PROJEKAT IZ DUBOKOG UCENJA

1. Opis problema i prikaz dataset-a

Naš projekat predstavlja **problem klasifikacije slika leptira i moljaca**. **Dataset** koji smo odabrali ('*100 butterfly moth species*') sadrži preko 13500 slika raznih vrsta leptira i moljaca. Sve slike iz skupa su dimenzije 224x224 piksela i u RGB formatu. U ovom dataset-u je predstavljeno 100 vrsta insekata, tj. naš problem klasifikacije će imati **100 klasa**. Sledi prikaz jednog primera iz svake klase:

In [181…

```
# Ignorise Tensorflow upozorenja
import warnings
warnings.filterwarnings("ignore")

# Importovanje potrebnih biblioteka
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from PIL import Image
from keras.utils import image_dataset_from_directory
from keras import layers
from keras.models import Sequential, load_model
from keras.optimizers.legacy import Adam
from keras.losses import SparseCategoricalCrossentropy
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score
from keras.callbacks import EarlyStopping
```

In [174…

```
# Prikazivanje jednog odbirka iz svake klase

# Putanja do direktorijuma sa folderima klasa
dataset_dir = '100_butterfly_moth_species/train/'

# Lista svih klasa (foldera) u direktorijumu
classes = os.listdir(dataset_dir)

# Broj kolona u grafiku
num_cols = 8
num_rows = (len(classes) + num_cols - 1) // num_cols

fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 50))
axes = axes.flatten()
for i, class_name in enumerate(classes):
    class_path = os.path.join(dataset_dir, class_name)
    image_file = os.listdir(class_path)[0] # Prva slika u direktorijumu za tu klasu
    image_path = os.path.join(class_path, image_file)
    image = Image.open(image_path)
    axes[i].imshow(image)
    axes[i].set_title(class_name)
    axes[i].axis('off')

# Uklanja nepopunjene subplot-ove
for j in range(len(classes), num_rows * num_cols):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

