Import delle librerie, web scraping, analisi, pulizia dei dati e costruzione del Dataset

La fase preliminare è quella dell'importazione di tutte le librerie necessarie:

- Numpy per il calcolo scientifico
- Pandas per la Data Analysis
- Matplotlib per i grafici
- Torch per il Machine Learning
- PIL per la gestione delle immagini
- Requests e BeautifulSoup per la gestione delle richieste HTTP e lo scraping dell'url che contiene il dataset

Viene inoltre installato il modulo Gradio tramite pip che permette di avere un'implementazione di una demo dei modelli di Machine Learning presentati tramite un interfaccia grafica.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from skimage import io
from sklearn.metrics import mean_absolute_error

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

import os

from os import path
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
from torchvision.io import read_image
from PIL import Image

import requests
from requests.adapters import HTTPAdapter
from urllib3.util.retry import Retry
from bs4 import BeautifulSoup
```

```
!pip install gradio
```

```
Requirement already satisfied: gradio in /usr/local/lib/python3.10/dist-packages (3.38.0)
Requirement already satisfied: aiofiles<24.0,>=22.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (23.1.0)
Requirement already satisfied: aiohttp~=3.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (3.8.4)
Requirement already satisfied: altair<6.0,>=4.2.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (4.2.2)
Requirement already satisfied: fastapi in /usr/local/lib/python3.10/dist-packages (from gradio) (0.100.0)
Requirement already satisfied: ffmpy in /usr/local/lib/python3.10/dist-packages (from gradio) (0.3.1)
Requirement already satisfied: gradio-client>=0.2.10 in /usr/local/lib/python3.10/dist-packages (from gradio) (0.2.10)
Requirement already satisfied: httpx in /usr/local/lib/python3.10/dist-packages (from gradio) (0.24.1)
Requirement already satisfied: huggingface-hub>=0.14.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (0.16.4)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (3.1.2)
Requirement already satisfied: markdown-it-py[linkify]>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (2.2.0)
Requirement already satisfied: markupsafe~=2.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (2.1.3)
Requirement already satisfied: matplotlib~=3.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (3.7.1)
Requirement already satisfied: mdit-py-plugins<=0.3.3 in /usr/local/lib/python3.10/dist-packages (from gradio) (0.3.3)
Requirement already satisfied: numpy~=1.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (1.22.4)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (3.9.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from gradio) (23.1)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (1.5.3)
Requirement already satisfied: pillow<11.0,>=8.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (8.4.0)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: pydub in /usr/local/lib/python3.10/dist-packages (from gradio) (0.25.1)
Requirement already satisfied: python-multipart in /usr/local/lib/python3.10/dist-packages (from gradio) (0.0.6)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (6.0.1)
Requirement already satisfied: requests~=2.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (2.27.1)
Requirement already satisfied: semantic-version~=2.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (2.10.0)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (4.7.1)
Requirement already satisfied: uvicorn>=0.14.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (0.23.1)
Requirement already satisfied: websockets<12.0,>=10.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (11.0.3)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp~=3.0->gradio) (23.1.0)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp~=3.0->gradi
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp~=3.0->gradio) (6.0.4
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.10/dist-packages (from aiohttp~=3.0->gradio
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp~=3.0->gradio) (1.9.2)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp~=3.0->gradio) (1.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp~=3.0->gradio) (1.3.1)
Requirement already satisfied: entrypoints in /usr/local/lib/python3.10/dist-packages (from altair<6.0,>=4.2.0->gradio) (0.4)
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.10/dist-packages (from altair<6.0,>=4.2.0->gradio) (4.3
```

```
Requirement already satisfied: toolz in /usr/local/lib/python3.10/dist-packages (from altair<6.0,>=4.2.0->gradio) (0.12.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from gradio-client>=0.2.10->gradio) (2023.6.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.14.0->gradio) (3.12.
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.14.0->gradio) (4
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py[linkify]>=2.0.0->gradi
Requirement already satisfied: linkify-it-py<3,>=1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py[linkify]>=2.0
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib~=3.0->gradio) (1.1.0
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib~=3.0->gradio) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib~=3.0->gradio) (4.41
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib~=3.0->gradio) (1.4.
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib~=3.0->gradio) (3.1.0
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib~=3.0->gradio) (2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas<3.0,>=1.0->gradio) (2022.7.1
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests~=2.0->gradio) (1.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests~=2.0->gradio) (2023.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests~=2.0->gradio) (3.4)
Requirement already satisfied: click>=7.0 in /usr/local/lib/python3.10/dist-packages (from uvicorn>=0.14.0->gradio) (8.1.6)
Requirement already satisfied: h11>=0.8 in /usr/local/lib/python3.10/dist-packages (from uvicorn>=0.14.0->gradio) (0.14.0)
Requirement already satisfied: starlette<0.28.0,>=0.27.0 in /usr/local/lib/python3.10/dist-packages (from fastapi->gradio) (0.27
```

```
URL = "https://iplab.dmi.unict.it/BCS/dataset.html"
session = requests.Session()
retry = Retry(connect=3, backoff_factor=0.5)
adapter = HTTPAdapter(max_retries=retry)
session.mount('http://', adapter)
session.mount('https://', adapter)
res = requests.get(URL)
soup = BeautifulSoup(res.content, 'html.parser')
tables = soup.find_all('table')
table = tables[4] # note that the first table (at index 0) is not relevant
tables = tables[4:]
# Print the first 250 characters of the html table
print(str(tables)[:250])
```

```
[<table align="center" width="800">
<tbody>
<tr>
<td height="14" width="20%">
<div align="center"><a href="DatasetImagesCows/71_4_10_20_158_cow.jpg" target="_blank"><img border="0" src="DatasetImagesCows/71_4_
```

A questo punto si effettua lo scraping inserendo in liste differenti i vari dati che successivamente verranno inseriti in un dataset. Per quanto riguarda l'indice BCS viene effettuata la media tra gli indici BCS1 e BCS2 forniti da esperti del settore.

