Case Study: VGG-16 vs DenseNet-121

Pablo García Fernández
pablo.garcia.fernandez2@rai.usc.es

***Abstract*—Convolutional neural networks (CNNs) have been widely used in computer vision tasks in recent years. There are many different architectures, each with its own characteristics. This work presents a comparative study between two of them: VGG-16 and DenseNet-121 in terms of accuracy, computational cost and transfer learning. Results show how DenseNet-121 outperforms VGG-16 in both performance (19 times fewer parameters, 1.3 times faster) and accuracy (91.23 vs 93.32) on Fashion MNIST dataset. Regarding transfer learning, using the backbone of both models trained in Fashion MNIST is useful for extracting MNIST features (94.1, 92.2 acc. retraining only the dense final layers).**

***Index Terms*—DenseNet, VGG, MNIST, transfer learning.**

I. Introduction

In recent years, deep learning (DL) has become very widespread in research and has been incorporated in a variety of applications. Among the different DL algorithms, Convolutional Neural Networks (CNN) have been extensively applied in a range of different fields, including computer vision. Many convolutional architectures have been proposed over the years to process (classify, detect, segment, etc.) images. The objective of this work is to make a comparison between two of them: VGG-16 and DenseNet-121.

Taking MNIST and Fashion MNIST as datasets, this comparison will be performed from two approaches. In the first, we will train both models under the same conditions and only in Fashion MNIST to analyze their differences in terms of accuracy, memory requirements, training time and inference cost. In the second, we will use the trained models to apply transfer learning techniques to achieve acceptable performance in MNIST and analyze the results.

To achieve these goals we will take advantage of the PyTorch framework.

II. Datasets

In this comparative report, two datasets are considered: MNIST and Fashion MNIST.

MNIST [1] is a large database of handwritten digits that is commonly used as bechmarking for different visual processing systems (especially machine learning models). It contains 60,000 training images and 10,000 testing images. Each of the images represents in a small square of 28×28 pixels a single handwritten grayscale digit between 0 and 9 as shown in "Fig. 1".

Fashion-MNIST is a direct drop-in alternative to the original MNIST dataset [2]. It consists of 60,000 training set pictures and 10,000 test set pictures. Each picture represents a garment in a gray-scale image of size 28×28, linked to one of the 10 categories shown in "Fig. 2".

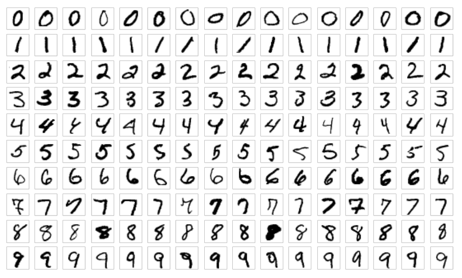


Fig. 1. MNIST dataset. Image extracted from [1]

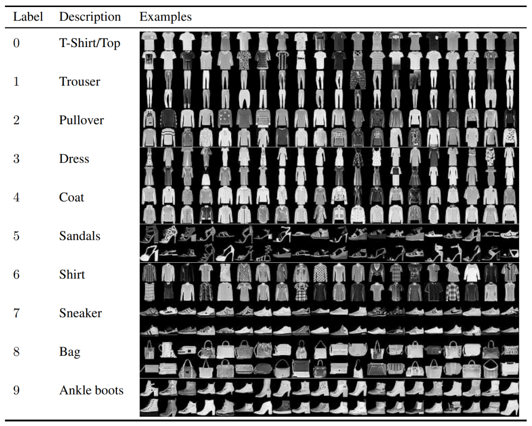


Fig. 2. Fashion MNIST Dataset. Image extracted from [2]

In this work, Fashion MNIST is the baseline for training and testing the models. MNIST will be used in the transfer learning task to test whether the features learned by the models in Fashion are general enough to run a classifier directly (or retraining the classifier part of the net) in MNIST.