ADONIS

AFRICAN GIANT SWALLOWTAIL

AMERICAN SNOOT

AN 88

APPOLLO

ARCIGERA FLOWER MOTH

ATALA

ATLAS MOTH

BANDED ORANGE HELICONIAN

BANDED PEACOCK

BANDED TIGER MOTH

BECKERS WHITE

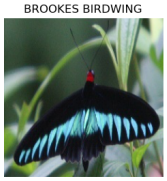
BIRD CHERRY ERMINE MOTH

BLACK HAIRSTREAK

BLUE MORPHO

BLUE SPOTTED CROW

BROOKES BIRDWING

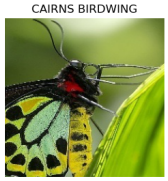


BROWN ARGUS

BROWN SIPROETA

CABBAGE WHITE

CAIRNS BIRDWING

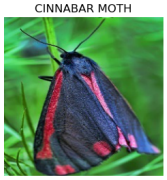


CHALK HILL BLUE

CHECQUERED SKIPPER

CHESTNUT

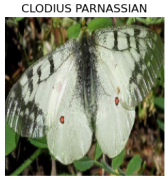
CINNABAR MOTH



CLEARWING MOTH

CLEOPATRA

CLODIUS PARNASSIAN

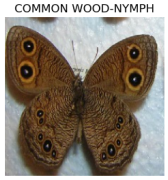


CLOUDED SULPHUR

COMET MOTH

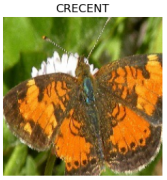
COMMON BANDED AWL

COMMON WOOD-NYMPH



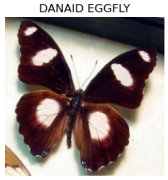
COPPER TAIL

CRECENT



CRIMSON PATCH

DANAID EGGFLY



EASTERN COMA

EASTERN DAPPLE WHITE

EASTERN PINE ELFIN

ELBOWED PIERROT

EMPEROR GUM MOTH

GARDEN TIGER MOTH

GIANT LEOPARD MOTH

GLITTERING SAPPHIRE

GOLD BANDED

GREAT EGGFLY

GREAT JAY

GREEN CELLED CATTLEHEART

GREEN HAIRSTREAK

GREY HAIRSTREAK

HERCULES MOTH

HUMMING BIRD HAWK MOTH

INDRA SWALLOW

IO MOTH

Iphiclus sister

JULIA

LARGE MARBLE

LUNA MOTH

MADAGASCAN SUNSET MOTH

MALACHITE

MANGROVE SKIPPER

MESTRA

METALMARK

MILBERTS TORTOISESHELL

MONARCH

MOURNING CLOAK

OLEANDER HAWK MOTH

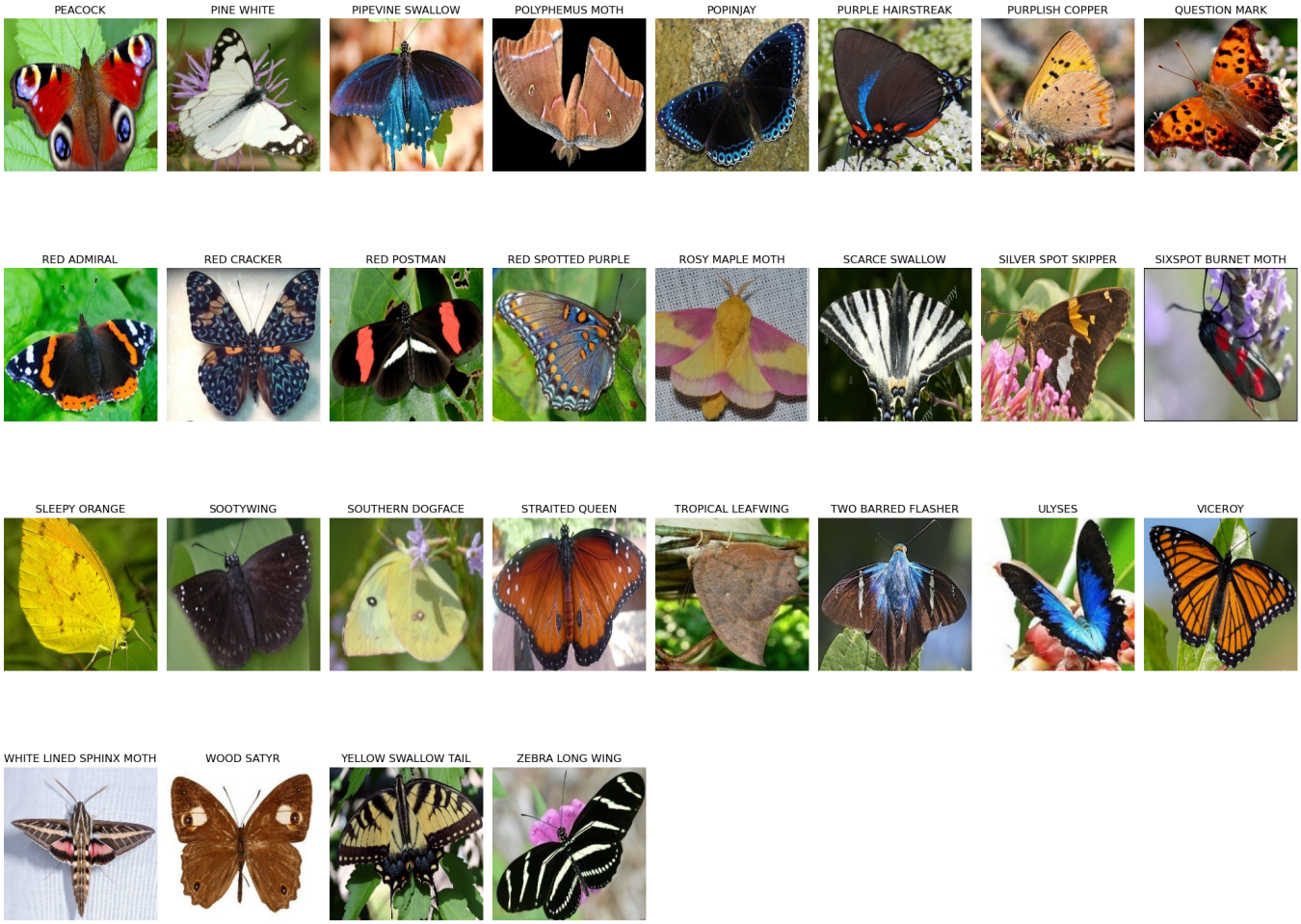
ORANGE OAKLEAF

ORANGE TIP

ORCHARD SWALLOW

PAINTED LADY

PAPER KITE



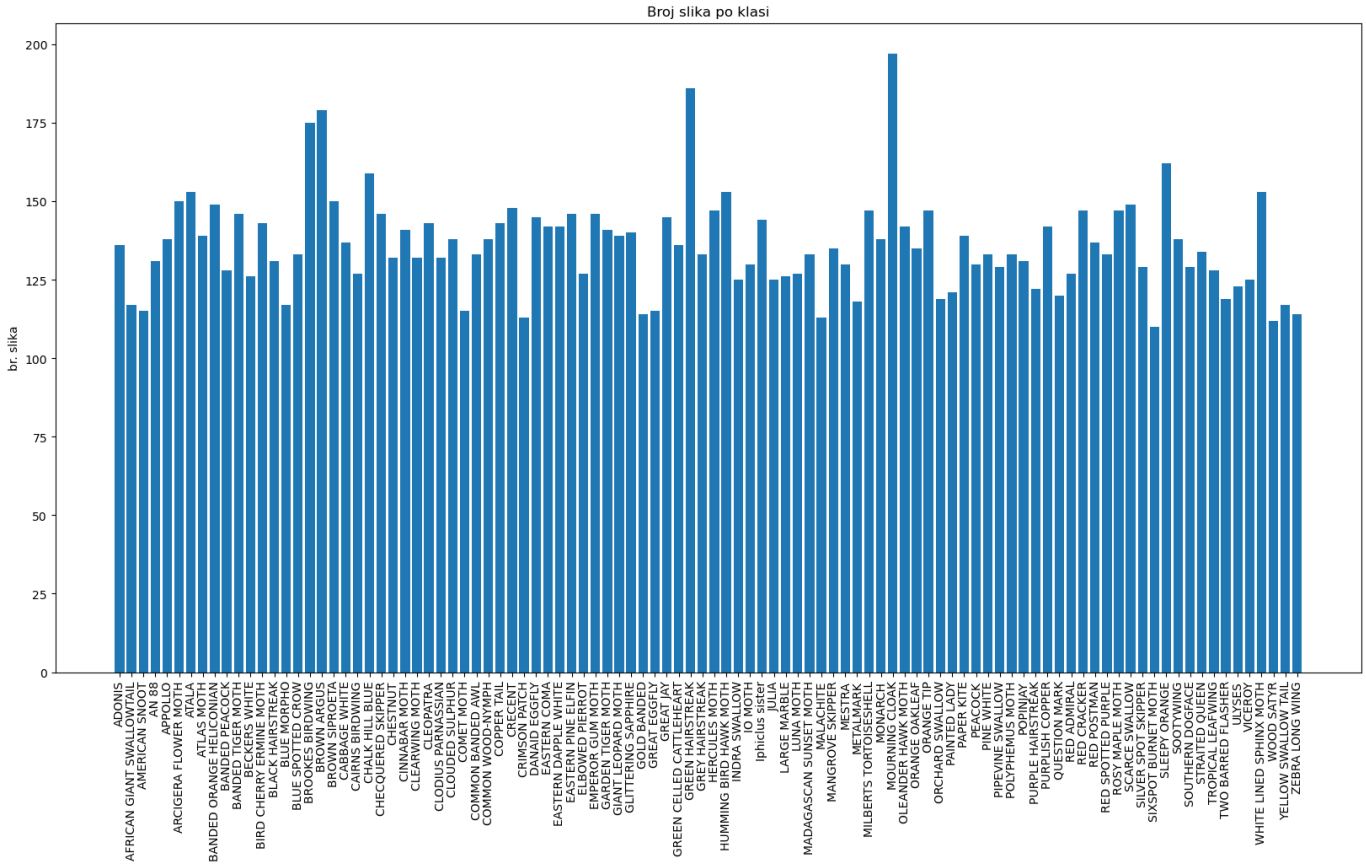
2. Balansiranost klasa

Sada ćemo prikazati broj slika po klasi. Iako su skupovi u našem dataset-u već podeljeni, mi ćemo prikazati sabrano koliko ima slika po klasi kroz sva tri skupa.

Zaključak: Klase su **dobro balansirane**. Određene klase imaju malo više odbiraka, ali ovo možemo zanemariti i smatrati da sve klase imaju približan broj odbiraka.

In [177…

