Arab Republic of Egypt
Ministry of Communications
and Information Technology

Object Detection Using CNN

21/12/2021

إبراهيم أيمن
خلود إبراهيم
نوران هشام
رحمه محمد
عمر حماد

Abstract

The field of machine learning has taken a dramatic twist in recent times, with the rise of the Artificial Neural Network (ANN). These biologically inspired computational models are able to far exceed the performance of previous forms of artificial intelligence in common machine learning tasks. One of the most impressive forms of ANN architecture is that of the Convolutional Neural Network (CNN). CNNs are primarily used to solve difficult image-driven pattern recognition tasks and with their precise yet simple architecture, offers a simplified method of getting started with ANNs. This document provides a brief introduction to CNNs, discussing recently published papers and newly formed techniques in developing these brilliantly fantastic image recognition models. This introduction assumes you are familiar with the fundamentals of ANNs and machine learning.

Introduction

VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR(Imagenet) competition in 2014. It is considered to be one of the excellent vision model architecture till date. Most unique thing about VGG16 is that instead of having a large number of hyper-parameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx) parameters.

Table Of Contents¶

1. Loading and preprocess Object dataset¶
2. Designing and training a CNN model in Keras¶
3. Plotting the Loss and Accuracy curve¶
4. Evaluating the model & Predicting the output class of a test Object¶
5. Visualizing the intermediate layer output of CNN¶
6. Plotting the confusion matrix for your result¶

1. Loading and preprocessing own dataset¶

The dataset that I am using for this kernel is my own accumulated dataset of 7 types of classes namely 'flowers', 'cars', 'cats', 'horses', 'human', 'bike', 'dogs' with total of 1803 image samples.

Model analysis with dataset:

VGG16 Model:

Code summary

```
!kaggle datasets download -d pavansanagapati/images-dataset
```

```
! unzip images-dataset
```

```
# Define data path

data_dir_list = ['bike', 'cars','cats','dogs','flowers','horses','human']
data_dir_list

img_rows=128
img_cols=128
num_channel=1
num_epoch=100
# Define the number of classes
num_classes = 7
img_data_list=[]
for dataset in data_dir_list:
  img_list=os.listdir("/content/drive/MyDrive/detection/data"+'/'+ dataset)
  print ('Loaded the images of dataset-'+'{}\n'.format(dataset))
  for img in img_list:
    input_img=cv2.imread("/content/drive/MyDrive/detection/data" + '/'+ dataset + '/'+ img )
    input_img=cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
    input_img_resize=cv2.resize(input_img,(128,128))
    img_data_list.append(input_img_resize)

img_data = np.array(img_data_list)
img_data = img_data.astype('float32')
img_data /= 255
print (img_data.shape)
```

```
if num_channel==1:
  if K.set_image_data_format=='channels_first':
    img_data= np.expand_dims(img_data, axis=1)
    print (img_data.shape)
  else:
    img_data= np.expand_dims(img_data, axis=3)
    print (img_data.shape)

else:
  if K.image_dim_ordering()=='channels_first':
    img_data=np.rollaxis(img_data,3,1)
    print (img_data.shape)
```

```
num_classes = 7
num_of_samples = img_data.shape[0]
labels = np.ones((num_of_samples,),dtype='int64')
labels[0:365]=0
```

```
labels[0:365]=0
labels[365:567]=1
labels[567:987]=2
labels[987:1189]=3
labels[1189:1399]=4
labels[1399:1601]=5
labels[1601:1803]=6
names = ['bike', 'cars', 'cats', 'dogs', 'flowers', 'horses', 'human']
```

```
Y = np_utils.to_categorical(labels, num_classes)
```

```
x,y = shuffle(img_data,Y, random_state=2)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2)
```

```
print("X_train shape = {}".format(X_train.shape))
print("X_test shape = {}".format(X_test.shape))
```

```
image = X_train[1203,:].reshape((128,128))
plt.imshow(image)
plt.show()
```

```
model = Sequential()
model.add(Conv2D(64, kernel_size=(3, 3),
    strides=(1,1),
    padding="same",
    activation='relu',
    input_shape=(128,128,1)))

model.add(Conv2D(64, (3, 3),strides=(1,1),padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),padding="same"))
model.add(Conv2D(128, (3, 3),strides=(1,1),padding="same", activation='relu'))
model.add(Conv2D(128, (3, 3),strides=(1,1),padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),padding="same"))
model.add(Conv2D(256, (3, 3),strides=(1,1),padding="same", activation='relu'))
model.add(Conv2D(256, (3, 3),strides=(1,1),padding="same", activation='relu'))
model.add(Conv2D(256, (3, 3),strides=(1,1),padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),padding="same"))

model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
```

