1. Importovanje potrebnih biblioteka za rad i instaliranje paketa

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

In [5]:
```
!jupyter nbconvert --to html '/content/drive/MyDrive/Colab Notebooks/Projekat2024/Projekat2024.ipynb'
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab Notebooks/Projekat2024/Projekat2024.ipynb to html
[NbConvertApp] Writing 3513725 bytes to /content/drive/MyDrive/Colab Notebooks/Projekat2024/Projekat2024.html
```

In [ ]:
```
!pip install scikeras
```

In [ ]:
```
import os
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import requests
from PIL import Image
from io import BytesIO
import tensorflow.keras.backend as K
from tensorflow.keras.preprocessing.image import ImageDataGenerator #Biblioteka za manipulaciju nad slikama
from tensorflow.keras.applications import NASNetMobile #importovanje NASNetMobile modela
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, BatchNormalization, Dense, Conv2D, MaxPooling2D, Add,
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, ParameterGrid
from sklearn.metrics import classification_report, confusion_matrix
import random
```

1. Putanja do dataset-a i priprema podataka. Uzecemo 1000 slika nasumicno iz originalnog trening dataset-a i agumentovati ih i dodati na trening set.

In [ ]:
```
# 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

# Originalni set bez augmentacije (samo normalizacija)
original_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)

# Originalni set za trening (bez augmentacije)
original_training_set = original_datagen.flow_from_directory(
    dataset_path,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training',
    shuffle=False  # Ne mesamo slike da bi bile u fiksnom redosledu za random sampling
)

# Generator za augmentovane slike
augmented_datagen = ImageDataGenerator(
    rescale=1./255,  # reskaliranje piksela na raspon od 0 do 1
    rotation_range=40,  # nasumično rotiranje slika od -40 do 40 stepeni
    zoom_range=0.2,  # nasumicno zumiranje slika +- 20%
    shear_range=0.2,  # Dodavanje shearing augmentacije
    width_shift_range=0.2,  # Horizontalno pomeranje slika
    height_shift_range=0.2,  # Vertikalno pomeranje slika
    horizontal_flip=True  # nasumicno horizontalno okretanje slika
)

# Funkcija koja preuzima sve slike iz originalnog training seta
def get_all_images(generator):
    images = []
    labels = []
    for i in range(generator.samples // batch_size):
        batch_images, batch_labels = next(generator)
        images.append(batch_images)
        labels.append(batch_labels)
    return np.vstack(images), np.vstack(labels)

original_images, original_labels = get_all_images(original_training_set)

# Nasumicno odaberi 1000 slika iz originalnog seta
```

```
num_samples = 1000 # Broj izabranih slika
indices = np.random.choice(original_images.shape[0], num_samples, replace=False)
sampled_images = original_images[indices]
sampled_labels = original_labels[indices]

# Augmentacija za nasumicnih 1000 slika
augmented_images = []
augmented_labels = []

for i in range(0, num_samples, batch_size):
    batch_images = sampled_images[i:i+batch_size]
    batch_labels = sampled_labels[i:i+batch_size]
    augmented_batch = augmented_datagen.flow(batch_images, batch_labels, batch_size=batch_size, shuffle=False)
    aug_images, aug_labels = next(augmented_batch)
    augmented_images.append(aug_images)
    augmented_labels.append(aug_labels)

augmented_images = np.vstack(augmented_images)
augmented_labels = np.vstack(augmented_labels)

# Kombinovanje originalnih i augmentovanih slika
combined_images = np.concatenate((original_images, augmented_images))
combined_labels = np.concatenate((original_labels, augmented_labels))

# Ispisivanje broja slika u kombinovanom setu
print(f'Number of original dataset pictures: {original_images.shape[0]}')
print(f'Number of agumented pictures from original dataset: {augmented_images.shape[0]}')
print(f'Number of pictures in combined dataset: {combined_images.shape[0]}')

# Kreiranje validacionog seta
validation_set = original_datagen.flow_from_directory(
    dataset_path,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 2939 images belonging to 5 classes.
Number of original dataset pictures: 2912
Number of agumented pictures from original dataset: 1000
Number of pictures in combined dataset: 3912
Found 731 images belonging to 5 classes.
```

1. NASNetMobile, zamrzavanje pretreniranog sloja i priprema modela

In [ ]:
```
# 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()
```

```
Model: "functional"
```

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)

