PROIECT INVATARE AUTOMATA

Cats vs Dogs Classification using CNN

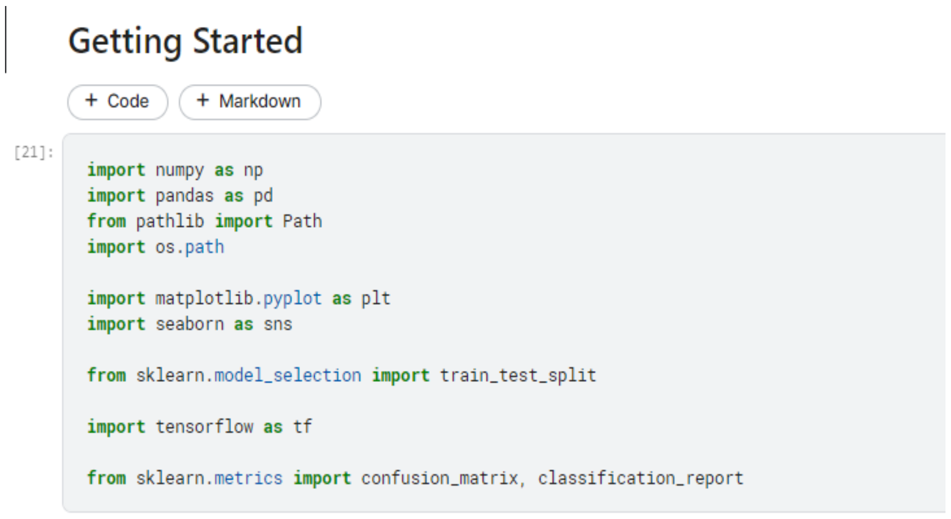
Student: Bucă-Ghebaură Elizabetha Alexandrina

Specializare: Calculatoare Romana

Grupa: 4.S1 A

Din foarte multe imagini cu pisici si caini, am incercat sa clasific ce animal este prezent intr-o anumita imagine, folosind *CNN -> Convolutional Neural Networks.*

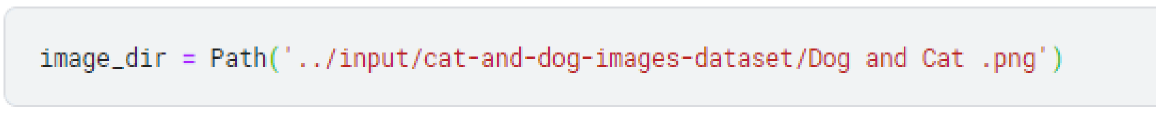
Pentru inceput, am importat librariile necesare pentru functionarea programului:


Getting Started

```
import numpy as np
import pandas as pd
from pathlib import Path
import os.path

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split

import tensorflow as tf

from sklearn.metrics import confusion_matrix, classification_report
```

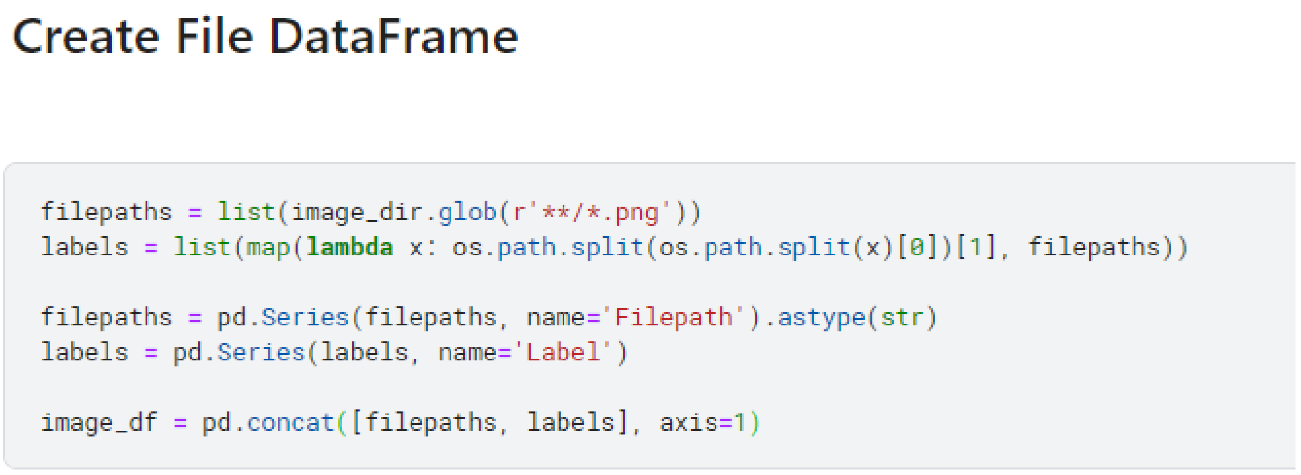
Ne vor trebui librariile numpy, pandas, os, matplotlib, seaborn, sklearn, tensorflow.

Am specificat apoi Path-ul dataset-ului:



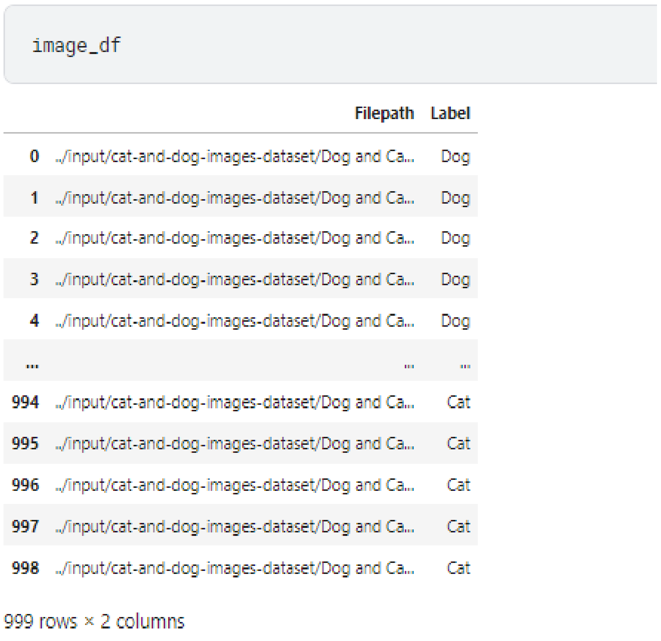
```
image_dir = Path('../input/cat-and-dog-images-dataset/Dog and Cat .png')
```

Mai departe, am initializat *filepaths* cu o lista de imagini in format *.png* si apoi m-am folosit de libraria *os.path.split*, utilizata pentru a imparti numele path-ului intr-o pereche head si tail. Tail este ultima componenta a numele path-ului, iar head este inceputul path-ului. Adica vom avea drept head-> */input/cat-and-dog-images-dataset/*, iar tail -> *Dog and Cat .png.*


Create File DataFrame

```
filepaths = list(image_dir.glob(r'**/*.png'))
labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], filepaths))

filepaths = pd.Series(filepaths, name='Filepath').astype(str)
labels = pd.Series(labels, name='Label')

image_df = pd.concat([filepaths, labels], axis=1)
```

Am afisat apoi efectiv dataset-ul in mod tabelar, in randuri si coloane. Avem 999 linii si 2 coloane (pisici si caini).

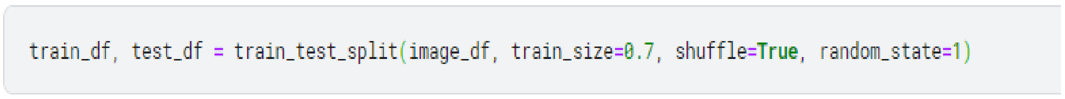


```
image_df
```

	Filepath	Label
0	../input/cat-and-dog-images-dataset/Dog and Ca...	Dog
1	../input/cat-and-dog-images-dataset/Dog and Ca...	Dog
2	../input/cat-and-dog-images-dataset/Dog and Ca...	Dog
3	../input/cat-and-dog-images-dataset/Dog and Ca...	Dog
4	../input/cat-and-dog-images-dataset/Dog and Ca...	Dog
...	...	...
994	../input/cat-and-dog-images-dataset/Dog and Ca...	Cat
995	../input/cat-and-dog-images-dataset/Dog and Ca...	Cat
996	../input/cat-and-dog-images-dataset/Dog and Ca...	Cat
997	../input/cat-and-dog-images-dataset/Dog and Ca...	Cat
998	../input/cat-and-dog-images-dataset/Dog and Ca...	Cat

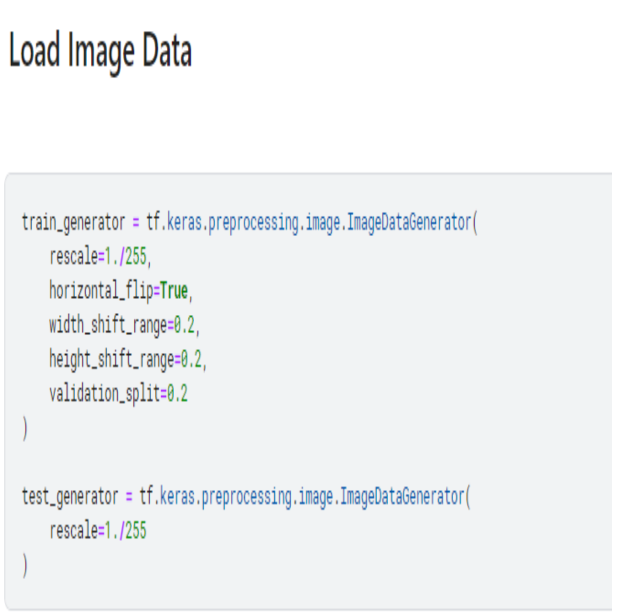
999 rows × 2 columns

Am facut split pentru train si test:



