1. Importovanje potrebnih biblioteka za rad i instaliranje paketa

In [4]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [15]:
```
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
```

1. Putanja do dataset-a i priprema podataka

In [16]:
```
# 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'
)
```

```
Found 2939 images belonging to 5 classes.
Found 731 images belonging to 5 classes.
```

1. NASNetMobile, zamrzavanje pretreniranog sloja i priprema modela

In [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()
```

Model: "functional_4"

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 [20]:
```
# 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
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
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
```

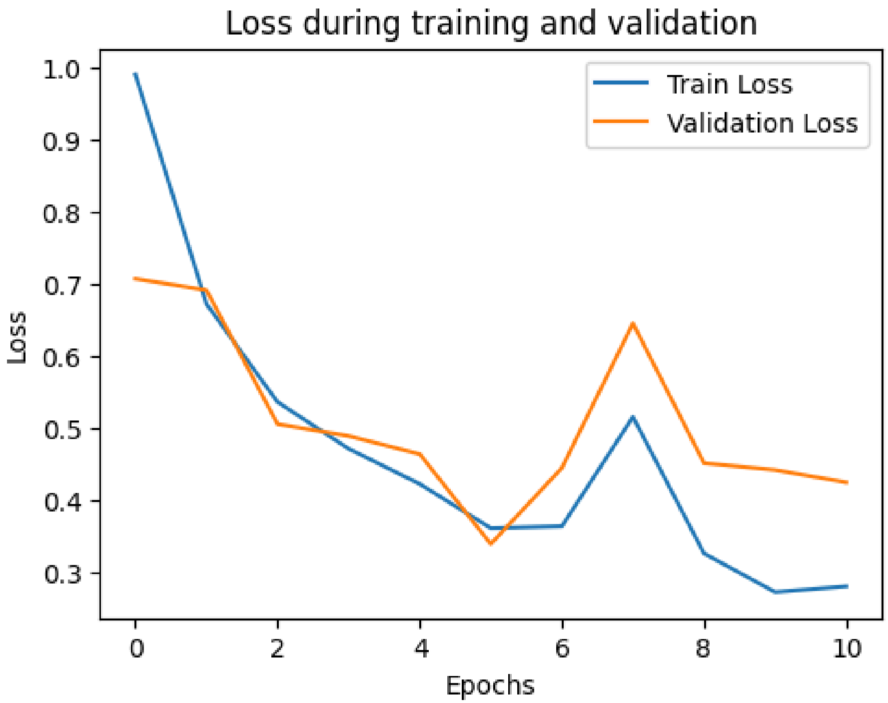
1. Evaluacija modela i graficki prikaz loss i accuracy-ja

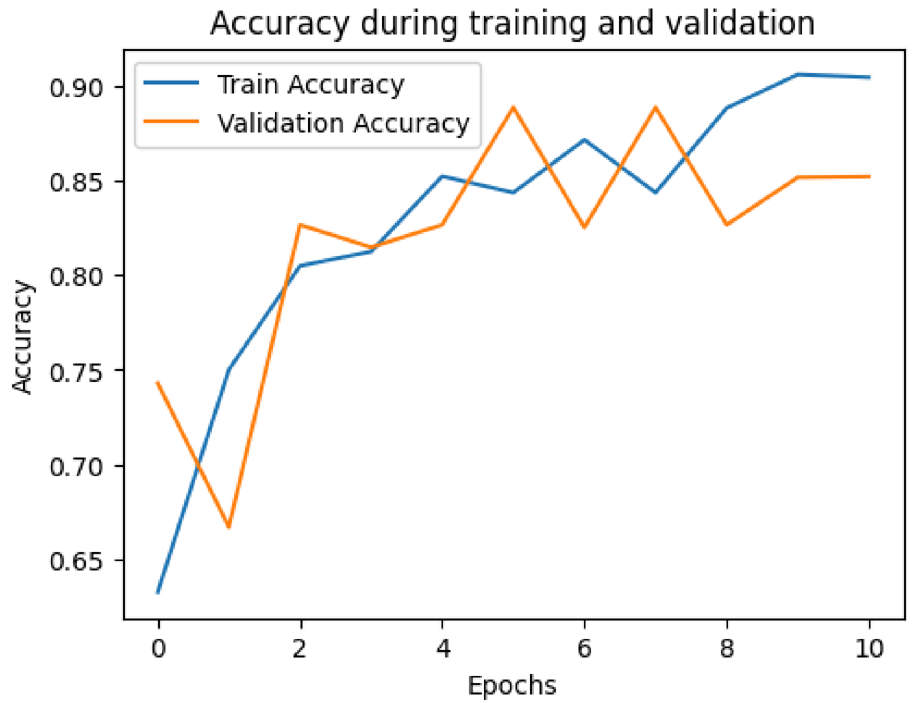
In [21]:
```
# Loss tokom epoha
plt.figure(figsize=(12, 4))
```