1. Obuka modela i koriscenje EarlyStopping tehnike

In [ ]:
```
# 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
    mode='max',             # Trazenje maksimuma (najbolji accuracy)
    min_delta=0.001,        # Minimalna promena u validacijskom accuracy koja se mora dogoditi da bi se smatralo da je doslo do p
    verbose=1               # Prikazivanje poruka o zaustavljanju obuke
)

# Model se trenira na training_set-u i validira na validation_set-u
history = model.fit(
    combined_images, combined_labels,
    epochs=20,
    validation_data=validation_set,
    callbacks=[early_stopping]
)
```

```
Epoch 1/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.4283 - loss: 1.3278
```

```
123/123 ━━━━━━━━━━━━━━━━━━━━ 660s 5s/step - accuracy: 0.4292 - loss: 1.3260 - val_accuracy: 0.7839 - val_loss: 0.6298
Epoch 2/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 677s 5s/step - accuracy: 0.6732 - loss: 0.7820 - val_accuracy: 0.8181 - val_loss: 0.4692
Epoch 3/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 625s 5s/step - accuracy: 0.7224 - loss: 0.6597 - val_accuracy: 0.8413 - val_loss: 0.4281
Epoch 4/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 676s 5s/step - accuracy: 0.7474 - loss: 0.5702 - val_accuracy: 0.8536 - val_loss: 0.4281
Epoch 5/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 619s 5s/step - accuracy: 0.7768 - loss: 0.5145 - val_accuracy: 0.8440 - val_loss: 0.4854
Epoch 6/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 620s 5s/step - accuracy: 0.7898 - loss: 0.4913 - val_accuracy: 0.8454 - val_loss: 0.4499
Epoch 7/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 619s 5s/step - accuracy: 0.8004 - loss: 0.4652 - val_accuracy: 0.8454 - val_loss: 0.4854
Epoch 8/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 618s 5s/step - accuracy: 0.8215 - loss: 0.4285 - val_accuracy: 0.8495 - val_loss: 0.5072
Epoch 9/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 617s 5s/step - accuracy: 0.8167 - loss: 0.4353 - val_accuracy: 0.8550 - val_loss: 0.5001
Epoch 10/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 621s 5s/step - accuracy: 0.8275 - loss: 0.4044 - val_accuracy: 0.8618 - val_loss: 0.4792
Epoch 11/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 619s 5s/step - accuracy: 0.8279 - loss: 0.4152 - val_accuracy: 0.8632 - val_loss: 0.5114
Epoch 12/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 620s 5s/step - accuracy: 0.8144 - loss: 0.4250 - val_accuracy: 0.8550 - val_loss: 0.5678
Epoch 13/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 617s 5s/step - accuracy: 0.8299 - loss: 0.3926 - val_accuracy: 0.8495 - val_loss: 0.5312
Epoch 14/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 620s 5s/step - accuracy: 0.8239 - loss: 0.3949 - val_accuracy: 0.8509 - val_loss: 0.5680
Epoch 15/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 621s 5s/step - accuracy: 0.8217 - loss: 0.4033 - val_accuracy: 0.8550 - val_loss: 0.5539
Epoch 16/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 637s 5s/step - accuracy: 0.8267 - loss: 0.3965 - val_accuracy: 0.8687 - val_loss: 0.5475
Epoch 17/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 673s 5s/step - accuracy: 0.8207 - loss: 0.3965 - val_accuracy: 0.8605 - val_loss: 0.5673
Epoch 18/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 693s 5s/step - accuracy: 0.8301 - loss: 0.3984 - val_accuracy: 0.8591 - val_loss: 0.5689
Epoch 19/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 669s 5s/step - accuracy: 0.8165 - loss: 0.4065 - val_accuracy: 0.8577 - val_loss: 0.5885
Epoch 20/20
123/123 ━━━━━━━━━━━━━━━━━━━━ 614s 5s/step - accuracy: 0.8214 - loss: 0.3987 - val_accuracy: 0.8577 - val_loss: 0.5617
Restoring model weights from the end of the best epoch: 16.
```

