Assignment 7

Image Classification Using CNN

Name : Athavna Ramgirkar
Registration Number: 19BCE0114
Submission Date : 13 July, 2021
Program : VIT-AI Industry Certifiation
Email : athavna.ramgirkar2019@vitstudent.ac.in

Other Assignments can be found in the link:
https://drive.google.com/drive/folders/1QGOLHyZyIxqj_CroTJu6-YkZWf3ZJZ-QH?usp=sharing

In [ ]:

Table of Content

- Importing Libraries
- Setting Image Modification Parameters
- Reading Images
- Model Building
  - Model Initialization
  - Convolution Layer
  - Pooling
  - Flatten(Input Layer))
  - Hidden Layers
  - Output Layer
- Compiling the Model
- Model Training and Testing
- Single Predictions
- Saving the Model
- Loading the Saved Model
- Making Predictions Using Loaded Model

In [ ]:

In [ ]:

In [ ]:

1. Importing Libraries

In [1]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Convolution2D,MaxPooling2D,Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import ImageFile
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

In [2]:
```
ImageFile.LOAD_TRUNCATED_IMAGES = True
```

In [3]:
```
import numpy as np
```

2. Setting Image Modification Parameters

Back to Top

In [4]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range=0.3,
                                   zoom_range=0.1,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

In [ ]:

In [ ]:

3. Reading Images

Back to Top

In [5]:
```
X_train = train_datagen.flow_from_directory("seg_train/seg_train/",
                                            target_size=(64,64),
                                            batch_size=32,
                                            class_mode="categorical")
X_test = test_datagen.flow_from_directory("seg_test/seg_test/",
                                          target_size=(64,64),
                                          batch_size=32,
                                          class_mode="categorical")
```

```
Found 5767 images belonging to 6 classes.
Found 1145 images belonging to 6 classes.
```

In [6]:
```
X_train.class_indices
```

Out[6]:
```
{'buildings': 0,
 'forest': 1,
 'glacier': 2,
 'mountain': 3,
 'sea': 4,
 'street': 5}
```

In [ ]:

4. Model Building

Back to Top

4.1 Model Initialization

In [ ]:
```
model = Sequential()
```

4.2 Convolution Layer

In [ ]:
```
model.add(Convolution2D(50,(5,5),input_shape=(64,64,3)))
```

4.3 Pooling

In [ ]:
```
model.add(MaxPooling2D((2,2)))
```

4.4 Flatten(Input Layer)

In [ ]:
```
model.add(Flatten())
```

4.5 Hidden Layers

In [ ]:
```
model.add(Dense(units = 100,
                kernel_initializer="random_uniform",
                activation="relu"))
```

In [ ]:
```
model.add(Dense(units = 100,
                kernel_initializer="random_uniform",
                activation="relu"))
```

4.6 Output Layer

In [ ]:
```
model.add(Dense(units = 6,
                kernel_initializer="random_uniform",
                activation="softmax"))
```

5. Compiling the Model

Back to Top

In [ ]:
```
model.compile(optimizer="adam",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

In [ ]:

6. Model Training and Testing

Back to Top

In [ ]:
```
model.fit_generator(X_train,
                    steps_per_epoch=116,
                    epochs=25,
                    validation_data=X_test,
                    validation_steps=35)
```

