

# Deeper Networks for Image Classification

Shelly Srivastava

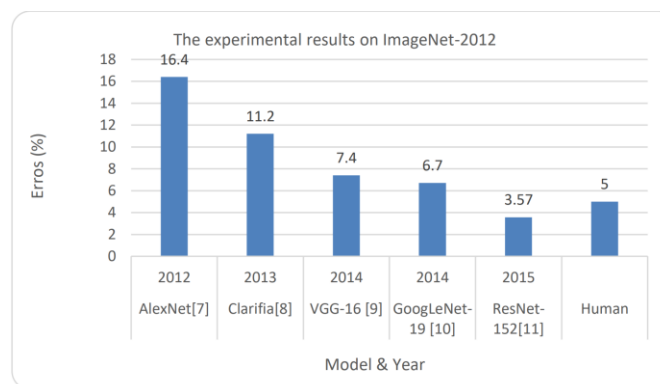
190385633

## 1. Introduction

Classical computer vision techniques cannot reach the human performance level at image classification/recognition tasks. On the other hand, deep neural network (which aims at imitating the biological brain and its neurons) can reach a comparable accuracy when it comes to image classification and can sometimes even outperform humans.

The aim of the project is to recreate and improve some of the established deeper networks and evaluate its performance on the MNIST handwriting dataset [1]. To further evaluate the performance of the model, other datasets are also used which have a similar composition such as Cifar-10 [18] and Fashion MNIST [2].

The use of Deep neural Networks began with Alexnet [3] in 2012 on Imagenet [7]. The error % (16.4) was the lowest at the time. Soon people began looking into deep convolutional neural networks. The figure below shows the error % of different networks since 2012 on Imagenet.



*Fig 1.1 Accuracy for imagenet challenge as given in [8]*

For the scope of the project, the deep models which are evaluated are Alexnet [3], VGG [4] and Resnet [5].

## 2. Literature Review

Since the publication of [3] and the introduction of Alexnet, the deep convolutional neural networks gained popularity for tasks which involved object recognition/ image classification. The CNN network structure was proposed by [9] in 1988 but was not used due to limitations of hardware. Deep CNN has many advantages over Deep neural network and one of the main advantages is that a CNN layer has much fewer trainable parameters than a fully connected layer and is very similar to human visual processing as said in [8].

There are many deep CNNs available and all of them share a few basic layers like, convolutional layer, max pooling layer and fully connected layer. Some of them are LeNet [10], AlexNet [3], VGG Net [4], NiN [11] and all convolutional (All Conv) [12]. GoogleNet [13], Residual Net [5], Dense Net [14] and FractalNet [15] are also few of the architecture but are considered more efficient and advanced. GoogleNet and ResNet has been designed for large scale application and FractalNet is a variation of Resnet.

Alexnet used has many trainable parameters due to its large filter size. Later in VGG, the filter size was reduced and the number of layers were increased from 8 in Alexnet to 16/19 in VGG (VGG-16 and VGG-19). This improved the performance of the network and thus, VGG is now considered a general architecture [8]. There are papers that say "Deeper is better" [16, 17] but a deep network also comes with its disadvantages like diminishing gradient. To tackle this problem, ResNet was introduced which has a skip connection (skips over a few convolution layers) to make sure that during back propagation, the error signal doesn't diminish.

There are many variants of Resnet which have been proposed over the years. Some of them are ResNeXt [19], DenseNet [14], Deep Network with Stochastic Depth [20], and Wide Resnets [21] etc. The Resnet architecture helps in having a network with even 1001 layers (Resnet-1001).

The above models perform very well on the MNIST dataset. The current state of the art result is obtained by [22], published in Jan 2020. The accuracy for a single model is 99.79% which also uses convolution with other architecture designs.

### 3. Model Description

A total of three models were used for the project. The model description and architecture are given below.

#### i. Alexnet

Alexnet as described by [3] is an 8 layers network with 5 convolution layers and 3 fully connected layers. The first conv layer uses 96 filters of size 11x11 with a stride of 4. The second layer uses 256 filters of size 5x5 with a stride of 1. The third and fourth layer uses 384 filters of size 3x3 each and the last conv layer uses 256 filters of size 3x3. The fully connected layer uses the output of the last conv layer and has 4096 units. The second fully connected layer also has 4096 units and the last layer has 10 units (number of classes). All the layers use ReLu activation except the last layer; it uses softmax activation.

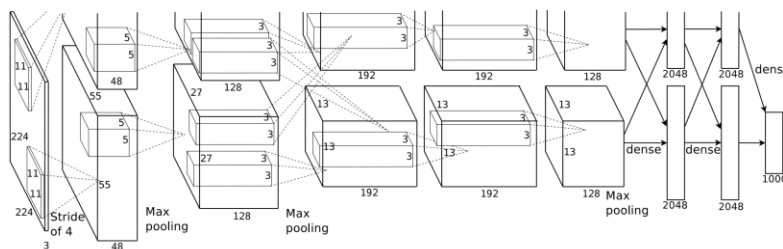


Fig. 3.1: Alexnet Architecture as proposed by [3]

#### ii. VGG-16

VGG-16 was originally proposed by [4]. This model has 16 layers as defined in the Cov-net configuration – model D in [4]. The first two layers have 64 filters with a kernel size of 3. The next two layers have 128 filters (kernel\_size = 3). The next 3 layers have 256 filters with the same kernel size and then the next 6 layers have 512 filters with the same kernel size. The next two layers are the fully connected layers with 4096 units and the last layer is output layer with softmax activation (all the other layers have ReLu activation).

