0.0831, acc: 97.9289
Accuracy on the test images: 87 %
Epoch number 10, duration: 118 s, loss: 0.0493, acc: 98.9461
Accuracy on the test images: 88 %
Finished Training
```

```
Text(0.5, 1.0, 'Test acc')
```

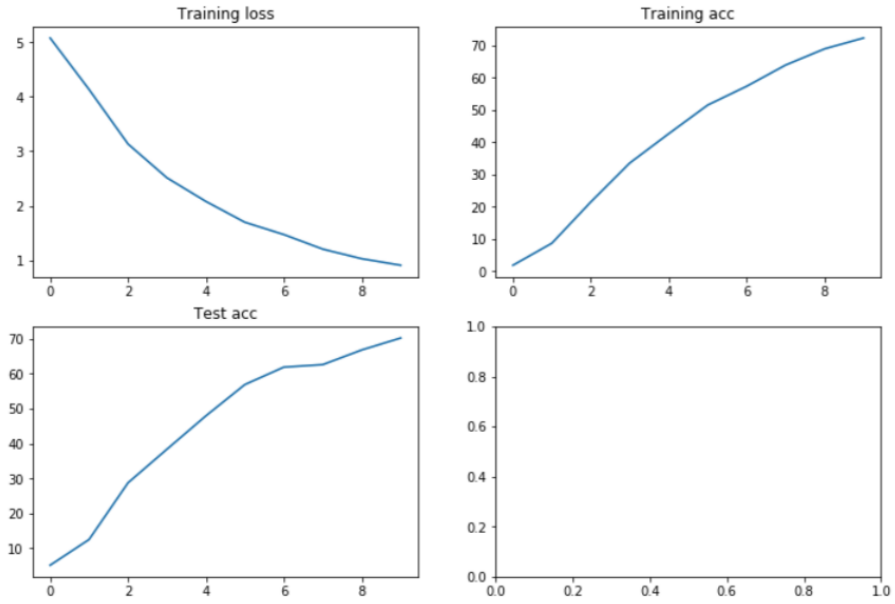


VGG19

```
# Load the pre-trained VGG19 model
model_ft = models.vgg19(pretrained=True)

# Modify the classifier (fc) layer for your specific task
# Replace the last fully connected layer with an untrained one (requires grad by default)
num_ftrs = model_ft.classifier[-1].in_features
model_ft.classifier[-1] = nn.Linear(num_ftrs, 196)  # 196 is the number of output classes

# Send the model to the device (GPU or CPU)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft = model_ft.to(device)

# Define your loss function
criterion = nn.CrossEntropyLoss()

# Define the optimizer
optimizer = optim.SGD(model_ft.parameters(), lr=0.01, momentum=0.9)

# Define a learning rate scheduler (ReduceLROnPlateau in this case)
lrscheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='max', patience=3, threshold=0.9)
```

```
# Training con:
# VGG19
# transformation_train = transforms.Compose([transforms.Resize((300, 300)),
#                                transforms.RandomHorizontalFlip(),
#                                transforms.RandomRotation(15),
#                                transforms.ToTensor(),
#                                transforms.Normalize((0.4708, 0.4602, 0.4550), (0.2612, 0.2603, 0.2653))])