```
# Prikaz odbiraka po klasama

# Direktorijum sa dataset-om
dataset_dir = '100_butterfly_moth_species'

# Recnik u kome cemo cuvati br slika po klasi
class_counts = {}

# Za svaku klasu odredjujemo broj slika u podfolderima
for class_name in classes:
    class_train_path = os.path.join(train_path, class_name)
    class_valid_path = os.path.join(valid_path, class_name)
    class_test_path = os.path.join(test_path, class_name)
    num_images = len(os.listdir(class_train_path)) + len(os.listdir(class_valid_path)) + len(
    class_counts[class_name] = num_images

# Prikaz histograma
plt.figure(figsize=(20, 10))
plt.bar(class_counts.keys(), class_counts.values());
plt.ylabel('br. slika');
plt.title('Broj slika po klasi');
plt.xticks(rotation=90);
```



3. Podela podataka na odgovarajuće skupove

Pri obučavanju neuralnih mreža, skup podataka se obavezno mora **podeliti** na:

- skup za obučavanje,
- skup za validaciju i
- skup za testiranje.

Podela na skupove sprečava pojavu **preobučavanja**. Takođe nam omogućava da istreniramo našu mrežu da ima sposobnost **generalizacije**. Skupove za validaciju i testiranje nikako ne treba koristiti za obučavanje, već samo za proveru već delom obučene ili u potpunosti obučene mreže. Što se tiče raspodele slika kroz skupove, najčešće se vrše podele 60/20/20 ili 80/10/10. Pošto je naš skup podataka izuzetno veliki, validacioni i test skup će imati još manji procenat slika.

Dataset koji smo mi odabrali je **već podeljen** na odgovarajuće skupove:

- za obucavanje (~12500 slika),
- za validaciju (~500 slika) i
- za testiranje (~500 slika).

4. Pretprocesiranje podataka

Za potrebe ovog problema, nismo morali da vršimo puno modifikacija nad originalnim slikama. Izvršili smo sledeće operacije nad ulazima:

- **Skaliranje:** Slike smo smanjili da budu dimenzija 90x90 piksela. Veća dimenzija slike znači da će biti potreban znatno veći broj parametara u mreži. Pošto naš problem ima veliki broj klasa i da bismo smanjili vreme potrebno za obučavanje, manja dimenzija početne slike se pokazalo kao odlično rešenje. Obučavanjem na raznim veličinama slike, ovako odabrana dimenzija ulazne slike je davala najbolji rezultat.

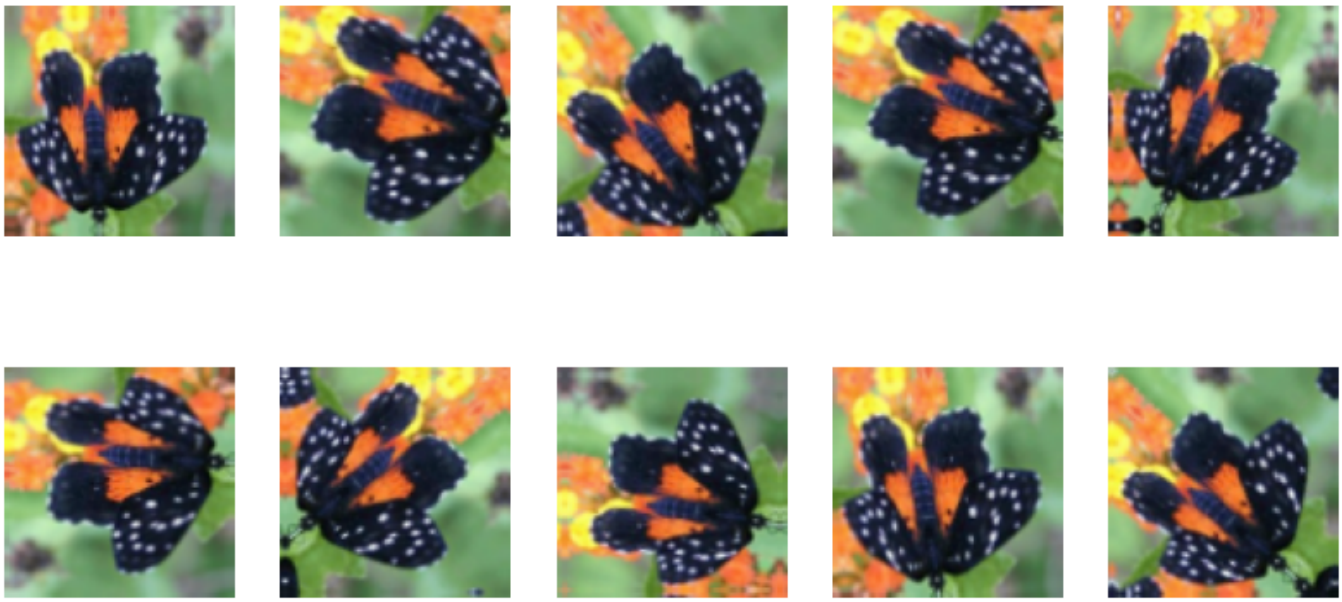
- **Augmentacija** ulazne slike: Kao **ulazni sloj** u mrežu smo dodali da se ulazne slike na razne načine modifikuju. Ovim postižemo veću generalizaciju modela. Od modifikacija nad ulaznom slikom smo uradili sledeće:
  - Nasumično okretanje slike
  - Nasumična rotacija
  - Nasumično zumiranje na sliku
- **Reskaliranje/Normalizacija**: Kao **drugi sloj** mreže smo ubacili reskaliranje, čime se intenziteti piksela ulaznih slika normalizuju.

In [66]:
```
# Ucitavanje dataset-a

# Putanje do skupova za treniranje, validaciju i testiranje
train_path = '100_butterfly_moth_species/train/'
valid_path = '100_butterfly_moth_species/valid/'
test_path = '100_butterfly_moth_species/test/'

# Parametri za podesavanje
batch_size = 64
image_dim = 90
image_size = (image_dim, image_dim)

# Ucitavanje skupova
train_dataset = image_dataset_from_directory(train_path, batch_size=batch_size, image_size=im
valid_dataset = image_dataset_from_directory(valid_path, batch_size=batch_size, image_size=im
test_dataset = image_dataset_from_directory(test_path, batch_size=batch_size, image_size=imag
```

```
Found 12594 files belonging to 100 classes.
Found 500 files belonging to 100 classes.
Found 500 files belonging to 100 classes.
```

In [178…
```
# Augmentacija ulaznih podataka
data_augmentation = Sequential(
  [
    layers.RandomFlip("horizontal", input_shape=(image_size[0], image_size[1], 3)),
    layers.RandomRotation(0.25),
    layers.RandomZoom(0.1),
  ]
)

# Prikaz modifikovane slike
N = 10
plt.figure(figsize=(10,5))
for img, lab in train_dataset.take(1):
    plt.suptitle(f'Modifikovan primer klase {class_names[lab[0]]}', fontsize=16)
    for i in range(N):
        aug_img = data_augmentation(img)
        plt.subplot(2, int(N/2), i+1)
        plt.imshow(aug_img[0].numpy().astype('uint8'))
        plt.axis('off')

# Sakriva upozorenja
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Modifikovan primer klase CRIMSON PATCH



5. Kreiranje modela i obučavanje

Pošto naš problem predstavlja problem klasifikacije više klasa, odabrali smo za kriterijumsku funkciju **SparceCategoricalCrossEntropy**.

Za aktivacione funkcije smo najčešće koristili **ReLu** aktivacionu funkciju, zato što uvodi dovoljno nelinearnosti u naš model, a računarski se jednostavno realizuje što ubrzava proces obučavanja. Osim nje smo u poslednjem sloju mreže koristili **SoftMax** aktivacionu funkciju, koja ima funkciju da klasifikuje na više klasa.