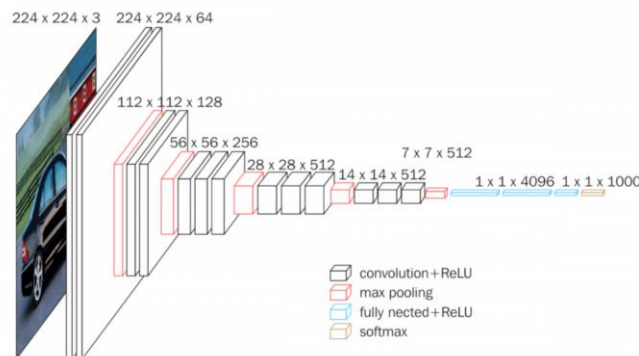


Fig 3.2 VGG-16 architecture as proposed by [4]

#### iii. Resnet-34

The Resnet models were introduced by [5] in 2015. The model has two kinds of blocks, one is called identity block and the other is called the convolution block. The identity block is used when the dimensions of the feature maps doesn't

change in the skip connection. The convolution block is used when the dimensions of the feature maps changes (due to map pooling) for the skip connection. In the figure 6.1 (in appendix), the dotted lines (skip connection) are conv\_block and the solid line skip connection is the identity block. The resnet has 3x3 convolutions except the first layer which is a 7x7 convolution. The last layer is a fully connected layer with 10 units (number of classes) and softmax activation. The original architecture uses ReLu as the Activation function for all other layers. To optimize the performance of resnet-34, LeakyRelu is used instead of ReLu. This improvement boosts the test accuracy by 3-4%.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10 <sup>9</sup>	3.6×10 <sup>9</sup>	3.8×10 <sup>9</sup>	7.6×10 <sup>9</sup>	11.3×10 <sup>9</sup>

Fig 3.3: Resnet architecture as proposed by [5]

## 4. Experiment

### i. Datasets

The dataset used for the project are MNIST, Fashion MNIST and CIFAR-10.

Dataset Name	# Training images	# Test images	Shape of image	# Classes
<b>MNIST</b>	60,000	10,000	28x28	10
<b>Fashion MNIST</b>	60,000	10,000	28x28	10
<b>Cifar-10</b>	50,000	10,000	32x32x3	10

Table 4.1: # of sample and size of the images in MNIST, Fashion\_MNIST and Cifar-10

MNIST, proposed by [1], has 60,000 training images and 10,000 test images. The shape of the grayscale images is 28x28 pixels. The dataset contains handwritten digits which have been size-normalized and centered in a fixed-size image. This dataset is a part of a bigger dataset called NIST. It is widely used for learning techniques for pattern recognition.

Fashion MNIST was proposed by [2] as a drop-in replacement for MNIST dataset. It has the same size and the same number of images for testing and training as the MNIST dataset. It is based on the fashion products on Zalando's website. It has 10 classes namely T-Shirt/Top, Trouser, Pullover, Dress, Coat, Sandals, Shirt, Sneaker, Bag and Ankle boots. It is proved to be a more challenging alternative than MNIST.

CIFAR-10 [18] is comparatively a smaller dataset than MNIST or Fashion MNIST. It has a total of 60,000 (50,000 train and 10,000 test images) images. The size of the RGB images is 32x32x3 and the dataset contains 10 classes. The classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. There are 6000 images per class. The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset.

### ii. Experimental Setup

The dataset is normalized for the experiment. The images are resized to 64x64xchannels and then normalized by mean subtraction.

The dataset described above were not sufficient and hence data augmentation was used to increase the variety of data, thus helping the model in generalization. There was an increase of about 10% in testing accuracy when the data was

augmented before training. The data augmentation performs a height and width shift of the pixels with random horizontal flips.

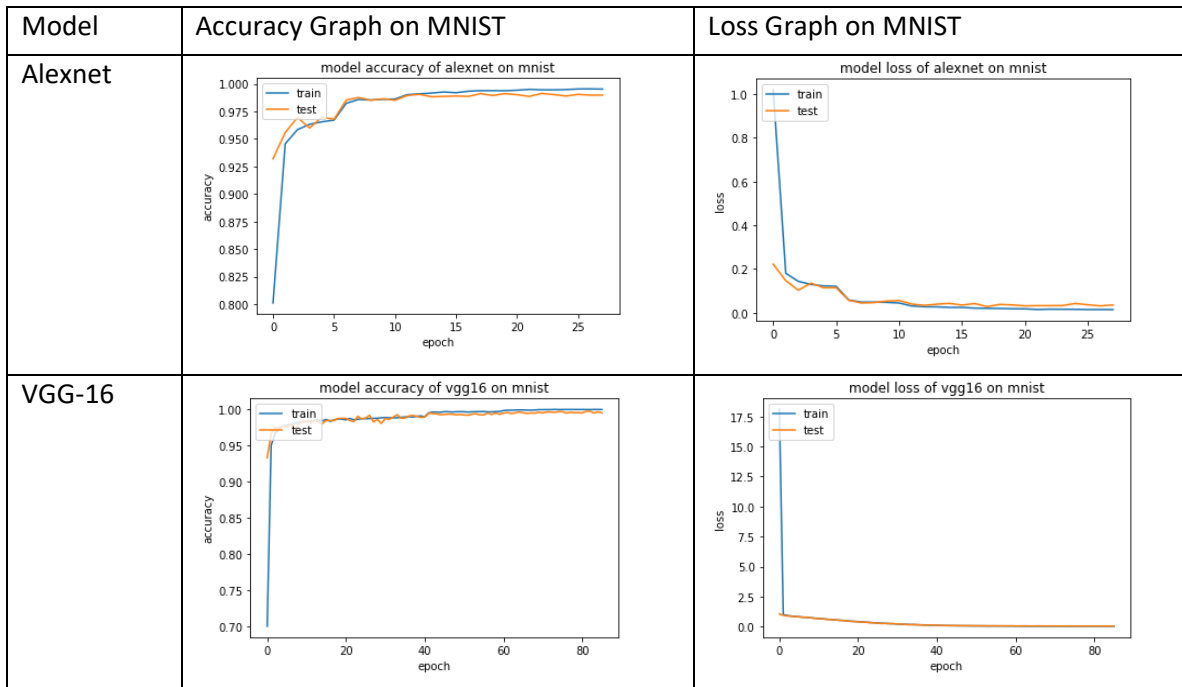
The all the models above use Adam optimization [6]. The original models used the Stochastic Gradient Descent. The loss function of the models is categorical\_crossentropy which is used for multiclass classification. The training procedure uses a call back for early stopping if the val\_loss is not decreasing after some defined epochs. The learning rate is decreased by using ReduceLROnPlateau. The LR is decreased by a given factor if the val\_loss doesn't decrease for a certain number of epochs.