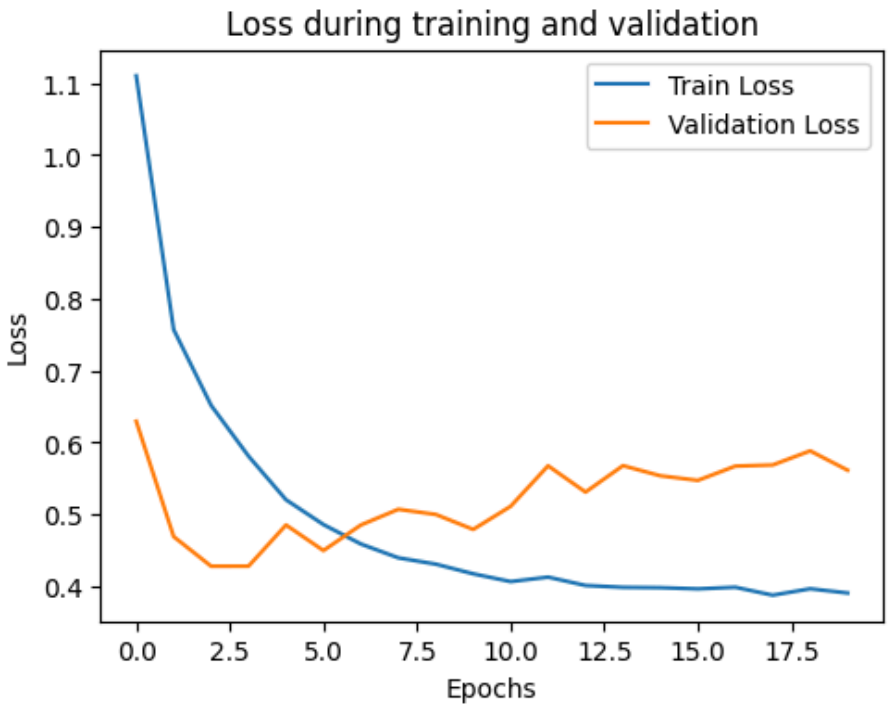
1. Evaluacija modela i graficki prikaz loss i accuracy-ja

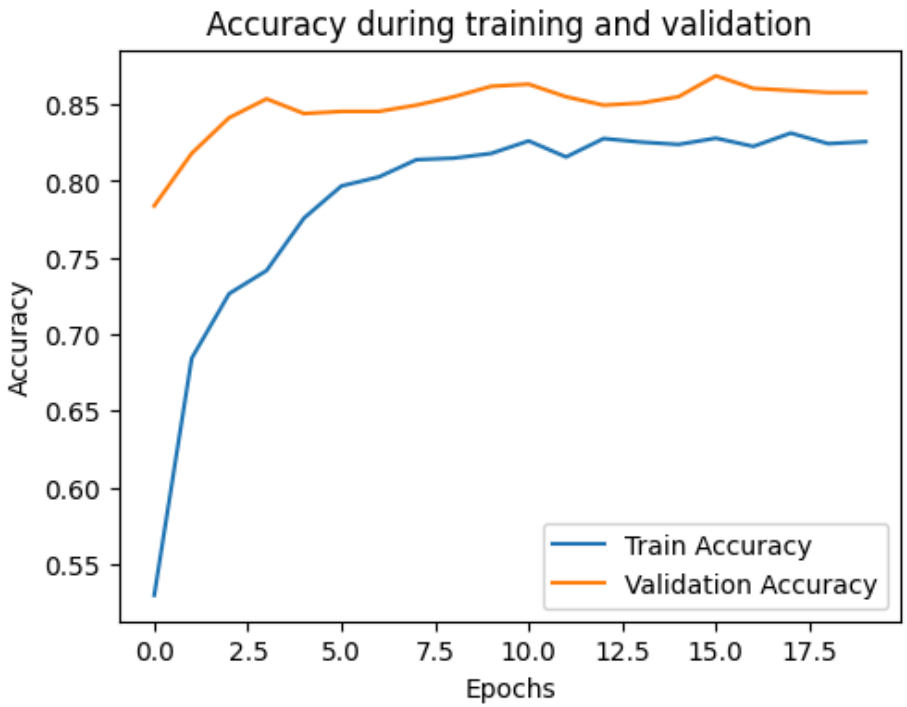
In [ ]:
```
# Loss tokom epoha
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss during training and validation')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Tacnost tokom epoha
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy during training and validation')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```





1. Klasifikacioni izvestaj,loss,accuracy i konfuziona matrica

In [ ]:
```
# Evaluiranje modela na testnom skupu
loss, accuracy = model.evaluate(validation_set, verbose=1)

# Predikcije modela
test_prediction = model.predict(validation_set)
test_prediction_classes = np.argmax(test_prediction, axis=1)

# Ucitavanje imena klasa
test_labels_classes = validation_set.classes
class_names = list(validation_set.class_indices.keys())

# Loss i Accuracy
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

# Klasifikacioni izvestaj
print("Klasifikacioni izvestaj:")
```

```
print(classification_report(test_labels_classes, test_prediction_classes, target_names=class_names))

