MLP Coursework 2: Investigating the Implementation and Optimization of Convolutional Networks

s1604556

Abstract

Convolutional neural networks are commonly used for analyzing visual imagery. In this investigation, the focus is on the performance of the convolutional neural network on the CIFAR 100 dataset. Firstly, different implementations of the convolutional layer are being compared with regard to their computational efficiency and analysis is provided for the chosen implementation. Secondly, a "broken CNN model" is presented and this model is suffering from the problem that comes with being a deeper neural network. As the depth of the neural network increases, the network is exposed to problems like gradient vanishing/exploding and accuracy degradation. In this work, "Batch normalization" and "Residual network" are used to counter those problems and improve the performance of the model. The two methods are being compared in terms of their effectiveness in improving the "broken model". Furthermore, two methods are combined to create a baseline model and three regularization methods: L2 regularisation, data augmentation, and dropout are used to further improve the generalised performance of the baseline model.

1. Introduction

Deep learning is one of the machine learning technique that mimics what comes naturally to humans: to learn from examples. It was first theorised in the 1980s but only recently become useful as more labeled data and computing power are available. Deep learning is based on ANN(artificial neural network), the learning process can be supervised, semi-supervised and unsupervised.(1) An ANN is a computing system that is inspired by biological neural network and it is intended to mimic the functionality of the human brain(2). The network consists of a collection of artificial neurons, computation takes place in every single neuron and the results are passed to other nodes through layers. An ANN usually consists of multiple layers, there is one input layer, one output layer, and the layers between the two are hidden layers, when talking about Deep learning, the term "deep" refers to the number of hidden layers. A traditional ANN may only contain 2-3 hidden layers whereas a deep neural network can have hidden layers up to 150.

One of the most common type of deep neural network is convolutional neural network(3). A convolutional neural network that is made primarily for 2D inputs such as imagery inputs. When dealing with imagery inputs, regular neural network struggles with high-definition images. For example, an image with dimension 200x200 and 3 color channels would have 200x200x3=120,000 weight values, and as we have more neurons, this will start to add up and significantly reduce the learning speed. A convolutional neural network uses convolution technique to significantly reduce the number of parameters in the network hence increases training speed. In this work, different implementation of the convolutional layer is discussed and compared, Many problems that emerged with deeper network is discussed and two solutions are tested and compared. Furthermore, different regularization methods are discussed in terms of the improvement of the convolutional neural network.

2. Implementing convolutional networks

The core for convolutional layer is the convolution operation. It takes a input image matrix and a kernel to produce a feature map. The convolution process is explain in figure 1

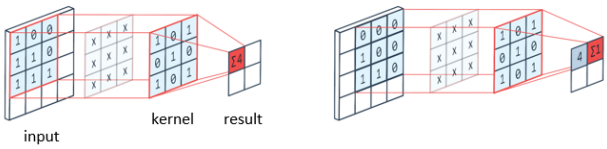

Figure 1. convolution operation

There are mainly four things in convolution operation, input, kernel, stride biases(biases are not required but usually used). In this example, the input matrix is of size 4x4 and kernel is 3x3, the result feature map will have a size of 2x2. The colored section in the source matrix is called the receptive field and its size will be equal to the size of the kernel. The two 3x3 matrix performs an element-wise multiplication, the results are summed up and bias is added in the end to get the first element of the feature map, Then the colour section is shifted to get the next receptive field, this process is repeated until the whole source image is covered. The stride value determines how much we shift each time, in this case, the stride value is 1 as we are only shifting by 1 pixel at a time. This convolution operation helps to reduce the dimension of the input as a larger image is now represented as a smaller feature map

hence the training speed is significantly increased.(3)

The above operation is only for 2-Dimensional inputs and kernels, in practical, convolutional layers are usually dealing with higher dimensions. This means that with one source image, it is possible to produce multiple feature maps with different kernels. For example, color images are usually represented with three channels, these can be three inputs channels for the convolutional layer, therefore for every single image, there are three inputs. For each color channel will have its own 3x3 kernel, applying each kernel to its corresponding input and obtain an intermediate output, the intermediate output is then summed up to obtain the final output for the input image, repeat the process for another set of 3x3 kernels to obtain another feature map(4). see Figure 3.

