**Transfer Learning**

Manish Gour

**1. Problem Statement:**

1. *Construct good denoising autoencoder using dataset D1. Lets call this constructed model as (M1).*
2. *Construct a classification model using dataset D2. Let’s call this constructed model as M2.*

*Obtain the performance of this model on some unseen test set from D2. Let’s call this performance as P.*

1. *Use M1 as a pretrained model to learn the new classification problem given in the dataset D2.*

*What this means is that: use the model M1 as a base model, and finetune this with the new dataset D2. Let’s call this new model that you built as M3. Obtain the performance of this new model M3 using the same test set that you used to evaluate M2. Let’s call this performance as P0.*

1. *Compare P and P0.*

**2. Setup:**

**Runtime Platform:**

Online platform: Colab

RAM: 12GB

**Data Set:**

MNIST ([http://yann.lecun.com/exdb/mnist/)](http://yann.lecun.com/exdb/mnist/),

Fashion MNIST [(https://www.kaggle.com/zalando-research/fashionmnist)](https://www.kaggle.com/zalando-research/fashionmnist) **External Libraries:**

*Numpy* -- Linear Algebra Operation *matplotlib.pyplot* -- Plotting graphs *keras.models import* Model -- Functional API for providing input and output of layers

*keras.callbacks import ModelCheckpoint* -- get a view on internal states and statistics of the model during training.

*keras.layers import (Input, Dense, Concatenate)* --Layer operation like concatenation *keras.utils import np\_utils* -- Converts a class vector (integers) to binary class matrix.

*keras.datasets* -- For loading that are already present in Keras Framework

*keras.layers.normalization import BatchNormalization* -- transform inputs so that they are standardized so that they will have a mean of zero and a standard deviation of one *sklearn.model\_selection import train\_test\_split* -- Splitting the data into train and test sets *keras.callbacks import EarlyStopping* -- Stops the training process based on the parameters *keras.layers import Conv2D* -- Convolution operation on layer

|  |  |  |  |
| --- | --- | --- | --- |
| *keras.layers import MaxPooling2D* -- | | Max pooling operation for spatial data. |  |
|  | | | |
| **3. Implementation:** |  | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Approach1:** |  | | | | | |
| In approach 1, we have added the layer to the pretrained model. | | | | |  | |
| |  | | --- | | ***Task 1: Denoising autoencoder on MNIST Dataset*** | | | |  | | | |
|  | | |
| **Representational Learning** | |  | | | | |
| |  | | --- | | It is a step further towards artificial intelligence as it is an automatic way of feature | | | | | | |  |
| |  | | --- | | engineering, i.e. discovery of new representations of data that are useful during modelling and enhance prediction scores. It replaces manual feature extraction – why to bother with | | | | | | |  |
| creating interactions, logarithms or other transformations of data and then guessing which | | | | | |
| |  | | --- | | of them are important as we can simply use Representation Learning. | | | | |  | |
|  | | | |



**Denoising Auto**

**encoders**

plying

can be used to learn superior representation of data. Sup

noisy version of data, forces the Autoencoder to perform better than its clean input

counterpart a

nd as a consequence it produces representation of data that is immune t

o

random noise

.

**MNIST Datase**

**t**

test

The MNIST database of handwritten digits, has a training set of 60,000 examples, and a

set of 10,000 examples. It is a subset of a larger set avail

able from NIST. The digits have

been size

-

normalized and cantered in a fixed

-

size

image

.

