# Object Recognition

## PSO-FNN-CNN-Trad.CNN

Priyanka Vyas pvyas2@kent.edu
Wayne Fincher dfincher@kent.edu
Department of Computer Science
Kent State University

**Abstract:**

This study is about training a neural network to classify objects in an image using two different methods and comparing them. The study aims to identify two types of pistachio which are frequently grown in Turkey, by classifying them using Particle Swarm Optimization and Convolutional Neural Network based models. The CNN approach is experimented using two different convolutional neural network models i.e a basic convolutional network network and an implementation of VGG16 model referred to in Ilker et al paper [1].

**Keywords:**

Particle swarm optimization, Fuzzy layer, Deep neural network, Convolutional neural networks, VGG16, Image Classification

## 1.Introduction

In this study the aim is to train a neural network utilizing the particle swarm optimization algorithm. With PSO, as inspired from nature in the way many organisms form a swarm which roams about in their habitat. In this approach, the swarm is a collection of sets of weights. Each particle is a weight, which can also be thought of as a position in space. The goal is to be able to correctly categorize an image into a set of classes based on feature data. Then, a more traditional CNN network is created in order to compare results with the PSO trained neural network. The overall idea is to create an application that recognizes the object in the captured photo and identify the category to which the object belongs using PSO and CNN trained models.

## 2. Approach I: Model Architecture:

### 2.1 Class Structure for Neural Network: Layer

**Layer (Abstract base class)**
This class implements a generic interface representing a single layer in a neural network. The layer is considered to take inputs which are outputs of the previous layer and are routed to the neurons in this layer. A vector of weights can also be taken as an input to the layer, with the inheriting classes specifying the particular structure of the neurons, their connections with the inputs and the application of the weights. Lastly, an activation function can be specified, which defaults to ReLU.

**InputLayer**

This layer determines the number of inputs that the model it is applied to will accept. No weights are used. The neurons in this layer simply pass through the input to the output. While not strictly needed, this class is implemented so as to be similar to Keras.

**SparseLayer**

This layer allows the individual connections from inputs to neurons to be specified. This specification is used to determine the number of weights needed.

**FuzzyMembershipLayer**

This layer is designed to provide a collection of fuzzy set membership functions. Underneath the hood, it uses a SpareLayer to handle the routing of the output of the fuzzy set membership functions to the neurons.

### DenseLayer

This layer maps every input to every neuron.

### Model

This class is a container for the layers of a neural network. Any number of Layer objects can be added so long as an InputLayer is the first one.

## 2.2. Data: Dataset (Abstract base class)

Dataset is a generic class specifying an interface between stored data and data consumers. Inheriting classes should specify where the data is being loaded from and how that data should be parsed.

### ArffDataset (Dataset)

This dataset specifies how .arff data is parsed.

### Pistachio (ArffDataset)

The Pistachio dataset is in .arff format. Therefore, this class inherits from ArffDataset. Since the method of parsing the data is already specified, all that is needed is to specify where the Pistachio dataset is located.

## 2.3. Particle Swarm Optimization

Central to this project is training a neural network using the particle swarm optimization algorithm (PSO). Each particle in the swarm is a vector consisting of weights used in the model for the neural network. The particles are given randomized starting locations and velocities. A loss function is used to determine the value associated with any particular weight vector.

Each particle keeps track of its own best positions, i.e. the position which minimizes the loss. After the loss is calculated for each particle, the global best position is updated if one of the particles has a position with less loss than the loss at the previous global best. Then the velocity for each particle is updated using the particle's current position, its own best position, and the global best position. The position is now updated using the updated velocity.

### Particle

Each particle in the PSO algorithm is a Particle object which keeps track of its own position, velocity, and best position.

## 2.4. Utility Classes

### Activation

The methods of this static class are various types of activation functions. Currently, ReLU and Softmax are implemented.

### Loss

The methods of this static class are various types of loss functions. Currently, just the Euclidean norm is specified.

## 2.5. Miscellaneous Classes

### Settings

This static class creates a settings json file if one does not exist. It then uses that file to load/save various settings such as paths and filenames.

