Optimiser ResNet

Devesh Singh
Fakultät für Informatik
Otto von Guericke University Magdeburg
Magdeburg, Germany
devesh.singh@ovgu.de

***Abstract*—Forward passes through residual blocks are found to be equivalent to gradient descent steps. This leads to questions about more efficient residual blocks. This paper explores the idea of modifying residual networks using the particle swarm optimization (PSO) algorithm. The ResPSO model architecture is introduced to achieve this. This study tries to understand the effect of introducing PSO algorithms into residual blocks. Different experiments like clustering and muting model components, were designed as introspection techniques to examine the learning behaviour of the ResPSO model. As swarm's particles are modeled to be equivalent to feature maps, clustering these particles allowed exploration of PSO's effectiveness in adding information to the residual block. Muting model components while training, helped in understanding the interaction between the model components. Robustness was experimented with, as Gaussian noise introduction in input. The ResPSO model was compared to the classical Resnet model. ResPSO models are less accurate on test set than Resnet, but are more robust. Muting components and clustering experiments show that PSO is a useful and integral part of the model. The ResPSO models can extract features like - background colour, object colour, object shape and object alignment.**

***Index Terms*—Resnet, Particle Swarm Optimisation, Clustering, Robustness**

I. Introduction

Deep learning models, specifically Convolution Neural Networks (CNNs) have achieved SOTA results for the many image processing tasks. One major advancement was the Resnet from Microsoft Research [1]. It was able to achieve top results for object detection and object detection with localization in the ILSVRC in 2015. At the core of this Resnet architecture is the residual block depicted in Fig.1. Here, the output feature map x_{i+1} of a residual block, has a direct additive identity component of input feature map x_i, while the other component is a convolutional transformation of the feature map, here defined as $F(.)$. Thus the residual block's update equation is:

$$x_{i+1} = F_i(x_i) + x_i \tag{1}$$

This was the first time, a deep network with as many as 150 layers was successfully trained, for it was able to alleviate the problems of exploding or vanishing gradients using skip connections. S. Jastrzebski et al. [2] found that residual blocks encourage feature maps to move in the half space of negative loss gradient. This implies a forward pass through a residual block is equivalent to a gradient descent step.

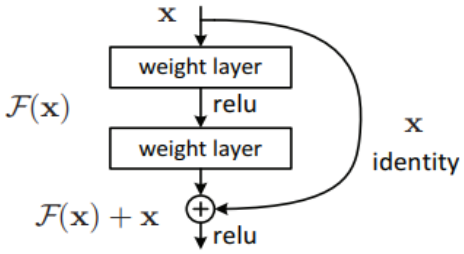


Fig. 1: Residual Block [1].

We can ask the question weather we can improve the gradient descent step and make residual blocks more efficient? Hence, heuristic optimisation algorithms like Particle Swarm Optimisation (PSO) [3] offer a possibility of incorporating a 'learned component' to feature maps while training the Resnet model.

Particle Swarm Optimisation is a heuristic optimisation algorithm based on fitness values of swarm particles at random points in the feature space. This nature inspired algorithm mimics the behaviour of a fish school or a flock of birds. Since its introduction in 1995, many variants of this algorithm have been explored and expounded upon.

In this paper I propose the ResPSO model. The model incorporates PSO algorithm in the residual block. This modified block is named as ResPSO block. ResPSO model architecture is a two-branched model, where the auxiliary networks are used to guide the swarm particles.

Section II of the paper contains the literature survey, while the section III contains the required background knowledge. Section IV explains all of the design details of ResPSO model. Section V deals with the experimental setup and analysis of ResPSO model. At last, the section VI contains conclusion and possible future work of this study.

II. Literature Survey

A. Residual network and improvements

Since the introduction of Resnet in 2015, there have been many improvements suggested to it. Often focusing of different factors of residual networks.

There have been improvements suggested to the structural use of existing residual blocks. One such example is Wide Resnets [4]. Wide Resnet tried to address the problem of diminishing feature reuse, where only few blocks of a deep

Resnet learn any useful representation. Wide Resnet increased the representational power of residual block by introducing more convolutional layers in a block and increasing the channel size. Authors found that Wide Resnet trained faster than classical Resnet and had better performance.

There have also been other models which were inspired by Resnet architecture. Densenet for instance also used identity (skip) connections [5]. Inside a dense block, there was an extensive use of skip connection. Each convolution layer takes all previous feature-maps as the input, and its output is used by all later layers as input. The feature maps were concatenated instead of being added. Densenets allow for an efficient feature reuse.

There were some other models which tried to incorporate other model and algorithms into the residual block. ResNeXt was an attempt to use the inception blocks inside the residual blocks, with skip connections between the blocks [6]. ResNeXt also uses a split-transform-merge paradigm, similar to inception blocks for feature updates. The number of paths inside the ResNeXt block is defined as 'cardinality'. Authors argue that while increasing the capacity of model, increasing cardinality is more effective than increasing depth or width of the model.

B. Heuristic optimisation of neural nets

Particle Swarm Optimization (PSO), has been used for training Neural Networks (NNs). NNs training could be thought of as a problem of finding optimal weights which minimise error on the training set. Where each swarm particle is a possible NN weights. Using network error as an optimisation criteria, particles move in the possible weights-space to reduce the error. In 1995, Eberhart and Kennedy were the first to show a successful utilisation of PSO to train Artificial Neural Networks [3], since then many variants of PSO have be used to train NNs. One study on simple feed forward neural network (FFNN) showed that PSO training can be an alternative to back-propagation [7].

Alternatively PSOs have been used to find optimal network typologies. Each particle represents a network topology and topology-space is explored to find the best network topology. For fully connected FFNN [8], Convolution Neural Network (CNN) [9] [10] or CNN Autoencoders [11]. Another approach by K.O. Stanley [12] uses evolutionary algorithms to update both network weight and network architecture, known as Topology and Weight Evolving Artificial Neural Network (TWEANN) algorithm.

III. Background

Here, we begin with the basics of PSO algorithm, its variants and relevant findings from Jastrzkebski [2] on residual block's behaviour.

A. Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) algorithm maintains a swarm of n possible solutions. Each solution has an associated fitness. Based on fitness, particles move in the search space to further optimise their fitness. The optimisation process is a non-deterministic process due to randomness in the algorithm. During each update iteration, individual particles try to move towards the best particle in their neighborhood as well towards their own best positions found in the past. Formally, each particle x_i^t moves in the search space with a velocity $V^t(x_i^t)$, where t is the time index denoting number of iterations and i is the particle index for each image.

$$x_i^{t+1} = x_i^t + V^{t+1}(x_i^t) \tag{2}$$

Now if $P_b(.)$ represents a function returning the swarm's global best particle at any given iteration and $P_i(.)$ represents a function returning a given particle's own best previous position. Then using scaling constants c_1, c_2, randomness r_1, r_2 and inertia w - the velocity updates as:

$$\begin{aligned} V^{t+1}(x_i^t) = w \cdot V^t(x_i^t) \\ + c_1 \cdot r_1 \cdot (P_i(x_i^t) - x_i^t) \\ + c_2 \cdot r_2 \cdot (P_b(x_i^t) - x_i^t) \end{aligned} \tag{3}$$

Equations 2 and 3 together define the basic particle swarm optimisation algorithm. Though the PSO algorithm have limitations like - prone to getting stuck in local minimum, low convergence rate to global optimum, velocity clipping resulting into an uncertain trade off between search space exploration and exploitation and dependence on initial parameter choices [13]–[15]. There have been multiple improvements to PSO algorithm, of which two relevant improvements are explained further.