Sledi opis arhitekture mreže po slojevima:

- **Augmentacioni sloj**: Opisali smo ga kao korak u pretprocesiranju podataka
- **Reskaliranje**: Takođe predstavlja korak pretprocesiranja podataka
- **Konvolucioni slojevi**: Odabrali smo da naša mreža ima **3 konvoluciona sloja**, pri čemu nakon svakog Conv sloja primenjujemo **MaxPooling**. Ideja je da prvi konvolucioni sloj vrši najjednostavnije operacije nad ulaznim slikama, pa smo za njega odabrali dimenzije filtra 3x3. Svaki sledeći sloj treba da ima mogućnost dalje detekcije složenijih obrazaca ulazne slike. Zato smo odabrali da 2. i 3. sloj imaju respektivno po filtre dimenzija 4x4 i 5x5. Ova odluka je bila kompromis između složenijeg (sposobnijeg) modela i bržeg obučavanja. Kada bismo odabrali veće filtre u konvolucionim slojevima, imali bismo znatno veći broj parametara mreže, pa bi se vreme obučavanja višestruko povećalo.
- **FullyConnected slojevi**: Naš model ima 3 ovakva sloja, pri čemu nakon prva dva uvodimo **Dropout** slojeve (sa verovatnoćama odbacivanja od 0.5 i 0.3 respektivno) koji služe za regularizaciju, dok nakon 3. sloja imamo **Softmax** aktivacionu funkciju.

Kao optimizator smo odabrali **ADAM** zato što ovaj optimizator omogućava da se prenebregnu lokalni minimumi u kriterijumskoj funkciji i efikasan je u njenoj minimizaciji.

Preobučavanje

Preobučavanje je pojava da se neuralna mreža obuči tako da odlično klasifikuje podatke nad kojim je obučena ali ne radi dobro generalizaciju nad podacima van trening skupa. Ovakva mreža neće dobro klasifikovati podatke koje su van njenog trening skupa. Da bi se sprečila ova pojava, uvode se tehnike

koje omogućavaju generalizaciju neuralne mreže. Ovih tehnika ima dosta, a one koje smo mi primenili su sledeće:

- **Pretprocesiranje**: U prvom sloju naše mreže smo uveli augmentaciju. Ovaj sloj ima ulogu da učini ulazne podatke raznovrsnijim, čime se poboljšava generalizacija i otklanja preobučavanje.
- **Dropout slojevi**: Nakon svakog FC sloja smo dodali da se određen procenat odbiraka odbacuje. Na ovaj način se povećava nasumičnost mreže i omogućava bolja generalizacija.
- **Early Stopping**: Ova tehnika se primenjuje u toku obučavanja. Za potrebe ove metode se koristi validacioni skup podataka. Tehnika funkcioniše tako što se prati kriterijumska funkcija nad trening skupom ali i nad validacionim skupom, koji ne ulazi u proces obučavanja. Ukoliko validaciona kriva počne da raste dok trening kriva nastavlja da opada, ustupilo je preobučavanje i tada možemo da prestanemo sa procesom jer će na dalje mreža samo da postane gora u generalizovanju.

In [179…

```
# Uklanja upozorenja
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Broj klasa
num_classes = len(class_names)

model = Sequential([
    # 1. sloj: Augmentacija
    data_augmentation,

    # 2. sloj: Reskaliranje (Normalizacija)
    layers.Rescaling(1./255, input_shape=(image_dim, image_dim, 3)),

    # Konvolucioni slojevi
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(64, 4, padding='same', activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(128, 5, padding='same', activation='relu'),
    layers.MaxPooling2D(),

    layers.Flatten(),

    # FC slojevi
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),

    layers.Dense(128, activation='relu'),
    layers.Dropout(0.3),

    # Izlazni sloj
    layers.Dense(num_classes, activation='softmax')
])

model.summary()

model.compile(Adam(learning_rate=0.001), loss=SparseCategoricalCrossentropy(), metrics='accur
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_8 (Sequential)   (None, 90, 90, 3)         0

 rescaling_4 (Rescaling)     (None, 90, 90, 3)         0

 conv2d_12 (Conv2D)          (None, 90, 90, 32)        896

 max_pooling2d_12 (MaxPooli  (None, 45, 45, 32)        0
 ng2D)

 conv2d_13 (Conv2D)          (None, 45, 45, 64)        32832

 max_pooling2d_13 (MaxPooli  (None, 22, 22, 64)        0
 ng2D)

 conv2d_14 (Conv2D)          (None, 22, 22, 128)       204928

 max_pooling2d_14 (MaxPooli  (None, 11, 11, 128)       0
 ng2D)

 flatten_4 (Flatten)         (None, 15488)             0

 dense_12 (Dense)            (None, 256)               3965184

 dropout_8 (Dropout)         (None, 256)               0

 dense_13 (Dense)            (None, 128)               32896

 dropout_9 (Dropout)         (None, 128)               0

 dense_14 (Dense)            (None, 100)               12900

=================================================================
Total params: 4249636 (16.21 MB)
Trainable params: 4249636 (16.21 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [79]:

```python
# Uklanja upozorenja
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Early stopping za regularizaciju
early_cb = EarlyStopping(monitor='val_loss', mode='min', patience=15, verbose=1, restore_best_

# Obucavanje
history = model.fit(train_dataset,
                    epochs=100,
                    validation_data=valid_dataset,
                    callbacks=[early_cb],
                    verbose=1)
```