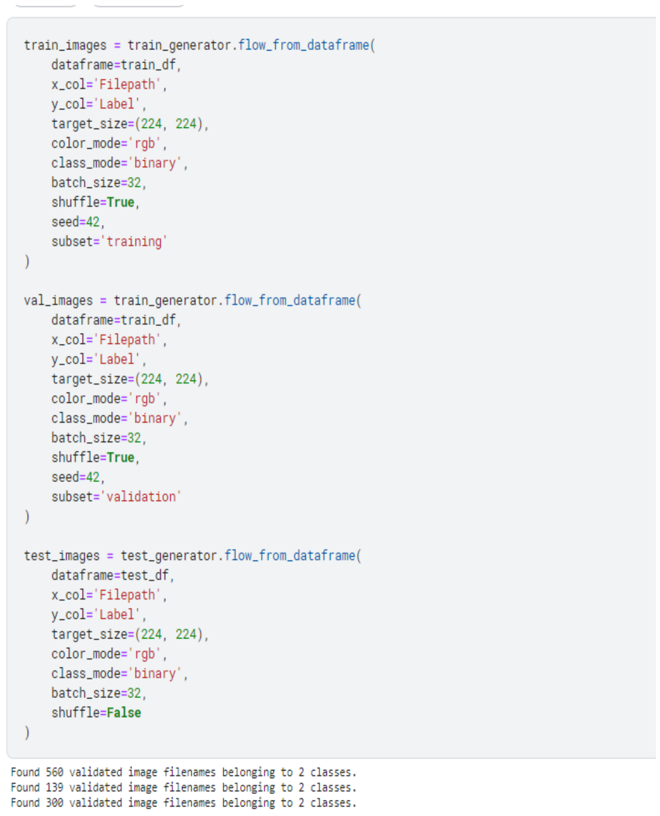
```
train_df, test_df = train_test_split(image_df, train_size=0.7, shuffle=True, random_state=1)
```

Mai departe, am incarcat practic datele dataset-ului si am gasit 560, 139 si 300 de imagini valide apartinand a 2 clase.



Load Image Data

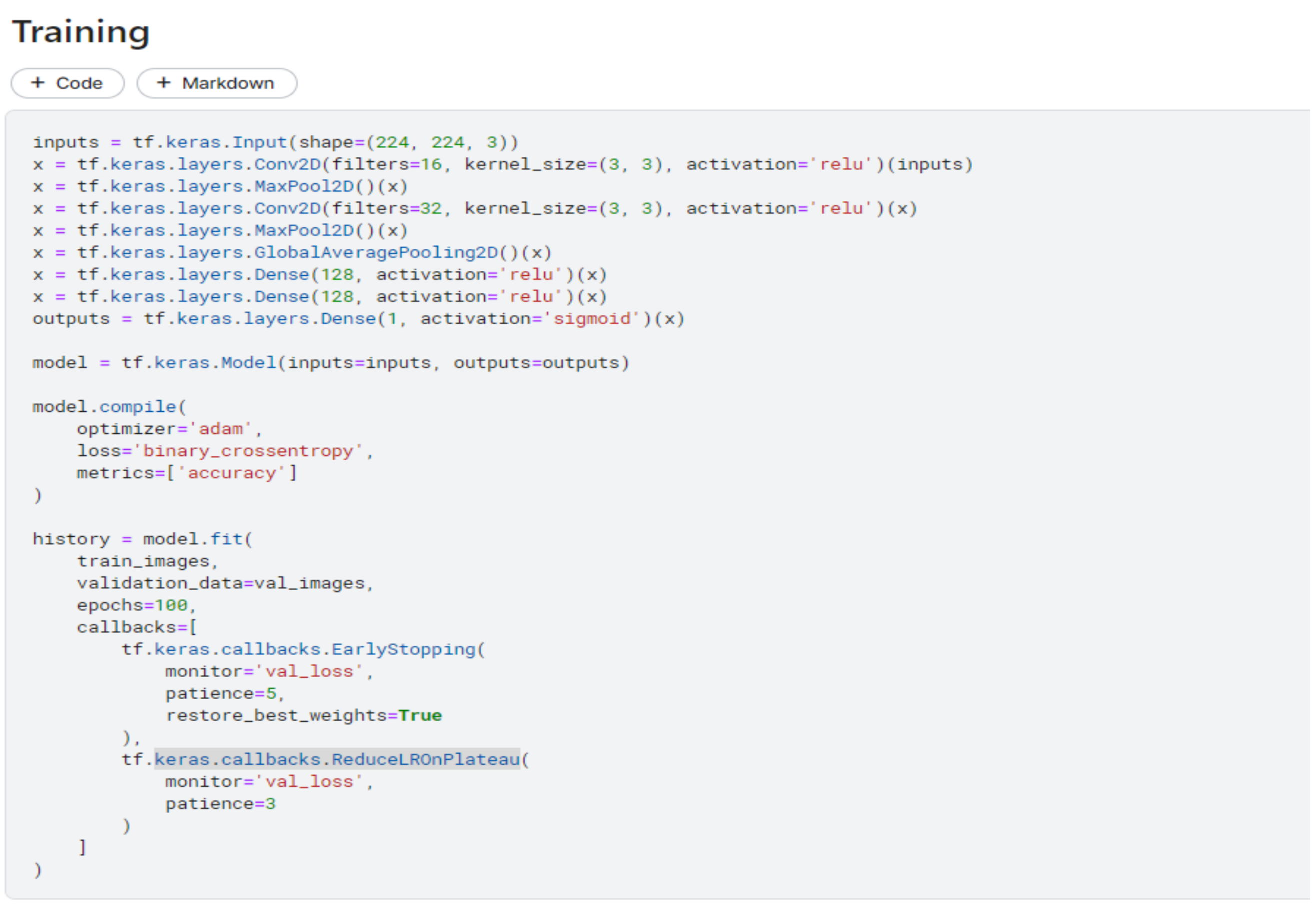
```
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    width_shift_range=0.2,
    height_shift_range=0.2,
    validation_split=0.2
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255
)
```



```
train_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    seed=42,
    subset='training'
)

val_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    seed=42,
    subset='validation'
)

test_images = test_generator.flow_from_dataframe(
    dataframe=test_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='binary',
    batch_size=32,
    shuffle=False
)
```

```
Found 560 validated image filenames belonging to 2 classes.
Found 139 validated image filenames belonging to 2 classes.
Found 300 validated image filenames belonging to 2 classes.
```

Mai departe, am antrenat modelele.



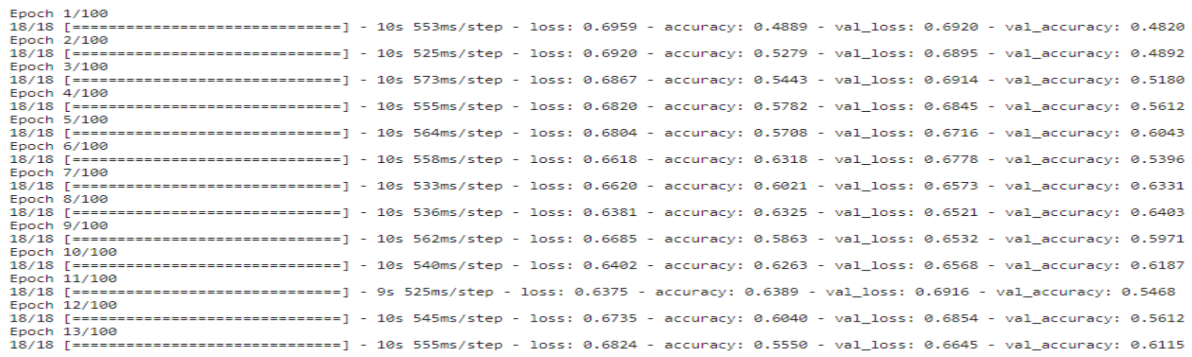
```
inputs = tf.keras.Input(shape=(224, 224, 3))
x = tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu')(inputs)
x = tf.keras.layers.MaxPool2D()(x)
x = tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu')(x)
x = tf.keras.layers.MaxPool2D()(x)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(128, activation='relu')(x)
x = tf.keras.layers.Dense(128, activation='relu')(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_images,
    validation_data=val_images,
    epochs=100,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=5,
            restore_best_weights=True
        ),
        tf.keras.callbacks.ReduceLROnPlateau(
            monitor='val_loss',
            patience=3
        )
    ]
)
```

Un *Epoch* este atunci cand toate datele de training sunt utilizate simultan si sunt definite ca numarul total de iteratii alte tuturor datelor de training sau sa zicem ca s-ar putea defini ca numarul de treceri pe care le face un set de date de training in jurul unui algoritm.

EarlyStopping ne permite sa „masuram" performanta programului, si odata ce va fi apelat, va opri procesul de training. Este practic o oprire timpurie a training-ului, printr-un apel invers.

ReduceLROnPlateau se va apela atunci cand o valoare nu se mai imbunatateste. Acest apel monitorizeaza o cantitate, si daca nu se observa nicio imbunatatire pentru un numar de „rabdare" de epchos, atunci rata de invatare este redusa. Astfel, am observat, ca nu mereu vom avea acelasi numar de epochs.