```
Epoch 1/25
116/116 [==============================] - 20s 173ms/step - loss: 1.3004 - accuracy: 0.4949 - val_loss: 1.2314 - val_accuracy: 0.5400
Epoch 2/25
116/116 [==============================] - 19s 168ms/step - loss: 1.0863 - accuracy: 0.5872 - val_loss: 0.9958 - val_accuracy: 0.6137
Epoch 3/25
116/116 [==============================] - 18s 159ms/step - loss: 1.0050 - accuracy: 0.6161 - val_loss: 1.2439 - val_accuracy: 0.5450
Epoch 4/25
116/116 [==============================] - 18s 157ms/step - loss: 0.9875 - accuracy: 0.6176 - val_loss: 0.9409 - val_accuracy: 0.6500
Epoch 5/25
116/116 [==============================] - 18s 159ms/step - loss: 0.9032 - accuracy: 0.6629 - val_loss: 1.1671 - val_accuracy: 0.5850
Epoch 6/25
116/116 [==============================] - 19s 161ms/step - loss: 0.8924 - accuracy: 0.6566 - val_loss: 0.9853 - val_accuracy: 0.6575
Epoch 7/25
116/116 [==============================] - 18s 159ms/step - loss: 0.8342 - accuracy: 0.6778 - val_loss: 0.9717 - val_accuracy: 0.6550
Epoch 8/25
116/116 [==============================] - 19s 167ms/step - loss: 0.7964 - accuracy: 0.6959 - val_loss: 0.8588 - val_accuracy: 0.7013
Epoch 9/25
116/116 [==============================] - 18s 154ms/step - loss: 0.8041 - accuracy: 0.6931 - val_loss: 0.9023 - val_accuracy: 0.6812
Epoch 10/25
116/116 [==============================] - 18s 156ms/step - loss: 0.7497 - accuracy: 0.7155 - val_loss: 0.8768 - val_accuracy: 0.7000
Epoch 11/25
116/116 [==============================] - 19s 165ms/step - loss: 0.7330 - accuracy: 0.7338 - val_loss: 1.1328 - val_accuracy: 0.5987
Epoch 12/25
116/116 [==============================] - 19s 165ms/step - loss: 0.6707 - accuracy: 0.7497 - val_loss: 0.9689 - val_accuracy: 0.6600
Epoch 13/25
116/116 [==============================] - 19s 162ms/step - loss: 0.6642 - accuracy: 0.7581 - val_loss: 0.9857 - val_accuracy: 0.6650
Epoch 14/25
116/116 [==============================] - 19s 161ms/step - loss: 0.6245 - accuracy: 0.7716 - val_loss: 1.0678 - val_accuracy: 0.6637
Epoch 15/25
116/116 [==============================] - 19s 165ms/step - loss: 0.5944 - accuracy: 0.7762 - val_loss: 1.0719 - val_accuracy: 0.6762
Epoch 16/25
116/116 [==============================] - 19s 160ms/step - loss: 0.5609 - accuracy: 0.7958 - val_loss: 1.0022 - val_accuracy: 0.6900
Epoch 17/25
116/116 [==============================] - 19s 168ms/step - loss: 0.5597 - accuracy: 0.7809 - val_loss: 1.0306 - val_accuracy: 0.6875
Epoch 18/25
116/116 [==============================] - 20s 170ms/step - loss: 0.5453 - accuracy: 0.7955 - val_loss: 0.9202 - val_accuracy: 0.6875
Epoch 19/25
116/116 [==============================] - 19s 164ms/step - loss: 0.5153 - accuracy: 0.8160 - val_loss: 1.0507 - val_accuracy: 0.6875
Epoch 20/25
116/116 [==============================] - 19s 162ms/step - loss: 0.5207 - accuracy: 0.8006 - val_loss: 0.8912 - val_accuracy: 0.7225
Epoch 21/25
116/116 [==============================] - 19s 164ms/step - loss: 0.4656 - accuracy: 0.8196 - val_loss: 1.2439 - val_accuracy: 0.6550
Epoch 22/25
116/116 [==============================] - 19s 165ms/step - loss: 0.4615 - accuracy: 0.8283 - val_loss: 1.0876 - val_accuracy: 0.6725
Epoch 23/25
116/116 [==============================] - 20s 170ms/step - loss: 0.4373 - accuracy: 0.8421 - val_loss: 1.1787 - val_accuracy: 0.7088
Epoch 24/25
116/116 [==============================] - 19s 165ms/step - loss: 0.4153 - accuracy: 0.8421 - val_loss: 1.1660 - val_accuracy: 0.6712
Epoch 25/25
116/116 [==============================] - 19s 166ms/step - loss: 0.4394 - accuracy: 0.8346 - val_loss: 1.0602 - val_accuracy: 0.6787
```

Out[24]:
```
<tensorflow.python.keras.callbacks.History at 0x2df2e138fd0>
```

In [ ]:

7. Single Predictions

Back to Top

In [28]:
```
img = image.load_img("seg_test/seg_test/buildings/22421.jpg",target_size=(64,64))
```

In [30]:
```
np.argmax(model.predict(np.expand_dims(img,axis=0)))
```

Out[30]:
```
0
```

In [31]:
```
img = image.load_img("seg_test/seg_test/forest/22854.jpg",target_size=(64,64))
```

In [32]:
```
np.argmax(model.predict(np.expand_dims(img,axis=0)))
```

Out[32]:
```
1
```

In [33]:
```
img = image.load_img("seg_test/seg_test/mountain/22537.jpg",target_size=(64,64))
```

In [34]:
```
np.argmax(model.predict(np.expand_dims(img,axis=0)))
```

Out[34]:
```
3
```

All three single predictions are correct

In [ ]:

8. Saving the Model

Back to Top

In [35]:
```
model.save("nature.h5")
```

In [ ]:

9. Loading the Saved Model

Back to Top

In [36]:
```
model_load = load_model("nature.h5")
```

In [ ]:

10. Making Predictions Using Loaded Model

Back to Top

In [39]:
```
np.argmax(model_load.predict(np.expand_dims(img,axis=0)))
```

Out[39]:
```
5
```

In [38]:
```
img = image.load_img("seg_test/seg_test/street/23253.jpg",target_size=(64,64))
```

In [40]:
```
np.argmax(model_load.predict(np.expand_dims(img,axis=0)))
```

Out[40]:
```
5
```

In [41]:
```
img = image.load_img("seg_test/seg_test/sea/22736.jpg",target_size=(64,64))
```

In [43]:
```
np.argmax(model_load.predict(np.expand_dims(img,axis=0)))
```

Out[43]:
```
4
```

2 out of 3 predictions are correct from the Loaded Model

In [ ]:

In [ ]:

In [ ]: