ME F376

Classifying Images of Mechanical Tools using CNN and Particle Swarm Optimization

Final Report

Prepared by:

Siddhant Setia

2018B4A40071G

Contents

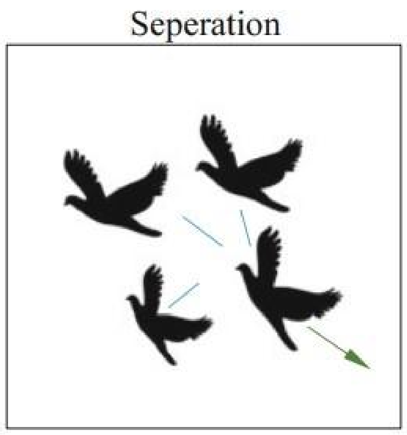
Introduction

This project is an attempt to explore whether particle swarm optimization can give a higher accuracy than other commonly used optimizers. A new dataset has been created using google images containing different types of mechanical tools. This data is then modified using a process known as data augmentation. Then the particle swarm optimization (PSO) algorithm is applied to the Convolutional Neural Network (CNN) model. Some commonly used optimizers are also applied and their performance is compared to the PSO CNN model.

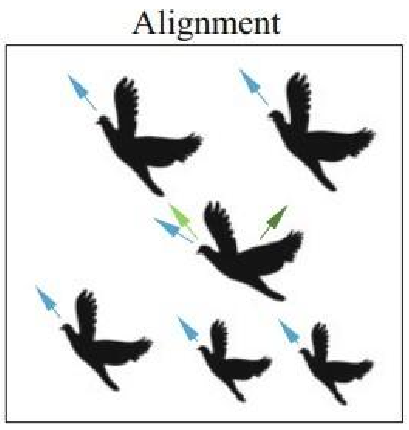
Particle Swarm Optimization

The PSO algorithm is inspired by the equations Raynold proposed for simulating interaction of individuals in flying swarms. Raynold shows there are 3 primitive principles in bird swarms: separation, alignment, and cohesion.

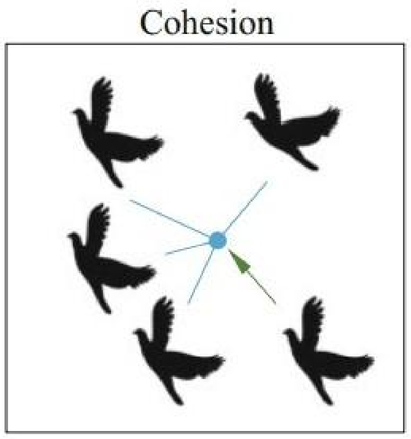
1) Separation: Separation applies forces to prevent collision



2) Alignment: The flying direction of an individual is adjusted based on its neighbors.



3) Cohesion: In cohesion, a certain distance is maintained between the individuals to avoid isolation.



Using the concept of bird swarms we can consider every solution of a given optimization problem as a particle which is able to move in a search landscape. In order to update the position of each particle, two vectors are considered: position vector and velocity vector. The position vector of particles is updated in each step of optimisation using the equation:

$$X_i(t + 1) = X_i(t) + V_i(t + 1)$$

Velocity vector of a particle is defined as:

$$V_i(t + 1) = V_i(t) + c_1r_1 (P_i(t) - x_i(t)) + c_2r_2 (G(t) - x_i(t))$$

Using these equations we can consider a function, provide initial positions of particles and find personal and global best values for each particle. Then we adjust the velocity of each particle according to this personal and global best. So, after each iteration we get closer to the optimal value.

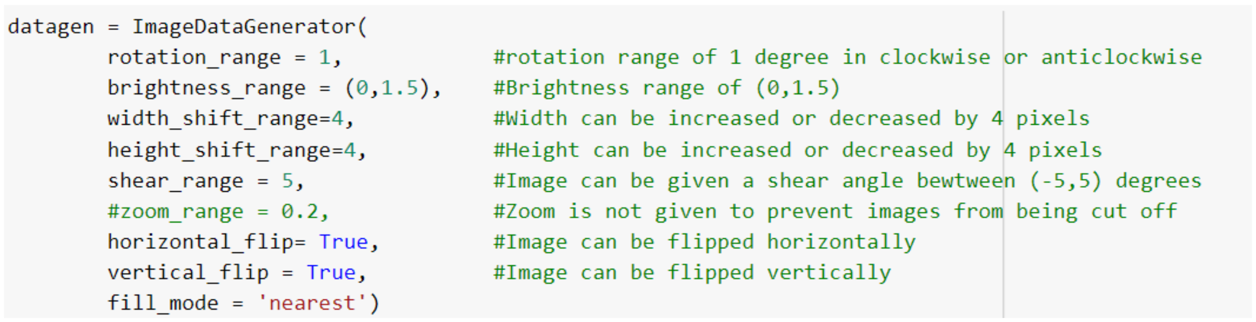
Dataset

The dataset contains images taken from google. These images are separated into 27 different classes. These classes are:

'Bolts Carriage', 'Bolts Eye', 'Bolts U', 'Chisels beveled edge', 'Chisels curved', 'Chisels dovetail', 'Chisels paring', 'Chisels skew', 'Files flat file', 'Files half rounded file', 'Files rounded file', 'Files square file', 'Files Triangular file', 'Nuts Hex', 'Nuts Lock' 'Nuts Shear', 'Nuts Wing', 'Screws Machine', 'Screws Self Tapping','Screws Socket', 'Screws Wood', 'Washer Cup', 'Washer External Tooth Lock', 'Washer Flat', 'Washer Internal Tooth Lock','Washer Spring', 'Washer Square Plate'.

After segregating the data, data cleaning is started. This included converting all images to the same format, cleaning the backgrounds and cropping the images.

Then, we start the Data Augmentation using the 'ImageDataGenerator' library. The following specifications were given:



```
datagen = ImageDataGenerator(
        rotation_range = 1,                #rotation range of 1 degree in clockwise or anticlockwise
        brightness_range = (0,1.5),        #Brightness range of (0,1.5)
        width_shift_range=4,               #Width can be increased or decreased by 4 pixels
        height_shift_range=4,              #Height can be increased or decreased by 4 pixels
        shear_range = 5,                   #Image can be given a shear angle bewtween (-5,5) degrees
        #zoom_range = 0.2,                 #Zoom is not given to prevent images from being cut off
        horizontal_flip= True,             #Image can be flipped horizontally
        vertical_flip = True,              #Image can be flipped vertically
        fill_mode = 'nearest')
```

All of these transformations are applied randomly to an image and ultimately we generate 20 images from each existing image. So, after data cleaning and augmentation the total number of images has increased from 554 to 11,393.

Code for Particle Swarm Optimization

Firstly I have defined a "particle" class which contains several functions which will define particles and update their velocity.

```python
class Particle:
    def __init__(self, min_layer, max_layer, max_pool_layers, input_width, input_height, input_channels, \
        conv_prob, pool_prob, fc_prob, max_conv_kernel, max_out_ch, max_fc_neurons, output_dim):
        self.input_width = input_width
        self.input_height = input_height
        self.input_channels = input_channels

        self.num_pool_layers = 0
        self.max_pool_layers = max_pool_layers

        self.feature_width = input_width
        self.feature_height = input_height
```

```python
        # Update initial velocity
        for i in range(len(self.layers)):
            if self.layers[i]["type"] != "fc":
                self.vel.append({"type": "keep"})
            else:
                self.vel.append({"type": "keep_fc"})

        self.model = None
        self.pBest = deepcopy(self)
```

Now, we define a population of particles. So, this population will try to find the best solution to a problem using a number of particles which we can control. Some of the inputs the function defined here takes is population size, height and width of images, minimum and maximum number of layers, probabilities of convolutional and pooling layers and output dimensions.

```python
class Population:
    def __init__(self, pop_size, min_layer, max_layer, input_width, input_height, input_channels, conv_prob, pool_prob, fc_prob,
        # Compute maximum number of pooling layers for any given particle
        max_pool_layers = 0
        in_w = input_width

        while in_w > 4:
            max_pool_layers += 1
            in_w = in_w/2

        self.particle = []
        for i in range(pop_size):
            self.particle.append(Particle(min_layer, max_layer, max_pool_layers, input_width, input_height, input_channels, con
```

Then we define a fit function which updates the velocities according to the accuracy of the global best.

```
def fit(self, Cg, dropout_rate):
    for i in range(1, self.n_iter):
        gBest_acc = self.gBest_acc[i-1]
        gBest_test_acc = self.gBest_test_acc[i-1]

        for j in range(self.pop_size):
            print('Iteration: ' + str(i) + ' - Particle: ' + str(j+1))

            # Update particle velocity
            self.population.particle[j].velocity(self.gBest.layers, Cg)
```

Then, we define functions to compile this model and give us the accuracy of the model.

```
def evaluate_gBest(self, batch_size):
    print("\nEvaluating gBest model on the test set...")

    metrics = self.gBest.model.evaluate(x=self.x_test, y=self.y_test, batch_size=batch_size)

    print("\ngBest model loss in the test set: " + str(metrics[0]) + " - Test set accuracy: " + str(metrics[1]))
    return metrics
```

Now, we define all the parameters the particle and population functions take.

```
number_runs = 10
number_iterations = 10
population_size = 20

batch_size_pso = 32
batch_size_full_training = 32

epochs_pso = 1
epochs_full_training = 100

max_conv_output_channels = 256
max_fully_connected_neurons = 300

min_layer = 3
max_layer = 20
```