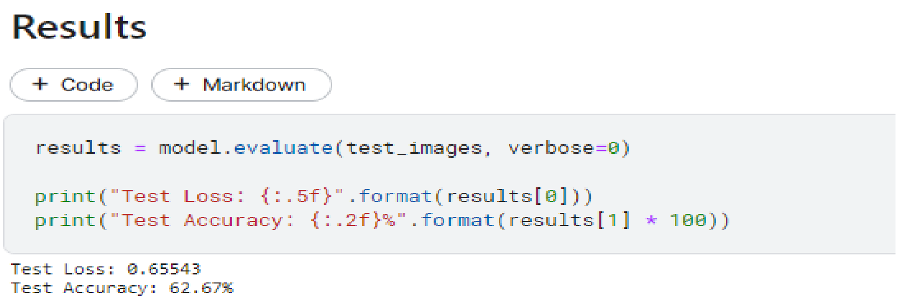
Output pentru epochs:



```
Epoch 1/100
18/18 [==============================] - 10s 553ms/step - loss: 0.6959 - accuracy: 0.4889 - val_loss: 0.6920 - val_accuracy: 0.4820
Epoch 2/100
18/18 [==============================] - 10s 525ms/step - loss: 0.6920 - accuracy: 0.5279 - val_loss: 0.6895 - val_accuracy: 0.4892
Epoch 3/100
18/18 [==============================] - 10s 573ms/step - loss: 0.6867 - accuracy: 0.5443 - val_loss: 0.6914 - val_accuracy: 0.5180
Epoch 4/100
18/18 [==============================] - 10s 555ms/step - loss: 0.6820 - accuracy: 0.5782 - val_loss: 0.6845 - val_accuracy: 0.5612
Epoch 5/100
18/18 [==============================] - 10s 564ms/step - loss: 0.6804 - accuracy: 0.5708 - val_loss: 0.6716 - val_accuracy: 0.6043
Epoch 6/100
18/18 [==============================] - 10s 558ms/step - loss: 0.6618 - accuracy: 0.6318 - val_loss: 0.6778 - val_accuracy: 0.5396
Epoch 7/100
18/18 [==============================] - 10s 533ms/step - loss: 0.6620 - accuracy: 0.6021 - val_loss: 0.6573 - val_accuracy: 0.6331
Epoch 8/100
18/18 [==============================] - 10s 536ms/step - loss: 0.6381 - accuracy: 0.6325 - val_loss: 0.6521 - val_accuracy: 0.6403
Epoch 9/100
18/18 [==============================] - 10s 562ms/step - loss: 0.6685 - accuracy: 0.5863 - val_loss: 0.6532 - val_accuracy: 0.5971
Epoch 10/100
18/18 [==============================] - 10s 540ms/step - loss: 0.6402 - accuracy: 0.6263 - val_loss: 0.6568 - val_accuracy: 0.6187
Epoch 11/100
18/18 [==============================] - 9s 525ms/step - loss: 0.6375 - accuracy: 0.6389 - val_loss: 0.6916 - val_accuracy: 0.5468
Epoch 12/100
18/18 [==============================] - 10s 545ms/step - loss: 0.6735 - accuracy: 0.6040 - val_loss: 0.6854 - val_accuracy: 0.5612
Epoch 13/100
18/18 [==============================] - 10s 555ms/step - loss: 0.6824 - accuracy: 0.5550 - val_loss: 0.6645 - val_accuracy: 0.6115
```

Aici ni se afiseaza in cate secunde se calculeaza, ce loss avem, ce acuratete, valoarea pentru loss si valoarea pentru acuratete.

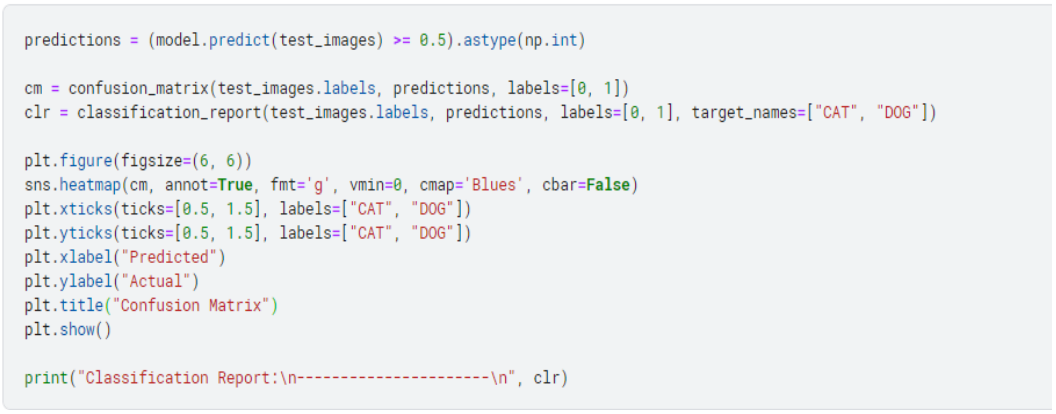
Rezultatele in urma testarii imaginilor: ***Test Loss****: 0.65543*; ***Test Accuracy****: 62.67%*



```
results = model.evaluate(test_images, verbose=0)

print("Test Loss: {:.5f}".format(results[0]))
print("Test Accuracy: {:.2f}%".format(results[1] * 100))
```

```
Test Loss: 0.65543
Test Accuracy: 62.67%
```

Urmeaza predictia:



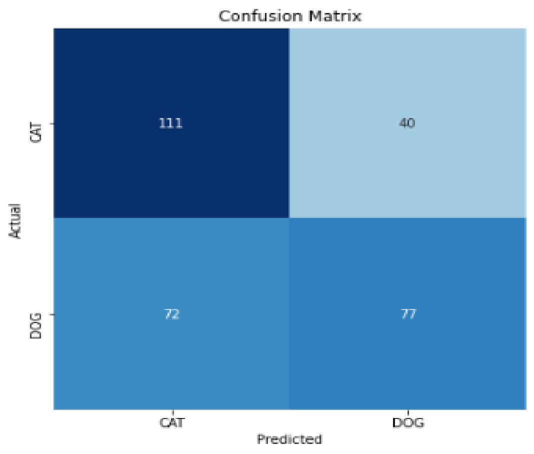
```
predictions = (model.predict(test_images) >= 0.5).astype(np.int)

cm = confusion_matrix(test_images.labels, predictions, labels=[0, 1])
clr = classification_report(test_images.labels, predictions, labels=[0, 1], target_names=["CAT", "DOG"])

plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues', cbar=False)
plt.xticks(ticks=[0.5, 1.5], labels=["CAT", "DOG"])
plt.yticks(ticks=[0.5, 1.5], labels=["CAT", "DOG"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

print("Classification Report:\n----------------------\n", clr)
```

Aici avem o matrice cu ce s-a prezis, si care sunt de fapt datele actuale corecte:

Actual: 111 cat, 72 dog

Predicted: 40 cat, 77 dog

Adica, in final am avea o precizie de 0.61 pentru pisici si 0.66 pentru caini.



```
Classification Report:
----------------------
              precision    recall  f1-score   support

         CAT       0.61      0.74      0.66       151
         DOG       0.66      0.52      0.58       149

    accuracy                           0.63       300
   macro avg       0.63      0.63      0.62       300
weighted avg       0.63      0.63      0.62       300
```

REFERINTE:

1. https://www.datacamp.com/tutorial/convolutional-neural-networks-python
2. https://www.geeksforgeeks.org/cat-dog-classification-using-convolutional-neural-network-in-python/
3. https://www.kaggle.com/code/kashit/cat-and-dog-classification-with-cnn/notebook
4. https://www.kaggle.com/code/gpreda/cats-or-dogs-using-cnn-with-transfer-learning/notebook
5. https://www.kaggle.com/code/mihailsavushkin/homework-2
6. https://www.kaggle.com/datasets/arpitjain007/dog-vs-cat-fastai/code
7. https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset/code
8. https://www.kaggle.com/competitions/petfinder-pawpularity-score/discussion/274259
9. https://www.kaggle.com/code/bulentsiyah/dogs-vs-cats-classification-vgg16-fine-tuning
10. https://www.youtube.com/watch?v=FLf5qmSOkwU

Concluzii

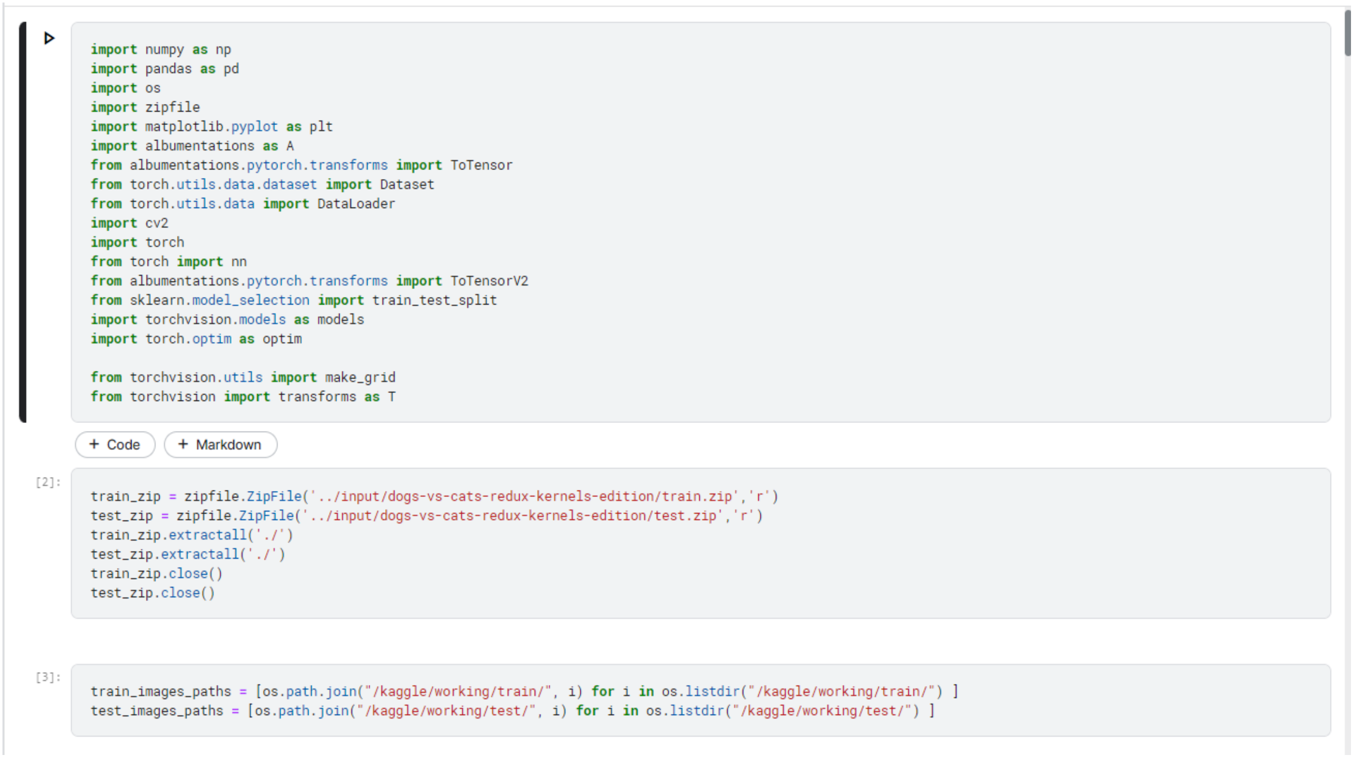
Facand research pentru acest proiect pe internet, am observat ca exista foarte multe metode de rezolvare a acestei teme. Initial incercasem sa citesc toate datele, facand cate un label pentru fiecare poza si incercand sa fac o identificare pentru ambele animale. Astfel, as fi parcurs intr-un fel aceiasi pasi pe care i-am parcurs si in aceasta varianta finala prezentata mai sus, doar ca in alte moduri. Exemplu de varianta pe care incercasem sa o fac: https://www.kaggle.com/code/rajansahu/dog-vs-cat

In final, consider ca am invatat multe lucruri interesante facand research pentru acest proiect (chiar daca toate cunostintele pe care le-am acumulat nu se afla in totalitate in varianta finala prezentata mai sus) si tema pe care mi-am ales-o m-a facut sa fiu cat mai curioasa - sa aflu mai multe moduri de rezolvare a problemei si sa observ cum functioneaza diferite metode folosite pentru solutionarea temei.