```
Epoch 1/100
197/197 [==============================] - 73s 372ms/step - loss: 4.3358 - accuracy: 0.0372 -
val_loss: 3.7698 - val_accuracy: 0.0880
Epoch 2/100
197/197 [==============================] - 75s 377ms/step - loss: 3.7006 - accuracy: 0.1016 -
val_loss: 3.4030 - val_accuracy: 0.1520
Epoch 3/100
197/197 [==============================] - 75s 378ms/step - loss: 3.2253 - accuracy: 0.1724 -
val_loss: 2.7765 - val_accuracy: 0.2700
Epoch 4/100
197/197 [==============================] - 78s 397ms/step - loss: 2.9138 - accuracy: 0.2271 -
val_loss: 2.6178 - val_accuracy: 0.3280
Epoch 5/100
197/197 [==============================] - 74s 375ms/step - loss: 2.6893 - accuracy: 0.2763 -
val_loss: 2.4411 - val_accuracy: 0.3620
Epoch 6/100
197/197 [==============================] - 75s 378ms/step - loss: 2.4923 - accuracy: 0.3213 -
val_loss: 2.6868 - val_accuracy: 0.3440
Epoch 7/100
197/197 [==============================] - 74s 375ms/step - loss: 2.3664 - accuracy: 0.3519 -
val_loss: 1.8927 - val_accuracy: 0.4800
Epoch 8/100
197/197 [==============================] - 75s 381ms/step - loss: 2.2348 - accuracy: 0.3838 -
val_loss: 2.0162 - val_accuracy: 0.4740
Epoch 9/100
197/197 [==============================] - 75s 381ms/step - loss: 2.1449 - accuracy: 0.4037 -
val_loss: 1.7023 - val_accuracy: 0.5380
Epoch 10/100
197/197 [==============================] - 75s 380ms/step - loss: 2.0446 - accuracy: 0.4243 -
val_loss: 1.5711 - val_accuracy: 0.5700
Epoch 11/100
197/197 [==============================] - 74s 373ms/step - loss: 1.9530 - accuracy: 0.4587 -
val_loss: 1.3863 - val_accuracy: 0.6040
Epoch 12/100
197/197 [==============================] - 74s 372ms/step - loss: 1.8625 - accuracy: 0.4747 -
val_loss: 1.5129 - val_accuracy: 0.5660
Epoch 13/100
197/197 [==============================] - 74s 374ms/step - loss: 1.8122 - accuracy: 0.4897 -
val_loss: 1.5718 - val_accuracy: 0.5760
Epoch 14/100
197/197 [==============================] - 74s 374ms/step - loss: 1.7695 - accuracy: 0.5022 -
val_loss: 1.5281 - val_accuracy: 0.5640
Epoch 15/100
197/197 [==============================] - 74s 374ms/step - loss: 1.7136 - accuracy: 0.5145 -
val_loss: 1.4337 - val_accuracy: 0.5960
Epoch 16/100
197/197 [==============================] - 74s 375ms/step - loss: 1.6935 - accuracy: 0.5180 -
val_loss: 1.2007 - val_accuracy: 0.6380
Epoch 17/100
197/197 [==============================] - 73s 372ms/step - loss: 1.6362 - accuracy: 0.5342 -
val_loss: 1.3529 - val_accuracy: 0.6160
Epoch 18/100
197/197 [==============================] - 74s 377ms/step - loss: 1.5854 - accuracy: 0.5432 -
val_loss: 1.1911 - val_accuracy: 0.6600
Epoch 19/100
197/197 [==============================] - 74s 376ms/step - loss: 1.5660 - accuracy: 0.5513 -
val_loss: 1.3772 - val_accuracy: 0.6040
Epoch 20/100
197/197 [==============================] - 74s 376ms/step - loss: 1.4970 - accuracy: 0.5700 -
val_loss: 1.0402 - val_accuracy: 0.6880
Epoch 21/100
197/197 [==============================] - 74s 376ms/step - loss: 1.4644 - accuracy: 0.5699 -
val_loss: 1.1869 - val_accuracy: 0.6700
Epoch 22/100
197/197 [==============================] - 74s 376ms/step - loss: 1.4444 - accuracy: 0.5821 -
val_loss: 1.1674 - val_accuracy: 0.6840
```

```
Epoch 23/100
197/197 [==============================] - 74s 374ms/step - loss: 1.4246 - accuracy: 0.5867 -
val_loss: 1.1746 - val_accuracy: 0.6700
Epoch 24/100
197/197 [==============================] - 74s 377ms/step - loss: 1.3970 - accuracy: 0.5987 -
val_loss: 1.1402 - val_accuracy: 0.6840
Epoch 25/100
197/197 [==============================] - 75s 379ms/step - loss: 1.3665 - accuracy: 0.6022 -
val_loss: 1.1253 - val_accuracy: 0.6800
Epoch 26/100
197/197 [==============================] - 74s 374ms/step - loss: 1.3630 - accuracy: 0.6061 -
val_loss: 1.1698 - val_accuracy: 0.6960
Epoch 27/100
197/197 [==============================] - 74s 375ms/step - loss: 1.3303 - accuracy: 0.6151 -
val_loss: 0.9640 - val_accuracy: 0.7360
Epoch 28/100
197/197 [==============================] - 74s 375ms/step - loss: 1.2875 - accuracy: 0.6278 -
val_loss: 1.0877 - val_accuracy: 0.7120
Epoch 29/100
197/197 [==============================] - 74s 376ms/step - loss: 1.2945 - accuracy: 0.6237 -
val_loss: 1.0262 - val_accuracy: 0.7320
Epoch 30/100
197/197 [==============================] - 74s 374ms/step - loss: 1.2577 - accuracy: 0.6335 -
val_loss: 0.8815 - val_accuracy: 0.7620
Epoch 31/100
197/197 [==============================] - 77s 389ms/step - loss: 1.2351 - accuracy: 0.6441 -
val_loss: 0.9224 - val_accuracy: 0.7400
Epoch 32/100
197/197 [==============================] - 76s 383ms/step - loss: 1.2024 - accuracy: 0.6440 -
val_loss: 0.9131 - val_accuracy: 0.7420
Epoch 33/100
197/197 [==============================] - 74s 374ms/step - loss: 1.1835 - accuracy: 0.6500 -
val_loss: 0.8614 - val_accuracy: 0.7480
Epoch 34/100
197/197 [==============================] - 75s 379ms/step - loss: 1.1659 - accuracy: 0.6540 -
val_loss: 0.9058 - val_accuracy: 0.7500
Epoch 35/100
197/197 [==============================] - 75s 378ms/step - loss: 1.1538 - accuracy: 0.6563 -
val_loss: 0.9654 - val_accuracy: 0.7380
Epoch 36/100
197/197 [==============================] - 74s 375ms/step - loss: 1.1521 - accuracy: 0.6548 -
val_loss: 0.9269 - val_accuracy: 0.7460
Epoch 37/100
197/197 [==============================] - 74s 376ms/step - loss: 1.1557 - accuracy: 0.6617 -
val_loss: 1.0009 - val_accuracy: 0.7280
Epoch 38/100
197/197 [==============================] - 74s 375ms/step - loss: 1.1286 - accuracy: 0.6671 -
val_loss: 0.8387 - val_accuracy: 0.7580
Epoch 39/100
197/197 [==============================] - 75s 378ms/step - loss: 1.1219 - accuracy: 0.6703 -
val_loss: 0.9166 - val_accuracy: 0.7440
Epoch 40/100
197/197 [==============================] - 76s 383ms/step - loss: 1.1065 - accuracy: 0.6717 -
val_loss: 0.7887 - val_accuracy: 0.7920
Epoch 41/100
197/197 [==============================] - 74s 373ms/step - loss: 1.0708 - accuracy: 0.6786 -
val_loss: 0.8060 - val_accuracy: 0.7740
Epoch 42/100
197/197 [==============================] - 74s 376ms/step - loss: 1.0873 - accuracy: 0.6813 -
val_loss: 0.9102 - val_accuracy: 0.7560
Epoch 43/100
197/197 [==============================] - 74s 376ms/step - loss: 1.0605 - accuracy: 0.6855 -
val_loss: 0.7688 - val_accuracy: 0.7900
Epoch 44/100
197/197 [==============================] - 74s 374ms/step - loss: 1.0576 - accuracy: 0.6846 -
val_loss: 0.9154 - val_accuracy: 0.7640
```

