Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Poročilo: Razvoj modela za prepoznavo oseb

Maribor, junij 2025

Avtor: Jaka Kočar

Vsebina

Kazalo slik

1. Uvod

V projektu smo razvili model za prepoznavo oseb na podlagi obraznih slik. Zaradi boljše učinkovitosti in natančnosti smo se odločili za uporabo metode prenosnega učenja, konkretno smo uporabili že predtrenirani model MobileNetV2. Cilj naloge je zgraditi model, ki za vsak obraz izračuna embedding (Vektor lastnosti, ki predstavlja identiteto osebe). Ti embeddingi se nato shranijo v bazo in pri prijavi v spletno stran uporabijo kot identifikacija osebe.

2. Parametri

- DATASET_DIR
  - Pot do mape s podatki (Slike obrazov)
- BATCH_SIZE
  - Velikost bachov podatkov, ki jih model prejme naenkrat med treningom
- EPOCHS
  - Število ponovitev treninga modela
- OPTIMIZER
  - Optimizacijski algoritem, ki posodablja uteži modela med treningom
- LR
  - Learning rate
- IMG_SIZE
  - Velikost slik, na katero se pretvorijo vse slike pred začetnim učenjem modela
- MAX_IMAGES
  - Maksimalno število slik za eno osebo
- MIN_IMAGES
  - Minimalno število slik za eno osebo
- MAX_PEOPLE
  - Maksimalno število ljudi za treniranje modela
- UNFROZEN_BLOCKS
  - Število slojev, uporabljenih za učenje

2.1.Izbrani parametri

- DATASET_DIR = 'dataset/train'
- BATCH_SIZE = 32
- EPOCHS = 50
- OPTIMIZER = Adam()
- LR = 1e-4
- IMG_SIZE = (224, 224)
- MAX_IMAGES = 100
- MIN_IMAGES = 50
- MAX_PEOPLE = 100
- UNFROZEN_BLOCKS = 10

3. Izbira slik in podatkov

- Za učenje modela smo uporabili znani podatkovni nabor VGG Face Dataset. Gre za javno dostopen nabor, ki vsebuje več kot 2,6 milijona obraznih slik za več kot 2600 znanih oseb.

- Razlogi za izbiro VGG Face Dataset:
    - Dovolj velik in raznolik za učinkovito učenje
    - Vsaka oseba ima več slik, kar omogoča dobro učenje med raznolikostmi istega obraza
    - Primeren za transfer learning, saj ima podobne slike, kot so zajete s kamere

- Priprava podatkov:
    - Iz celotnega nabora smo izbrali podmnožico oseb, pri čemer smo pazili, da ima vsaka oseba najmanj 50 in največ 100 slik
    - Vse slike smo spremenili na dimenzije 224x224, kar ustreza vhodu v MobileNetV2
    - Slike razdelili na učni del (80%) in validacijski del (20%)

4. Priprava podatkov za strojno učenje modela

Za učinkovito učenje modela smo morali najprej ustrezno pripraviti slikovne podatke. Proces priprave podatkov obsega več korakov.

4.1.Filtriranje in izbira slik

Za izbiro slik smo uvedli naslednji kriterij:

- Osebe morajo imeti med MIN_IMAGES in MAX_IMAGES slik
- Skupno število oseb je omejeno z MAX_PEOPLE

To pomeni, da se ustvari seznam oseb, ki ustrezajo pogojem, in snotraj seznama pot do posameznih slik. Vsaki osebi se nato pripiše oznaka (label). Za to implementacijo se uporablja funkcija »get_image_paths_and_labels(root_dir)«.

```python
def get_image_paths_and_labels(root_dir):
    classes = sorted([d for d in os.listdir(root_dir) if os.path.isdir(os.path.join(root_dir, d))])  #Get sorted list of class folders

    filtered_classes = []                                          #List of class names that meet image count criteria
    class_image_dict = {}                                          #Dictionary to store filtered image paths per class

    for cls in classes:
        cls_dir = os.path.join(root_dir, cls)                      #Path to class folder
        imgs = [os.path.join(cls_dir, f) for f in os.listdir(cls_dir)
                if f.lower().endswith(('.png', '.jpg', '.jpeg'))]

        if MIN_IMAGES <= len(imgs) <= MAX_IMAGES:
            class_image_dict[cls] = imgs
            filtered_classes.append(cls)

    #Limit number of classes to MAX_PEOPLE
    max_classes = min(len(filtered_classes), MAX_PEOPLE)
    selected_classes = filtered_classes[:max_classes]              #Take only the first N classes

    all_paths = []
    all_labels = []
    classes_final = []

    for idx, cls in enumerate(selected_classes):
        imgs = class_image_dict[cls]
        all_paths.extend(imgs)
        all_labels.extend([idx] * len(imgs))
        classes_final.append(cls)

    return all_paths, all_labels, classes_final  #Return final filtered paths, labels, class names
```