**Ste**

**ps:**

**1.**

**Load, reshape, scale and add noise to data,**

**2.**

**Defining DAE, t**

**rain DAE on merged training and testing data,**

**3.**

**Get neuron outputs from DAE as new features**

**1**

**.**

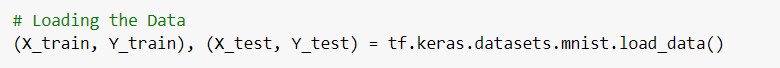
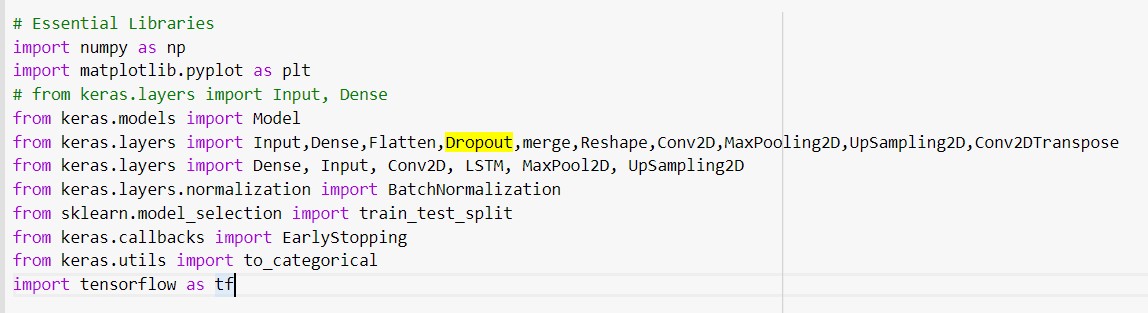
**Load, reshape, scale and add noise to data**

**1.1**

import libraries

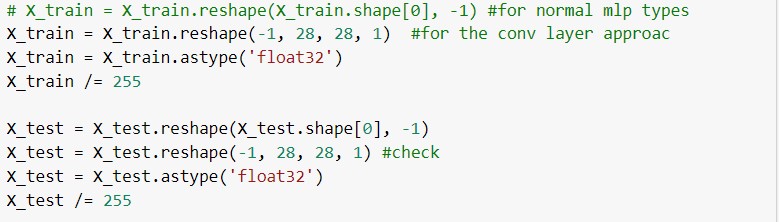
and Loading the Dataset

.



|  |  |
| --- | --- |
| **1.2 Data Preprocessing** |  |
| Conversion of 28 x 28 image of train and test set into a matrix of size 28 x 28 x 1 to feed in | |

the network. Followed by it, conversion of numpy arrays in float32 format for storing floating values.



**1.3 Adding noise**

introducing Gaussian random noise, noise factor controls the noisiness of images and we clip the values to make sure that the elements of feature vector representing image are

|  |  |  |
| --- | --- | --- |
| between 0 and 1. It will be | used to learn superior representation of data. |  |
|  | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2.** | **Defining DAE, train DAE on merged training and testing data** | | |  |
|  | | | | |
| **2.1 A** | | **rchitecture of Denoising Autoencoder** |  | |

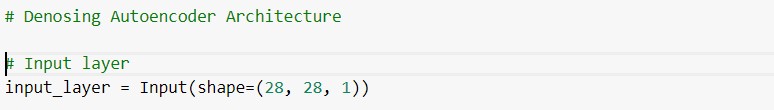
**Encoder**: It has 4 Convolution blocks; each block has a convolution layer with ‘relu’ as activation function, followed by a batch normalization layer. Max-pooling layer is used after the first and second convolution blocks.

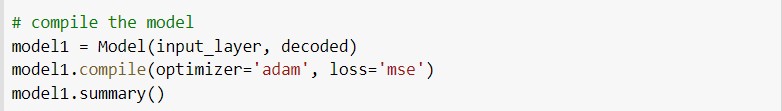
* The first convolution block will have 32 filters of size 3 x 3, followed by a down sampling (max-pooling) layer,
* The second block will have 64 filters of size 3 x 3, followed by another down sampling layer,
* The third block of encoder will have 128 filters of size 3 x 3,  The fourth block of encoder will have 256 filters of size 3 x 3.

**Decoder:**

It has 3 Convolution blocks, each block has a convolution layer followed by a batch normalization layer. Upsampling layer is used after the second and third convolution blocks.

* The first block will have 128 filters of size 3 x 3,
* The second block will have 64 filters of size 3 x 3 followed by another upsampling layer,
* The third block will have 32 filters of size 3 x 3 followed by another upsampling layer,
* The final layer of encoder will have 1 filter of size 3 x 3 which will reconstruct back the input having a single channel.