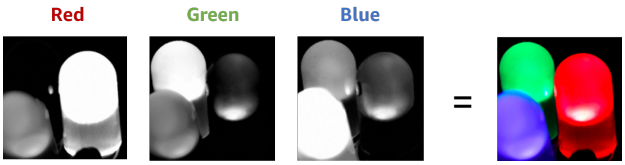

Figure 2.

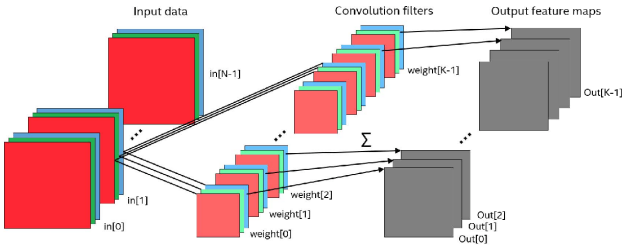

Figure 3. convolution operation in higher dimension

In this implement, both forward and backward propagation require multiplication of 4d tensors,there are multiple ways to achieve this(5):

- Explicitly loop over the dimensions: most straightforward way to implement, however nested loop is very inefficient and should be avoided if possible
- Serialisation: By replicating input and weight matrices, it is possible to perform the required 4D tensor multiplication into a large dot product. Depending on what the algorithm implemented it can have complexity from O($n^{(2.3728639}$), Le Gall (2014) to O($n^{(3}$), naive approach(6).
- Convolutions: use convolution functions that comes with Scipy package

In this work, convolutional layer is implemented using 3 for-loop and Scipy convolution functions. Compared to serialisation method, when the matrices are larger enough, serialisation can be more efficient. However, with smaller input matrices, my implementation is faster as reshaping in serialisation is more time-consuming. Storage-wise, serialisation method Will take up more storage as it replicates a lot of matrices, it is also a lot more difficult to implement as it requires careful manipulation of indices.

The convolve2D function comes with Scipy package is not the only function available, there are other packages that also have convolution function such as Pytorch. Function from different packages are not compared in this work, therefore it is hard to tell which one is more efficient.

3. Optimization problems in convolutional networks

The two convolutional neural networks used in this investigation are built and trained from the VGG network architecture developed by Visual Geometry Group from the University of Oxford. The VGG-08 is a working convolutional neural network that has 7 convolution layers and 1 fully connected layer. The VGG-38 is the "broken" network with 37 convolutional layers and 1 fully connected layer, it is "broken" because it is unable to minimise its train/validation loss. By intuition we know that extra layers means more abstraction power, parameters and capacity, so we should expect deeper model to be doing better than the shallow one, or at least can perform the same as the shallow counterpart, however this is not the case, the deeper model is performing poorly. For both training and validation set, during the training process, it gains little accuracy(around 1%) at start and quickly converges towards 0 and its loss stayed the same throughout the training. see Figure 4

Normally as a model gets trained it fits to the data and should reduce the loss and increase the accuracy until to a point you have over-fit the training data. However, looking at the graph and the collect metrics we can see that VGG-38 is not learning at all from the training data regardless of the epoch number. It is as if the model has never seen the data set and the weight is never been tuned for the dataset. There must be some problem during and backward propagation where the weights are updated incorrectly. The problem in this model is the vanishing/exploding gradient problem(7). We know backward propagation compute gradient by chain rule and with more layers more derivatives are being multiplied in the chain rule, so when the derivatives of the function is less than 1, the result of the chain rule is vanishing small, when the derivative of the function is greater than 1, the result of the chain rule gets larger goes to infinity. both of these will make front layers of a network to train very slowly or completely stop the neural network from training further. In the model architecture provided, Leaky ReLU is used to prevent this problem since the derivative of ReLu when x>1 is just 1, However, it does not fix this problem when the input is too large or too small as gradient with respect to parameter depending on both input and derivatives. The two solutions given in this work are Residual network and Batch normalisation, both are intended to prevent vanishing/exploding gradient problem.