1) Velocity Constriction: introduced by M. Clerc [16], [17], is a method to balance the exploration–exploitation trade-off, where the particle velocity is limited or constricted by a constant χ. The velocity updates then becomes:

$$\begin{aligned} V^{t+1}(x_i^t) = \chi \cdot [V^t(x_i^t) \\ + \phi_1 \cdot (P_i(x_i^t) - x_i^t) \\ + \phi_2 \cdot (P_b(x_i^t) - x_i^t)] \end{aligned} \tag{4}$$

And constriction χ is defined as,

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi(\phi - 4)}|} \tag{5}$$

where $\phi = \phi_1 + \phi_2$, $\phi_1 = c_1 \cdot r_1$ and $\phi_2 = c_2 \cdot r_2$. Equation 4 can be used when $\phi \geq 4$ and $\kappa \in [0, 1]$.

A constricted update of velocity is proven to converge to a stable point. Since $\chi \in [0, 1]$ ensures that velocity reduces at each time step. The parameter κ controls the trade off between exploration and exploitation in the search space [18], [19].

2) Perturb: In their study, K.D. Abeyrathnaa and C. Jeenanuntab [20] developed a window based mechanism for perturbing the swarm particles when they get stuck in a local minimum in a high dimensional search space. The location of particles is perturbed by changing the scaling factors c_1 *and* c_2

from equation 3, if the optimal fitness value does not decrease in a window of $t = 10$ iteration steps. The old position of the particles is also saved in case the found optimum was the global optimum. This perturbation was only utilised twice, as it was deemed enough for the particles to escape the local minimum.

In this study inspired from this method, for image particles a perturbation method was developed. Gaussian Noise was added to the particles if the average fitness of the swarm had not improved in the last 20 iteration steps.

B. Residual block interpretation

Jastrzkebski et al. [2] showed that residual blocks can be interpreted as an iterative gradient decent where features a refined over a number of layers. Following two ideas in this subsection are directly reproduced from the paper. These ideas also shape the intuition behind the proposed ResPSO blocks.

a) Feature map movement in gradient loss space: Assuming there are k Resnet blocks in the network followed by the loss $\mathcal{L}$ computation. Then from equation 1 would could see:

$$x_k = F_{k-1}(x_{k-1}) + x_{k-1} \tag{6}$$

The Taylor Series expansion of loss $\mathcal{L}$ would mean:

$$\mathcal{L}(x_k) = \mathcal{L}(F_{k-1}(x_{k-1}) + x_{k-1})$$

$$= \mathcal{L}(x_{k-1}) + F_{k-1}(x_{k-1}).\frac{\partial \mathcal{L}(x_{k-1})}{\partial x_{k-1}} + \mathcal{O}(F^2_{k-1}(x_{k-1}))$$

Recursively expanding feature map representations will give.

$$\mathcal{L}(x_k) = \mathcal{L}(x_i) + \sum_{j=i}^{k-1} F_j(x_j) \cdot \frac{\partial \mathcal{L}(x_j)}{\partial x_j} + \mathcal{O}(F^2_j(x_j))$$

This expansion shows that minimising the loss $\mathcal{L}$, is equivalent to minimizing the dot product between $F_j(x_j)$ and $\frac{\partial \mathcal{L}(x_j)}{\partial x_j}$, which can be achieved by making $F_i(x_i)$ point in the towards $-\frac{\partial \mathcal{L}(x_j)}{\partial x_j}$. Hence feature maps are naturally encouraged to move in the half space of negative loss gradient.

b) Movement in loss gradient space is equivalent to a gradient descent step: Let $x_o = W \cdot h + b$ be the output of the first convolution layer of a Resnet. Where weights W, bias b are the shared parameters of this layer and h is the raw image input. If x_o moves in the half space of $-\frac{\partial \mathcal{L}(x_o)}{\partial x_o}$, then it is equivalent to updating the parameters W, b of the convolution layer using a gradient update step. Assuming η to be the learning rate. Then updating x_o would be given by the change:

$$\Delta x_o = (W - \eta \frac{\partial \mathcal{L}}{\partial W})h + (b - \eta \frac{\partial \mathcal{L}}{\partial b}) - (Wh + b)$$

$$= -\eta \frac{\partial \mathcal{L}}{\partial W} h - \eta \frac{\partial \mathcal{L}}{\partial b}$$

$$= -\eta \frac{\partial \mathcal{L}}{\partial x_o} \left(\frac{\partial x_o}{\partial W} h + \frac{\partial x_o}{\partial b} \right)$$

$$= -\eta \frac{\partial \mathcal{L}}{\partial x_o} (\|h\|^2 + 1)$$

$$\propto -\frac{\partial \mathcal{L}}{\partial x_o}$$

Thus, moving x_o in the half space of $-\frac{\partial \mathcal{L}}{\partial x_o}$ has the same effect as updating the parameters W, b using gradient descent.

IV. Model

This section will explain all components of the proposed ResPSO model architecture. It beings with presenting details of a ResPSO block in section IV-A. In section IV-B describes the auxiliary network setup. At last, the section IV-C explains some additional architecture details which are required for training. In Fig.2 you can see the whole model.

A. ResPSO Block

ResPSO block is a modification of residual blocks. It contains an extra PSO component. The PSO tries to speed up the process of gradient descent steps of the residual block. Each particle of the swarm is an alteration of feature maps. The learnings of the swarm are added back to the ResPSO block.

1) Feature Map Update: Results from Jastrzkebski [2] points towards the possibility that incorporating an extra component $Q(x_i)$ which by proxy tries to optimise the loss $\mathcal{L}$. The ResPSO block's feature update looks like, also seen in Fig.3:

$$x_{i+1} = x_i + F_i(x_i) + Q(x_i) \tag{7}$$

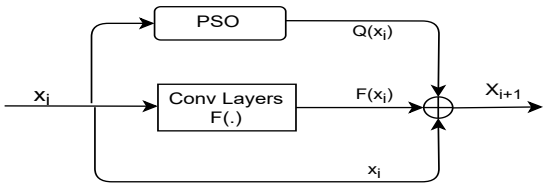


Fig. 3: ResPSO Block

Since each PSO blocks runs for a N number of iterations, passing the gradients through the PSO branch of the block meant having a computation graph which essentially occupies the whole available GPU memory. Due to this memory constraint, PSO processing was not included in the computation graph of the model.

a) Image-particle: After seeing all the basic details of PSO update in section III-A, it is now relevant to define the image-particle index i seen in equation 2.

