ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – TP HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN

REPORT

CSC14003 – AI PROJECT

Object classification using Neural networks

GROUP MEMBERS

Đinh Phi Long – 18127263

Trần Quốc Tuấn – 18127246

LECTURERS

Mr. Le Ngoc Thanh

Mrs. Ho Thi Thanh Tuyen

Ms. Phan Thi Phuong Uyen

Thành phố Hồ Chí Minh 2020

I. INTRODUCTION

1. ABSTRACT

This project is aimed at implementing an artificial neural network, either traditional networks or deep networks, for object classification on some prepared datasets. The approached problem which we are interested in is image classification. This task is quite easy for human but it is a bit more tricky for computers. So that our job is to implement a model to help our computers classify what one image contains.

Our group can separate this problem into 4 subtasks (Overall plan):

- Preparing datasets for training and testing.
- Building the neural network model.
- Training data.
- Classify images.

2. ENVIROMENT

Language: Python 3.7+

Environment Software: Visual Studio Code, PyCharm

OS Build: Window 10 (64 bits)

Libraries: keras, cv2 from opencv, numpy, PIL and matplotlib

**Note: This project is carried out with tensorflow 2.2 library – which is necessary for keras*

II. EXPERIMENT

Our group choose to implement advanced CNN to classify whether images contain either a dog or a cat, which needs the Kaggle dataset of cats and dogs (https://www.kaggle.com/c/dogs-vs-cats).

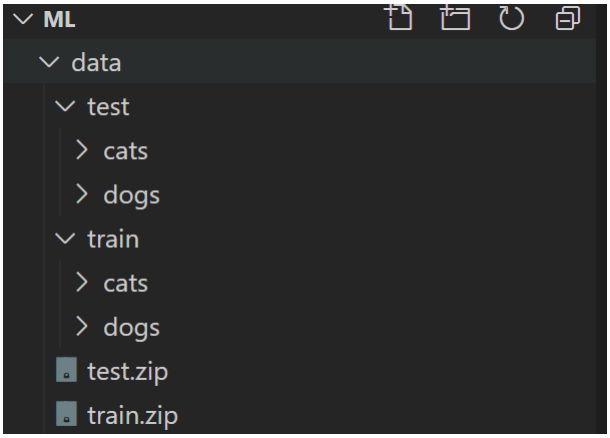
1. Understanding

We will collect enormous number of images containing cat and dog, then build a model and train it with those images. This progress can be considered as having a computer learn how to distinguish cats from dogs. After the training, we will use that model to classify some pictures which are not from the dataset we use for the training. The purpose is to see how accurately that model can predict objects from those pictures.

2. Approach

- **Preparing the dataset**

We will download the dataset from kaggle (link above). There are two main folders: train and test. Inside the train folder, we have two child folders called cats and dogs. Each folder contains 12500 images. Test folder has the same structure with train folder but each subfolder contains only 2500 images.



Our dataset is ready.

- **Building the CNN model**

We need several layers for our model since we are using deep learning. *Conv2D, Activation, MaxPooling2D, Dense, Flatten,* and *Dropout* are different types of layers that are available in *keras* to build our model.

```
model = Sequential()
```

Initialize a convolutional neural network using the sequential model of keras. We are using sequential here to build our model. The sequential API helps us to create models in a layer-by-layer format.

+ Convolutional Layer: we added a convolutional layer as the first layer. Conv2D stands for a 2-dimensional convolutional layer.

```
model.add(Conv2D(32, (3,3) ,input_shape=(64, 64, 3)
```

Here, 32 is the number of filters needed. A filter is an array of numeric values. (3,3) is the size of the filter, which means 3 rows and 3 columns.

The output of this layer will be some feature maps. The training images will go through this layer, and we will obtain some feature maps at the end of this layer

+ Activation Layer: pass the feature maps through an activation layer called ReLU. ReLU stands for the rectified linear unit. ReLU is an activation function. ReLU replaces all the negative pixel values in the feature map with 0.

```
 activation='relu'))
```

+ Pooling Layer: Pooling helps to reduce the dimensionality of each feature map and retains the essential information, decrease the computational complexity of our network.