Slika 4.1: Pridobitev in filtriranje slik

4.2.Branje in obdelava slik

Vsaka slika je najprej prebrana z OpenCV in pomanjšana na dimenzije določene z IMG_SIZE. Nato sledi augmentacija (Če je set slik za treniranje in NE validacijo), ter normalizacija z uporabo funkcije mobilenet_v2.preprocess_input, kar je nujno za kompatibilnosti z modelom MobileNetV2.

```
def parse_and_preprocess(path, label, augment=False):
    def _load_and_preprocess(path_str):
        img = cv.imread(path_str.decode(), cv.IMREAD_COLOR)     #Read with OpenCV
        img = cv.resize(img, IMG_SIZE)                          #Resize to target size

        if augment:
            img = custom_augment(img)                           #Apply custom augmentation

        img = img.astype(np.float32)
        img = tf.keras.applications.mobilenet_v2.preprocess_input(img)  #Normalize for MobileNetV2
        return img

    img = tf.py_function(func=_load_and_preprocess, inp=[path], Tout=tf.float32)
    img.set_shape((*IMG_SIZE, 3))  #Set shape to IMG_SIZE
    return img, label
```

Slika 4.2: Branje in obdelava slik

4.3.Augmentacija slik

Da bi povečali robustnost modela na razlikah v izrazu, svetlobi, nagibu in kakovosti slike, smo uporabili lastne augmentacije, ki vsebujejo naključne transformacije:

- Rotacija (do ± 20°)
- Sprememba svetlosti (± 40)
- Horizontalno zrcaljenje (s 50% verjetnostjo)
- Premik po X/Y osi (± 10 pikslov)
- Zameglitev z gaussovim jedrom (med 0.5 in 1.5)

```
def custom_augment(img):
    img = rotate_img(img, angle=np.random.uniform(-20, 20))                          #Random rotation
    img = change_brightness(img, factor=np.random.uniform(-40, 40))                  #Random brightness
    img = mirror_img(img) if np.random.rand() > 0.5 else img                         #Random horizontal flip
    img = move_img(img, x=np.random.randint(-10, 10), y=np.random.randint(-10, 10))  #Random translation
    img = filter_with_gausso_core(img, sigma=np.random.uniform(0.5, 1.5))            #Optional blur
    return img
```

Slika 4.3: Augmentacija slik

5. Arhitektura modela

Za izračun embeddingov uporabljamo MobileNetV2, kot backbone v kombinaciji z dodatnim slojem za generiranje embeddingov in klasifikacijskim izhodom.

5.1.Hrbtenica (Backbone)

Kot osnovni ekstraktor značilk uporabljamo MobileNetV2 z že naučenimi utežmi na podatkovni zbirki ImageNet. Vrhnji klasifikacijski sloj je odstranjen (include_top=False), namseto njega uporabljamo prostorsko povprečje preko »pooling=avg«. S tem pridobimo vektorski izhod, ki služi za dodatno obdelavo

```
inp = Input(shape=input_shape)  #Define the input layer
backbone = MobileNetV2(input_tensor=inp, include_top=False, pooling='avg', weights='imagenet')  #Load pre-trained MobileNetV2 without top layer
x = backbone.output                                    #Get output from backbone
emb = Dense(EMBED_DIM, activation=None, name='embedding')(x)    #Add embedding layer without activation
```

Slika 5.1 Backbone modela

5.2.Dodane plasti in izhod

Na izhod hrbtenice dodamo gosto plast, ki služi kot embedding. Če je vključen klasifikacijski izhod, se nanj doda še plast Dense, tako da model lahko vrača samo embeddinge ali pa embeddinge skupaj s klasifikacijo.

```
if num_classes:
    logits = Dense(num_classes, use_bias=False, name='logits')(emb)
```

Slika 5.2: Dodatna plast za klasifikacijo

5.3.Zamrzovanje in učenje slojev

Privzeto zamrznemo vse sloje modela, s čimer ohranimo že prednaučene značilnosti. Za izboljšanje rezultatov z fine-tunningom delno odmrznemo zadnje konvolucijske bloke, določene s parametrom UNFROZEN_BLOCKS. S tem pristopom izkoristimo prednaučeno znanje, hkrati pa omogočimo dodatno učenje pomembnejših globljih značilnosti.

```
for layer in backbone.layers:
    layer.trainable = False  #Freeze all layers in backbone

#Unfreeze the last few convolutional blocks
blocks = [l for l in backbone.layers if 'conv' in l.name or 'block' in l.name]  #Select conv or block layers
for block in blocks[-UNFROZEN_BLOCKS:]:
    block.trainable = True                                    #Unfreeze last N blocks for fine-tuning
```

Slika 5.3: Zamrzovanje in učenje določenih slojev

5.4.Kompilacija modela

Model nato kompiliramo z optimizatorjem Adam, in učno stopnjo določeno s parametrom LR. Če se uporablja klasifikacijski izhod, določimo funkcijo izgube »SparseCatagoricalCrossentrophy« in spremljamo točnost. Če model vrne le embeddinge, se klasifikacijski del ne uporablja.

```
    model = Model(inputs=inp, outputs=[emb, logits])                #Output both embedding and classification logits

    model.compile(
        optimizer=Adam(learning_rate=LR),                           #Set optimizer
        loss=[None, SparseCategoricalCrossentropy(from_logits=True)],   #Use crossentropy for classification only
        metrics=[None, 'accuracy']                                  #Track accuracy for classification
    )
else:
    model = Model(inputs=inp, outputs=emb)  #If no classification, output only embeddings
```

Slika 5.4: Kompilacija modela

6. Postopek treniranja

6.1.Priprava podatkov

V tem delu se implementira celotni postopek treniranja modela za prepoznavo obrazov. Najprej se naloži pot do slik, njihove oznake in seznam oseb iz izbranega nabora podatkov. Ti podatki se nato razdelijo na učni in validacijski del v razmerju 80:20, pri čemer je bila razdelitev stratificirana glede na oznake, da ohranimo sorazmerje med razredi.

```
#1. Load all image paths, labels, and the list of class names
all_paths, all_labels, classes = get_image_paths_and_labels(data_root)

#2. Split the data into training and validation sets (80/20 split)
train_paths, val_paths, train_labels, val_labels = train_test_split(
    all_paths, all_labels, test_size=0.2, random_state=42, stratify=all_labels
)

#3. Create TensorFlow datasets for training and validation
train_ds = make_dataset(train_paths, train_labels, augment=True, batch_size=BATCH_SIZE)
val_ds = make_dataset(val_paths, val_labels, augment=False, batch_size=BATCH_SIZE)
```

Slika 6.1: Priprava podatkov

6.2.Gradnja modela

S funkcijo build_model() smo sestavili model, s številom razredov, ki ustreza številu oseb v podatkih, ter z ustrezno vhodno obliko (velikost slik).

```
#4. Build the model
model = build_model(num_classes=len(classes), input_shape=(*IMG_SIZE, 3))
```

Slika 6.2: Build model

6.3.Definicija callback funkcij

- Model Checkpoint
  - Shrani najboljši model glede na validacijsko izgubo
- EarlyStopping
  - Ustavi učenje, če validacijska izguba 5 zaporednih epochov ne izboljša rezultata
- TensorBoard
  - Beleženje statistike treniranja za lažjo vizualizacijo

```
#- Save the best model based on validation loss
checkpoint = ModelCheckpoint(
    filepath='model/mobilefacenet_best.keras',
    save_best_only=True,
    monitor='val_loss'
)

#- Stop training early if validation loss doesn't improve
earlystop = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

#- Log training progress for TensorBoard visualization
tensorboard = TensorBoard(
    log_dir='logs',
    write_graph=True,
    histogram_freq=1
)
```

Slika 6.3: Definicija callback funkcij

6.4.Treniranje in shranjevanje končnega modela

Z model.fit() natreniramo model, ter ga na koncu shranimo. Kot zadnji korak še shranimo samo embedding model.

```
#8. Train the model
model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=[checkpoint, earlystop, tensorboard]
)

#9. Save the final trained model
model.save('model/final_mobilefacenet.keras')

#10. Save an embedding-only version of the model for feature extraction
embedding_model = tf.keras.Model(
    inputs=model.input,
    outputs=model.get_layer('embedding').output
)
embedding_model.save('model/embedding_model.keras')
```

Slika 6.4: Model.fit in shranjevanje