Kompjuterski Vid 2024.

Klasifikacija slika korišćenjem CNN arhitekture

Sadržaj

Student:

Stefan Aleksandrić 43/2016

Pregled projekta

Ovaj projekat se fokusira na izgradnju modela za klasifikaciju slika koristeći NASNetMobile arhitekturu za klasifikaciju cveća. Ključni elementi uključuju:

- **Priprema Podataka:** Učitavanje dataset-a sa Google Drive-a uz primenu tehnika augmentacije slika poput rotacije, zumiranja i okretanja radi povećanja robusnosti modela.
- **Link kada Flowers datasetu:**
    - https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
- **Arhitektura Modela:** Korišćenje NASNetMobile kao baznog modela sa dodatnim konvolucionim slojevima za poboljšano izdvajanje karakteristika. Gornji slojevi baznog modela su isključeni, a primenjene su i skip konekcije za bolji protok informacija kroz slojeve mreže.
- **Transfer Learning:** Zamrzavanje svih slojeva baznog modela osim poslednjih 20 kako bi se sačuvale pretrenirane težine, uz omogućeno fino podešavanje na specifičnom dataset-u cveća.
- **Strategija Treniranja:** Model je kompajliran koristeći SGD optimizator sa momentom, a implementirano je rano zaustavljanje kako bi se sprečio overfitting i optimizovalo korišćenje resursa.
- **Evaluacija:** Model je treniran i evaluiran na validacionom setu, postigavši zadovoljavajući nivo tačnosti u klasifikaciji slika cveća.

Link ka Github repozitorijumu:

https://github.com/stefan-aleksandric/Keras-AiImageClassificationModel

Ovaj dokument pruža kratak opis metodologija, koraka implementacije i rezultata postignutih u projektu tj. biće prikazane razne iteracije pokušaja i rezultata treniranja modela.

1. Iteracija

U prvoj iteraciji korišćen je predefinisan batch_size veličine 32 i slikama je podešena rezolucija na 224x224 piksela. Za normalizaciju podataka i agumentaciju podataka koristi se biblioteka ImageDataGenerator koja omogućava laku manipulaciju nad podacima.

Podaci su reskalirani na vrednosti od 0 do 1 za sve tri boje. U početku su imali vrednosti od 0 do 255 za RGB svakog piksela redosledom. Ovim se postiže stabilnije treniranje modela.

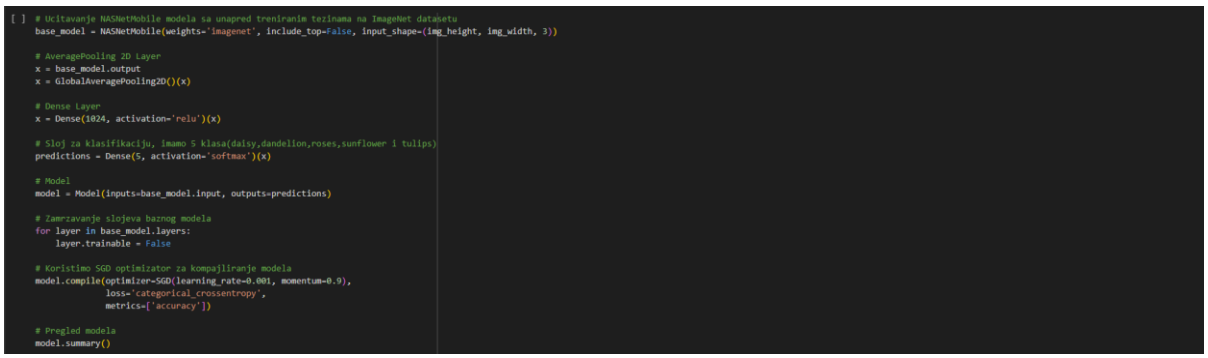
Za agumentaciju nasumično su slike okretane od -20 do 20 stepeni, kao i uveličavane nasumično +- 20%. Takodje su i horizontalno okretane nasumično.

Podaci su podeljeni 80-20 tj. trening set će iznositi nekih 80% svih podataka, dok će validacioni ili testni set iznositi 20% podataka iz dataseta. Kada se pokrene ovaj kod dobijemo rezultat da je 2939 slika u po 5 kategorija smešteno u trening set, dok u validacionom setu imamo 731 sliku u po 5 kategorija.

```
[ ] # Putanja do dataset-a na Google Drive-u
    dataset_path = '/content/drive/MyDrive/Colab Notebooks/Projekat2024/datasets/flower_photos'

    # Parametri za obuku
    batch_size = 32
    img_height, img_width = 224, 224

    # Kreiranje ImageDataGenerator-a za augmentaciju i normalizaciju
    train_datagen = ImageDataGenerator(
        rescale=1./255, # reskaliranje piksela na raspon od 0 do 1
        rotation_range=20, # nasumicno rotiranje slika od -20 do 20 stepeni
        zoom_range=0.2, # nasumicno zumiranje slika +- 20%
        horizontal_flip=True, # nasumicno horizontalno okretanje slika
        validation_split=0.2  # Podela za validaciju tj. 80% za obuku i 20% za validaciju
    )

    # Set za trening
    training_set = train_datagen.flow_from_directory(
        dataset_path,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical',
        subset='training'
    )

    # Set za validaciju
    validation_set = train_datagen.flow_from_directory(
        dataset_path,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical',
        subset='validation'
    )
```

Ovo je prvi bazični model koji je proban. Isključeni su top layeri ovog modela dok su ostali layeri zamrznuti za treniranje. Na ovo je dodat jedan GlobalAveragePooling2D, pa Dense Layer. Pošto imamo 5 klasa dodelili smo još jedan Dense layer u kom su razvrstani podaci po odgovarajućim klasama. Korišćene su relu i softmax funkcije, konkretno relu za potpuno povezane slojeve dok se softmax koristi za poslednje slojeve gde se predvidjaju klase i konvertuju se izlazi u verovatnoće koje zajedno čine 1.

Model se zatim kompajluje sa SGD optimizatorom i u ovoj iteraciji su korišćeni learning_rate od 0.001 i momentum od 0.9 . Koristi se isto categorical_crossentropy i accuracy metrika tokom treniranja.



```
[ ] # Ucitavanje NASNetMobile modela sa unapred treniranim tezinama na ImageNet datasetu
    base_model = NASNetMobile(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))

    # AveragePooling 2D Layer
    x = base_model.output
    x = GlobalAveragePooling2D()(x)

    # Dense Layer
    x = Dense(1024, activation='relu')(x)

    # Sloj za klasifikaciju, imamo 5 klasa(daisy,dandelion,roses,sunflower i tulips)
    predictions = Dense(5, activation='softmax')(x)

    # Model
    model = Model(inputs=base_model.input, outputs=predictions)

    # Zamrzavanje slojeva baznog modela
    for layer in base_model.layers:
        layer.trainable = False

    # Koristimo SGD optimizator za kompajliranje modela
    model.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    # Pregled modela
    model.summary()
```

Ovde možemo videti broj parametara i brojeve filtera koje smo dodali, kao i konačni broj parametara u Dense layeru razvrstanih po klasama.

activation_375 (Activation)	(None, 7, 7, 1056)	0	normal_concat_12[0][0]
global_average_pooling2d… (GlobalAveragePooling2D)	(None, 1056)	0	activation_375[0][0]
dense_2 (Dense)	(None, 1024)	1,082,368	global_average_poolin…
dense_3 (Dense)	(None, 5)	5,125	dense_2[0][0]

```
Total params: 5,357,209 (20.44 MB)
Trainable params: 1,087,493 (4.15 MB)
Non-trainable params: 4,269,716 (16.29 MB)
```

Prva iteracija je radjena u 10 epoha. Izračunati su koraci po epohama tj. ukupan broj slika iz trening seta podeljen sa batch_size-om (2939/32). Isto se primenjuje I na validacionom skupu podataka. Moglo se odmah zaključiti da je osnovni model sa ovim malim izmenama zadovoljavajuće istrenirao. Ako se tačnost na trening setu povećava dok se tačnost na validacionom setu smanjuje, to obično znači da je model pretreniran. Model dobro funkcioniše na podacima na kojima je treniran, ali loše generalizuje na nove podatke. Ovo može biti znak da je potrebno dodati regularizaciju, prilagoditi hiperparametre ili koristiti više podataka za obuku.

```
[ ]  # Broj epoha
     epochs = 10

     # Model se trenira na training_set-u i validira na validation_set-u
     history = model.fit(
         training_set,
         steps_per_epoch=training_set.samples // training_set.batch_size,
         validation_data=validation_set,
         validation_steps=validation_set.samples // validation_set.batch_size,
         epochs=epochs
     )
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDatas
  self._warn_if_super_not_called()
91/91 ━━━━━━━━━━━━━━━━━━━━ 1298s 13s/step - accuracy: 0.4503 - loss: 1.3363 - val_accuracy: 0.7628 - val_loss: 0.7286
Epoch 2/10
 1/91 ━━━━━━━━━━━━━━━━━━━━ 6s 75ms/step - accuracy: 0.8438 - loss: 0.6392/usr/lib/python3.10/contextlib.py:153: UserWarning: Y
  self.gen.throw(typ, value, traceback)
91/91 ━━━━━━━━━━━━━━━━━━━━ 16s 175ms/step - accuracy: 0.8438 - loss: 0.6392 - val_accuracy: 0.6296 - val_loss: 0.8989
Epoch 3/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 105s 607ms/step - accuracy: 0.7535 - loss: 0.7009 - val_accuracy: 0.7727 - val_loss: 0.6193
Epoch 4/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 10s 109ms/step - accuracy: 0.8438 - loss: 0.5463 - val_accuracy: 0.8148 - val_loss: 0.5480
Epoch 5/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 61s 634ms/step - accuracy: 0.8018 - loss: 0.5884 - val_accuracy: 0.8125 - val_loss: 0.5621
Epoch 6/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7812 - loss: 0.4936 - val_accuracy: 0.7778 - val_loss: 0.6172
Epoch 7/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 80s 615ms/step - accuracy: 0.8177 - loss: 0.5206 - val_accuracy: 0.8011 - val_loss: 0.5403
Epoch 8/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.8125 - loss: 0.5837 - val_accuracy: 0.8519 - val_loss: 0.3996
Epoch 9/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 80s 603ms/step - accuracy: 0.8441 - loss: 0.4679 - val_accuracy: 0.8082 - val_loss: 0.5149
Epoch 10/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 10s 106ms/step - accuracy: 0.9375 - loss: 0.3854 - val_accuracy: 0.7407 - val_loss: 0.6986
```

Na grafiku možemo videti kako se naš model ponašao tokom treninga.


```
val_loss, val_accuracy = model.evaluate(validation_set, steps=validation_set.samples // validation_set.batch_size)
print(f"Gubitak: {val_loss:.4f}")
print(f"Tacnost: {val_accuracy:.4f}")
```

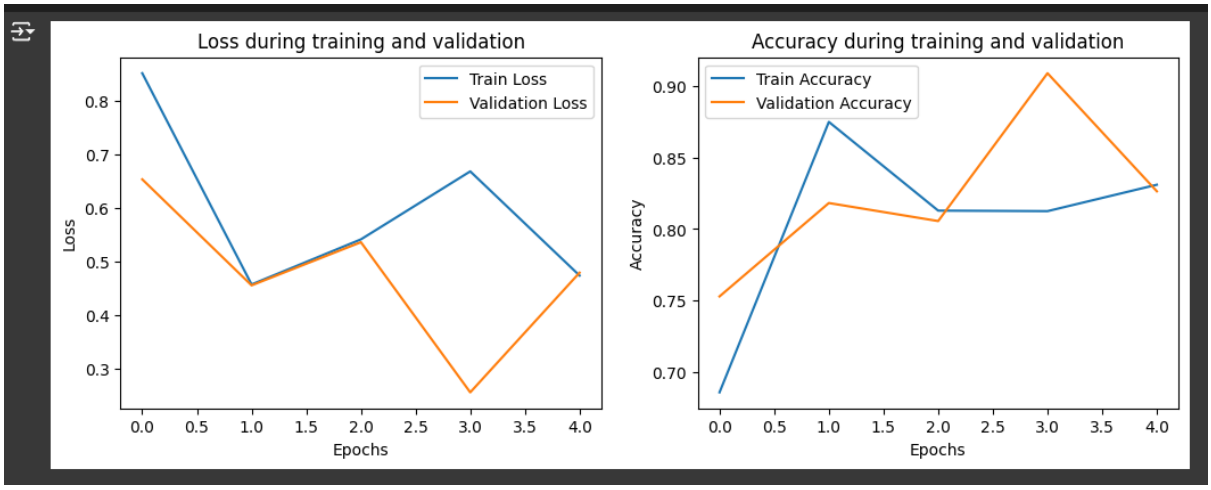
```
22/22 ━━━━━━━━━━━━━━━━━━━━ 12s 536ms/step - accuracy: 0.8086 - loss: 0.5482
Validacioni gubitak: 0.5306
Validaciona tačnost: 0.8139
```

Zatim model je proban sa 5 epoha i batch_size je postavljen na 16. Vidimo da je accuracy trening seta naglo porastao pa opao dok se accuracy na validacionom skupu bolje ponašao, konkretno accuracy u 3. epohi za trening je bio 0.87 dok za trening skup je iznosio 0.81.

