

# Arcfelismerő rendszer beléptetéshez\_clean

December 29, 2019

## 1 Arcfelismerés alapú beléptető rendszer deszka

A következőkben felépítünk egy teljes környezetet, amely arcfelismerés alapon tud vezérelni beléptető rendszert. Első körben deszkamodell kerül kialakításra, amelyben a koncepciókat valósítjuk meg, alapfunkciók kerülnek kialakításra, illetve a megközelítés alkalmazhatósága kerül megvizsgálásra.

Az alábbi részfeladatok kerülnek végrehajtásra:

- Az adathalmaz betöltése és a fényképek előfeldolgozása
- A háló tanítása az adathalmazon
- A tanított háló használata predikcióra

A felismerés két környezetben valósul meg. Egyrészt fénykép alapú felismerés - a beléptető rendszer snapshot képet készít a belépni készülőről, majd a képet küldi be a felismerő rendszernek, amely egy osztályozón belül eldönti, hogy a kép hasonlít-e valamely betanított személyre, másrészt élő video streamből is jöhet a kép. Akár folyamatosan, akár a kamerába integrált mozgásérzékelés következtében. Ilyenkor is a végén egy kivágott kép kerülmegadásra a felismerő rendszernek.

Az első lépésben at importokat szokás megvalósítani.

```
[1]: # Imports here
from collections import OrderedDict
import json
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import torch
from torch import nn
from torch import optim
import torch.nn.functional as F
from torchvision import datasets, transforms, models
import datetime
import cv2
import matplotlib.pyplot as plt
from imutils.video import VideoStream

import face_recognition
import imutils
import time
```

```
import timeit
```

## 1.1 Network osztály

Az Udacity [Deep Learning kurzusának](#) segédosztályát használjuk fel változatlan formában a neurális háló definiálására.

```
[2]: # Segédosztály a fenti kurzusból, változatlanul
class Network(nn.Module):
    def __init__(self, input_size, output_size, hidden_layers, drop_p=0.5):
        ''' Builds a feedforward network with arbitrary hidden layers.

        Arguments
        -----
        input_size: integer, size of the input
        output_size: integer, size of the output layer
        hidden_layers: list of integers, the sizes of the hidden layers
        drop_p: float between 0 and 1, dropout probability
        '''
        super().__init__()
        # Add the first layer, input to a hidden layer
        self.hidden_layers = nn.ModuleList([nn.Linear(input_size,
→hidden_layers[0])])

        # Add a variable number of more hidden layers
        layer_sizes = zip(hidden_layers[:-1], hidden_layers[1:])
        self.hidden_layers.extend([nn.Linear(h1, h2) for h1, h2 in layer_sizes])

        self.output = nn.Linear(hidden_layers[-1], output_size)

        self.dropout = nn.Dropout(p=drop_p)

    def forward(self, x):
        ''' Forward pass through the network, returns the output logits '''

        # Forward through each layer in `hidden_layers`, with ReLU activation
→and dropout
        for linear in self.hidden_layers:
            x = F.relu(linear(x))
            x = self.dropout(x)

        x = self.output(x)

        return F.log_softmax(x, dim=1)
```

## 1.2 Validációs és teszt segédfüggvények

Az Udacity [Deep Learning kurzusának](#) segédfüggvényeit használjuk fel kisebb módosításokkal.

```
[3]: # A tanítási folyamat segédfüggvényei
# Kisebb módosításokkal a fenti forrásból.
# Validációs segédfüggvény
def validation(model, testloader, criterion, device):
    test_loss = 0
    accuracy = 0
    total = 0
    correct = 0

    with torch.no_grad():
        for images, labels in testloader:

            images, labels = images.to(device), labels.to(device)

            output = model.forward(images)
            test_loss += criterion(output, labels).item()

            predicted = torch.max(output.data, 1)
            total += labels.size(0)
            correct += (predicted[1] == labels).sum().item()

    accuracy = correct / total * 100

    return test_loss, accuracy

# Kisebb módosításokkal a fenti forrásból.
# Teszthalmazon mért predikciós pontosságot számoló segédfüggvény
def check_accuracy_on_test(model, testloader, device):
    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data

            images, labels = (images.to(device), labels.to(device))

            outputs = model(images)
            predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted[1] == labels).sum().item()

    print('A hálózat pontossága %d teszt képen visszamérve: %d %%' % (total,
→(100 * correct / total)))
```

### 1.3 Tanítási ciklus

Az Udacity [Deep Learning kurzusának](#) tanítási ciklusát használjuk tanításra.

```
[4]: # Tanítási ciklus a fenti forrásból
# Kiseb b m dos t sokkal
# Benchmark adatokkal kieg sz tve
def do_deep_learning(model, trainloader, validloader, epochs, print_every,
    ↪criterion, optimizer, output_size, criterion_str, optimizer_str, checkpoint,
    ↪device='cpu'):
    epochs = epochs
    print_every = print_every
    steps = 0
    start_time = datetime.datetime.now()
    max_accuracy = 0

    print("Tan t s elkezdve:\t{}".format(start_time))
    # Amennyiben van hozz f r s nk GPU-hoz,  s azt adjuk meg, akkor használja a
    ↪GPU-t.
    # Alap ertelmezetten cput használ.
    model.to(device)

    for e in range(epochs):
        running_loss = 0
        for ii, (inputs, labels) in enumerate(trainloader):
            steps += 1
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()

            # Forward and backward passes
            outputs = model.forward(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            if steps % print_every == 0:
                # A k vetkeztet skor a h l nak eval m dban kell lennie
                # Ilyenkor nincs gr diens sz m t s  s visszaterjeszt s
                model.eval()

                # Valid ci kor nincs sz ks g a g rdiensekre (kevesebb mem ria
                ↪ s sz m t s)
                with torch.no_grad():
                    test_loss, accuracy = validation(model, validloader,
                    ↪criterion, device)
```

```

        print("Epoch: {}/{}\n[{}]"
        .format(e+1, epochs, datetime.
        ↪datetime.now()),
            "\n\tTanítási veszteség:\t{:.3f}"
        .format(running_loss/
        ↪len(trainloader)),
            "\tValidációs veszteség:\t{:.3f}"
        .format(test_loss/
        ↪len(validloader)),
            "\tValidációs pontosság:\t{:.3f}%"
        .format(accuracy))
        if accuracy > max_accuracy:
            max_accuracy = accuracy
            max_accuracy_string = "{:.3f}"
            .format(max_accuracy)
            checkpoint_name = 'checkpoint_accuracy_' +
        ↪max_accuracy_string + '_output_' + str(output_size) + '_epoch_' +
        ↪str(epochs) + '_step_' + str(steps) + '_' + arch + '_hidden_layers_' +
        ↪str(hidden_layers) + '.pth'
            save_checkpoint(checkpoint, model, checkpoint_name)
            print("\tCheckpoint mentése:\t{}\tModell pontosság: {:.
        ↪3f}%", ".format(checkpoint_name, max_accuracy))

            running_loss = 0

            # Vissza kell kapcsolni a tanítási módot a következő ciklusra
            model.train()

        end_time = datetime.datetime.now()

        print("Tanítás befejezve:\t{}".format(end_time))
        print("Tanítás ideje:\t{}".format(end_time - start_time))
        print("Elért pontosság:\t{}".format(max_accuracy))

```

## 1.4 Adatok betöltése

Az adatok betöltéséhez a `torchvision` funkciókat használjuk. ([Dokumentáció](#)). Az adatoknak a notebook mellett kell elhelyezhenni, hogy a notebook python kódja elérhesse.

Az adathalmaz szokásoknak megfelelően három részre van osztva - training, - validation, - testing.

A tanítási folyamatban a tanulás robusztussága, illetve a konvergencia javítása érdekében érdemes véletlen skálázási, croppolási, tükrözési transzformációkat alkalmazni. Transfer tanulás esetén a tanító adatokat ugyanolyanná kell konvertálni, mint amivel az eredeti hálózatot tanították. Ebben az esetben 224x224 pixelesre kell skálázni a képeket.

A tanuklási folyamat hatékonyságának mérésére validációs és teszt adatokat használunk. A validációs adatokat a tanulási ciklusokban, a teszt adatokat a tanulás végén használjuk. Fontos, hogy mind a validációs, mind pedig a tesztadatok olyan adatok (jelen esetben képek) legyenek, amelyek nem vettek részt a tanításban. Ezeket nem érdemes transformálni, de a méretüket ezeknek is be kell állítani.

A tanítás stabilitása miatt az adatok átlagát és szórását normalizálni kell, transfer learning es-

etén az eredeti előtanított hálózat megfelelő értékeivel, jelen esetben az átlagra: [0.485, 0.456, 0.406], míg a szórásra: [0.229, 0.224, 0.225]. Ennek hatására a színsatornák értékei [-1;1] intervallumba kerülnek a korábbi [0;1] intervallumból.

```
[5]: # Az adatok a data/allowed_people könyvtár alatt helyezkednek el, a megfelelő
      ↪(train, valid, test) könyvtárakban
```

```
data_dir = 'data/allowed_people'
train_dir = data_dir + '/train'
valid_dir = data_dir + '/valid'
test_dir = data_dir + '/test'
```

```
[6]: # A transzfomációkat külön-külön definiálni kell a tanító, validációs és teszt
      ↪adathalmazokra.
```

```
# Mint korábban is jeleztük, a tanító halmaz esetén számos transzfomációt
      ↪érdemed végrehajtani, de kötelező
```

```
# a 224-es méretre vágás `transforms.RandomResizedCrop(224)`, illetve a
      ↪normalizálás
```

```
# `transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]).
```

```
# A normalizálás előtt tenzorrá kell alakítani a képeket `transforms.
      ↪ToTensor()`.
```

```
train_transforms = transforms.Compose([transforms.RandomRotation(30),
                                       transforms.RandomResizedCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485, 0.456, 0.
      ↪406],
                                                           [0.229, 0.224, 0.
      ↪225]]))
```

```
# Validációs halmaz esetén csak a 224-es méretre vágás, tenzprrá alakítás és a
      ↪normalizálás szükséges.
```

```
valid_transforms = transforms.Compose([transforms.Resize(256),
                                       transforms.CenterCrop(224),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485, 0.456, 0.
      ↪406],
                                                           [0.229, 0.224, 0.
      ↪225]]))
```

```
# Teszt halmaz esetén csak a 224-es méretre vágás, tenzprrrá alakítás és a
      ↪normalizálás szükséges.
```

```
test_transforms = transforms.Compose([transforms.Resize(256),
                                       transforms.CenterCrop(224),
                                       transforms.ToTensor(),
```

```

                                transforms.Normalize([0.485, 0.456, 0.
↪406],
                                [0.229, 0.224, 0.
↪225]]))

# A könnyebb használhatóság érdekében a transzformációkat a megfelelő nevekkal
↪egy szótárba rendezzük.
data_transforms = {}
data_transforms['train'] = train_transforms
data_transforms['valid'] = valid_transforms
data_transforms['test'] = test_transforms

# Az adathalmazokat az `ImageFolder` helper osztály segítségével töltjük be.
train_data = datasets.ImageFolder(data_dir + '/train',
↪transform=train_transforms)
valid_data = datasets.ImageFolder(data_dir + '/valid',
↪transform=valid_transforms)
test_data = datasets.ImageFolder(data_dir + '/test', transform=test_transforms)

# A könnyebb használhatóság érdekében a kép adathalmazokat a megfelelő nevekkal
↪egy szótárba rendezzük.
image_datasets = {}
image_datasets['train'] = train_data
image_datasets['valid'] = valid_data
image_datasets['test'] = test_data

# Az adatloadereket létre hozzuk.
# Létre hozunk egy inferenceloadert is a kényelmes predikció támogatásához.
trainloader = torch.utils.data.DataLoader(train_data, batch_size=16,
↪shuffle=True)
validloader = torch.utils.data.DataLoader(valid_data, batch_size=8)
testloader = torch.utils.data.DataLoader(test_data, batch_size=8)
inferenceloader = torch.utils.data.DataLoader(test_data, batch_size=1)

# A könnyebb használhatóság érdekében a dataloadereket a megfelelő nevekkal egy
↪szótárba rendezzük.
dataloaders = {}
dataloaders['train'] = trainloader
dataloaders['valid'] = validloader
dataloaders['test'] = testloader