3.1. Batch normalisation

Batch normalisation(8) was originally introduced in 2015 to address the *internal covariate shift* phenomenon where it reduces the covariate shift and improves the training speed of models with saturating nonlinearities. This will help with the vanishing/exploding gradient problem since the lower covariate shift means a more stable distribution of values, so no extreme large and small values.

To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation, However, after this operation, the weights in the next layer are no longer optimal. SGD (Stochastic gradient descent) will undo this normalization if that is how it minimises the loss. To bypass this problem, batch normalization introduces two more parameters gamma(standard deviation) and beta(mean), it allows SGD to reverse the normalization operation by allowing it changing only these two weights for each activation instead of all the weights. See Figure 4 for the algorithm

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 4. Batch normalisation algorithm

3.2. Residual network

We know that vanishing/exploding gradient problem prevent neural network training properly and hamper convergence from the beginning, and Batch normalisation have largely addressed such problem. However, when the deeper networks are able to start converging, its performance gets saturated or even starts degrading rapidly, this is the *degradation* problem come with deeper networks. Residual network(9) was first introduced in 2015 to deal with degradation problem. The main idea of ResNet is introducing a so-called “identity shortcut connection” that skips one or more layers, as shown in the following figure:

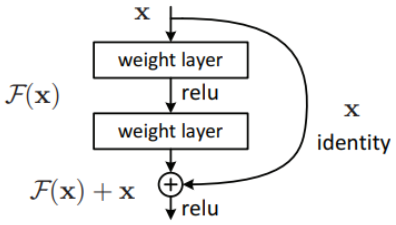

Figure 5. Residual network short cut

Let’s say the desired mapping we want is $H(x)$, we let the stacked layers fit the residual mapping of $F(x) := H(x)$-x. It is said that it is more difficult to optimize the unreferenced mapping $H(x)$ than residual mapping $F(x)$.With short cut carrying identity x and added to $F(x)$, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.

3.3. Comparison

The two methods given above can be both used to fix our "broken" model, however, they do so by different means. Batch normalisation fixes the problem by reducing the internal covariate shift so the gradient does not get too large or small. The residual network applies short cut technique which allows the first few layers to get update quickly, then the vanishing gradient is no longer a problem, and it also allows gradients to propagate further in a deeper network so training is more effective. In the experiment below Two methods are implemented separately, and the performance of two models on the validation set is compared to find out which method is more effective in fixing the vanishing/exploding gradient problem. Two methods are then combined to see if they work better together. The best performing model will be used as the baseline model and will be further improved on the generalisation performance using different regularisation.

4. Improving the generalization performance of convolutional networks

Generalisation performance refers to how well a model performs to new, previously unseen data, drawn from the same distribution as the one used to create the model.

When training a neural network, we usually have a training set, validation set, and testing set, where a model is first built from the training set then making fine-tuning to hyperparameter with the validation set. The testing set can then be used to give an unbiased estimate of the performance of the final tuned model. If a model is able fit to the training set as well as the testing set, then minimal overfitting has taken place. If we have a better fitting of the training set compared to the testing set, then this indicates we have overfitted the model to the training set, it means that the model is too complicated to generalise for new data hence the poor generalisation performance(10).

Therefore to increase the generalisation performance of a neural network, we need to prevent the model from overfitting while keeping the model well trained with the training/validation set. There are many regularisation methods available, these methods are chosen to improve the generalisation performance: L2 regularisation, data augmentation, and dropout. Three methods are implemented separately to compare their effect on the baseline model.

4.1. L2 regularisation(weight decay)

L2 regularisation(11), also known as Ridge it adds square of the magnitude of weight coefficient to the cost formula. It is like putting a "spring" on weights, If training data puts a consistent force on a weight, it will outweigh weight decay, If training does not consistently push weight in a direction, then weight decay will dominate and weight will decay to 0. L2 ensures that less important variables are reduced to very small value close to zero but not zero hence reduce the effective number of parameters, resulting in a simpler model hence no overfitting and better generalization performance.