```
Epoch 45/100
197/197 [==============================] - 74s 375ms/step - loss: 1.0266 - accuracy: 0.6975 -
val_loss: 1.0931 - val_accuracy: 0.7160
Epoch 46/100
197/197 [==============================] - 73s 372ms/step - loss: 1.0343 - accuracy: 0.6911 -
val_loss: 0.8394 - val_accuracy: 0.7620
Epoch 47/100
197/197 [==============================] - 75s 378ms/step - loss: 1.0223 - accuracy: 0.6984 -
val_loss: 0.7595 - val_accuracy: 0.7840
Epoch 48/100
197/197 [==============================] - 76s 384ms/step - loss: 1.0061 - accuracy: 0.7026 -
val_loss: 0.8783 - val_accuracy: 0.7560
Epoch 49/100
197/197 [==============================] - 74s 376ms/step - loss: 1.0108 - accuracy: 0.7005 -
val_loss: 0.8191 - val_accuracy: 0.7620
Epoch 50/100
197/197 [==============================] - 74s 377ms/step - loss: 0.9935 - accuracy: 0.7054 -
val_loss: 0.8432 - val_accuracy: 0.7680
Epoch 51/100
197/197 [==============================] - 75s 378ms/step - loss: 0.9888 - accuracy: 0.7080 -
val_loss: 0.8189 - val_accuracy: 0.7820
Epoch 52/100
197/197 [==============================] - 74s 376ms/step - loss: 0.9826 - accuracy: 0.7088 -
val_loss: 0.8148 - val_accuracy: 0.7660
Epoch 53/100
197/197 [==============================] - 75s 378ms/step - loss: 0.9645 - accuracy: 0.7118 -
val_loss: 0.8457 - val_accuracy: 0.7720
Epoch 54/100
197/197 [==============================] - 74s 377ms/step - loss: 0.9722 - accuracy: 0.7091 -
val_loss: 0.8122 - val_accuracy: 0.7840
Epoch 55/100
197/197 [==============================] - 74s 377ms/step - loss: 0.9469 - accuracy: 0.7165 -
val_loss: 0.8491 - val_accuracy: 0.7780
Epoch 56/100
197/197 [==============================] - 74s 376ms/step - loss: 0.9542 - accuracy: 0.7209 -
val_loss: 0.9364 - val_accuracy: 0.7660
Epoch 57/100
197/197 [==============================] - 75s 380ms/step - loss: 0.9653 - accuracy: 0.7138 -
val_loss: 0.7433 - val_accuracy: 0.8020
Epoch 58/100
197/197 [==============================] - 74s 373ms/step - loss: 0.9355 - accuracy: 0.7240 -
val_loss: 0.7857 - val_accuracy: 0.7900
Epoch 59/100
197/197 [==============================] - 74s 374ms/step - loss: 0.9294 - accuracy: 0.7219 -
val_loss: 0.8537 - val_accuracy: 0.7700
Epoch 60/100
197/197 [==============================] - 76s 385ms/step - loss: 0.9108 - accuracy: 0.7263 -
val_loss: 0.8941 - val_accuracy: 0.7800
Epoch 61/100
197/197 [==============================] - 74s 375ms/step - loss: 0.9126 - accuracy: 0.7239 -
val_loss: 0.7812 - val_accuracy: 0.7780
Epoch 62/100
197/197 [==============================] - 74s 375ms/step - loss: 0.9078 - accuracy: 0.7265 -
val_loss: 0.7032 - val_accuracy: 0.7880
Epoch 63/100
197/197 [==============================] - 75s 380ms/step - loss: 0.9273 - accuracy: 0.7220 -
val_loss: 0.7186 - val_accuracy: 0.8000
Epoch 64/100
197/197 [==============================] - 74s 375ms/step - loss: 0.8849 - accuracy: 0.7350 -
val_loss: 0.7603 - val_accuracy: 0.8060
Epoch 65/100
197/197 [==============================] - 74s 375ms/step - loss: 0.8809 - accuracy: 0.7377 -
val_loss: 0.8706 - val_accuracy: 0.7640
Epoch 66/100
197/197 [==============================] - 74s 375ms/step - loss: 0.8930 - accuracy: 0.7299 -
val_loss: 0.6996 - val_accuracy: 0.7960
```

```
Epoch 67/100
197/197 [==============================] - 74s 376ms/step - loss: 0.8868 - accuracy: 0.7377 -
val_loss: 0.9174 - val_accuracy: 0.7500
Epoch 68/100
197/197 [==============================] - 74s 377ms/step - loss: 0.8766 - accuracy: 0.7407 -
val_loss: 0.7171 - val_accuracy: 0.8160
Epoch 69/100
197/197 [==============================] - 74s 376ms/step - loss: 0.8684 - accuracy: 0.7410 -
val_loss: 0.6977 - val_accuracy: 0.7980
Epoch 70/100
197/197 [==============================] - 75s 377ms/step - loss: 0.8667 - accuracy: 0.7385 -
val_loss: 0.7969 - val_accuracy: 0.7920
Epoch 71/100
197/197 [==============================] - 74s 375ms/step - loss: 0.8644 - accuracy: 0.7454 -
val_loss: 0.7055 - val_accuracy: 0.8180
Epoch 72/100
197/197 [==============================] - 73s 371ms/step - loss: 0.8665 - accuracy: 0.7384 -
val_loss: 0.7833 - val_accuracy: 0.7900
Epoch 73/100
197/197 [==============================] - 73s 371ms/step - loss: 0.8436 - accuracy: 0.7439 -
val_loss: 0.7330 - val_accuracy: 0.7860
Epoch 74/100
197/197 [==============================] - 73s 370ms/step - loss: 0.8558 - accuracy: 0.7423 -
val_loss: 0.7892 - val_accuracy: 0.7980
Epoch 75/100
197/197 [==============================] - 74s 375ms/step - loss: 0.8283 - accuracy: 0.7515 -
val_loss: 0.6844 - val_accuracy: 0.8120
Epoch 76/100
197/197 [==============================] - 73s 370ms/step - loss: 0.8390 - accuracy: 0.7450 -
val_loss: 0.7616 - val_accuracy: 0.8060
Epoch 77/100
197/197 [==============================] - 73s 369ms/step - loss: 0.8360 - accuracy: 0.7482 -
val_loss: 0.7068 - val_accuracy: 0.8180
Epoch 78/100
197/197 [==============================] - 74s 372ms/step - loss: 0.8280 - accuracy: 0.7529 -
val_loss: 0.6521 - val_accuracy: 0.8220
Epoch 79/100
197/197 [==============================] - 73s 370ms/step - loss: 0.8300 - accuracy: 0.7484 -
val_loss: 0.7747 - val_accuracy: 0.7780
Epoch 80/100
197/197 [==============================] - 73s 371ms/step - loss: 0.8115 - accuracy: 0.7567 -
val_loss: 0.7549 - val_accuracy: 0.8040
Epoch 81/100
197/197 [==============================] - 73s 371ms/step - loss: 0.8101 - accuracy: 0.7536 -
val_loss: 0.7237 - val_accuracy: 0.8000
Epoch 82/100
197/197 [==============================] - 73s 370ms/step - loss: 0.8133 - accuracy: 0.7590 -
val_loss: 0.6943 - val_accuracy: 0.7920
Epoch 83/100
197/197 [==============================] - 73s 371ms/step - loss: 0.8121 - accuracy: 0.7571 -
val_loss: 0.7312 - val_accuracy: 0.8100
Epoch 84/100
197/197 [==============================] - 73s 370ms/step - loss: 0.7962 - accuracy: 0.7651 -
val_loss: 0.7320 - val_accuracy: 0.8080
Epoch 85/100
197/197 [==============================] - 73s 371ms/step - loss: 0.7964 - accuracy: 0.7621 -
val_loss: 0.6519 - val_accuracy: 0.8220
Epoch 86/100
197/197 [==============================] - 73s 371ms/step - loss: 0.8069 - accuracy: 0.7565 -
val_loss: 0.8087 - val_accuracy: 0.7760
Epoch 87/100
197/197 [==============================] - 73s 369ms/step - loss: 0.7795 - accuracy: 0.7639 -
val_loss: 0.6450 - val_accuracy: 0.8260
Epoch 88/100
197/197 [==============================] - 73s 371ms/step - loss: 0.7867 - accuracy: 0.7645 -
val_loss: 0.6046 - val_accuracy: 0.8280
```