The batch size is set to 128 and the validation split is of 10%. The max number of epochs allowed is 100.

### iii. Testing Results

Model	Dataset	Train Acc	Train Loss	Val Acc	Val Loss	Test Acc	Test Loss
Alexnet	MNIST	99.53	0.0149	98.98	0.0359	99.22	0.0244
Alexnet	Fashion MNIST	92.39	0.1986	91.03	0.2550	90.86	0.2755
Alexnet	Cifar-10	78.62	0.6036	73.28	0.7797	73.86	0.7612
VGG-16	MNIST	99.92	0.0348	99.45	0.0552	99.56	0.0516
VGG-16	Fashion MNIST	96.64	0.1723	93.93	0.2795	93.46	0.2913
VGG-16	Cifar-10	98.41	0.2835	89.60	0.6947	89.02	0.7009
Resnet-34	MNIST	99.74	0.0378	99.50	0.0468	99.58	0.0453
Resnet-34	Fashion MNIST	94.16	0.2324	92.93	0.2862	92.23	0.2942
Resnet-34	Cifar-10	92.93	0.3710	88.52	0.5229	89.28	0.5269

Table 4.2: Accuracy and Loss table for Alexnet, VGG-16 and Resnet-34 on MNIST, Fashion\_MNIST and Cifar-10



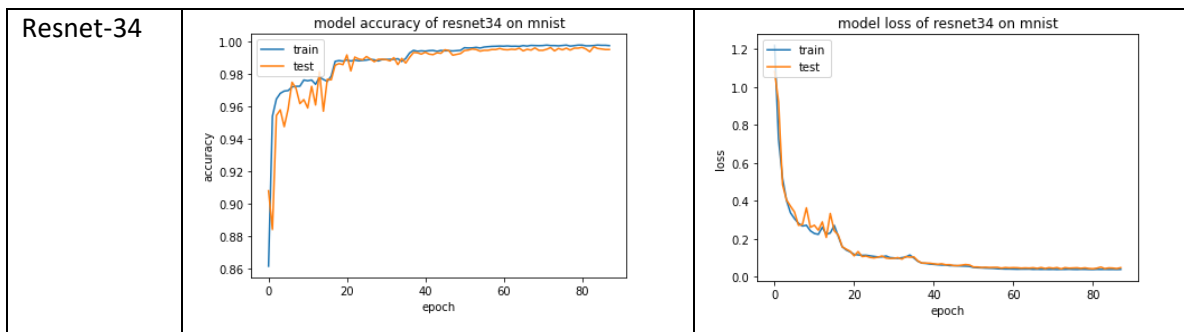


Table 4.3: Accuracy and Loss over the epochs for Alexnet, VGG-16 and Resnet-34 on MNIST

Table 4.3 shows the model accuracy and loss over the epochs for the different models on MNIST. Refer to Table 6.1 and 6.2 for the model's performance on Fashion\_MNIST and Cifar-10 in the Appendix section.

Model	Success Cases on MNIST	Fail Cases on MNIST
Alexnet	<p>Success Cases of alexnet on mnist</p>	<p>Fail Cases of alexnet on mnist</p>
VGG-16	<p>Success Cases of vgg16 on mnist</p>	<p>Fail Cases of vgg16 on mnist</p>
Resnet-34	<p>Success Cases of resnet34 on mnist</p>	<p>Fail Cases of resnet34 on mnist</p>

Table 4.4: Success and Fail cases for Alexnet, VGG-16 and Resnet-34 on Fashion\_MNIST

Table 4.3 shows the success and fail cases for the different models on MNIST. Refer to Table 6.3 and 6.4 for the model's performance on Fashion\_MNIST and Cifar-10 in the Appendix section.

#### iv. Evaluation

All the models have a test accuracy of more than 99% on MNIST. The MNIST dataset proved to be a simpler classification task as compared to Fashion\_MNIST and Cifar-10. The lowest test accuracy on Fashion\_MNIST is of 90% by Alexnet and the max accuracy is of 93% by VGG-16. Resnet is close with 92% accuracy. On Cifar-10, the lowest performance is of Alexnet with a test accuracy of 73% and the max accuracy is of 89% by Resnet and VGG. All these datasets are relatively very smaller than ImageNet [7].