### Log

A static class which is essentially a dictionary of labeled sets of logs. These logs can be saved to log files with the label appended to the filename.

## 2.6. Modeling: Modeling w/Pistachios dataset

The process of modeling a fuzzy neural network trained via particle swarm using the Pistachio dataset begins with loading and formatting the data. The Dataset class allows us to bring different datasets using a unified interface. With the ArffDataset, any dataset in the .arff format can be used right away. The Dataset.Fetch method allows SQL access to the data, with "SELECT" and "LIMIT" clauses available. The "SELECT" clause allows particular features to be used. The "LIMIT" clause allows taking a certain number of items for training and the others for validation.

To build the model, we used an InputLayer, FuzzyMembershipLayer, and three DenseLayers. The FuzzyMembershipLayer routes each feature to two membership functions since there are only two classes in the Pistachio dataset. Next the model object is given to the PSO object for training. That object will create a specified number of particles with starting positions and velocities randomized. The PSO.ExecuteMany() method allows each dataset item and Particle pair to be inserted into the fuzzy neural network a specified number of times, refining the weights corresponding to the particular Particle on each iteration. When those rounds of training finish, the next item in the dataset is passed into the model. This process is repeated until all the items have been processed.

When the training is finished, a record of the best weights that were found during training are returned. The last item in the set is of course the last best set of weights overall. We can now use these weights directly with the model to validate it on the validation training set.

## 2.7. Experiment Results

While the Layers, Model, Dataset, and PSO classes all appear to be operating correctly, the Particles are losing velocity in only a few iterations. A number of parameters were changed, such as increasing the inertia term so as to slow down the rate of decrease in velocity. While this did slow down the decrease in velocity, convergence was still happening within ten to twenty iterations. We then tried adding a boost to the velocity of the particles every time the input value was changed. Thus, if the PSO algorithm was specified to do 20 iterations for input, the particles would converge within those 20 iterations. And then a new input is brought in and a boost to the velocity of any Particles with a speed of less than one unit. However, even this boost was not enough as the drop in velocity appears to be exponential while the boost is linear.

When validating the model, we found that the predictions were always biased to one of the classes. It turns out that the class of the first data set item used in training determined which way the model was biased. This may be explained by the very fast convergence of the model, as if overtraining was occurring near the start of training.

# 3. Approach II: Model Architecture

## 3.1 Convolutional Neural Network

Convolutional neural network, the approach is learned from the mathematical linear operation between matrices called convolution. CNN has multiple layers, including

**Convolution Layer:**

Convolution layer is the first layer and it transforms the input image with a kernel/filter. The filter size is smaller than the actual image size and is used to extract features from the input image. Each filter convolves with the image and creates an activation map. It preserves the relationship between pixels by learning the image features using small squares of input data.

**Activation Layer:**

The activation layer is used to adjust or cut off the generated output. The Rectified Linear Unit (Relu) is used as Relu has a constant gradient for the positive input. Relu creates a sparser representation, as the zero in the gradient leads to obtaining a complete zero.

**Pooling Layer**

Pooling layer is used for down-sampling i.e. to reduce the resolution for further layers. In this study, Max Pooling for size 2x2 is used for down sampling.

**Flatten Layer**

The output from the pooling layer is fed to the flatten layer, which then converts the resultant 2D arrays from pooled feature maps into a linear vector.

**Dense Layer**

Dense layer is the fully connected layer. The flatten matrix is fed as input to the fully connected layer to classify the image. The inside neurons of the dense layer are connected to every neuron in the preceding layer.

## 3.2 VGG16 CNN

The research paper Ilker et al paper [1] refers to a VGG16 model which is a very deep convolutional neural network and has multiple convolutional and Fully connected layers having tunable parameters as total 16. It has 13 convolutional layers , 3 fully connected layers

and 5 Max pooling layers. The activation function used for hidden layers is Relu.

**Dropout layer**

A dropout layer has also been added to help prevent overfitting. It is added after convolutional and pooling layers and is applied to each element in the feature map.

**Dense layer**

Amongst the three fully connected layers the first two has 4096 channels each and the third has 1000 channels, 1 for each class.

While implementing the traditional model, during this study we also tried implementing the approach to recognise images using VGG16. This model showed the same accuracy but also huge delays while processing the epochs as the network is very deep and has multiple layers.

## 3.3 Dataset

The dataset used to model and train the network are Image Datasets, these images were used as an input parameter and are fed into the CNN model to train and get the output as the label defined for that image. The images used as an input to CNN are RGB images,therefore each channel is individually convoluted and then combined to form a pixel.

## Pistachio (Image Dataset)

In CNN model approach, Pistachio image dataset(.jpg) is used as an input to classify the Pistachio type in an effective way. The dataset includes a total of 2148 images, 1232 of Kirmizi type and 916 of Siirt type.

## PetImages (Image Dataset)

In order to evaluate the efficiency of the CNN model for pistachio dataset, the same model was loaded with the PetImages dataset to assess the approach and accuracy of the existing model.

PetImages is a custom image dataset (.jpg) of cats and dogs including a total of 10032 images, 5013 of cats and 5019 of Dogs.

**Image data Parameters for Model Building**

1. Total number of Images

2. Image size

3. Number of classes: Categories in data folder

4. Training set size: One category per image

5. Test size image: One category per image

6. Number of layer and filters

7. Dropout layer (if needed)

## 3.4 Modeling

The neural network receives raw pixels (color images) as an input with input image size as 150x150x3. In order to connect the input layer to only one neuron, 32x32x3 weight connections are used for the dataset in the first convolution layer for feature extraction. The model has 32 filters in the first convolution layer and further two additional convolution layers of filter size 64 each is added. There are a total of three convolution layers and two Max Pooling layers. A flatten layer is added and then two dense layers with filter size 64 and 10 neurons at the end and ‘relu’ as the last layer's activation function. The dense layer after the flatten layer has more parameters due to flattening out the volume of the preceding layer and feeding it to the dense layer. The number of filter has also been gradually increased and then gradually decreased from first layer to last layer to balance the parameter in the model. To maintain the balance maxpooling for size 2x2 is added to gradually taper down from many filters to low filters.

The model training is done using Keras and the tools used to compile and train the model is Google Collab Following are the steps performed for model building and training:

**Loading the dataset:** Define path to the base directories and count the number of images in the folder to be evident that the dataset is read with each category as defined.

**Image Processing and Data augmentation:** Define the input image size and augment the images using the function ImageDataGenerator. This method reads the images directly from the directory and augments them while the neural network is learning in training data. Images from different classes are stored in different directories, although they all belong to the same parent folder. After rescaling once the parameters training, validation, image class and labels are defined, the data is

verified using Matplotlib.pyplot to verify the image received as an output after preprocessing.

**Compile and Training the model**

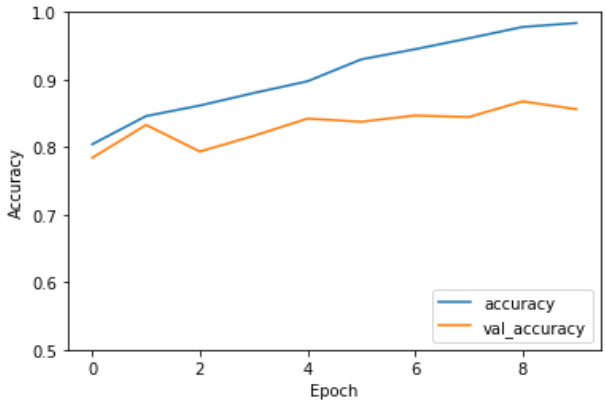
A sequential model is created with three convolution, two max pooling , one flatten and two dense layers and is compiled and trained using model.compile and model.fit function. Model.compile configures the model for training used loss function and optimizer i.e. Adam optimizer to train the model. Model.fit gets all the parameters to a correct value to map the defined input to defined labels as outputs. It trains the model for a fixed number of epochs. In this study, the number of iterations on a dataset is 10.

**Prediction**

Lastly, evaluate the model for training accuracy and validation accuracy. Training accuracy defines that the model is able to classify the images while performing training on training dataset and Validation accuracy defines that model is able to classify images with the validation dataset.

## 3.5 Experiment Results

Through this model, it was observed that the model achieved a training accuracy of 95% and validation accuracy of 85% . The model does not show a possibility of underfitting or overfitting as the same was dealt by adding the number of layers and using a cross validation approach for training the data.




This model was also experimented using PetImages and showed training accuracy of 65% and validation accuracy of 65%, This low accuracy for PetImages dataset was observed as the Image dataset had thumb and corrupt files in dataset the model could not train and perform well. The dataset was also in large numbers compared to the Pistachio image dataset.

PetImages dataset was then reduced and split into two folders as training and validation sets and was used as an input to the same model and achieved a training and validation accuracy of almost 78% .

| Dataset | Images | Train_acc | Val_acc |
|---|---|---|---|
| Pistachio | 2148 | 95% | 85% |
| PetImages | 12067 | 65% | 65% |

## 4. Challenges and Limitations:

While performing the imports dependency the system was experiencing library compatibility issues and could not process the arguments in the code.

Image Dataset required cleaning as the image dataset has thumb file, corrupt files, .db file which interrupted the model training and provided false outputs.

Image data set size is an important parameter and requires tuning through epochs and learning rate.

Underfitting or Overfitting affects the mapping of input to output and in turn affects the overall performance of the model leading to high computational speed. Through this study we aimed to strike an optimum balance while developing the model and this could be achieved by a balance between the bias and invariance to avoid both underfitting and overfitting scenarios.

The PSO trained model is highly biased and the particle velocity decreases too fast.

## 5. Conclusion

The next step in this project is to determine why the model trained via PSO has such a strong bias. Additionally, more sources of data with different sizes and numbers of features needs to be tested. With CNN model for image recognition, the existing model could be extended by introducing LSTM to learn feasibility for image caption generation.

**References:**


*[1].Singh, Dilbag, Yavuz S. Taspinar, Ramazan Kursun, Ilkay Cinar, Murat Koklu, Ilker A. Ozkan, and Heung-No Lee. 2022. "Classification and Analysis of Pistachio Species with Pre-Trained Deep Learning Models" Electronics 11, no. 7: 981. https://doi.org/10.3390/electronics11070981*

*[2]He Zhenya et al., "Extracting rules from fuzzy neural network by particle swarm optimisation," 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360), 1998, pp. 74-77, doi: 10.1109/ICEC.1998.699325.*

*[3]J. Kennedy, "The particle swarm: social adaptation of knowledge," Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97), 1997, pp. 303-308, doi: 10.1109/ICEC.1997.592326.*

*[4]Dohnal, M., 1983. Linguistics and fuzzy models. Computers in industry, 4(4), pp.341-345.*

*[5]Wu, Defeng & Warwick, Kevin & Ma, Zi & Gasson, Mark & Burgess, Jonathan & Pan, Song & Aziz, Tipu. (2010). Prediction of Parkinson's disease tremor onset using radial basis function neural network based on particle swarm optimization. International journal of neural systems. 20. 109-16. 10.1142/S0129065710002292.*

*[6]Jia, W. et al. (2020) 'Detection and segmentation of overlapped fruits based on optimized mask R-CNN application in apple harvesting robot', Computers and Electronics in Agriculture, 172. doi: 10.1016/j.compag.2020.105380.*

*[7]Ren, S. et al. (2015) 'Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks'. Available at:https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip&db=edsarx&AN=edsarx.1506.01497&site=eds-live&scope=site*

*[8]Shelhamer, E., Long, J. and Darrell, T. (2016) 'Fully Convolutional Networks for Semantic Segmentation'. Availableat:https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip&db=edsarx&AN=edsarx.1605.06211&site=eds-live&scope=site He, K. κ.ά. (2017) 'Mask R-CNN'. arXiv. doi: 10.48550/ARXIV.1703.06870.*

*[9]Huang, G. et al. (2017) 'Densely Connected Convolutional Networks', 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, CVPR, pp. 2261–2269. doi: 10.1109/CVPR.2017.243.*