Animal Recognition Using TensorFlow

Zixuan Yu, Chung-Yang Li
yu.zix@husky.neu.edu li.chun@husky.neu.edu
INFO 7390, Summer 2018, Northeastern University

● Abstract

Animal recognition is a computer technology using machine learning and TensorFlow. This project aims to recognize various animals by applying machine learning and TensorFlow into the real world. By giving sufficient number of datasets, we can train our classified model and further accurately recognize various animals (six species). There are several types of models for CNN, we have test numerous hyper-parameter to reach best performance. These hyper-parameters are important; iteration, learning rate and activation functions, which affect speed and accuracy. In our training process, we only use 600 images (6 species, each species has 100) and could reach approximately 50% accuracy score. We believe that if we have enough training images, our model will come out with a great performance.

● Introduction

In current circumstance, although most zoos have indicators for animals, yet sometimes we found that they look different from the signs. There are many wild zoos in the world, most of the time children are curious about different kinds of animals when it presents in their eyes. Children might frequently ask their parents what it is, but in most cases their parents don't know either. Our animal identifying functionality is designed to solve this problem. The purpose of our project is to help recognizing animals by applying machine learning and TensorFlow technique into the real world. By giving sufficient datasets to train our model, we can accurately recognize various animals. Therefore, if someone use this functionality toward animals, he could further learn more knowledge about the target creature. The paper gets a better accuracy by changing these hyper-parameters. (iteration, learning rate and activation functions)

● Dataset

The dataset contains 600 images which are classified as cat, dog, monkey, bird, lion and horse. We download images one by one, and the resources are ImageNet, google image, and pexels.com.

- **Convolution Neural Network**

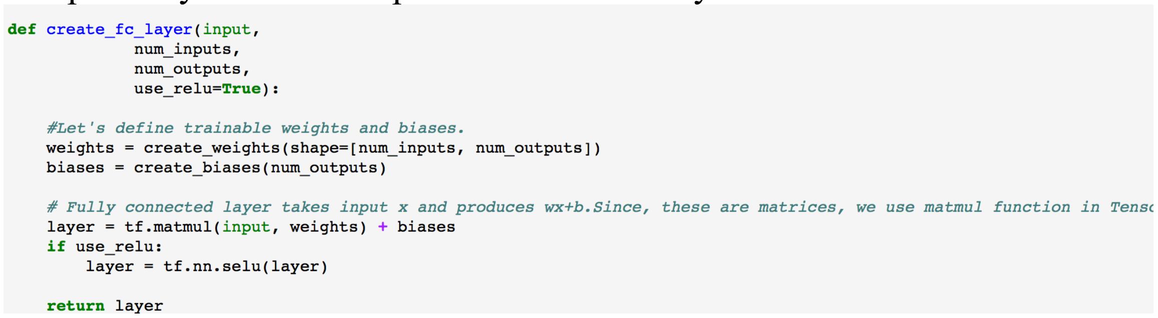
In machine learning, a convolutional neural network is a class of deep, feed-forward artificial neural networks, most commonly applied to analyzing visual imagery.
There are three main types of layers to build CNN architectures:

- **Various model using CNN:**

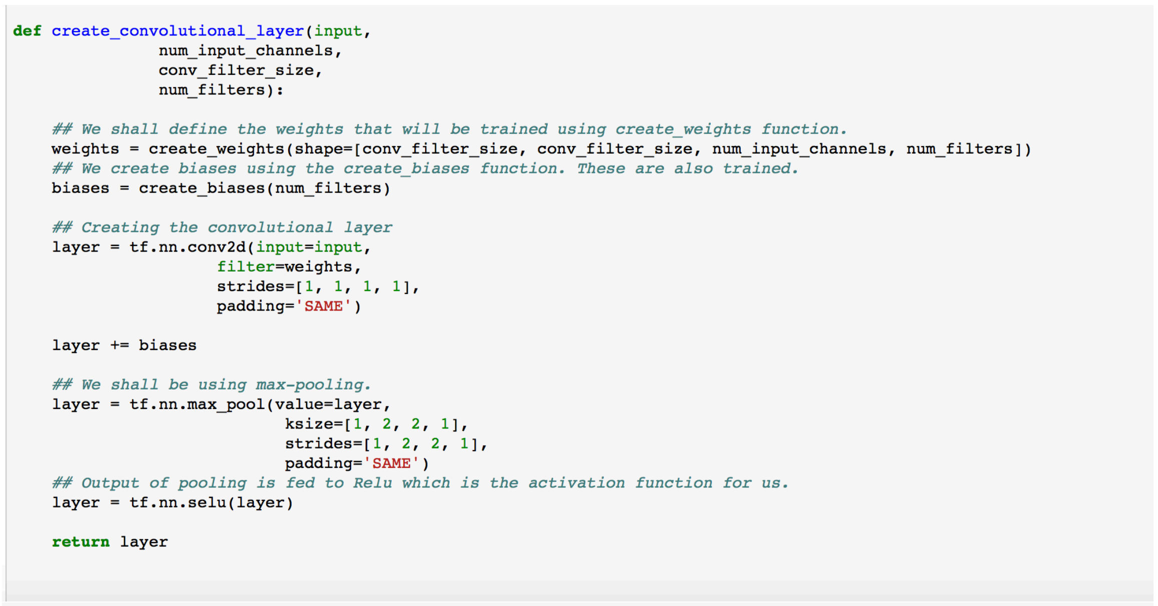
First, we create a convolution layer. Convolution is a mathematical operation which used in single processing to filter signals, find patterns in signals etc. In a convolutional layer, all neurons apply convolution operation to the inputs, hence they are called convolutional neurons. The most important parameter in a convolutional neuron is the filter size, we have three layers with filter size 3*3*32, 3*3*32 and 3*3*64. Here we use the max pooling method to reduce the spatial size. This reduces the number of parameters, and also less number of parameters avoid overfitting.

Second, we create a flatten layer. The Output of a convolutional layer is a multi-dimensional tensor. We want to convert this into a one-dimensional tensor. This is done in the flatten layer.

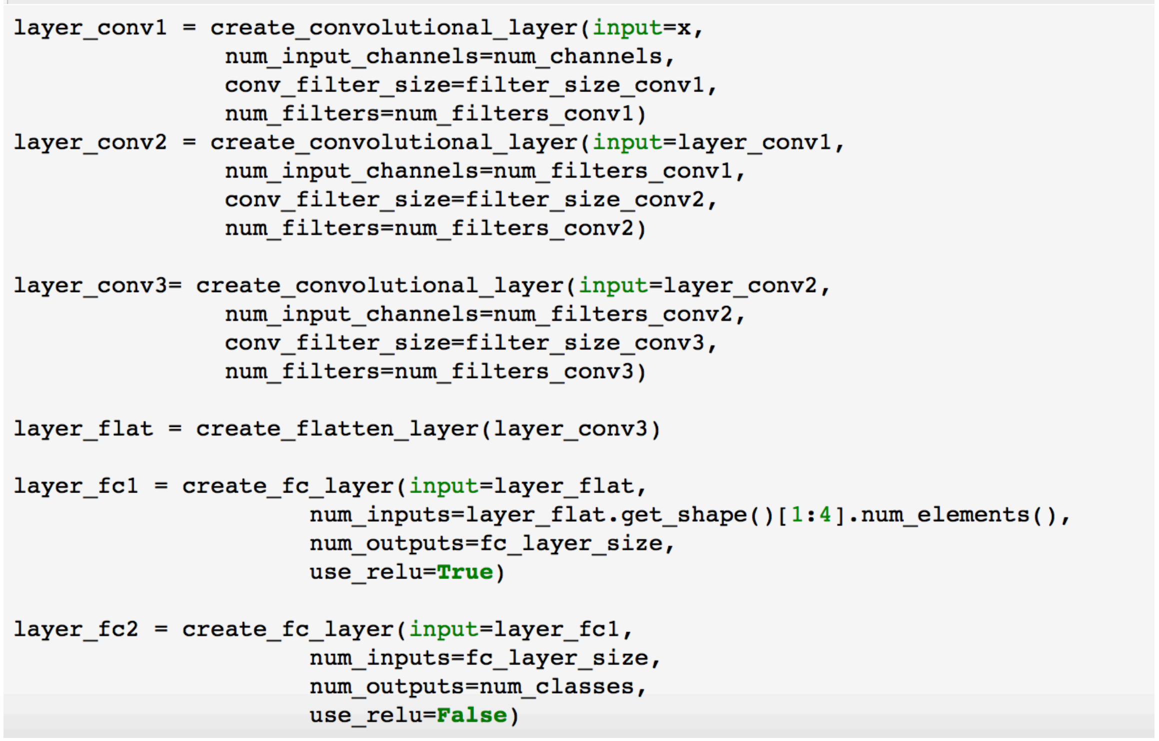
Third, we create a fully connected layer. Each neuron in a layer receives input from all the neurons in the previous layer. The output of this layer is computed by matrix multiplication followed by bias offset.



```
def create_fc_layer(input,
             num_inputs,
             num_outputs,
             use_relu=True):

    #Let's define trainable weights and biases.
    weights = create_weights(shape=[num_inputs, num_outputs])
    biases = create_biases(num_outputs)

    # Fully connected layer takes input x and produces wx+b.Since, these are matrices, we use matmul function in Tenso
    layer = tf.matmul(input, weights) + biases
    if use_relu:
        layer = tf.nn.selu(layer)

    return layer
```



```
def create_convolutional_layer(input,
               num_input_channels,
               conv_filter_size,
               num_filters):

    ## We shall define the weights that will be trained using create_weights function.
    weights = create_weights(shape=[conv_filter_size, conv_filter_size, num_input_channels, num_filters])
    ## We create biases using the create_biases function. These are also trained.
    biases = create_biases(num_filters)

    ## Creating the convolutional layer
    layer = tf.nn.conv2d(input=input,
                     filter=weights,
                     strides=[1, 1, 1, 1],
                     padding='SAME')

    layer += biases

    ## We shall be using max-pooling.
    layer = tf.nn.max_pool(value=layer,
                            ksize=[1, 2, 2, 1],
                            strides=[1, 2, 2, 1],
                            padding='SAME')
    ## Output of pooling is fed to Relu which is the activation function for us.
    layer = tf.nn.selu(layer)

    return layer
```



```
layer_conv1 = create_convolutional_layer(input=x,
               num_input_channels=num_channels,
               conv_filter_size=filter_size_conv1,
               num_filters=num_filters_conv1)
layer_conv2 = create_convolutional_layer(input=layer_conv1,
               num_input_channels=num_filters_conv1,
               conv_filter_size=filter_size_conv2,
               num_filters=num_filters_conv2)

layer_conv3= create_convolutional_layer(input=layer_conv2,
               num_input_channels=num_filters_conv2,
               conv_filter_size=filter_size_conv3,
               num_filters=num_filters_conv3)

layer_flat = create_flatten_layer(layer_conv3)

layer_fc1 = create_fc_layer(input=layer_flat,
                     num_inputs=layer_flat.get_shape()[1:4].num_elements(),
                     num_outputs=fc_layer_size,
                     use_relu=True)

layer_fc2 = create_fc_layer(input=layer_fc1,
                     num_inputs=fc_layer_size,
                     num_outputs=num_classes,
                     use_relu=False)
```

- **Results**

By testing different values of different hyper-parameters, we found the best combination is:
learning rate: 0.0003
activation function: Selu
iteration: 1250

- **Discussion**

The comparison of different values of hyper-parameters.

1. Iteration: the high iteration will lead to overlearning, while low iteration will lead to lack of learning. We compare the iteration with 250, 1000 and 1500.

2. Activation functions: We use different activation functions which has different algorithm to test the accuracy, Relu and Selu.

3. Learning rate: A low learning rate is more precise, but calculating the gradient is time-consuming, so it will take us a very long time to get to the bottom. While a high learning rate may risk overshooting the lowest point since the slope of the hill is constantly changing. We compare the learning rate in 0.001, 0.0005 and 0.0001.

- **Evaluation**

  Learning rate: 0.0001

activation functions\iterations	**250**	**1000**	**1500**
Relu	**25/90=0.28**	**40/90=0.44**	**39/90=0.43**
Selu	**39/90=0.43**	**38/90=0.42**	**33/90=0.37**

Learning rate: 0.0005

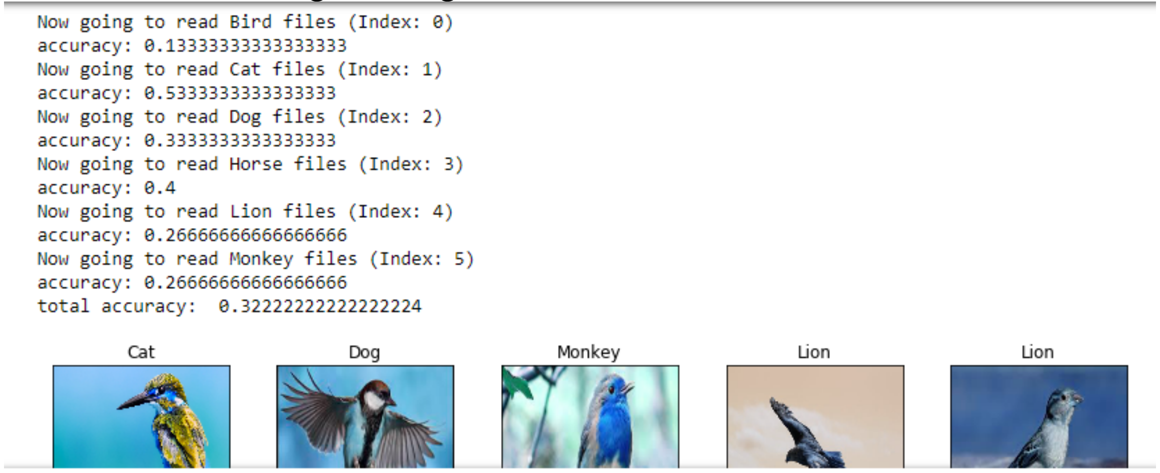
activation functions\iterations	**250**	**1000**	**1500**
Relu	**31/90=0.34**	**35/90=0.39**	**33/90=0.37**
Selu	**36/90=0.40**	**38/90=0.42**	**40/90=0.44**

Learning rate: 0.001

activation functions\iterations	**250**	**1000**	**1500**
Relu	**26/90=0.29**	**31/90=0.34**	**31/90=0.34**
selu	**31/90=0.34**	**38/90=0.42**	**40/90=0.44**

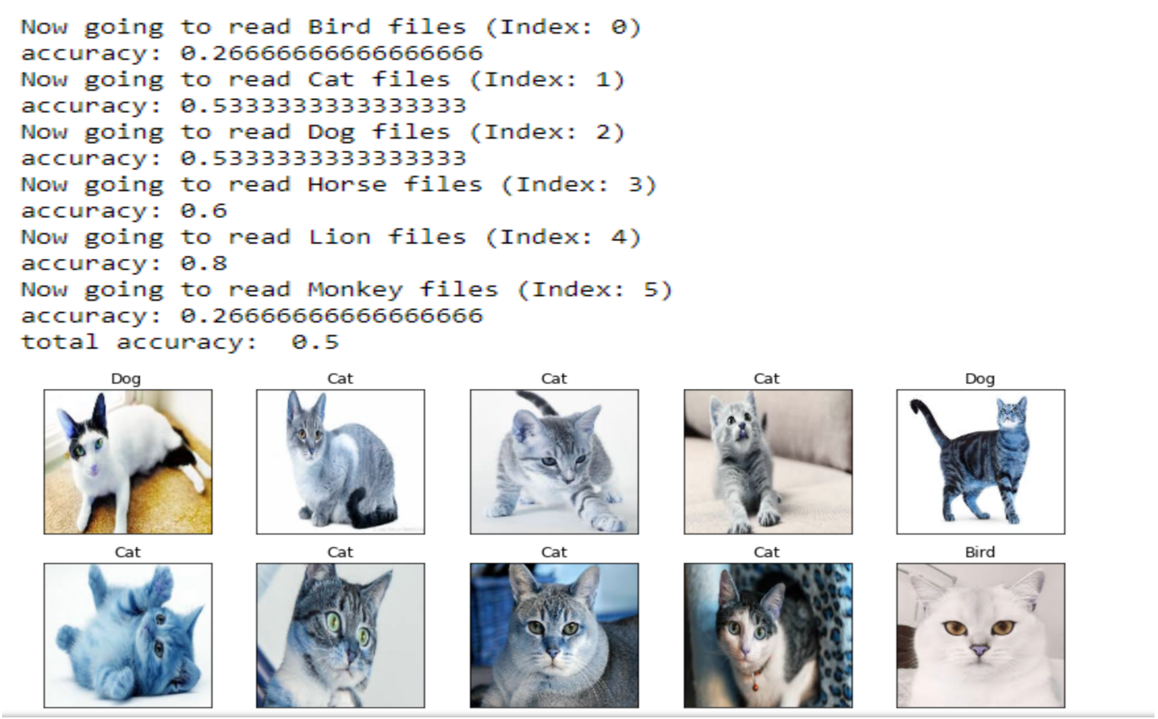
4. Data size: the more data for training leads to higher predicted accuracy. We originally use total of 60 images to train our model and get a 30% accuracy on prediction. Further, we add more images to training dataset. The total accuracy of prediction has increased to 50%.

Train CNN using 60 images:



```
Now going to read Bird files (Index: 0)
accuracy: 0.13333333333333333
Now going to read Cat files (Index: 1)
accuracy: 0.5333333333333333
Now going to read Dog files (Index: 2)
accuracy: 0.3333333333333333
Now going to read Horse files (Index: 3)
accuracy: 0.4
Now going to read Lion files (Index: 4)
accuracy: 0.26666666666666666
Now going to read Monkey files (Index: 5)
accuracy: 0.26666666666666666
total accuracy:  0.32222222222222224
```

Train CNN using 600 images:



```
Now going to read Bird files (Index: 0)
accuracy: 0.26666666666666666
Now going to read Cat files (Index: 1)
accuracy: 0.5333333333333333
Now going to read Dog files (Index: 2)
accuracy: 0.5333333333333333
Now going to read Horse files (Index: 3)
accuracy: 0.6
Now going to read Lion files (Index: 4)
accuracy: 0.8
Now going to read Monkey files (Index: 5)
accuracy: 0.26666666666666666
total accuracy:  0.5
```

We have reach to 50% accuracy with following hyper-parameter:

Activation function: Selu
Cost function: Cross Entropy
Gradient estimation: ADAM
Learning rate: 0.0003
Iteration: 1250

● Conclusion

The result of our project could reach quite high accuracy on identifying animal images. During the process we realized that two things are most import: hyper-parameters and training data size. Because we download each animal image on our own, in the beginning we trained our model with 60 images could have 30% accuracy. And further, only 100 images for each animal already able to train our model and get 50% accuracy. We believe if we could have enough data for training, our model could achieve higher accuracy on animals' classification.

● References

1. https://en.wikipedia.org/wiki/Convolutional_neural_network
2. https://github.com/nikbearbrown/INFO_7390/blob/master/Week_10/Convolutional_Neural_Network.ipynb
3. https://www.tensorflow.org
4. https://www.pexels.com/search/animal/
5. https://github.com/sankit1/cv-tricks.com/blob/master/Tensorflow-tutorials/tutorial-2-image-classifier/predict.py
6. https://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/