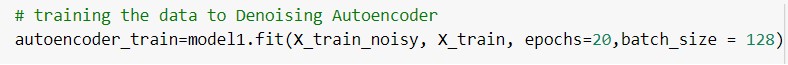


Adam Optimizer for controlling learning rate and Mean squared error as loss function. We train the Denoising Autoencoder with 20 epochs and batch\_size=4096.

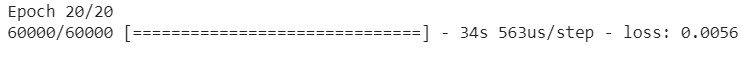
|  |  |  |  |
| --- | --- | --- | --- |
| Callbacks are used to | get a view on internal states and statistics of the model during | |  |
| |  | | --- | | training. The latest best model according to the quantity monitored will not be | | |  | |
| overwritten. | |



Constructed Noisy data (X\_train\_noisy) is provided as input to get the ouput as X\_train (original Input Data).



After 20 epoch loss has been reduced to 0.076.



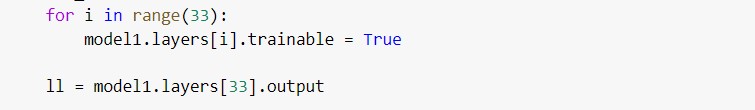
|  |  |
| --- | --- |
| **3. Get neuron outputs from DAE as new features** |  |
|  | |

Using Output layer of the Denosing Autoencoder can be further connected as pretrained layer for training the related datsets. Layers of the autoencoder will be having learned weights which can converge faster and can further improvises the learning process.

3.1 Extracting the output layer using fuctional apis

Model for Denosing Autoencoder contains 33 layers with pretrained weights, whose last layer can be extracted out as below and can be connected to any model which will eventaully transfer its learning to the new model.

Also, layers are kept trainable so that weight of pretrained model can be modify at run time. We have tested that while keeping layers trainable provides us more accuracy comapred to keeping it freezable.



***Task 2: Training Fashion MNIST model:***

# 2.1 Fashion MNIST Dataset

Fashion-MNIST dataset is a 28x28 grayscale images of 70,000 fashion products from 10 categories, with 7,000 images per category. The training set has 60,000 images, and the test set has 10,000 images. Fashion-MNIST is a replacement for the original MNIST dataset for producing better results, the image dimensions, training and test splits are similar to the original MNIST dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| **2. 2 Model Architecture for training Fashion MNIST** | |  | |
| |  | | --- | | It contains four layers mainly input layer, flatten layer, followed by dense layer and output | | | |  |
| layer. |  | |



First layer,

Input layer takes 28\*28

image

as an input



Second Layer, flatten

ments

layer is have the shape that is equal to the number of ele

ned in tensor

contai

non including the batch dimension which has 784 as output

shape.



2

-

Dense layer contains

64

neurons

each

with ‘relu’ as activation function.



Output layer contains 10 neuro

ns represents category of

e

ach input

type

and uses

‘

softmax

’

as activation function.

Model

has been t

rained with batch size

128

till

2

0

epochs

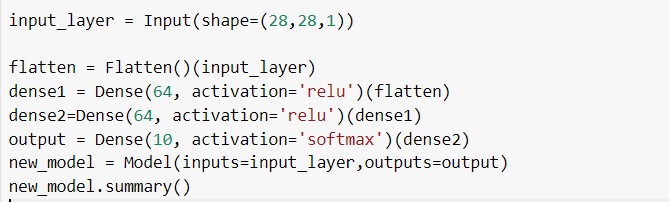
. Learning rate is controlled using

‘

adam

’

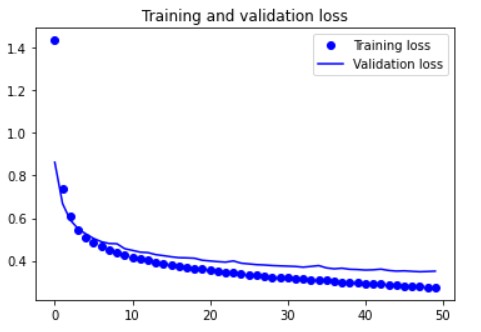
optimizer.