# Konfuziona matrica
print("Konfuziona matrica:")
print(confusion_matrix(test_labels_classes, test_prediction_classes))
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 75s 3s/step - accuracy: 0.8808 - loss: 0.4840
23/23 ━━━━━━━━━━━━━━━━━━━━ 91s 4s/step
Test Loss: 0.5475
Test Accuracy: 0.8687
Klasifikacioni izvestaj:
              precision    recall  f1-score   support

       daisy       0.19      0.17      0.18       126
   dandelion       0.25      0.26      0.25       179
       roses       0.16      0.16      0.16       128
  sunflowers       0.23      0.25      0.24       139
      tulips       0.25      0.24      0.25       159

    accuracy                           0.22       731
   macro avg       0.22      0.22      0.22       731
weighted avg       0.22      0.22      0.22       731

Konfuziona matrica:
[[22 27 29 24 24]
 [28 46 30 41 34]
 [27 34 20 24 23]
 [21 33 18 35 32]
 [20 43 27 31 38]]
```

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

In [ ]:
```
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 ━━━━━━━━━━━━━━━━━━━━ 8s 8s/step
Model predvidja da je ova slika: daisy
```

9.1. Isprobavanje na 16 random slika iz dataset-a

In [ ]:
```
# 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=1
    plt.axis('off')
plt.show()
```

```
Found 3670 images belonging to 5 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
```

Predicted class: daisy
Real class: daisy

Predicted class: tulips
Real class: tulips

Predicted class: roses
Real class: daisy

Predicted class: dandelion
Real class: dandelion

Predicted class: dandelion
Real class: dandelion

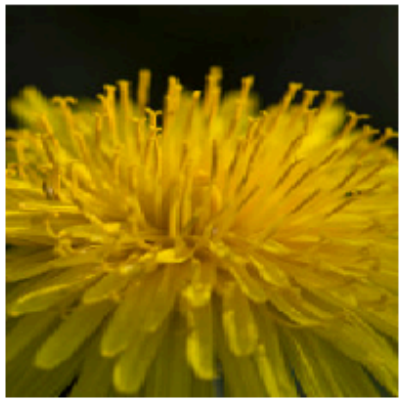
Predicted class: roses
Real class: roses

Predicted class: dandelion
Real class: dandelion

Predicted class: sunflower
Real class: sunflower

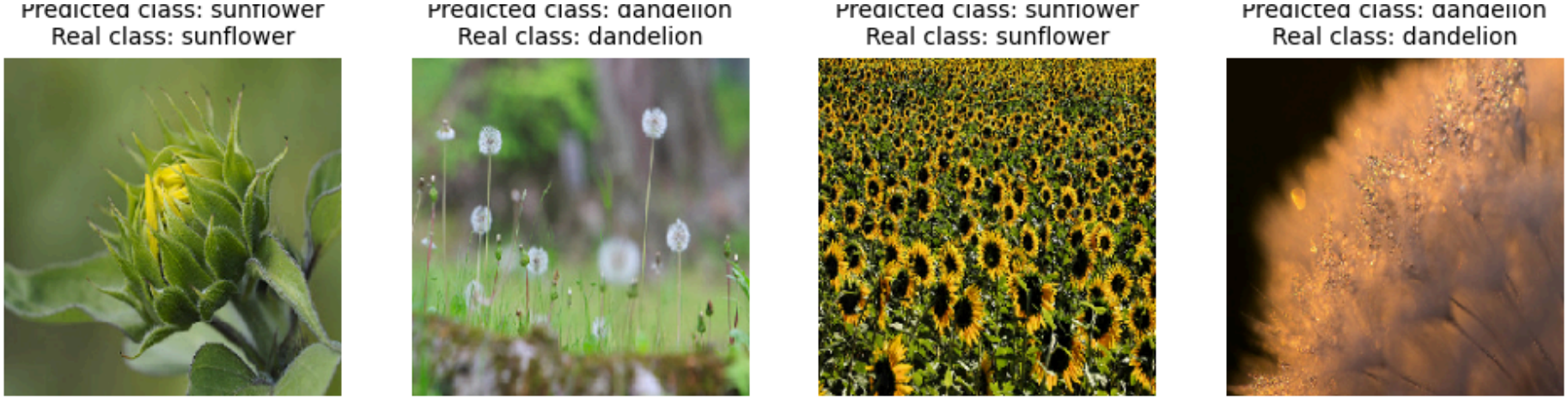
Predicted class: tulips
Real class: tulips

Predicted class: dandelion
Real class: dandelion



Predicted class: sunflower
Real class: sunflower

Predicted class: dandelion
Real class: dandelion



1. RandomSearch -> Koristicemo KerasClassifier koji predstavlja wrapper klasu koja nam omogucava rad sa Random ili Grid Search-om i modelom, dok cemo sam model smestiti u funkciju koja prihvata parametre learning_rate i momentum

In [ ]:
```
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 ili GridSearch