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

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_28 (Conv2D)          (None, 64, 128, 1)        73792     
                                                                 
 conv2d_29 (Conv2D)          (None, 64, 128, 1)        36928     
                                                                 
 max_pooling2d_11 (MaxPoolin  (None, 64, 64, 1)        0         
 g2D)                                                            
                                                                 
 conv2d_30 (Conv2D)          (None, 128, 64, 1)        73856     
                                                                 
 conv2d_31 (Conv2D)          (None, 128, 64, 1)        147584    
                                                                 
 max_pooling2d_12 (MaxPoolin  (None, 128, 32, 1)       0         
 g2D)                                                            
                                                                 
 conv2d_32 (Conv2D)          (None, 256, 32, 1)        295168    
                                                                 
 conv2d_33 (Conv2D)          (None, 256, 32, 1)        590080    
                                                                 
 conv2d_34 (Conv2D)          (None, 256, 32, 1)        590080    
                                                                 
 max_pooling2d_13 (MaxPoolin  (None, 256, 16, 1)       0         
 g2D)                                                            
                                                                 
 flatten_3 (Flatten)         (None, 4096)              0         
                                                                 
 dropout_8 (Dropout)         (None, 4096)              0         
                                                                 
 dense_8 (Dense)             (None, 1024)              4195328   
                                                                 
 dropout_9 (Dropout)         (None, 1024)              0         
                                                                 
 dense_9 (Dense)             (None, 256)               262400    
                                                                 
 dropout_10 (Dropout)        (None, 256)               0         
                                                                 
 dense_10 (Dense)            (None, 7)                 1799      
                                                                 
=================================================================
Total params: 6,267,015
Trainable params: 6,267,015
Non-trainable params: 0
_________________________________________________________________
```

First model with:

In this model, the train and test are implemented at 1400 photos the hyper parameter of that model is optimizer = **adam**, loss function = categorical_crossentropy , DropOut = 0.4 , epoch = 100 , patch Size = 32

Model summary

```
hist = model.fit(X_train, y_train, batch_size=32, epochs=100 ,verbose=1, validation_data=(X_test, y_test))
```

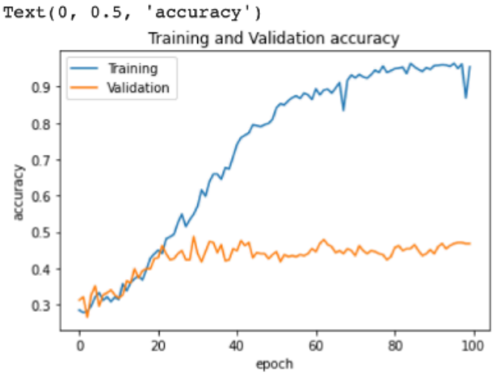
```
Epoch 31/100
46/46 [==============================] - 1s 27ms/step - loss: 1.0855 - accuracy: 0.5714 - val_loss: 1.4471 - val_accuracy: 0.4404
Epoch 32/100
46/46 [==============================] - 1s 27ms/step - loss: 0.9643 - accuracy: 0.6158 - val_loss: 1.5860 - val_accuracy: 0.4183
Epoch 33/100
46/46 [==============================] - 1s 27ms/step - loss: 0.9951 - accuracy: 0.5985 - val_loss: 1.4813 - val_accuracy: 0.4460
Epoch 34/100
46/46 [==============================] - 1s 26ms/step - loss: 0.9019 - accuracy: 0.6387 - val_loss: 1.5893 - val_accuracy: 0.4737
Epoch 35/100
46/46 [==============================] - 1s 26ms/step - loss: 0.8533 - accuracy: 0.6595 - val_loss: 1.7128 - val_accuracy: 0.4709
Epoch 36/100
46/46 [==============================] - 1s 26ms/step - loss: 0.8437 - accuracy: 0.6595 - val_loss: 1.7252 - val_accuracy: 0.4432
Epoch 37/100
46/46 [==============================] - 1s 26ms/step - loss: 0.9215 - accuracy: 0.6449 - val_loss: 1.6777 - val_accuracy: 0.4654
Epoch 38/100
46/46 [==============================] - 1s 26ms/step - loss: 0.8294 - accuracy: 0.6768 - val_loss: 1.8784 - val_accuracy: 0.4211
Epoch 39/100
46/46 [==============================] - 1s 26ms/step - loss: 0.8161 - accuracy: 0.6734 - val_loss: 1.7144 - val_accuracy: 0.4238
Epoch 40/100
46/46 [==============================] - 1s 27ms/step - loss: 0.7500 - accuracy: 0.7074 - val_loss: 1.7631 - val_accuracy: 0.4543
Epoch 41/100
46/46 [==============================] - 1s 27ms/step - loss: 0.6898 - accuracy: 0.7413 - val_loss: 1.7855 - val_accuracy: 0.4488
Epoch 42/100
46/46 [==============================] - 1s 26ms/step - loss: 0.6815 - accuracy: 0.7594 - val_loss: 1.8128 - val_accuracy: 0.4765
Epoch 43/100
46/46 [==============================] - 1s 26ms/step - loss: 0.6191 - accuracy: 0.7663 - val_loss: 2.0215 - val_accuracy: 0.4626
Epoch 44/100
46/46 [==============================] - 1s 27ms/step - loss: 0.5894 - accuracy: 0.7725 - val_loss: 1.9505 - val_accuracy: 0.4709
Epoch 45/100
46/46 [==============================] - 1s 27ms/step - loss: 0.5301 - accuracy: 0.7947 - val_loss: 2.2270 - val_accuracy: 0.4294
Epoch 46/100
46/46 [==============================] - 1s 27ms/step - loss: 0.5585 - accuracy: 0.7926 - val_loss: 1.8255 - val_accuracy: 0.4432
Epoch 47/100
46/46 [==============================] - 1s 27ms/step - loss: 0.5877 - accuracy: 0.7899 - val_loss: 1.9574 - val_accuracy: 0.4404
Epoch 48/100
46/46 [==============================] - 1s 27ms/step - loss: 0.5183 - accuracy: 0.7954 - val_loss: 1.9513 - val_accuracy: 0.4404
Epoch 49/100
46/46 [==============================] - 1s 26ms/step - loss: 0.5012 - accuracy: 0.7989 - val_loss: 2.1366 - val_accuracy: 0.4266
Epoch 50/100
46/46 [==============================] - 1s 26ms/step - loss: 0.4716 - accuracy: 0.8100 - val_loss: 2.0981 - val_accuracy: 0.4377
```