|  |  |  |  |
| --- | --- | --- | --- |
| **2.3 Training and Validation accuracy – loss** | |  | |
|  | | | |
| After 50 epochs 92.21% training accuracy while 88.24% accuracy has been obtained as | | |  |
| |  | | --- | | below: | |  | | |
|  |



|  |  |  |
| --- | --- | --- |
| It can be seen from below figure that training and validation loss are close to each other | |  |
| |  | | --- | | implying that our model is not overfitting. | |  | |
|  |

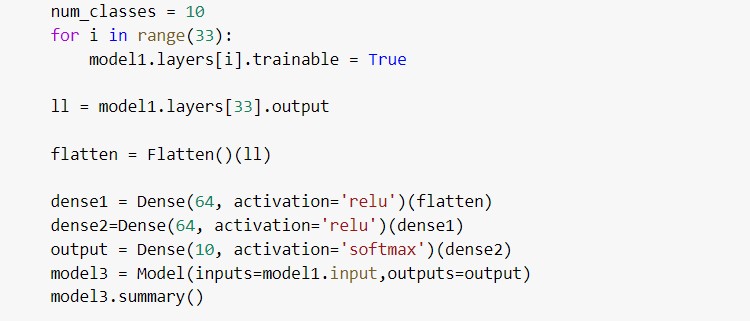


# Task 3: Transfer learning from denosing autoencoder model to Fashion MNISt model

Data loading and preprocessing task will be similar to task1 and task2.

**3.1 Architecture:**

From the task1, layer33 (output layer for Denoising autoencoder) has been extracted and connected to the model used in the task2. It will results into 37 layers which includes 33 layers from pretrained denoising autoencoder while 4 additional layer from model used in task2.



## 3.2 Training and Validation accuracy – loss

Training accuracy of the model is recorded as 98.42% while validation accuracy as 92.68% after 20 epochs with batch size 128.

**Task 4: Performance Comparision of Task2 (P, Model2)and Task3(P0, Model3):**

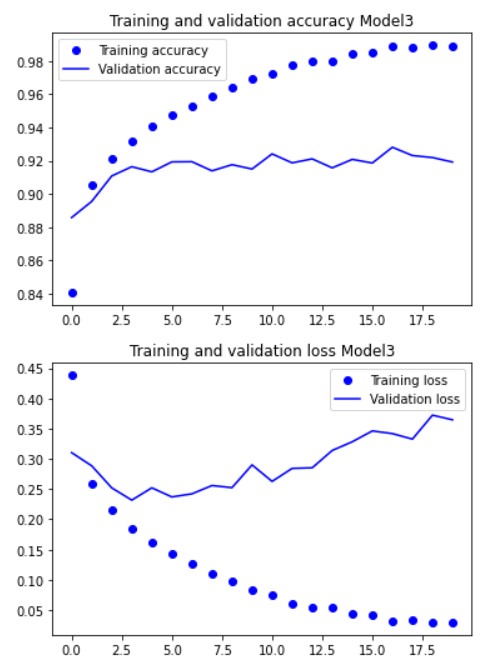
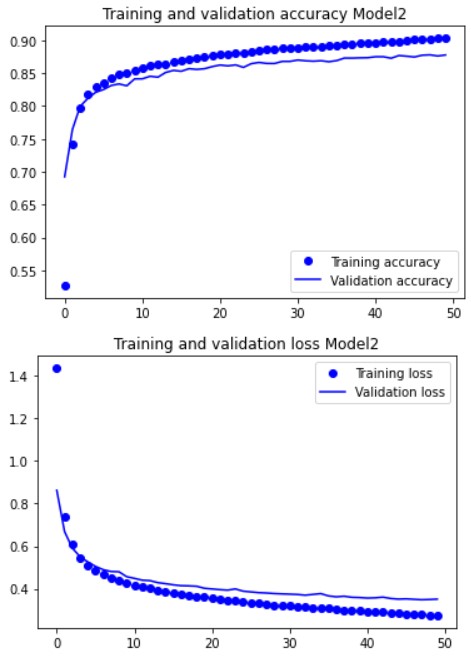
Model2 model used in Task2 without transfer learning

Model3 model used in Task3 with transfer learning

Compariosn between Tranining - validation accuracy as well Tranining – validation Loss can be seen in the below figure.

