TRIED US330X TP Carl Robinson, Simon Palma - 11/01/2018

Neural Networks and Deep Learning

Deep Learning with Keras

Objective: To use the Keras library to build and train deep feedforward neural networks to recognise handwritten digits from the MNIST database.
Data: 70000 binary (black and white) 28*28 pixel images of handwritten digits, split into 60000 train / 10000 test.

Exercice 1 : Logistic Regression with Keras

To begin, we create a simple multi-layered perceptron consisting of an input layer of 784 input nodes fully connected to an output layer of 10 softmax activation nodes. After training, this network will function as a simple handwritten digit class predictor.

Softmax regression (or multinomial logistic regression) is a generalization of logistic regression that enables us to handle multiple classes. In logistic regression we assume that the labels are binary, whereas softmax regression allows us to handle K number of classes e.g. K={1, 2,...,10} for ten MNIST handwritten digit classes.

Given a test input x, we want our network to estimate the probability P(y=k | x) for each value of K i.e. we estimate the probability of the class label taking on each of the K different possible values. By using a softmax activation our network outputs a K-dimensional vector (whose elements sum to 1) giving us our K estimated probabilities.

Architecture of network:

- Input layer: 784 nodes, one for each of the 28*28 pixels in an image
- Output layer: 10 nodes, one for each possible digit class.
    - Fully connected to input nodes.
    - Softmax activation - a generalisation of a logistic regression classifier that can handle >2 classes.
- Total parameters: 7850
- Hyperparameters
    - Optimiser: Stochastic Gradient Descent
    - Learning rate: 0.5
    - Loss function: Categorical Cross Entropy
    - Batch Size: 300
    - Number of Epochs: 10

Model Summary:

```
Layer (type)                 Output Shape              Param #
=================================================================
fc1 (Dense)                  (None, 10)                7850
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

Results on the test set:

- Loss: 27.19%
- Accuracy: 92.31% correct

Conclusions:

- The accuracy is good considering the simplicity of the model, and serves as a good baseline for more advanced methods.
- The low number of parameters mean the model is very fast during training and when making predictions.

Exercice 2 : Perceptron with Keras

A multilayer perceptron (MLP), perceptron used for this experience, is a feedforward neural network that generates a set of outputs from a set of inputs after a training has been made to this network. A MLP is characterized by several layers of neurons connected as a directed graph between one layer i and its following layer j (j>1). MLP uses the backpropagation algorithm for training the network, which calculates the error contribution of each neuron after a batch of input data is processed. Contrary to a single perceptron, the MLP is able to solve nonlinear problems. In our case a 3-layer MLP was implemented, as described below.

Architecture of network:

- Input layer: 784 nodes.
- Hidden Layer: 100 nodes.
    - Fully connected to input nodes.
    - Sigmoid activation for non-linearity.
    - Glorot_uniform (Xavier uniform) initialiser.
- Output layer: 10 nodes.
    - Fully connected to hidden nodes.
    - Softmax activation for classification.
    - Glorot_uniform (Xavier uniform) initialiser.
- Total parameters: 79,510
- Hyperparameters
    - Optimiser: Stochastic Gradient Descent
    - Learning rate: 0.5
    - Loss function: Categorical Cross Entropy
    - Batch Size: 300
    - Number of Epochs: 10

Initialiser:
To avoid having weights/gradients that either diminish or explode as they propagate through the network, the Glorot method is used to allow for scaling of the weight distribution on a layer-by-layer basis. The Glorot_uniform initialiser draws samples from a uniform distribution with a centered mean, and a standard deviation scaled to the layer's number of input and output neurons. This keeps the weights in a reasonable range across the entire network. The Glorot method performs better than random or constant (e.g. zeros or ones) initialisations, resulting in lower initial loss, faster convergence and fewer epochs.

Model Summary:

```
Layer (type)                 Output Shape              Param #
=================================================================
fc1 (Dense)                  (None, 100)               78500
_________________________________________________________________
activation_1 (Activation)    (None, 100)               0
_________________________________________________________________
fc2 (Dense)                  (None, 10)                1010
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

Results:

- Five iterations were performed to account for the random initialisation of weights by Keras:
    - Iteration 1: loss: 17.68%, acc: 94.94%
    - Iteration 2: loss: 17.84%, acc: 94.91%
    - Iteration 3: loss: 18.28%, acc: 94.67%
    - Iteration 4: loss: 17.79%, acc: 94.94%
    - Iteration 5: loss: 17.13%, acc: **95.01% (best)**

Conclusions:

- Adding a hidden layer causes the number of parameters to increase dramatically by around 10-fold, to 79,510.
- There is a marginal increase of ~3% points in accuracy compared to the network consisting of only a single softmax layer. This is a poor result considering the large number of parameters added.

Exercice 3 : Convolutional Neural networks with Keras

From Exercise 2 we can intuit that Neural Networks receive an input (unidimensional vector) and transform it through a series of hidden layers to an output layer that represents the class scores. It is important to point out that a fully connected structure will not efficiently scale for images of a more greater size, for example 500x500x3. This would lead to neurons having 500*500*3 = 750,000 weights each, representing an exponential increase in the number of parameters. As a result, this full connectivity is not only computationally wasteful, but the huge number of parameters can easily causing overfitting.

Convolutional Neural Networks (CNNs) take advantage of the data structure of the input images, and this allows us constrain the architecture in a more adapted way. In particular, unlike a regular Neural Network, the layers of the CNNs have neurons arranged in 3 dimensions: **width, height, depth**. (where *depth* refers to the third dimension of an activation volume, not to the depth of a full Neural Network). These particular properties in the architecture make the forward function more efficient to implement and considerably reduce the amount of parameters in the network.

Architecture of network:

- Input layer: 28*28*1 tensor
- Convolutional Layer 1: 32 x 5*5 filters
    - ‘Same’ padding, to ensure output is the same size as the input
    - Sigmoid activation for non-linearity.
    - Glorot_uniform (Xavier uniform) initialiser.
- Pooling Layer 1: 2*2 spatial aggregation layers (pooling) to provide invariance to local translations
- Convolutional Layer 2: 64 x 5*5 filters
    - ‘Same’ padding, to ensure output is the same size as the input
    - Sigmoid activation for non-linearity.
    - Glorot_uniform (Xavier uniform) initialiser.
- Pooling Layer 2: 2*2 spatial aggregation layers (pooling) to provide invariance to local translations
- Flattening Layer: To convert 3D tensor into a 1D, 3136 dimension vector.
- Hidden Layer: 100 nodes.
    - Fully connected to input vector nodes.
    - Sigmoid activation for non-linearity.
    - Glorot_uniform (Xavier uniform) initialiser.
- Output layer: 10 nodes.
    - Fully connected to hidden nodes.
    - Softmax activation for classification.
    - Glorot_uniform (Xavier uniform) initialiser.
- Total parameters: 366,806
- Hyperparameters
    - Optimiser: Stochastic Gradient Descent
    - Learning rate: 0.5
    - Loss function: Categorical Cross Entropy
    - Batch Size: 300
    - Number of Epochs: 10

