MULTI-SCALE DENSE NETWORKS - REPRODUCIBILITY CHALLENGE

Alex, Gregory
ajg1u18

Cristian, Niculae
cin1g15

Sebastian, Gherhes
sag1g15

1 INTRODUCTION

MSDNet was introduced on ICLR 2018 by Gao Huang, and co. Huang et al. (2018b) and was received with high appreciation at the conference. Their idea was to create a neural network that once trained would have a good prediction on low resources. MSDNet's core idea is to contain multiple interconnected branches (see Figure 1) and to use multiple classifiers that would allow for an early forward propagation exit if a level of certainty is reached. This paper aims at replicating Gao Huang's MSDNet, and reproducing its results. An implementation of the network has been done from scratch and it can be seen in the organisation repo[1]. However, the paper misses out some details, so we imagine that our implementation may differ slightly from the actual implementation.

2 ARCHITECTURE

In general the MSDNet paper gave sufficient details on the structure of the network. However, the authors sometimes missed a few specific details. These details could mostly be found in the DenseNet paper Huang et al. (2016). In this section, we will discuss some of the more contentious areas of our implementation of MSDNet. We expect these areas will be different from the actual implementation of MSDNet.

2.1 BLOCKS

We struggled to understand exactly how the transition layer worked in the MSDNet paper. So, we implemented a design partly partly inspired by DenseNet (Huang et al., 2016). Figure 1 illustrates the typical structure of a block in our implementation. The final layer in the block is the transition layer. This layer takes the feature maps produced in the previous layer and applies a (1×1) convolution that halves the number feature maps that were produced in the previous layer. We included transition layers on every block besides the final one.

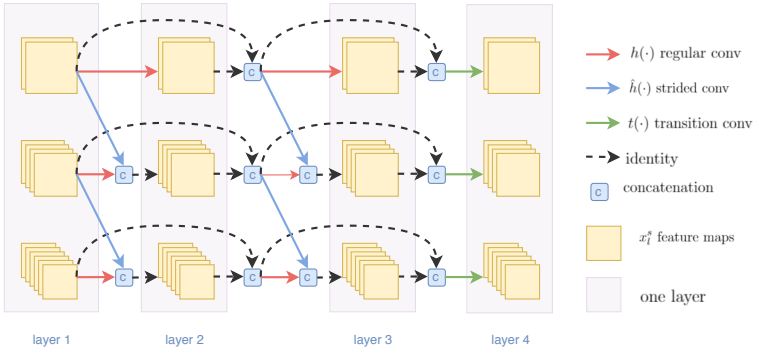

Figure 1: This diagram shows the structure of a typical block in out implementation of MSDNet. This block has 3 scales and a depth of 3 (we do not include the transition layer in the depth).

[1] https://github.com/COMP6248-Reproducability-Challenge/MSDNet-Reproducibility-Challenge

2.2 NUMBER OF FEATURE MAPS

One of the most difficult parts of the paper to replicate was the number of feature maps produced by each convolution, and the number of feature maps stored at each layer. We often relied on the DenseNet paper (Huang et al., 2016) to fill in some details in this area. For our implementation, we allowed the user to define a parameter k_0 which represented the number of feature maps produced by the first convolution in the network (in other words h_1^1 as defined in Huang et al. (2018a)). We then allowed the user to define a growth rate g that dictated how many feature maps were produced at each scale. The following equation gives the number of feature maps n produced by the convolutions at scale s.

$$n = k_0 \cdot g^s \quad \text{for} \quad s \in \{0, 1, 2, \dots\}. \tag{1}$$

A value of 6 was picked for the k_0 parameter, which is the same valued used by the authors of MSDNet. Table 1 shows the number of feature maps produced in a typical MSDNet with 3 blocks of depth 2.

	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$	$l=7$	$l=8$	$l=9$
$s=1$	6	12	18	9					
$s=2$	12	36	60	30	54	66	33		
$s=3$	24	72	120	60	108	156	78	126	150

Table 1: This table shows the number of feature maps produced at each layer of an MSDNet. The model has 3 blocks of depth 2, 3 scales and a k_0 value of 6.