Zatim vidimo da je accuracy za trening set u 4. epohi bio 0.81 dok je na validacionom skupu iznosio 0.9091. Što je zadovoljavajući rezultat. Može se zaključiti da je bazni model već dobro istreniran i da su potrebne male izmene kako bi se model jos bolje prilagodio i bio stabilniji tokom učenja nad ovim datasetom.

```
Epoch 1/5
183/183 ━━━━━━━━━━━━━━━━━━━━ 787s 4s/step - accuracy: 0.5540 - loss: 1.1161 - val_accuracy: 0.7528 - val_loss: 0.6533
Epoch 2/5
  1/183 ━━━━━━━━━━━━━━━━━━━━ 2:57 978ms/step - accuracy: 0.8750 - loss: 0.4570/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; interrupting training. Make sure tha
  self.gen.throw(typ, value, traceback)
183/183 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8750 - loss: 0.4570 - val_accuracy: 0.8182 - val_loss: 0.4553
Epoch 3/5
183/183 ━━━━━━━━━━━━━━━━━━━━ 365s 2s/step - accuracy: 0.8093 - loss: 0.5524 - val_accuracy: 0.8056 - val_loss: 0.5359
Epoch 4/5
183/183 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8125 - loss: 0.6681 - val_accuracy: 0.9091 - val_loss: 0.2554
Epoch 5/5
183/183 ━━━━━━━━━━━━━━━━━━━━ 327s 2s/step - accuracy: 0.8138 - loss: 0.4982 - val_accuracy: 0.8264 - val_loss: 0.4790
```

Grafik sa novim batch_size-om i brojom epoha 5.



Uradjen je klasifikacioni report sa potrebnim metrikama I dodata je konfuziona matrica.

```
46/46 ━━━━━━━━━━━━━━━━━━━━ 66s 1s/step - accuracy: 0.8517 - loss: 0.4451
46/46 ━━━━━━━━━━━━━━━━━━━━ 89s 2s/step
Test Loss: 0.4800
Test Accuracy: 0.8304
Classification Report:
              precision    recall  f1-score   support

       daisy       0.18      0.17      0.17       126
   dandelion       0.24      0.24      0.24       179
       roses       0.20      0.18      0.19       128
  sunflowers       0.19      0.21      0.20       139
      tulips       0.22      0.23      0.23       159

    accuracy                           0.21       731
   macro avg       0.21      0.21      0.21       731
weighted avg       0.21      0.21      0.21       731

Confusion Matrix:
[[21 33 11 29 32]
 [25 43 34 39 38]
 [23 27 23 26 29]
 [18 37 25 29 30]
 [31 41 22 28 37]]
```

Primeri gde je model pogodio i grešio.

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step
Model predvidja da je ova slika: roses
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 103ms/step
Model predvidja da je ova slika: sunflower
```

```
Model predvidja da je ova slika: sunflower
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
Model predvidja da je ova slika: tulips
```

Dodat je kod u projekat pomoću kog se može iskoristi bilo koja slika sa interneta da se istestira predikcija modela. Potrebno je samo dati dobar url do željene slike.

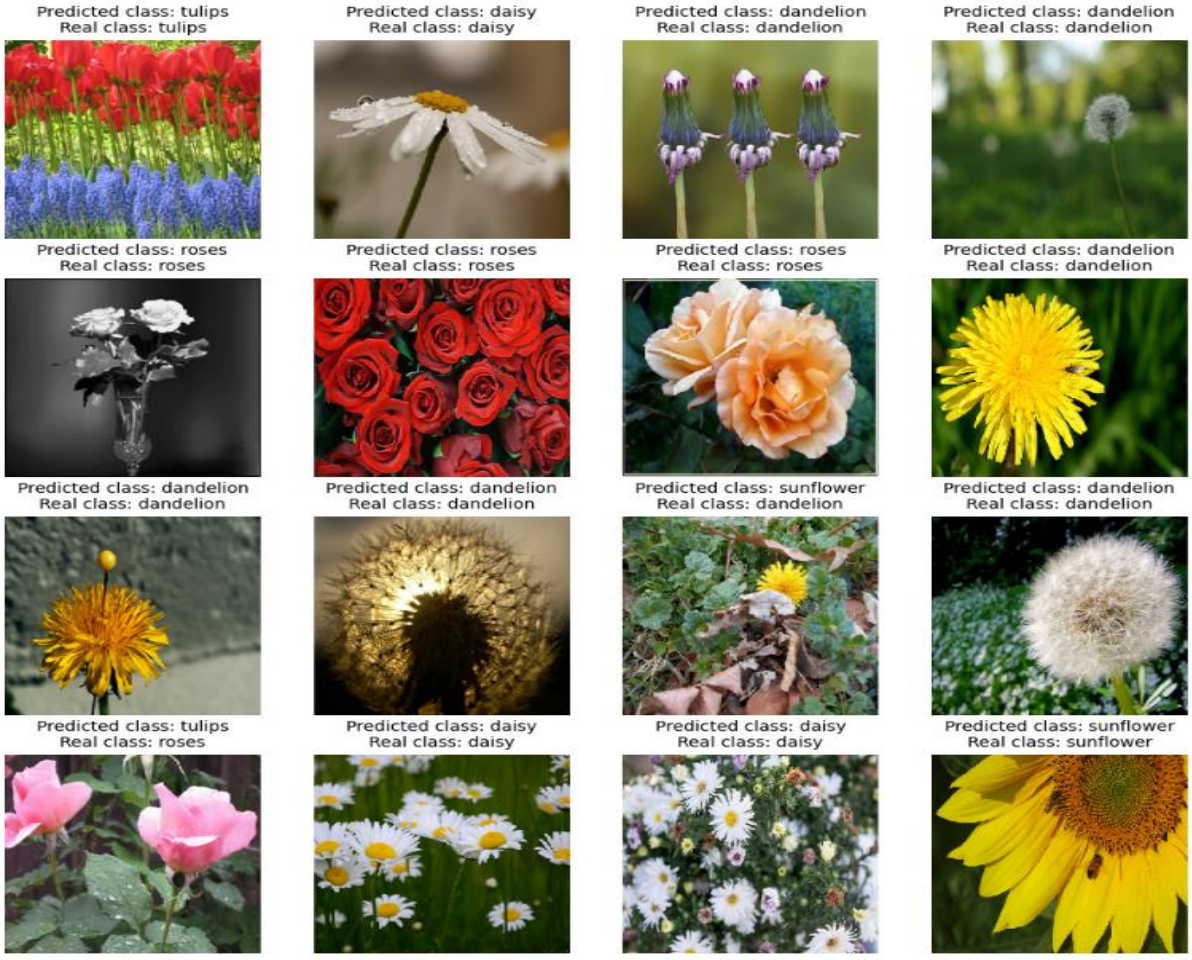
```
import requests
from PIL import Image
from io import BytesIO
def load_image_from_url(url, target_size=(224, 224)):
    response = requests.get(url)
    img = Image.open(BytesIO(response.content))
    img = img.resize(target_size)
    img_array = np.array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array / 255.0  # Normalizacija piksela
    return img_array

# URL slike
image_url = 'https://cdn.britannica.com/36/82536-050-7E968918/Shasta-daisies.jpg'

# Ucitavanj slike
img_array = load_image_from_url(image_url)

# Prikaz slike
plt.imshow(img_array[0])
plt.axis('off')
plt.show()

# Napravi predikciju
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions, axis=1)
class_names = ['daisy', 'dandelion', 'roses', 'sunflower', 'tulips']

print(f"Model predvidja da je ova slika: {class_names[predicted_class[0]]}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 245ms/step
Model predvidja da je ova slika: daisy
```

Zatim model je proban na random batch-evima od 16 slika iz dataset-a.

```python
# Parametri za ucitavanje podataka
batch_size = 16
img_height, img_width = 224, 224

# Kreiranje ImageDataGenerator-a samo za validaciju bez augmentacije
datagen = ImageDataGenerator(rescale=1./255)

# Ucitavanje podataka iz dataset-a
data_generator = datagen.flow_from_directory(
    dataset_path,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True  # Mesanje slika za prikaz
)

# Ucitavanje jedne grupe slika (jedan batch)
images, labels = next(data_generator)

# Predikcija klasa za slike u batch-u
predictions = model.predict(images)
predicted_classes = np.argmax(predictions, axis=1)

# Imena klasa
class_names = ['daisy', 'dandelion', 'roses', 'sunflower', 'tulips']

plt.figure(figsize=(12, 12))
for i in range(batch_size):
    plt.subplot(4, 4, i+1)
    plt.imshow(images[i])
    plt.title(f"Predicted class: {class_names[predicted_classes[i]]}\nReal class: {class_names[np.argmax(labels[i])]}",fontsize=10)
    plt.axis('off')
plt.show()
```

Na ovoj slici možemo videti rezultate na tih 16 slika.



Na dalje ova prva iteracija i kod će se koristiti i nadogradjivati kako bi se isprobale razne tehnike i različiti parametri. Ostatak projekta se nadogradjuje na prvu iteraciju.

2.Iteracija

U drugoj iteraciji model je izmenjen I dodati su konvolucioni slojevi od 512 i 256 filtera kao i MaxPooling2D.

```
# Ucitavanje NASNetMobile modela sa unapred treniranim tezinama na ImageNet datasetu
base_model = NASNetMobile(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))

# AveragePooling 2D Layer
x = base_model.output
x = Conv2D(512, (3, 3), activation='relu', padding='same')(x) # Konvolucioni sloj 512
x = Conv2D(256, (3, 3), activation='relu', padding='same')(x) # Konvolucioni sloj 256
x = MaxPooling2D((2, 2))(x) # 2D Pooling layer
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dense(5, activation='softmax')(x)

# Model
model = Model(inputs=base_model.input, outputs=x)

# Zamrzavanje slojeva baznog modela
for layer in base_model.layers:
    layer.trainable = False

# Koristimo SGD optimizator za kompajliranje modela
model.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Pregled modela
model.summary()
```

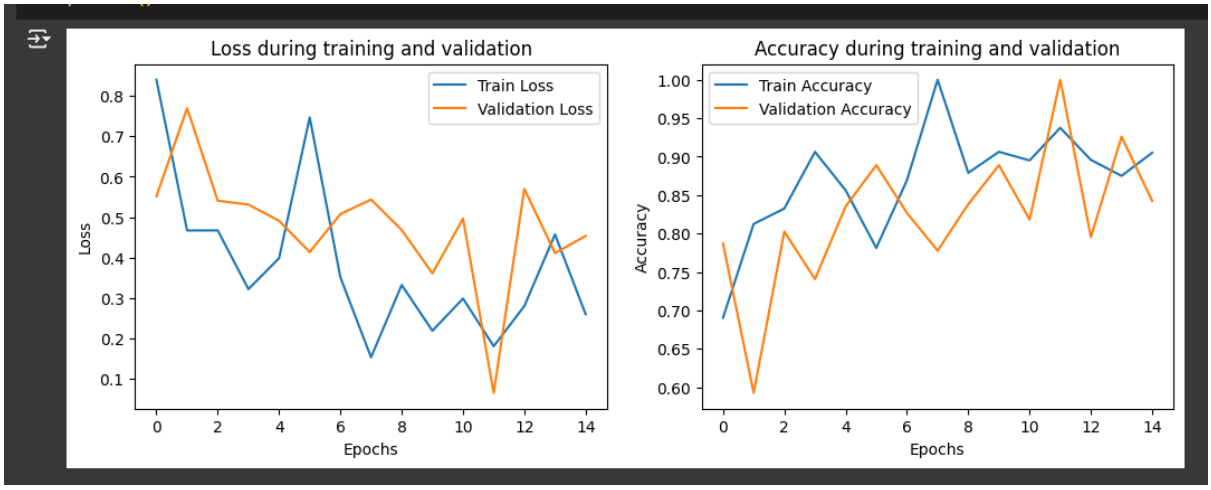
activation_1315 (Activation)	(None, 7, 7, 1056)	0	normal_concat_12[0][0]
conv2d_1 (Conv2D)	(None, 7, 7, 512)	4,866,560	activation_1315[0][0]
conv2d_2 (Conv2D)	(None, 7, 7, 256)	1,179,904	conv2d_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 3, 3, 256)	0	conv2d_2[0][0]
global_average_pooling2d… (GlobalAveragePooling2D)	(None, 256)	0	max_pooling2d_1[0][0]
dense_8 (Dense)	(None, 1024)	263,168	global_average_poolin…
dense_9 (Dense)	(None, 5)	5,125	dense_8[0][0]

```
Total params: 10,584,473 (40.38 MB)
Trainable params: 6,314,757 (24.09 MB)
Non-trainable params: 4,269,716 (16.29 MB)
```

Koristio se batch_size od 32 opet ali ovog puta treniranje je bilo u 15 epoha. U početku, model se možda pretrenirava, što uzrokuje visoku tačnost na trening setu i nisku na validacionom setu. Kasnije, poboljšanje na validacionom setu ukazuje na bolje generalizovanje modela.