Model Summary:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        51264
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
fc1 (Dense)                  (None, 100)               313700
_________________________________________________________________
activation_1 (Activation)    (None, 100)               0
_________________________________________________________________
fc2 (Dense)                  (None, 10)                1010
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 366,806
Trainable params: 366,806
Non-trainable params: 0
_________________________________________________________________
```

Explanation of architecture:

- Convolutional Layer 1 and Pooling Layer 1:
    - We convolve 32 filters of 5x5x1 on the flat 28x28 images. This will **detect 2D spatial patterns** in the black and white image.
    - Each 5x5 filter is initialised with Glorot_uniform weights. These are updated later with backpropagation.
    - As we use 32 of these filters with 'same' padding, this gives us an output tensor of 28x28x32
    - We then pool this output using squares of 2x2 and taking the maximum value of each square. Using a stride of 2 gives an output of 14x14x32 (half the width and height, but still the same depth).
- Convolutional Layer 2 and Pooling Layer 2:
    - Next we convolve 64 filters of 5x5x32 with the output of the previous pooling. These filters are initialised with Glorot_uniform weights in all 3 dimensions. This allows us to detect correlations (patterns/relationships) **between the 32 layers** in the tensor produced by the 32 filters in the previous step.
    - The output tensor is now 64 'images' deep, due to 64 filters used i.e. we get an output tensor of 14x14x64
    - We again pool this output using squares of 2x2 using a stride of 2, which gives an output of 7x7x64 (half the width and height, but still the same depth).
- Flattening
    - We could continue to convolve and pool to detect higher level features and relationships, however we opt to simply flatten the resulting tensor to get to a 3136 dimension vector.
    - We then use this vector as the input into a fully connected network for classification.

Results:

- Loss: 6.75%
- Accuracy: **97.94% correct**. This is an increase of another 3% points from that obtained with the MLP.
- Only one iteration was calculated due to the high computational intensity of the algorithm. It may be possible to achieve marginally better results than these with repeated iterations.

Conclusions:

- The loss of 6.75% is much lower than that of the MLP (17.13%).