Training
Epoch 1, duration: 206 s, loss: 5.0783, acc: 1.9118
Accuracy on the test images: 5 %
Epoch 2, duration: 204 s, loss: 4.1333, acc: 8.7377
Accuracy on the test images: 12 %
Epoch 3, duration: 205 s, loss: 3.1333, acc: 21.5196
Accuracy on the test images: 28 %
Epoch 4, duration: 204 s, loss: 2.5122, acc: 33.5784
Accuracy on the test images: 38 %
Epoch 5, duration: 205 s, loss: 2.0793, acc: 42.6471
Accuracy on the test images: 47 %
Epoch 6, duration: 204 s, loss: 1.6972, acc: 51.5074
Accuracy on the test images: 56 %
Epoch 7, duration: 205 s, loss: 1.4721, acc: 57.3284
Accuracy on the test images: 61 %
Epoch 8, duration: 205 s, loss: 1.2046, acc: 63.8971
Accuracy on the test images: 62 %
Epoch 9, duration: 204 s, loss: 1.0286, acc: 68.9093
Accuracy on the test images: 66 %
Epoch 10, duration: 204 s, loss: 0.9094, acc: 72.2672
Accuracy on the test images: 70 %
Finished Training
```

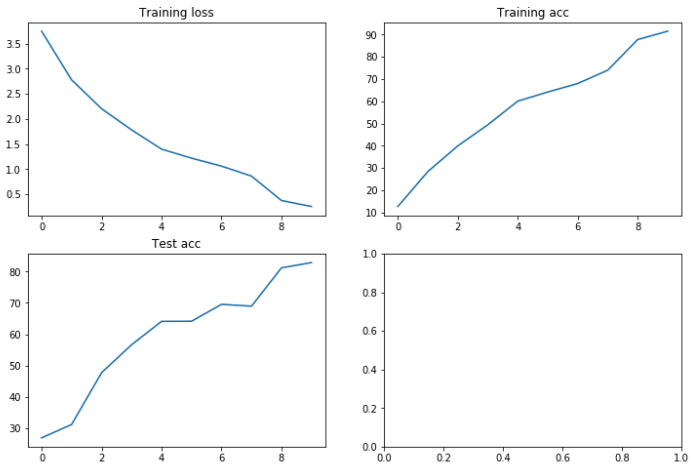
```
Text(0.5, 1.0, 'Test acc')
```




```
# Training con:
# VGG19
# transformation_train = transforms.Compose([transforms.Resize((400, 400)),
#                                transforms.RandomHorizontalFlip(),
#                                transforms.RandomRotation(15),
#                                transforms.ToTensor(),
#                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])


Training
Epoch 1, duration: 315 s, loss: 3.7442, acc: 12.6961
Accuracy of the network on the test images: 26 %
Epoch 2, duration: 316 s, loss: 2.7768, acc: 28.3946
Accuracy of the network on the test images: 31 %
Epoch 3, duration: 316 s, loss: 2.2032, acc: 39.9510
Accuracy of the network on the test images: 47 %
Epoch 4, duration: 316 s, loss: 1.7825, acc: 49.4240
Accuracy of the network on the test images: 56 %
Epoch 5, duration: 315 s, loss: 1.3998, acc: 60.0735
Accuracy of the network on the test images: 64 %
Epoch 6, duration: 316 s, loss: 1.2202, acc: 64.1667
Accuracy of the network on the test images: 64 %
Epoch 7, duration: 316 s, loss: 1.0618, acc: 68.0025
Accuracy of the network on the test images: 69 %
Epoch 8, duration: 316 s, loss: 0.8636, acc: 73.9828
Accuracy of the network on the test images: 68 %
Epoch 9, duration: 316 s, loss: 0.3777, acc: 87.7696
Accuracy of the network on the test images: 81 %
Epoch 10, duration: 316 s, loss: 0.2597, acc: 91.5196
Accuracy of the network on the test images: 82 %
Finished Training
```

```
# plot the stats

f, axarr = plt.subplots(2,2, figsize = (12, 8))
axarr[0, 0].plot(training_losses)
axarr[0, 0].set_title("Training loss")
axarr[0, 1].plot(training_accs)
axarr[0, 1].set_title("Training acc")
axarr[1, 0].plot(test_accs)

axarr[1, 0].set_title("Test acc")
```

```
Text(0.5, 1.0, 'Test acc')
```



Valutazione con modello di una singola immagine presa dal TestSet

Dopo aver salvato i vari modelli possono essere caricati e utilizzati singolarmente come se fossimo in un ambiente di produzione. passando una immagine scelta dal testset e ottenere il risultato del classificatore

```
# tie the class indices to their names
def find_classes(dir):
    classes = os.listdir(dir)
    classes.sort()
    class_to_idx = {classes[i]: i for i in range(len(classes))}
    return classes, class_to_idx

classes, c_to_idx = find_classes(dataset_dir+"train")
```

```
# modello con caratteristiche a quello che si prende in input
modelUp = models.resnet18(pretrained=True)
num_ftrs = modelUp.fc.in_features
modelUp.fc = nn.Linear(num_ftrs, 196)
modelUp = modelUp.to(device)