As seen if Fig.2 and Fig.3, the input image is used in a tensor shaped like (**B**atch,**C**hannel,**H**eight,**W**idth). Each image

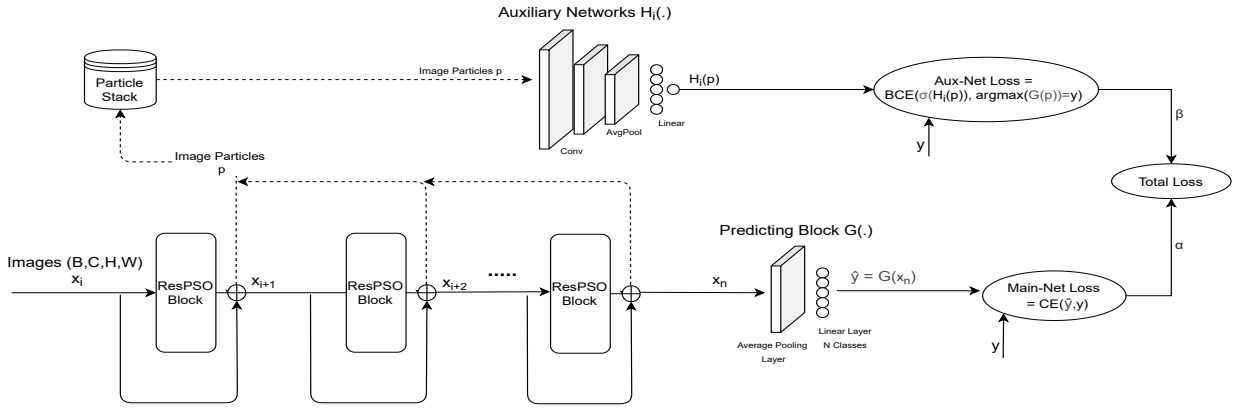


Fig. 2: ResPSO Network architecture

has related swarm particles in $C \times H \times W$ dimensional search space.

Since PSO is a heuristic optimisation algorithm, which explores the search space at with some randomness, it becomes imperative to explore this space as efficiently as possible. Hence, for each input image, four other image augmentations - namely color jittering, affine transformation and horizontal and vertical flips, were also included as image-particles. This addition of image-particles per input image transforms the input tensor from (B,C,H,W) to a (B,P,C,H,W) shaped tensor, where P is number of image augmentations.

b) Drop-in Feature: After all iterations of PSO algorithm inside the ResPSO block, it has to be decided how the learning of the swarm should be incorporated in the feature update. Adding drop-in features could fasten the process of gradient descent steps on ResPSO blocks and make them more efficient. For each image, there are two options for drop-in feature $DF(x_i)$:

1) *Best Feature:* Add the Global Best image-particle.
2) *Average Feature:* Add the average of all the P image-particles.

This leads up to the extra component $Q(x_i)$ as:

$$Q(x_i) = q \cdot (DF(x_i) - x_i) \tag{8}$$

where q is a scalar value used for regulating the influence of the drop-in feature $DF(x_i)$ on the next set of feature maps x_{i+1}. In equation 8, the identity x_i was subtracted from the drop-in feature because (in equation 7) we want the current feature map x_i to move in the direction of $DF(x_i)$.

B. Auxiliary Network

As seen in the Fig.2, the whole model is a multi-branched network. Here the main network is a conventional Resnet like architecure, in which k-ResPSO blocks are used one after the other followed by a predicting block $G(.)$ (see Fig.4). The main network predicts the class-probability of input images. While the auxiliary network $\mathcal{H}(.)$ attempts to guide the image-particles used inside each ResPSO block based on an approximation of the main network loss.

In terms of layer design, predicting block $G(.)$ takes the output of the last ResPSO block, applies average pooling, flattens the tensor and passes it through a linear layer. The auxiliary network $\mathcal{H}(.)$ (see Fig.5) is similar to the predicting block, except an extra linear layer, to get only a single activation value indicating correctness of $G(.)$'s prediction for each image-particle.

The auxiliary network tries to learn if the predicted label y' is correct or not. Mathematically the loss for Auxiliary network is:

$$y' = \arg\max_j(G_i(p)), \quad \forall p \in \mathcal{P} \tag{9}$$

$$Aux\text{-}Loss(p) = BCE(\ \sigma(\mathcal{H}_i(p))\ ,\ y' = y\ ) \tag{10}$$

Here j is the index of the image class. $\mathcal{P}$ is the Particle Stack. Some image-particles are kept aside during the forward pass to train the auxiliary net, this set is called the particle stack. Refer section IV-C2 for more details. σ is the sigmoid function while BCE is the Binary Cross Entropy Loss.

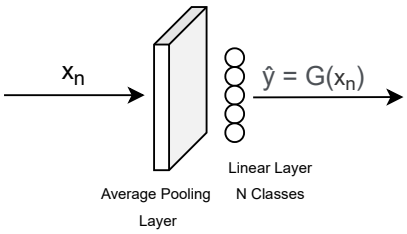


Fig. 4: Predicting Block $G(.)$

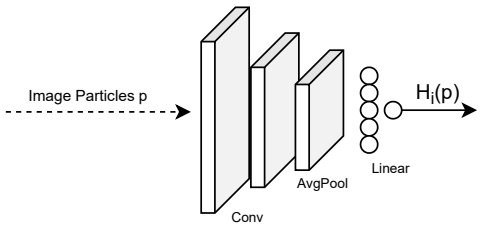


Fig. 5: Auxiliary Networks $\mathcal{H}_i(.)$

a) PSO Optimisation Criteria: The last detail that is required to make a functioning PSO, is the fitness value attached to any image-particle exploring the feature space. As also seen in the Fig.6 the fitness for each image-particle p is defined as $1 - \sigma(\mathcal{H}(p))$, which is the predicted error probability.

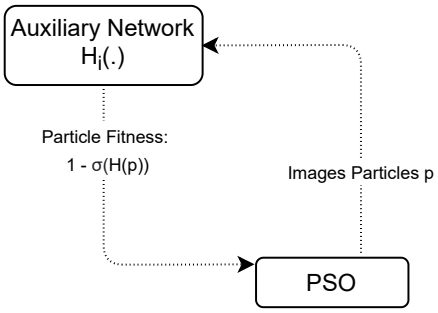


Fig. 6: PSO Optimisation Criteria

In this setup the optimisation goal is to minimise the fitness values.

b) Dynamic Feature Space: As it had been first introduced in the VGGnet [21], and has become a standard practice for CNN models since; as one goes deeper into a network from input to output - increase the number of channels C and reduce the size of feature maps $C \times H$. This architectural practice helps in reducing the number of features, while increasing the receptive field of each feature map. This is also responsible for making initial layers extract simple features such as edges while final layers extract textures and patterns [22].

This standard practice was also adopted for ResPSO architecture. Changing numbers of channels or size of feature maps semantically changes the search space in which image-particles are trying to optimise the loss function $\mathcal{L}$. This gives rise to a changing feature space due to downsampling.

To keep the image-particles semantically consistent, the same convolution layers from the main-network were used on image-particles which were used for downsampling the feature maps. Formally, if convolution layer $\mathcal{C}(.)$ was responsible for doubling the channels and halving both height and width of feature maps, then this same layer $\mathcal{C}(.)$ will be applied to the image particles.