Now, we call the PSO CNN model:

```
for i in range(number_runs):
    print("Run number: " + str(i))
    start_time = time.time()
    pso = psoCNN(dataset=dataset, n_iter=number_iterations, pop_size=population_size,
                 batch_size=batch_size_pso, epochs=epochs_pso, min_layer=min_layer, max_layer=max_layer,
                 conv_prob=probability_convolution, pool_prob=probability_pooling,
                 fc_prob=probability_fully_connected, max_conv_kernel=max_conv_kernel_size,
                 max_out_ch=max_conv_output_channels, max_fc_neurons=max_fully_connected_neurons,
                 dropout_rate=dropout)

    pso.fit(Cg=Cg, dropout_rate=dropout)

    print(pso.gBest_acc)
```

PSO Results

The model took more than 5 hours to train and the best accuracy result was 93.14%. We might be able to get a better accuracy by changing the number of particles or number of iterations.

Coding other optimization Algorithms

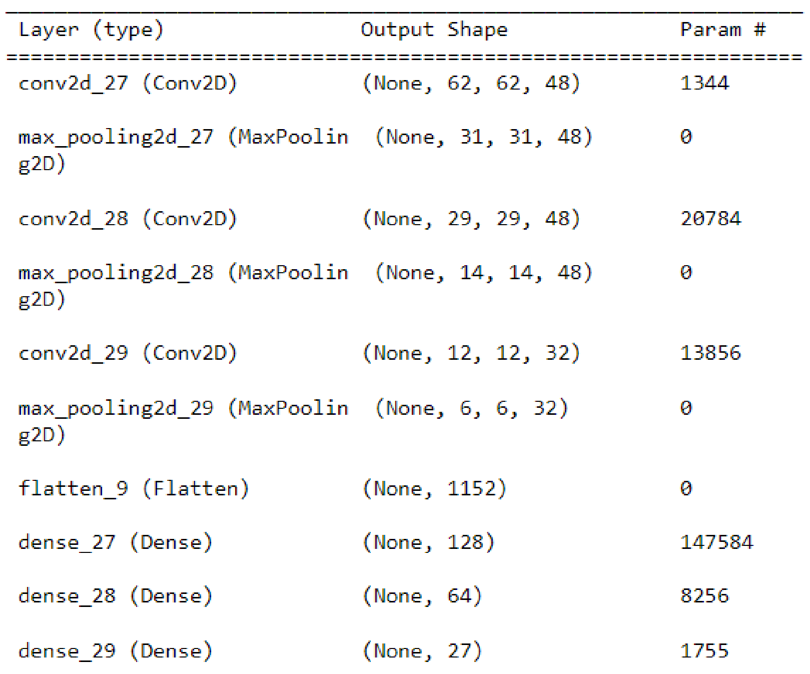
Optimization algorithms are important for deep learning. Training a complex deep learning model can take hours, so, the performance of optimization algorithms affect the model's training efficiency.

Understanding the principles of different optimization algorithms and the role of their hyperparameters will help us in tuning hyperparameters in such a manner that improves the performance of the model.

Here 4 optimizers are tested:

1. Adam
2. Stochastic Gradient Descent (SGD)
3. AdaGrad
4. RMS Prop

All optimizers are tested using the same model with 10 layers and using 3 epochs(number of iterations). The model summary is given below.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_27 (Conv2D)          (None, 62, 62, 48)        1344

 max_pooling2d_27 (MaxPoolin  (None, 31, 31, 48)       0
 g2D)

 conv2d_28 (Conv2D)          (None, 29, 29, 48)        20784

 max_pooling2d_28 (MaxPoolin  (None, 14, 14, 48)       0
 g2D)

 conv2d_29 (Conv2D)          (None, 12, 12, 32)        13856

 max_pooling2d_29 (MaxPoolin  (None, 6, 6, 32)         0
 g2D)

 flatten_9 (Flatten)         (None, 1152)              0

 dense_27 (Dense)            (None, 128)               147584

 dense_28 (Dense)            (None, 64)                8256

 dense_29 (Dense)            (None, 27)                1755
```

Results

The accuracy results of PSO as compared to other optimization algorithms are given below.

Optimization Algorithm	Accuracy(in %)
PSO	93.14
RMS Prop	74.54
Adam	71.3
SGD	16.67
AdaGrad	8.8

References

[1] Image classification using particle swarm optimization - M Omran, A Salman, A Angelbrecht.
https://www.researchgate.net/publication/284700349_Image_Classification_using_Particle_Swarm_Optimization

[2] Optimization of Convolutional Neural Networks Architectures Using PSO for Sign Language Recognition- Jonathan Fregoso, Claudia I. Gonzalez, Gabriela E. Martinez

https://www.mdpi.com/2075-1680/10/3/139

[3] https://github.com/feferna/psoCNN