4.2. data augmentation

Data augmentation(12) is a method to increase the diversity of data available for training models without actually having more training data. Generalisation performance increases with more training data since the model becomes more general by virtue of being trained on more examples. There are multiple data augmentation methods available, two methods are used here: Random rotation and Random erasing. Both of these are available in torchvision package, these two augmentations will be compared.

4.3. dropout

Dropout(13) is a method where it randomly deletes a fraction of hidden units in each minibatch. This means that their contribution to the activation of downstream units is temporally removed on the forward pass and any weight updates are not applied to the units on the backward pass. This makes each node more generalised and less dependent on other nodes as it needs to make predication for the missing node.

5. Experiments

The experiment are carried out using model VGG-38 and the data used is CIFAR-100 dataset. CIFAR-100 consists of 60,000 images where each image has 32x32 pixels and 3 RGB colour channels, therefore the input is of dimension 32x32x3, it has 100 classes, with 600 images per class. The training set used contains 47,500 images, the validation set contain 2,500 images and testing sets contain 10,000 images. The parameter used for VGG-38 are:

- batch_size=100
- seed=0
- num_filters=32
- num_stages=3
- num_blocks_per_stage=5
- experiment_name=VGG_38_experiment
- use_gpu=True
- num_classes=100
- block_type=conv_block
- Default learning rate and learning rate scheduler are used

5.1. Performance of the two optimisation solutions

Experiments are carried out to find out which one of the two optimisation solutions is better for our "broken" model. Result is shown in figure 6 and 7.

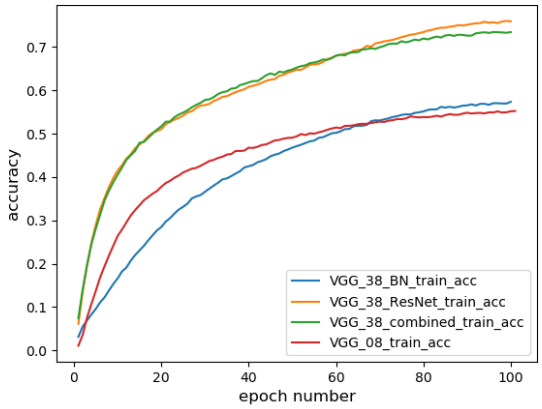

Figure 6. training set accuracy of batch normalisation and Residual net

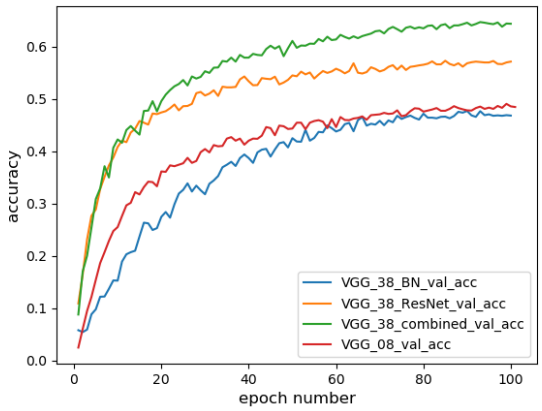

Figure 7. validation set accuracy of batch normalisation and Residual net

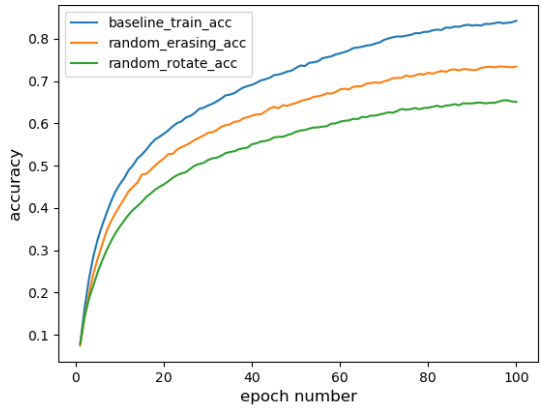

Figure 8. training set accuracy of different data augmentation compared to baseline