None of the models are overfitting the MNIST dataset as it can be seen in the Table 4.3. As it can be seen in the table 6.1 and 6.2, the models overfit the Fashion\_MNIST and Cifar-10 dataset. As these models are deep models, it needs more data to learn a generalized function for a complex dataset.

It can be seen that Alexnet has a poorer performance. VGG-16 and Resnet-34 have a comparable performance on these small datasets. As stated before, Resnet is a more efficient models and thus works better on larger datasets such as ImageNet. There is a lack of data for the model to learn its trainable parameters. For smaller datasets the performance is similar for the accuracy.

It can also be seen in the Fig 6.5 and 6.8 that the VGG-16 model takes about 87s for one epoch and the Resnet model takes 40s. The Resnet model was using LeakyRelu instead and was still two times faster than VGG-16. The resnet model is faster in general than the VGG model.

#### **5. Conclusion**

The project focused on the implementation of original Alexnet, VGG-16 and Resnet-34 (with some improvements) to study the performance of these models on smaller datasets such as MNIST, Fashion\_MNIST and Cifar-10. Deeper models such as VGG and Resnet have a better performance than Alexnet. VGG and Resnet have a comparable performance in terms of test accuracy but VGG is much slower than Resnet during training. Moreover, all the models have a good test accuracy on MNIST (99 %).

## 6. Appendix

34-layer residual

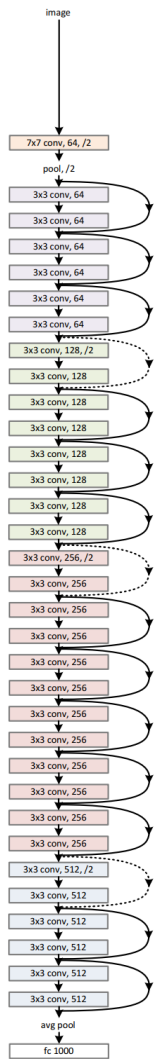


Fig 6.1 Resnet-34 Architecture

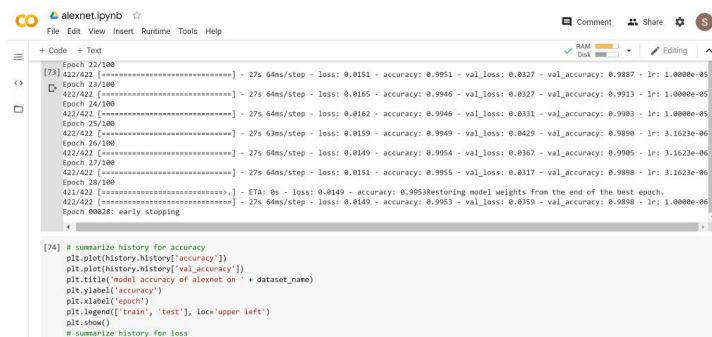


Fig 6.2: Alexnet on MNIST

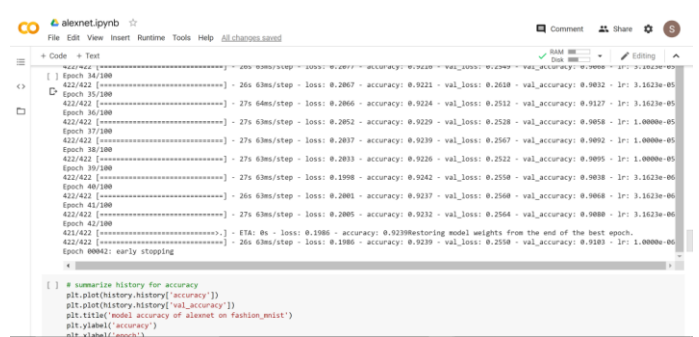


Fig 6.3: Alexnet on Fashion\_MNIST

```
alexnet.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[49] 52/52 [=====] - 52s 140ms/step - loss: 0.6027 - accuracy: 0.7882 - val_loss: 0.7668 - val_accuracy: 0.7496 - lr: 3.1623e-05
Epoch 50/100
52/52 [=====] - 52s 140ms/step - loss: 0.6090 - accuracy: 0.7867 - val_loss: 0.7744 - val_accuracy: 0.7362 - lr: 3.1623e-05
Epoch 51/100
52/52 [=====] - 52s 147ms/step - loss: 0.6091 - accuracy: 0.7851 - val_loss: 0.7817 - val_accuracy: 0.7356 - lr: 1.0000e-05
Epoch 52/100
52/52 [=====] - 52s 140ms/step - loss: 0.6027 - accuracy: 0.7874 - val_loss: 0.7678 - val_accuracy: 0.7392 - lr: 1.0000e-05
Epoch 53/100
52/52 [=====] - 52s 140ms/step - loss: 0.6045 - accuracy: 0.7876 - val_loss: 0.7779 - val_accuracy: 0.7372 - lr: 1.0000e-05
Epoch 54/100
52/52 [=====] - 52s 140ms/step - loss: 0.6003 - accuracy: 0.7886 - val_loss: 0.7915 - val_accuracy: 0.7278 - lr: 5.0000e-07
Epoch 55/100
52/52 [=====] - 52s 147ms/step - loss: 0.6019 - accuracy: 0.7892 - val_loss: 0.7784 - val_accuracy: 0.7342 - lr: 5.0000e-07
Epoch 56/100
52/52 [=====] - 52s 148ms/step - loss: 0.6054 - accuracy: 0.7866 - val_loss: 0.7762 - val_accuracy: 0.7336 - lr: 5.0000e-07
Epoch 57/100
52/52 [=====] - ETA: 0s - loss: 0.6036 - accuracy: 0.7862Restoring model weights from the end of the best epoch.
52/52 [=====] - 52s 140ms/step - loss: 0.6036 - accuracy: 0.7862 - val_loss: 0.7797 - val_accuracy: 0.7328 - lr: 5.0000e-07
Epoch 0000: early stopping
x

[50] # summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy of alexnet on ' + dataset_name)
plt.ylabel('accuracy')
plt.xlabel('epoch')
```

Fig 6.4: Alexnet on Cifar-10

```
VGG16.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[422/422] [=====] - 87s 207ms/step - loss: 0.0349 - accuracy: 0.9992 - val_loss: 0.0536 - val_accuracy: 0.9948 - lr: 1.0000e-06
Epoch 80/100
[422/422] [=====] - 87s 206ms/step - loss: 0.0348 - accuracy: 0.9992 - val_loss: 0.0535 - val_accuracy: 0.9948 - lr: 3.1623e-06
Epoch 81/100
[422/422] [=====] - 87s 205ms/step - loss: 0.0350 - accuracy: 0.9992 - val_loss: 0.0554 - val_accuracy: 0.9945 - lr: 3.1623e-06
Epoch 82/100
[422/422] [=====] - 87s 205ms/step - loss: 0.0351 - accuracy: 0.9991 - val_loss: 0.0466 - val_accuracy: 0.9965 - lr: 3.1623e-06
Epoch 83/100
[422/422] [=====] - 87s 205ms/step - loss: 0.0351 - accuracy: 0.9991 - val_loss: 0.0474 - val_accuracy: 0.9968 - lr: 1.0000e-06
Epoch 84/100
[422/422] [=====] - 86s 205ms/step - loss: 0.0347 - accuracy: 0.9993 - val_loss: 0.0536 - val_accuracy: 0.9943 - lr: 1.0000e-06
Epoch 85/100
[422/422] [=====] - 87s 205ms/step - loss: 0.0349 - accuracy: 0.9992 - val_loss: 0.0479 - val_accuracy: 0.9958 - lr: 1.0000e-06
Epoch 86/100
[422/422] [=====] - ETA: 0s - loss: 0.0348 - accuracy: 0.9992Restoring model weights from the end of the best epoch.
[422/422] [=====] - 87s 205ms/step - loss: 0.0348 - accuracy: 0.9992 - val_loss: 0.0552 - val_accuracy: 0.9945 - lr: 5.0000e-06
Epoch 0000: early stopping
x

[12] # summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy of vgg16 on mnist')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

Fig 6.5: VGG16 on MNIST

```
VGG16.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[422/422] [=====] - 132s 312ms/step - loss: 0.1874 - accuracy: 0.9015 - val_loss: 0.2922 - val_accuracy: 0.8343 - lr: 1.0000e-06
Epoch 79/100
[422/422] [=====] - 132s 312ms/step - loss: 0.1894 - accuracy: 0.9018 - val_loss: 0.2731 - val_accuracy: 0.8367 - lr: 1.0000e-06
Epoch 80/100
[422/422] [=====] - 132s 311ms/step - loss: 0.1845 - accuracy: 0.9016 - val_loss: 0.2835 - val_accuracy: 0.8343 - lr: 1.0000e-06
Epoch 81/100
[422/422] [=====] - 132s 312ms/step - loss: 0.1782 - accuracy: 0.9045 - val_loss: 0.2792 - val_accuracy: 0.8340 - lr: 3.1623e-06
Epoch 82/100
[422/422] [=====] - 132s 312ms/step - loss: 0.1776 - accuracy: 0.9041 - val_loss: 0.2734 - val_accuracy: 0.8397 - lr: 3.1623e-06
Epoch 83/100
[422/422] [=====] - 132s 312ms/step - loss: 0.1748 - accuracy: 0.9056 - val_loss: 0.2740 - val_accuracy: 0.8393 - lr: 3.1623e-06
Epoch 84/100
[422/422] [=====] - 132s 311ms/step - loss: 0.1740 - accuracy: 0.9054 - val_loss: 0.2795 - val_accuracy: 0.8393 - lr: 3.1623e-06
Epoch 85/100
[422/422] [=====] - 132s 311ms/step - loss: 0.1741 - accuracy: 0.9058 - val_loss: 0.2791 - val_accuracy: 0.8397 - lr: 3.1623e-06
Epoch 86/100
[422/422] [=====] - ETA: 0s - loss: 0.1723 - accuracy: 0.9048Restoring model weights from the end of the best epoch.
[422/422] [=====] - 132s 312ms/step - loss: 0.1723 - accuracy: 0.9064 - val_loss: 0.2795 - val_accuracy: 0.8393 - lr: 1.0000e-06
Epoch 0000: early stopping
x

[ ] # summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy of vgg16 on fashion_mnist')
```

Fig 6.6: VGG16 on Fashion\_MNIST

```
VGG16.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[352/352] [=====] - 113s 321ms/step - loss: 0.3217 - accuracy: 0.9700 - val_loss: 0.4683 - val_accuracy: 0.8950 - lr: 1.0000e-06
Epoch 82/100
[352/352] [=====] - 113s 322ms/step - loss: 0.3165 - accuracy: 0.9755 - val_loss: 0.4416 - val_accuracy: 0.8992 - lr: 1.0000e-06
Epoch 83/100
[352/352] [=====] - 113s 321ms/step - loss: 0.3121 - accuracy: 0.9769 - val_loss: 0.4611 - val_accuracy: 0.8908 - lr: 1.0000e-06
Epoch 84/100
[352/352] [=====] - 113s 322ms/step - loss: 0.3077 - accuracy: 0.9774 - val_loss: 0.4701 - val_accuracy: 0.8950 - lr: 1.0000e-06
Epoch 85/100
[352/352] [=====] - 113s 322ms/step - loss: 0.2983 - accuracy: 0.9803 - val_loss: 0.4595 - val_accuracy: 0.9010 - lr: 3.1623e-06
Epoch 86/100
[352/352] [=====] - 113s 322ms/step - loss: 0.2918 - accuracy: 0.9826 - val_loss: 0.4814 - val_accuracy: 0.8952 - lr: 3.1623e-06
Epoch 87/100
[352/352] [=====] - 113s 322ms/step - loss: 0.2914 - accuracy: 0.9820 - val_loss: 0.4916 - val_accuracy: 0.8928 - lr: 3.1623e-06
Epoch 88/100
[352/352] [=====] - 113s 322ms/step - loss: 0.2857 - accuracy: 0.9834 - val_loss: 0.4843 - val_accuracy: 0.8994 - lr: 3.1623e-06
Epoch 89/100
[352/352] [=====] - ETA: 0s - loss: 0.2835 - accuracy: 0.9845Restoring model weights from the end of the best epoch.
[352/352] [=====] - 113s 322ms/step - loss: 0.2835 - accuracy: 0.9841 - val_loss: 0.4847 - val_accuracy: 0.8966 - lr: 3.1623e-06
Epoch 0000: early stopping
x

[14] # summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
```

Fig 6.7: VGG16 on Cifar-10

```
resnet34.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Epoch 81/100
[422/422] [=====] - 41s 97ms/step - loss: 0.0372 - accuracy: 0.9977 - val_loss: 0.0413 - val_accuracy: 0.9963 - lr: 5.0000e-06
Epoch 82/100
[422/422] [=====] - 42s 100ms/step - loss: 0.0380 - accuracy: 0.9973 - val_loss: 0.0443 - val_accuracy: 0.9953 - lr: 5.0000e-06
Epoch 83/100
[422/422] [=====] - 41s 97ms/step - loss: 0.0376 - accuracy: 0.9973 - val_loss: 0.0501 - val_accuracy: 0.9937 - lr: 5.0000e-06
Epoch 84/100
[422/422] [=====] - 40s 98ms/step - loss: 0.0379 - accuracy: 0.9974 - val_loss: 0.0421 - val_accuracy: 0.9967 - lr: 5.0000e-06
Epoch 85/100
[422/422] [=====] - 40s 95ms/step - loss: 0.0375 - accuracy: 0.9977 - val_loss: 0.0458 - val_accuracy: 0.9957 - lr: 5.0000e-06
Epoch 86/100
[422/422] [=====] - 40s 96ms/step - loss: 0.0377 - accuracy: 0.9976 - val_loss: 0.0440 - val_accuracy: 0.9953 - lr: 5.0000e-06
Epoch 87/100
[422/422] [=====] - 40s 96ms/step - loss: 0.0374 - accuracy: 0.9976 - val_loss: 0.0423 - val_accuracy: 0.9958 - lr: 5.0000e-06
Epoch 88/100
[422/422] [=====] - ETA: 0s - loss: 0.0378 - accuracy: 0.9978Restoring model weights from the end of the best epoch.
[422/422] [=====] - 40s 96ms/step - loss: 0.0378 - accuracy: 0.9974 - val_loss: 0.0468 - val_accuracy: 0.9958 - lr: 5.0000e-06
Epoch 0000: early stopping
x

[12] # summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy of resnet34 on ' + dataset_name)
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Fig 6.8: Resnet-34 on MNIST

```
resnet34.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Epoch 82/100
[422/422] [=====] - 186s 252ms/step - loss: 0.2324 - accuracy: 0.9414 - val_loss: 0.2985 - val_accuracy: 0.9247 - lr: 5.0000e-06
Epoch 83/100
[422/422] [=====] - 186s 255ms/step - loss: 0.2311 - accuracy: 0.9421 - val_loss: 0.2929 - val_accuracy: 0.9228 - lr: 5.0000e-06
Epoch 84/100
[422/422] [=====] - 186s 255ms/step - loss: 0.2324 - accuracy: 0.9422 - val_loss: 0.2837 - val_accuracy: 0.9300 - lr: 5.0000e-06
Epoch 85/100
[422/422] [=====] - 186s 255ms/step - loss: 0.2342 - accuracy: 0.9411 - val_loss: 0.2884 - val_accuracy: 0.9243 - lr: 5.0000e-06
Epoch 86/100
[422/422] [=====] - 186s 255ms/step - loss: 0.2337 - accuracy: 0.9410 - val_loss: 0.2900 - val_accuracy: 0.9258 - lr: 5.0000e-06
Epoch 87/100
[422/422] [=====] - 186s 255ms/step - loss: 0.2324 - accuracy: 0.9410 - val_loss: 0.2884 - val_accuracy: 0.9243 - lr: 5.0000e-06
Epoch 88/100
[422/422] [=====] - ETA: 0s - loss: 0.2324 - accuracy: 0.9410Restoring model weights from the end of the best epoch.
[422/422] [=====] - 186s 252ms/step - loss: 0.2324 - accuracy: 0.9416 - val_loss: 0.2862 - val_accuracy: 0.9293 - lr: 5.0000e-06
Epoch 0000: early stopping
x

[27] # summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy of resnet34 on ' + dataset_name)
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Fig 6.9: Resnet-34 on Fashion\_MNIST

```
resnet34.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Epoch 57/100
[352/352] [=====] - 96s 272ms/step - loss: 0.3736 - accuracy: 0.9205 - val_loss: 0.5047 - val_accuracy: 0.8920 - lr: 3.1623e-06
Epoch 58/100
[352/352] [=====] - 96s 272ms/step - loss: 0.3744 - accuracy: 0.9208 - val_loss: 0.5095 - val_accuracy: 0.8952 - lr: 3.1623e-06
Epoch 59/100
[352/352] [=====] - 96s 272ms/step - loss: 0.3670 - accuracy: 0.9311 - val_loss: 0.5204 - val_accuracy: 0.8882 - lr: 1.0000e-06
Epoch 60/100
[352/352] [=====] - 95s 271ms/step - loss: 0.3698 - accuracy: 0.9306 - val_loss: 0.5057 - val_accuracy: 0.8908 - lr: 1.0000e-06
Epoch 61/100
[352/352] [=====] - 96s 272ms/step - loss: 0.3699 - accuracy: 0.9296 - val_loss: 0.5137 - val_accuracy: 0.8922 - lr: 1.0000e-06
Epoch 62/100
[352/352] [=====] - 96s 272ms/step - loss: 0.3692 - accuracy: 0.9287 - val_loss: 0.4987 - val_accuracy: 0.8948 - lr: 5.0000e-06
Epoch 63/100
[352/352] [=====] - 96s 272ms/step - loss: 0.3697 - accuracy: 0.9300 - val_loss: 0.5114 - val_accuracy: 0.8900 - lr: 5.0000e-06
Epoch 64/100
[352/352] [=====] - ETA: 0s - loss: 0.3710 - accuracy: 0.9293Restoring model weights from the end of the best epoch.
[352/352] [=====] - 96s 272ms/step - loss: 0.3710 - accuracy: 0.9293 - val_loss: 0.5229 - val_accuracy: 0.8852 - lr: 5.0000e-06
Epoch 0000: early stopping
x

[12] # summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy of resnet34 on ' + dataset_name)
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

Fig 6.10: Resnet-34 on Cifar-10



Model	Accuracy graph on Fashion_MNIST	Loss graph on Fashion_MNIST
Alexnet	<p>model accuracy of alexnet on fashion_mnist</p>	<p>model loss of alexnet on fashion_mnist</p>
VGG-16	<p>model accuracy of vgg16 on fashion_mnist</p>	<p>model loss of vgg16 on fashion_mnist</p>
Resnet-34	<p>model accuracy of resnet34 on fashion_mnist</p>	<p>model loss of resnet34 on fashion_mnist</p>

Table 6.1: Accuracy and Loss over the epochs for Alexnet, VGG-16 and Resnet-34 on Fashion\_MNIST

Model	Accuracy Graph on Cifar-10	Loss Graph on Cifar-10
Alexnet	<p>model accuracy of alexnet on cifar-10</p>	<p>model loss of alexnet on cifar-10</p>
VGG-16	<p>model accuracy of vgg16 on cifar-10</p>	<p>model loss of vgg16 on cifar-10</p>

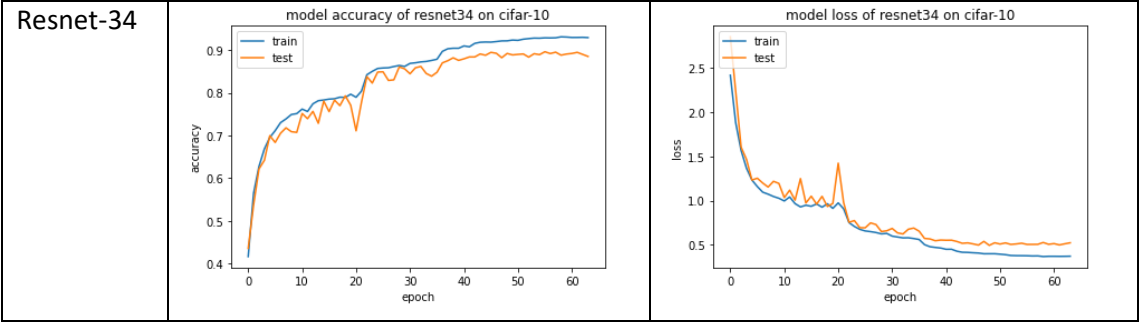


Table 6.2: Accuracy and Loss over the epochs for Alexnet, VGG-16 and Resnet-34 on Cifar-10

Model	Success Cases on Fashion_MNIST	Fail Cases on Fashion_MNIST
Alexnet	<p>Success Cases of alexnet on fashion_mnist</p>	<p>Fail Cases of alexnet on fashion_mnist</p>
VGG-16	<p>Success Cases of vgg16 on fashion_mnist</p>	<p>Fail Cases of vgg16 on fashion_mnist</p>
Resnet-34	<p>Success Cases of resnet34 on fashion_mnist</p>	<p>Fail Cases of resnet34 on fashion_mnist</p>