```
original_images=soup.select("[height] a img")
shape_anatomical=soup.select("td:nth-of-type(2) img")
data=soup.select("font a[href]",href=True)

bcs_1=soup.select("table:nth-of-type(n+5) tr:nth-of-type(1) td:nth-of-type(4) div")
bcs_2=soup.select("table:nth-of-type(n+5) tr:nth-of-type(1) td:nth-of-type(5) div")
```

```
BCS_1=[]
BCS_2=[]
BCS=[]
for i in bcs_1:
  BCS_1.append(i.text)

for i in bcs_2:
  BCS_2.append(i.text)

if (len(bcs_1)==len(BCS_2)):
  for i in range(len(BCS_1)):
    BCS.append((float(BCS_1[i])+float(BCS_2[i]))/2)


temp=[]
for i in original_images:
  temp.append(i['src'].removeprefix("DatasetImagesCows/"))



d={"Shape": temp, "BCS": BCS}
df=pd.DataFrame(d)
df.drop_duplicates(inplace=True)
df
```

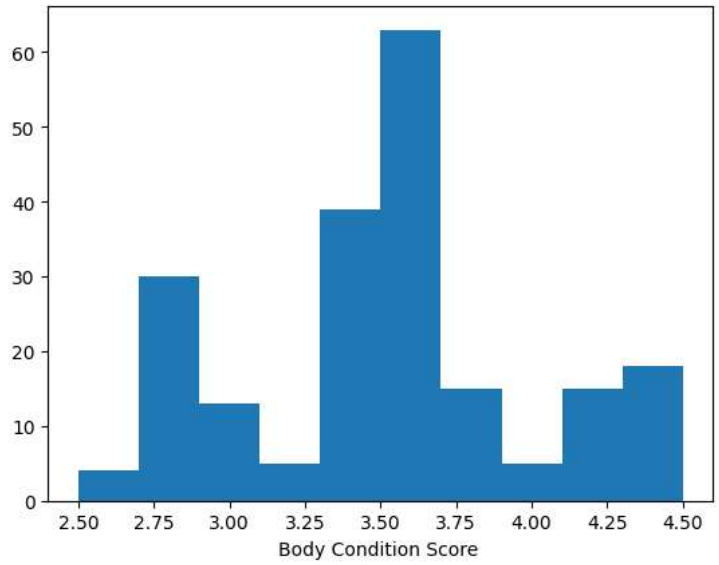
	Shape	BCS
0	71_4_10_20_158_cow.jpg	3.000
1	700_4_11_10_298_cow.jpg	3.375
2	700_4_11_10_300_cow.jpg	3.375
3	128_4_4_10_132_cow.jpg	4.250
4	71_4_10_20_156_cow.jpg	3.000
...	...	...
202	132_4_20_21_112_cow.jpg	4.500
203	49_4_26_12_109_cow.jpg	3.375
204	49_4_26_12_111_cow.jpg	3.375
205	700_4_29_12_90_cow.jpg	3.375

```
df.to_csv("dataset.csv")
```

207 rows × 2 columns

```
plt.hist(df['BCS'])
plt.xlabel("Body Condition Score ")
```

```
Text(0.5, 0, 'Body Condition Score ')
```



L'istogramma rappresenta la distribuzione dei BCS dei bovini all'interno del dataset.

A questo punto si costuisce un custom dataset che permette la gestione e il caricamento delle immagini che rappresentano i samples. Le transforms utilizzate inizialmente sono:

- resize a dimensione 32x42 per mantenere inalterato il rapporto originale 4:3 delle immagini
- trasformazione in Tensore per poter successivamente calcolare media e deviazione standard e poter effettuare di conseguenza una normalizzazione dei dati

```
class BCSDataset(Dataset):

  def __init__(self,base_path,data,transform=None):
    self.base_path=base_path
    self.images = data
    self.transform = transform

  def __getitem__(self, index):
    img_path = os.path.join(self.base_path, self.images.iloc[index, 0])
    image = Image.open(path.join(self.base_path, self.images.iloc[index, 0]))

    label = self.images.iloc[index, 1]
    label=float(label)
    if self.transform:
      image = self.transform(image)

    return {'image' : image, 'label':label}

  def __len__(self):
    return len(self.images)
```

```
#media: 2.9967
#std: 3.4603
transform = transforms.Compose([transforms.Resize((32,42)),
                                transforms.ToTensor()])
dataset=BCSDataset("sample_data/dataset_cows", df, transform=transform)
print(dataset[0]['image'].shape)
print(dataset[0]['label'])
```

```
torch.Size([3, 32, 42])
3.0
```

```
m = 0
for sample in dataset:
 m+=sample['image'].sum() #accumuliamo la somma di tutti i pixel
#dividiamo per il numero di immagini moltiplicato per il numero di pixel
m=m/(len(dataset)*32*42)

#procedura simile per calcolare la deviazione standard
s=0
for sample in dataset:
 s+=((sample['image']-m)**2).sum()

s=np.sqrt(s/(len(dataset)*32*42))



print("Media: ",m)
print("Std: ",s)
```

```
Media:  tensor(1.3486)
Std:  tensor(1.6272)
```

Una volta calcolate media e deviazione standard si procede con nuove Transforms per effettuare normalizzazione e Data Augmentation essendo il dataset originale molto piccolo.

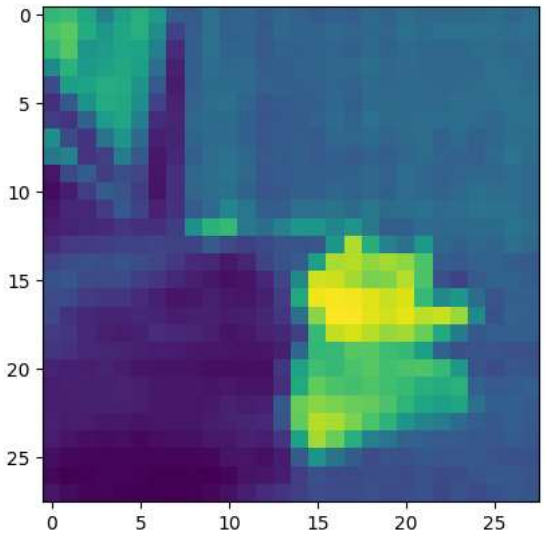
```
#media: 2.9967
#std: 3.4603
transform = transforms.Compose([transforms.Resize((32,42)),
                                transforms.RandomHorizontalFlip(), transforms.RandomVerticalFlip(), transforms.ColorJitter(), transforms
                                transforms.ToTensor(), transforms.Normalize(m,s)])

dataset=BCSDataset("sample_data/dataset_cows", df, transform=transform)
print(dataset[0]['image'].shape)
print(dataset[0]['label'])
```

```
torch.Size([3, 28, 28])
3.0
```

```
plt.imshow(dataset[0]['image'].numpy()[0])
```

```
<matplotlib.image.AxesImage at 0x79992dce9e10>
```



Suddivisione del Dataset e definizione delle funzioni di training e testing

A questo punto si effettua uno split random del dataset originale in:

- Training Set (rappresentante l'80% del totale)
- Test Set (rappresentante il 20% del totale)

La dimensione dei batch viene impostata a 16 per l'esecuzione di SGD.

```
train_set, test_set = torch.utils.data.random_split(dataset, [0.8,0.2])

batch_size=16
num_workers=2
train_loader=DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=num_workers)
test_loader=DataLoader(test_set, batch_size=batch_size, shuffle=True, num_workers=num_workers)


class AverageValueMeter():
 def __init__(self):
  self.reset()

 def reset(self):
  self.sum = 0
  self.num = 0

 def add(self, value, num):
  self.sum += value*num
  self.num += num

 def value(self):
  try:
    return self.sum/self.num
  except:
    return None
```

La seguente funzione permette di effettuare il training del regressore. Come Loss function è stata utilizzata la MSE Loss con Stochastic Gradient Descent. Alla fine di ogni epoch viene stampato il valore di loss e salvato il modello. La funzione di test_regressor effettua invece testa il modello regressivo sul test set e ne valuta le prestazioni.

```
from torch.optim import SGD
from torch.utils.tensorboard import SummaryWriter
from sklearn.metrics import mean_squared_error
from os.path import join
from torch.optim import Adam
def train_regressor(model, train_loader, test_loader, exp_name='experiment', lr=0.01, epochs=10, momentum=0.99, weight_decay=0, logdir='
  criterion = nn.MSELoss()
  optimizer = SGD(model.parameters(), lr=lr,momentum=momentum, weight_decay=weight_decay)

  loss_meter = AverageValueMeter()

  writer = SummaryWriter(join(logdir, exp_name))

  device = "cuda" if torch.cuda.is_available() else "cpu"
  model.to(device)

  loader = {'train' : train_loader, 'test' : test_loader}
  global_step=0


  for e in range(epochs):
    for mode in ['train','test']:
      loss_meter.reset()

      model.train() if mode=='train' else model.eval()
      with torch.set_grad_enabled(mode=='train'):

        for i, batch in enumerate(loader[mode]):
          x=batch['image'].to(device)
         # print(x.size())
          y=batch['label'].to(device).float()
          #print(y.size())
          output=model(x)
          n = x.shape[0]

          global_step+=n
          l = criterion(output.view(-1),y)
```

```
            if mode=='train':
              l.backward()
              optimizer.step()
              optimizer.zero_grad()

            loss_meter.add(l.item(),n)


            if mode=='train':
              writer.add_scalar('loss/train', loss_meter.value(), global_step=global_step)

        writer.add_scalar('loss/' + mode, loss_meter.value(), global_step=global_step)


      print("Epoch: ",e,"Loss: ",loss_meter.value())

      if ((e+1)%10==0):
        torch.save(model.state_dict(),'%s-%d.pth'%(exp_name,e+1))
    return model


def test_regressor(model, loader):
    device = "cuda" if torch.cuda.is_available() else "cpu"
    model.to(device)
    predictions, labels = [], []
    for batch in loader:
      x = batch['image'].to(device)
      y = batch['label'].to(device)
      output = model(x)
      preds = output.to('cpu').detach().numpy()
      labs = y.to('cpu').numpy()
      print("Labels: ", labs)
      predictions.extend(list(preds))
      labels.extend(list(labs))

    return np.array(predictions), np.array(labels)
```

Si è implementata una funzione di training del regressore anche tramite ottimizzatore Adam per effettuare un confronto a livello di prestazioni per la risoluzione del problema in questione. E' stato inoltre utilizzato un Learning Rate Scheduler per gestire in funzione delle iterazioni dell'algoritmo SGD il parametro di learning per poter ottenere risultati ancora migliori. Lo scheduler utilizzato è di tipo step-decay, che ogni 30 epochs diminuisce il valore di learning rate.

```
from torch.optim import SGD
from torch.utils.tensorboard import SummaryWriter
from sklearn.metrics import mean_squared_error
from os.path import join
from torch.optim import Adam
from torch.optim.lr_scheduler import StepLR


def train_regressor_adam(model, train_loader, test_loader, exp_name='experiment', lr=0.01, epochs=10, weight_decay=0, logdir='logs', lr_
    criterion = nn.MSELoss()
    optimizer = Adam(model.parameters(), lr=lr, weight_decay=weight_decay)
    scheduler = StepLR(optimizer, step_size=30, gamma=0.9)

    loss_meter = AverageValueMeter()

    writer = SummaryWriter(join(logdir, exp_name))

    device = "cuda" if torch.cuda.is_available() else "cpu"
    model.to(device)

    loader = {'train' : train_loader, 'test' : test_loader}
    global_step=0


    for e in range(epochs):

      for mode in ['train','test']:
        loss_meter.reset()

        model.train() if mode=='train' else model.eval()
        with torch.set_grad_enabled(mode=='train'):

          for i, batch in enumerate(loader[mode]):
            x=batch['image'].to(device)
           # print(x.size())
            y=batch['label'].to(device).float()
```

```
            #print(y.size())
            output=model(x)

            n = x.shape[0]

            global_step+=n
            l = criterion(output.view(-1),y)

            if mode=='train':
              l.backward()
              optimizer.step()
              optimizer.zero_grad()


            loss_meter.add(l.item(),n)


            if mode=='train':
              writer.add_scalar('loss/train', loss_meter.value(), global_step=global_step)

      writer.add_scalar('loss/' + mode, loss_meter.value(), global_step=global_step)
    scheduler.step()

    print("Epoch: ",e,"Loss: ",loss_meter.value())
    print(scheduler.state_dict()['_last_lr'])
    if ((e+1)%10==0):
      torch.save(model.state_dict(),'%s-%d.pth'%(exp_name,e+1))
  return model
```

La prima architettura presentata prende spunto da MiniAlexNet, adattandola agli input a tre canali RGB di dimensioni 32x42. Sono mantenuti i 5 Convolutional Layers e i 2 Fully Connected Layers

MINIALEXNET

Di seguito sono presenti le tre implementazioni differenti di MiniAlexNet:

- MiniAlexNet utilizza immagini RGB
- MiniAlexNetV2 utilizza immagini RGB con padding differente
- MiniAlexNetV3 utilizza immagini in scala di grigi

```
class MiniAlexNet(nn.Module):
    def __init__(self):
        super(MiniAlexNet, self).__init__()

        self.feature_extractor = nn.Sequential(
        #Conv1
        nn.Conv2d(3, 16, 3, padding=3),
        nn.ReLU(),

        #Conv2
        nn.BatchNorm2d(16),
        nn.Conv2d(16, 32, 3, padding=3),
        nn.MaxPool2d(2),
        nn.ReLU(),

        #Conv3
        nn.BatchNorm2d(32),
        nn.Conv2d(32, 64, 3, padding=0),
        nn.ReLU(),

        #Conv4
        nn.BatchNorm2d(64),
        nn.Conv2d(64, 128, 3, padding=0),
        nn.ReLU(),

        #Conv5
        nn.BatchNorm2d(128),
        nn.Conv2d(128, 256, 3, padding=0),
        nn.MaxPool2d(2),
        nn.ReLU(),
        )

        self.regressor = nn.Sequential(

        nn.Dropout(0.4),

        #FC6
        nn.BatchNorm1d(9216 ),
```

```
            nn.Linear(9216  , 4096),
            nn.ReLU(),

            nn.Dropout(0.4),

            #FC7
            nn.BatchNorm1d(4096),
            nn.Linear(4096, 2048),
            nn.ReLU(),


            #FC8
            nn.BatchNorm1d(2048),
            nn.Linear(2048, 1)

            )


        def forward(self,x):
            #Applichiamo le diverse trasformazioni in cascata
            x = self.feature_extractor(x)
            x = self.regressor(x.view(x.shape[0],-1))
            return x


    minialexnet=MiniAlexNet()
    minialexnet=train_regressor(minialexnet,train_loader, test_loader, exp_name="test/MiniAlexNet3",lr=1e-4, momentum=0.9, weight_decay=0.1,
```

```
Epoch:  0 Loss:  12.9503131261686
Epoch:  1 Loss:  13.13039065570366
Epoch:  2 Loss:  11.831337021618355
Epoch:  3 Loss:  10.888208203199433
Epoch:  4 Loss:  10.125850770531631
Epoch:  5 Loss:  10.272249338103503
Epoch:  6 Loss:  10.172282846962533
Epoch:  7 Loss:  9.276813367517983
Epoch:  8 Loss:  8.95511757455221
Epoch:  9 Loss:  8.745527337237101
Epoch:  10 Loss:  8.074594986147996
Epoch:  11 Loss:  7.4224175709049875
Epoch:  12 Loss:  7.081185480443443
Epoch:  13 Loss:  6.472076648619117
Epoch:  14 Loss:  5.879671306144901
Epoch:  15 Loss:  6.195604743027106
Epoch:  16 Loss:  5.105180705465922
Epoch:  17 Loss:  5.4798424534681365
Epoch:  18 Loss:  5.250263202481154
Epoch:  19 Loss:  5.044281238462867
Epoch:  20 Loss:  4.341019851405446
Epoch:  21 Loss:  3.3773754108242873
Epoch:  22 Loss:  3.2783533421958366
Epoch:  23 Loss:  3.037532719170175
Epoch:  24 Loss:  3.320690178289646
Epoch:  25 Loss:  3.0929189775048234
Epoch:  26 Loss:  3.1558890575315894
Epoch:  27 Loss:  2.7870955292771504
Epoch:  28 Loss:  2.4858814216241605
Epoch:  29 Loss:  2.624500943393242
Epoch:  30 Loss:  2.1282160747341994
Epoch:  31 Loss:  3.191649942863278
Epoch:  32 Loss:  1.6767459613520925
Epoch:  33 Loss:  1.4738631888133724
Epoch:  34 Loss:  1.605358312769634
Epoch:  35 Loss:  2.4152775159696254
Epoch:  36 Loss:  2.635113908023369
Epoch:  37 Loss:  1.9273823906735676
Epoch:  38 Loss:  1.6624339848029903
Epoch:  39 Loss:  1.6121862777849523
Epoch:  40 Loss:  0.9040047577241572
Epoch:  41 Loss:  1.5091803190184803
Epoch:  42 Loss:  1.2491804535796003
Epoch:  43 Loss:  1.9484623583351695
Epoch:  44 Loss:  1.6555201367634098
Epoch:  45 Loss:  1.3268645609297403
Epoch:  46 Loss:  1.1628413636509964
Epoch:  47 Loss:  1.8392656838021628
Epoch:  48 Loss:  1.4910323910596894
Epoch:  49 Loss:  0.8101082953011117
Epoch:  50 Loss:  0.8445442958575923
Epoch:  51 Loss:  0.8445712647786955
Epoch:  52 Loss:  0.8720791834156688
Epoch:  53 Loss:  0.7096068161289867
Epoch:  54 Loss:  0.6148389228960363
Epoch:  55 Loss:  0.5181763615550065
Epoch:  56 Loss:  0.48699823821463234
Epoch:  57 Loss:  0.8498108990308715
Epoch:  58 Loss:  0.6021278675009565
Epoch:  59 Loss:  0.6598314548411021
Epoch:  60 Loss:  0.7205604896312807
Epoch:  61 Loss:  0.4985445054565988
Epoch:  62 Loss:  0.7156983584892459
Epoch:  63 Loss:  0.6710618894274641
Epoch:  64 Loss:  1.0142273321384336
Epoch:  65 Loss:  0.9404372500210274
Epoch:  66 Loss:  0.4533664409707232
Epoch:  67 Loss:  0.6091871552351045
Epoch:  68 Loss:  0.5222844866717734
Epoch:  69 Loss:  0.44362602219349
Epoch:  70 Loss:  0.6263039926203285
Epoch:  71 Loss:  0.751876604266283
Epoch:  72 Loss:  0.7534480894484171
Epoch:  73 Loss:  0.8285336785200166
Epoch:  74 Loss:  0.3704271520056376
Epoch:  75 Loss:  0.5134707049625676
Epoch:  76 Loss:  0.5592849966956348
Epoch:  77 Loss:  0.2845745166627372
Epoch:  78 Loss:  0.5233285812343039
Epoch:  79 Loss:  0.5662224866994997
Epoch:  80 Loss:  0.26175066537973357
Epoch:  81 Loss:  0.4123274596726022
Epoch:  82 Loss:  0.31511219463697293
Epoch:  83 Loss:  0.2766073731387534
Epoch:  84 Loss:  0.5914521071969009
Epoch:  85 Loss:  0.2831607170221282
Epoch:  86 Loss:  0.4342508083436547
Epoch:  87 Loss:  0.583714931476407
Epoch:  88 Loss:  0.2185307232345023
Epoch:  89 Loss:  0.2963951986010482
```