```
Epoch 89/100
197/197 [==============================] - 74s 376ms/step - loss: 0.7742 - accuracy: 0.7658 -
val_loss: 0.6271 - val_accuracy: 0.8320
Epoch 90/100
197/197 [==============================] - 73s 370ms/step - loss: 0.7711 - accuracy: 0.7718 -
val_loss: 0.7190 - val_accuracy: 0.8200
Epoch 91/100
197/197 [==============================] - 73s 370ms/step - loss: 0.7619 - accuracy: 0.7693 -
val_loss: 0.6059 - val_accuracy: 0.8400
Epoch 92/100
197/197 [==============================] - 73s 371ms/step - loss: 0.7651 - accuracy: 0.7712 -
val_loss: 0.7216 - val_accuracy: 0.8000
Epoch 93/100
197/197 [==============================] - 73s 371ms/step - loss: 0.7654 - accuracy: 0.7693 -
val_loss: 0.7469 - val_accuracy: 0.7980
Epoch 94/100
197/197 [==============================] - 73s 371ms/step - loss: 0.7732 - accuracy: 0.7684 -
val_loss: 0.6609 - val_accuracy: 0.8180
Epoch 95/100
197/197 [==============================] - 73s 370ms/step - loss: 0.7839 - accuracy: 0.7639 -
val_loss: 0.6655 - val_accuracy: 0.8280
Epoch 96/100
197/197 [==============================] - 73s 371ms/step - loss: 0.7437 - accuracy: 0.7751 -
val_loss: 0.7698 - val_accuracy: 0.7920
Epoch 97/100
197/197 [==============================] - 73s 369ms/step - loss: 0.7424 - accuracy: 0.7775 -
val_loss: 0.5671 - val_accuracy: 0.8420
Epoch 98/100
197/197 [==============================] - 73s 370ms/step - loss: 0.7698 - accuracy: 0.7712 -
val_loss: 0.7271 - val_accuracy: 0.8000
Epoch 99/100
197/197 [==============================] - 73s 371ms/step - loss: 0.7367 - accuracy: 0.7804 -
val_loss: 0.8172 - val_accuracy: 0.7900
Epoch 100/100
197/197 [==============================] - 73s 371ms/step - loss: 0.7521 - accuracy: 0.7766 -
val_loss: 0.6892 - val_accuracy: 0.7980
```

In [187…

```
# Cuvanje keras modela
# model.save('model.keras')

# Ucitavanje keras modela
model = load_model('model.keras')

# Cuvanje istorije obucavanja
# hist_df = pd.DataFrame(history.history)
# hist_csv_file = 'history.csv'
# with open(hist_csv_file, mode='w') as f:
#     hist_df.to_csv(f)

# Ucitavanje istorije obucavanja
```

6. Rezultati obučavanja

Tačnost našeg modela nad test skupom je 81.6%. U nastavku slede prikazi grafika performansi tokom obučavanja, prikaz matrice konfuzije i prikaz primera dobro i loše klasifikovanih slika iz test skupa.

Napomena: Prilikom nalaženja matrice konfuzije, morali smo da je grupišemo, jer je originalna matrica dimenzija 100x100. Spajanjem elemenata smo je smanjili na dimenzije 20x20.

In [188…

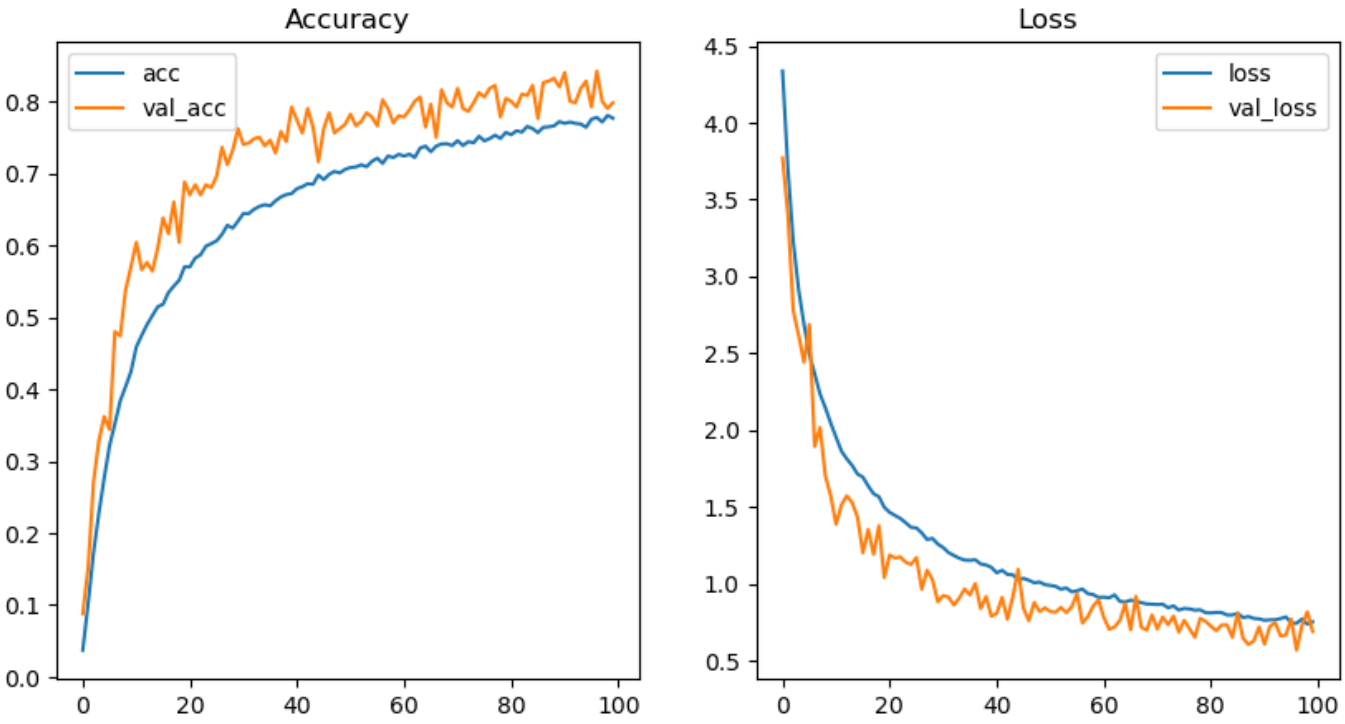
```
labels = np.array([])
pred = np.array([])
for img, lab in test_dataset:
    labels = np.append(labels, lab)
```