```
Epoch 1/15
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDa
  self._warn_if_super_not_called()
91/91 ━━━━━━━━━━━━━━━━━━━━ 456s 5s/step - accuracy: 0.5720 - loss: 1.1256 - val_accuracy: 0.7869 - val_loss: 0.5517
Epoch 2/15
 1/91 ━━━━━━━━━━━━━━━━━━━━ 4:12 3s/step - accuracy: 0.8125 - loss: 0.4670/usr/lib/python3.10/contextlib.py:153: UserWarning
  self.gen.throw(typ, value, traceback)
91/91 ━━━━━━━━━━━━━━━━━━━━ 5s 27ms/step - accuracy: 0.8125 - loss: 0.4670 - val_accuracy: 0.5926 - val_loss: 0.7695
Epoch 3/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 461s 5s/step - accuracy: 0.8254 - loss: 0.4835 - val_accuracy: 0.8026 - val_loss: 0.5405
Epoch 4/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - accuracy: 0.9062 - loss: 0.3219 - val_accuracy: 0.7407 - val_loss: 0.5311
Epoch 5/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 411s 4s/step - accuracy: 0.8621 - loss: 0.3780 - val_accuracy: 0.8352 - val_loss: 0.4908
Epoch 6/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - accuracy: 0.7812 - loss: 0.7466 - val_accuracy: 0.8889 - val_loss: 0.4136
Epoch 7/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 442s 5s/step - accuracy: 0.8614 - loss: 0.3675 - val_accuracy: 0.8267 - val_loss: 0.5076
Epoch 8/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - accuracy: 1.0000 - loss: 0.1532 - val_accuracy: 0.7778 - val_loss: 0.5435
Epoch 9/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 431s 4s/step - accuracy: 0.8870 - loss: 0.3171 - val_accuracy: 0.8381 - val_loss: 0.4683
Epoch 10/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 13s 114ms/step - accuracy: 0.9062 - loss: 0.2188 - val_accuracy: 0.8889 - val_loss: 0.3609
Epoch 11/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 427s 4s/step - accuracy: 0.8966 - loss: 0.2961 - val_accuracy: 0.8182 - val_loss: 0.4963
Epoch 12/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - accuracy: 0.9375 - loss: 0.1805 - val_accuracy: 1.0000 - val_loss: 0.0650
Epoch 13/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 438s 4s/step - accuracy: 0.9054 - loss: 0.2651 - val_accuracy: 0.7955 - val_loss: 0.5699
Epoch 14/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - accuracy: 0.8750 - loss: 0.4573 - val_accuracy: 0.9259 - val_loss: 0.4111
Epoch 15/15
91/91 ━━━━━━━━━━━━━━━━━━━━ 409s 4s/step - accuracy: 0.9070 - loss: 0.2509 - val_accuracy: 0.8423 - val_loss: 0.4536
```

Grafik 2. iteracije.

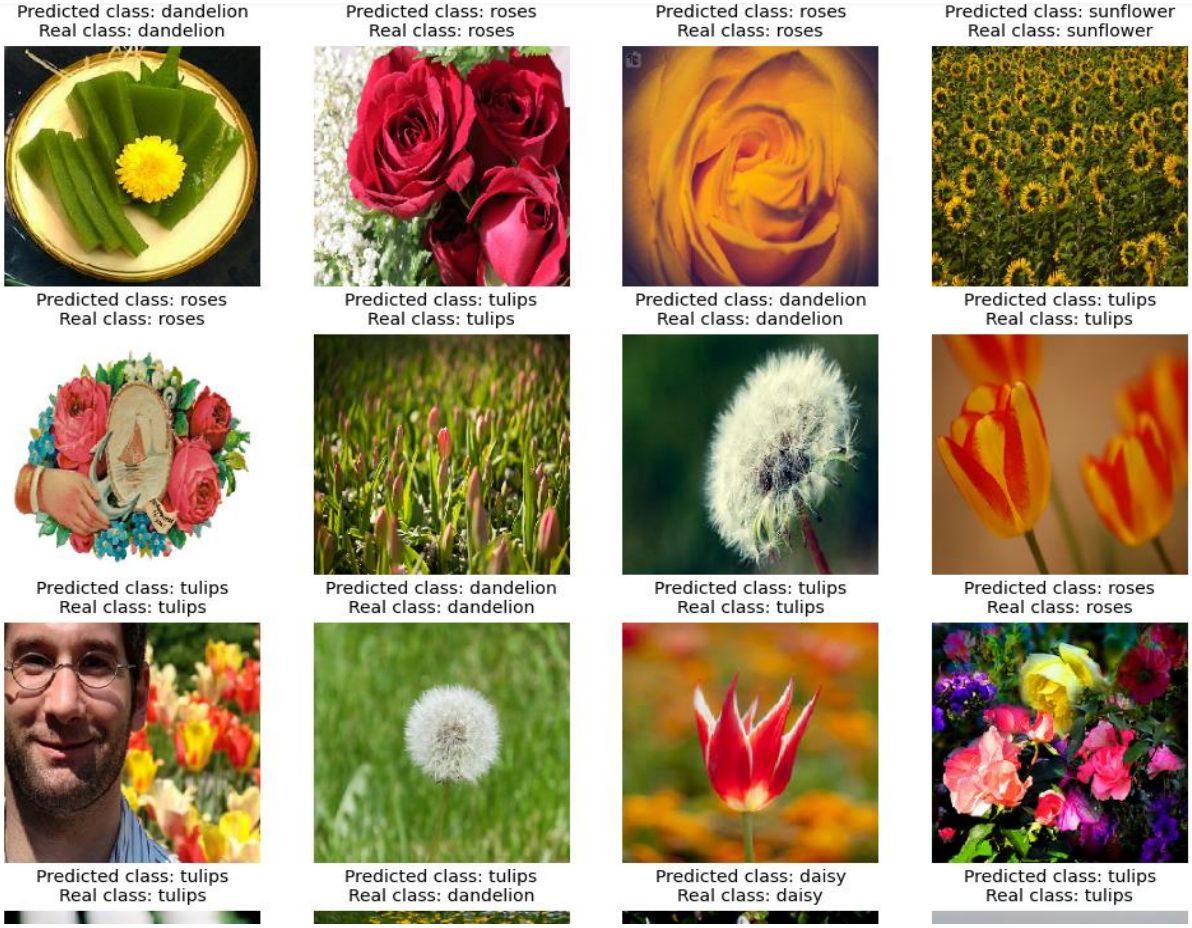


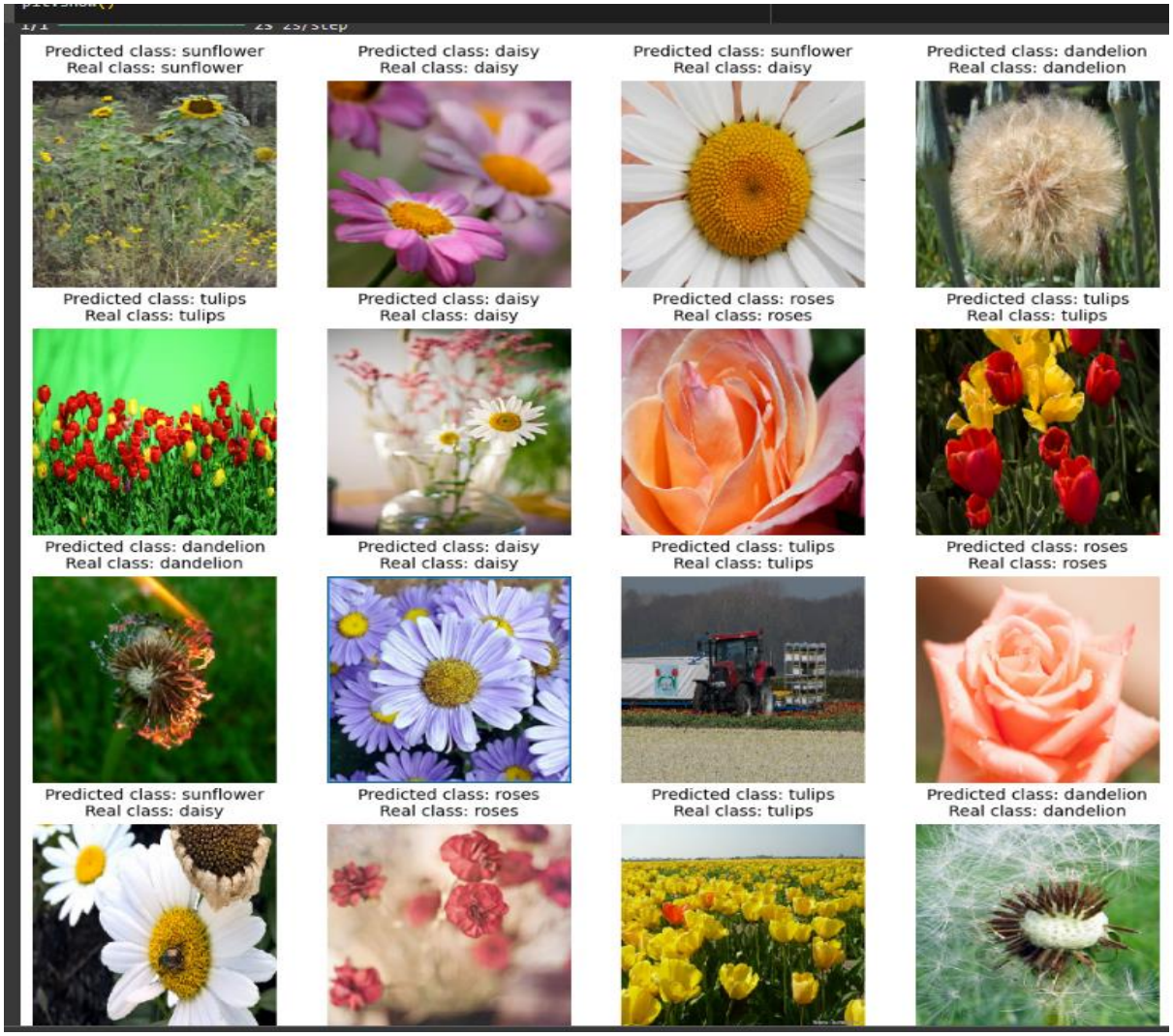
Parametri 2. Iteracije.

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 71s 3s/step - accuracy: 0.8485 - loss: 0.4318
23/23 ━━━━━━━━━━━━━━━━━━━━ 90s 3s/step
Test Loss: 0.4329
Test Accuracy: 0.8523
Klasifikacioni izvestaj:
              precision    recall  f1-score   support

       daisy       0.18      0.17      0.18       126
   dandelion       0.22      0.20      0.21       179
       roses       0.15      0.16      0.16       128
  sunflowers       0.20      0.22      0.21       139
      tulips       0.23      0.23      0.23       159

    accuracy                           0.20       731
   macro avg       0.20      0.20      0.20       731
weighted avg       0.20      0.20      0.20       731

Konfuziona matrica:
[[22 23 26 28 27]
 [24 36 38 37 44]
 [29 34 21 27 17]
 [21 33 22 30 33]
 [27 36 31 28 37]]
```

9. Isprobavanje random slika sa interneta (potrebno je zameniti image_url ako zelimo drugu sliku)

Isprobavanje modela na nasumičnim slikama u grupacijama od 16 slika iz dataset-a.





3.Iteracija

Model u trećoj iteraciji.

```
# Ucitavanje NASNetMobile modela sa unapred treniranim tezinama na ImageNet datasetu
base_model = NASNetMobile(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))

# AveragePooling 2D Layer
x = base_model.output
x = Conv2D(512, (3, 3), activation='relu', padding='same')(x) # Konvolucioni sloj 512 filtera
x = Conv2D(256, (3, 3), activation='relu', padding='same')(x) # Konvolucioni sloj 256 filtera
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x) # Konvolucioni sloj 128 filtera
x = MaxPooling2D((2, 2))(x) # 2D Pooling layer
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x) # Droput layer
x = Dense(1024, activation='relu')(x)
x = Dense(5, activation='softmax')(x)

# Model
model = Model(inputs=base_model.input, outputs=x)

# Zamrzavanje slojeva baznog modela
for layer in base_model.layers:
    layer.trainable = False

# Koristimo SGD optimizator za kompajliranje modela
model.compile(optimizer=SGD(learning_rate=0.001, momentum=0.9),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Pregled modela
model.summary()
```

Dodat je još jedan konvolucioni sloj sa manjim brojem filtera i jedan Droput layer kako bismo sprečili pretreniranje koje se pojavljivalo.

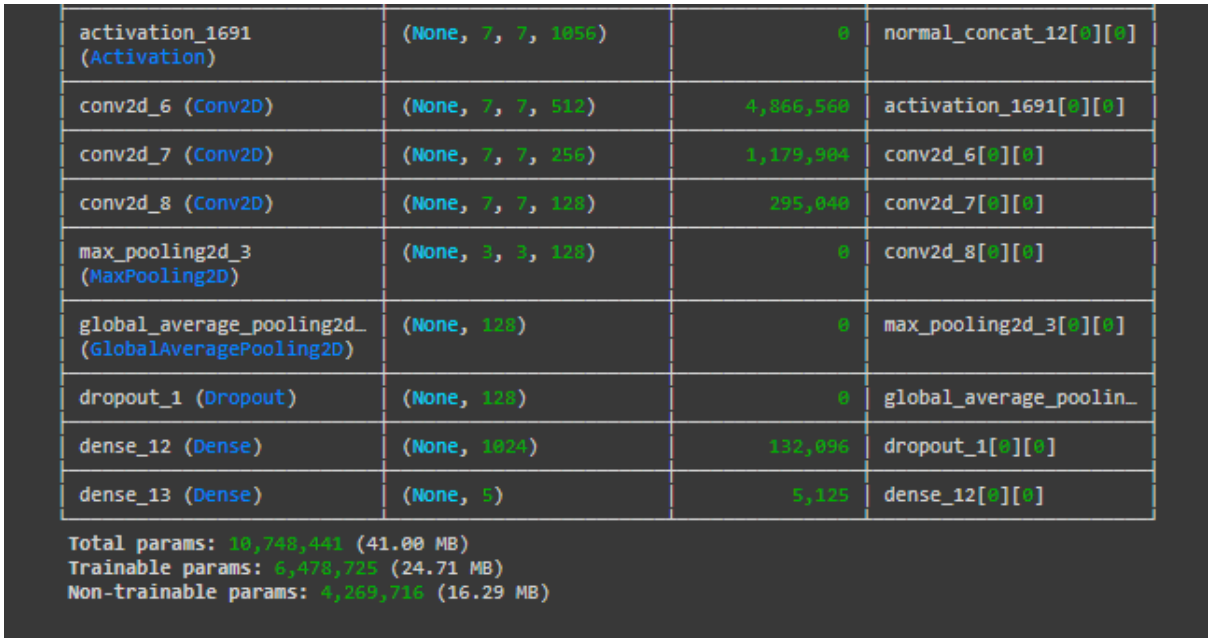
Learning rate i momentum su ostali isti za sad.

Sa tri konvoluciona 2D filtera postižemo malo specifičnije prepoznavanje karakteristika.

512 filtera prepoznaje fine detalje, 256 srednje složene karakteristike dok 128 filtera može da prepozna apstraktne karakteristike.

Pooling layer se koristi da bi se smanjio broj parametara što čini model bržim i takodje manje sklonim pretreniranju.

Dropout-om od 0.5 postižemo da se pola neurona nasumično postavi na nulu tj. da budu neaktivni u sloju pri svakoj iteraciji treniranja.



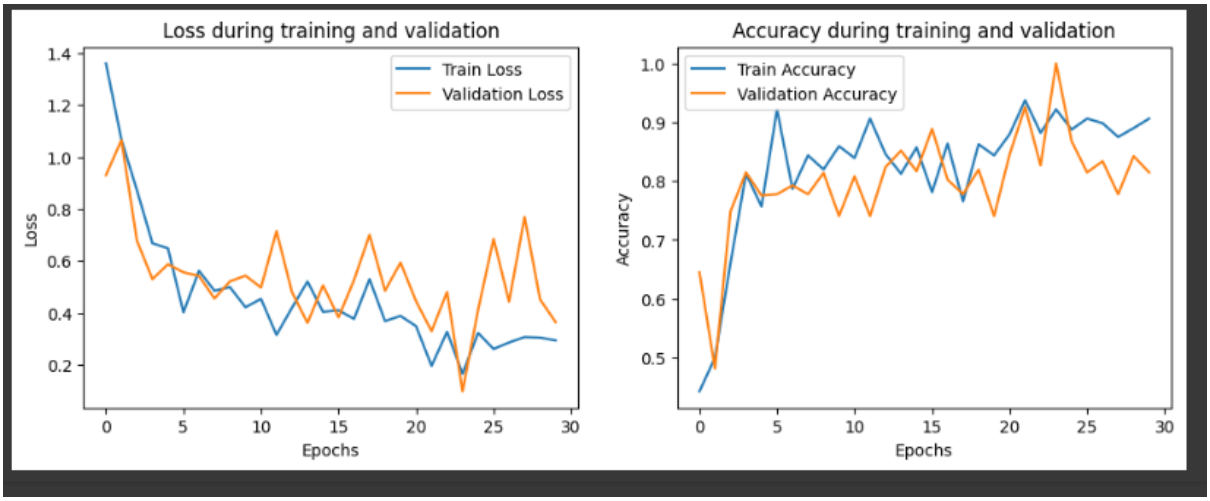
activation_1691 (Activation)	(None, 7, 7, 1056)	0	normal_concat_12[0][0]
conv2d_6 (Conv2D)	(None, 7, 7, 512)	4,866,560	activation_1691[0][0]
conv2d_7 (Conv2D)	(None, 7, 7, 256)	1,179,904	conv2d_6[0][0]
conv2d_8 (Conv2D)	(None, 7, 7, 128)	295,040	conv2d_7[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 128)	0	conv2d_8[0][0]
global_average_pooling2d_ (GlobalAveragePooling2D)	(None, 128)	0	max_pooling2d_3[0][0]
dropout_1 (Dropout)	(None, 128)	0	global_average_poolin_
dense_12 (Dense)	(None, 1024)	132,096	dropout_1[0][0]
dense_13 (Dense)	(None, 5)	5,125	dense_12[0][0]

Total params: 10,748,441 (41.00 MB)
Trainable params: 6,478,725 (24.71 MB)
Non-trainable params: 4,269,716 (16.29 MB)

U ovoj iteraciji koristio se veći batch od 64 i broj epoha je bio 30 tj. probano je nešto duže treniranje modela.

```
Epoch 6/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 64s 1s/step - accuracy: 0.9219 - loss: 0.4015 - val_accuracy: 0.7778 - val_loss: 0.5557
Epoch 7/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 436s 9s/step - accuracy: 0.7782 - loss: 0.5707 - val_accuracy: 0.7926 - val_loss: 0.5419
Epoch 8/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 8s 46ms/step - accuracy: 0.8438 - loss: 0.4848 - val_accuracy: 0.7778 - val_loss: 0.4547
Epoch 9/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 430s 9s/step - accuracy: 0.8200 - loss: 0.5094 - val_accuracy: 0.8139 - val_loss: 0.5210
Epoch 10/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 67s 1s/step - accuracy: 0.8594 - loss: 0.4210 - val_accuracy: 0.7407 - val_loss: 0.5428
Epoch 11/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 441s 9s/step - accuracy: 0.8396 - loss: 0.4494 - val_accuracy: 0.8082 - val_loss: 0.4971
Epoch 12/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 8s 46ms/step - accuracy: 0.9062 - loss: 0.3144 - val_accuracy: 0.7407 - val_loss: 0.7144
Epoch 13/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 433s 9s/step - accuracy: 0.8445 - loss: 0.4198 - val_accuracy: 0.8239 - val_loss: 0.4791
Epoch 14/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 68s 1s/step - accuracy: 0.8125 - loss: 0.5203 - val_accuracy: 0.8519 - val_loss: 0.3606
Epoch 15/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 435s 9s/step - accuracy: 0.8561 - loss: 0.4106 - val_accuracy: 0.8168 - val_loss: 0.5053
Epoch 16/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 10s 57ms/step - accuracy: 0.7812 - loss: 0.4093 - val_accuracy: 0.8889 - val_loss: 0.3824
Epoch 17/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 490s 9s/step - accuracy: 0.8686 - loss: 0.3582 - val_accuracy: 0.8026 - val_loss: 0.5258
Epoch 18/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 67s 1s/step - accuracy: 0.7656 - loss: 0.5293 - val_accuracy: 0.7778 - val_loss: 0.7003
Epoch 19/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 433s 9s/step - accuracy: 0.8495 - loss: 0.3848 - val_accuracy: 0.8196 - val_loss: 0.4850
Epoch 20/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 9s 60ms/step - accuracy: 0.8438 - loss: 0.3875 - val_accuracy: 0.7407 - val_loss: 0.5932
Epoch 21/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 435s 9s/step - accuracy: 0.8849 - loss: 0.3365 - val_accuracy: 0.8452 - val_loss: 0.4444
Epoch 22/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 66s 1s/step - accuracy: 0.9375 - loss: 0.1953 - val_accuracy: 0.9259 - val_loss: 0.3290
Epoch 23/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 435s 9s/step - accuracy: 0.8862 - loss: 0.3241 - val_accuracy: 0.8267 - val_loss: 0.4800
Epoch 24/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 10s 51ms/step - accuracy: 0.9219 - loss: 0.1646 - val_accuracy: 1.0000 - val_loss: 0.0969
Epoch 25/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 491s 9s/step - accuracy: 0.8868 - loss: 0.3263 - val_accuracy: 0.8679 - val_loss: 0.4104
Epoch 26/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 8s 48ms/step - accuracy: 0.9062 - loss: 0.2602 - val_accuracy: 0.8148 - val_loss: 0.6849
Epoch 27/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 436s 9s/step - accuracy: 0.8992 - loss: 0.2939 - val_accuracy: 0.8338 - val_loss: 0.4429
Epoch 28/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 10s 48ms/step - accuracy: 0.8750 - loss: 0.3061 - val_accuracy: 0.7778 - val_loss: 0.7691
Epoch 29/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 487s 9s/step - accuracy: 0.8873 - loss: 0.3087 - val_accuracy: 0.8423 - val_loss: 0.4510
Epoch 30/30
45/45 ━━━━━━━━━━━━━━━━━━━━ 68s 1s/step - accuracy: 0.9062 - loss: 0.2933 - val_accuracy: 0.8148 - val_loss: 0.3639
```