III. Selected ConvNets

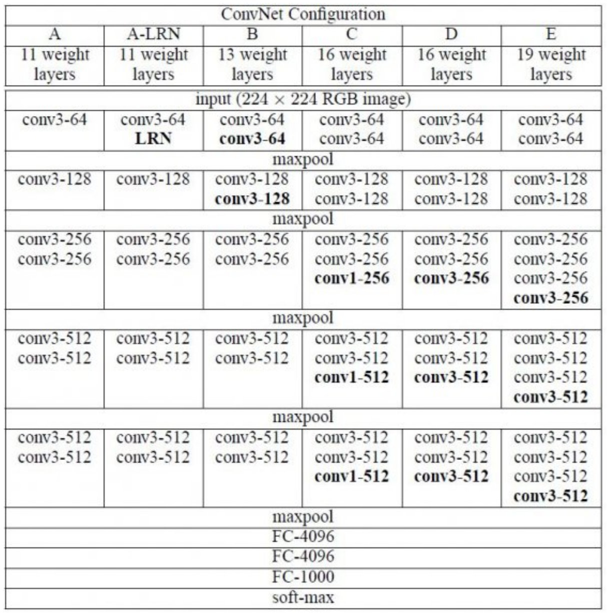
The models chosen for the comparative part are VGG-16 and DenseNet-121. Here, we briefly describe their general architecture and discuss the tricks we used to make them work on the mentioned datasets. This second question is summarized in: 1) modifying the last layer of both models to adjust the number of classes, and 2) resize the images to the input netwrok dimensions, so that they can be fed directly to the models.

The code for training and testing both models is developed in Python using the PyTorch framework. Specifically, we take advantage of the torchvision library to load the datasets and models. When loading the models we made sure not to use any pre-trained weights, so as not to spoil the comparison.

A. VGG-16

VGG-16 belongs to the VGG family of models introduced in [3]. The main difference between previous architectures is the replacement of the large filters with 3×3 stacked filters. It is easy to see that a two-layer 3×3 conv. stack has an effective receptive field of 5×5. This allows: 1) incorporating three layers of rectification layers instead of just one (which makes the decision function more discriminative), and 2) decrease the number of parameters (which may lead to a deeper architecture).

"Fig. 3" shows the configuration of 6 different VGG models. VGG-16 corresponds to the letter D. The input is fixed-size to a 224 × 224 RGB image. The image is passed through a stack of convolutional layers with small receptive fields (3×3). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution. Max-pooling is performed over a 2 × 2 pixel window, with stride 2.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fig. 3. Configuration of 6 VGG models, VGG-16 corresponds to D. Image extracted from [3]

Since the images in the dataset have dimensions of 28x28x1, we have to resize them to 224x224x3. First, we enlarge them to 224x224 with a simple interpolation method. To enlarge them to three channels, we repeat the resized grayscale image three times and concatenate the results on the depth axis. To perform these operations we use the `transforms.Compose` utilities from the torchvision library.

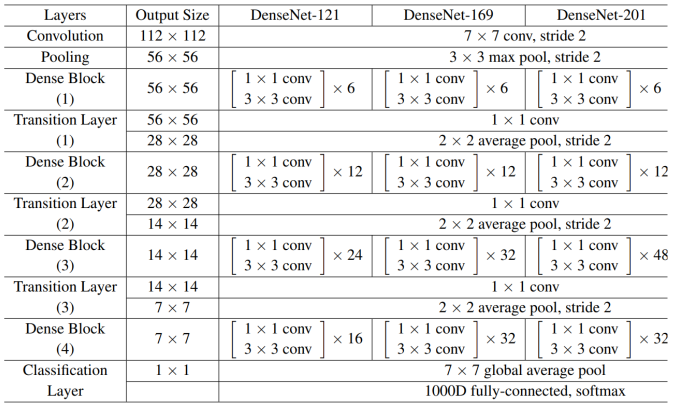
We also have to modify the architecture, to adapt it to the 10 classes of the MNIST Fashion. To do this, we change the output dimension of the last dense layer of the network: `vgg_model.classifier[6] = torch.nn.Linear(4096, 10)`.

Finally, mention that the Fashion MNIST data is loaded in a range of [0, 1] and then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225], as expected by the model.

B. DenseNet-121

DenseNet-121 belongs to the DenseNet family of models introduced in [4]. DenseNet models can be seen as a natural extension of ResNet where instead of adding the input to the mapping function, all of them are concatenated. DenseNet is made up of a set of dense blocks. Inside a dense block, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. Since each layer receives feature maps from all preceding layers, network can be thinner and compact, i.e. number of channels can be fewer. This allows better performance with less complexity

"Fig. 4" shows the configuration of 3 different DenseNet models. In this work we use DenseNet-121, which is compose of 4 dense blocks on 224×224 input imagess. The 1×1 convolutions (to modify the depth) followed by 2×2 average pooling are used as the transition layers between two contiguous dense blocks. At the end of the last dense block, a global average pooling is performed and then a softmax classifier is attached.



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201
Convolution	112 × 112	7 × 7 conv, stride 2		
Pooling	56 × 56	3 × 3 max pool, stride 2		
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv		
	28 × 28	2 × 2 average pool, stride 2		
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv		
	14 × 14	2 × 2 average pool, stride 2		
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$
Transition Layer (3)	14 × 14	1 × 1 conv		
	7 × 7	2 × 2 average pool, stride 2		
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$
Classification Layer	1 × 1	7 × 7 global average pool		
		1000D fully-connected, softmax		

Fig. 4. Configuration of 3 DenseNet models, we use DenseNet-121. Image extracted from [4]

As in VGG-16, to run DenseNet-121 on Fashion-MNIST, the images must be resized (using the aforementioned strategy) and the last linear layer must be adapted to have an output of 10 classes (`model.classifier = torch.nn.Linear(1024, 10)`).

IV. Experimental results

Both networks are trained using Adaptive Estimation of Moments (ADAM) for 10 epochs **on Fashion MNIST**. The batch size is set to 32. The learning rate is fixed to 0.001 and the decay of the L2 weights to 0.0004. As a loss function we use the Cross Entropy. Data augmentation techniques were not considered because training times are already sufficiently long with the 60,000 images.

The hardware on which the experiments were conducted consists of 16 GB of RAM, AMD Ryzen 5800X 8-core processor and NVIDIA GeForce RTX 3070 GPU.

A. *Comparative between VGG-16 and DenseNet-121*

"Tab. I" shows a comparison of both models in terms of accuracy, memory requirements, training time, and inference cost. The number of parameters and the memory consumed are estimated using the torchinfo summary function. To count the number of layers we take into account all convolutional and pooling layers (e.g., within the first dense block of DenseNet-121 there are 6 3x3 convolutions and 6 1x1 convolutions; this counts as 12 layers). The forward/backward size is estimated by considering both the space needed to store the weights and the space needed to store the gradient to be used in backpropagation. Finally, the inference time is in both cases the time required to process the 10,000 test images in batches of size 32.

TABLE I
COMPARISON BETWEEN VGG-16 AND DENSENET-121

	VGG-16	**DenseNet-121**
top1-acc. (%)	91.23	**93.32**
No. parameters	134,301,514	**6,964,106**
Param size (MB)	537.21	**27.86**
No. layers	21	**126**
Forward/backward pass size (MB)	**3470**	5777
Training time (s/epoch).	511	**393**
Inference time (s)	30	**22**

"Fig. 5" and "6" show the evolution of the error function during training for VGG-16 and DenseNet-121 respectively.

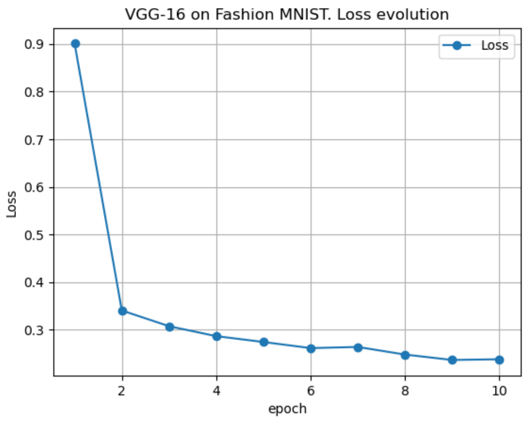


Fig. 5. Loss function evolution in VGG-16 training

Overall, we can see how DenseNet-121, with a much lighter architecture (19 times fewer parameters), is able to achieve better accuracy (91.23 vs 93.32). Of course, having fewer parameters means fewer operations and faster computation times both in training (511 vs 393 s/epoc), and testing (30 vs 22 s).