```
    pred = np.append(pred, np.argmax(model.predict(img, verbose=0), axis=1))

print('Tačnost modela je: ' + str(100*accuracy_score(labels, pred)) + '%')
```

```
Tačnost modela je: 81.6%
```

In [190…

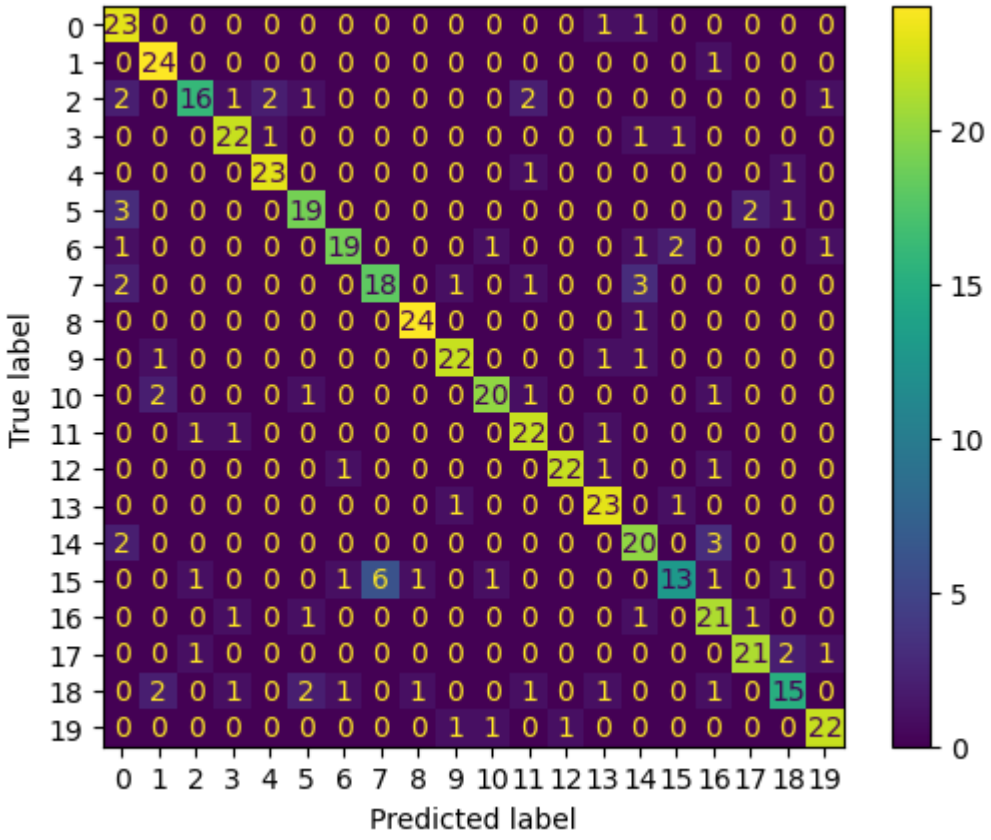
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

# Prikaz grafika
plt.figure(figsize=(10,5))
plt.subplot(121)
plt.plot(acc, label='acc')
plt.plot(val_acc, label='val_acc')
plt.title('Accuracy')
plt.legend()
plt.subplot(122)
plt.plot(loss, label='loss')
plt.plot(val_loss, label='val_loss')
plt.title('Loss')
plt.legend()
plt.show()
```



In [189…

```
# Konfuziona matrica 100x100
cm = confusion_matrix(labels, pred)

def aggregate_confusion_matrix(cm, aggregation_size):
    num_classes = cm.shape[0]
    aggregated_size = num_classes // aggregation_size
    aggregated_cm = np.zeros((aggregated_size, aggregated_size))

    for i in range(aggregated_size):
        for j in range(aggregated_size):
            aggregated_cm[i, j] = np.sum(
                cm[i * aggregation_size : (i + 1) * aggregation_size, j * aggregation_size :
            )

    return aggregated_cm

# Faktor skupljanja konfuzione matrice
```

```
aggregation_size = 5
aggregated_cm = aggregate_confusion_matrix(cm, aggregation_size)

# Prikaz konfuzione matrice
disp = ConfusionMatrixDisplay(confusion_matrix=aggregated_cm)
disp.plot();
```



In [191…

```
# Liste slika iz test skupa i njihovih labela
test_images = []
test_labels = []

for images_batch, labels_batch in test_dataset:
    for image, label in zip(images_batch, labels_batch):
        # Dodavanje slike u listu
        if image.shape[0] == 90 and image.shape[1] == 90 and image.shape[2] == 3:
            test_images.append(image)
            test_labels.append(label.numpy())

test_images = np.array(test_images)
test_labels = np.array(test_labels)

# Predikcija
predictions = model.predict(test_images)
predicted_labels = np.argmax(predictions, axis=1)

# Dobro klasifikovani leptiri
correct_indices = np.where(predicted_labels == test_labels)[0]

# Lose klasifikovani leptiri
incorrect_indices = np.where(predicted_labels != test_labels)[0]

# Ovo je potrebno da bi slike bile odgovarajuceg formata za prikaz
for i in range(test_images.shape[0]):
    image = test_images[i]
    image = np.asarray(image)
    image = image.astype(np.float32) / 255.0
    test_images[i] = image

# Prikaz 10 nasumicnih dobro klasifikovanih slika iz test skupa
```

```
random_indices = np.random.choice(correct_indices, size=10, replace=False)
plt.figure(figsize=(15, 7))
plt.suptitle('Dobro klasifikovane slike', fontsize=16)
for i, idx in enumerate(random_indices):
    plt.subplot(2, 5, i + 1)
    plt.imshow(test_images[idx])
    plt.title(f"Klasifikovano: {predicted_labels[idx]},\n Pripada: {test_labels[idx]}")
    plt.axis('off')
plt.show()

# Prikaz 10 nasumicnih lose klasifikovanih slika iz test skupa
random_indices = np.random.choice(incorrect_indices, size=10, replace=False)
plt.figure(figsize=(15, 7))
plt.suptitle('Lose klasifikovane slike', fontsize=16)
for i, idx in enumerate(random_indices):
    plt.subplot(2, 5, i + 1)
    plt.imshow(test_images[idx])
    plt.title(f"Klasifikovano: {predicted_labels[idx]},\n Pripada: {test_labels[idx]}")
    plt.axis('off')
plt.show()
```

```
16/16 [==============================] - 1s 41ms/step
```