2.3 CONVOLUTION DETAILS

MSDNet is made of two main types of convolutions; h and $\hat{h}$ convolutions. h convolutions have a stride of 1. $\hat{h}$ convolutions have a stride greater than 1. Equation (2) shows the general structure of the h and $\hat{h}$ convolutions where **BN** stands for batch normalisation. For $\hat{h}$ convolutions the (1×1) convolution has a stride greater than 1.

$$\textbf{CONV(1}\times\textbf{1)} \rightarrow \textbf{BN} \rightarrow \textbf{CONV(3}\times\textbf{3)} \rightarrow \textbf{BN} \tag{2}$$

Suppose (2) appears at scale s. The number of feature maps produced by the (1×1) convolution are $4 \cdot k_0 \cdot g^s$. The (3×3) convolution produces $k_0 \cdot g^s$ feature maps. This closely follows the same structure described in the Huang et al. (2017).

3 FASHIONMNIST

In order to reduce the amount of training required for our network, a decision has been made to test it using the FashionMNIST dataset. The reasoning behind this was that it is relatively easier to train compared to CIFAR10 (only needs a small number of epochs to get a good accuracy), and it was a more difficult dataset compared to MNIST. The training set consisted of 60000 images. We have used 10000 of these images as a validation set. The absence of dropout layers in the model is justified by the data augmentation that was performed on the images before training (Huang et al., 2018a). The data has been normalised with a mean of 0.5 and a standard deviation of 0.5. The training set has been augmented using a random centre crop with padding 2 and the images were flipped horizontally with a probability of 0.5.

3.1 POSITION OF THE CLASSIFIER

To test the effect that the position of an intermediate classifer has on the final classifier, we trained various models with classifiers placed at different positions and evaluated their performance on the test set. We used a network with 3 scales, and 3 equally sized blocks of depth 2. We trained the models for 15 epochs using stochastic gradient descent using the same settings that were used in the MSDNet paper on the CIFAR10 dataset. We started with a learning rate of 0.1, and then decreased it 0.01 after 5 epochs. The results can be seen in figure 2. From the left plot, we see that the accuracy

of the intermediate classifier tends to improve as it is moved deeper into the network. From the right plot, the accuracy of the final classifier seems to be almost independent of the position of the intermediate classifier.

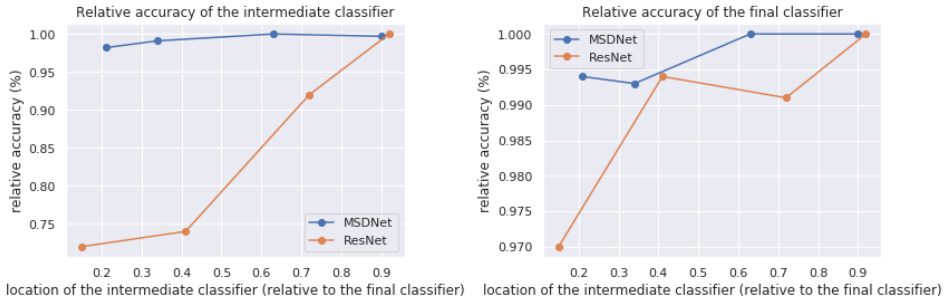

Figure 2: The figures show how the model is affected by the position of its intermediate classifier. The left plot shows the relative accuracy of the intermediate classifier with respect to its position. The right plot shows the relative accuracy of the final classifier with respect to the position of the intermediate classifier.

3.2 BUDGETED BATCH CLASSIFICATION

For this section, we used the same architecture as described in section 3.1 but trained it for 30 epochs and put a classifier at the end of each block. We used a learning rate of 0.1 and divided it by 10 at epoch 5 and 25. The results from training the model can be seen in Figure 2. From the left plot, the model appeared to not overfit. From the plot on the right, we found that the last classifier had the best accuracy at the end of training as expected. It can be seen in the mentioned figure that similarly with the original paper, other networks such as ResNet perform poorly on earlier classification, in our case obtaining even around 75% of the network's best intermediate accuracy. This makes such networks not so reliable on early classifications, showing that MSDNet, which on our example achieves 99% relative accuracy on the earliest classifier, being a more reliable network for early classifications. The MSDNet and ResNet used on the examples were trained accordingly to the paper.