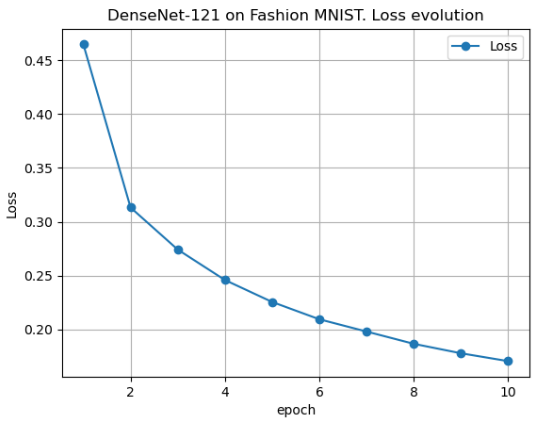


Fig. 6. Loss function evolution in DenseNet-121 training

Delving a little deeper into the results, in terms of memory requirements, since the DenseNet-121 parameters are smaller than those of VGG-16, the memory needed to store them is also smaller (537 vs 27 MB). However, since there are more layers in DenseNet-121 than in VGG-16 (i.e., Densenet-121 is a deeper model), there are more intermediate feature maps and the space needed to store the forward/backward results is larger.

Looking at "Fig. 5" and "6", we can see how DenseNet-121 is easier to train. The loss decreases faster (at epoch 4 it is 0.3 for VGG-16 and 0.25 for DenseNet-121) and drops to lower values (0.2381 vs 0.1704).

B. *Transfer learning from Fashion-MNIST to MNIST*

To perform the transfer learning task between Fashion MNIST and MNIST we do the following steps with both models:

1) We take the weights pretrained in FashionMNIST and load them into the model.
2) We freeze the convolutional part of both models. That is, we use them as a fixed feature extractor. To this end we set `requires_grad = False` the parameters so that the gradients were not computed during `backward()`.
3) We train on MNIST only the classification part (dense final layers) of both models. Having previously initialized them in a random manner.

The transfer learning training is carried out (for both models) with: Stochastic Gradient Descent (SGD) optimizer, 0.01 learning rate, 0.9 momentum, 64 batch size, and 10 epochs.

"Tab. II" shows the results. The first column contains the accuracy obtained by directly applying the model trained in Fashion-MNIST on MNIST. Logically, the results are very poor because the network was trained with very different data. The second column shows the accuracy obtained by keeping the convolutional part of the models fixed and retraining the classifier. In this case the results are quite good. The third column represents the accuracy that the model is able to obtain with the whole network trained on the new data (i.e. without transfer learning). The latter are taken from the literature and are used for comparison.

TABLE II
TRANSFER LEARNING COMPARISON

	acc. dir.	acc. transfer_l.	acc. ref	time (s/epoch)
VGG-16	13.3	**94.1**	+99.0	222
DenseNet-121	10.7	**92.2**	+99.0	154

From the results in the table we can conclude that this transfer learning strategy obtains very competitive results despite the large difference between the datasets (there is only a difference of 5 and 7 points with respect to directly training the models on MNIST). The main advantage is the saving of resources. Instead of training a new model, only the last classification layers are modified. This allows much shorter training times.

The fact that the initial dataset (Fashion MNIST) was more complex surely helps to get good results even though the resemblance between the numbers and the clothing is poor. In fact, this same process from MNIST to Fashion MNIST would probably be worse.

Another interesting aspect is to check how the transfer learning for VGG-16 is better than for DenseNet-121, even though the latter had obtained a higher accuracy over Fashion-MNIST. This is probably because in DenseNet-121 the loss was reduced more than in VGG-16 making the model more adaptive to Fashion-MNIST and less generalizable for the other dataset (MNIST).

V. CONCLUSIONS

In this work, the VGG-16 and DenseNet-121 models have been comparatively analyzed. It has been observed that the latter, with a much lower complexity (in terms of number of parameters and time consuming) is able to obtain a higher accuracy on the Fashion MNIST dataset (91.23 vs 93.32).

It has also been observed that the transfer learning strategy of keeping the backbone fixed and only modifying the classifier parameters to adapt them to the new dataset is valid in this case. Although the resemblance between numbers and clothing is not very great, by using the models trained on the more complex dataset (Fashion MNIST) as a basis; the extrapolation of the extracted features is more general. This results in models that can be used with MNIST with acceptable confidence and with shorter training times.

REFERENCES

[1] LeCun, Y., Cortes, C., & Burges, C. (2010). MNIST handwritten digit database. ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist, 2.

[2] Han Xiao, Kashif Rasul, & Roland Vollgraf. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. ArXiv, abs/1708.07747.

[3] Karen Simonyan & Andrew Zisserman. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR 2015.

[4] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. Proceedings of the IEEE CVPR (pp. 4700–4708)