In [ ]:
```
# 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. GridSearch manuelna implementacija

In [ ]:
```
# 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]
```

In [ ]:
```
# 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])
```

```
Training with parameters: {'batch_size': 16, 'epochs': 10, 'learning_rate': 0.0001, 'momentum': 0.85}
Epoch 1/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1248s 13s/step - accuracy: 0.2283 - loss: 1.6009 - val_accuracy: 0.2457 - val_loss: 1.5819
Epoch 2/10
 1/91 ━━━━━━━━━━━━━━━━━━━━ 18:03 12s/step - accuracy: 0.1562 - loss: 1.5939
```

```
91/91 ━━━━━━━━━━━━━━━━━━━━ 15s 28ms/step - accuracy: 0.1562 - loss: 1.5939 - val_accuracy: 0.2222 - val_loss: 1.5811
Epoch 3/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1156s 13s/step - accuracy: 0.2637 - loss: 1.5903 - val_accuracy: 0.2457 - val_loss: 1.5589
Epoch 4/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 14s 39ms/step - accuracy: 0.3125 - loss: 1.5688 - val_accuracy: 0.2963 - val_loss: 1.5572
Epoch 5/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1212s 13s/step - accuracy: 0.2820 - loss: 1.5660 - val_accuracy: 0.2571 - val_loss: 1.5364
Validation Accuracy: 0.2571
Training with parameters: {'batch_size': 16, 'epochs': 10, 'learning_rate': 0.0001, 'momentum': 0.9}
Epoch 1/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1264s 13s/step - accuracy: 0.2165 - loss: 1.6095 - val_accuracy: 0.2472 - val_loss: 1.5756
Epoch 2/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 15s 38ms/step - accuracy: 0.2188 - loss: 1.5956 - val_accuracy: 0.1852 - val_loss: 1.5918
Epoch 3/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1158s 13s/step - accuracy: 0.2542 - loss: 1.5814 - val_accuracy: 0.2472 - val_loss: 1.5484
Epoch 4/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 13s 32ms/step - accuracy: 0.1562 - loss: 1.6433 - val_accuracy: 0.3333 - val_loss: 1.5296
Epoch 5/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1215s 13s/step - accuracy: 0.2938 - loss: 1.5580 - val_accuracy: 0.3253 - val_loss: 1.5142
Validation Accuracy: 0.3253
Training with parameters: {'batch_size': 16, 'epochs': 10, 'learning_rate': 0.0001, 'momentum': 0.95}
Epoch 1/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1265s 13s/step - accuracy: 0.2036 - loss: 1.6238 - val_accuracy: 0.2514 - val_loss: 1.5601
Epoch 2/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 15s 28ms/step - accuracy: 0.3438 - loss: 1.5536 - val_accuracy: 0.2963 - val_loss: 1.5405
Epoch 3/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1162s 12s/step - accuracy: 0.2924 - loss: 1.5553 - val_accuracy: 0.3665 - val_loss: 1.4855
Epoch 4/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 14s 24ms/step - accuracy: 0.5312 - loss: 1.4815 - val_accuracy: 0.3704 - val_loss: 1.4617
Epoch 5/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1145s 12s/step - accuracy: 0.3760 - loss: 1.4821 - val_accuracy: 0.5355 - val_loss: 1.3386
Validation Accuracy: 0.5355
Training with parameters: {'batch_size': 16, 'epochs': 10, 'learning_rate': 0.001, 'momentum': 0.85}
Epoch 1/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1232s 12s/step - accuracy: 0.2603 - loss: 1.5817 - val_accuracy: 0.4460 - val_loss: 1.3454
Epoch 2/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 14s 27ms/step - accuracy: 0.4375 - loss: 1.3953 - val_accuracy: 0.4074 - val_loss: 1.3815
Epoch 3/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 1101s 12s/step - accuracy: 0.5407 - loss: 1.1541 - val_accuracy: 0.4886 - val_loss: 1.3065
Epoch 4/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 72s 699ms/step - accuracy: 0.7500 - loss: 0.6469 - val_accuracy: 0.6667 - val_loss: 1.1061
Epoch 5/10
91/91 ━━━━━━━━━━━━━━━━━━━━ 0s 11s/step - accuracy: 0.7535 - loss: 0.6485
```