```

Hasznos lehet látni, hogy milyen a tanító adatok szegmentálása, hány elemből áll a trainloader.

```
[7]: len(trainloader)
```

```
[7]: 107
```

### 1.4.1 Címke (név) hozzárendelés

A predikció egy egész értéket fog kiadni, amit egy szótár segítségével lehet nevekhez rendelni. A hozzárendelést `cat_to_name.json` file tartalmazza. A JSON objektumot a `json` modullal lehet beolvasni. Ez egy szótár összerendelést ad a kategóriák és a beengedhető emberek nevei közt.

```
[8]: with open('cat_to_name.json', 'r') as f:
      cat_to_name = json.load(f)
```

Érdemes megnézni a nevek listáját:

```
[9]: print(cat_to_name)
```

```
{'1': 'Gyuszi', '2': 'Gabi', '3': 'Agi', '4': 'Gyula', '5': 'Geri', '6':
'Gyuri', '7': 'Roli', '8': 'Zsombor', '9': 'Rezso', '10': 'Rezsi', '11':
'Toncsi'}
```

## 2 Az osztályozó létrehozása és tanítása

Az adatok betöltése után az osztályozó létrehozása következik. Transzfer tanulási technikát alkalmazunk, vagyis nagy (tipikusan több száz ezres, akár milliós) mintán tanított hálózatot alkalmazunk a képek jellemzőinek felismerésére és a hálózat végén lévő osztályozót csacseréljük le egy saját osztályozóra, majd a folyamatban ezt a saját osztályozót tanítjuk a saját fotóinkkal.

Az alábbi lépéseket kell megtenni: \* Egy `előtanított háló` betöltése \* Egy osztályozó definiálása \* Az osztályozó tanítása \* Kísérletezés a hiperparaméterekkel

Fontos, hogy a tanítás során csak az osztályozó súlyát szabad rissíteni (az előre tanított hálózatrész súlyait változatlanul kell hagyni).

Többfajta modellt lehet használni képek feldolgozására transzfer tanulásra. A leggyakrabban használtak: - vgg16 - alexnet - resnet18 - squeezeNet1\_0 - densenet121 - inception\_v3

Az alapértelmezett modellnek a vgg16-ot használjuk.

```
[10]: # Előtanított modellek letölthetőek
      # A vgg16-os modellt használjuk
      # Az alábbi változót átírva más modellre, az adott modell kerül felhasználásra
      ↪ a transzfer tanulásra

      arch = "vgg16"

      if arch == "vgg16":
          model = models.vgg16(pretrained=True)
      elif arch == "alexnet":
          model = models.alexnet(pretrained=True)
```



```

elif arch == "resnet18":
    model = models.resnet18(pretrained=True)
elif arch == "squeezenet1_0":
    model = models.squeezenet1_0(pretrained=True)
elif arch == "densenet121":
    model = models.densenet121(pretrained=True)
elif arch == "inception_v3":
    model = models.inception_v3(pretrained=True)
else:
    model = models.vgg16(pretrained=True)

```

A modellt érdemes megnézni:

```
[11]: model
```

```

[11]: VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)

```

```

(26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(27): ReLU(inplace=True)
(28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(29): ReLU(inplace=True)
(30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
)
(avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
(classifier): Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace=True)
  (2): Dropout(p=0.5, inplace=False)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace=True)
  (5): Dropout(p=0.5, inplace=False)
  (6): Linear(in_features=4096, out_features=1000, bias=True)
)
)

```

```

[12]: # Mivel előre tanított hálót használunk, az előre tanított paramétereket nem
      ↳ szabad módosítani, ezért
      # azok grádiensét letiltjuk, visszaterjesztésnél nem módosítjuk azokat
      for param in model.parameters():
          param.requires_grad = False

      # A kimenetek száma meg kell egyezzen a betanított emberek számával
      # Amennyiben változik a kimenet szám, akkor újra kell tanítani az osztályozót
      output_size = 11

      # Az osztályozót csaréljuk majd le a hálózat végén, azt definiáljuk
      # Az osztályozó felépítésével érdemes kísérletezni, hogy a legjobb osztályozási
      ↳ hatékonyságot érhessük el.
      #hidden_layers = [4096, 1000, 250, 80]
      hidden_layers = [1024]

      classifier = Network(25088, output_size, hidden_layers, drop_p=0.5)
      #classifier = Network(25088, output_size, [1024], drop_p=0.5)

      # Lecseréljük az osztályozót
      model.classifier = classifier

```

```

[13]: # Hiper paraméterek
      # A kísérletek azt mutatják, hogy már 6-8 epoch is jó eredményt ad
      # Sajnos GPU nélkül a jelenlegi kép mennyiséggel egy epoch kb. egy órát vesz
      ↳ igénybe
      # GPU használatával ez perces nagyságrendre csökkent
      epochs = 8

```

```

print_any = 10
lr = 0.001

# Definiáljuk a hiba függvényt és az optimalizáló algoritmust
criterion_str = 'nn.NLLLoss()'
optimizer_str = 'optim.Adam(model.classifier.parameters(), lr=lr)'

```

```

[14]: # A hibafüggvényt és az optimalizálót hozzárendeljük a megfelelő változókhoz,
      ↪ amelyeket paraméterként
      # átadunk a tanító függvénynek
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.classifier.parameters(), lr=lr)

```

## 2.1 Ellenőrzési pont (checkpoint) mentése

Mivel a hálózat tanítása nagyon időigényes, érdemes a tanított hálózatot checkpoint formájában mindig elmenteni. Ekkor a mentésből a modell újra építhető a checkpoint betöltésével, így az bármikor használható majd következtetésre.

Érdemes a modellhez elmenteni más hasznos információt is, pl a mi esetünkben az osztály-index leképezést. Ezt az alábbi kóddal nyerhetjük ki:

```
image_datasets['train'].class_to_idx
```

Ezt az adatot egy attribútumban rendelhetjük a modellhez:

```
model.class_to_idx = image_datasets['train'].class_to_idx
```

Amennyiben elmentjük az epochok számát, illetve az optimalizáló állapotát is, a modellt bármikor tovább taníthatjuk, javítva ezzel a következtetési hatékonyságot. Az optimalizáló állapota az alábbi módon érhető el:

```
optimizer.state_dict
```

```

[15]: # Definiáljuk a checkpoint mentéshez szükséges adatokat, illetve magát a
      ↪ függvényt.
def save_checkpoint(checkpoint, model, path):
    torch.save(checkpoint, path)

model.class_to_idx = image_datasets['train'].class_to_idx
checkpoint = {'input_size': 25088,
             'output_size': output_size,
             'hidden_layers': [each.out_features for each in model.classifier.
             ↪ hidden_layers],
             'class_to_idx': model.class_to_idx,
             'state_dict': model.classifier.state_dict(),
             'epochs': epochs,
             'lr': lr,
             'print_any': print_any,
             'criterion_str': criterion_str,

```

```
'optimizer_str': optimizer_str}
```

```
[16]: # A tanítás egy hosszú folyamat, még kis epoch szám mellett is. Egy epoch akár  
      ↪egy órát is eltarthat.  
      # 10 epoch nagyjából 8-10 óra  
      do_deep_learning(model, trainloader, validloader, epochs, print_any, criterion,  
      ↪optimizer, output_size, criterion_str, optimizer_str, checkpoint, 'cuda')
```

Tanítás elkezdve: 2019-12-29 18:16:31.359004

Epoch: 1/8

[2019-12-29 18:16:41.666563]

Tanítási veszteség: 0.265 Validációs veszteség: 1.901

Validációs pontosság: 40.237%

Checkpoint mentése: checkpoint\_accuracy\_40.237\_output\_11\_epoch\_8\_step\_10\_vgg16\_hidden\_layers\_[1024].pth Modell pontosság: 40.237%,

Epoch: 1/8

[2019-12-29 18:16:49.859913]

Tanítási veszteség: 0.260 Validációs veszteség: 2.776

Validációs pontosság: 38.462%

Epoch: 1/8

[2019-12-29 18:16:57.142115]

Tanítási veszteség: 0.226 Validációs veszteség: 2.188

Validációs pontosság: 51.479%

Checkpoint mentése: checkpoint\_accuracy\_51.479\_output\_11\_epoch\_8\_step\_30\_vgg16\_hidden\_layers\_[1024].pth Modell pontosság: 51.479%,

Epoch: 1/8

[2019-12-29 18:17:05.352552]

Tanítási veszteség: 0.163 Validációs veszteség: 1.343

Validációs pontosság: 59.763%

Checkpoint mentése: checkpoint\_accuracy\_59.763\_output\_11\_epoch\_8\_step\_40\_vgg16\_hidden\_layers\_[1024].pth Modell pontosság: 59.763%,

Epoch: 1/8

[2019-12-29 18:17:13.378666]

Tanítási veszteség: 0.134 Validációs veszteség: 1.161

Validációs pontosság: 66.272%

Checkpoint mentése: checkpoint\_accuracy\_66.272\_output\_11\_epoch\_8\_step\_50\_vgg16\_hidden\_layers\_[1024].pth Modell pontosság: 66.272%,

Epoch: 1/8

[2019-12-29 18:17:21.139666]

Tanítási veszteség: 0.134 Validációs veszteség: 1.064

Validációs pontosság: 61.538%

Epoch: 1/8

[2019-12-29 18:17:28.779859]

Tanítási veszteség: 0.103 Validációs veszteség: 1.383

Validációs pontosság: 56.805%

Epoch: 1/8

[2019-12-29 18:17:37.535165]