Here, we used max-pooling with a 2*2 filter. The filter will take the max values from each pool.

```
model.add(MaxPooling2D(pool_size= (2,2)))
```

+ Dropout Layer: To prevent overfitting, we use the dropout layer in our model. Overfitting is a modeling error that occurs to make an overly complex model. This layer drops out a random set of activations in that layer by setting them to zero as data flows through it.

```
model.add(Dropout(0.6))
```

To prepare our model for dropout,we flatten the feature map to 1-dimension.

```
model.add(Flatten())
```

Then we want to initialize a fully connected network by using the Dense function and apply the ReLu activation function to it.

```
model.add(Dense(units=64, activation='relu', kernel_initializer='uniform'))
```

After dropout, we'll initialize 1 more fully connected layer, which is applied a softmax activation, which will convert the data into probabilities for each class

```
model.add(Dense(units=2, activation='softmax'))
```

Last, we compile the model before training it.

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

- **Training data**

  First, we need to do Data augmentation before training. We will import the *ImageDataGenerator* method from the *keras* library. We set these parameters so that the machine will get trained with the images at different positions and details to improve the accuracy.

```
train_datagen = ImageDataGenerator(rescale =1./255,
                                   shear_range =0.1,
                                   zoom_range = 0.1,
                                   horizontal_flip =True)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

  Then, we need to set the train and test directories by *flow_from_directory()* method from Keras.

```
training_set = train_datagen.flow_from_directory('data/train',
                                                 target_size=(64,64),
                                                 batch_size= 32,
                                                 class_mode='categorical')

test_set = test_datagen.flow_from_directory('data/test',
                                            target_size = (64,64),
                                            batch_size = 32,
                                            class_mode ='categorical')
```

  Next, we initializing earlystopping - allows us to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset - and reduce-LR-on-Plateau - will adjust the learning rate when a plateau in model performance is detected.

```
earlystop = EarlyStopping(patience=10)
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=2,
                                            verbose=1,
                                            factor=0.75,
                                            min_lr=0.00005)

callbacks = [earlystop, learning_rate_reduction]
```

The callback is designed to reduce the learning rate after the model stops improving with the hope of fine-tuning model weights.

Now is the training part. We will train the model using the *fit_generator()* method of keras library.

```
FAST_RUN = False
epochs=5 if FAST_RUN else 30

model.fit_generator(training_set,
                    steps_per_epoch=8000//32,
                    epochs=epochs,
                    validation_data=test_set,
                    validation_steps=2000//32,
                    callbacks=callbacks)
```

For different models and differents datasets, the values of *epochs* and *step_per_epoch* are different. As these values become larger, the longer it will take the machine to learn everything. But the longer it takes, the better will be the accuracy.

```
Epoch 26/30
250/250 [==============================] - 73s 290ms/step - loss: 0.4149 - accuracy: 0.8123 - val_loss: 0.3885 - val_ac
curacy: 0.8140 - lr: 3.1641e-04
Epoch 27/30
250/250 [==============================] - ETA: 0s - loss: 0.4187 - accuracy: 0.8079
Epoch 00027: ReduceLROnPlateau reducing learning rate to 0.00023730468819849193.
250/250 [==============================] - 77s 306ms/step - loss: 0.4187 - accuracy: 0.8079 - val_loss: 0.3981 - val_ac
curacy: 0.8216 - lr: 3.1641e-04
Epoch 28/30
250/250 [==============================] - 85s 338ms/step - loss: 0.4153 - accuracy: 0.8117 - val_loss: 0.3662 - val_ac
curacy: 0.8483 - lr: 2.3730e-04
Epoch 29/30
250/250 [==============================] - 85s 341ms/step - loss: 0.3990 - accuracy: 0.8163 - val_loss: 0.3319 - val_ac
curacy: 0.8619 - lr: 2.3730e-04
Epoch 30/30
250/250 [==============================] - 81s 326ms/step - loss: 0.4027 - accuracy: 0.8119 - val_loss: 0.3576 - val_ac
curacy: 0.8387 - lr: 2.3730e-04
```

We got a validation accuracy of 83.87%.

You have patiently trained your model, and now, you must save this model if you want to use this model in the future.

```
model.save('src/catdog_cnn_model.h5')
```

- **Classify image**

  Load model using *load_model()* method from keras.

```
classifier = load_model('src/catdog_cnn_model.h5')
```

  Loading image for predicting, using same method with loading image for training

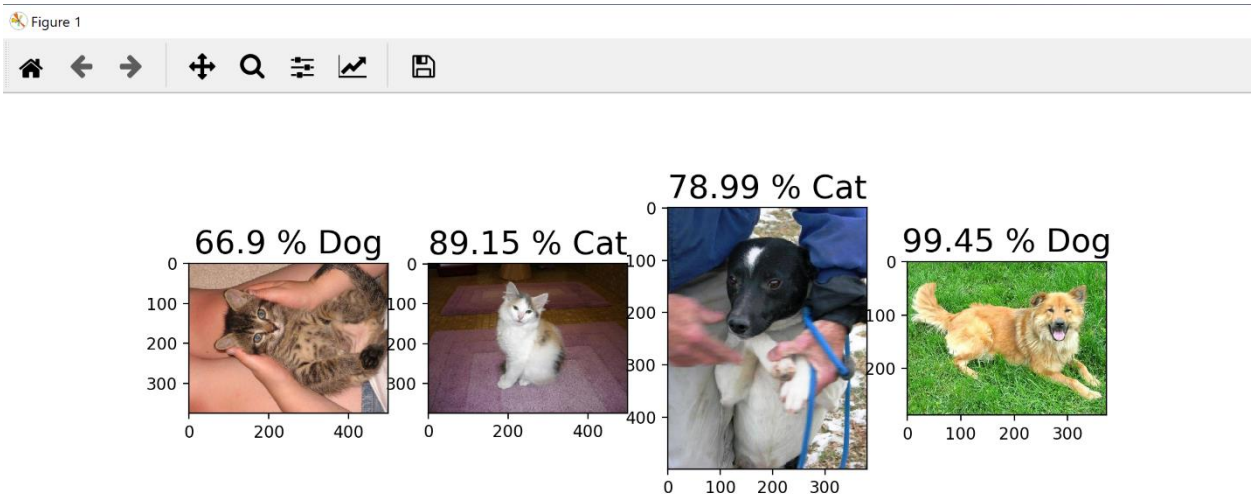
```
pred_datagen = ImageDataGenerator(rescale=1./255)
pred_set = pred_datagen.flow_from_directory('src/predict',
                                            target_size=(64,64),
                                            class_mode='categorical',
                                            shuffle = False)
```

  Predicting image using *predict_generator()* method.

```
pred_prob = classifier.predict_generator(pred_set)
pred_prob = np.round(pred_prob*100,2)
```

  Method np.round just for demonstrating results into percentage values.

3. Result



III. BACKGROUND

During the progress of doing this project, we have widen our knowledge by reading materials about CNN model, image processing and keras library.

Some references:

[1]. https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/

[2]. https://medium.com/@ipaar3/how-i-built-a-convolutional-image-classifier-using-tensorflow-from-scratch-f852c34e1c95

[3]. https://viblo.asia/p/lam-quen-voi-keras-gGJ59mxJ5X2

IV. DEMO PROJECT VIDEO

https://www.youtube.com/watch?v=EK7i4jvvGmA

V. CONCLUSION

We have successfully created an image classifier using deep learning with the keras library of Python. Our image classifier predicted the results with an accuracy of 83.87 percentage.

From this project, we can see that by having learned same objects several times, computers can develop an ability to detect objects from the pictures we test on, even though those pictures are not included in the datasets computers learned. In the future, computers can be trained to classify more and more sofisticated objects, and would have the ability to think, collect information themselves without human's help. For instance, robot with machine learning.

VI. REFERENCES

[1] . AI document of CSC14003 (18CLC1)

[2]. https://medium.com/velacorpblog/l%C3%A0m-quen-v%E1%BB%9Bi-convolutional-neural-network-v%E1%BB%9Bi-b%C3%A0i-to%C3%A1n-nh%E1%BA%ADn-di%E1%BB%87n-%E1%BA%A3nh-ba6af4ed01fb

[3] . https://github.com/Xx-Ashutosh-xX/Cats_vs_Dogs

[4]. https://www.pythonistaplanet.com/image-classification-using-deep-learning/