```
Epoch:  90 Loss:  0.22643158930103954
Epoch:  91 Loss:  0.4132237666990699
Epoch:  92 Loss:  0.3913415897546745
Epoch:  93 Loss:  0.3138596484573876
Epoch:  94 Loss:  0.2506963319894744
Epoch:  95 Loss:  0.6368693113327026
Epoch:  96 Loss:  0.3262220668356593
Epoch:  97 Loss:  0.2530495913290396
Epoch:  98 Loss:  0.3412533286141186
Epoch:  99 Loss:  0.2663205782087838
Epoch:  100 Loss:  0.1976361903475552
Epoch:  101 Loss:  0.24692473542399523
Epoch:  102 Loss:  0.2658862417064062
Epoch:  103 Loss:  0.24639670950610462
Epoch:  104 Loss:  0.23230127227015612
Epoch:  105 Loss:  0.3973281038243596
Epoch:  106 Loss:  0.35364643030050325
Epoch:  107 Loss:  0.4261100699261921
Epoch:  108 Loss:  0.21734414013420664
Epoch:  109 Loss:  0.214090991310957
Epoch:  110 Loss:  0.2773682409670295
Epoch:  111 Loss:  0.24187158593317357
Epoch:  112 Loss:  0.38521970263341576
Epoch:  113 Loss:  0.2905938363656765
Epoch:  114 Loss:  0.4578070655101683
Epoch:  115 Loss:  0.31002481354445943
Epoch:  116 Loss:  0.3473589267672562
Epoch:  117 Loss:  0.25792211076108423
Epoch:  118 Loss:  0.27199426366061696
Epoch:  119 Loss:  0.2234711187278352
Epoch:  120 Loss:  0.2918686277982665
Epoch:  121 Loss:  0.2648094605381896
Epoch:  122 Loss:  0.2668234974872775
Epoch:  123 Loss:  0.24584286278340875
Epoch:  124 Loss:  0.28797103391914836
Epoch:  125 Loss:  0.42947875726513746
Epoch:  126 Loss:  0.2659699066383083
Epoch:  127 Loss:  0.2787356485680836
Epoch:  128 Loss:  0.24659334432061125
Epoch:  129 Loss:  0.24279143970187117
Epoch:  130 Loss:  0.2310367415590984
Epoch:  131 Loss:  0.26785738584471913
Epoch:  132 Loss:  0.2522698117465508
Epoch:  133 Loss:  0.2584468896796064
Epoch:  134 Loss:  0.2622510659985426
Epoch:  135 Loss:  0.27962879364083454
Epoch:  136 Loss:  0.3424346141698884
Epoch:  137 Loss:  0.26959299605067183
Epoch:  138 Loss:  0.27323949264317027
Epoch:  139 Loss:  0.26617364185612374
Epoch:  140 Loss:  0.24817833958602534
Epoch:  141 Loss:  0.2531391756563652
Epoch:  142 Loss:  0.260158055439228
Epoch:  143 Loss:  0.2429125839617194
Epoch:  144 Loss:  0.22729691063485494
Epoch:  145 Loss:  0.23240738234868863
Epoch:  146 Loss:  0.21107727725331377
Epoch:  147 Loss:  0.2106950086791341
Epoch:  148 Loss:  0.21592980804966716
Epoch:  149 Loss:  0.2688939745833234
Epoch:  150 Loss:  0.3039013307269027
Epoch:  151 Loss:  0.27066132789704855
Epoch:  152 Loss:  0.2076264532600961
Epoch:  153 Loss:  0.25763354795735055
Epoch:  154 Loss:  0.20031787673147713
Epoch:  155 Loss:  0.24346955411317872
Epoch:  156 Loss:  0.1993699456315215
Epoch:  157 Loss:  0.30756107463342386
Epoch:  158 Loss:  0.268294937726928
Epoch:  159 Loss:  0.29451090533558916
Epoch:  160 Loss:  0.2538154379623692
Epoch:  161 Loss:  0.246814244404071
```

```
test_predictions, labels_test = test_regressor(
 minialexnet, test_loader)
print("MSE: ", mean_squared_error(labels_test, test_predictions))
```

```
Labels:  [3.375 3.375 4.25  3.5   3.    3.625 3.5   4.5   3.    3.5   2.875 3.375
 3.5   3.75  2.875 4.5  ]
Labels:  [3.125 2.625 3.5   3.5   4.25  2.875 3.5   3.375 3.5   2.75  2.875 3.625
 4.5   3.5   3.375 3.375]
Labels:  [2.875 3.375 3.75  3.375 2.625 3.125 3.625 4.5   3.375]
MSE:  0.27084973875264495
```

```
Epoch:  173 Loss:  0.4536523281074152
```

```
class MiniAlexNetV2(nn.Module):
    def __init__(self):
        super(MiniAlexNetV2, self).__init__()
        self.feature_extractor = nn.Sequential(
        #Conv1
```

```
            nn.Conv2d(3, 16, 5, padding=2),
            nn.MaxPool2d(2),
            nn.ReLU(),

            #Conv2
            nn.BatchNorm2d(16),
            nn.Conv2d(16, 32, 5, padding=2),
            nn.MaxPool2d(2),
            nn.ReLU(),

            #Conv3
            nn.BatchNorm2d(32),
            nn.Conv2d(32, 64, 3, padding=1),
            nn.ReLU(),

            #Conv4
            nn.BatchNorm2d(64),
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(),

            #Conv5
            nn.BatchNorm2d(128),
            nn.Conv2d(128, 256, 3, padding=1),
            nn.MaxPool2d(2),
            nn.ReLU()
            )

            self.regressor = nn.Sequential(
            nn.Dropout(0.6),
            #FC6
            nn.BatchNorm1d(2304),
            nn.Linear(2304, 4096),
            nn.ReLU(),

            nn.Dropout(0.6),
            #FC7
            nn.BatchNorm1d(4096),
            nn.Linear(4096, 1024),
            nn.ReLU(),

            #FC8
            nn.BatchNorm1d(1024),
            nn.Linear(1024, 1)
            )


        def forward(self,x):

            x = self.feature_extractor(x)
            x = self.regressor(x.view(x.shape[0],-1))
            return x


    minialexnetv2=MiniAlexNetV2()
    minialexnet=train_regressor(minialexnetv2,train_loader, test_loader, exp_name="test/MiniAlexNetV2",lr=1e-3, momentum=0.5, weight_decay=0
```

```
Epoch:  171 Loss:  0.26472569238848803
Epoch:  172 Loss:  0.25019419447677893
Epoch:  173 Loss:  0.31849808990955353
Epoch:  174 Loss:  0.2605969487893872
Epoch:  175 Loss:  0.23831971534868565
Epoch:  176 Loss:  0.2341983238371407
Epoch:  177 Loss:  0.2506592869758606
Epoch:  178 Loss:  0.22997352771642732
Epoch:  179 Loss:  0.2441588160468311
Epoch:  180 Loss:  0.2078412371437724
Epoch:  181 Loss:  0.23833251726336596
Epoch:  182 Loss:  0.2555168852573488
Epoch:  183 Loss:  0.23649234415554418
Epoch:  184 Loss:  0.24343852589770063
Epoch:  185 Loss:  0.22401523299333526
Epoch:  186 Loss:  0.20630948427246837
Epoch:  187 Loss:  0.2351839433356029
Epoch:  188 Loss:  0.22127256724165706
Epoch:  189 Loss:  0.1989126412606821
Epoch:  190 Loss:  0.24156077787643526
Epoch:  191 Loss:  0.2010054522898139
Epoch:  192 Loss:  0.22327679177609885
Epoch:  193 Loss:  0.22208515827248737
Epoch:  194 Loss:  0.21658742209760154
Epoch:  195 Loss:  0.20627573714023684
Epoch:  196 Loss:  0.20934927027399947
Epoch:  197 Loss:  0.25118933690757284
Epoch:  198 Loss:  0.2239600790709984
Epoch:  199 Loss:  0.23092568066061997
```

```
test_predictions, labels_test = test_regressor(
 minialexnetv2, test_loader)
print("MSE: ", mean_squared_error(labels_test, test_predictions))
```

```
Labels:  [2.875 3.125 4.25  3.5   3.375 4.25  3.375 3.125 3.375 3.5   3.5   4.5
 3.5   3.5   3.    3.5  ]
Labels:  [2.875 3.375 3.375 4.5   3.    3.75  2.875 3.5   3.375 2.625 3.5   2.625
 3.375 3.375 4.5   2.75 ]
Labels:  [3.5   4.5   2.875 3.375 2.875 3.75  3.625 3.625 3.625]
MSE:  0.23146817932512415
```

Si effettua a questo punto una transform per ottenere immagini 32x42 in scala di grigi e si allena il modello MiniAlexNetV3 con Adam e step decay learning rate scheduler.

```
#media: 2.9967
#std: 3.4603
transform = transforms.Compose([transforms.Resize((32,42)),
                                transforms.RandomHorizontalFlip(), transforms.RandomVerticalFlip(), transforms.Grayscale(),
                                transforms.ToTensor(), transforms.Normalize(m,s)])

dataset=BCSDataset("sample_data/dataset_cows", df, transform=transform)
print(dataset[0]['image'].shape)
print(dataset[0]['label'])
```

```
torch.Size([1, 32, 42])
3.0
```

```
train_set, test_set = torch.utils.data.random_split(dataset, [0.8,0.2])

batch_size=24
num_workers=2
train_loader=DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=num_workers)
test_loader=DataLoader(test_set, batch_size=batch_size, shuffle=True, num_workers=num_workers)


class MiniAlexNetV3(nn.Module):
    def __init__(self):
        super(MiniAlexNetV3, self).__init__()

        self.feature_extractor = nn.Sequential(
        #Conv1
        nn.Conv2d(1, 16, 5, padding=2),
        nn.MaxPool2d(2),
        nn.ReLU(),

        #Conv2
        nn.BatchNorm2d(16),
        nn.Conv2d(16, 32, 5, padding=2),
        nn.MaxPool2d(2),
        nn.ReLU(),

        #Conv3
```

```
            nn.BatchNorm2d(32),
            nn.Conv2d(32, 64, 3, padding=1),
            nn.ReLU(),

            #Conv4
            nn.BatchNorm2d(64),
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(),

            #Conv5
            nn.BatchNorm2d(128),
            nn.Conv2d(128, 256, 3, padding=1),
            nn.MaxPool2d(2),
            nn.ReLU()
            )

            self.regressor = nn.Sequential(
            nn.Dropout(0.6),
            #FC6
            nn.BatchNorm1d(5120),
            nn.Linear(5120, 4096),
            nn.ReLU(),

            nn.Dropout(0.6),
            #FC7
            nn.BatchNorm1d(4096),
            nn.Linear(4096, 1024),
            nn.ReLU(),

            #FC8
            nn.BatchNorm1d(1024),
            nn.Linear(1024, 1)
            )


        def forward(self,x):

            x = self.feature_extractor(x)
            x = self.regressor(x.view(x.shape[0],-1))
            return x


    minialexnetv3=MiniAlexNetV3()
    minialexnetv3=train_regressor_adam(minialexnetv3,train_loader, test_loader, exp_name="test/MiniAlexNetV3",lr=1e-3, weight_decay=0.1, epo
```

```
[0.00025418658283290005]
Epoch:  392 Loss:  0.19302407052458787
[0.00025418658283290005]
Epoch:  393 Loss:  0.2705282434457686
[0.00025418658283290005]
Epoch:  394 Loss:  0.27916283105931633
[0.00025418658283290005]
Epoch:  395 Loss:  0.1532705397140689
[0.00025418658283290005]
Epoch:  396 Loss:  0.1869789051210008
[0.00025418658283290005]
Epoch:  397 Loss:  0.1982755457482687
[0.00025418658283290005]
Epoch:  398 Loss:  0.15771590427654544
[0.00025418658283290005]
Epoch:  399 Loss:  0.22163563494275257
[0.00025418658283290005]
```

```
test_predictions, labels_test = test_regressor(
 minialexnetv3, test_loader)
print("MSE: ", mean_squared_error(labels_test, test_predictions))
```

```
Labels:  [3.375 4.5   4.5   2.625 3.5   3.5   3.625 3.375 4.25  3.5   2.75  4.5
 2.625 3.    3.125 3.375 3.375 4.5   3.5   4.    2.625 3.5   4.5   2.875]
Labels:  [3.5   3.5   4.5   2.875 3.375 3.5   3.625 3.625 3.75  2.875 3.375 3.75
 3.125 3.375 3.5   4.5   3.75 ]
MSE:  0.21443357444357572
```

CNN Regressor

CNN Regressor utilizza, a differenza di MiniAlexNet, solo 3 layer convoluzionali. In questo caso sono state utilizzate solo immagini RGB e una batch size pari a 24.

```
#media: 2.9967
#std: 3.4603
transform = transforms.Compose([transforms.Resize((32,42)),
                                transforms.RandomHorizontalFlip(), transforms.RandomVerticalFlip(), transforms.ColorJitter(),
                                transforms.ToTensor(), transforms.Normalize(m,s)])

dataset=BCSDataset("sample_data/dataset_cows", df, transform=transform)
print(dataset[0]['image'].shape)
print(dataset[0]['label'])
train_set, test_set = torch.utils.data.random_split(dataset, [0.8,0.2])

batch_size=24
num_workers=2
train_loader=DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=num_workers)
test_loader=DataLoader(test_set, batch_size=batch_size, shuffle=True, num_workers=num_workers)
```

```
torch.Size([3, 32, 42])
3.0
```

```
class CNNRegressorV2(nn.Module):
    def __init__(self,l1,l2):
        super(CNNRegressorV2, self).__init__()

        self.feature_extractor = nn.Sequential(
        #Conv1
        nn.Conv2d(3, 32, 5, padding=1),
        nn.MaxPool2d(2),
        nn.ReLU(),

        #Conv2
        nn.BatchNorm2d(32),
        nn.Conv2d(32, 64, 5, padding=1),
        nn.MaxPool2d(2),
        nn.ReLU(),

        #Conv3
        nn.BatchNorm2d(64),
        nn.Conv2d(64, 128, 3, padding=1),
        nn.MaxPool2d(2),
        nn.ReLU(),

        #Conv4
        nn.BatchNorm2d(128),
        nn.Conv2d(128, 256, 2, padding=1),
        nn.MaxPool2d(2),
        nn.ReLU()
        )
```