It can be clearly observed that both tanining and Validation Accuracy is higher in Model3 compared to Model2 while opposite to it tanining and Validation Loss is higher in Model2.

Also, below figure shows closer gap between accuracy and loss in case of model2 while wider gaps in case of model3 which denoted that *model3 is being overfitted****.***



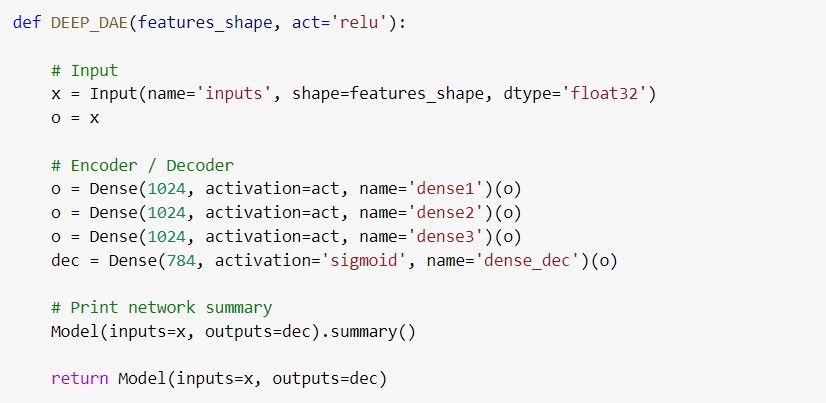
**Approach2**

In Approach2 we have copied all the weights of pretrained model to new model keeping the architecture same. We have used dense layer in this architecture for Denosing autoencoder as well as for model.

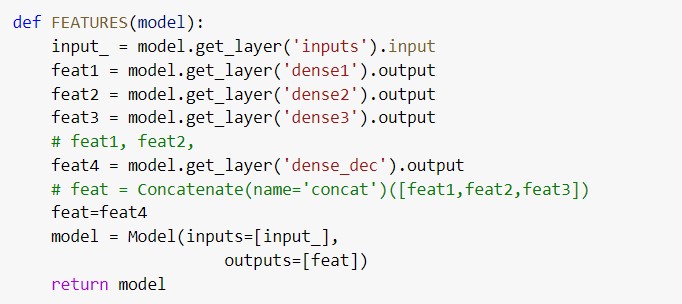
All the three Models used in the task have same architecutre, they differs in the wayy how they use prelearned weight and randomly initialiazed weight.

**Denosing Architecture**

It contains Input layer followed by three Dense layer act as encoding and decoding layers and last layer as output layer.

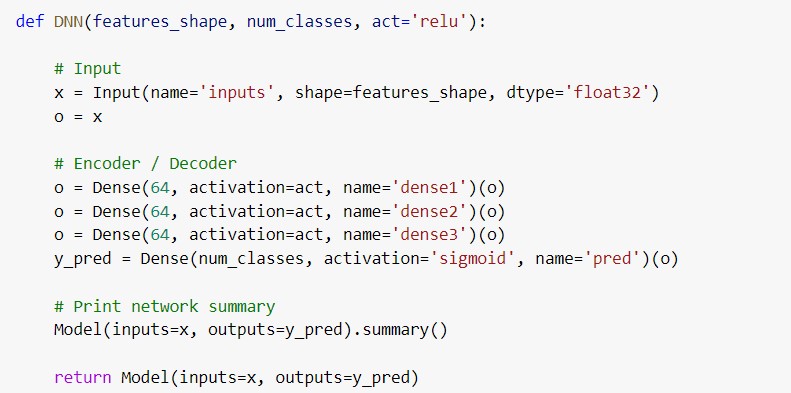


**Using Output of Denosing layers for feature engineering of the task1** Here all three layers can be used to do the feature engineering, we have used output layer to further train the model.