In continuare, pe viitor, voi incerca sa rezolv in modul/varianta despre care vorbeam mai sus (cel cu citirea efectiva a pozelor, afisare a anumitor poze si incercarea de a „ghici” ce fel de animal se afla acolo).

VARIANTA 2

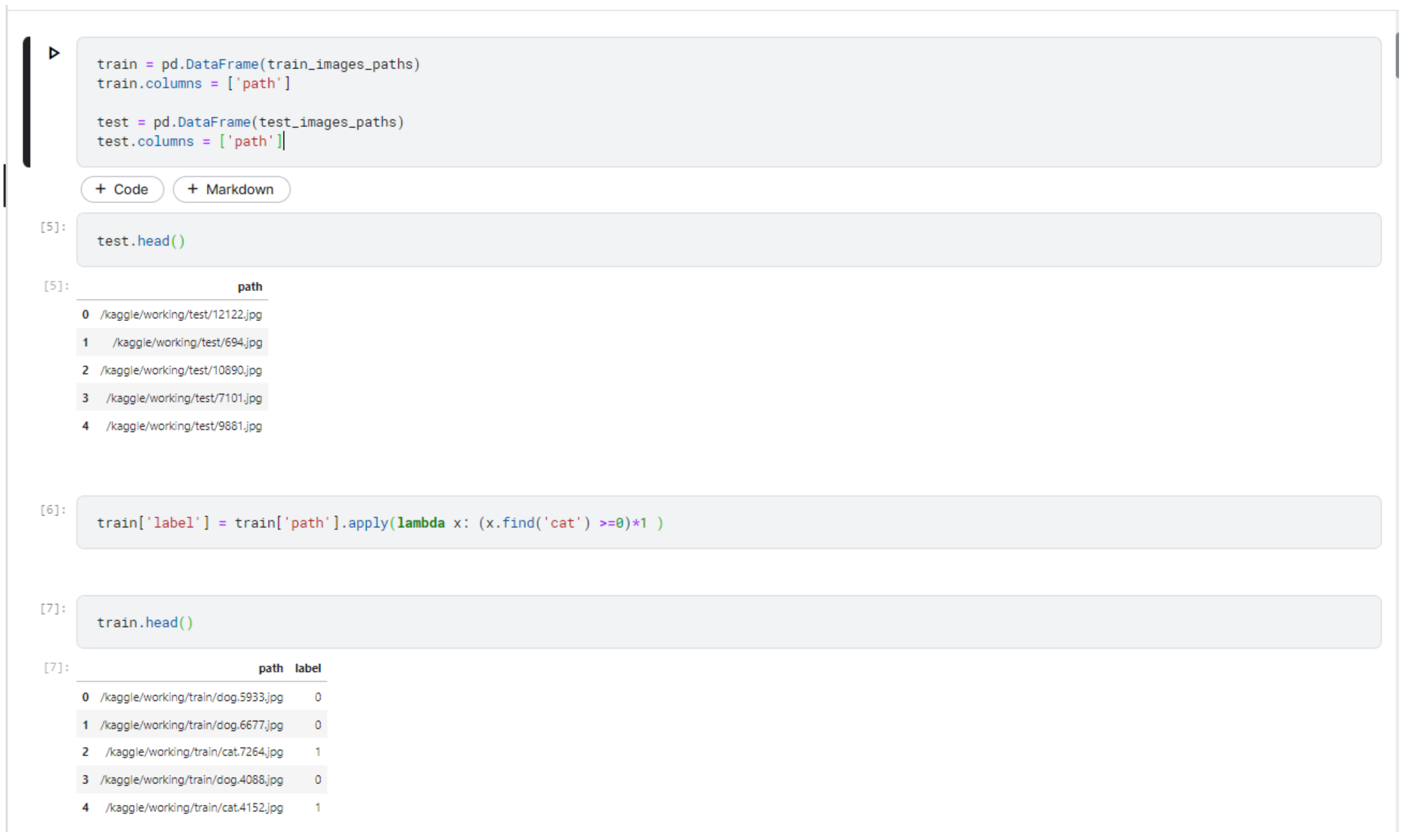
Importare librarii si citire dataset



```
import numpy as np
import pandas as pd
import os
import zipfile
import matplotlib.pyplot as plt
import albumentations as A
from albumentations.pytorch.transforms import ToTensor
from torch.utils.data.dataset import Dataset
from torch.utils.data import DataLoader
import cv2
import torch
from torch import nn
from albumentations.pytorch.transforms import ToTensorV2
from sklearn.model_selection import train_test_split
import torchvision.models as models
import torch.optim as optim

from torchvision.utils import make_grid
from torchvision import transforms as T
```

```
train_zip = zipfile.ZipFile('../input/dogs-vs-cats-redux-kernels-edition/train.zip','r')
test_zip = zipfile.ZipFile('../input/dogs-vs-cats-redux-kernels-edition/test.zip','r')
train_zip.extractall('./')
test_zip.extractall('./')
train_zip.close()
test_zip.close()
```

```
train_images_paths = [os.path.join("/kaggle/working/train/", i) for i in os.listdir("/kaggle/working/train/") ]
test_images_paths = [os.path.join("/kaggle/working/test/", i) for i in os.listdir("/kaggle/working/test/") ]
```



```
train = pd.DataFrame(train_images_paths)
train.columns = ['path']

test = pd.DataFrame(test_images_paths)
test.columns = ['path']
```

```
test.head()
```

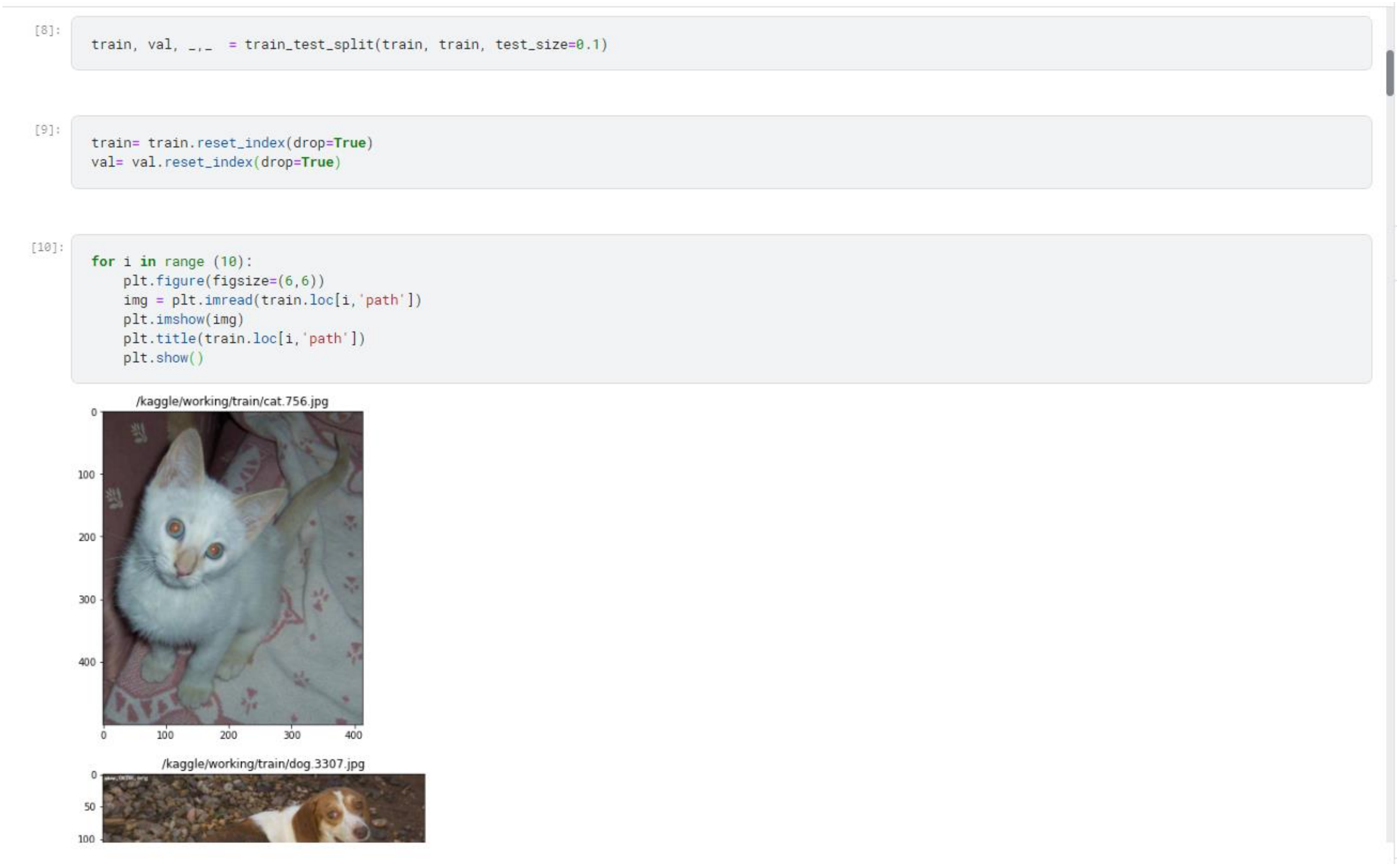
```
                        path
0  /kaggle/working/test/12122.jpg
1    /kaggle/working/test/694.jpg
2  /kaggle/working/test/10890.jpg
3   /kaggle/working/test/7101.jpg
4   /kaggle/working/test/9881.jpg
```

```
train['label'] = train['path'].apply(lambda x: (x.find('cat') >=0)*1 )
```

```
train.head()
```

```
                              path  label
0  /kaggle/working/train/dog.5933.jpg      0
1  /kaggle/working/train/dog.6677.jpg      0
2   /kaggle/working/train/cat.7264.jpg     1
3  /kaggle/working/train/dog.4088.jpg      0
4   /kaggle/working/train/cat.4152.jpg     1
```