- The accuracy increase is due in part to the pooling stage, which provides invariance and robustness to deformation and translation. Handwritten digits vary enormously in size, shape and position, both between human writers and even for the same writer. The CNN is much better at coping with these differences than the MLP, as it takes into account the topology of the input data. This is due to the CNN exploiting the 2D spatial relationships in the data structure of the images, which the MLP cannot (MLP would give the same performance with a permuted image).
- The total number of weights are approximately four times the number used by the MLP. This represents an acceptable increase in relation to the increase in accuracy gained. The difference in the number of weights used by the CNN compared to the MLP is relatively low, as the CNN uses sparse connectivity and shared weights.

Deep Learning and Manifold Untangling

Exercice 1 : Visualisation with t-SNE

The t-Distributed Stochastic Neighbor Embedding (t-SNE) is a non-linear dimension reduction method which models any multidimensional observation into by a lower dimensional point (usually two or three) in such a way that close observations in the original space are represented by close points in the projected space and dissimilar observations are modeled by distant points.

Here we project the raw image data onto 2D space and label each of the points with the digit class it belongs to. In the next exercise we will visualise the distribution of these projections, and should expect to see points of the same class clustered together.

Please note: The GPU server was inaccessible, so to reduce computation time we used only the first 1000 images of the test set.

[t-SNE] Error after 1000 iterations: error = 1.0134439, gradient norm = 0.0000701 (50 iterations in 1.355s)

Exercice 2 : Visualisation and class separation metric

Visualising the ensemble of points projected in 2D can help us to define the right criteria to analyse the separability of the classes. To this end, three metrics will be computed:

Neighboring hit

- The neighboring hit (NH) metric uses the k-nearest neighbor algorithm to compute the similarity of the classes considering each point of the dataset.
- For each point, the NH metric consists in computing, for the k nearest neighbours of that point, the rate of neighbours that belong to the same class of the considered point. The NH metric es then normalized over the ensemble of the dataset.
- The neighboring hit provides a scalar value as metric where the closer to 1 represents a better separability between classes.

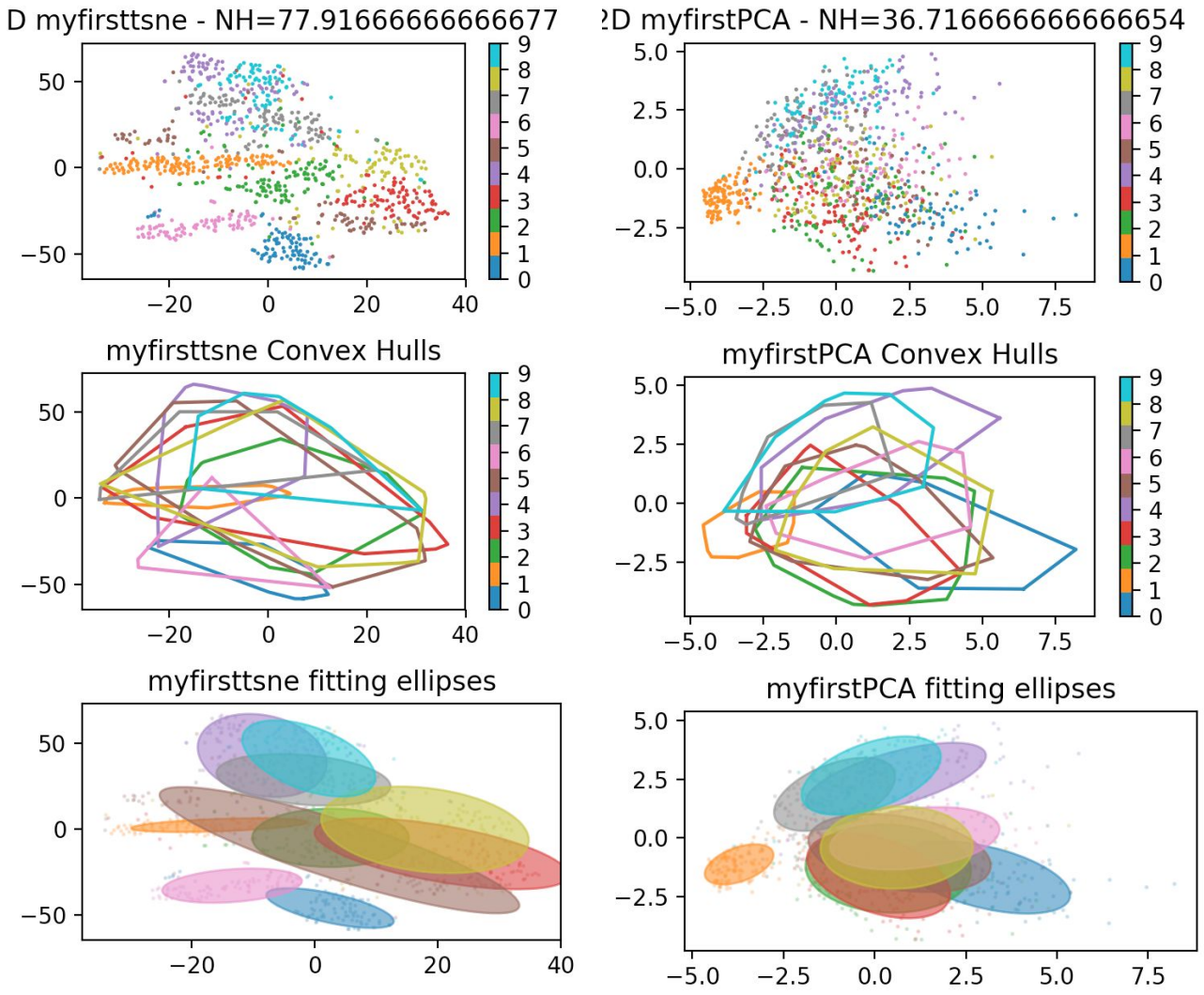
Convex hull

- The convex hull of a given set of points is just the smallest envelope set that contains all the points of the set.
- The convex hulls metric is vulnerable to outliers since it get strongly modified when there is at least one value spatially separated from the rest.