Thus during the forward pass different ResPSO blocks will have image-particles that are of different dimensions. There are different auxiliary networks for each variant of feature space. Essentially, if there are n convolution layer $\mathcal{C}(.)$ which downsample the feature maps in the main-network, then there will be n auxiliary networks. Index i marks each of the these n variants of auxiliary networks $\mathcal{H}_i(.)$.

C. Training

In this section, I will detail some of the ResPSO architecture's features relevant for training.

1) Chain Mode: Between any two consecutive ResPSO blocks, there are three ways in which swarm populations can be initialised to learn from swarm population of the previous block:

- Exploitative: After all iterations of a ResPSO block, save the position of image-particles. The next ResPSO block's swarm is initialised at the saved position. This allows consecutive block's swarm population to exploitatively explore known regions of the feature space.
- Explorative: Random initialisation of the swarm population for better coverage of the feature space. Swarm populations from consecutive ResPSO blocks are not dependent on each other.
- Hybrid: An approach which combines both of the approaches mentioned above. Half of the swarm population gets initialized with an exploitative strategy while half of gets initialized with an explorative strategy.

2) Particle Stack: The auxiliary networks $\mathcal{H}_i(.)$ have learnable parameters in its convolution layers and linear layers. To train these parameters a loss associated with each of these network has to be calculated. It checks if the auxiliary networks output is close to the correctness of predicting block's $G(.)$ prediction. (See equations 9 and 10)

Hence, during a forward pass through main-network image-particles should be stored in a stack so that they can be used determine the *Aux-loss(.)*. Given the size of the swarm population and its repetitive usage over multiple ResPSO blocks makes it impossible to store all the image-particles across all the block. The memory requirements explode. To counter this, a fixed sized particle stack $\mathcal{P}$ was created. Image-particles are sampled at random from the swarm population and placed in this stack. If the stack is full and a new image-particle has to added, then one of the existing particle will be replaced at random from the stack. Each of the auxiliary network variants have its respective particle stack.

3) Network scaling: In the Fig.2, the total loss for the whole architecture is defined as a addition of losses from main-network and auxiliary networks.

$$\begin{aligned} \hat{y} &= Main\text{-}Net(x) \\ Main\text{-}Loss(x) &= CE(\hat{y}, y) \\ Avg\text{-}Aux\text{-}Loss(p) &= \frac{1}{\|\mathcal{P}\|\|B\|} \sum_{p}^{\mathcal{P}} \sum_{i}^{B} Aux\text{-}Loss_i(p) \end{aligned} \tag{11}$$

Here x is a batch of input images and their respective class labels being y. The $\mathcal{P}$ is the particle stack containing a sample of the image-particles. Index i identifies the variant of auxiliary network used. $CE(.)$ is the Cross Entropy Loss function. Then total loss would be defined as:

$$\begin{aligned} Total\text{-}Loss = \alpha \cdot Main\text{-}Loss(x) + \\ \beta \cdot Avg\text{-}Aux\text{-}Loss(p) \end{aligned} \tag{12}$$

α and β are two scalar weight variables used to tune the contribution of each network's loss towards the total loss. These scalars are chosen such as $\alpha + \beta = 1$ and $\alpha, \beta \in [0, 1]$.

V. Analysis

We begin this section with some of the implementation details of ResPSO architecture and then move to the experiments performed on the model.

The data-set used in this study is the CIFAR-10 [23]. There are 60,000 images in the data-set across 10 classes. The classes are balanced. The data-set is split into training set consisting of 50,000 images and test set consisting of 10,000 images. The images are of low resolution and are called 'Tiny images' by the authors of the data-set. The images have 3 channels with 32x32 pixels in each channels.

In the ResPSO block (Fig. 3), the convolution layers $F(.)$ represented two convolution layers of, each with 3x3 kernels. The convolution layers are followed by a batch normalisation operation. The ReLU function was chosen as the non-linearity.

As seen in equation 12, the loss components of main network and auxiliary network are weighted. The values of α and β were chosen as 0.8 and 0.2 respectively. In equation 8 The value of q used to scale the influence of drop-in feature was set to 0.75. The size of particle stack $\mathcal{P}$ is set at twice the swam size.

For PSO algorithm inside ResPSO block, the initial velocity of the PSO was set to zero. It was found that initializing the swarm particles with random values slows the process of convergence to the optimum. Random initialization of initial swarm velocity causes the particles to roam in the search space [24]. For the constricted velocity updates (equation 4), the random constants were set in the same range, $\phi_1, \phi_2 \in [2.2, 2.3]$ while the $\kappa \in [0, 1]$ coefficient which controlled the trade-off between exploration and exploitation was set to 0.5. Boundary handling is an often used PSO mechanism to keep the swarm population contained in the valid regions of search space and reinitialise the swarm particles if they go out of it. While implementing the PSO algorithm, no boundary handling was utilised as the whole search space $[-\infty, \infty]^{|C \times H \times W|}$ is deemed valid.

Model's training schedule was 25, 20, 15 epochs (a total of 60 epochs). Checkpoints were created after every five epochs of training. An ADAM optimiser was used to update the network's parameters. The learning rate given to the optimiser, was set at 1e-3 and reduced by a factor of 10 after each set of training epochs.

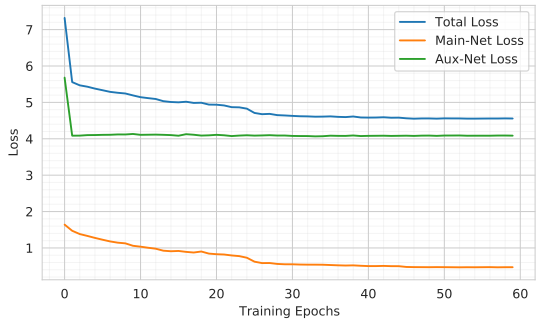


Fig. 7: Training loss of ResPSO model's network branches

A. Model performance

Chain mode, drop-in feature, image-particle perturbing, swarm size and model's depth were some the variables which were experimented with. To see how the model perform its accuracy on validation set over the training epochs, loss curve, training time and test accuracy was recorded.

From multiple runs it was found, that for the drop-in feature swarm's 'global best' image-particle instead of swarm's 'average position' helped the model perform better while other variables like perturbing image-particles and swarm size did not affect the model's test accuracy considerably. On the contrary, increasing the swam size forced the model to perform more computations while not adding to the overall generalisability.

Similarly for the chain mode between ResPSO blocks, adding extra randomness in the model using 'Explorative' or 'Hybrid' mode hindered with swarm's optimisation. Models with these chain modes consistently achieved less test accuracy than model with 'Exploitative' chain mode.

Following above experimentation, a ResPSO model with an 'exploitative' chain-mode, swarm size of 5 particles, 'global best' drop-in feature, no perturbation and of depth 21 blocks was chosen as an representative ResPSO model. After every 7 ResPSO blocks the feature maps were downsized. Further experiment analysis are based on this model.

From Fig. 7 we could see main and auxiliary network's loss components of the total architecture. The loss of main network keeps on decreasing with the training epoch. The loss for auxiliary networks hits a bottleneck.