```
            self.regressor = nn.Sequential(
            nn.Dropout(0.2),
            #FC6
            nn.BatchNorm1d(1024),
            nn.Linear(1024, l1),
            nn.ReLU(),

            nn.Dropout(0.2),
            #FC7
            nn.BatchNorm1d(l1),
            nn.Linear(l1,l2),
            nn.ReLU(),

            nn.BatchNorm1d(l2),
            nn.Linear(l2,1)
            )


        def forward(self,x):

            x = self.feature_extractor(x)
            x = self.regressor(x.view(x.shape[0],-1))
            return x


    cnnreg2=CNNRegressorV2(1024,512)
    cnnreg2=train_regressor_adam(cnnreg2,train_loader, test_loader, exp_name="test/CNNRegV2",lr=3e-4, weight_decay=0.1, epochs=400)
```

```
Epoch:  371 Loss:  0.17470424713158025
[8.47288609443e-05]
Epoch:  372 Loss:  0.14051249968569454
[8.47288609443e-05]
Epoch:  373 Loss:  0.19098412772504295
[8.47288609443e-05]
Epoch:  374 Loss:  0.10663254522695774
[8.47288609443e-05]
Epoch:  375 Loss:  0.10311564948500657
[8.47288609443e-05]
Epoch:  376 Loss:  0.10956070572137833
[8.47288609443e-05]
Epoch:  377 Loss:  0.10183337966843349
[8.47288609443e-05]
Epoch:  378 Loss:  0.12834388526474558
[8.47288609443e-05]
Epoch:  379 Loss:  0.11135923008366329
[8.47288609443e-05]
Epoch:  380 Loss:  0.198286466482209
[8.47288609443e-05]
Epoch:  381 Loss:  0.16482368075266118
[8.47288609443e-05]
Epoch:  382 Loss:  0.19785753837445888
[8.47288609443e-05]
Epoch:  383 Loss:  0.1257403471120974
[8.47288609443e-05]
Epoch:  384 Loss:  0.10691385062002554
[8.47288609443e-05]
Epoch:  385 Loss:  0.11942879601222713
[8.47288609443e-05]
Epoch:  386 Loss:  0.12515485141335464
[8.47288609443e-05]
Epoch:  387 Loss:  0.22472896408743975
[8.47288609443e-05]
Epoch:  388 Loss:  0.2152136513372747
[8.47288609443e-05]
Epoch:  389 Loss:  0.13050549550027382
[7.625597484987e-05]
Epoch:  390 Loss:  0.10872552453017817
[7.625597484987e-05]
Epoch:  391 Loss:  0.07901915380867516
[7.625597484987e-05]
Epoch:  392 Loss:  0.16797729090946475
[7.625597484987e-05]
Epoch:  393 Loss:  0.0904523037919184
[7.625597484987e-05]
Epoch:  394 Loss:  0.1006130939576684
[7.625597484987e-05]
Epoch:  395 Loss:  0.09901543997409867
[7.625597484987e-05]
Epoch:  396 Loss:  0.11601039876298207
[7.625597484987e-05]
Epoch:  397 Loss:  0.0858179599773593
[7.625597484987e-05]
Epoch:  398 Loss:  0.08759541500632356
[7.625597484987e-05]
Epoch:  399 Loss:  0.08907406195634748
[7.625597484987e-05]
```

```
test_predictions, labels_test = test_regressor(
 cnnreg2, test_loader)
print("MSE: ", mean_squared_error(labels_test, test_predictions))
```

```
     Labels:  [4.5   3.5   4.25  3.75  3.375 3.5   3.375 3.5   2.75  3.5   3.375 3.75
      2.625 4.25  3.5   3.    3.    3.625 3.375 3.375 4.25  4.    2.75  3.75 ]
     Labels:  [3.375 4.5   3.375 3.625 3.125 3.75  3.375 2.875 3.5   2.75  3.5   3.125
      3.5   3.375 2.875 3.5   2.875]
     MSE:  0.11280636091807936
```

```
class CNNRegressorV3(nn.Module):
    def __init__(self):
        super(CNNRegressorV3, self).__init__()

        self.feature_extractor = nn.Sequential(
        #Conv1
        nn.Conv2d(3, 32, 5, padding=3),
        nn.MaxPool2d(2),
        nn.ReLU(),

        #Conv2
        nn.BatchNorm2d(32),
        nn.Conv2d(32, 64, 5, padding=3),
        nn.MaxPool2d(2),
        nn.ReLU(),

        #Conv3
        nn.BatchNorm2d(64),
        nn.Conv2d(64, 128, 3, padding=1),
        nn.MaxPool2d(2),
        nn.ReLU(),

        #Conv4
        nn.BatchNorm2d(128),
        nn.Conv2d(128, 256, 3, padding=1),
        nn.MaxPool2d(2),
        nn.ReLU(),

        #Conv5
        nn.BatchNorm2d(256),
        nn.Conv2d(256, 512, 3, padding=1),
        nn.MaxPool2d(2),
        nn.ReLU()
        )

        self.regressor = nn.Sequential(
        nn.Dropout(0.2),
        #FC6
        nn.BatchNorm1d(512),
        nn.Linear(512, 1024),
        nn.ReLU(),

        nn.Dropout(0.2),
        #FC7
        nn.BatchNorm1d(1024),
        nn.Linear(1024,128),
        nn.ReLU(),

        nn.BatchNorm1d(128),
        nn.Linear(128,1)
        )


    def forward(self,x):

        x = self.feature_extractor(x)
        x = self.regressor(x.view(x.shape[0],-1))
        return x


cnn3=CNNRegressorV3()
cnn3=train_regressor(cnn3,train_loader, test_loader, exp_name="test/CNNRegV3",lr=1e-3, momentum=0.98, weight_decay=0.1, epochs=400)
```