Approximation ellipse

- The approximation ellipse metric uses the gaussian mixture model and it assigns to the samples of the same class the Gaussian it mostly probably belong to.
- The ellipse metric contains the parameters of the gaussians for each one of the classes.

Here we display the three metrics calculated on the t-SNE projections, and the PCA projections for comparison.



Clear differences between the t-SNE and PCA projections are visible:

2D Cloud & Neighboring hit

- This can be seen well in the first visualization, as the cloud of points created by the non-linear t-SNE method is relatively well segregated into clusters of the same colour (class), each separated from the next by clear white space or a low level of intermixing.
- However, for the linear PCA method there is very little segregation, with a single multi-coloured cloud of points from all different classes mixed together.
- The neighboring hit metric is much lower when using PCA as the algorithm for dimensionality reduction (36.72 against 77.92). This shows that with PCA, there is a much greater mixing of projected points of different classes than there is with t-SNE i.e. the neighbours of a particular PCA point are much more likely to be of a different class.

Convex hull

- This visualisation is much less useful, as the convex hulls for almost all classes overlap significantly for both t-SNE and PCA projections.
- This is due to the outliers that are present for all classes in both point clouds, that stretch the convex hulls out over a large area, while the concentrations of the distributions of the points are not represented at all.
- Convex hulls can help us make sense of the point cloud, particularly the PCA cloud where the points are too mixed to see where the class boundaries occur.

- However, relying only on a convex hulls visualisation would lead us to believe that t-SNE and PCA are roughly equal in their ability to separate the classes, which we know not to be the case having examined the point cloud itself.

Approximation ellipse

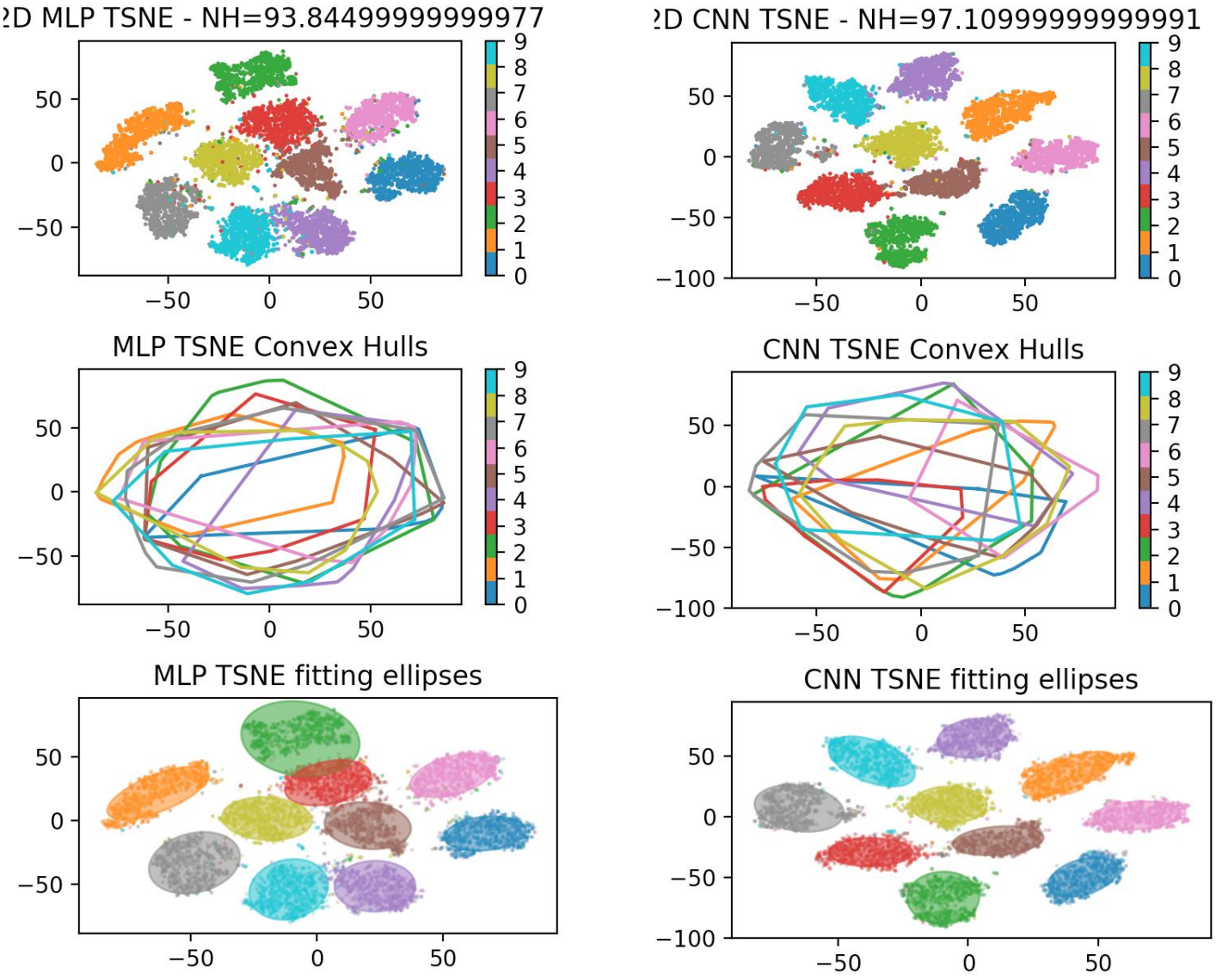
- The Gaussian approximation ellipses are a more reliable indicator of class separation than the convex hulls, as they take into account the distribution (mean and standard deviation in two axes) of each class, and are therefore resistant to outliers.
- We can see that for both t-SNE and PCA there are a number of ellipses that do not overlap at all, or do so only marginally.
- Consistent with other the other visualisations, there is more overlapping of ellipses for PCA points in the reduced space than for the t-SNE points.

The PCA method provides a lower-quality representation compared to the t-SNE method. This suggests that the t-SNE model is more effective at representing similar objects by nearby points and dissimilar objects by distant points than PCA method. This is likely due to t-SNE being a non-linear method, as opposed to the linear PCA method. However, both representations are less than ideal, with considerable overlapping due to the use of raw data.

Exercice 3 : Separability of classes and internal representations of neural networks

In neuroscience, the “manifold untangling” problem refers to the capacity of neural networks to separate examples of different classes in the space of learned representations. The ability to rapidly recognize objects of the same class, despite significant variations in appearance, is solved in the brain via a cascade of reflexive feedforward computations that culminate into powerful neuronal representations. CNNs are artificial, human-inspired models that are capable of classifying images. We will use CNNs to illustrate how the internal representation of these models are capable of recreating what human brains do when recognizing differents types of objects.

To illustrate this capacity, we used t-SNE to visualise the learned internal representations of the deep convolutional neural network, and for comparison, the fully connected multi-layered perceptron with one hidden layer that we trained in the previous exercises. These representations are the output values of the first dense fully connected hidden layer in both networks.



The strong capability of neural networks to solve the manifold untangling problem is clearly visible. Both neural networks offer a considerable improvement when compared with the raw data. The hidden layers are able to accurately extract features that allows a good discernment between classes, as seen by the clearly defined and well separated clusters in the 2D point cloud and the Gaussian ellipses.

The CNN is found to offer a better solution to the manifold untangling problem than the MLP, since the internal layers are capable of finding descriptors to better separate the classes.

- From the Neighboring Hit metric, we see that the groups are more homogeneous in the CNN than in the MLP.
- From the fitting ellipse metric, it can be observed that there is no overlapping at all in the CNN, whereas there is a small amount with the MLP. This means that the ellipses that best represent the points in each of the classes are spatially located in a way such that the ensembles can be identified more accurately compared to the MLP.
- Once again, the convex hulls representation offers us little in the way of comparison, with the outliers causing almost all hulls to overlap considerably for both methods.