**Model Architecture**

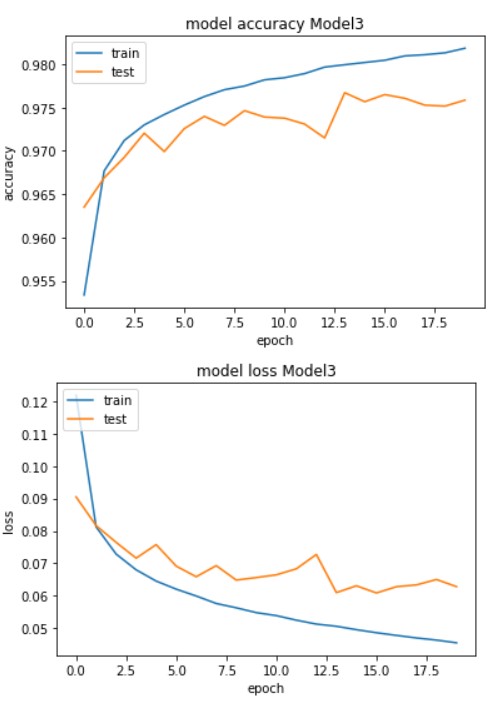
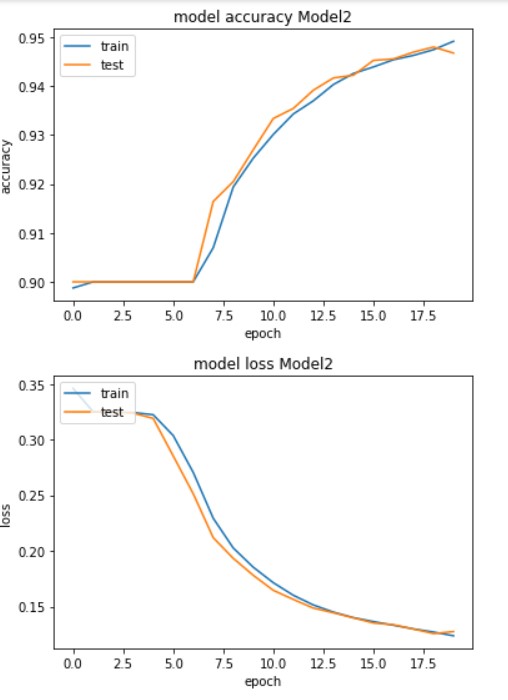
This Architecture is maintained same across all three task. Model1 (task1) uses Denosing output as input for this model while Model2 (task2) takes training data directly while Third model uses the pretrained weights learned from model 1.



**Performance Comparision (Approach 2)**

It has been obserbed that Pretrained model has performed really well with validation accuracy of

97.58% while non-pretrained model has validation accuracy of 94.67%

**4. Conclusion:**

1. Denoising Autoencoders can be used to learn superior representation of data. Supplying noisy version of data, forces the Autoencoder to perform better than its clean input counterpart and as a consequence it produces representation of data that is immune to random noise.
2. Accuracy of the model can be improved with pretrained models. Pretrained weights can significantly improve the performace and can converge the model faster if choosen carefully.
3. Sometime, Pretrained model can overfit the model which results in large gap between validation and training loss / Accuracy but if trained carefully they gives good accuracy without being overfit the data.

**5. References:**

* [https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transferlearning-with-real-world-applications-in-deep-learning-212bf3b2f27a](https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a)

* <https://machinelearningmastery.com/transfer-learning-for-deep-learning/>

* <http://yann.lecun.com/exdb/mnist/>

* <https://www.kaggle.com/zalando-research/fashionmnist>

* <https://keras.io/layers/core/>

* <https://dkopczyk.quantee.co.uk/dae-part3/>

* http://www.jmlr.org/ papers/v11/erhan10a.html

* <https://keras.io/layers/convolutional/>