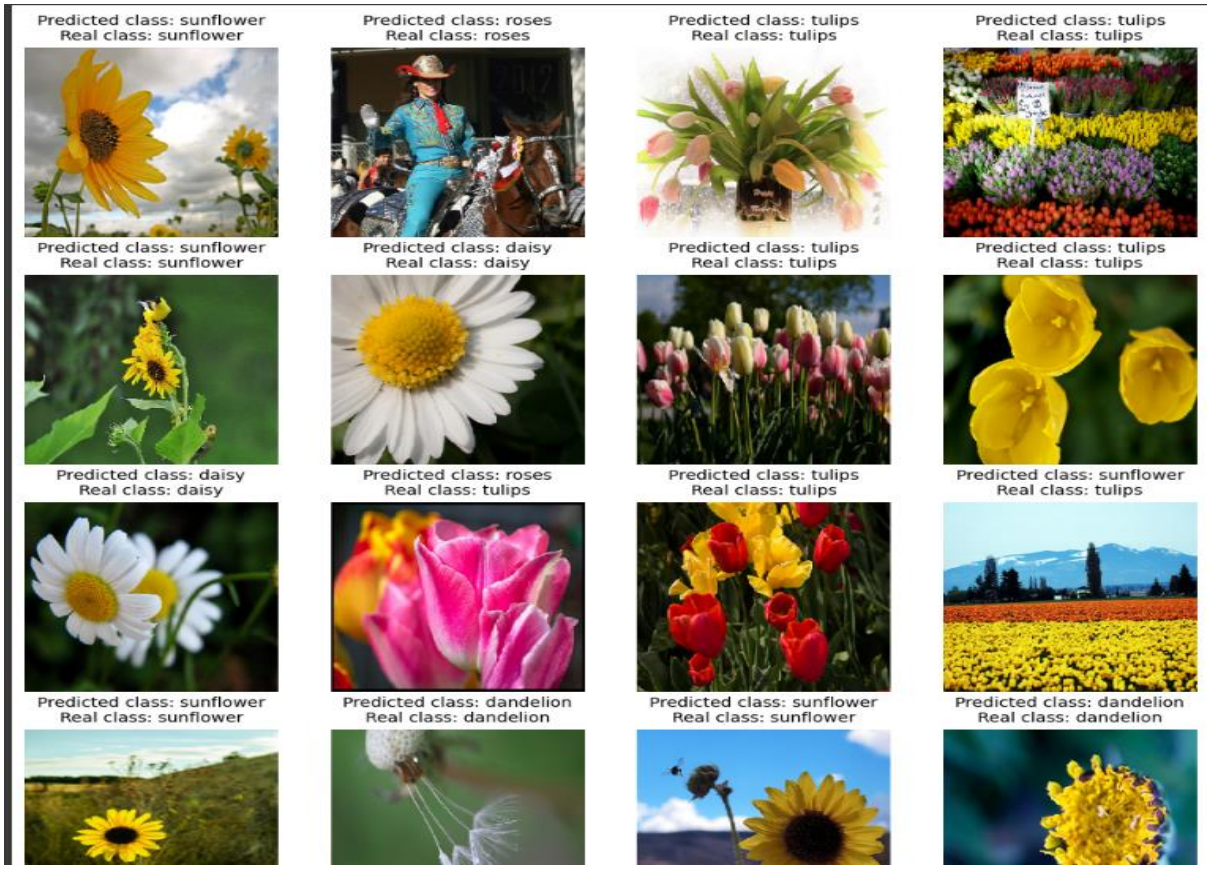
Grafik 3. iteracije.



Parametri 3. iteracije.

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 79s 6s/step - accuracy: 0.8639 - loss: 0.4237
12/12 ━━━━━━━━━━━━━━━━━━━━ 99s 7s/step
Test Loss: 0.4585
Test Accuracy: 0.8468
Klasifikacioni izvestaj:
              precision    recall  f1-score   support

       daisy       0.18      0.19      0.18       126
   dandelion       0.23      0.20      0.21       179
       roses       0.21      0.22      0.21       128
  sunflowers       0.17      0.20      0.19       139
      tulips       0.21      0.19      0.20       159

    accuracy                           0.20       731
   macro avg       0.20      0.20      0.20       731
weighted avg       0.20      0.20      0.20       731

Konfuziona matrica:
[[24 35 26 22 19]
 [25 36 32 48 38]
 [29 22 28 28 21]
 [22 32 22 28 35]
 [34 35 25 35 30]]
```

Nasumično isproban model na 16 slika opet. Stiče se utisak da bi i ljudsko oko pogrešilo izabranim slikama na mestima gde je model pogrešio.



5. Iteracija

Model tokom 5. iteraciji je ostao isto kao u prethodnoj samo su probani drugi hiperparametri. Konkretno learning rate je smanjen sa 0.001 na 0.0001, dok je momentum povećan sa 0.9 na 0.95.

```
[16] # Ucitavanje NASNetMobile modela sa unapred treniranim tezinama na ImageNet datasetu
     base_model = NASNetMobile(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))

     # AveragePooling 2D Layer
     x = base_model.output
     # Dodavanje konvolucionih slojeva sa batch normalization
     x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
     x = BatchNormalization()(x)
     x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
     x = BatchNormalization()(x)
     x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
     x = BatchNormalization()(x)
     x = GlobalAveragePooling2D()(x)
     x = Dense(1024, activation='relu')(x)
     x = Dropout(0.5)(x)

     predictions = Dense(5, activation='softmax')(x)
     model = Model(inputs=base_model.input, outputs=predictions)

     # Zamrzavanje slojeva baznog modela
     for layer in base_model.layers:
         layer.trainable = False

     # Koristimo SGD optimizator za kompajliranje modela
     model.compile(optimizer=SGD(learning_rate=0.0001, momentum=0.95),
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

     # Pregled modela
     model.summary()
```

Treniranje modela tokom 5. iteracije. Batch size je vraćen na 32. Dok je bro epoha bio 20.

```
Epoch 1/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 148s 1s/step - accuracy: 0.2918 - loss: 1.5629 - val_accuracy: 0.6506 - val_loss: 1.4182
Epoch 2/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 9s 94ms/step - accuracy: 0.7188 - loss: 1.2006 - val_accuracy: 0.5926 - val_loss: 1.4205
Epoch 3/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 75s 777ms/step - accuracy: 0.6242 - loss: 1.2142 - val_accuracy: 0.7159 - val_loss: 1.0618
Epoch 4/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7812 - loss: 1.0032 - val_accuracy: 0.8148 - val_loss: 1.0768
Epoch 5/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 74s 771ms/step - accuracy: 0.7103 - loss: 0.9803 - val_accuracy: 0.7557 - val_loss: 0.8364
Epoch 6/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8438 - loss: 0.8310 - val_accuracy: 0.6296 - val_loss: 1.0713
Epoch 7/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 83s 783ms/step - accuracy: 0.7304 - loss: 0.8463 - val_accuracy: 0.7486 - val_loss: 0.7413
Epoch 8/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8750 - loss: 0.6341 - val_accuracy: 0.8519 - val_loss: 0.6011
Epoch 9/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 81s 773ms/step - accuracy: 0.7637 - loss: 0.7501 - val_accuracy: 0.7685 - val_loss: 0.6536
Epoch 10/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7500 - loss: 0.7532 - val_accuracy: 0.8519 - val_loss: 0.6178
Epoch 11/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 74s 758ms/step - accuracy: 0.7849 - loss: 0.6509 - val_accuracy: 0.7827 - val_loss: 0.6198
Epoch 12/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.7188 - loss: 0.7172 - val_accuracy: 0.5926 - val_loss: 0.6471
Epoch 13/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 83s 795ms/step - accuracy: 0.8127 - loss: 0.5747 - val_accuracy: 0.7969 - val_loss: 0.5797
Epoch 14/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8750 - loss: 0.6032 - val_accuracy: 0.7407 - val_loss: 0.7499
Epoch 15/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 73s 741ms/step - accuracy: 0.8267 - loss: 0.5313 - val_accuracy: 0.8153 - val_loss: 0.5416
Epoch 16/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 7s 71ms/step - accuracy: 0.8438 - loss: 0.4317 - val_accuracy: 0.7778 - val_loss: 0.6739
Epoch 17/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 73s 726ms/step - accuracy: 0.8330 - loss: 0.5179 - val_accuracy: 0.8153 - val_loss: 0.5392
Epoch 18/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8519 - loss: 0.4878 - val_accuracy: 0.8889 - val_loss: 0.3248
Epoch 19/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 88s 799ms/step - accuracy: 0.8386 - loss: 0.4890 - val_accuracy: 0.8295 - val_loss: 0.5018
Epoch 20/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8438 - loss: 0.5606 - val_accuracy: 0.8519 - val_loss: 0.4850
```

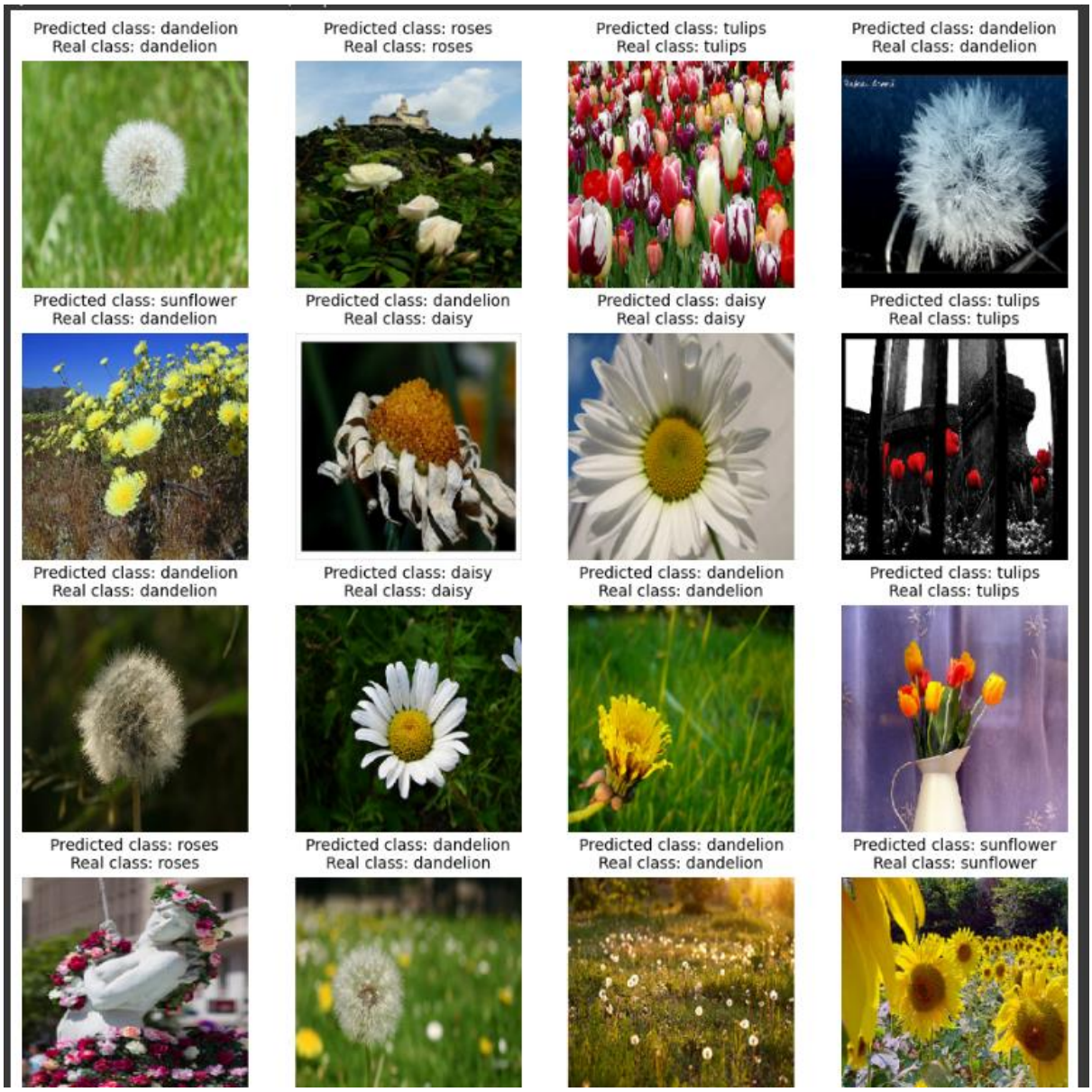
5. Evaluacija modela

Grafik 5. iteracije.

Metrike 5. iteracije.

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 15s 635ms/step - accuracy: 0.8160 - loss: 0.4834
23/23 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step
Test Loss: 0.4865
Test Accuracy: 0.8126
Klasifikacioni izvestaj:
              precision    recall  f1-score   support

       daisy       0.19      0.18      0.19       126
   dandelion       0.30      0.29      0.29       179
       roses       0.18      0.16      0.17       128
  sunflowers       0.22      0.23      0.22       139
      tulips       0.28      0.30      0.29       159

    accuracy                           0.24       731
   macro avg       0.23      0.23      0.23       731
weighted avg       0.24      0.24      0.24       731

Konfuziona matrica:
[[23 29 18 32 24]
 [31 52 30 34 32]
 [23 35 21 20 29]
 [17 32 19 32 39]
 [26 28 27 30 48]]
```

Isprobavanje modela tokom 5. iteracije. Vidimo da model greši u veoma specifičnim slučajevima.