```
Epoch:  352 Loss:  0.16822212898149724
Epoch:  353 Loss:  0.13466176990328765
Epoch:  354 Loss:  0.30599494242086644
Epoch:  355 Loss:  0.19847357345790398
Epoch:  356 Loss:  0.16088465364967905
Epoch:  357 Loss:  0.3087317056772185
Epoch:  358 Loss:  0.19841978462730966
Epoch:  359 Loss:  0.13735021469069691
Epoch:  360 Loss:  0.13974572536421986
Epoch:  361 Loss:  0.3482363107727795
Epoch:  362 Loss:  0.27158645376926516
Epoch:  363 Loss:  0.20657172181257388
Epoch:  364 Loss:  0.16693408496496154
Epoch:  365 Loss:  0.13828250947521953
Epoch:  366 Loss:  0.17083643740270196
Epoch:  367 Loss:  0.22817434261484845
Epoch:  368 Loss:  0.20436275768570783
Epoch:  369 Loss:  0.12060176344906412
Epoch:  370 Loss:  0.15471804087481847
Epoch:  371 Loss:  0.1746753603219986
Epoch:  372 Loss:  0.1944162387673448
Epoch:  373 Loss:  0.13091458561943797
Epoch:  374 Loss:  0.11328255476021185
Epoch:  375 Loss:  0.16692006587982178
Epoch:  376 Loss:  0.18387260415205142
Epoch:  377 Loss:  0.19798134230985875
Epoch:  378 Loss:  0.21720954194301512
Epoch:  379 Loss:  0.15457531046576617
Epoch:  380 Loss:  0.15211379019225515
Epoch:  381 Loss:  0.2876147464281175
Epoch:  382 Loss:  0.12798384740585234
Epoch:  383 Loss:  0.3379567323661432
Epoch:  384 Loss:  0.2861990888671177
Epoch:  385 Loss:  0.23607765683313695
Epoch:  386 Loss:  0.19673539080270908
Epoch:  387 Loss:  0.15257735005239162
Epoch:  388 Loss:  0.14819527126666976
Epoch:  389 Loss:  0.11682546792960749
Epoch:  390 Loss:  0.1434239950485346
Epoch:  391 Loss:  0.1774161563413899
Epoch:  392 Loss:  0.13672637775903795
Epoch:  393 Loss:  0.17663597697164954
Epoch:  394 Loss:  0.13649396271240422
Epoch:  395 Loss:  0.15050640811280505
Epoch:  396 Loss:  0.3022797623785531
Epoch:  397 Loss:  0.22390093345467638
Epoch:  398 Loss:  0.13575714063353655
Epoch:  399 Loss:  0.17020487858027947
```

```
test_predictions, labels_test = test_regressor(
 cnn3, test_loader)
print("MSE: ", mean_squared_error(labels_test, test_predictions))
```

```
Labels:  [4.25  2.75  3.5   3.5   3.375 3.5   4.25  4.5   3.75  4.25  3.5   3.625
 3.75  3.125 2.75  2.625 2.875 3.5   3.625 3.375 3.5   2.75  3.5   2.875]
Labels:  [3.375 4.    3.375 2.875 3.5   3.375 4.5   3.375 3.75  3.    3.5   3.
 3.75  3.375 3.375 3.125 3.375]
MSE:  0.12725472513973524
```

```
cnn3adam=CNNRegressorV3()
cnn3adam=train_regressor_adam(cnn3adam,train_loader, test_loader, exp_name="test/CNNRegV3Adam",lr=1e-3, weight_decay=0.1, epochs=400)
```

```
[0.00028242953648100003]
Epoch:  385 Loss:  0.1576879275281255
[0.00028242953648100003]
Epoch:  386 Loss:  0.08132349518014163
[0.00028242953648100003]
Epoch:  387 Loss:  0.12485068600352217
[0.00028242953648100003]
Epoch:  388 Loss:  0.10135763683697073
[0.00028242953648100003]
Epoch:  389 Loss:  0.17103271186351776
[0.00025418658283290005]
Epoch:  390 Loss:  0.08862604509766508
[0.00025418658283290005]
Epoch:  391 Loss:  0.07014073249770374
[0.00025418658283290005]
Epoch:  392 Loss:  0.12958932940552875
[0.00025418658283290005]
Epoch:  393 Loss:  0.13919941971941693
[0.00025418658283290005]
Epoch:  394 Loss:  0.09988452839415247
[0.00025418658283290005]
Epoch:  395 Loss:  0.13930641877941968
[0.00025418658283290005]
Epoch:  396 Loss:  0.09444017671957249
[0.00025418658283290005]
Epoch:  397 Loss:  0.07263006597030454
[0.00025418658283290005]
Epoch:  398 Loss:  0.08581903976638143
[0.00025418658283290005]
Epoch:  399 Loss:  0.08009430875138539
[0.00025418658283290005]
```

```
test_predictions, labels_test = test_regressor(
 cnn3adam, test_loader)
print("MSE: ", mean_squared_error(labels_test, test_predictions))
```

```
Labels:  [2.875 3.375 3.5   4.5   2.75  3.75  3.5   3.375 3.5   4.25  4.25  3.375
 3.625 3.625 3.375 2.75  3.75  3.5   3.5   2.875 3.375 3.375 3.    3.375]
Labels:  [3.5   4.25  3.5   2.875 3.75  3.75  3.125 3.5   3.    3.125 2.75  2.625
 4.    4.5   3.375 3.5   3.375]
MSE:  0.06846286193142699
```

Demo del funzionamento della rete neurale

A questo punto si importa Gradio e si effettua la costruzione della demo dimostrativa per valutare le prestazioni del modello. La funzione predict permette di effettuare le predizioni sugli input resizati e trasformati sulla base del modello. Il comando gr.Interface genera invece la GUI della demo dividendola in due finestre:

- La prima rappresenta la zona in cui importare l'input;
- La seconda permette di visualizzare la label predetta

```
import gradio as gr

def predict(image):
  image = transforms.Compose([transforms.Resize((32,42)),
                              transforms.ToTensor(), transforms.Normalize(m,s)])(image)
  with torch.no_grad():
    prediction = model(image.unsqueeze(0)).float()
    return prediction

model=CNNRegressorV3()

model.load_state_dict(torch.load("test/CNNRegV3Adam-400.pth"))
model.eval()
demo = gr.Interface(predict, inputs=gr.Image(type="pil"),outputs=gr.Textbox(label='Predicted Score'))
demo.launch(debug=True)
```

```
Colab notebook detected. This cell will run indefinitely so that you can see errors and logs. To turn o
Note: opening Chrome Inspector may crash demo inside Colab notebooks.

To create a public link, set `share=True` in `launch()`.
Keyboard interruption in main thread... closing server.
```

```
def predict(image):
  image = transforms.Compose([transforms.Resize((32,42)), transforms.Grayscale(),
                              transforms.ToTensor(), transforms.Normalize(m,s)])(image)
  with torch.no_grad():
    prediction = model(image.unsqueeze(0)).float()
    return prediction
```

```
model=MiniAlexNetV3()

model.load_state_dict(torch.load("test/MiniAlexNetV3-400.pth"))
model.eval()
demo = gr.Interface(predict, inputs=gr.Image(type="pil"),outputs=gr.Textbox(label='Predicted Score'))
demo.launch(debug=True)
```

```
Colab notebook detected. This cell will run indefinitely so that you can see errors and logs. To turn o
Note: opening Chrome Inspector may crash demo inside Colab notebooks.

To create a public link, set `share=True` in `launch()`.
Keyboard interruption in main thread... closing server.
```

Valuazione dei risultati tramite interfaccia TensorBoard

```
%load_ext tensorboard
%tensorboard --logdir logs
```