```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation losses')
plt.xlabel('epoch')
plt.ylabel('Loss')
# plt.ylim([0,1])
```

```
Text(0, 0.5, 'Loss')
```


```
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
# plt.ylim([0,1])
```

```
Text(0, 0.5, 'accuracy')
```



Second model with:

In this model, the train and test are implemented at 1400 photos the hyper parameter of that model is optimizer = **SGD**, loss function = categorical_crossentropy , DropOut = 0.4 , epoch = 100 , patch Size = 32

Model summary

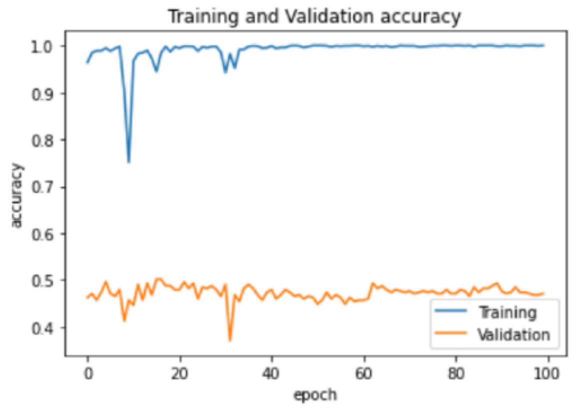
```
Epoch 83/100
46/46 [==============================] - 1s 30ms/step - loss: 0.0016 - accuracy: 0.9993 - val_loss: 7.6452 - val_accuracy: 0.4765
Epoch 84/100
46/46 [==============================] - 1s 31ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 7.5144 - val_accuracy: 0.4654
Epoch 85/100
46/46 [==============================] - 2s 35ms/step - loss: 0.0043 - accuracy: 0.9979 - val_loss: 7.4773 - val_accuracy: 0.4848
Epoch 86/100
46/46 [==============================] - 1s 30ms/step - loss: 9.2919e-04 - accuracy: 1.0000 - val_loss: 7.2070 - val_accuracy: 0.4737
Epoch 87/100
46/46 [==============================] - 1s 31ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 7.3919 - val_accuracy: 0.4820
Epoch 88/100
46/46 [==============================] - 1s 31ms/step - loss: 9.8525e-04 - accuracy: 1.0000 - val_loss: 7.3419 - val_accuracy: 0.4820
Epoch 89/100
46/46 [==============================] - 2s 36ms/step - loss: 9.9213e-04 - accuracy: 1.0000 - val_loss: 7.5570 - val_accuracy: 0.4875
Epoch 90/100
46/46 [==============================] - 1s 31ms/step - loss: 0.0030 - accuracy: 0.9986 - val_loss: 7.2615 - val_accuracy: 0.4931
Epoch 91/100
46/46 [==============================] - 1s 30ms/step - loss: 0.0037 - accuracy: 0.9986 - val_loss: 6.7752 - val_accuracy: 0.4765
Epoch 92/100
46/46 [==============================] - 1s 30ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 6.9094 - val_accuracy: 0.4709
Epoch 93/100
46/46 [==============================] - 1s 30ms/step - loss: 0.0012 - accuracy: 0.9993 - val_loss: 7.1234 - val_accuracy: 0.4737
Epoch 94/100
46/46 [==============================] - 1s 30ms/step - loss: 0.0017 - accuracy: 0.9993 - val_loss: 7.1996 - val_accuracy: 0.4848
Epoch 95/100
46/46 [==============================] - 1s 30ms/step - loss: 0.0025 - accuracy: 0.9986 - val_loss: 7.0334 - val_accuracy: 0.4737
Epoch 96/100
46/46 [==============================] - 1s 30ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 7.3062 - val_accuracy: 0.4737
Epoch 97/100
46/46 [==============================] - 1s 30ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 7.4083 - val_accuracy: 0.4709
Epoch 98/100
46/46 [==============================] - 1s 31ms/step - loss: 4.8859e-04 - accuracy: 1.0000 - val_loss: 7.4585 - val_accuracy: 0.4681
Epoch 99/100
46/46 [==============================] - 1s 31ms/step - loss: 0.0028 - accuracy: 0.9993 - val_loss: 7.5774 - val_accuracy: 0.4681
Epoch 100/100
46/46 [==============================] - 1s 31ms/step - loss: 5.3052e-04 - accuracy: 1.0000 - val_loss: 7.6402 - val_accuracy: 0.4709
```

```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation losses')
plt.xlabel('epoch')
plt.ylabel('Loss')
# plt.ylim([0,1])
```

```
Text(0, 0.5, 'Loss')
```


```
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
# plt.ylim([0,1])
```

```
Text(0, 0.5, 'accuracy')
```



Third model with:

In this model, the train and test are implemented at 1400 photos the hyper parameter of that model is optimizer = **RMSprop**, loss function = categorical_crossentropy , DropOut = 0.4 , epoch = 100 , patch Size = 32

Model summary

```
Epoch 87/100
46/46 [==============================] - 1s 18ms/step - loss: 0.5142 - accuracy: 0.8766 - val_loss: 3.8128 - val_accuracy: 0.4958
Epoch 88/100
46/46 [==============================] - 1s 18ms/step - loss: 0.3725 - accuracy: 0.8904 - val_loss: 3.5388 - val_accuracy: 0.4931
Epoch 89/100
46/46 [==============================] - 1s 18ms/step - loss: 0.3972 - accuracy: 0.8759 - val_loss: 3.7517 - val_accuracy: 0.4321
Epoch 90/100
46/46 [==============================] - 1s 18ms/step - loss: 0.5239 - accuracy: 0.8481 - val_loss: 3.3796 - val_accuracy: 0.4681
Epoch 91/100
46/46 [==============================] - 1s 18ms/step - loss: 0.3984 - accuracy: 0.8953 - val_loss: 2.5853 - val_accuracy: 0.5069
Epoch 92/100
46/46 [==============================] - 1s 18ms/step - loss: 0.3895 - accuracy: 0.8883 - val_loss: 2.6628 - val_accuracy: 0.5125
Epoch 93/100
46/46 [==============================] - 1s 16ms/step - loss: 0.3878 - accuracy: 0.8682 - val_loss: 4.0040 - val_accuracy: 0.5014
Epoch 94/100
46/46 [==============================] - 1s 16ms/step - loss: 0.3632 - accuracy: 0.8939 - val_loss: 3.4505 - val_accuracy: 0.4349
Epoch 95/100
46/46 [==============================] - 1s 16ms/step - loss: 0.4163 - accuracy: 0.8724 - val_loss: 3.7567 - val_accuracy: 0.5014
Epoch 96/100
46/46 [==============================] - 1s 16ms/step - loss: 0.4030 - accuracy: 0.8814 - val_loss: 6.1181 - val_accuracy: 0.4072
Epoch 97/100
46/46 [==============================] - 1s 16ms/step - loss: 0.3974 - accuracy: 0.8849 - val_loss: 4.3367 - val_accuracy: 0.4931
Epoch 98/100
46/46 [==============================] - 1s 18ms/step - loss: 0.3809 - accuracy: 0.8932 - val_loss: 3.1830 - val_accuracy: 0.4931
Epoch 99/100
46/46 [==============================] - 1s 16ms/step - loss: 0.3835 - accuracy: 0.8883 - val_loss: 3.2862 - val_accuracy: 0.5042
Epoch 100/100
46/46 [==============================] - 1s 18ms/step - loss: 0.5012 - accuracy: 0.8675 - val_loss: 4.1007 - val_accuracy: 0.5180
```

```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation losses')
plt.xlabel('epoch')
plt.ylabel('Loss')
# plt.ylim([0,1])
```

```
Text(0, 0.5, 'Loss')
```


```
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
# plt.ylim([0,1])
```

```
Text(0, 0.5, 'accuracy')
```