Tanítási veszteség: 0.097    Validációs veszteség: 0.907  
 Validációs pontosság: 72.189%  
 Checkpoint mentése: checkpoint\_accuracy\_72.189\_output\_11\_epoch\_8\_step\_80\_vgg16\_hidden\_layers\_[1024].pth    Modell pontosság: 72.189%,  
 Epoch: 1/8  
 [2019-12-29 18:17:45.541724]  
 Tanítási veszteség: 0.091    Validációs veszteség: 1.179  
 Validációs pontosság: 65.089%  
 Epoch: 1/8  
 [2019-12-29 18:17:53.732003]  
 Tanítási veszteség: 0.091    Validációs veszteség: 0.986  
 Validációs pontosság: 69.822%  
 Tanítás befejezve: 2019-12-29 18:17:55.867734  
 Tanítás ideje: 0:01:24.508730  
 Elért pontosság: 72.18934911242604  
 Epoch: 2/8  
 [2019-12-29 18:18:02.063447]  
 Tanítási veszteség: 0.031    Validációs veszteség: 0.886  
 Validációs pontosság: 71.598%  
 Epoch: 2/8  
 [2019-12-29 18:18:10.030240]  
 Tanítási veszteség: 0.083    Validációs veszteség: 1.048  
 Validációs pontosság: 63.905%  
 Epoch: 2/8  
 [2019-12-29 18:18:17.950183]  
 Tanítási veszteség: 0.074    Validációs veszteség: 1.101  
 Validációs pontosság: 66.864%  
 Epoch: 2/8  
 [2019-12-29 18:18:25.624401]  
 Tanítási veszteség: 0.077    Validációs veszteség: 1.143  
 Validációs pontosság: 68.639%  
 Epoch: 2/8  
 [2019-12-29 18:18:34.409634]  
 Tanítási veszteség: 0.083    Validációs veszteség: 1.050  
 Validációs pontosság: 68.047%  
 Epoch: 2/8  
 [2019-12-29 18:18:41.844750]  
 Tanítási veszteség: 0.064    Validációs veszteség: 1.046  
 Validációs pontosság: 69.822%  
 Epoch: 2/8  
 [2019-12-29 18:18:49.463132]  
 Tanítási veszteség: 0.066    Validációs veszteség: 0.669  
 Validációs pontosság: 79.882%  
 Checkpoint mentése: checkpoint\_accuracy\_79.882\_output\_11\_epoch\_8\_step\_170\_vgg16\_hidden\_layers\_[1024].pth    Modell pontosság: 79.882%,  
 Epoch: 2/8  
 [2019-12-29 18:18:57.111437]  
 Tanítási veszteség: 0.057    Validációs veszteség: 1.083

Validációs pontosság: 72.189%  
Epoch: 2/8  
[2019-12-29 18:19:05.366608]  
Tanítási veszteség: 0.067 Validációs veszteség: 0.918  
Validációs pontosság: 75.740%  
Epoch: 2/8  
[2019-12-29 18:19:13.680278]  
Tanítási veszteség: 0.063 Validációs veszteség: 0.776  
Validációs pontosság: 77.515%  
Epoch: 2/8  
[2019-12-29 18:19:21.631522]  
Tanítási veszteség: 0.049 Validációs veszteség: 1.059  
Validációs pontosság: 70.414%  
Tanítás befejezve: 2019-12-29 18:19:22.580976  
Tanítás ideje: 0:02:51.221972  
Elért pontosság: 79.88165680473372  
Epoch: 3/8  
[2019-12-29 18:19:29.748902]  
Tanítási veszteség: 0.033 Validációs veszteség: 0.941  
Validációs pontosság: 73.964%  
Epoch: 3/8  
[2019-12-29 18:19:37.814949]  
Tanítási veszteség: 0.053 Validációs veszteség: 0.983  
Validációs pontosság: 74.556%  
Epoch: 3/8  
[2019-12-29 18:19:46.049301]  
Tanítási veszteség: 0.064 Validációs veszteség: 0.721  
Validációs pontosság: 75.740%  
Epoch: 3/8  
[2019-12-29 18:19:54.315222]  
Tanítási veszteség: 0.068 Validációs veszteség: 0.674  
Validációs pontosság: 76.331%  
Epoch: 3/8  
[2019-12-29 18:20:02.150609]  
Tanítási veszteség: 0.060 Validációs veszteség: 0.829  
Validációs pontosság: 75.148%  
Epoch: 3/8  
[2019-12-29 18:20:09.688127]  
Tanítási veszteség: 0.064 Validációs veszteség: 0.935  
Validációs pontosság: 77.515%  
Epoch: 3/8  
[2019-12-29 18:20:17.426619]  
Tanítási veszteség: 0.075 Validációs veszteség: 0.698  
Validációs pontosság: 83.432%  
Checkpoint mentése: checkpoint\_accuracy\_83.432\_output\_11\_epoch\_8\_step\_280\_vgg16\_hidden\_layers\_[1024].pth Modell pontosság: 83.432%,  
Epoch: 3/8  
[2019-12-29 18:20:24.537589]

Tanítási veszteség: 0.067    Validációs veszteség: 0.694  
 Validációs pontosság: 79.290%  
 Epoch: 3/8  
 [2019-12-29 18:20:32.907868]

Tanítási veszteség: 0.050    Validációs veszteség: 0.735  
 Validációs pontosság: 76.923%  
 Epoch: 3/8  
 [2019-12-29 18:20:41.356189]

Tanítási veszteség: 0.071    Validációs veszteség: 0.918  
 Validációs pontosság: 74.556%  
 Epoch: 3/8  
 [2019-12-29 18:20:49.263935]

Tanítási veszteség: 0.069    Validációs veszteség: 0.786  
 Validációs pontosság: 76.331%  
 Tanítás befejezve: 2019-12-29 18:20:49.424892  
 Tanítás ideje: 0:04:18.065888  
 Elért pontosság: 83.4319526627219  
 Epoch: 4/8  
 [2019-12-29 18:20:56.589847]

Tanítási veszteség: 0.053    Validációs veszteség: 0.776  
 Validációs pontosság: 75.740%  
 Epoch: 4/8  
 [2019-12-29 18:21:04.940589]

Tanítási veszteség: 0.064    Validációs veszteség: 0.530  
 Validációs pontosság: 84.024%  
 Checkpoint mentése: checkpoint\_accuracy\_84.024\_output\_11\_epoch\_8\_step\_340\_vgg16\_hidden\_layers\_[1024].pth    Modell pontosság: 84.024%,  
 Epoch: 4/8  
 [2019-12-29 18:21:12.683200]

Tanítási veszteség: 0.053    Validációs veszteség: 1.338  
 Validációs pontosság: 68.047%  
 Epoch: 4/8  
 [2019-12-29 18:21:21.158284]

Tanítási veszteség: 0.057    Validációs veszteség: 0.612  
 Validációs pontosság: 79.882%  
 Epoch: 4/8  
 [2019-12-29 18:21:28.781496]

Tanítási veszteség: 0.064    Validációs veszteség: 0.364  
 Validációs pontosság: 89.349%  
 Checkpoint mentése: checkpoint\_accuracy\_89.349\_output\_11\_epoch\_8\_step\_370\_vgg16\_hidden\_layers\_[1024].pth    Modell pontosság: 89.349%,  
 Epoch: 4/8  
 [2019-12-29 18:21:36.681534]

Tanítási veszteség: 0.058    Validációs veszteség: 0.678  
 Validációs pontosság: 76.923%  
 Epoch: 4/8  
 [2019-12-29 18:21:44.950936]

Tanítási veszteség: 0.060    Validációs veszteség: 0.670

Validációs pontosság: 85.207%  
Epoch: 4/8  
[2019-12-29 18:21:53.468469]  
Tanítási veszteség: 0.042 Validációs veszteség: 0.807  
Validációs pontosság: 80.473%  
Epoch: 4/8  
[2019-12-29 18:22:00.634251]  
Tanítási veszteség: 0.054 Validációs veszteség: 0.625  
Validációs pontosság: 80.473%  
Epoch: 4/8  
[2019-12-29 18:22:08.554390]  
Tanítási veszteség: 0.047 Validációs veszteség: 0.556  
Validációs pontosság: 80.473%  
Tanítás befejezve: 2019-12-29 18:22:11.119421  
Tanítás ideje: 0:05:39.760417  
Elért pontosság: 89.3491124260355  
Epoch: 5/8  
[2019-12-29 18:22:17.001432]  
Tanítási veszteség: 0.011 Validációs veszteség: 0.866  
Validációs pontosság: 76.331%  
Epoch: 5/8  
[2019-12-29 18:22:25.089127]  
Tanítási veszteség: 0.045 Validációs veszteség: 0.716  
Validációs pontosság: 80.473%  
Epoch: 5/8  
[2019-12-29 18:22:32.888568]  
Tanítási veszteség: 0.064 Validációs veszteség: 0.846  
Validációs pontosság: 79.882%  
Epoch: 5/8  
[2019-12-29 18:22:40.719381]  
Tanítási veszteség: 0.044 Validációs veszteség: 0.708  
Validációs pontosság: 81.657%  
Epoch: 5/8  
[2019-12-29 18:22:48.352756]  
Tanítási veszteség: 0.039 Validációs veszteség: 1.056  
Validációs pontosság: 74.556%  
Epoch: 5/8  
[2019-12-29 18:22:56.755281]  
Tanítási veszteség: 0.052 Validációs veszteség: 0.682  
Validációs pontosság: 80.473%  
Epoch: 5/8  
[2019-12-29 18:23:04.972331]  
Tanítási veszteség: 0.057 Validációs veszteség: 0.682  
Validációs pontosság: 81.657%  
Epoch: 5/8  
[2019-12-29 18:23:12.686573]  
Tanítási veszteség: 0.052 Validációs veszteség: 1.105  
Validációs pontosság: 71.598%



Epoch: 5/8  
[2019-12-29 18:23:20.528757]  
Tanítási veszteség: 0.033 Validációs veszteség: 0.571  
Validációs pontosság: 82.249%

Epoch: 5/8  
[2019-12-29 18:23:28.717885]  
Tanítási veszteség: 0.054 Validációs veszteség: 0.333  
Validációs pontosság: 89.941%  
Checkpoint mentése: checkpoint\_accuracy\_89.941\_output\_11\_epoch\_8\_step\_520\_vgg16\_hidden\_layers\_[1024].pth Modell pontosság: 89.941%,

Epoch: 5/8  
[2019-12-29 18:23:36.492929]  
Tanítási veszteség: 0.067 Validációs veszteség: 0.779  
Validációs pontosság: 81.065%  
Tanítás befejezve: 2019-12-29 18:23:38.215215  
Tanítás ideje: 0:07:06.856211  
Elért pontosság: 89.94082840236686

Epoch: 6/8  
[2019-12-29 18:23:44.399078]  
Tanítási veszteség: 0.015 Validációs veszteség: 0.669  
Validációs pontosság: 82.840%

Epoch: 6/8  
[2019-12-29 18:23:52.753862]  
Tanítási veszteség: 0.048 Validációs veszteség: 0.872  
Validációs pontosság: 81.065%

Epoch: 6/8  
[2019-12-29 18:24:01.340191]  
Tanítási veszteség: 0.061 Validációs veszteség: 0.800  
Validációs pontosság: 81.657%

Epoch: 6/8  
[2019-12-29 18:24:09.056805]  
Tanítási veszteség: 0.047 Validációs veszteség: 0.619  
Validációs pontosság: 82.249%

Epoch: 6/8  
[2019-12-29 18:24:17.259536]  
Tanítási veszteség: 0.051 Validációs veszteség: 0.447  
Validációs pontosság: 86.391%

Epoch: 6/8  
[2019-12-29 18:24:25.522917]  
Tanítási veszteség: 0.062 Validációs veszteség: 0.936  
Validációs pontosság: 77.515%

Epoch: 6/8  
[2019-12-29 18:24:33.638959]  
Tanítási veszteség: 0.054 Validációs veszteség: 0.809  
Validációs pontosság: 76.331%

Epoch: 6/8  
[2019-12-29 18:24:41.363424]  
Tanítási veszteség: 0.062 Validációs veszteség: 0.658

Validációs pontosság: 79.290%  
Epoch: 6/8  
[2019-12-29 18:24:49.319973]  
Tanítási veszteség: 0.071 Validációs veszteség: 0.431  
Validációs pontosság: 84.024%  
Epoch: 6/8  
[2019-12-29 18:24:56.930115]  
Tanítási veszteség: 0.043 Validációs veszteség: 0.402  
Validációs pontosság: 85.207%  
Epoch: 6/8  
[2019-12-29 18:25:04.833248]  
Tanítási veszteség: 0.045 Validációs veszteség: 0.748  
Validációs pontosság: 79.882%  
Tanítás befejezve: 2019-12-29 18:25:05.223090  
Tanítás ideje: 0:08:33.864086  
Elért pontosság: 89.94082840236686  
Epoch: 7/8  
[2019-12-29 18:25:13.137342]  
Tanítási veszteség: 0.066 Validációs veszteség: 0.770  
Validációs pontosság: 80.473%  
Epoch: 7/8  
[2019-12-29 18:25:21.124387]  
Tanítási veszteség: 0.058 Validációs veszteség: 0.965  
Validációs pontosság: 75.148%  
Epoch: 7/8  
[2019-12-29 18:25:30.034338]  
Tanítási veszteség: 0.038 Validációs veszteség: 1.121  
Validációs pontosság: 73.373%  
Epoch: 7/8  
[2019-12-29 18:25:38.143041]  
Tanítási veszteség: 0.060 Validációs veszteség: 0.610  
Validációs pontosság: 85.207%  
Epoch: 7/8  
[2019-12-29 18:25:45.748355]  
Tanítási veszteség: 0.046 Validációs veszteség: 0.627  
Validációs pontosság: 81.065%  
Epoch: 7/8  
[2019-12-29 18:25:53.239522]  
Tanítási veszteség: 0.051 Validációs veszteség: 0.881  
Validációs pontosság: 76.923%  
Epoch: 7/8  
[2019-12-29 18:26:00.752010]  
Tanítási veszteség: 0.052 Validációs veszteség: 0.754  
Validációs pontosság: 78.107%  
Epoch: 7/8  
[2019-12-29 18:26:08.860149]  
Tanítási veszteség: 0.047 Validációs veszteség: 0.582  
Validációs pontosság: 82.840%

Epoch: 7/8  
 [2019-12-29 18:26:16.009787]  
 Tanítási veszteség: 0.037 Validációs veszteség: 0.707  
 Validációs pontosság: 82.249%

Epoch: 7/8  
 [2019-12-29 18:26:24.613104]  
 Tanítási veszteség: 0.039 Validációs veszteség: 0.600  
 Validációs pontosság: 86.391%  
 Tanítás befejezve: 2019-12-29 18:26:26.937173  
 Tanítás ideje: 0:09:55.578169  
 Elért pontosság: 89.94082840236686

Epoch: 8/8  
 [2019-12-29 18:26:32.591454]  
 Tanítási veszteség: 0.001 Validációs veszteség: 1.109  
 Validációs pontosság: 74.556%

Epoch: 8/8  
 [2019-12-29 18:26:40.612867]  
 Tanítási veszteség: 0.053 Validációs veszteség: 1.060  
 Validációs pontosság: 75.148%

Epoch: 8/8  
 [2019-12-29 18:26:48.942339]  
 Tanítási veszteség: 0.044 Validációs veszteség: 1.146  
 Validációs pontosság: 69.822%

Epoch: 8/8  
 [2019-12-29 18:26:56.501353]  
 Tanítási veszteség: 0.052 Validációs veszteség: 1.008  
 Validációs pontosság: 73.964%

Epoch: 8/8  
 [2019-12-29 18:27:04.348649]  
 Tanítási veszteség: 0.040 Validációs veszteség: 0.624  
 Validációs pontosság: 78.698%

Epoch: 8/8  
 [2019-12-29 18:27:13.044833]  
 Tanítási veszteség: 0.042 Validációs veszteség: 0.609  
 Validációs pontosság: 81.657%

Epoch: 8/8  
 [2019-12-29 18:27:20.824633]  
 Tanítási veszteség: 0.045 Validációs veszteség: 1.242  
 Validációs pontosság: 72.189%

Epoch: 8/8  
 [2019-12-29 18:27:29.079265]  
 Tanítási veszteség: 0.058 Validációs veszteség: 1.004  
 Validációs pontosság: 70.414%

Epoch: 8/8  
 [2019-12-29 18:27:37.209481]  
 Tanítási veszteség: 0.038 Validációs veszteség: 0.709  
 Validációs pontosság: 81.657%

Epoch: 8/8

[2019-12-29 18:27:45.269669]

Tanítási veszteség: 0.059 Validációs veszteség: 0.911  
Validációs pontosság: 82.249%  
Epoch: 8/8

[2019-12-29 18:27:52.781379]

Tanítási veszteség: 0.034 Validációs veszteség: 0.815  
Validációs pontosság: 72.189%  
Tanítás befejezve: 2019-12-29 18:27:54.475999  
Tanítás ideje: 0:11:23.116995  
Elért pontosság: 89.94082840236686

A tanítás végén érdemes ellenőrizni a modell pontosságát. Ezt a `check_accuracy_on_test` segéd-függvénnyel lehet megtenni.

A függvény neve kissé félrevezető - a loadertől függ, hogy milyen adathalmazra fut le a teszt. Amennyiben validációs loadert adunk meg, akkor a validációs halmazra, ha teszt loadert adunk meg, akkor a teszt halmazra futtatja a teszteket és ad összesítést.

```
[17]: # A pontosság ellenőrzése a validációs halmazon
check_accuracy_on_test(model, validloader, 'cuda')
```

A hálózat pontossága 169 teszt képen visszamérve: 72 %

## 2.2 A hálózat tesztelése

Érdemes a modellt olyan teszt adatokkal is tesztelni, amelyet a tanítás során nem látott a rendszer. Jelen esetben ez a tanítás során fel nem használt fotókat jelent. Ez jó becslést ad arra, hogyan fog majd viselkedni a modell új, akár frissen készített fotók esetén is az osztályozás során.

```
[18]: # A pontosság ellenőrzése a teszt halmazon
check_accuracy_on_test(model, testloader, 'cuda')
```

A hálózat pontossága 146 teszt képen visszamérve: 57 %

## 2.3 Ellenőrzési pont visszatöltése

Ahhoz, hogy a későbbiekben is tudjuk használni a modellünket, vissza kell tudnunk tölteni a korábban elmentett modellünket a checkpointból.

```
[19]: # Checkpoint visszatöltése
# A mentési lépések fordítottja
def load_checkpoint(filepath, device='cpu'):
    checkpoint = torch.load(filepath)

    # Az osztályozó felépítése
    classifier = Network(checkpoint['input_size'],
                        checkpoint['output_size'],
                        checkpoint['hidden_layers'])
    print(checkpoint['hidden_layers'])
    # Itt most be van drótozva a vgg16 modell.
```

```

# Ez általánosítható egy modell paraméter megadásával - lásd korábbi cella
→ az előre tanított modellekkel
model = models.vgg16(pretrained=True)

# A modell paramétereit be kell fagyasztani, hogy tanításkor (osztályozó)
→ ne módosuljonak a súlyok
for param in model.parameters():
    param.requires_grad = False

model.classifier = classifier
model.classifier.load_state_dict(checkpoint['state_dict'])
model.class_to_idx = checkpoint['class_to_idx']

# Hiperparaméterek betöltése
epochs = checkpoint['epochs']
print_any = checkpoint['print_any']
lr = checkpoint['lr']
criterion_str = checkpoint['criterion_str']
optimizer_str = checkpoint['optimizer_str']

criterion = eval(criterion_str)
optimizer = eval(optimizer_str)

model = model.to(device)

return model, epochs, print_any, lr, criterion, optimizer

```

```

[ ]: checkpoint_name = 'checkpoint_accuracy_89.
→ 941_output_11_epoch_8_step_520_vgg16_hidden_layers_[1024].pth'
model, epochs, print_any, lr, criterion, optimizer =
→ load_checkpoint(checkpoint_name, device='cuda')

```

### 3 Következtetés

A továbbiakban magát a következtető függvényt készítjük el. Jelen esetben a következtetés egy osztályozási ciklus lefuttatása, vagyis egy megadott kép tanítás soráni osztályba sorolása. A beléptető rendszer esetében a képet egy kamera készíti (akár snapshot, akár folyamatos módban a stream egy frame-jének kivágásával).

Ezt a képet adjuk át a neurális hálónak következtetésre.

A függvény megkap egy képet és egy modellt, és visszaadja a top  $K$  legvalószínűbb találatot a beléptető adatbázisból azok megfelelő valószínűségeivel.

A működése:

```

probs, classes = predict(image_path, model)
print(probs)
print(classes)
> [ 0.01558163  0.01541934  0.01452626]
> ['1', '3', '5']

```

Első lépésben a képet kell transzformálni. Fontos, hogy a képet ugyanolyanra alakítsuk, mint a tanítás során.

### 3.1 Képfeldolgozás

A képek betöltéséhez érdemes a hatékony PIL könyvtár függvényeit használni (lásd a [dokumentációt](#)).

Egy függvényben összevonjuk az előfeldolgozási lépéseket, amelyek után a képet már bemenetként oda lehet adni a modellnek. Az előfeldolgozás során ugyanazokat a lépéseket kell végrehajtani, mint a tanítási folyamat során.

A transzformációs lépések:

- átméretezés (ezt lehet két lépésben is tenni, egy 256x256 leskálázás, és utána egy 224x224-es közép kivágás)
- adattranszformáció a színcsatornákra (színcsatornák 0-255 értéktartományban vannak, a modell 0,1 közötti floatokat vár (PIL kép egyszerűen numpy tömbbé alakítható: `np_image = np.array(pil_image)`)
- normalizálás (a modell [0.485, 0.456, 0.406] átlaggal és [0.229, 0.224, 0.225] szórással számol.
- oszlopsorrend átalakítás (PyTorch a színcsatornát első dimenzióként várja, PIL képek esetén viszont az a 3. dimenzió. A dimenziókat át kell sorrendezni, a színcsatorna kerül az első helyre, a többi dimenzió változatlan marad.

`ndarray.transpose` függvény használható.

```

[20]: # Segédfüggvény közép kivágáshoz
def centeredCrop(img, new_height, new_width):
    width = img.width
    height = img.height

    left = np.ceil((width - new_width)/2.)
    top = np.ceil((height - new_height)/2.)

    right = left + new_width
    bottom = top + new_height

    return img.crop((left, top, right, bottom))

def process_image(image):
    ''' Leskálázza, croppolja, normalizálja a PIL képet a PyTorch modellhez,

```

```

        Numpy array-t ad vissza
        '''

        # Képfeldolgozás
        orig_width, orig_height = image.size
        aspect_ratio = orig_width / orig_height
        if (aspect_ratio <= 1):
            new_width = 256
            new_height = new_width / aspect_ratio
        else:
            new_height = 256
            new_width = new_height * aspect_ratio

        transformed_img = centeredCrop(image.resize((int(round(new_width)),
        ↪int(round(new_height)))), 224, 224)
        np_image = np.array(transformed_img)

        # Érték normalizálás (színcsatorna értékek modellhez)
        # [0,255] tartományból az értékek [0,1] tartományba transzformálódnak
        np_image = np_image / 255.0

        # Normalizálás - átlag és szórás beállítása
        mean = np.array([0.485, 0.456, 0.406])
        std = np.array([0.229, 0.224, 0.225])
        np_image = (np_image - mean) / std

        np_image = np_image.transpose((2, 0, 1))

        return np_image

```

Érdemes leellenőrizni a képfeldolgozó függvényeket. Egyszerűen kép megadásával és megjelenítésvel.

```

[21]: def imshow_direct(image, ax=None, title=None):
        if ax is None:
            fig, ax = plt.subplots()

        # Matplotlib a színcsatornát a 3. dimenzióban várja, a preprocessálás
        ↪elsőre teszi.
        # Vissza kell állítani
        image = image.transpose((1, 2, 0))

        # A normalizálások visszaállítása
        mean = np.array([0.485, 0.456, 0.406])
        std = np.array([0.229, 0.224, 0.225])
        image = std * image + mean

```

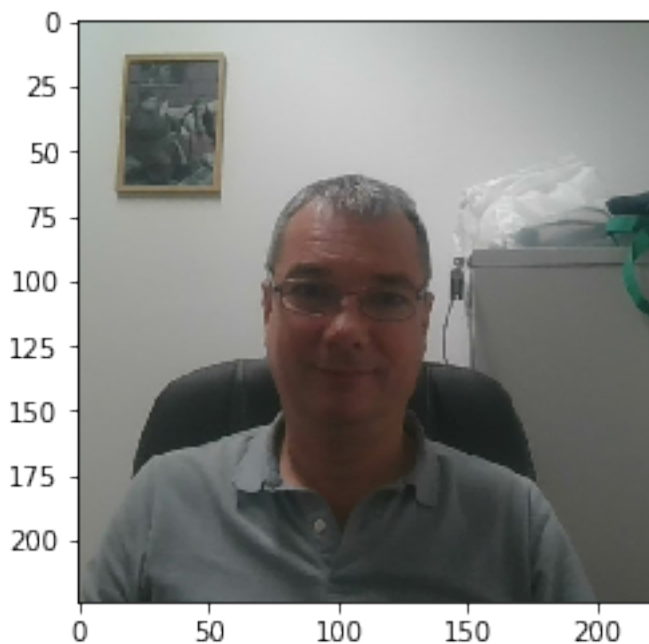
```
# Zajos kép elkerülése érdekében clip-pelni kell 0,1 közé
image = np.clip(image, 0, 1)

ax.imshow(image)

return ax
```

```
[22]: img = Image.open(test_dir + '/004.Gyula/WIN_20170619_16_08_18_Pro.jpg')
img = process_image(img)
imshow_direct(img)
```

```
[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb503c6a790>
```



### 3.2 Predikció

Osztályozáskor a leggyakoribb megközelítés, hogy a top 3-5 legvalószínűbb osztályt adja vissza a prediktor. A beléptető rendszer esetében az osztályok a beengedendő személyek. A beadott mintát a rendszer megpróbálja besorolni az ismert osztályokba. Érdemes egy küszöbértéket definiálni, amely alatt nem tekintjük felismertnek a képet.

A predikció visszaadja a top- $K$  osztályok indexeit és a hozzájuk tartozó valószínűségeket.

Természetesen a predikció során kiszámítjuk az összes osztály találati valószínűségét, maj abból visszaadjuk a legjobb  $K$ -t.

A top- $K$  számításhoz felhasználjuk a `x.topk(k)` függvényt. A függvény visszaadja a  $k$  legnagyobb valószínűséget és a hozzájuk tartozó indexeket. Az indexek alapján nevekre oldható fel az osztály



illeszkedés.

Mint korábban láttuk, a predikció egy kép elérési utat vár.

```
probs, classes = predict(image_path, model)
print(probs)
print(classes)
> [ 0.01558163  0.01541934  0.01452626]
> ['70', '3', '45']
```

Az arcok felismerésére az opencv CascadeClassifier osztályát használjuk, a haarcascade\_frontalface\_alt.xml tanított arcfelismerő modellel. A képeken így be tudjuk azonosítani az arcokat és ha a prediktor megtalált, felismert egy személyt, a képen feliratozni tudjuk, illetve kerettel ki tudjuk jelölni vizuálisan az arcot.

```
[23]: # Engedélyezzük a böngészőbe rajzolt diagramokat
      %matplotlib inline

      # Betöltjük az arcfelismerő modellt.
      face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.
      ↪xml')
```

```
[24]: # Udacity helper, kicsit módosítva
      def view_classify(img, label, ps, labels, topk=5):
          ''' Function for viewing an image and it's predicted classes.
              '''
          # Egy refaktor esetén ennek máshol lenne a helye és paraméterként kellene ↪
          ↪átadni
          recognition_limit = 0.3

          ps = ps.cpu().data.numpy().squeeze()

          #img = img.numpy().transpose((1, 2, 0)).squeeze()

          orig_img = img

          # A korábbi normalizálások visszavonása
          mean = np.array([0.485, 0.456, 0.406])
          std = np.array([0.229, 0.224, 0.225])
          img = std * img + mean

          # A képeket 0-1 közé kell klippelni, egyébként zajnak néznének ki
          img = np.clip(img, 0, 1)

          ### A keretezés és a név megadás itt történik
          # A könyvtár grayscale-lel dolgozik, BGR formátumot konvertálni kell
          #img=img.astype(np.float32)
          gray = cv2.cvtColor(orig_img, cv2.COLOR_BGR2GRAY)
```

```

#gray = np.array(gray, dtype='uint8')
# Az arcok megkeresése a képekben
faces = face_cascade.detectMultiScale(gray)

# A megtalált arcok bekeretező téglalapjai
for (x,y,w,h) in faces:
    # A téglalapokat hozzáadjuk a képhez
    cv2.rectangle(orig_img, (x,y), (x+w,y+h), (255,0,0), 2)
    name = labels[0]
    if ps[0] < recognition_limit:
        name = "Ismeretlen"
    # Feliratozzuk a képet a felismert személy nevével
    cv2.rectangle(orig_img, (x, y), (x+w, y+h), (0, 255, 0), 2)
    y = y - 15 if y - 15 > 15 else y + 15
    cv2.putText(orig_img, "[{}] - {:.06.2f}%".format(name, ps[0]*100), (x,
→y), cv2.FONT_HERSHEY_SIMPLEX,
        0.75, (0, 0, 255), 2)

#img = img.transpose((1, 2, 0))
#resize_(3, 224, 224)

fig, (ax1, ax2) = plt.subplots(figsize=(6,9), nrows=2)
ax1.imshow(orig_img)
ax1.axis('off')
ax1.set_title(label)
ax2.barh(np.arange(topk), ps)
ax2.set_aspect(0.1)
ax2.set_yticks(np.arange(topk))

ax2.set_yticklabels(labels, size='small');

ax2.set_title('Osztály valószínűségek')
ax2.set_xlim(0, 1.1)

plt.tight_layout()

def imshow(image, ax=None, title=None):
    if ax is None:
        fig, ax = plt.subplots()

    # PyTorch tenzorok esetén a színcsatorna az első dimenzió
    # Viszont matplotlib a harmadik dimenzióban várja a színcsatornát
    # Ennek megfelelően kell transzformálni
    image = image.numpy().transpose((1, 2, 0))

```

```

# Előfeldolgozás visszavonása
mean = np.array([0.485, 0.456, 0.406])
std = np.array([0.229, 0.224, 0.225])
image = std * image + mean

# A képeket 0-1 közé kell klippelni, egyébként zajnak néznének ki
image = np.clip(image, 0, 1)
ax.set_title('Az eredeti kép')
ax.imshow(image)

return ax

def predict(image_path, model, topk=5, device='cpu'):
    ''' Egy kép osztályát (osztályait) adja meg tanított NN-nel
    '''

    orig_img = Image.open(image_path)

    img = process_image(orig_img)

    orig_img = np.array(orig_img)

    img = torch.from_numpy(img).unsqueeze(0).to(device).float()

    # Az osztály valószínűségek kiszámítása (softmax)
    with torch.no_grad():
        output = model.forward(img)

    ps = torch.exp(output)
    ps_topk, indices_topk = torch.topk(ps, topk)
    labels_topk = []
    for i in indices_topk.cpu().numpy()[0]:
        labels_topk.append(cat_to_name.get(str(i+1)))

    class_of_img = 'Osztályozandó kép'

    probabilities = ps_topk.cpu().numpy()[0]
    print("[Valószínűség]\t[Felismert személy]")
    for i in range(len(labels_topk)):
        print("{:06.2f}%\t{}".format(probabilities[i]*100, labels_topk[i]))

    # A kép és a valószínűségek megjelenítése
    #view_classify(img.cpu().squeeze(), class_of_img, ps_topk, labels_topk,
    ↪topk)
    view_classify(orig_img, class_of_img, ps_topk, labels_topk, topk)

    imshow(img.cpu().squeeze())

```

```

# Segédfüggvény a következő kép generálásához egyszerű adathalmazból
def predict_iter_helper(dataiter, inferenceloader, model, device, topk=3):

    images, labels = dataiter.next()
    images, labels = (images.to(device), labels.to(device))
    img = images

    # Az osztály valószínűségek kiszámítása (softmax)
    with torch.no_grad():
        output = model.forward(img)

    predicted = torch.max(output.data, 1)
    ps = torch.exp(output)

    ps_topk, indices_topk = torch.topk(ps, topk)

    labels_topk = []
    for i in indices_topk.cpu().numpy()[0]:
        labels_topk.append(cat_to_name.get(str(i+1)))

    class_of_img = 'Osztályozandó kép'

    labels_topk = []
    for i in indices_topk.cpu().numpy()[0]:
        labels_topk.append(cat_to_name.get(str(i+1)))

    class_of_img = cat_to_name.get(str(labels.cpu()[0].numpy()+1))

    probabilities = ps_topk.cpu().numpy()[0]
    print("[Valószínűség]\t[Felismert személy]")
    for i in range(len(labels_topk)):
        print("[{:06.2f}%]\t[{}]" .format(probabilities[i]*100,
→labels_topk[i]))

    # A kép és a valószínűségek megjelenítése
    # view_classify(img[0].cpu().squeeze(), class_of_img, ps_topk, labels_topk,
→topk)
    imshow(img[0].cpu())

# Indirekt osztályozás, kényelmi használathoz teszteléshez
# A tesztelés megkönnyítéséhez - kép elérési út helyett egy loader-t vár
# és a predict_iter_helper() segédfüggvénnyel új tesztek hajthatók végre
def predict_indirect(inferenceloader, model, device, topk=3):

    model.eval()

```

```

dataiter = iter(inferenceloader)
predict_iter_helper(dataiter, inferenceloader, model, device, topk)

return dataiter

def predict_from_stream(image_from_stream, model, topk=5, device='cpu'):
    ''' Egy kép osztályát (osztályait) adja meg tanított NN-nel - streamből '''

    img = image_from_stream
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_pil = Image.fromarray(img)
    img = process_image(img_pil)

    img = torch.from_numpy(img).unsqueeze(0).to(device).float()

    # Az osztály valószínűségek kiszámítása (softmax)
    with torch.no_grad():
        output = model.forward(img)

    ps = torch.exp(output)
    ps_topk, indices_topk = torch.topk(ps, topk)
    labels_topk = []
    for i in indices_topk.cpu().numpy()[0]:
        labels_topk.append(cat_to_name.get(str(i+1)))

    return(ps_topk.cpu().numpy()[0], labels_topk)

def pure_predict(image_path, model, topk=5, device='cpu'):
    ''' Egy kép osztályát (osztályait) adja meg tanított NN-nel '''

    orig_img = Image.open(image_path)

    img = process_image(orig_img)

    orig_img = np.array(orig_img)

    img = torch.from_numpy(img).unsqueeze(0).to(device).float()

    # Az osztály valószínűségek kiszámítása (softmax)
    with torch.no_grad():
        output = model.forward(img)

    ps = torch.exp(output)
    ps_topk, indices_topk = torch.topk(ps, topk)

```

```

labels_topk = []
for i in indices_topk.cpu().numpy()[0]:
    labels_topk.append(cat_to_name.get(str(i+1)))

probabilities = ps_topk.cpu().numpy()[0]
print("[Valószínűség]\t[Felismert személy]")
for i in range(len(labels_topk)):
    print("{:06.2f}%\t[{}]" .format(probabilities[i]*100,
↪labels_topk[i]))

```

Az osztály indexek az alábbiak szerint mappelődnek a felismerendő személyek neveire:

```
[25]: cat_to_name
```

```
[25]: {'1': 'Gyuszi',
      '2': 'Gabi',
      '3': 'Agi',
      '4': 'Gyula',
      '5': 'Geri',
      '6': 'Gyuri',
      '7': 'Roli',
      '8': 'Zsombor',
      '9': 'Rezso',
      '10': 'Rezsi',
      '11': 'Toncsi'}
```

Egyszerű tesztet hajtunk végre és mérjük az osztályozás idejét a 'timeit.timeit' függvénnyel, hogy lássuk, a gyakorlati felhasználáshoz megfelelő osztályozási időket kapunk-e.

```
[26]: timeit.timeit("pure_predict(test_dir + '/004.Gyula/WIN_20170619_16_08_18_Pro.
↪jpg', model, device='cuda')", number=1, globals=globals())
```

[Valószínűség]	[Felismert személy]
[099.86%]	[Gyula]
[000.12%]	[Gyuszi]
[000.02%]	[Gabi]
[000.00%]	[Roli]
[000.00%]	[Zsombor]

```
[26]: 0.02585265412926674
```

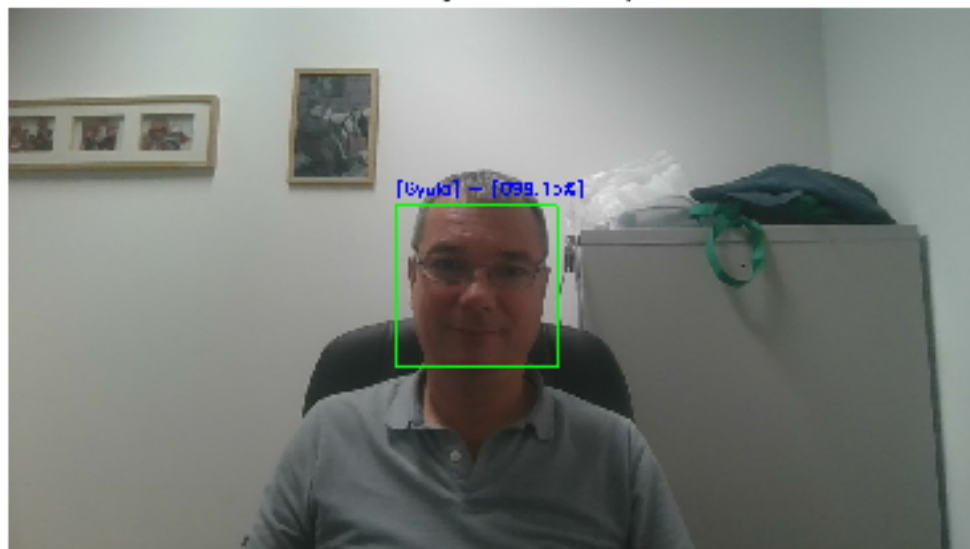
```
[27]: predict(test_dir + '/004.Gyula/WIN_20170619_16_08_18_Pro.jpg', model,
↪device='cuda')
```

[Valószínűség]	[Felismert személy]
[099.15%]	[Gyula]
[000.69%]	[Gyuszi]
[000.15%]	[Gabi]

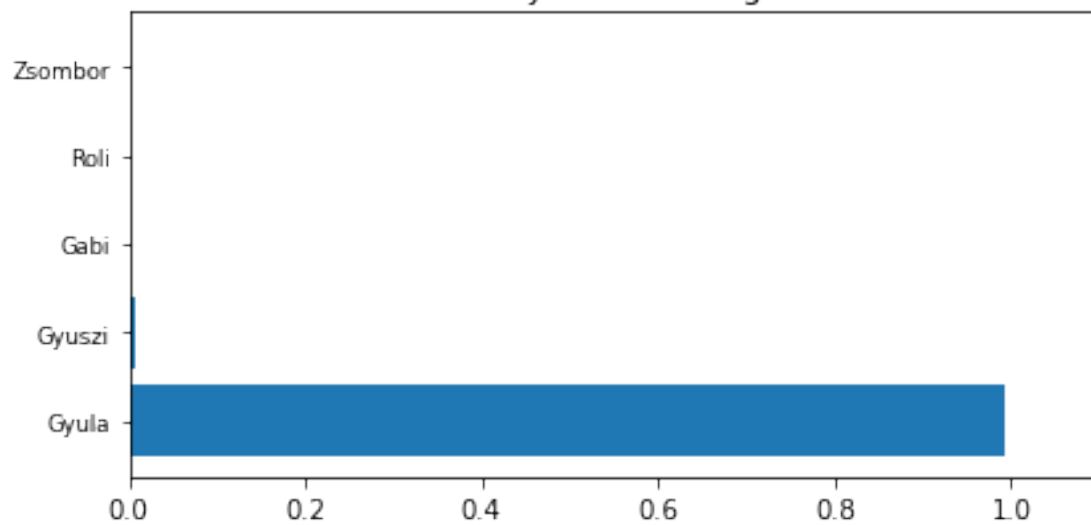
[000.00%]  
[000.00%]

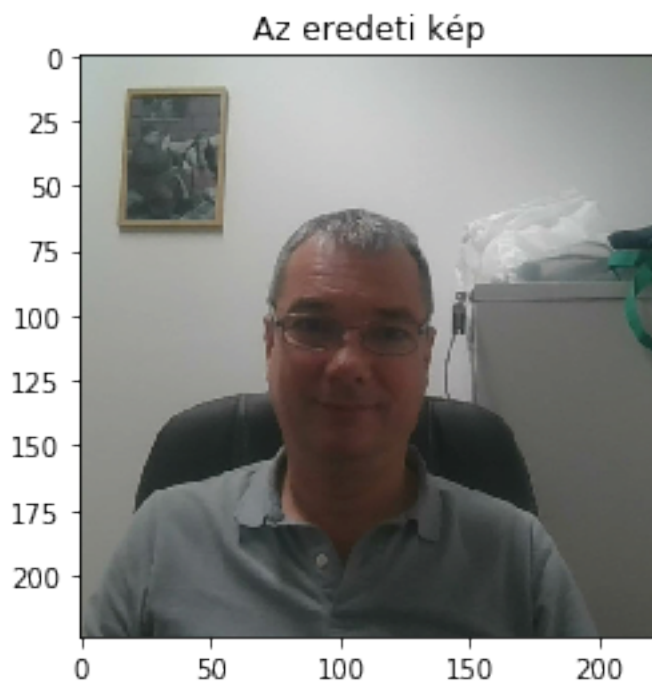
[Roli]  
[Zsombor]