Train si Test split, afisare primele 10 poze din dataset



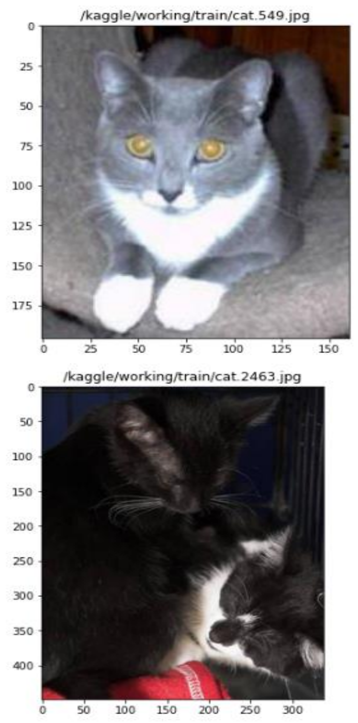
```
[8]:  train, val, _,_  = train_test_split(train, train, test_size=0.1)
```

```
[9]:  train= train.reset_index(drop=True)
      val= val.reset_index(drop=True)
```

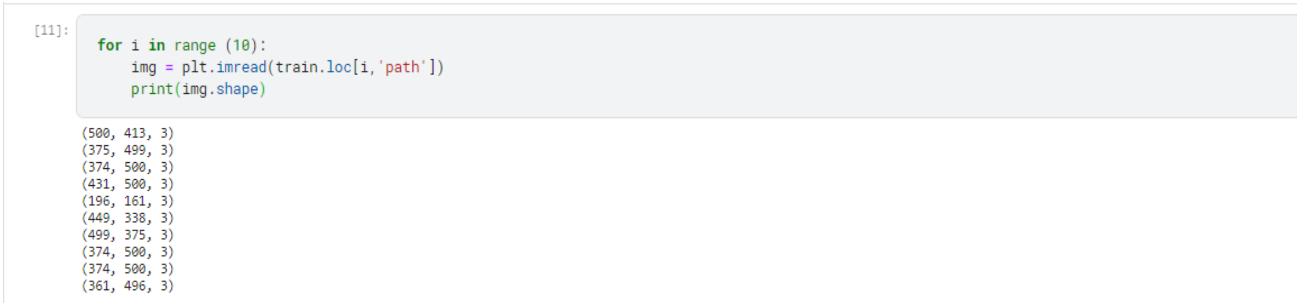
```
[10]: for i in range (10):
          plt.figure(figsize=(6,6))
          img = plt.imread(train.loc[i,'path'])
          plt.imshow(img)
          plt.title(train.loc[i,'path'])
          plt.show()
```

/kaggle/working/train/cat.756.jpg
/kaggle/working/train/dog.3307.jpg

/kaggle/working/train/dog.3307.jpg
/kaggle/working/train/cat.4090.jpg
/kaggle/working/train/cat.6762.jpg


/kaggle/working/train/cat.549.jpg
/kaggle/working/train/cat.2463.jpg

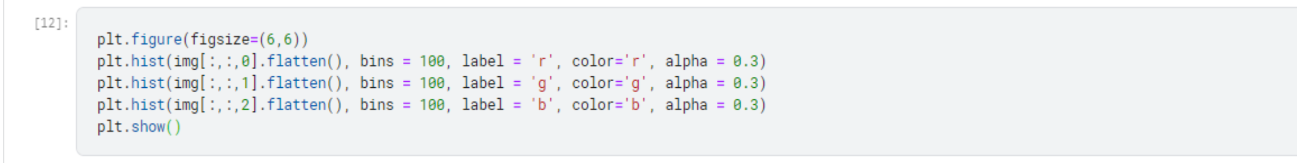
Afisare path primele 10 imagiini si histograma



[11]:

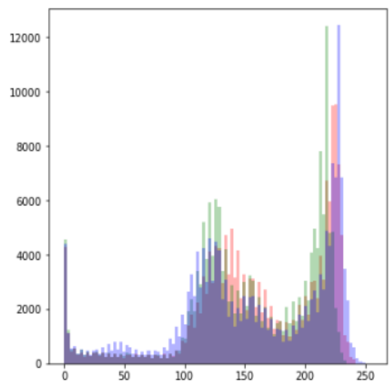
```
for i in range (10):
    img = plt.imread(train.loc[i,'path'])
    print(img.shape)
```

```
(500, 413, 3)
(375, 499, 3)
(374, 500, 3)
(431, 500, 3)
(196, 161, 3)
(449, 338, 3)
(499, 375, 3)
(374, 500, 3)
(374, 500, 3)
(361, 496, 3)
```

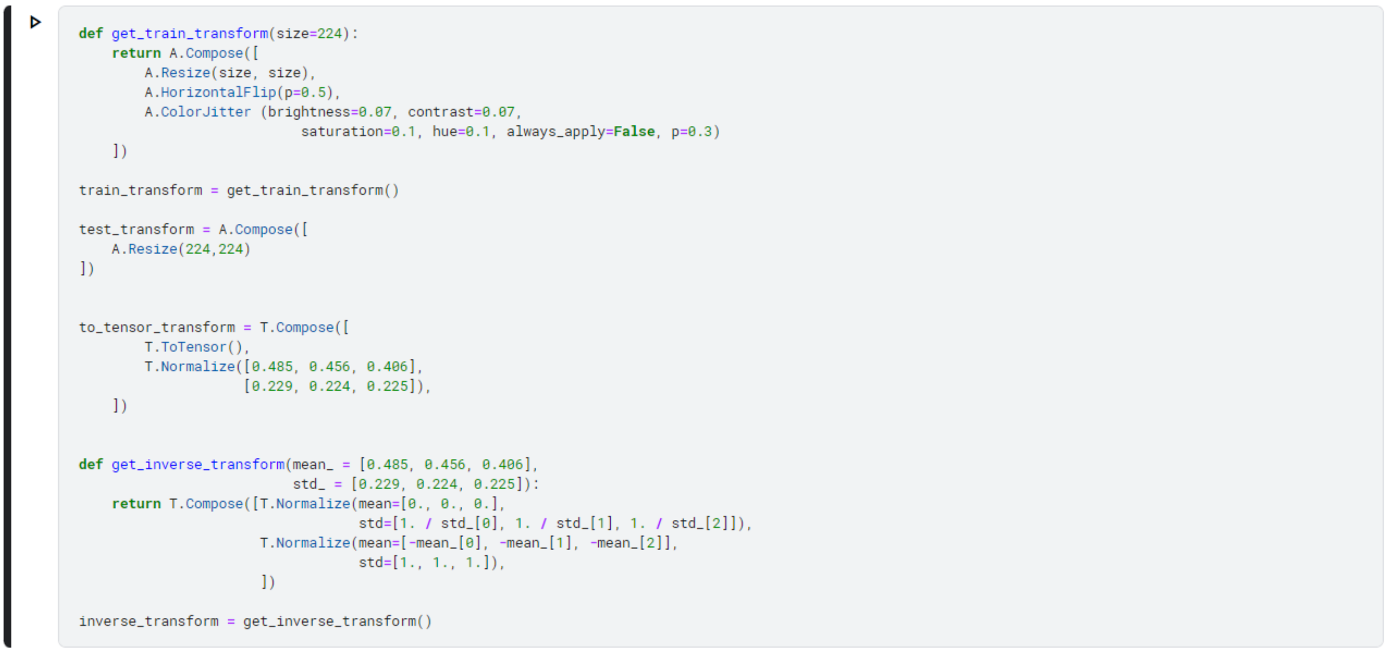


[12]:

```
plt.figure(figsize=(6,6))
plt.hist(img[:,:,0].flatten(), bins = 100, label = 'r', color='r', alpha = 0.3)
plt.hist(img[:,:,1].flatten(), bins = 100, label = 'g', color='g', alpha = 0.3)
plt.hist(img[:,:,2].flatten(), bins = 100, label = 'b', color='b', alpha = 0.3)
plt.show()
```



Aceleasi functii din varianta 1, dar scrise in alt mod



```
def get_train_transform(size=224):
    return A.Compose([
        A.Resize(size, size),
        A.HorizontalFlip(p=0.5),
        A.ColorJitter (brightness=0.07, contrast=0.07,
                       saturation=0.1, hue=0.1, always_apply=False, p=0.3)
    ])

train_transform = get_train_transform()

test_transform = A.Compose([
    A.Resize(224,224)
])


to_tensor_transform = T.Compose([
        T.ToTensor(),
        T.Normalize([0.485, 0.456, 0.406],
                    [0.229, 0.224, 0.225]),
    ])


def get_inverse_transform(mean_ = [0.485, 0.456, 0.406],
                          std_ = [0.229, 0.224, 0.225]):
    return T.Compose([T.Normalize(mean=[0., 0., 0.],
                                  std=[1. / std_[0], 1. / std_[1], 1. / std_[2]]),
                      T.Normalize(mean=[-mean_[0], -mean_[1], -mean_[2]],
                                  std=[1., 1., 1.]),
                      ])

inverse_transform = get_inverse_transform()
```

[14]:
```
class TrainDataset(Dataset):
    def __init__(self, train, transform=None, is_test= False ):
        self.X = train['path']
        self.is_test = is_test
        self.transform = transform
        if not self.is_test:
            self.y = train['label']

    def __len__(self):
        return len(self.X)

    def __getitem__(self, index):
        image = cv2.imread(self.X[index])
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        if self.transform:
                image = self.transform(image=image)['image'] #.astype(np.float32)

        if self.is_test:
             return   to_tensor_transform(image) #.permute(2, 0 ,1).float()
        else:
            label = self.y[index]
            return  to_tensor_transform(image) , label  # .permute(2, 0 ,1).float()

    # H , W , C
    # C , H, W
```

[15]:
```
train_dataset = TrainDataset(train, train_transform)
val_dataset = TrainDataset(val, train_transform)

train_dataloader =   DataLoader(train_dataset, batch_size = 16, shuffle = True)
val_dataloader =   DataLoader(val_dataset, batch_size = 16, shuffle = False)
```

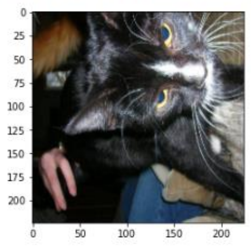
Functie afisare imagine

[16]:
```
def _imshow(img):
    print(img.shape)
    img = inverse_transform(img) #    # unnormalize
    npimg = img.numpy()
    npimg = np.transpose(npimg, (1, 2, 0))
    plt.figure(figsize=(20,20))
    plt.imshow(npimg)
    plt.show()
```

[17]:
```
dataiter = iter(train_dataloader)
images, labels = dataiter.next()
```

[18]:
```
#look at single image
plt.imshow(inverse_transform(images[0]).permute(1,2,0).numpy())
```

[18]:
```
<matplotlib.image.AxesImage at 0x7f2415300410>
```



Grid de imagini pentru batch (numarul de exemple de training-uri utilizate intr-o singura iteratie)

```
[19]:
#make image grid for batch
dataiter = iter(train_dataloader)
images, labels = dataiter.next()
_imshow(make_grid(images))
```

```
torch.Size([3, 454, 1810])
```

```
[20]:
test_dataset = TrainDataset(test, test_transform, is_test=True)
test_dataloader =  DataLoader(test_dataset, batch_size = 16, shuffle = False)
```

Resnet

```
[22]:
resnet = models.resnet101(pretrained = True)
```

```
Downloading: "https://download.pytorch.org/models/resnet101-5d3b4d8f.pth" to /root/.cache/torch/hub/checkpoints/resnet101-5d3b4d8f.pth
100%                170M/170M [00:08<00:00, 24.1MB/s]
```

```
[23]:
resnet
```

```
[23]: ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
```

```
      (22): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
    )
    (layer4): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
    )
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=2048, out_features=1000, bias=True)
  )
```

[24]:

```
# transfer learning
for param in resnet.parameters():
    param.requires_grad=False
```

[25]:

```
class NN(nn.Module):
    def __init__(self, resnet_pretrained):
        super().__init__()
        self.resnet_pretrained = resnet_pretrained
        self.fc1 = nn.Linear(1000, 2)

    def forward(self, x):
        x = torch.relu(self.resnet_pretrained(x))
        x = self.fc1(x)
        return x
```

[26]:

```
our_resnet_model = NN(resnet)
```

[27]:

```
device = torch.device('cuda:0' if torch.cuda.is_available else 'cpu')
```

[28]:

```
device
```

[28]:

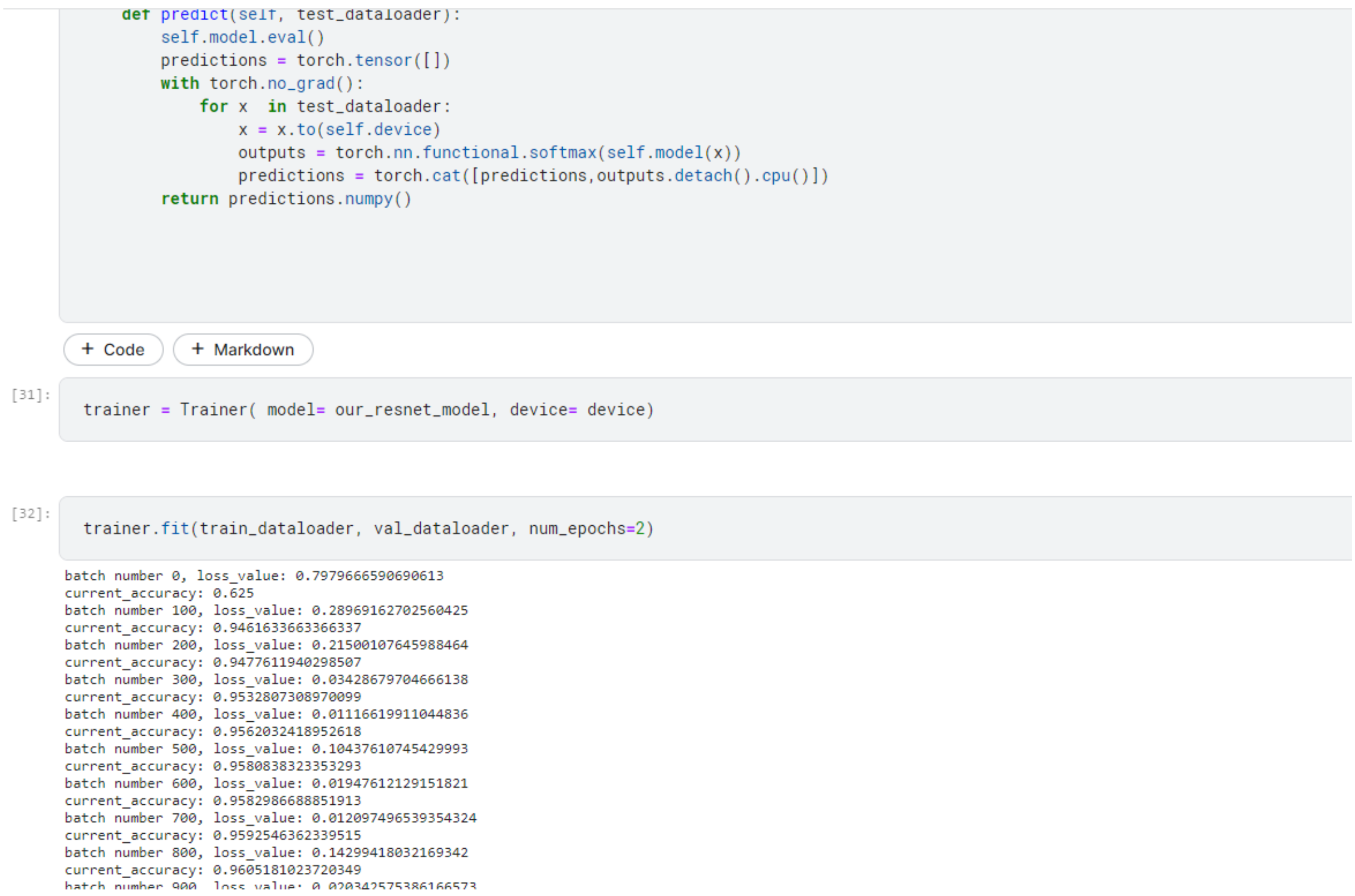
```
device(type='cuda', index=0)
```

[29]:

```
our_resnet_model = our_resnet_model.to(device)
```

```python
class Trainer:
    def __init__(self, model, device):
        self.model = model
        self.device = device

        self.loss = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam( [param for param in self.model.parameters() if param.requires_grad],
                                    lr=0.001)

    def fit(self, train_dataloader, val_dataloader, num_epochs):
        total = 0
        correct= 0

        loss_values = []
        accuracy_values = []

        for epoch in range(num_epochs):
            self.model.train()
            batch_number = 0
            loss_values_batch = []
            for x, y in train_dataloader:
                self.optimizer.zero_grad()
                x = x.to(self.device)
                y = y.to(self.device)
                outputs = self.model(x)
                l = self.loss(outputs, y)
                l.backward()
                self.optimizer.step()

                _, predicted = torch.max(outputs, 1)
                total += y.size(0)
                correct += (predicted == y).sum().item()
                loss_value = l.item()
                loss_values.append(loss_value)
                accuracy_values.append(correct/total)
```

```python
                if batch_number%100 ==0:
                    print(f"batch number {batch_number}, loss_value: {loss_value}")
                    current_accuracy = correct/total
                    print(f"current_accuracy: {current_accuracy}")
#                     plt.figure(figsize=(5,5))
#                     plt.plot(loss_values_batch)
#                     plt.show()

                batch_number+= 1

            epoch_accuracy = correct/total
            print(f"epoch_accuracy: {epoch_accuracy}")
            print(f"end of epoch {epoch}")
            #make validation

            epoch_val_loss = []

            correct = 0
            total = 0
            self.model.eval()
            with torch.no_grad():
                for x, y in val_dataloader:
                    x = x.to(self.device)
                    y = y.to(self.device)
                    outputs = self.model(x)
                    l = self.loss(outputs, y)
                    loss_value = l.item()
                    epoch_val_loss.append(loss_value)
                    _, predicted = torch.max(outputs, 1)
                    total += y.size(0)
                    correct += (predicted == y).sum().item()

            print(f"Total {total} Correct {correct} Accuracy {correct/total}")

#         plt.figure(figsize=(5,5))
#         plt.plot(loss_values_batch)
#         plt.show()
```



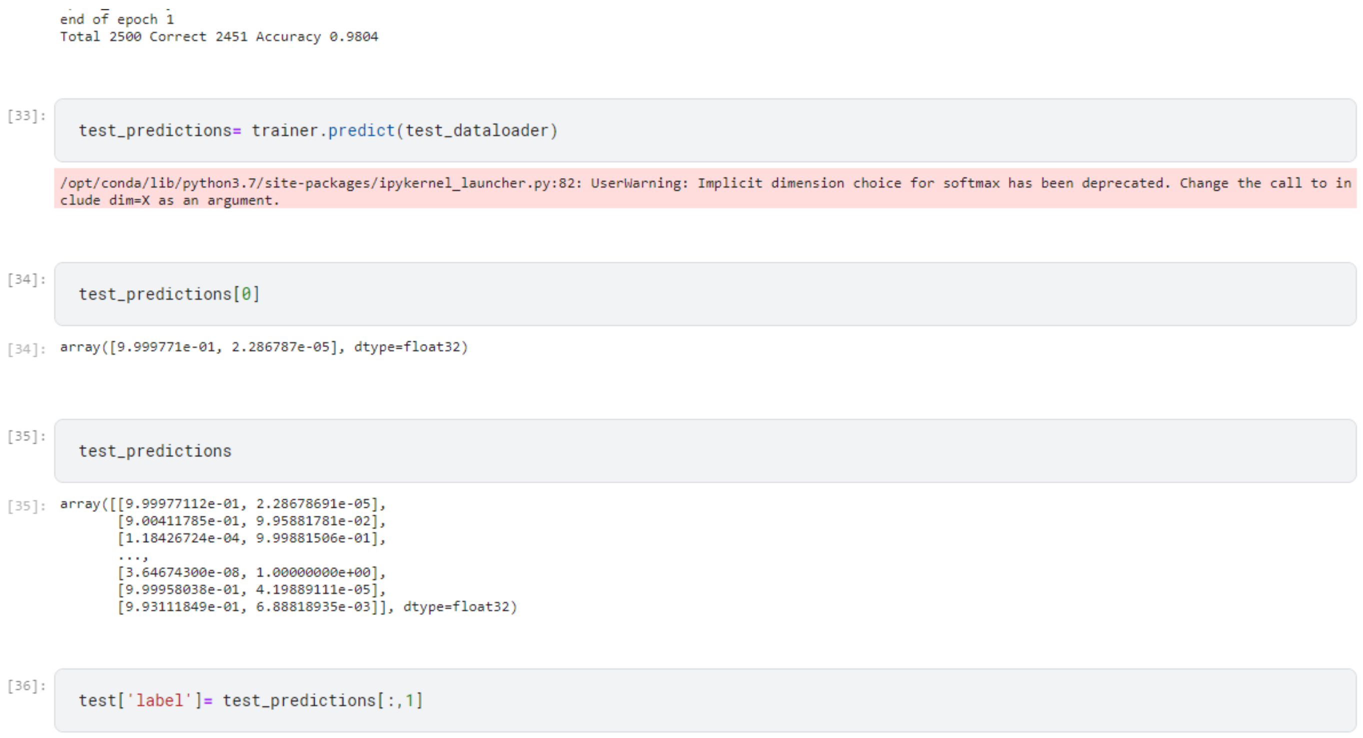
```
    def predict(self, test_dataloader):
        self.model.eval()
        predictions = torch.tensor([])
        with torch.no_grad():
            for x  in test_dataloader:
                x = x.to(self.device)
                outputs = torch.nn.functional.softmax(self.model(x))
                predictions = torch.cat([predictions,outputs.detach().cpu()])
        return predictions.numpy()
```

```
[31]: trainer = Trainer( model= our_resnet_model, device= device)
```

```
[32]: trainer.fit(train_dataloader, val_dataloader, num_epochs=2)
```

```
batch number 0, loss_value: 0.7979666590690613
current_accuracy: 0.625
batch number 100, loss_value: 0.28969162702560425
current_accuracy: 0.9461633663366337
batch number 200, loss_value: 0.21500107645988464
current_accuracy: 0.9477611940298507
batch number 300, loss_value: 0.03428679704666138
current_accuracy: 0.9532807308970099
batch number 400, loss_value: 0.01116619911044836
current_accuracy: 0.9562032418952618
batch number 500, loss_value: 0.10437610745429993
current_accuracy: 0.9580838323353293
batch number 600, loss_value: 0.01947612129151821
current_accuracy: 0.9582986688851913
batch number 700, loss_value: 0.012097496539354324
current_accuracy: 0.9592546362339515
batch number 800, loss_value: 0.14299418032169342
current_accuracy: 0.9605181023720349
batch number 900, loss_value: 0.020342575386166573
```

Epochs



```
end of epoch 1
Total 2500 Correct 2451 Accuracy 0.9804
```

```
[33]: test_predictions= trainer.predict(test_dataloader)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:82: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to in
clude dim=X as an argument.
```

```
[34]: test_predictions[0]
```

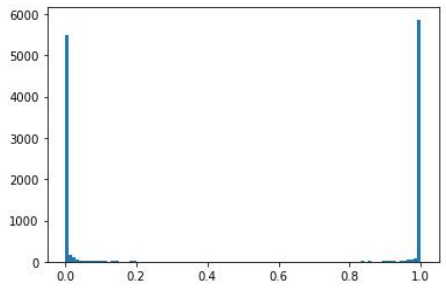
```
[34]: array([9.999771e-01, 2.286787e-05], dtype=float32)
```

```
[35]: test_predictions
```

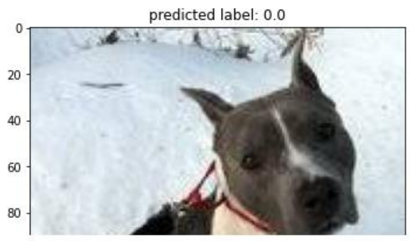
```
[35]: array([[9.99977112e-01, 2.28678691e-05],
             [9.00411785e-01, 9.95881781e-02],
             [1.18426724e-04, 9.99881506e-01],
             ...,
             [3.64674300e-08, 1.00000000e+00],
             [9.99958038e-01, 4.19889111e-05],
             [9.93111849e-01, 6.88818935e-03]], dtype=float32)
```

```
[36]: test['label']= test_predictions[:,1]
```

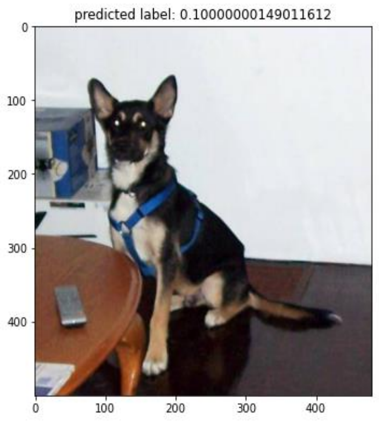
[37]:
```
plt.hist(test['label'], bins = 100)
plt.show()
```

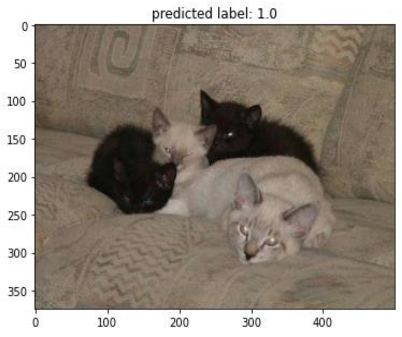


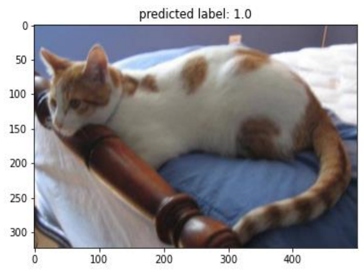
[38]:
```
for i in range(0, 40):
    plt.figure(figsize=(6,6))
    img = plt.imread(test.loc[i,'path'])
    plt.imshow(img)
    plt.title(f"predicted label: {round(test.loc[i,'label'],3)}")
    plt.show()
```

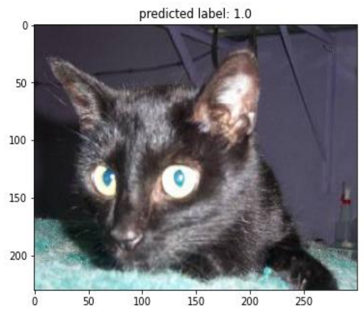


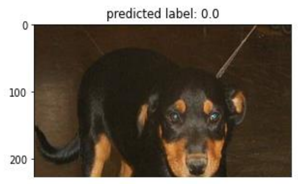
Rezultate



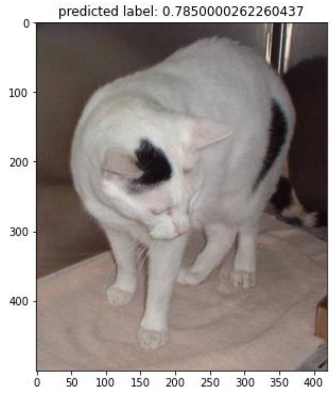




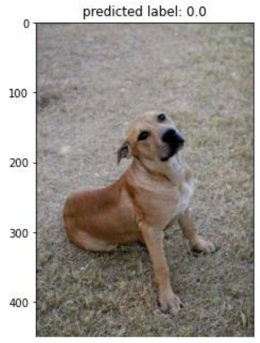




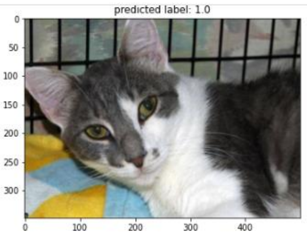
predicted label: 0.7850000262260437



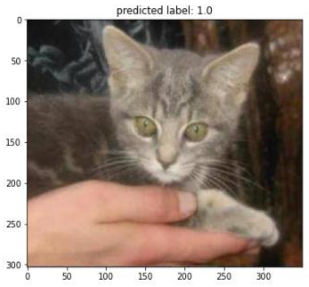
predicted label: 0.0



predicted label: 1.0

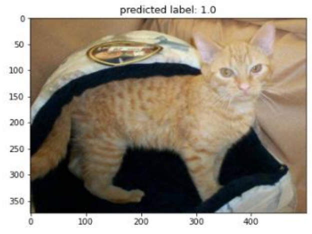


predicted label: 1.0

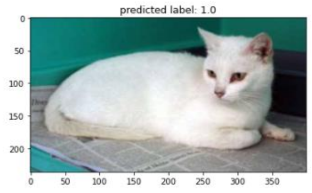


predicted label: 0.9990000128746033

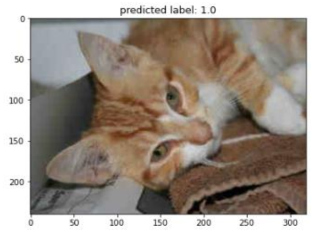
predicted label: 1.0



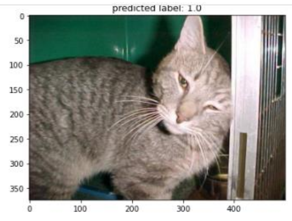
predicted label: 1.0



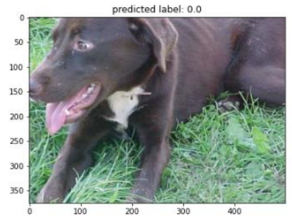
predicted label: 1.0



predicted label: 1.0

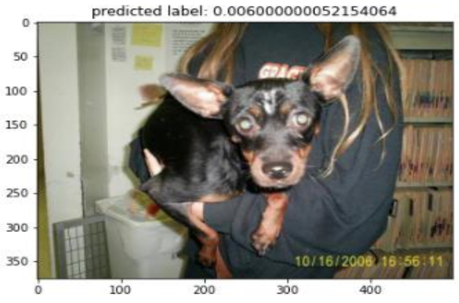


predicted label: 0.0

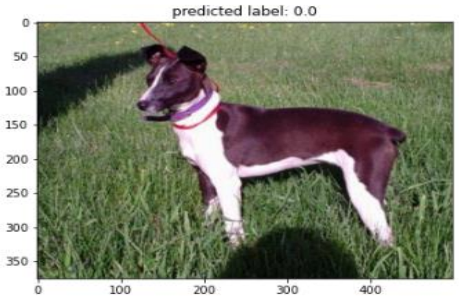


predicted label: 0.00800000037997961

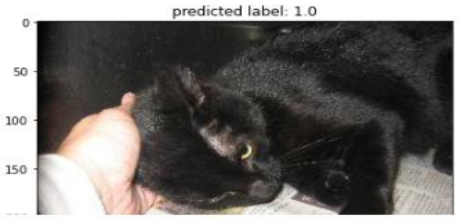
predicted label: 0.006000000052154064



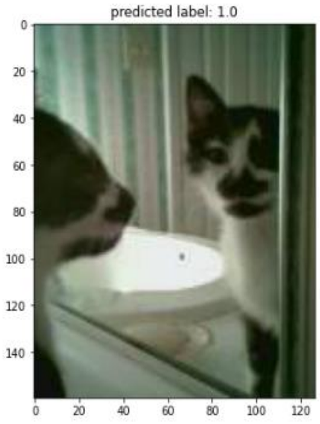
predicted label: 0.0



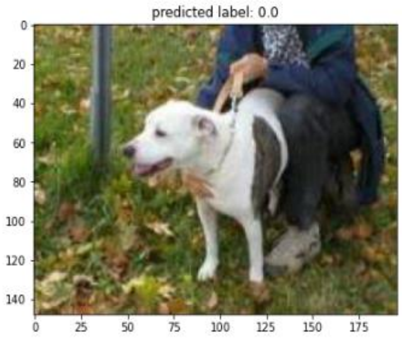
predicted label: 1.0



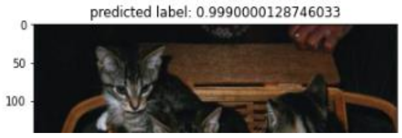
predicted label: 1.0



predicted label: 0.0

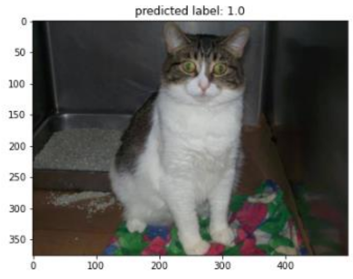


predicted label: 0.9990000128746033

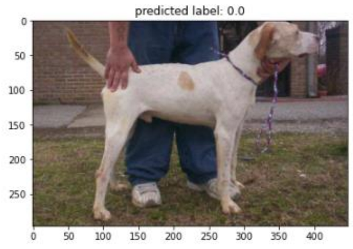


0 100 200 300 400

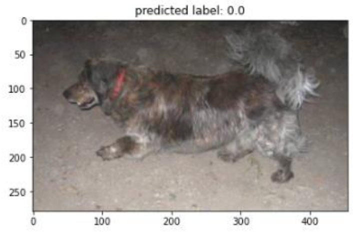
predicted label: 1.0



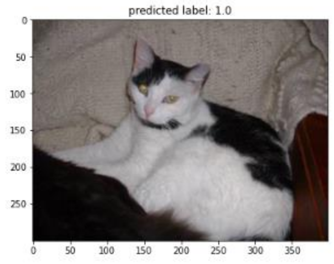
predicted label: 0.0



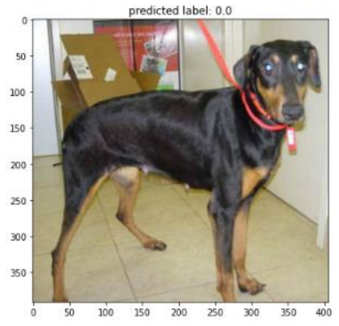
predicted label: 0.0



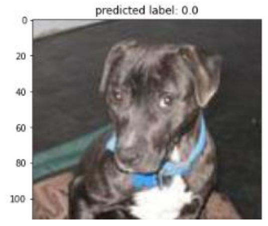
predicted label: 1.0

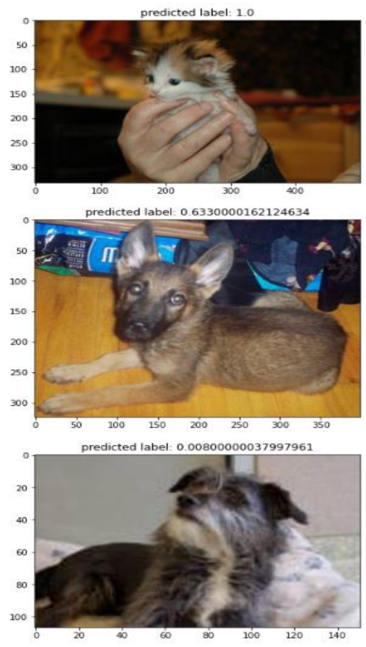


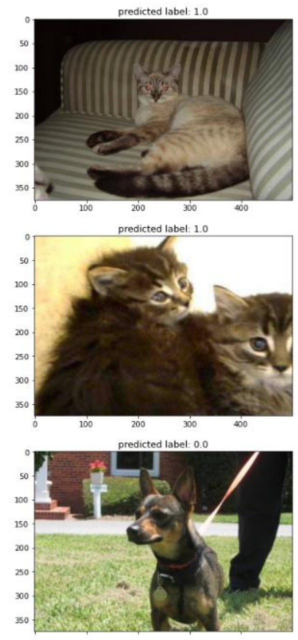
predicted label: 0.0

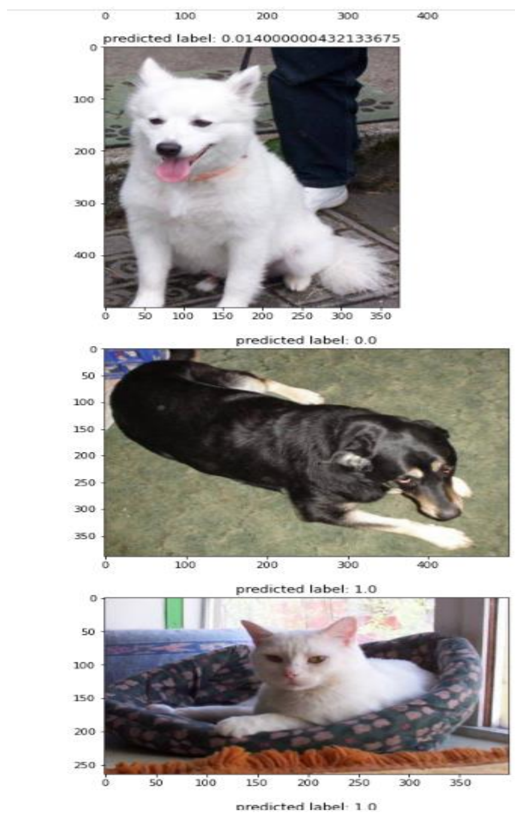


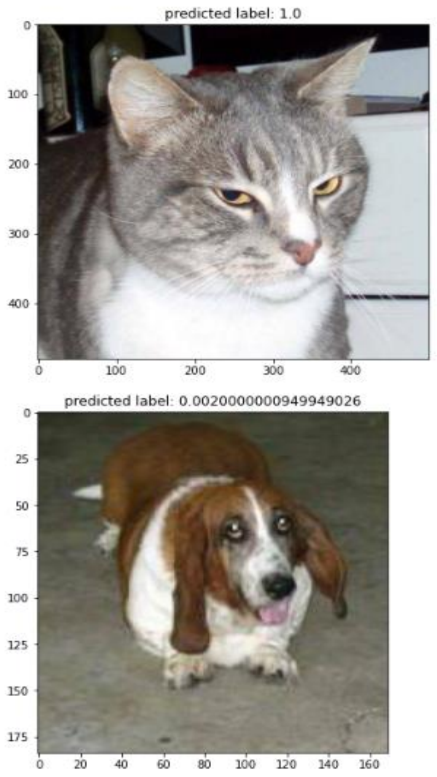
predicted label: 0.0




predicted label: 1.0
predicted label: 0.6330000162124634
predicted label: 0.0080000037997961







Vom avea 1.0 **cat** si 0.0 **dog**

Exista rezultate si de 0.002 de exemplu care, rotunjit, inseamna 0, deci rezultatul ar fi dog.