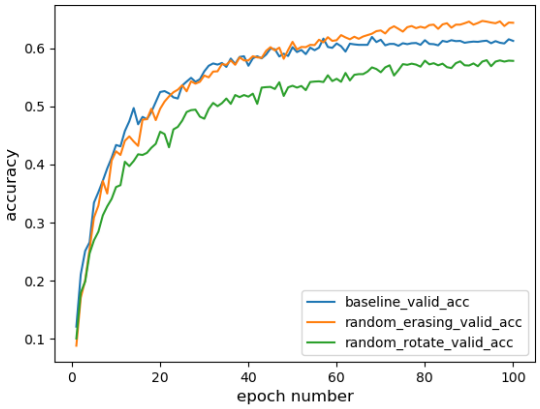

Figure 9. validation set accuracy of different data augmentation compared to baseline

As we can see from the figure 7, both bath normalisation and residual network are able to fix the "broken" VGG-38. All the curves of VGG-38 model converges as epoch number increases, and they are all performing better than the shallow counterpart which is what is initially expected from deeper network. The result is also showing that apply Residual network solely is better than applying batch normalisation network. This expected as Residual network fixes the vanishing/exploding gradient problem at the early stage of training and it also fixes degradation problem that occurs later in training, whereas batch normalisation is only capable of fixing the vanishing/exploding gradient problem. The result is showing that the combined optimisation solution performing the best,at epoch number 100, achieving 0.6128 valid accuracy and 0.842463 training accuracy. Which is expected as model utilises both methods to create an even more stable model. This combined model is used as a baseline in later experiments.

5.2. improve the generalisation performance

As mentioned above, when training accuracy is higher than testing accuracy we have a overfitting problem, it is clearly the case for our baseline model.

5.2.1. DATA AUGMENTATION

For Random rotating, the rotation angle is set to ±45 and random erasing is using the default p=0.5. From figure 8 and 9, we can see that both random rotating and erasing are helping with the overfitting problem. However, random rotating may have overdone its job and caused an underfitting of the training set which may explain what random rotating model is performing worse than the baseline model. The random erasing model is as expected, where the training accuracy is decreased by a little but validation accuracy surpasses the baseline model at around epoch 75. This is also true for testing set where random erase model got testing accuracy of 0.6312 compared to baseline's 0.6055

5.2.2. L2 REGULARISATION

since random erasing model improved the baseline model, we now compare random erase with L2 regularisation. The L2 coefficient used in this experiment is 0.001.

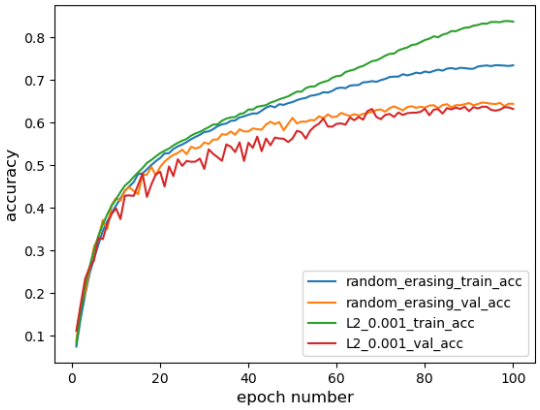

Figure 10. Model accuracy of random erasing and L2 regularisation compared

From Figure 10, we can see L2 coefficient of 0.001 is consistently worse than random erasing, and at epoch 100, L2 model is overfitting just like the baseline model. However, L2 model still manages to get a higher validation accuracy(0.632) compared to the baseline(0.6128). Furthermore, looking at the test set accuracy, L2 model scored 0.6365, which is better than both random erasing and baseline. This may indicate that the L2 model is not actually overfitting the training set, it just has better generalisation performance overall. L2 coefficient of 0.01 has also been experimented and it is performing very poorly with a validation set accuracy of 0.4572.