Osztályozandó kép



Osztály valószínűségek





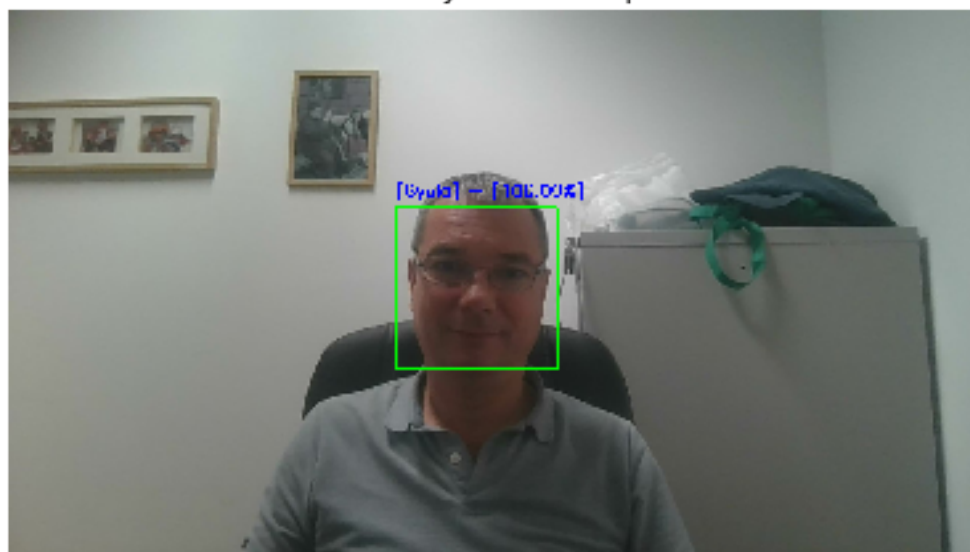
```
[28]: timeit.timeit("predict(test_dir + '/004.Gyula/WIN_20170619_16_08_18_Pro.jpg',
↳model, device='cuda')", number=1, globals=globals())
```

[Valószínűség]	[Felismert személy]
[100.00%]	[Gyula]
[000.00%]	[Gyuszi]
[000.00%]	[Roli]
[000.00%]	[Gabi]
[000.00%]	[Zsombor]

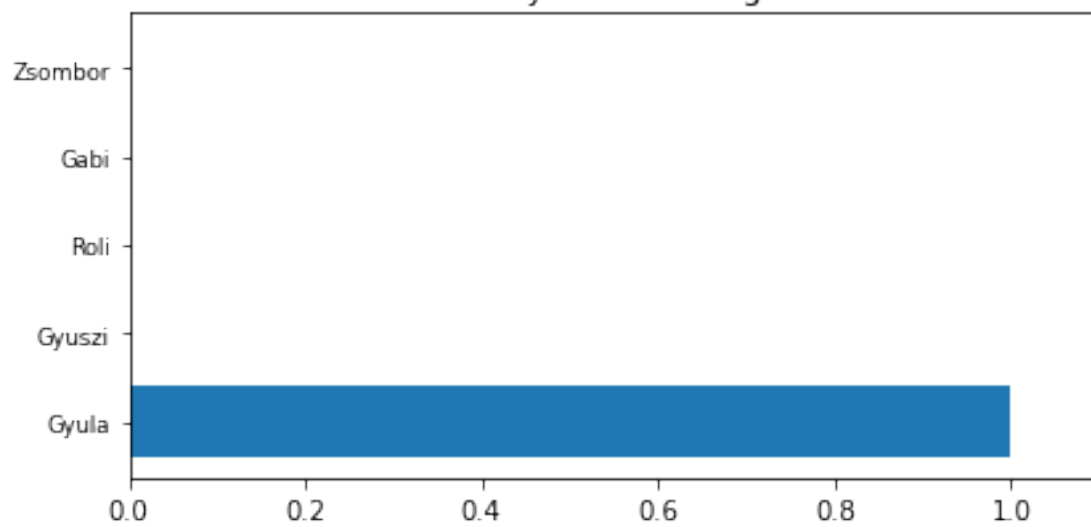
```
[28]: 0.1138538345694542
```

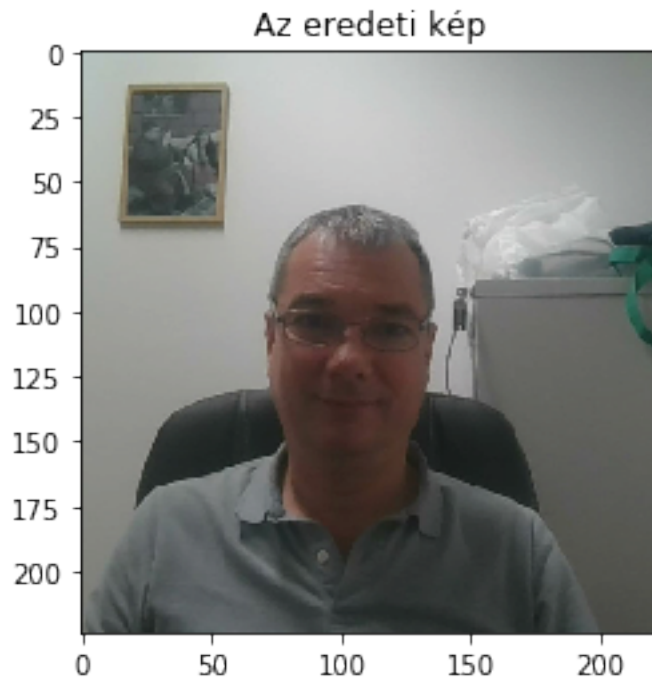


Osztályozandó kép



Osztály valószínűségek

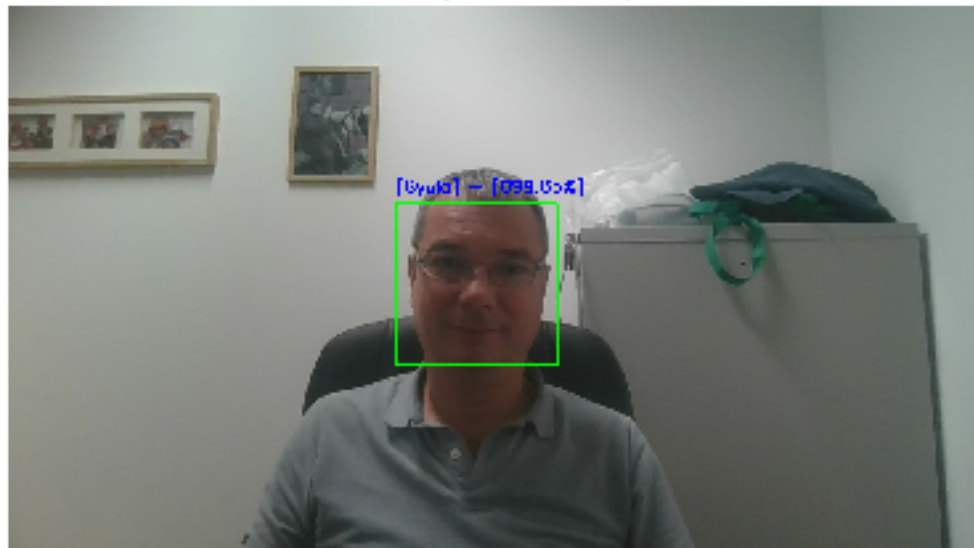




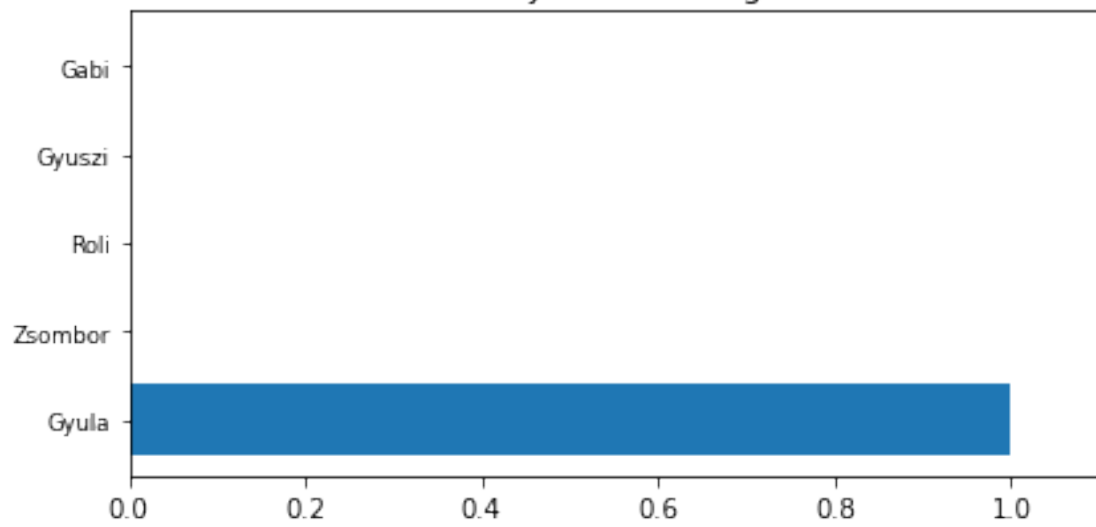
```
[29]: predict(test_dir + '/004.Gyula/WIN_20170619_16_08_18_Pro.jpg', model, device='cuda')
```

[Valószínűség]	[Felismert személy]
[099.85%]	[Gyula]
[000.07%]	[Zsombor]
[000.06%]	[Roli]
[000.02%]	[Gyuszi]
[000.00%]	[Gabi]

Osztályozandó kép



Osztály valószínűségek





Egyszerű kísérlet a kamerával, arc megkeresése és megjelölése a streamen. A kód az alábbi [tutorial](#) kódra épít.

```
[30]: video_capture = VideoStream(src="http://user:password@url:port/video.cgi").  
      ↪start()  
      #cv2.VideoCapture(0)  
  
      cv2.namedWindow('Video', cv2.WINDOW_NORMAL)  
      cv2.resizeWindow('Video', 600, 600)  
  
      while True:  
          # Képkockák kivágása  
          frame = video_capture.read()  
  
          gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  
  
          # Arcok megkeresése  
          faces = face_cascade.detectMultiScale(  
              gray,  
              scaleFactor=1.1,  
              minNeighbors=5,  
              minSize=(30, 30),  
              flags=cv2.CASCADE_SCALE_IMAGE  
          )
```

```

# Arcok bekeretezése
for (x, y, w, h) in faces:
    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Megjelenítés
cv2.imshow('Video', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):
    break

cv2.destroyAllWindows()
video_capture.stop()

```

A fenti megközelítés kiegészítése: - kép kiemelése a streamből - arc megkeresése - predikció végrehajtása a képen - arc bekeretezése - a predikció szerinti legvalószínűbb névvel való címkézés

Érdemes megjegyezni, hogy a tanítás teljes fotókon történt, így jelenleg a teljes képpel való osztályozás sokkal pontosabb, mint a kivágott arccal.