```
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 [22]:
```
# 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 ━━━━━━━━━━━━━━━━━━━━ 84s 4s/step - accuracy: 0.8033 - loss: 0.4940
```

```
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]]
```

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

In [12]:
```python
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 ━━━━━━━━━━━━━━━━━━━━ 7s 7s/step
Model predvidja da je ova slika: daisy
```

9.1. Isprobavanje na 16 random slika iz dataset-a

In [23]:
```
# 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 ━━━━━━━━━━━━━━━━━━━━ 9s 9s/step
```

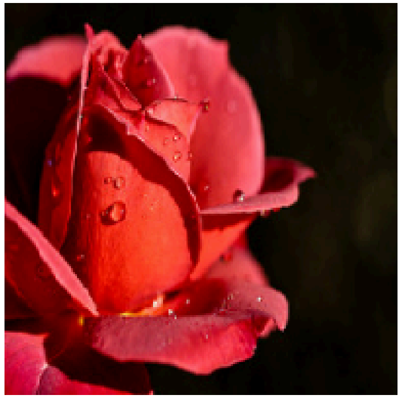
Predicted class: tulips
Real class: tulips

Predicted class: dandelion
Real class: dandelion

Predicted class: dandelion
Real class: dandelion

Predicted class: tulips
Real class: tulips

Predicted class: roses
Real class: roses



Predicted class: roses
Real class: roses

Predicted class: daisy
Real class: daisy

Predicted class: roses
Real class: roses

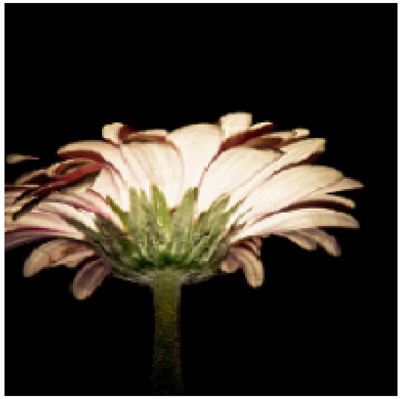
Predicted class: roses
Real class: roses

Predicted class: daisy
Real class: daisy

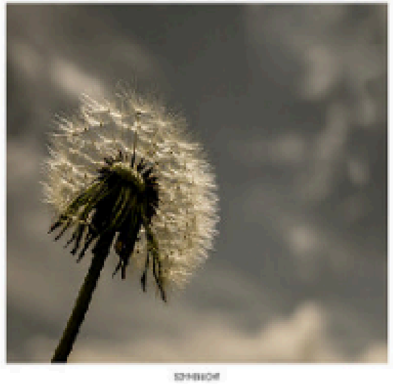
Predicted class: sunflower
Real class: sunflower

Predicted class: daisy
Real class: daisy

Predicted class: daisy
Real class: daisy



Predicted class: dandelion
Real class: dandelion



Predicted class: tulips
Real class: tulips

Predicted class: roses
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
```