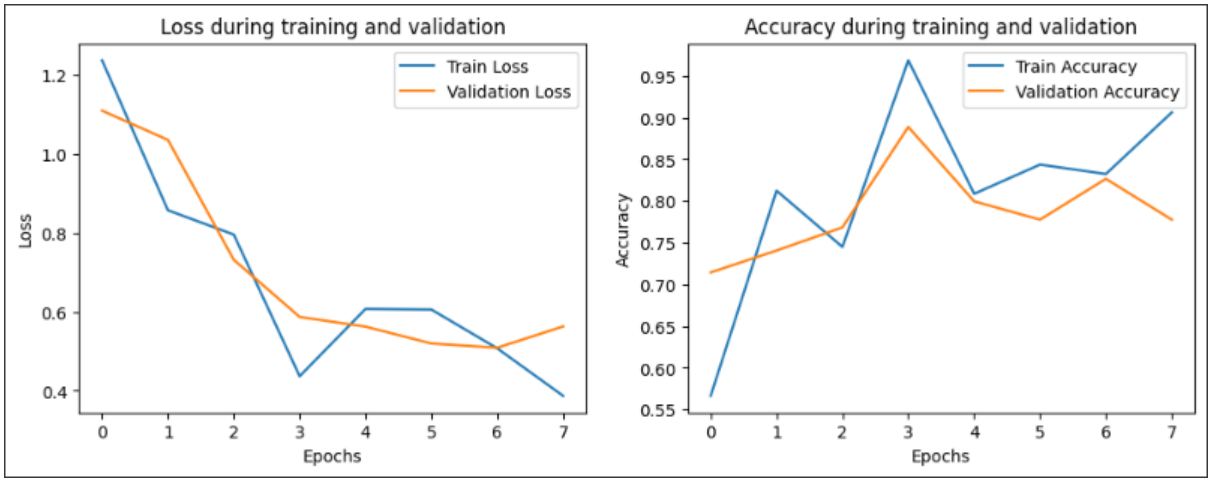
6. Iteracija

```
[22] # Ucitavanje NASNetMobile modela sa unapred treniranim tezinama na ImageNet datasetu
     base_model = NASNetMobile(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))

     # AveragePooling 2D Layer
     x = base_model.output
     # Dodavanje konvolucionih slojeva sa batch normalization
     x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
     x = BatchNormalization()(x)
     x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
     x = BatchNormalization()(x)
     x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
     x = BatchNormalization()(x)
     x = GlobalAveragePooling2D()(x)
     x = Dense(1024, activation='relu')(x)
     x = Dropout(0.5)(x)

     predictions = Dense(5, activation='softmax')(x)
     model = Model(inputs=base_model.input, outputs=predictions)

     # Zamrzavanje slojeva baznog modela
     for layer in base_model.layers:
         layer.trainable = False

     # Koristimo SGD optimizator za kompajliranje modela
     model.compile(optimizer=SGD(learning_rate=0.001, momentum=0.8),
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

     # Pregled modela
     model.summary()
```

U 6. iteraciji mkoristio se takodje isti model samo sada sa learning rate-om od 0.001 dok je momentum postavljen na 0.8. Broj epoha je postavljen na 8 dok je batch iznosio 32 opet.

```
Epoch 1/8
91/91 ━━━━━━━━━━━━━━━━━━━━ 145s 1s/step - accuracy: 0.4398 - loss: 1.4177 - val_accuracy: 0.7145 - val_loss: 1.1096
Epoch 2/8
 1/91 ━━━━━━━━━━━━━━━━━━━━ 13s 152ms/step - accuracy: 0.8125 - loss: 0.8573/usr/lib/python3.10/contextlib.py:153: UserWarnin
  self.gen.throw(typ, value, traceback)
91/91 ━━━━━━━━━━━━━━━━━━━━ 7s 79ms/step - accuracy: 0.8125 - loss: 0.8573 - val_accuracy: 0.7407 - val_loss: 1.0355
Epoch 3/8
91/91 ━━━━━━━━━━━━━━━━━━━━ 93s 790ms/step - accuracy: 0.7323 - loss: 0.8653 - val_accuracy: 0.7685 - val_loss: 0.7317
Epoch 4/8
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9688 - loss: 0.4370 - val_accuracy: 0.8889 - val_loss: 0.5871
Epoch 5/8
91/91 ━━━━━━━━━━━━━━━━━━━━ 78s 773ms/step - accuracy: 0.8094 - loss: 0.6252 - val_accuracy: 0.7997 - val_loss: 0.5626
Epoch 6/8
91/91 ━━━━━━━━━━━━━━━━━━━━ 7s 71ms/step - accuracy: 0.8438 - loss: 0.6058 - val_accuracy: 0.7778 - val_loss: 0.5198
Epoch 7/8
91/91 ━━━━━━━━━━━━━━━━━━━━ 75s 774ms/step - accuracy: 0.8286 - loss: 0.5258 - val_accuracy: 0.8267 - val_loss: 0.5091
Epoch 8/8
91/91 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9062 - loss: 0.3872 - val_accuracy: 0.7778 - val_loss: 0.5630
```

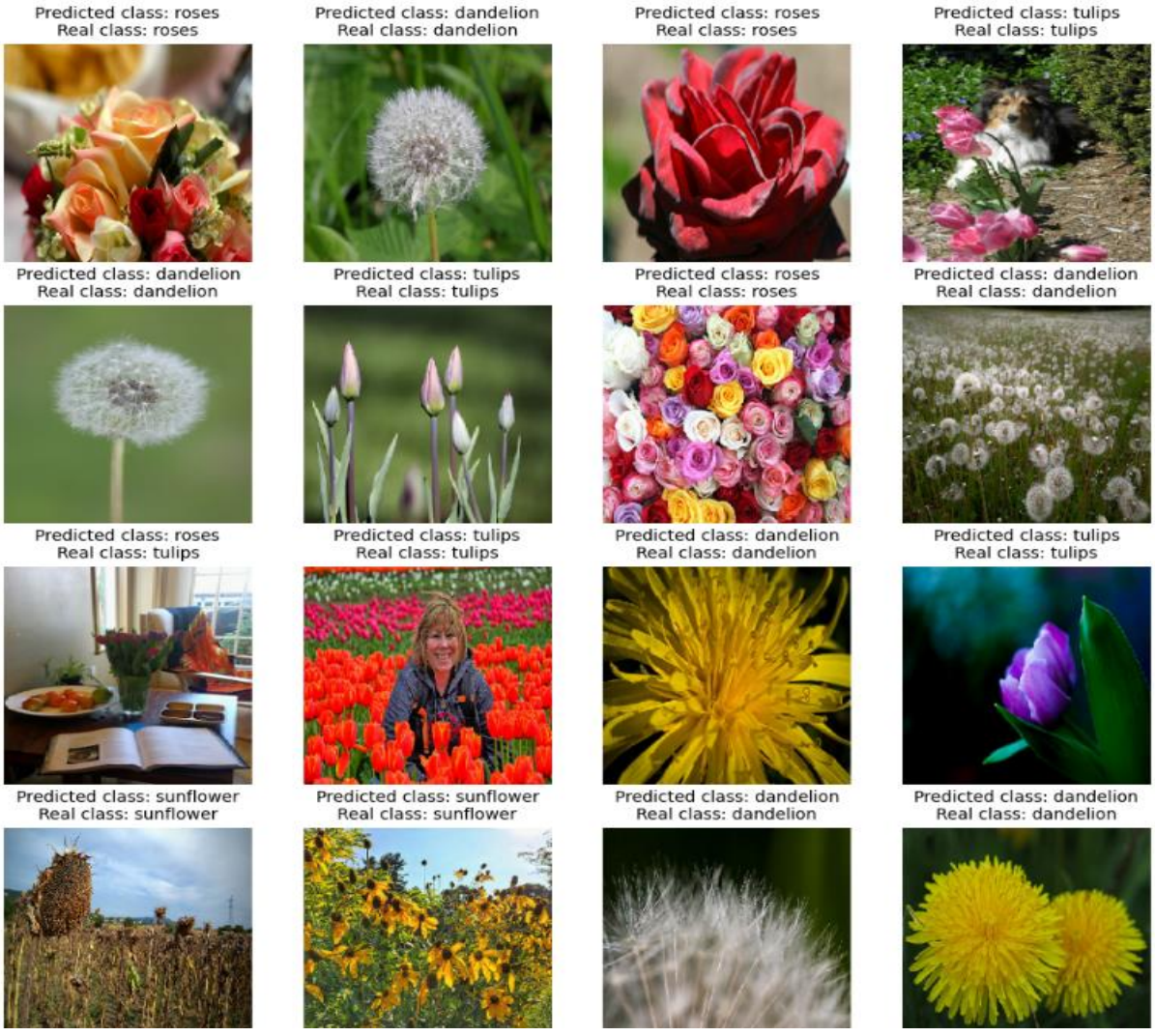
Grafik 6. iteracije.



Metrika 6. iteracije.

```
Test Loss: 0.5052
Test Accuracy: 0.8345
Klasifikacioni izvestaj:
              precision    recall  f1-score   support

       daisy       0.19      0.17      0.18       126
   dandelion       0.25      0.25      0.25       179
       roses       0.23      0.21      0.22       128
  sunflowers       0.19      0.22      0.21       139
      tulips       0.18      0.18      0.18       159

    accuracy                           0.21       731
   macro avg       0.21      0.21      0.21       731
weighted avg       0.21      0.21      0.21       731

Konfuziona matrica:
[[22 36 15 25 28]
 [25 45 29 46 34]
 [21 29 27 18 33]
 [21 34 19 31 34]
 [26 35 29 40 29]]
```

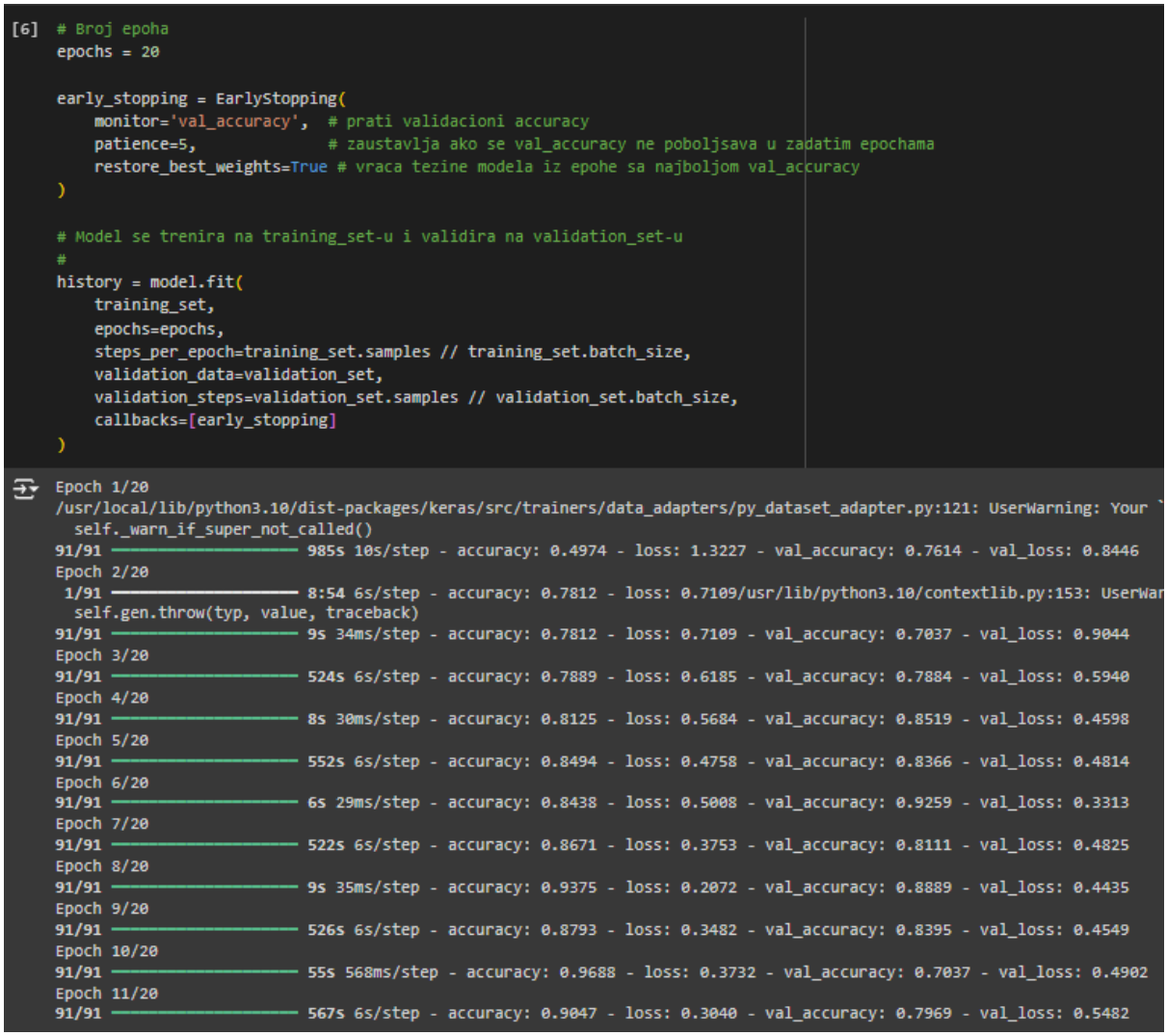
Isprobavanje modela 6. iteracije. Uzimajući specifičnosti na kojima model greši možemo reći da je već dobro istreniran i da su greške koje se pojavljuju opet veoma specifični slučajevi.



7. Iteracija

U 7 iteraciji se isprobava EarlyStopping tehnika za sprečavanje pretreniranja. Batch je opet 32 slike dok je broj epoha postavljen na 20. Learning rate i momentum su zadržani iz prethodne iteracije treniranja. Možemo videti da je model prestao sa treningom na 11 epohi jer je accuracy na validacionom skupu krenuo da opada i samim tim model se zaustavio tokom treniranja. Možemo zaključiti da je za treniranje modela u ovom slučaju bilo sasvim dovoljno 8 epoha.

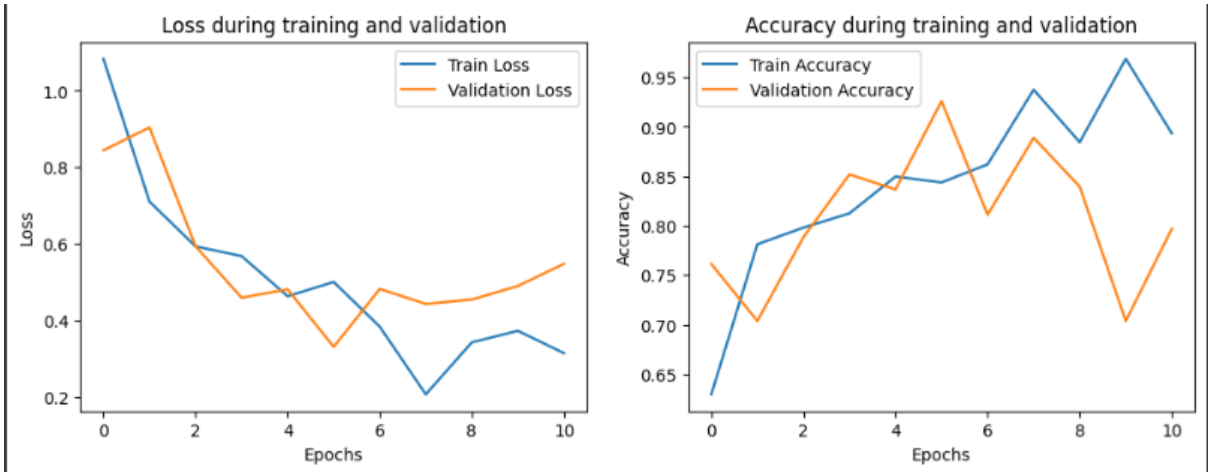
Patience je postavljen na 5 što znači da će se treniranje zaustaviti ako se performanse na validacionom skupu ne poboljšaju tokom 5 uzastopnih epoha. Takodje mogli smo postaviti manji patience npr. 3 što bi značilo da će se treniranje zaustaviti brže ukoliko želimo manje vreme obuke ili npr. ako povećamo na više od 5 ako očekujemo da će se model poboljšati. Povećavanjem zauzimamo više resursa dok je poenta da se model stavi na što veći broj epoha dok se ne pronadju dobre performanse modela. Samo za primer kako funkcioniše postavili smo na 20 zbog ograničenih resursa.



```
[6]  # Broj epoha
     epochs = 20

     early_stopping = EarlyStopping(
         monitor='val_accuracy',  # prati validacioni accuracy
         patience=5,              # zaustavlja ako se val_accuracy ne poboljsava u zadatim epochama
         restore_best_weights=True # vraca tezine modela iz epohe sa najboljom val_accuracy
     )

     # Model se trenira na training_set-u i validira na validation_set-u
     #
     history = model.fit(
         training_set,
         epochs=epochs,
         steps_per_epoch=training_set.samples // training_set.batch_size,
         validation_data=validation_set,
         validation_steps=validation_set.samples // validation_set.batch_size,
         callbacks=[early_stopping]
     )
```

```
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `
  self._warn_if_super_not_called()
91/91 ━━━━━━━━━━━━━━━━━━━━ 985s 10s/step - accuracy: 0.4974 - loss: 1.3227 - val_accuracy: 0.7614 - val_loss: 0.8446
Epoch 2/20
 1/91 ━━━━━━━━━━━━━━━━━━━━ 8:54 6s/step - accuracy: 0.7812 - loss: 0.7109/usr/lib/python3.10/contextlib.py:153: UserWar
  self.gen.throw(typ, value, traceback)
91/91 ━━━━━━━━━━━━━━━━━━━━ 9s 34ms/step - accuracy: 0.7812 - loss: 0.7109 - val_accuracy: 0.7037 - val_loss: 0.9044
Epoch 3/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 524s 6s/step - accuracy: 0.7889 - loss: 0.6185 - val_accuracy: 0.7884 - val_loss: 0.5940
Epoch 4/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 8s 30ms/step - accuracy: 0.8125 - loss: 0.5684 - val_accuracy: 0.8519 - val_loss: 0.4598
Epoch 5/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 552s 6s/step - accuracy: 0.8494 - loss: 0.4758 - val_accuracy: 0.8366 - val_loss: 0.4814
Epoch 6/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 6s 29ms/step - accuracy: 0.8438 - loss: 0.5008 - val_accuracy: 0.9259 - val_loss: 0.3313
Epoch 7/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 522s 6s/step - accuracy: 0.8671 - loss: 0.3753 - val_accuracy: 0.8111 - val_loss: 0.4825
Epoch 8/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 9s 35ms/step - accuracy: 0.9375 - loss: 0.2072 - val_accuracy: 0.8889 - val_loss: 0.4435
Epoch 9/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 526s 6s/step - accuracy: 0.8793 - loss: 0.3482 - val_accuracy: 0.8395 - val_loss: 0.4549
Epoch 10/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 55s 568ms/step - accuracy: 0.9688 - loss: 0.3732 - val_accuracy: 0.7037 - val_loss: 0.4902
Epoch 11/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 567s 6s/step - accuracy: 0.9047 - loss: 0.3040 - val_accuracy: 0.7969 - val_loss: 0.5482
```

Grafik 7. iteracije.



Metrike 7. iteracije.

```
Test Loss: 0.4755
Test Accuracy: 0.8317
Klasifikacioni izvestaj:
              precision    recall  f1-score   support

       daisy       0.11      0.10      0.11       126
   dandelion       0.18      0.18      0.18       179
       roses       0.20      0.18      0.19       128
  sunflowers       0.18      0.19      0.18       139
      tulips       0.24      0.26      0.25       159

    accuracy                           0.19       731
   macro avg       0.18      0.18      0.18       731
weighted avg       0.18      0.19      0.18       731

Konfuziona matrica:
[[13 33 23 26 31]
 [30 32 33 38 46]
 [26 32 23 22 25]
 [20 41 20 26 32]
 [28 40 16 33 42]]
```

8. Iteracija

U 8. iteraciji pokušali smo da implementiramo GridSearch za podešavanje hiperparametara. U ovom slučaju podešavali smo learning _rate, momentum, broj epoha i batch_size. Pokušano je preko biblioteke sklearn da se implementira GridSearchCV kao i RandomSearchCV klasa ali je iskočio problem oko parametara learning_rate i momentum jer nisu bili dobro prosledjivani u naš model koji je sada postavljen u funkciju koja prima argumente learning_rate i momentum koji su potrebni za SGD optimizator. Takodje za potrebe ovog pokušaja probana je i KerasClassifier biblioteka koja je nepohodna za gore navedene klase.

Odlučeno je da se koristi malo drugačiji pristup i GridSearch je implementiran manuelno uz pomoć ParameterGrid klase iz iste biblioteke.

Grid je podešen da redom isprobava sve parametre koji su željeni u svim kombinacijama i da vraća skor za svaki pokušaj treniranja. Skor se poredi prvi svakoj iteraciji sa različitim parametrima i izlaz nakon izvršavanja će dati parametre koji su dali najbolji skor tokom treniranja.

10. RandomSearch -> Koristicemo KerasClassifier koji predstavlja wrapper klasu koja nam omogucava rad sa RandomSearch-om i dok cemo sam model smestiti u funkciju koja prihvata parametre learning_rate i momentum

```
[ ]  def create_model(learning_rate,momentum):
        base_model = NASNetMobile(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))
        x = base_model.output
        x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
        x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
        x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
        x = GlobalAveragePooling2D()(x)
        x = Dropout(0.5)(x)
        x = Dense(1024, activation='relu')(x)
        x = Dense(5, activation='softmax')(x)
        model = Model(inputs=base_model.input, outputs=x)

        model.compile(optimizer=SGD(learning_rate=learning_rate, momentum=momentum),
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])
        return model
```

10.2. Definisanje hiperametara za RandomSearch i GridSearch

```
[ ]  # Definisanje opsega hiperparametara za Random Search
     param_grid = {
         'learning_rate': [0.0001, 0.001, 0.01],
         'momentum': [0.85, 0.9, 0.95],
         'batch_size': [16, 32],
         'epochs': [10, 20]
     }
```

10.3. RandomSearch

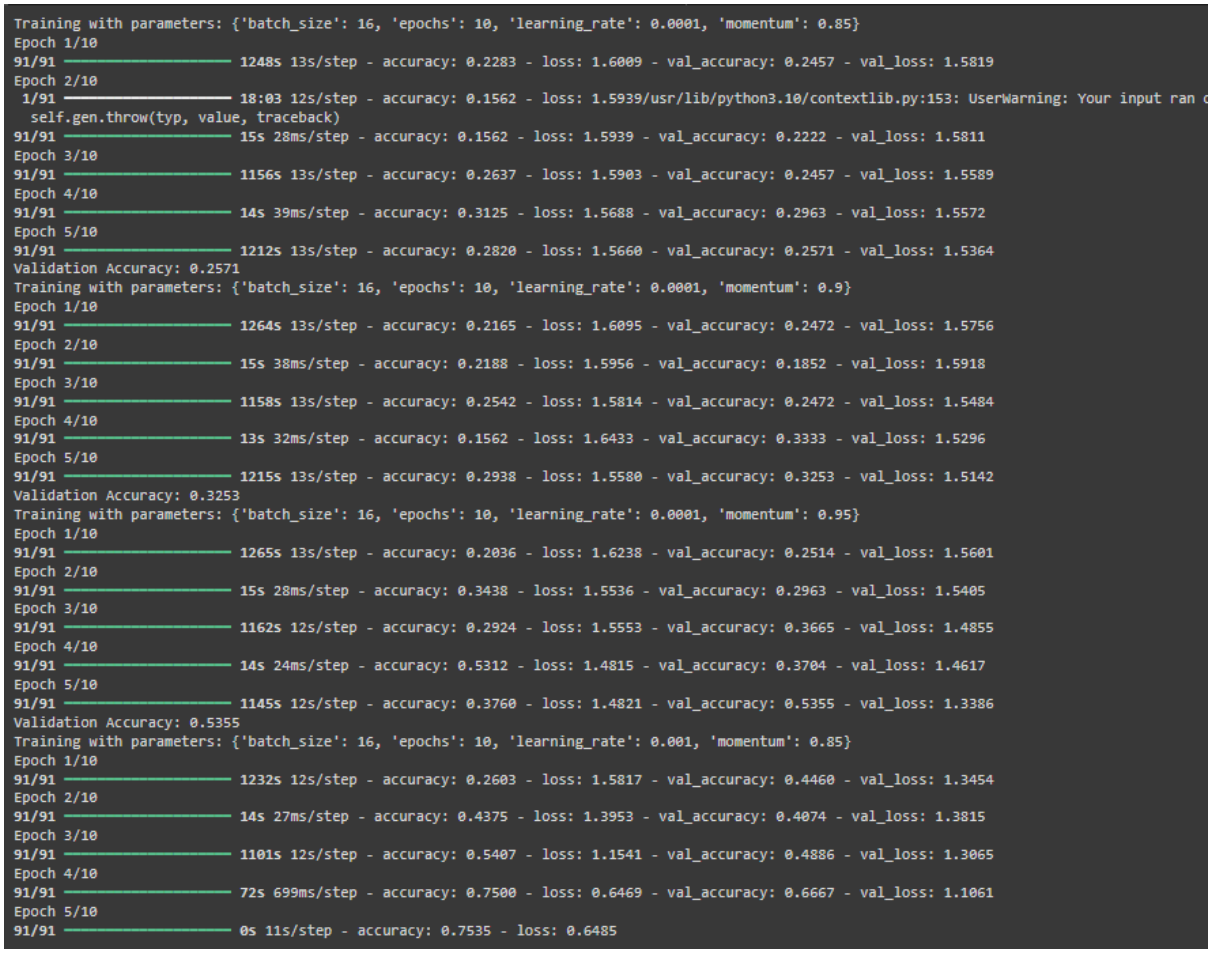
```
[ ]  # Funkcija koja trenira model na prosledjenim parametrima i vraca accuracy
     def evaluate_model(model, training_set, validation_set, epochs, batch_size):
         history = model.fit(training_set,
                             epochs=epochs,
                             steps_per_epoch=training_set.samples // training_set.batch_size,
                             validation_data=validation_set,
                             validation_steps=validation_set.samples // validation_set.batch_size,
                             callbacks=[early_stopping],
                             verbose=1)

         return history.history['val_accuracy'][-1]
```

```
# Inicijalizacija ParameterGrid-a
grid = ParameterGrid(param_grid)

# Lista koja cuva rezultate
results = []

# Random Search
for params in grid:
    print(f"Training with parameters: {params}")

    learning_rate = params['learning_rate']
    momentum = params['momentum']
    batch_size = params['batch_size']
    epochs = params['epochs']

    model = create_model(learning_rate, momentum)
    val_accuracy = evaluate_model(model, training_set, validation_set, epochs, batch_size)

    results.append({
        'params': params,
        'val_accuracy': val_accuracy
    })

    print(f"Validation Accuracy: {val_accuracy:.4f}")

# Sortiranje skorova
sorted_results = sorted(results, key=lambda x: x['val_accuracy'], reverse=True)

# Najbolji parametri
print("Best parameters:")
print(sorted_results[0])
```

Iz priložene slike možemo videti različite parametre koji se koriste tokom svake iteracije treniranja.

Koristili smo isti model kao u prethodnim iteracijama čak i sa primenjivanjem EarlyStopping tehnike ali zbog ograničenih resursa na Google Colab-u dogadjale su se runtime greške ili timeout-i sesija.

Probano je na više različitih varijanti runtime-a. Nažalost ovo je hardversko ograničenje neplaćene verzije Google Collab-a. Slika služi kao primer treniranja modela sa raznim hiperaparametrima koristeći ovu funkciju.



9.Iteracija - Finalna

U ovoj iteraciji baznom modelu smo `odmrzli` poslednjih 20 slojeva kako bismo trenirali i na njima i tim postigli Fine-Tuning modela i poboljšali performanse modela. Takodje implementirana je skip konekcija na modelu koji koristimo.

Skip konekcije omogućavaju prolaz informacija iz jednog sloja direktno do sloja koji se nalazi nekoliko slojeva dalje u mreži, zaobilazeći jedan ili više slojeva između.

Veoma duboke mreže, kao što su mreže sa mnogo slojeva, u njima može doći do problema sa obukom zbog nestanka gradijenata ili eksplozije gradijenata.Skip konekcija u tom slučaju omogućavaju direktan put za gradijente tokom obuke, što pomaže u efikasnijem učenju i smanjuje ove probleme.

Primer modela iz 9. iteracije.

```
19] # Ucitavanje NASNetMobile modela
    base_model = NASNetMobile(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

    # Zamrzavanje svih slojeva baznog modela
    for layer in base_model.layers:
        layer.trainable = False

    # Odmrzavanje poslednjih 20 slojeva
    for layer in base_model.layers[-20:]:
        layer.trainable = True

    # Kreiranje skip konekcije
    skip_connection = Conv2D(128, (1, 1), padding='same')(base_model.output)
    skip_connection = GlobalAveragePooling2D()(skip_connection)

    # Dodavanje konvolucionih slojeva sa batch normalization
    x = Conv2D(512, (3, 3), activation='relu', padding='same')(base_model.output)
    x = BatchNormalization()(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    x = GlobalAveragePooling2D()(x)

    # Dodavanje skip konekcije
    x = Add()([x, skip_connection])

    # Nastavak modela
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)

    predictions = Dense(5, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    # Kreiranje optimizatora sa specificnim learning_rate i momentum parametrima
    optimizer = SGD(learning_rate=0.001, momentum=0.9)

    # Kompajliranje modela
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    # Pregled modela
    model.summary()
```

Pregled parametara modela tokom 9. iteracije

normal_concat_12 (Concatenate)	(None, 7, 7, 1056)	0	adjust_bn_12[0][0], normal_add_1_12[0][0], normal_add_2_12[0][0], normal_add_3_12[0][0], normal_add_4_12[0][0], normal_add_5_12[0][0]
activation_939 (Activation)	(None, 7, 7, 1056)	0	normal_concat_12[0][0]
conv2d_16 (Conv2D)	(None, 7, 7, 512)	4,866,560	activation_939[0][0]
batch_normalization_12 (BatchNormalization)	(None, 7, 7, 512)	2,048	conv2d_16[0][0]
conv2d_17 (Conv2D)	(None, 7, 7, 256)	1,179,904	batch_normalization_1…
batch_normalization_13 (BatchNormalization)	(None, 7, 7, 256)	1,024	conv2d_17[0][0]
conv2d_18 (Conv2D)	(None, 7, 7, 128)	295,040	batch_normalization_1…
batch_normalization_14 (BatchNormalization)	(None, 7, 7, 128)	512	conv2d_18[0][0]
conv2d_15 (Conv2D)	(None, 7, 7, 128)	135,296	activation_939[0][0]
global_average_pooling2d… (GlobalAveragePooling2D)	(None, 128)	0	batch_normalization_1…
global_average_pooling2d… (GlobalAveragePooling2D)	(None, 128)	0	conv2d_15[0][0]
add_24 (Add)	(None, 128)	0	global_average_poolin… global_average_poolin…
dense_9 (Dense)	(None, 1024)	132,096	add_24[0][0]
dropout_4 (Dropout)	(None, 1024)	0	dense_9[0][0]
dense_10 (Dense)	(None, 5)	5,125	dropout_4[0][0]

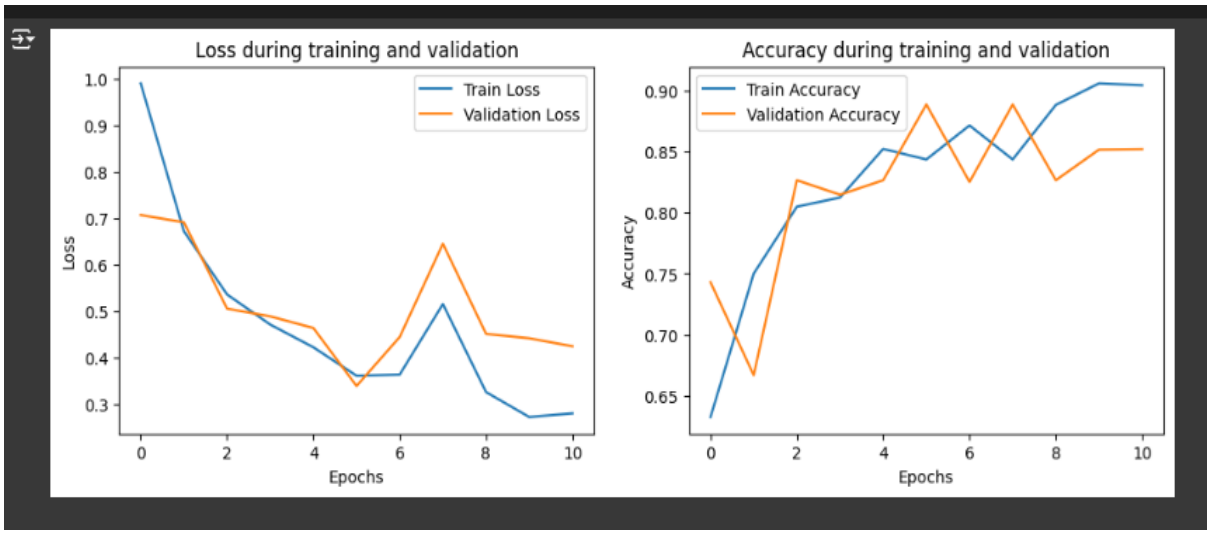
```
Total params: 10,887,321 (41.53 MB)
Trainable params: 6,786,005 (25.89 MB)
Non-trainable params: 4,101,316 (15.65 MB)
```

Treniranje modela tokom 9. iteracije. Kao što vidimo treniranje se zaustavilo na 11 epohi od 20 zbog EarlyStopping tehnike. Batch size je ostavljen na 32 dok je learning rate na vrednosti od 0.001 i momentum na vrednost od 0.9.

```
# Model se trenira na training_set-u i validira na validation_set-u
history = model.fit(
    training_set,
    epochs=epochs,
    steps_per_epoch=training_set.samples // training_set.batch_size,
    validation_data=validation_set,
    validation_steps=validation_set.samples // validation_set.batch_size,
    callbacks=[early_stopping]
)
```

```
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `P
  self._warn_if_super_not_called()
91/91 ━━━━━━━━━━━━━━━━━━━━ 575s 6s/step - accuracy: 0.4893 - loss: 1.2668 - val_accuracy: 0.7429 - val_loss: 0.7076
Epoch 2/20
 1/91 ━━━━━━━━━━━━━━━━━━━━ 5:30 4s/step - accuracy: 0.7500 - loss: 0.6728/usr/lib/python3.10/contextlib.py:153: UserWarn
  self.gen.throw(typ, value, traceback)
91/91 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.7500 - loss: 0.6728 - val_accuracy: 0.6667 - val_loss: 0.6919
Epoch 3/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 532s 6s/step - accuracy: 0.7935 - loss: 0.5749 - val_accuracy: 0.8267 - val_loss: 0.5059
Epoch 4/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 8s 29ms/step - accuracy: 0.8125 - loss: 0.4717 - val_accuracy: 0.8148 - val_loss: 0.4894
Epoch 5/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 534s 6s/step - accuracy: 0.8360 - loss: 0.4613 - val_accuracy: 0.8267 - val_loss: 0.4642
Epoch 6/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 6s 27ms/step - accuracy: 0.8438 - loss: 0.3613 - val_accuracy: 0.8889 - val_loss: 0.3393
Epoch 7/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 536s 6s/step - accuracy: 0.8646 - loss: 0.3657 - val_accuracy: 0.8253 - val_loss: 0.4449
Epoch 8/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 9s 35ms/step - accuracy: 0.8438 - loss: 0.5159 - val_accuracy: 0.8889 - val_loss: 0.6458
Epoch 9/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 531s 6s/step - accuracy: 0.8776 - loss: 0.3448 - val_accuracy: 0.8267 - val_loss: 0.4516
Epoch 10/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 7s 26ms/step - accuracy: 0.9062 - loss: 0.2725 - val_accuracy: 0.8519 - val_loss: 0.4421
Epoch 11/20
91/91 ━━━━━━━━━━━━━━━━━━━━ 559s 6s/step - accuracy: 0.9090 - loss: 0.2675 - val_accuracy: 0.8523 - val_loss: 0.4249
```

Grafik tokom 9. iteracije.



Metrike 9. iteracije.

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 84s 4s/step - accuracy: 0.8033 - loss: 0.4940
WARNING:tensorflow:5 out of the last 26 calls to <function TensorFlowTrainer.make_predict
23/23 ━━━━━━━━━━━━━━━━━━━━ 97s 4s/step
Test Loss: 0.4807
Test Accuracy: 0.8194
Klasifikacioni izvestaj:
              precision    recall  f1-score   support

       daisy       0.10      0.09      0.09       126
   dandelion       0.28      0.29      0.29       179
       roses       0.19      0.16      0.18       128
  sunflowers       0.19      0.21      0.20       139
      tulips       0.22      0.24      0.23       159

    accuracy                           0.21       731
   macro avg       0.20      0.20      0.20       731
weighted avg       0.20      0.21      0.20       731

Konfuziona matrica:
[[11 33 19 26 37]
 [24 52 26 33 44]
 [26 32 21 25 24]
 [20 35 25 29 30]
 [32 33 20 36 38]]
```

Isprobavanje na 16 nasumičnih slika. U ovom slučaju model je pogodio sve slike jer mu ovaj batch slika odgovara iako je accuracy pristojan, daljim treniranjem se mogu postići jos bolji rezultati.