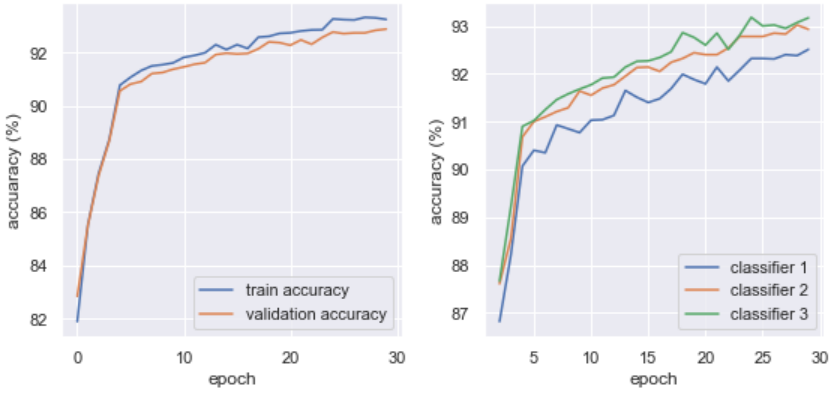

Figure 3: The left plot shows the average validation and training accuracy of all of the classifiers at each epoch. The right plot shows the validation accuracies of all of the classifers at each epoch.

For budgeted batch classification we simplified the problem somewhat. For each batch of images, we assigned a budget B. The budget was a time in seconds with which the images had to be classified by. We then evaluated the model on the test set using different budgets. If the budget for a batch was exceeded. The images that were not yet classified were counted as wrongly classified. The model has performed better as the budget was increased. The accuracies of the model has started to plateau at a budget value between 0.8 and 1.0, achieving an overall accuracy of about 90%.

4 CIFAR10

The network has been trained on the CIFAR10 dataset as well. The dataset consists of 60000 32x32 images, with 50000 of them being used for training and 10000 of them for testing. We have created a sample function to create a validation set of 5000 images that will be used to evaluate our model.

The parameters used for the network were the same as the ones in the original paper. We have trained the MSDNet over 300 epochs, using the same approach for data augmentation that was described in the FashionMNIST section. We have used a different mean ($[0.4914, 0.4824, 0.4467]$) and a different standard deviation ($[0.2471, 0.2435, 0.2616]$) for the transforms on the CIFAR10 dataset. The mean and standard deviation that were used for this transform was $[0.4914, 0.4824, 0.4467]$ and $[0.2471, 0.2435, 0.2616]$. These values were inspired by an MSDNet Github repo that has used these values to train the network (Kalviny). The initial learning rate was set at 0.1, but we have used a scheduler to update the learning rate at two specific milestones, 150 and 225. The learning rate was lowered to 0.01 and 0.001, which improved the accuracy of our model as our network was able to learn more from the input.

The best validation accuracy achieved after 300 epochs was 89.90545455. Figure 4 shows the accuracies achieved by the network during training. Even though we have used the transforms on the images to generalise the problem, our model managed to still overfit on the CIFAR10 dataset. We believe an extension to the initial paper would be to include a few dropout layers that could help with training the data and reduce overfitting.

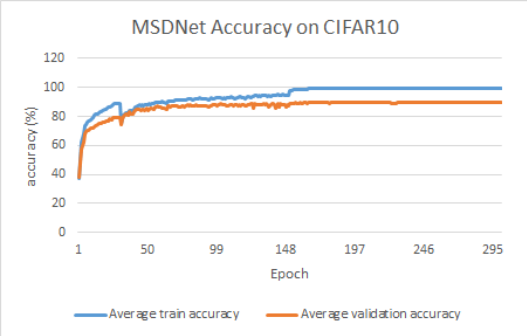

Figure 4: Training and validation accuracy of the MSDNet on the CIFAR10 dataset. Our training set achieves an overall trainig accuracy close to 100% (overfitting), while our validation stagnates at a 89.8 - 89.9 accuracy.

5 Conclusion

Even though the MSDNet paper omitted details regarding their implementation, we believe that their submitted article is fully reproducible. We have managed to duplicate their results and our implementation is very close to their description. Some issues we have encountered were mainly related to the fact that certain things were not mentioned in the paper and it relied on the authors i.e us to know the DenseNet paper in order to better understanding how the network worked. For that reason, we believe that both papers are needed in order to reproduce the results.

REFERENCES

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL http://arxiv.org/abs/1608.06993.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Densely connected convolutional networks, Jul 2017. URL https://www.youtube.com/watch?v=-W6y8xnd--U.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018a. URL https://openreview.net/forum?id=Hk2aImxAb.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018b. URL https://openreview.net/forum?id=Hk2aImxAb.

Kalviny. kalviny/msdnet-pytorch. URL https://github.com/kalviny/MSDNet-PyTorch/blob/master/dataloader.py.