To see how active the swam is over the training epochs, swarm's movement was averaged over each epoch. From equation 2, we see that movement at any given iteration is directly the result of its velocity. Hence to calculate average swarm movement in an epoch, swarm's absolute velocity was averaged over all its optimisation iteration. From Fig. 8, we see that the swarm's average movement keeps on increasing till 25 epochs, after which it drops and stabilises. This is an artifact of the job schedule and the learning rate. Even though the scale of movement is small, this increasing and then stabilising behaviour of swarm movement was noticed across multiple runs.

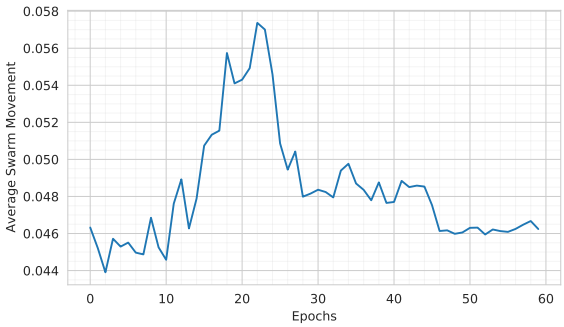


Fig. 8: Average movement of swarm particles over training epochs

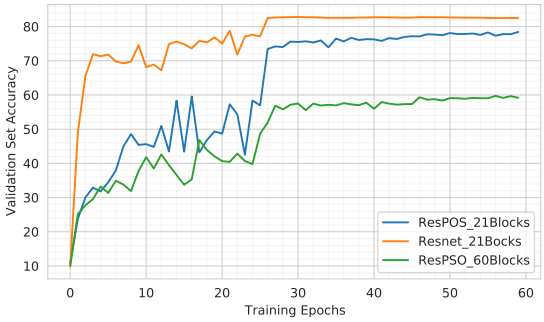


Fig. 9: Validation set accuracy over training epochs

To see how the model perform against a Resnet architecture, both the ResPSO and Resnet models were trained using the same training conditions. A deeper ResPSO model with 60 blocks (120 layers) was also trained.

From Fig. 9 and Fig. 10, we could see that Resnet outperforms ResPSO model, both in accuracy and training time. We can see a hike in model's accuracy around epoch 25, which is an artifact of the job schedule. And as we increase the depth of the ResPSO model, its performance drops.

This comparison with the Resnet model pushes us to further analyse the behaviour of ResPSO model and understand the

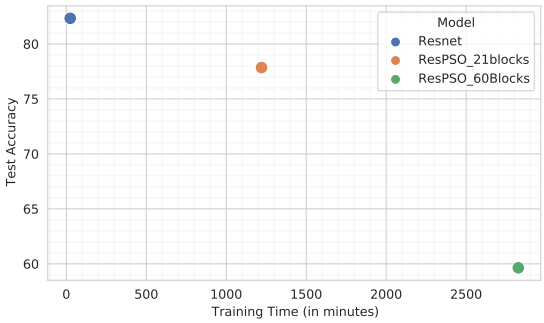


Fig. 10: Training time and test accuracy of the models

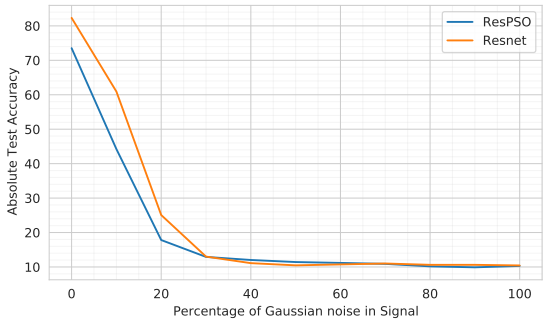


Fig. 11: Absolute test accuracy vs noise percentage

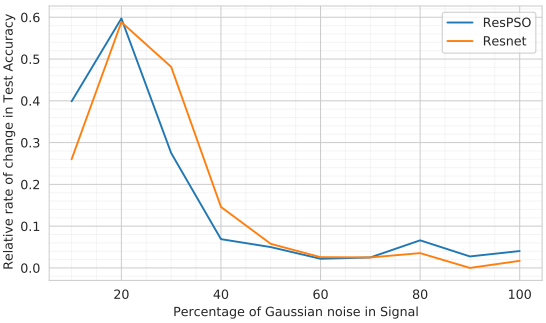


Fig. 12: Relative rate of change in test accuracy vs noise percentage

effects of incorporating PSO algorithm in a residual block.

B. Robustness

It could be hypothesised that using ResPSO blocks leads to a more robust network because of more extensive exploration of the search space by the swarm's image-particles. To test this hypothesis, it was experimented with how the trained ResPSO and Resnet network perform against increasing Gaussian noise N(0,1) in the input.

The first iteration begins with a noise-free test set images. In each successive iteration 10% more Gaussian noise is added, resulting in the signal reduction by 10%. In the last iteration input is pure noise and no signal.

The results of robustness experiment could be seen in the Fig. 11. Post the mark of 50% noise in the input signal, both of the models have about 10% test accuracy and could be ignored for further analysis as the model fail to classify.

Since the two models do not reach the same test accuracy, to compare them more fairly the relative rate of change in test accuracy on y-axis is plotted Fig. 12. As the relative rate of change in test accuracy is less for ResPSO model than that of Resnet with increment of noise, we could conclude that the ResPSO model is more robust than the Resnet model.

C. Muting model components

There are three components of the ResPSO block - residual or identity connection, convolution layers $F(.)$ and the PSO

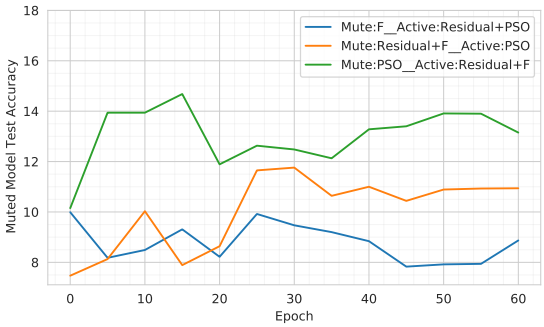


Fig. 13: Test accuracy of muted models

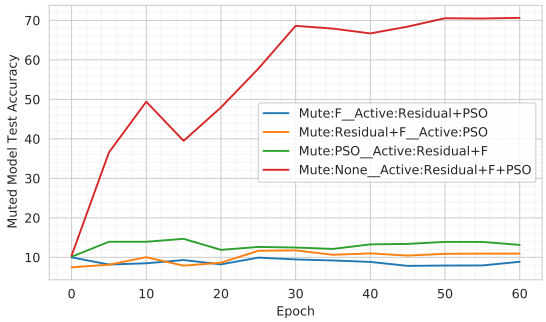


Fig. 14: Muted models in relation to a complete ResPSO

component $Q(.)$. To quantify by proxy which component add to the ResPSO model and in which manner, the model components were muted. Using the checkpoints created during training, to mute a model component meant to train up to that checkpoint with the three components and then simply taking out the component from ResPSO blocks and not using it while testing. The muted model's test accuracy was recorded. To quantify similarity between the resulting performances and the complete ResPSO model, Pearson correlation was used.