Table 6.3: Success and Fail cases for Alexnet, VGG-16 and Resnet-34 on Fashion\_MNIST

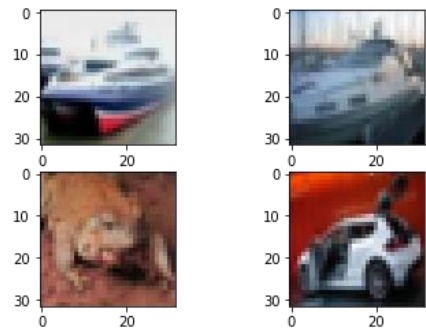
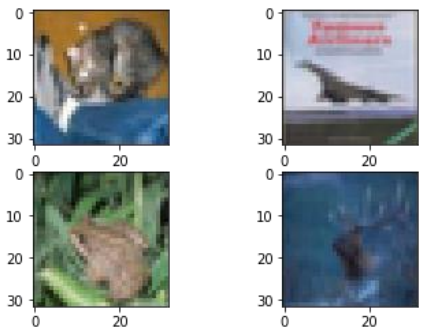
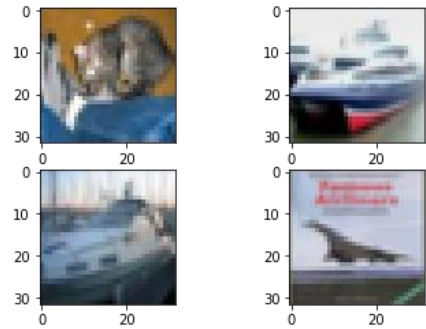

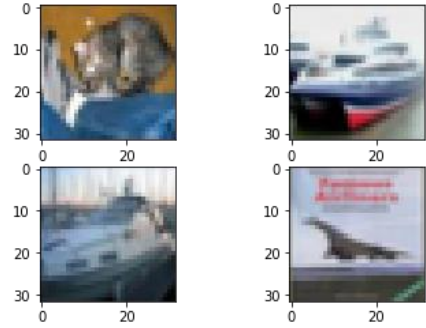
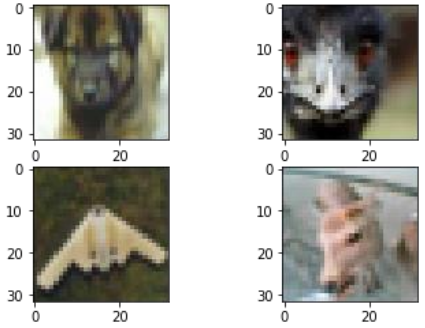
Model	Success Cases on Cifar-10	Fail Cases on Cifar-10
Alexnet	<p>Success Cases of alexnet on cifar-10</p> 	<p>Fail Cases of alexnet on cifar-10</p> 
VGG-16	<p>Success Cases of vgg16 on cifar-10</p> 	<p>Fail Cases of vgg16 on cifar-10</p> 
Resnet-34	<p>Success Cases of resnet34 on cifar-10</p> 	<p>Fail Cases of resnet34 on cifar-10</p> 

Table 6.4: Success and Fail cases for Alexnet, VGG-16 and Resnet-34 on Cifar-10

## References

- [1] LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86, no. 11 (1998): 2278-2324.
- [2] Xiao, Han, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." arXiv preprint arXiv:1708.07747 (2017).
- [3] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.
- [4] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
- [5] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

- [6] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
- [7] Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In 2009 IEEE conference on computer vision and pattern recognition, pp. 248-255. Ieee, 2009.
- [8] Alom, Md Zahangir, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C. Van Esesn, Abdul A. S. Awwal, and Vijayan K. Asari. "The history began from alexnet: A comprehensive survey on deep learning approaches." arXiv preprint arXiv:1803.01164 (2018).
- [9] Fukushima, Kunihiro. "Neocognitron: A hierarchical neural network capable of visual pattern recognition." Neural networks 1, no. 2 (1988): 119-130.
- [10] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.
- [11] Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." arXiv preprint arXiv:1312.4400 (2013).
- [12] Springenberg, Jost Tobias, et al. "Striving for simplicity: The all convolutional net." arXiv preprint arXiv:1412.6806 (2014).
- [13] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
- [14] Huang, Gao, et al. "Densely connected convolutional networks." arXiv preprint arXiv:1608.06993 (2016).
- [15] Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich. "FractalNet: Ultra-Deep Neural Networks without Residuals." arXiv preprint arXiv:1605.07648 (2016).
- [16] Ba, Jimmy, and Rich Caruana. "Do deep nets really need to be deep?." Advances in neural information processing systems. 2014.
- [17] Urban, Gregor, et al. "Do deep convolutional nets really need to be deep and convolutional?." stat 1050 (2016): 4.
- [18] Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009): 7.
- [19] S. Xie, R. Girshick, P. Dollar, Z. Tu and K. He. Aggregated Residual Transformations for Deep Neural Networks. arXiv preprint arXiv:1611.05431v1,2016.
- [20] G. Huang, Y. Sun, Z. Liu, D. Sedra and K. Q. Weinberger. Deep Networks with Stochastic Depth. arXiv:1603.09382v3,2016.
- [21] Zagoruyko, Sergey, and Nikos Komodakis. "Wide residual networks." arXiv preprint arXiv:1605.07146 (2016).
- [22] Byerly, Adam, Tatiana Kalganova, and Ian Dear. "A Branching and Merging Convolutional Network with Homogeneous Filter Capsules." arXiv preprint arXiv:2001.09136 (2020).