# modello in input
modelUp.load_state_dict(torch.load('/kaggle/input/resnet37/trainedModelFinalResnet18.pt'))
```

```
# test the model on random images

# switch the model to evaluation mode to make dropout and batch norm work in eval mode
modelUp.eval()

# Registra il tempo di inizio
start_time = time.time()

# transforms for the input image
loader = transforms.Compose([transforms.Resize((400, 400)),
                             transforms.ToTensor(),
                             transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
image = Image.open(dataset_dir+"test/BMW 1 Series Convertible 2012/01795.jpg")
image = loader(image).float()
image = torch.autograd.Variable(image, requires_grad=True)
image = image.unsqueeze(0)
image = image.cuda()
output = modelUp(image)
conf, predicted = torch.max(output.data, 1)

# get the class name of the prediction
display(Image.open(dataset_dir+"test/BMW 1 Series Convertible 2012/01795.jpg"))
print(classes[predicted.item()], "\nconfidence: ", conf.item())

end_time = time.time()
elapsed_time = end_time - start_time

print(f"Time: {elapsed_time} seconds")

eval_model(modelUp)
```

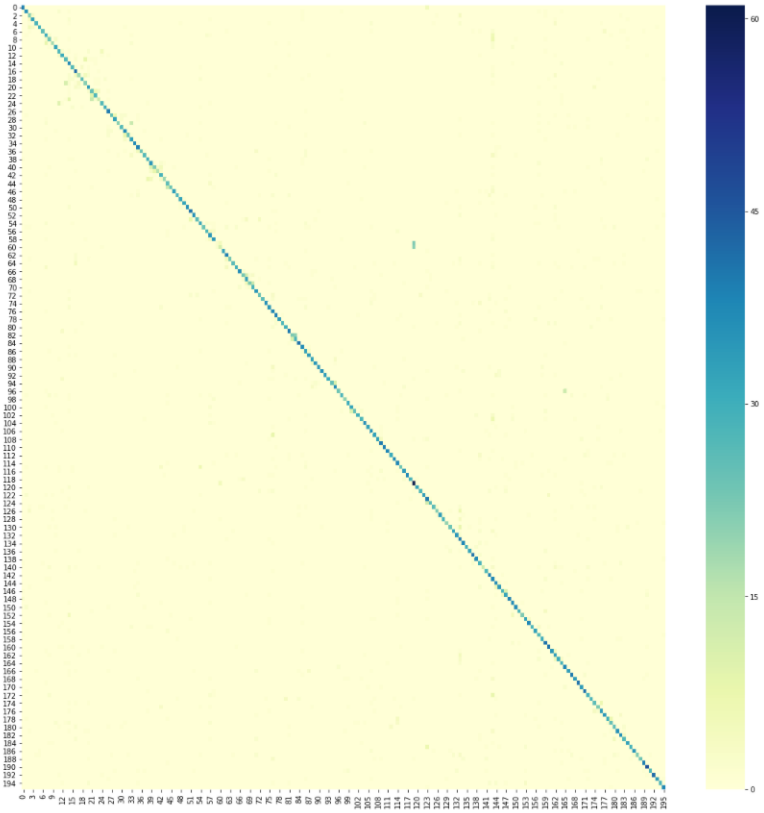
```
BMW 1 Series Convertible 2012
confidence:  14.405200958251953
Time: 0.19545388221740723 seconds
```

Matrice di confusione - Heatmap

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

```
# n classi
nb_classes = 196

# matrice di confusione
confusion_matrix = np.zeros((nb_classes, nb_classes))

#  figura
plt.figure(figsize=(20, 20))
```

```
with torch.no_grad():
    for inputs, classes in testloader:
        inputs = inputs.to(device)
        classes = classes.to(device)
        outputs = modelUp(inputs)
        _, preds = torch.max(outputs, 1)
        for t, p in zip(classes.view(-1), preds.view(-1)):
            confusion_matrix[t.long(), p.long()] += 1

# dataframe partendo da matrice
df_cm = pd.DataFrame(confusion_matrix)

# heatmap
sns.heatmap(df_cm, annot=True, fmt="d", cmap="YlGnBu")

# asse X-Y
plt.yticks(rotation=0, fontsize=15)
plt.ylabel('True label')

plt.xticks(rotation=45, fontsize=15)
plt.xlabel('Predicted label')


# file immagine
#plt.savefig("heatmap.png")

plt.show()
```



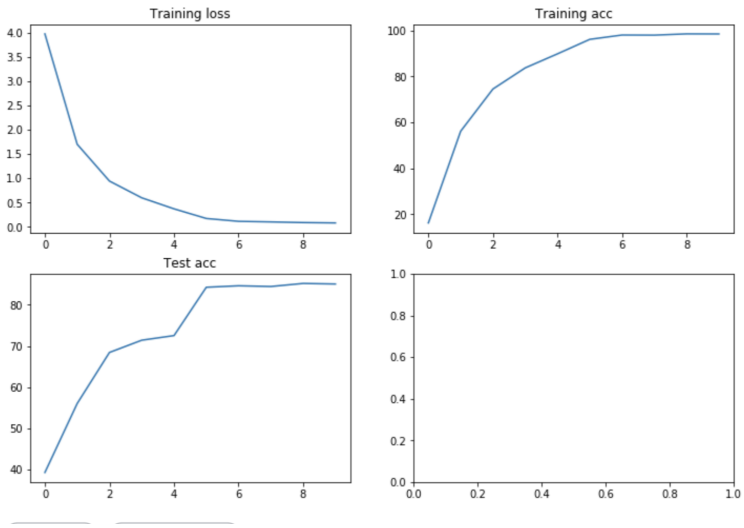
Sembra esserci buona capacità di classificazione notando la forte presenza di valori alti sulla diagonale. Tuttavia si vede una particolare frequenza di errore intorno alle classi con enumerazione 58-60.

Augmentation tests

Vengono eseguiti questi due test per cercare di migliorare ancora le performance sul test set con lo stesso numero di epoche per il training. Utilizzando diverse tecniche di augmentation.

```
# Training con:
# Resnet18
# transformation_train = transforms.Compose(
#                                [transforms.Resize((300, 300)),
#                                 transforms.RandomRotation(15),
#                                 transforms.RandomCrop((250, 250)),
#                                 transforms.ToTensor(),
#                                 transforms.Normalize((0.4708, 0.4602, 0.4550), (0.2612, 0.2603, 0.2653))])

Training
Epoch number 1, duration: 113 s, loss: 3.9810, acc: 16.1520
Accuracy on the test images: 39 %
Epoch number 2, duration: 99 s, loss: 1.7035, acc: 56.1520
Accuracy on the test images: 55 %
Epoch number 3, duration: 98 s, loss: 0.9439, acc: 74.5711
Accuracy on the test images: 68 %
Epoch number 4, duration: 96 s, loss: 0.5983, acc: 83.7868
Accuracy on the test images: 71 %
Epoch number 5, duration: 101 s, loss: 0.3704, acc: 89.9142
Accuracy on the test images: 72 %
Epoch number 6, duration: 99 s, loss: 0.1718, acc: 96.2377
Accuracy on the test images: 84 %
Epoch number 7, duration: 99 s, loss: 0.1127, acc: 98.1250
Accuracy on the test images: 84 %
Epoch number 8, duration: 98 s, loss: 0.1008, acc: 98.0760
Accuracy on the test images: 84 %
Epoch number 9, duration: 98 s, loss: 0.0870, acc: 98.6275
Accuracy on the test images: 85 %
Epoch number 10, duration: 99 s, loss: 0.0798, acc: 98.5907
Accuracy on the test images: 85 %
Finished Training
```

```
Text(0.5, 1.0, 'Test acc')
```



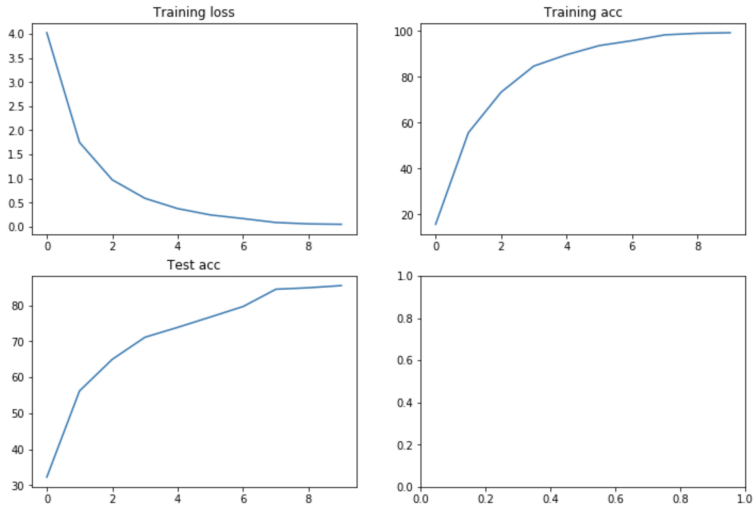
In questo caso vengono creati due diversi trainSet. Con diverse trasformazioni, in modo tale da utilizzare in diverse epoche. Le epoche pari avranno il primo set, mentre quelle dispari il secondo

```
#Transform code
transformation_train = transforms.Compose(
                               [transforms.Resize((300, 300)),
                                transforms.RandomRotation(15),
                                transforms.RandomCrop((250, 250)),
                                transforms.ToTensor(),
                                transforms.Normalize((0.4708, 0.4602, 0.4550), (0.2612, 0.2603, 0.2653))])
transformation_train2 = transforms.Compose(
                               [transforms.Resize((300, 300)),
                                transforms.RandomHorizontalFlip(),
                                transforms.ColorJitter(brightness=0.5, contrast=1, saturation=0.1, hue=0.5),
                                transforms.ToTensor(),
                                transforms.Normalize((0.4708, 0.4602, 0.4550), (0.2612, 0.2603, 0.2653))])

#...

for epoch in range(n_epochs):
        since = time.time()
        running_loss = 0.0
        running_correct = 0.0
        if epoch % 2 == 0:
            trainloader_diverso = trainloader
            print("trainloader1")
        else:
            trainloader_diverso = trainloader2
            print("trainloader2")
        # ...
```

```
Training
trainloader1
Epoch number 1, duration: 100 s, loss: 4.0249, acc: 15.6373
Accuracy on the test images: 32 %
trainloader2
Epoch number 2, duration: 98 s, loss: 1.7489, acc: 55.6005
Accuracy on the test images: 56 %
trainloader1
Epoch number 3, duration: 101 s, loss: 0.9723, acc: 73.2843
Accuracy on the test images: 64 %
trainloader2
Epoch number 4, duration: 98 s, loss: 0.5877, acc: 84.6936
Accuracy on the test images: 71 %
trainloader1
Epoch number 5, duration: 98 s, loss: 0.3750, acc: 89.6324
Accuracy on the test images: 73 %
trainloader2
Epoch number 6, duration: 99 s, loss: 0.2435, acc: 93.6275
Accuracy on the test images: 76 %
trainloader1
Epoch number 7, duration: 100 s, loss: 0.1675, acc: 95.7721
Accuracy on the test images: 79 %
trainloader2
Epoch number 8, duration: 99 s, loss: 0.0862, acc: 98.3088
Accuracy on the test images: 84 %
trainloader1
Epoch number 9, duration: 98 s, loss: 0.0578, acc: 98.9951
Accuracy on the test images: 84 %
trainloader2
Epoch number 10, duration: 99 s, loss: 0.0492, acc: 99.2525
```

```
Accuracy on the test images: 85 %
Finished Training
```

```
Text(0.5, 1.0, 'Test acc')
```



Conclusioni

Vengono utilizzati **diversi modelli** con **diverse teniche di augmentation** per provare a migliorare il più possibile il risultato di **accuracy** sul testset.

I modelli sembrano andare molto bene con **performance vicine al 90%**. Si può notare come in una grande percentuale dei casi vengano raggiunti degli stalli di performance circa dall'epoca n°6/7. In quanto il modello non si scosta da un certo **valore target**. Continuando ad **oscillare** in un intorno che non porta un miglioramento effettivo al modello.

Pare inoltre che **aumentando complessità** tramite modelli più grandi questo non incida particolarmente sulle perfomance, utilizzando le **stesse configurazioni usate in precedenza**. Questo può significare che non è per forza necessaria più complessità a livello di modello ma che forse la **ricerca per migliorare** può continuare in direzione augmentation oppure in un **dataset più ricco** e ancora più complesso.

Vengono registrate ottime performance sul **train set** quasi sempre.

La heatmap rappresentante la **matrice di confusione** rende bene tramite la diagonale della matrice la potenza del modello testato, si ha una certa **precisione** in buona parte delle classi. È inoltre ben visibile come le classi 58-60 abbiano un leggero **calo di performance**.

Alberto Coffrini