A Appendix

One of the main features of MSDNET is that it is a network that allows for multiple classifiers. The network is able to produce predictions at multiple points throughout the network and potentially exit forward propagation early if it can classify an image with certainty above a threshold. To make an accurate classification, coarse level features are needed for a classifier. MSDNET ensures each classifier has coarse features to use in their classification by maintaining coarse representations of an input image throughout the network.

Figure 5 shows a typical MSDNET architecture. The network maintains representations of its input image at 3 different scales. Each row corresponds to a different scales. The bottom contains the coarsest representations of the image. Feature from the bottom row are the only features directly used by the classifiers.

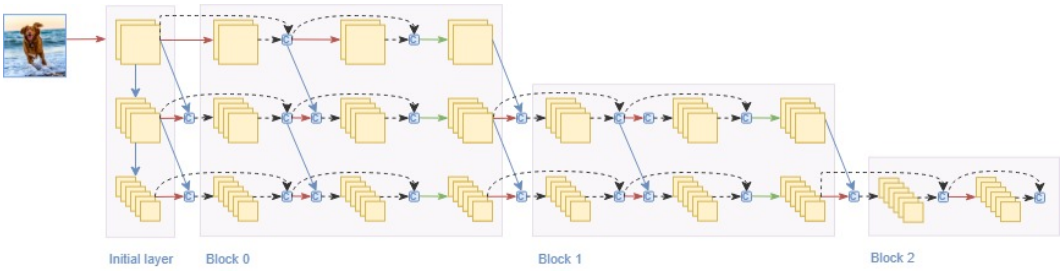

Figure 5: MSDNet architecture

A.1 Dense Connectivity

MSDNET builds on DenseNet described in Huang et al. (2016). MSNDNET sees concatenations so feature maps produced in by convolutions in earlier are maintained through later layers. This means that classifiers in later layers still use feature maps produced in earlier layers.

MSDNET uses concatenations so feature maps produced in previous layers at the same scale are maintained at throughout the network. Table 2 shows the dependencies of the convolutions in the network. The effect of this structure is that convolutions in later layers are directly connected to feature maps produced at earlier. The reason this is done is that including early classifiers means that feature maps at that layer are heavily optimised for that layer and may not be as useful for subsequent layers. This structure means that subsequent layers still have a direct connection to feature maps produced in previous layers before the classifier and thus feature maps optimised for an early classifier will not be as detrimental the classification accuracy of later classifiers.

	$l=1$	$l=2$	$l=3$	$l=4$
$s=0$	$h_1^0([x_0^0])$	$h_2^0([x_1^0])$	$h_3^0([x_1^0, x_2^0])$	$t([x_1^0, x_2^0, x_3^0])$
$s=1$	$\hat{h}_1^1([x_0^0])$	$\begin{bmatrix}\hat{h}_1^1([x_1^0]) \\ h_1^1([x_1^1])\end{bmatrix}$	$\begin{bmatrix}\hat{h}_3^1([x_1^0, x_2^0]) \\ h_3^1([x_1^1, x_2^1])\end{bmatrix}$	$t([x_1^1, x_2^1, x_3^1])$
$s=2$	$\hat{h}_1^2([x_1^1])$	$\begin{bmatrix}\hat{h}_2^1([x_1^1]) \\ h_2^1([x_1^2])\end{bmatrix}$	$\begin{bmatrix}\hat{h}_3^2([x_1^1, x_2^1]) \\ h_3^2([x_1^2, x_2^2])\end{bmatrix}$	$t([x_1^2, x_2^2, x_3^2])$

Table 2: This table shows which feature outputs the convolutions in figure 5 depend on. This table depicts the initial layer, and the block 0 from figure 5. The subscripts l and superscripts s represent the layer and scale respectively. h represents regular convolutions. $\hat{h}$ represents strided convolutions. t represents transition convolutions. $[\dots]$ represent convolution operations. The initial layer ($l=1$) seeds the images at different scales. Each row then produces features maps of the same size at each row. As the layers increase, the convolutions begin to depend on more previous layers. Convolutions in the transition layer then depend on all of the outputs of all of the convolutions at its respective row.

A.2 Loss Function

The loss function for this structure is an extension binary cross entropy. Let L_i be the cross entropy loss of the i-th classifier. The loss L of the entire network is a weighted sum of each loss L_i. The paper recommends using equal weight for all of the classifiers so we calculated used the mean of the losses L_i. So, we have

$$L = \frac{1}{n} \sum_{i=1}^{n} L_i \tag{3}$$

where n is the number of classifiers.