The results of muting model components experiment could be seen in the Fig. 13. The training accuracy of a complete ResPSO was added to the previous plot for context in Fig. 14.

From these plots we could see that neither of the components could account for the complete ResPSO model's performance on its own. For instance when considering the two setup of muted model which are symmetric to each other - a model with muted identity connection and convolution layers and active PSO and, another model with active identity residual connection and convolution layers and muted PSO. Simply adding up test accuracy of these two muted models does not match the test accuracy of a complete ResPSO model. More interestingly all the muted models perform almost equally bad. Their test accuracy never going above 15% (where a random model could theoretically achieve 10%). This points towards the fact that for a trained ResPSO model, neither of the three components could be ignored. The ResPSO model learns from some non-linear interaction of its three components.

For the model with active PSO and muted identity connection and convolution, an unexpectedly high Pearson correlation of 0.88 was found with complete ResPSO model's test accuracy. Thus we could assume that an ideal PSO would increase the generalisability of the complete model. Though this result comes from one run of the model and more runs are required to factor in PSO algorithm's randomness.

D. Clustering

The experiments in this section were designed to test if the image-particles add meaningful information to the ResPSO model. Clustering was used as an method of introspection on the way image-particles of the swarm were positioned in the feature space. This helps in estimating effectiveness of auxiliary network's setup.

Component(s)	Pearson Correlation to complete a ResPSO
Active: Residual and PSO Mute: Convolution	-0.39
Active: PSO Mute: Residual and Convolution	**0.88**
Active: Residual and Convolution Mute: PSO	0.40

TABLE I: Pearson Correlation between accuracy curves of muted models and complete ResPSO

1) Total test set: All the 50,000 image-particles related to the 10,000 images in test set were clustered. K-means clustering (K=10) algorithm was used. We try to understand if the ten image-classes in the data-set were meaningfully distinguished by the model.

Cosine distance was chosen as the distance metric because of high-dimensional nature of feature space. Ten random image-particles were chosen to initialise the centroids. The K-means algorithm ran for ten iterations. For the clustering found, silhouette coefficient was used as the metric of clustering qualify. Homogeneity and purity of each cluster was also looked into.

```
Cluster: #8
Cluster Size: 7925
Class: plane | count: 2193 | Purity (percentage): 27.67
Class: car   | count: 574  | Purity (percentage): 7.24
Class: bird  | count: 723  | Purity (percentage): 9.12
Class: cat   | count: 398  | Purity (percentage): 5.02
Class: deer  | count: 418  | Purity (percentage): 5.27
Class: dog   | count: 307  | Purity (percentage): 3.87
Class: frog  | count: 176  | Purity (percentage): 2.22
Class: horse | count: 347  | Purity (percentage): 4.38
Class: ship  | count: 1946 | Purity (percentage): 24.56
Class: truck | count: 843  | Purity (percentage): 10.64

Cluster: #9
Cluster Size: 892
Class: plane | count: 46  | Purity (percentage): 5.16
Class: car   | count: 80  | Purity (percentage): 8.97
Class: bird  | count: 57  | Purity (percentage): 6.39
Class: cat   | count: 86  | Purity (percentage): 9.64
Class: deer  | count: 69  | Purity (percentage): 7.74
Class: dog   | count: 66  | Purity (percentage): 7.40
Class: frog  | count: 42  | Purity (percentage): 4.71
Class: horse | count: 268 | Purity (percentage): 30.04
Class: ship  | count: 58  | Purity (percentage): 6.50
Class: truck | count: 120 | Purity (percentage): 13.45
```

Fig. 15: Sample Clustering Result

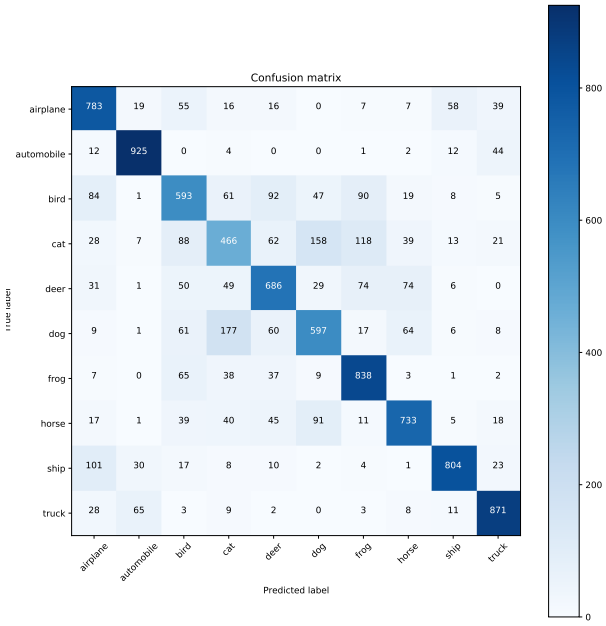
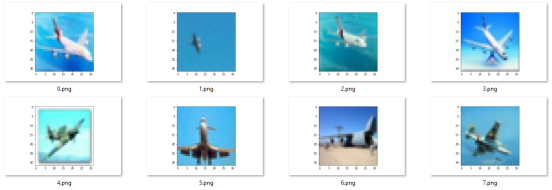


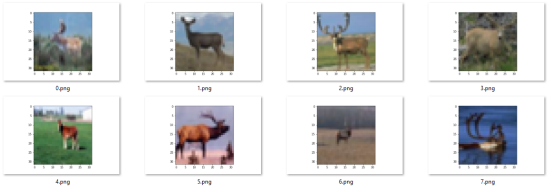
Fig. 16: ResPSO model's confusion matrix



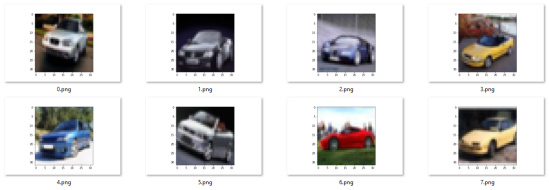
(a) Planes with blue background

(b) Planes with white background

Fig. 17: Feature extracted: Background colour



(a) Deer with antlers



(b) Cars with slanted front facing alignment

Fig. 18: Feature extracted: Object shape

In Fig. 15 we see clustering result's extracts. Two out of ten clusters formed by clustering all the image-particles of test set are shown. The complete clustering is reported in the Appendix.

Cluster #8 had two classes predominantly, '(Air)Plane' and 'Ship', while the cluster #9 had class 'Horse' predominantly. In the test set, each image class had $1000 \times 5 = 5000$ image-particles associated with it. Thus cluster #8 had approximately 40% of total 'plane' image-particles and 40% of total 'ship' image-particles in it. This makes cluster #8 a big vehicle type cluster, having each prominent class's purity being $\sim 25.0\%$.

Many times two or three predominant class exist in a cluster and sometimes no predominant class exist in a cluster. In an ideal clustering, the 10 clusters would be completely homogeneous and well separated. Presence of multiple predominant class in a cluster suggest that instead of distinguishing between 'classes', the swarm learns 'features' which might overlap across classes.

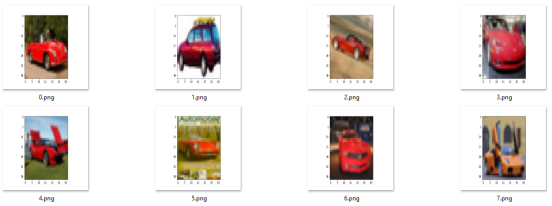
Swam clusters were able to distinguish between vehicle image classes and animal image classes, as seen from the clusters in Fig. 15. This is also confirmed by the confusion matrix of the model. In Fig. 16 the rows are true classes and the columns are the predicted classes. We see a square being formed by the animal classes - bird, cat, deer, dog, frog and horse, showing that these classes are confused with each other but not with the vehicle classes.

To quantify clustering quality silhouette coefficient was used. Silhouette coefficient range from -1 to +1. A value of +1 represents clusters that are well separated and the each image-particle lies well within its cluster. We find a silhouette coefficient of 0.07 for our clustering. This shows that there was much overlap between the clusters found. This low clustering quality could come from, K-means algorithm's bias for spherical clusters or poor initial centroid choice. K-means is also prone to giving poor results in high-dimensional spaces.

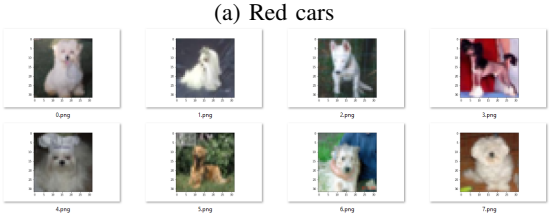
This imply that image-particles might be positioned in a more suitable manner than K-means algorithm suggests. Density based clustering or subspace clustering algorithms are more suitable algorithms in this scenario but were avoided due to their complexity.

2) Pure class clustering: In this experiment all the image-particles from a single image class were grouped in sub-clusters. Using clustering solely for a single class helped in finding features extracted by the swarm.

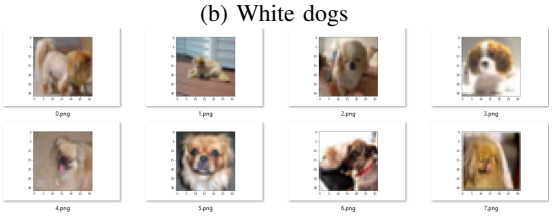
Again K-means algorithm, with cosine distance metric was applied. To find the optimal number of sub-clusters, knee point of sum of squared error vs number of clusters plot was used. To find the feature extracted in a sub-cluster, it was visually analysed. For the visual analysis, eight images from each were inspected. Three sub-cluster representative images which were closest to the sub-cluster centroid and five random images in the sub-cluster.



(a) Red cars



(b) White dogs



(c) Brown dogs

Fig. 19: Feature extracted: Object colour

As established in the previous experiment the clusters are not of high quality. Extracted feature overlapped between sub-clusters. But there were some clusters with evident presence of an extracted feature.

From Fig 17, Fig. 18 and Fig. 19, we could understand that swarm's image-particles were able to extract features like background colour, object colour, object shape and object alignment.

VI. Conclusion

Jastrzkebski et al. [2] in their study found that residual blocks perform iterative gradient descent. To speed up this gradient descent process the ResPSO architecture was introduced. A particle swarm optimisation based modification of the residual block.

The classical Resnet model outperform ResPSO models in terms of test accuracy and training time. But the ResPSO model are more robust against Gaussian noise than Resnet model.

Further analysis reveals that the ResPSO model learns from some non-linear interaction of its three components - identity connection, convolution layers and PSO. None of these components can be muted out. From the clustering analysis of the swarm's image particles finds that model was able to extract features like background colour, object shapes, object colour and object alignment.

Hence there lies much potential for ResPSO models. Following are some of the possible future works.

- By design ResPSO architecture is modular. Hence other CNN blocks such as a dense block or inception block could be incorporated in the ResPSO architecture and experimented with.
- The bottleneck in loss curves (see Fig. 7) and high Pearson correlation between model with only PSO active and muted identity connection and convolution layers (see Table I), point towards the fact that an 'ideal PSO' could increase the generalisability of the ResPSO model. As output of auxiliary network acts a fitness for swarm particles, alternative setup of auxiliary network should be experimented with. PSO variants which perform well in high-dimensional space should also be looked into.
- Adding convolution layer $F(.)$ as the fourth component of the velocity update equation (eq 3), and pass gradients through whole of the ResPSO block. Each ResPSO block could then be though of as parameter shared unrolled Resnet blocks. This again ties back to Jastrzkebski's finding that parameters sharing in Resnet architecture could be beneficial.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[2] S. Jastrzebski, D. Arpit, N. Ballas, V. Verma, T. Che, and Y. Bengio, "Residual connections encourage iterative inference," *arXiv preprint arXiv:1710.04773*, 2017.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[4] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[5] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[6] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

[7] V. G. Gudise and G. K. Venayagamoorthy, "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks," in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*, pp. 110–117, IEEE, 2003.

[8] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj, "Evolutionary artificial neural networks by multi-dimensional particle swarm optimization," *Neural networks*, vol. 22, no. 10, pp. 1448–1462, 2009.

[9] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, IEEE, 2018.

[10] F. E. F. Junior and G. G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm and Evolutionary Computation*, vol. 49, pp. 62–74, 2019.

[11] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "A particle swarm optimization-based flexible convolutional autoencoder for image classification," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 8, pp. 2295–2309, 2018.

[12] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[13] D. E. Gbenga and E. I. Ramlan, "Understanding the limitations of particle swarm algorithm for dynamic optimization tasks: A survey towards the singularity of pso for swarm robotic applications," *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, pp. 1–25, 2016.

[14] M. Li, W. Du, and F. Nian, "An adaptive particle swarm optimization algorithm based on directed weighted complex network," *Mathematical problems in engineering*, vol. 2014, 2014.

[15] Z. Abdmouleh, A. Gastli, L. Ben-Brahim, M. Haouari, and N. A. Al-Emadi, "Review of optimization techniques applied for the integration of distributed generation from renewable energy sources," *Renewable Energy*, vol. 113, pp. 266–280, 2017.
[16] M. Clerc, "The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization," vol. 3, p. 1957 Vol. 3, 02 1999.
[17] M. Clerc and J. Kennedy, "Kennedy, j.: The particle swarm: Explosion, stability and convergence in a multi-dimensional complex space. ieee trans. on evolutionary computation 6, 58-73," *Evolutionary Computation, IEEE Transactions on*, vol. 6, pp. 58 – 73, 03 2002.
[18] A. P. Engelbrecht, *Computational intelligence: an introduction*. John Wiley & Sons, 2007.
[19] M. R. Bonyadi, Z. Michalewicz, and X. Li, "An analysis of the velocity updating rule of the particle swarm optimization algorithm," *Journal of Heuristics*, vol. 20, no. 4, pp. 417–452, 2014.
[20] K. D. Abeyrathna and C. Jeenanunta, "Escape local minima with improved particle swarm optimization algorithm," in *Norsk IKT-konferanse for forskning og utdanning*, 2019.
[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[22] C. Molnar, *Interpretable Machine Learning*. 2019. https://christophm.github.io/interpretable-ml-book/.
[23] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.
[24] A. Engelbrecht, "Particle swarm optimization: Velocity initialization," in *2012 IEEE congress on evolutionary computation*, pp. 1–8, IEEE, 2012.

APPENDIX

In the following images, we see all the clusters that were found by using the K-means clustering on test set's image particles. Predominant classes in a cluster were highlighted.

Cluster: #0
Cluster Size: 2407

Class: plane	count: 35	Purity (percentage): 1.45
Class: car	count: 15	Purity (percentage): 0.62
Class: bird	count: 703	Purity (percentage): 29.21
Class: cat	count: 71	Purity (percentage): 2.95
Class: deer	count: 853	Purity (percentage): 35.44
Class: dog	count: 165	Purity (percentage): 6.86
Class: frog	count: 424	Purity (percentage): 17.62
Class: horse	count: 126	Purity (percentage): 5.23
Class: ship	count: 2	Purity (percentage): 0.08
Class: truck	count: 13	Purity (percentage): 0.54

Cluster: #1
Cluster Size: 265

Class: plane	count: 25	Purity (percentage): 9.43
Class: car	count: 60	Purity (percentage): 22.64
Class: bird	count: 10	Purity (percentage): 3.77
Class: cat	count: 35	Purity (percentage): 13.21
Class: deer	count: 20	Purity (percentage): 7.55
Class: dog	count: 30	Purity (percentage): 11.32
Class: frog	count: 15	Purity (percentage): 5.66
Class: horse	count: 5	Purity (percentage): 1.89
Class: ship	count: 5	Purity (percentage): 1.89
Class: truck	count: 60	Purity (percentage): 22.64

Cluster: #2
Cluster Size: 4760

Class: plane	count: 261	Purity (percentage): 5.48
Class: car	count: 693	Purity (percentage): 14.56
Class: bird	count: 407	Purity (percentage): 8.55
Class: cat	count: 702	Purity (percentage): 14.75
Class: deer	count: 310	Purity (percentage): 6.51
Class: dog	count: 665	Purity (percentage): 13.97
Class: frog	count: 493	Purity (percentage): 10.36
Class: horse	count: 404	Purity (percentage): 8.49
Class: ship	count: 438	Purity (percentage): 9.20
Class: truck	count: 387	Purity (percentage): 8.13

Cluster: #3
Cluster Size: 12231

Class: plane	count: 341	Purity (percentage): 2.79
Class: car	count: 1440	Purity (percentage): 11.77
Class: bird	count: 1156	Purity (percentage): 9.45
Class: cat	count: 1814	Purity (percentage): 14.83
Class: deer	count: 1290	Purity (percentage): 10.55
Class: dog	count: 1783	Purity (percentage): 14.58
Class: frog	count: 1244	Purity (percentage): 10.17
Class: horse	count: 1558	Purity (percentage): 12.74
Class: ship	count: 415	Purity (percentage): 3.39
Class: truck	count: 1190	Purity (percentage): 9.73

Cluster: #4
Cluster Size: 11131

Class: plane	count: 1818	Purity (percentage): 16.33
Class: car	count: 1657	Purity (percentage): 14.89
Class: bird	count: 672	Purity (percentage): 6.04
Class: cat	count: 730	Purity (percentage): 6.56
Class: deer	count: 376	Purity (percentage): 3.38
Class: dog	count: 559	Purity (percentage): 5.02
Class: frog	count: 349	Purity (percentage): 3.14
Class: horse	count: 858	Purity (percentage): 7.71
Class: ship	count: 2026	Purity (percentage): 18.20
Class: truck	count: 2086	Purity (percentage): 18.74

Cluster: #5
Cluster Size: 774

Class: plane	count: 29	Purity (percentage): 3.75
Class: car	count: 120	Purity (percentage): 15.50
Class: bird	count: 70	Purity (percentage): 9.04
Class: cat	count: 105	Purity (percentage): 13.57
Class: deer	count: 113	Purity (percentage): 14.60
Class: dog	count: 73	Purity (percentage): 9.43
Class: frog	count: 133	Purity (percentage): 17.18
Class: horse	count: 65	Purity (percentage): 8.40
Class: ship	count: 17	Purity (percentage): 2.20
Class: truck	count: 49	Purity (percentage): 6.33

Cluster: #6
Cluster Size: 3984

Class: plane	count: 137	Purity (percentage): 3.44
Class: car	count: 96	Purity (percentage): 2.41
Class: bird	count: 652	Purity (percentage): 16.37
Class: cat	count: 211	Purity (percentage): 5.30
Class: deer	count: 833	Purity (percentage): 20.91
Class: dog	count: 366	Purity (percentage): 9.19
Class: frog	count: 785	Purity (percentage): 19.70
Class: horse	count: 795	Purity (percentage): 19.95
Class: ship	count: 35	Purity (percentage): 0.88
Class: truck	count: 74	Purity (percentage): 1.86

Cluster: #7
Cluster Size: 5631

Class: plane	count: 115	Purity (percentage): 2.04
Class: car	count: 265	Purity (percentage): 4.71
Class: bird	count: 550	Purity (percentage): 9.77
Class: cat	count: 848	Purity (percentage): 15.06
Class: deer	count: 718	Purity (percentage): 12.75
Class: dog	count: 986	Purity (percentage): 17.51
Class: frog	count: 1339	Purity (percentage): 23.78
Class: horse	count: 574	Purity (percentage): 10.19
Class: ship	count: 58	Purity (percentage): 1.03
Class: truck	count: 178	Purity (percentage): 3.16

Cluster: #8
Cluster Size: 7925

Class: plane	count: 2193	Purity (percentage): 27.67
Class: car	count: 574	Purity (percentage): 7.24
Class: bird	count: 723	Purity (percentage): 9.12
Class: cat	count: 398	Purity (percentage): 5.02
Class: deer	count: 418	Purity (percentage): 5.27
Class: dog	count: 307	Purity (percentage): 3.87
Class: frog	count: 176	Purity (percentage): 2.22
Class: horse	count: 347	Purity (percentage): 4.38
Class: ship	count: 1946	Purity (percentage): 24.56
Class: truck	count: 843	Purity (percentage): 10.64

Cluster: #9
Cluster Size: 892

Class: plane	count: 46	Purity (percentage): 5.16
Class: car	count: 80	Purity (percentage): 8.97
Class: bird	count: 57	Purity (percentage): 6.39
Class: cat	count: 86	Purity (percentage): 9.64
Class: deer	count: 69	Purity (percentage): 7.74
Class: dog	count: 66	Purity (percentage): 7.40
Class: frog	count: 42	Purity (percentage): 4.71
Class: horse	count: 268	Purity (percentage): 30.04
Class: ship	count: 58	Purity (percentage): 6.50
Class: truck	count: 120	Purity (percentage): 13.45

Fig. 22: Test set image-particles clustering