Érdemes lehetne kipróbálni, hogy a tanítási szakaszba szintén beépítsünk egy arcfelismerést, arc kivágást és a kivágott arccal tanítani a hálót. Akkor a streamből kivágott arccal lehetne a predikciót hatékonyan futtatni.

```

[31]: # A video stream inicializálása
      # Időt kell hagyni a kamera szenzornak, hogy bemelegedjen

      print("[INFO] video stream indítása...")

      # lokális webkamerea esetén:
      # vs = VideoStream(src=0).start()
      # Hálózati webkamera elérése

      vs = VideoStream(src="http://user:password@url:port/video.cgi").start()
      writer = None
      time.sleep(2.0)

      # végtelen ciklus a streamből frameket olvasva
      while True:

          # képkocka kivágása

          frame = vs.read()

          # A streamből BGR kódolással jön a képfolyam, át kell konvertálni RGB-re
          # Érdemes leskálázni ésszerű méretre, pl. 750px
          # Ezzel a feldolgozási sebesség és a memória igény jelentősen csökkenthető

          rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

```

```

rgb = imutils.resize(frame, width=750)
r = frame.shape[1] / float(rgb.shape[1])

# Az arcokat bekeretező téglalapok (x, y) koordinátáinak megkeresése
# minden egyes képkockára, majd a predikció elvégzése

boxes = face_recognition.face_locations(rgb,
    model='hog')

#print(labels)
# Korábbi kódrészlet, a zip miatt bennhagyva
#names = labels[0] * len(boxes)

# Az arcokon végig lehet iterálni
for ((top, right, bottom, left),) in zip(boxes):
    # A skálázást az arcok koordinátáira is el kell végezni
    top = int(top * r)
    right = int(right * r)
    bottom = int(bottom * r)
    left = int(left * r)

    # Az arc kihasítása slicing-gal megoldható
    face = rgb[top:bottom, left:right].copy()

    # A fenti bevezetőben említett szerint mivel a tanítás nem közvetlenül
    ↪ az arcokon történt,
    # így itt is a teljes képkockát adjuk át következtetésre
    # A következtetés visszaadja a valószínűségeket és a neveket (top 5)

    ps, labels = predict_from_stream(rgb, model, device='cuda')

    # Az osztályozással megtalált legvalószínűbb nevet a képre írjuk, a
    ↪ keret bal felső sarkához
    # Kiegészítjük a valószínűséggel

    cv2.rectangle(frame, (left, top), (right, bottom),
        (0, 255, 0), 2)
    y = top - 15 if top - 15 > 15 else top + 15
    # cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
    cv2.putText(frame, "[{}] - {:.06.2f}%".format(labels[0], ps[0]*100),
    ↪ (left, y), cv2.FONT_HERSHEY_SIMPLEX,
        0.75, (0, 255, 0), 2)

    cv2.namedWindow('Frame',cv2.WINDOW_NORMAL)
    cv2.resizeWindow('Frame', 600,600)
    cv2.imshow("Frame", frame)

```

```

key = cv2.waitKey(1) & 0xFF

# `q` megnyomásával lehet kilépni a végtelen ciklusból
if key == ord("q"):
    break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
print("[INFO] video stream leállítva...")

```

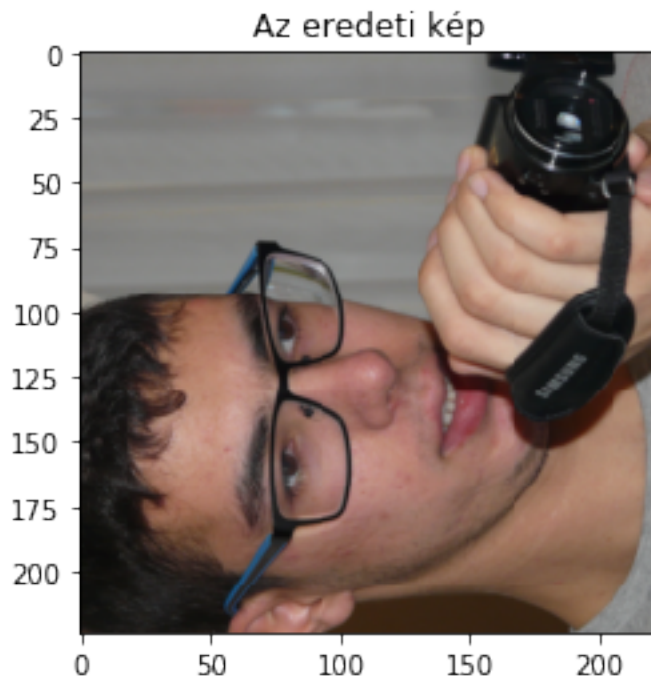
[INFO] video stream indítása...

[INFO] video stream leállítva...

A `predict_indirect()` függvénnyel egy iterátorral tesztelni lehet képekkel a predikciót. Az első hívás után már csak a `predict_iter_helper()` függvényt kell tetszőleges számban (az iterátor hossza korlátoz csak) meghívni ismételt predikciókhoz.

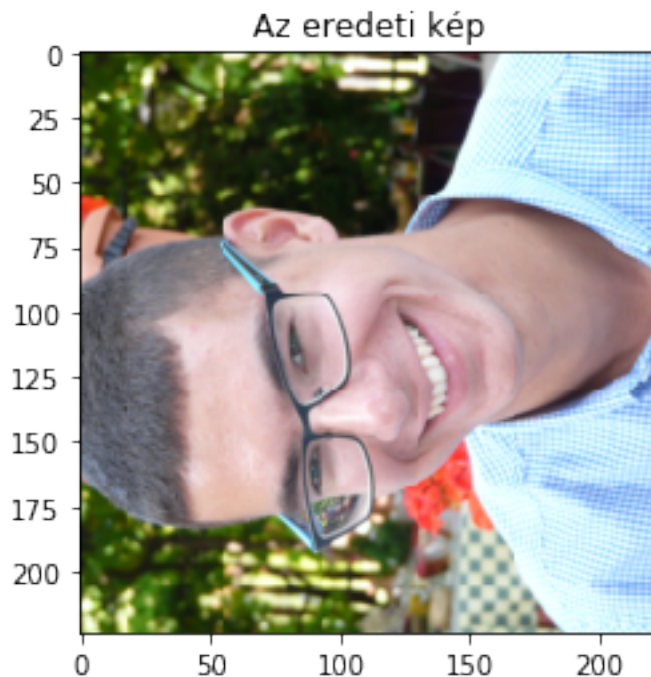
```
[32]: dataiter = predict_indirect(inferenceloader, model, 'cuda')
```

[Valószínűség]	[Felismert személy]
[100.00%]	[Gyuszi]
[000.00%]	[Toncsi]
[000.00%]	[Roli]



```
[33]: device = 'cuda'
      topk = 3
      predict_iter_helper(dataiter, inferenceloader, model, device, topk)
```

[Valószínűség]	[Felismert személy]
[100.00%]	[Gyuszi]
[000.00%]	[Geri]
[000.00%]	[Gabi]



### 3.3 Egy kis ellenőrzés

Érdekes valós képekkel ellenőrizni (lásd fent). Annak ellenére, hogy a validációs pontosság magas, még előfordulhatnak félre osztályozások. Sajnos a fenti példák mutatják, hogy elő is fordulnak.

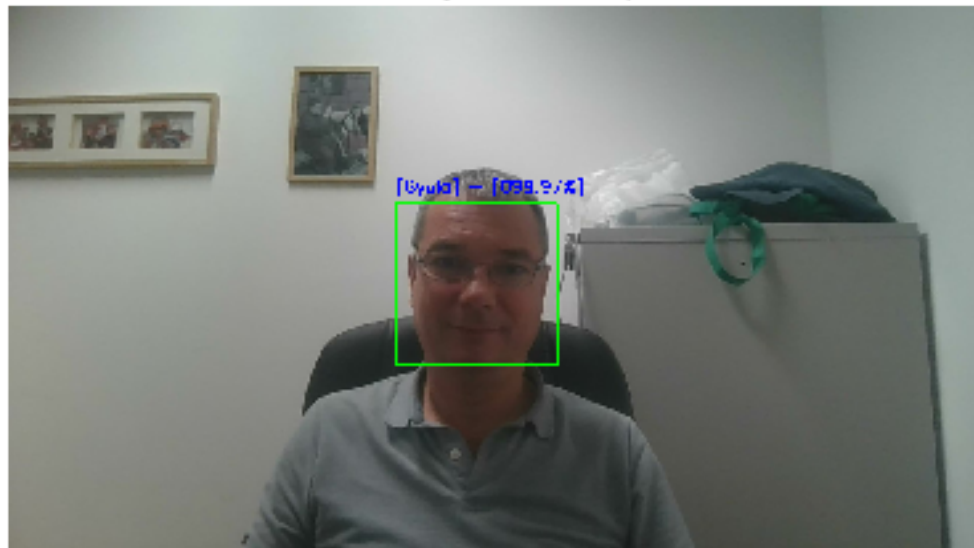
A hasonló arcok, illetve a hasonló vonások félrevezetők lehetnek. Pl. szemüveg, hosszú haj. A rokonai hasonlóság exponálódhat.

```
[34]: predict(test_dir + '/004.Gyula/WIN_20170619_16_08_18_Pro.jpg', model,
             ↪device='cuda')
```

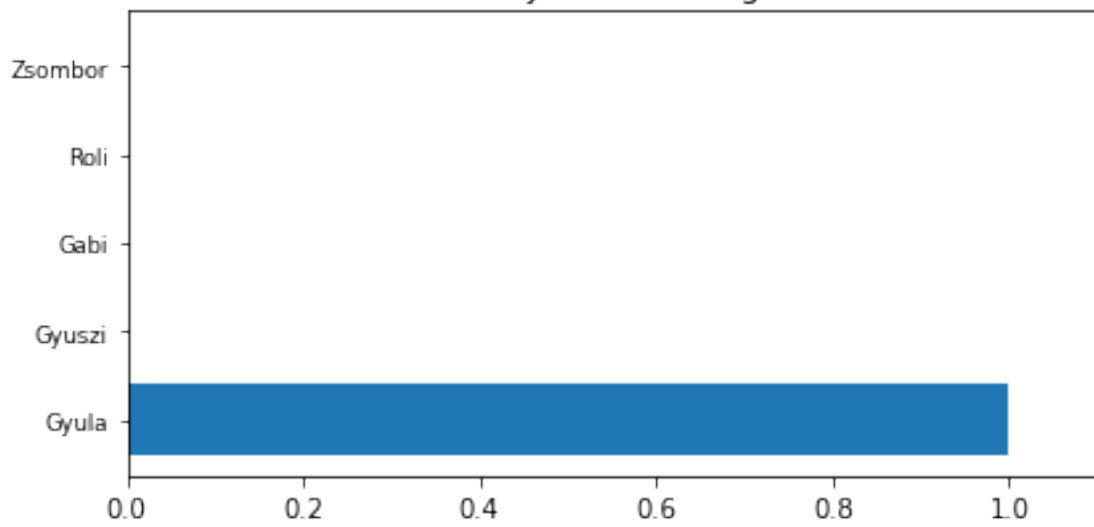
[Valószínűség]	[Felismert személy]
[099.97%]	[Gyula]
[000.02%]	[Gyuszi]
[000.00%]	[Gabi]
[000.00%]	[Roli]
[000.00%]	[Zsombor]



Osztályozandó kép



Osztály valószínűségek



Az eredeti kép