5.2.3. DROPOUT

In this experiment dropout is using the default value p=0.5, and it is implemented in the fully connected layer

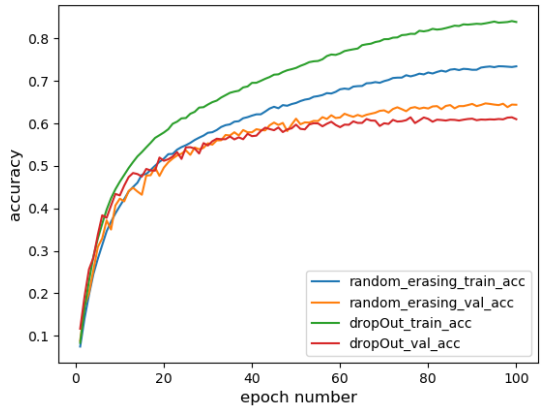

Figure 11. Model accuracy of random erasing and dropout compared

Once again, we can see from figure 11, random erasing is still the best regularisation method so far on the validation set. looking at the metric collected, dropoff model have almost identical training accuracy to the baseline, indicating it did not help much with the overfitting problem, this is also reflected with a test set accuracy of 0.6072

6. Discussion

From this series of experiment, we found that Residual network is significantly better than Batch normalisation when dealing with deep networks, the reason is as mentioned before, not only ResNet fix the vanishing/exploding gradient problem, it also fix the degradation problem, the overall performance of a deep network is improved and this is what residual network was originally proposed to do. It is no surprise that combining batch normalisation and Residual network result in an even better model(9). For regularisation methods, dropout did not improve any generalisation performance of the baseline model, since in some cases Batch normalisation eliminating the need for dropout(8), this may explain why dropout is not affecting the baseline model. Both random erasing and L2 regularisation have improved validation and testing set accuracy, random erasing is showing the expected curve on the graph, whereas L2 regularisation is showing signs of overfitting but manages to get the highest testing set accuracy of 6.3265. However, I think this is within the error range, random erasing is still overall a better regularisation method. Therefore the final choice of model is VGG-38 with batch normalisation , residual network and random erasing, with training set accuracy of 0.7345, validation set accuracy of 0.6440 and testing set accuracy of 0.6312

7. Conclusions

In this investigation, I have learned different methods of implementing convolutional layer, know the differences between each method in terms of computation efficiency. I have also learned problems with deeper networks such as vanishing/exploding gradient and degradation problem, the cause of each problem and learned two solutions: batch normalisation and residual network. Furthermore, I have learned that regularisation method is important for improving generalisation performance and know the reasoning behind each regularisation method.

Due to limited google cloud credit, I was not allowed to experiment with more different setups, for example, combining different regularisation methods should further improve the model performance. Also, there are two more optimisation method suggested not experimented: Densely connected convolutional networks(14) and Deeply-supervised nets(15), it is interesting to see if these can further improve our model.

References

[1] What is deep learning? | how it works, techniques applications, 2019.

[2] Josh. Everything you need to know about artificial neural networks, 12 2015.

[3] Cs231n convolutional neural networks for visual recognition, 2012.

[4] Thom Lane. Multi-channel convolutions explained with... ms excel!, 10 2018.

[5] Hakan Bilen. Convolutional networks 2: Training, deep convolutional networks, 2019.

[6] Thomas Sauerwald. Ii. matrix multiplication, 2016.

[7] Chi-Feng Wang. The vanishing gradient problem, 01 2019.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition.

[10] Will Koehrsen. Overfitting vs. underfitting: A conceptual explanation, 01 2018.

[11] Raimi Karim. Intuitions on l1 and l2 regularisation, 12 2018.

[12] Arun Gandhi. Data augmentation | how to use deep learning when you have limited data, 11 2018.

[13] Amar Budhiraja. Learning less to learn better—dropout in (deep) machine learning, 12 2016.

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Weinberger. Densely connected convolutional networks, 2018.

[15] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets.