CSCI946: Big Data Analytics
Assignment - 3

Venkata Sandeep Kumar. Karamsetty, Sudent No: 6228975,

Mail-Id : vskk033@uowmail.edu.au

1 Task 1 – Understanding CNNs for image analysis

1.1 Describe deep convolutional neural networks and discuss why they suit image analysis tasks

1.1.1 Convolution Neural Network:

A Convolution Neural Network(ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

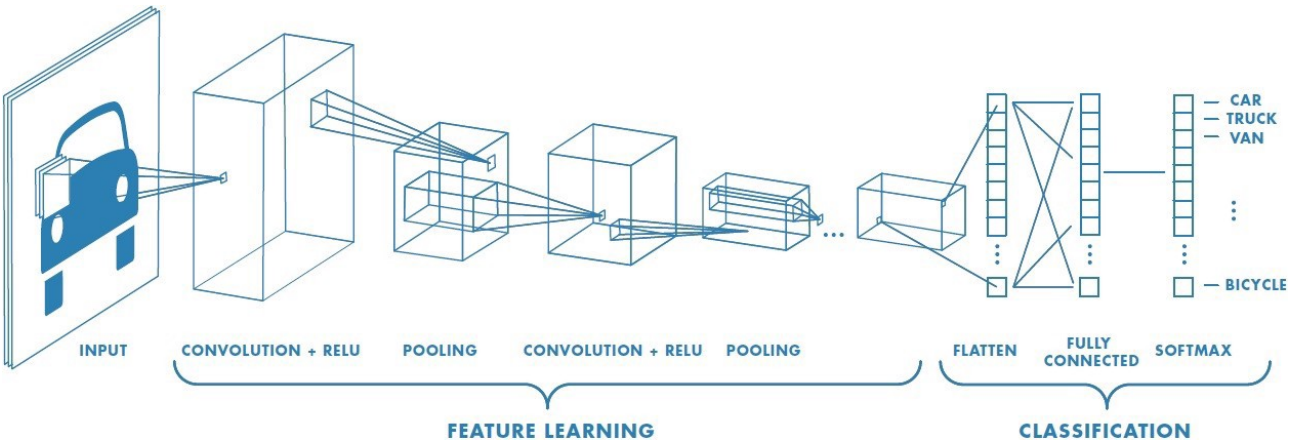


Figure 1: Convolutional Neural Network applied on image analysis[Ref.No - 1]

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection

of such fields overlap to cover the entire visual area. The main applications for CNN are image and video recognition, recommender systems, and natural language processing. A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers as shown in figure 1.

Deep Convolution Neural Network:
a series of convolution layer with filter (kernels), pooling and fully connected layers and apply a softmax function to classify an object with probabilistic value between 0 and 1. This deep convolution neural network is used in figure 1 under the classification to classify the images.

The various layers involved in the CNN image classification are:[Ref.No:1]:
1.) Convolution Layer — The Kernel
2.) Pooling Layer
3.) Classification — Fully Connected Layer (FC Layer)

1.1.2 why deep convolutional neural networks suit image analysis tasks?

Image recognition and analysis is not an easy task to achieve. A good way to think about achieving it is through applying metadata to unstructured data. Hiring human experts for manually tagging the libraries of music and movies may be a daunting task but it becomes highly impossible when it comes to challenges such as teaching the driverless car's navigation system to differentiate pedestrians crossing the road from various other vehicles or filtering, categorizing or tagging millions of videos and photos uploaded by the users that appear daily on social media.

One way to solve this problem would be through the utilization of neural networks. We can make use of conventional neural networks for analyzing images in theory, but in practice, it will be highly expensive from a computational perspective. Take for example, a conventional neural network trying to process a small image(let it be 30*30 pixels) would still need 0.5 million parameters and 900 inputs. A reasonably powerful machine can handle this but once the images become much larger(for example, 500*500 pixels), the number of parameters and inputs needed increases to very high levels.

There is another problem associated with the application of neural networks to image recognition: overfitting. In simple terms, overfitting happens when a model tailors itself very closely to the data it has been trained on. Generally, this leads to added parameters(further increasing the computational costs) and model's exposure to new data results in a loss in the general performance[Ref No. 2]. Hence, these problems can be solved using **deep convolutional neural network**. Let us take an example into consideration, if you consider any image, proximity has a strong relation with similarity in it and convolutional neural networks specifically take advantage of this fact. This implies, in a given image, two pixels that are nearer to each other are more likely to be related than the two pixels that are apart from each other.

Also general applicability of neural networks is one of their advantages, but this advantage turns

into a liability when dealing with images. The convolutional neural networks make a conscious tradeoff: if a network is designed for specifically handling the images, some generalizability has to be sacrificed for a much more feasible solution. If you consider any image, proximity has a strong relation with similarity in it and convolutional neural networks specifically take advantage of this fact. This implies, in a given image, two pixels that are nearer to each other are more likely to be related than the two pixels that are apart from each other. Nevertheless, in a usual neural network, every pixel is linked to every single neuron. The added computational load makes the network less accurate in this case.

By killing a lot of these less significant connections, convolution solves this problem. In technical terms, convolutional neural networks make the image processing computationally manageable through filtering the connections by proximity. In a given layer, rather than linking every input to every neuron, convolutional neural networks restrict the connections intentionally so that any one neuron accepts the inputs only from a small subsection of the layer before it(say like 5*5 or 3*3 pixels). Hence, each neuron is responsible for processing only a certain portion of an image.(Incidentally, this is almost how the individual cortical neurons function in your brain. Each neuron responds to only a small portion of your complete visual field).

Drawbacks of using fully connected layers on image input [Ref No: 3 4]
FC layers don't exploit local structure, FC layer is not equivariant under translation (translated pattern can correspond to completely different feature) and For big images FC layers have huge number of parameters (for example for 1000 hidden units, 1000 x 1000 x 3 image such layer has 3 billion weights)

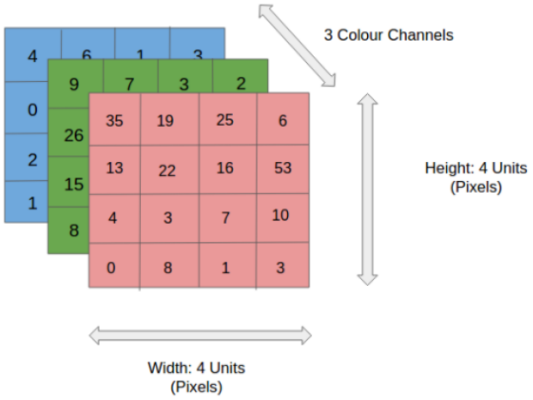
In contrast to that, CNN layers:
Extract local features (conv layer outputs only depend on adjacent pixels of previous layer), CNN are equivariant to translation and Have lot less parameters, since they share kernel over the patches of whole input images.

In addition to that, CNN networks use pooling layers, which provide (some) translation invariance (pooling makes each feature more and more invariant to translation).

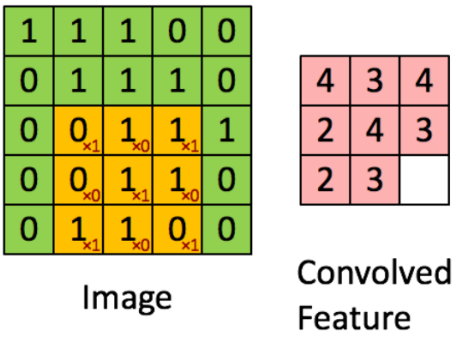
1.2 Describe the functions of "convolution layer" and "pooling layer" in deep CNNs

1.2.1 Convolution layer — The Kernel:

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.



(a) Sample RGB input image



(b) Convolution kernel applied on image

Figure 2: Convolution layer application

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB) In the above demonstration, the green section resembles our 5x5x1 input image, I. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow. We have selected K as a 3x3x1 matrix as shown in figuure 2 - (b). The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering. The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would. There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying Valid Padding in case of the former, or Same Padding in the case of the latter.

When we augment the 5x5x1 image into a 6x6x1 image and then apply the 3x3x1 kernel over it, we find that the convolved matrix turns out to be of dimensions 5x5x1. Hence the name — Same Padding. On the other hand, if we perform the same operation without padding, we are presented with a matrix which has dimensions of the Kernel (3x3x1) itself — Valid Padding.

To determine the dimensions of the activation map:

(N+2P-F)/S+1, where N = dimension of image (input) file
P = padding,
F = dimension of filter, S = stride.

1.2.2 Pooling layer:

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.

Types of Pooling The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power. After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes. Formula for the output after max pooling:

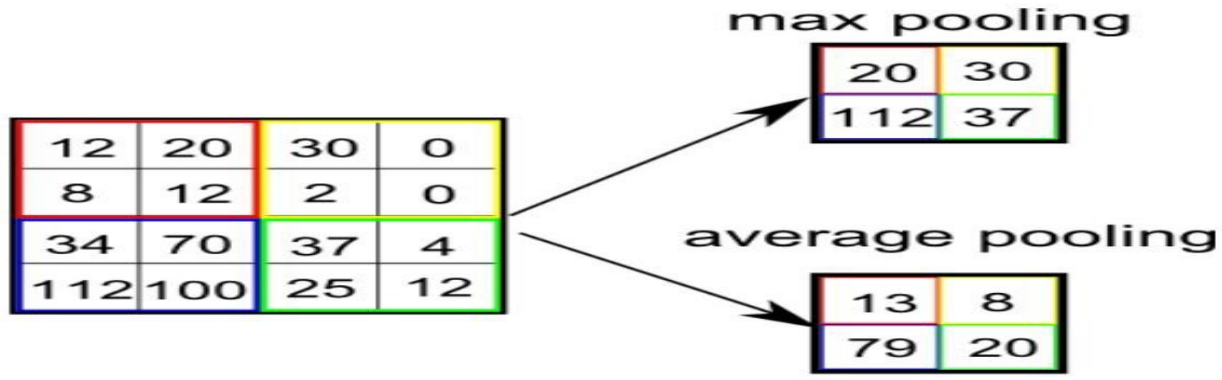


Figure 3: Convolutional Neural Network applied on image analysis[Ref.No - 1]

(N — F)/ S + 1 where N = Dimension of input to pooling layer
F = Dimension of filter
S = Stride

1.3 Explain the following concepts in training deep CNNs: a) activation function; b) epoch number; c) batch size; d) learning rate; and e) momentum.

1.3.1 a) activation

It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. They help to decide if the neuron would fire or not. Types of activation functions are shown in figure - 4.

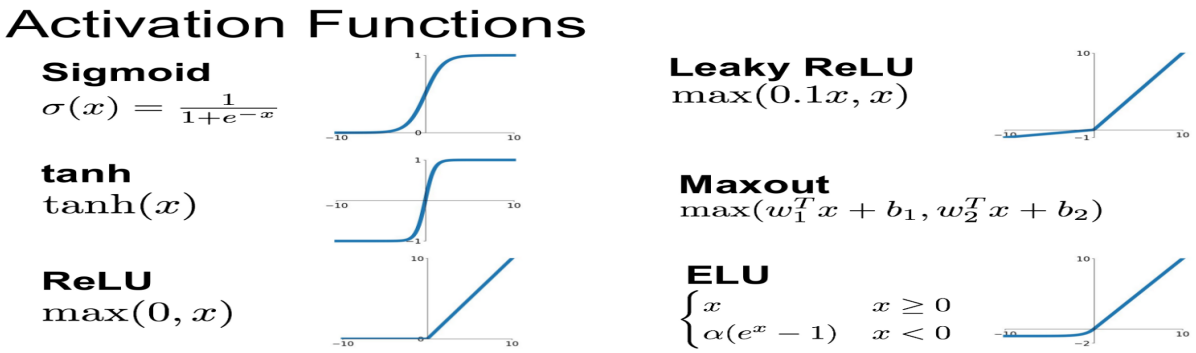


Figure 4: Activation function types - [Ref.No - 5]

1.3.2 b) epoch number [Ref No - 6]

As the large datasets are present in machine learning which makes computer system unable to handle at once we use Epoch to handle the datasets. One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE. Since one epoch is too big to feed to the computer at once we divide it in several smaller batches. **Selecting a good Epoch number:** passing the entire dataset through a neural network is not enough. And we need to pass the full dataset multiple times to the same neural network. But always remind that we are using a limited dataset and to optimise the learning and the graph we are using Gradient Descent which is an iterative process. So, updating the weights with single pass or one epoch is not enough. One epoch leads to under-fitting of the curve. Also the number of epochs increases, more number of times the weight are changed in the neural network and the curve goes from under-fitting to optimal to over-fitting curve. Hence the selecting the correct Epoch is different for different data-sets, but you can say that the numbers of epochs is related to how diverse your data is.

1.3.3 c) batch size

Total number of training examples present in a single batch is called batch size. A huge data-set will divided into Number of Batches or sets or parts. These divided batches can be easily accessed for training the neural networks. **Iteration:** Iterations is the number of batches needed to complete

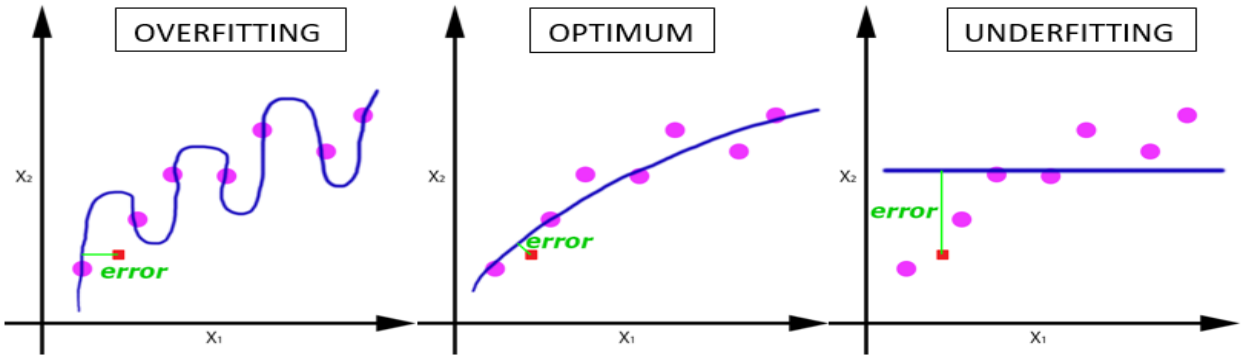


Figure 5: Epoch - [Ref.No - 6]

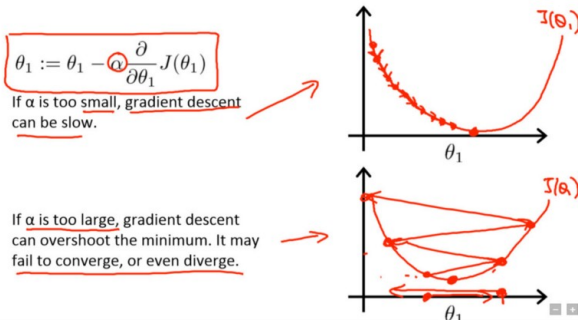
one epoch. **Example of application of above variables:**
Let's say we have 2000 training examples that we are going to use.
We can divide the data-set of 2000 examples into batches of 500 then it will take 4 iterations to complete 1 epoch.

1.3.4 d) learning rate

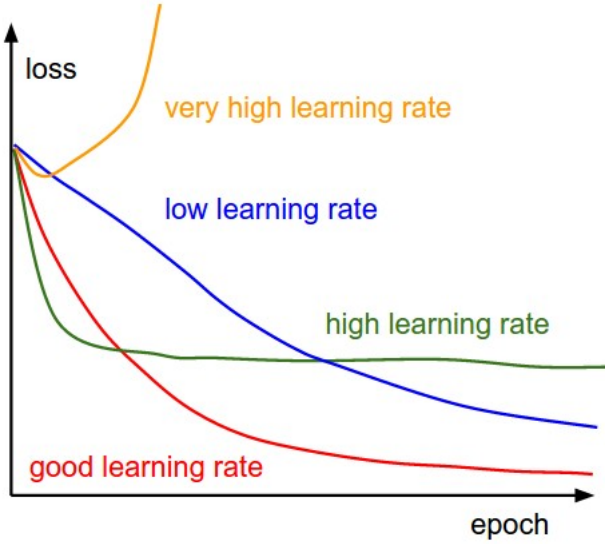
Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. The lower the value, the slower we travel along the downward slope. While this might be a good idea (using a low learning rate) in terms of making sure that we do not miss any local minima, it could also mean that we'll be taking a long time to converge — especially if we get stuck on a plateau region[7].
The following formula shows the relationship.
$new\_weight = existing\_weight$ - $learning\_rate * gradient$

Typically learning rates are configured naively at random by the user. At best, the user would leverage on past experiences (or other types of learning material) to gain the intuition on what is the best value to use in setting learning rates.

As such, it's often hard to get it right. The below diagram demonstrates the different scenarios one can fall into when configuring the learning rate shown in figure 6-b. A good learning rate by training the model initially with a very low learning rate and increasing it (either linearly or exponentially) at each iteration. If we record the learning at each iteration and plot the learning rate (log) against loss; we will see that as the learning rate increase, there will be a point where the loss stops decreasing and starts to increase. In practice, our learning rate should ideally be somewhere to the left to the lowest point of the graph (as demonstrated in below graph). In this case, 0.001 to 0.01 as shown in figure 7.



(a) Source: Andrew Ng's Machine Learning course on Coursera



(b) Effect of various learning rates on convergence

Figure 6: Learning rate - [Ref.No - 6]

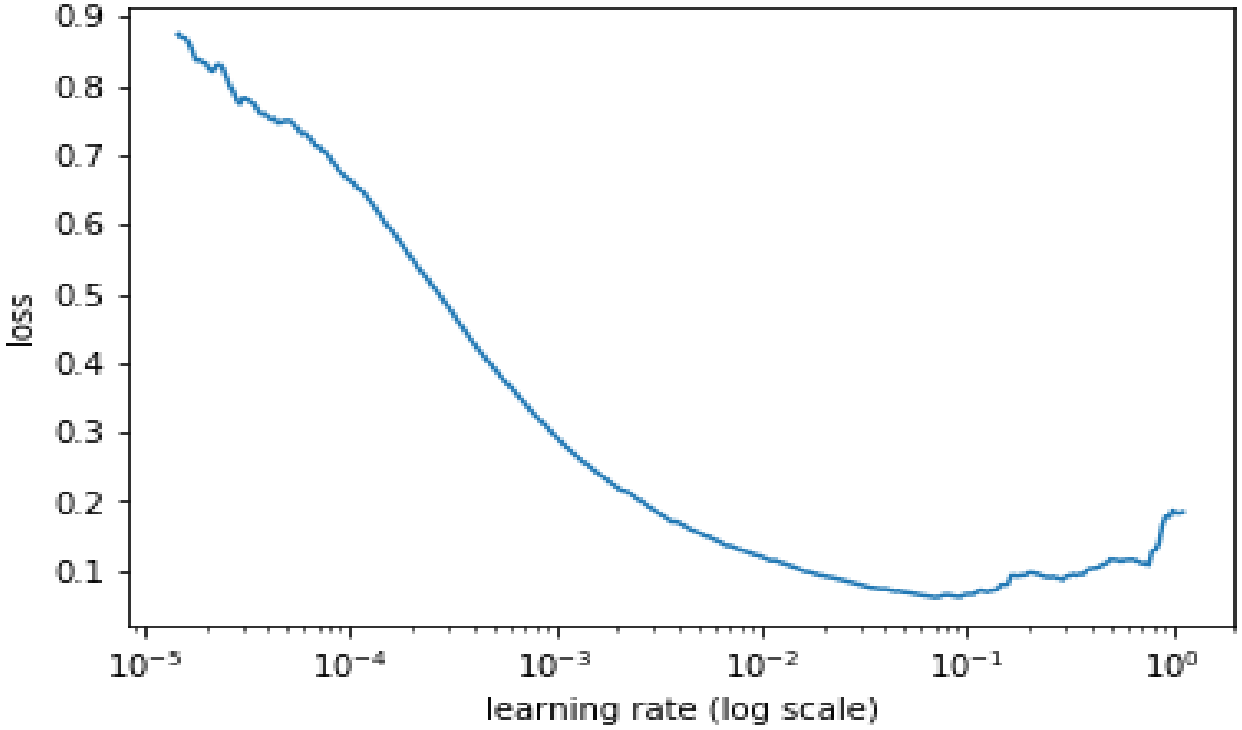
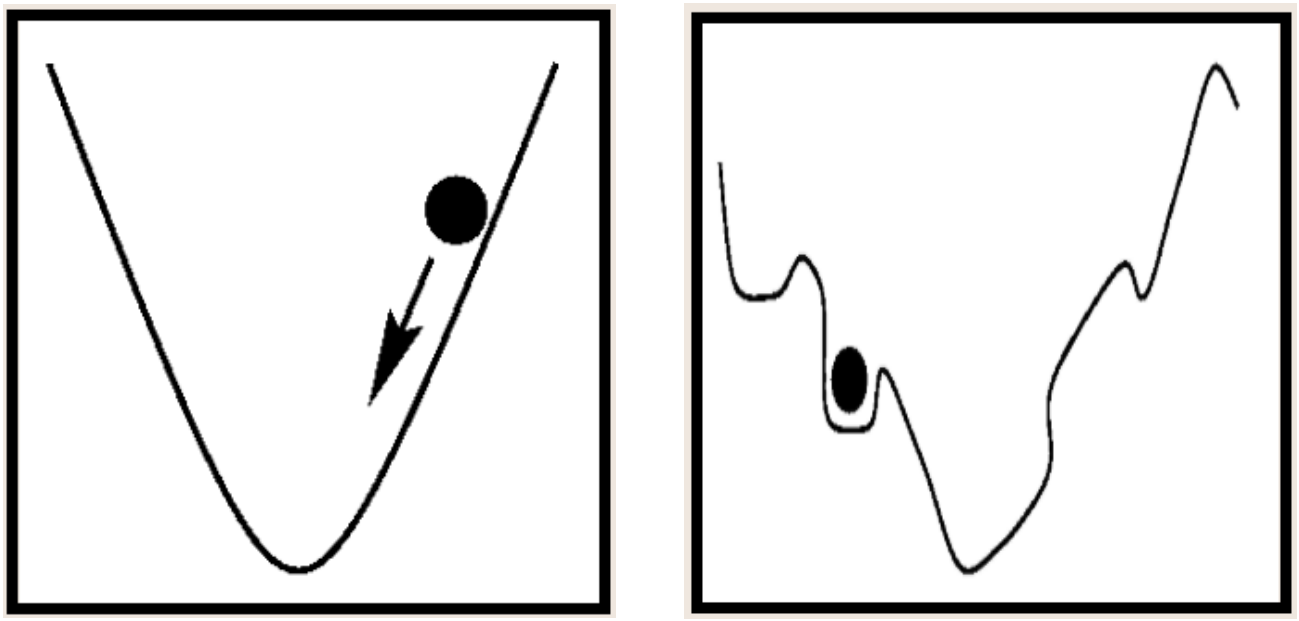


Figure 7: Learning rate - [Ref.No - 7]



(a) Ideal Error Function (b) Effect of various learning rates on convergence

Figure 8: Learning rate - [Ref.No - 9]

1.3.5 e) momentum

It is an optimization algorithm that is used to train the deep CNN. In neural networks, we use gradient descent optimization algorithm to minimize the error function to reach a global minima. In an ideal world the error function would look like shown in figure - 8(a). So we are guaranteed to find the global optimum because there are no local minimum where your optimization can get stuck. However in real the error surface is more complex, may comprise of several local minima and may look like shown in figure - 8(b). In the figure 8(b) case, you can easily get stuck in a local minima and the algorithm may think you reach the global minima leading to sub-optimal results. To avoid this situation, we use a momentum term in the objective function, which is a value between 0 and 1 that increases the size of the steps taken towards the minimum by trying to jump from a local minima. If the momentum term is large then the learning rate should be kept smaller. A large value of momentum also means that the convergence will happen fast. But if both the momentum and learning rate are kept at large values, then you might skip the minimum with a huge step. A small value of momentum cannot reliably avoid local minima, and can also slow down the training of the system. Momentum also helps in smoothing out the variations, if the gradient keeps changing direction. A right value of momentum can be either learned by hit and trial or through cross-validation.

1.4 References

1.) https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

2.) https://www.kdnuggets.com/2017/08/convolutional-neural-networks-image-recognition.html

3.) https://stats.stackexchange.com/questions/344616/why-use-convolutions-to-image-processing-tasks
4.) http://www.cs.toronto.edu/ hinton/csc2535/notes/lec6a.pdf
5.) https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17
6.) https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9
7.) https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10
8.) https://www.quora.com/What-does-momentum-mean-in-neural-networks
9.) https://www.willamette.edu/ gorr/classes/cs449/momrate.html

2 Task 2 – Handwritten Digits Classification with CNNs

2.1 Describe this MNIST data set and its training and test subsets.

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. This database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels [Ref No: 1]. Sample image is shown in figure 9.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

2.2 Describe how you reshape each image into a long vector and how you train the LRC or SVM.

SVM stands for support-vector machines.
LRC stands for Logistic regression classifier
Keras supplies the MNIST dataset which has structure(image, width, height) with a gray scale image of two dimensions of width and height 28*28. By this we can observe that data is in a 3D array. For preparing the data for training we need to convert these 3D arrays into matrices by reshaping width

Figure 9: Sample images from MNIST test dataset - [Ref.No - 1]

and height into a single dimension image. After tha above operation we can convert the grayscale values from integers ranging between 0 to 255 into floating point values rainging between 0 and 1.

Code used for Reshaping:

trainSamples, trainWidth, trainHeight = X_train.data.shape
testSamples, testWidth, testHeight = X_test.data.shape
X_train = X_train.reshape((trainSamples,trainWidth*trainHeight))
X_test = X_test.reshape((testSamples,testWidth*testHeight))

No hence the data is reshaped we are ready to use for support-vector machines (SVM) Logistic regression classifier (LRC)

Training of SVM in python - Code(function module) and parameters:

$svmClassifier = LinearSVC(C = 1.0, class\_weight = None, dual = True, fit\_intercept = True, intercept$
$1, loss =' squared\_hinge', max\_iter = 1000, multi\_class =' ovr', penalty =' l2', random\_state =$
$None, tol = 0.0001, verbose = 0)$
$svmClassifier.fit(X_train, y_train)$

Parameters of SVM:[Ref No. 3]

penalty : string, 'l1' or 'l2' (default='l2')
Specifies the norm used in the penalization. The 'l2' penalty is the standard used in SVC. The 'l1' leads to coef_ vectors that are sparse.

loss : string, 'hinge' or 'squared_hinge' (default='squared_hinge')
Specifies the loss function. 'hinge' is the standard SVM loss (used e.g. by the SVC class) while

'squared_hinge' is the square of the hinge loss.

dual : bool, (default=True)
Select the algorithm to either solve the dual or primal optimization problem. Prefer dual=False when n_samples > n_features.

tol : float, optional (default=1e-4)
Tolerance for stopping criteria.

C : float, optional (default=1.0)
Penalty parameter C of the error term.

multi_class : string, 'ovr' or 'crammer_singer' (default='ovr'): Determines the multi-class strategy if y contains more than two classes. "ovr" trains n_classes one-vs-rest classifiers.

fit_intercept : boolean, optional (default=True)
Whether to calculate the intercept for this model. If set to false, no intercept will be used in calculations (i.e. data is expected to be already centered).

intercept_scaling : float, optional (default=1)
class_weight : dict, 'balanced', optional
verbose : int, (default=0)
Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in liblinear that, if enabled, may not work properly in a multithreaded context.
random_state : int, RandomState instance or None, optional (default=None)
max_iter : int, (default=1000) - The maximum number of iterations to be run.

So, hence by using the above functional modules and data we can train SVM.

Training of LRC in python - Code(function module) and parameters:

```
lrcClassifier = LogisticRegression(solver = 'lbfgs',multi_class='ovr',random_state = 1)
lrcClassifier.fit(X_train, y_train)
```

Parameters of LCR:[Ref No. 4]
solver : str, 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga', default: 'liblinear': For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss; 'liblinear' is limited to one-versus-rest schemes.

multi_class: str, 'ovr', 'multinomial', 'auto', default: 'ovr'If the option chosen is 'ovr', then a binary problem is fit for each label.

random_state: The seed of the pseudo random number generator to use when shuffling.

2.3 Describe the convolutional neural network "LeNet" and how you train it. Try other settings of the network parameters and/or training parameters, and describe the changes on classification accuracy and training time.

LeNet-5, a pioneering 7-level convolutional network by LeCun et al. in 1998, that classifies digits, was applied by several banks to recognize hand-written numbers on checks (British English: cheques) digitized in 32x32 pixel images. The ability to process higher resolution images requires larger and more layers of convolutional neural networks, so this technique is constrained by the availability of computing resources[Ref No. 6].

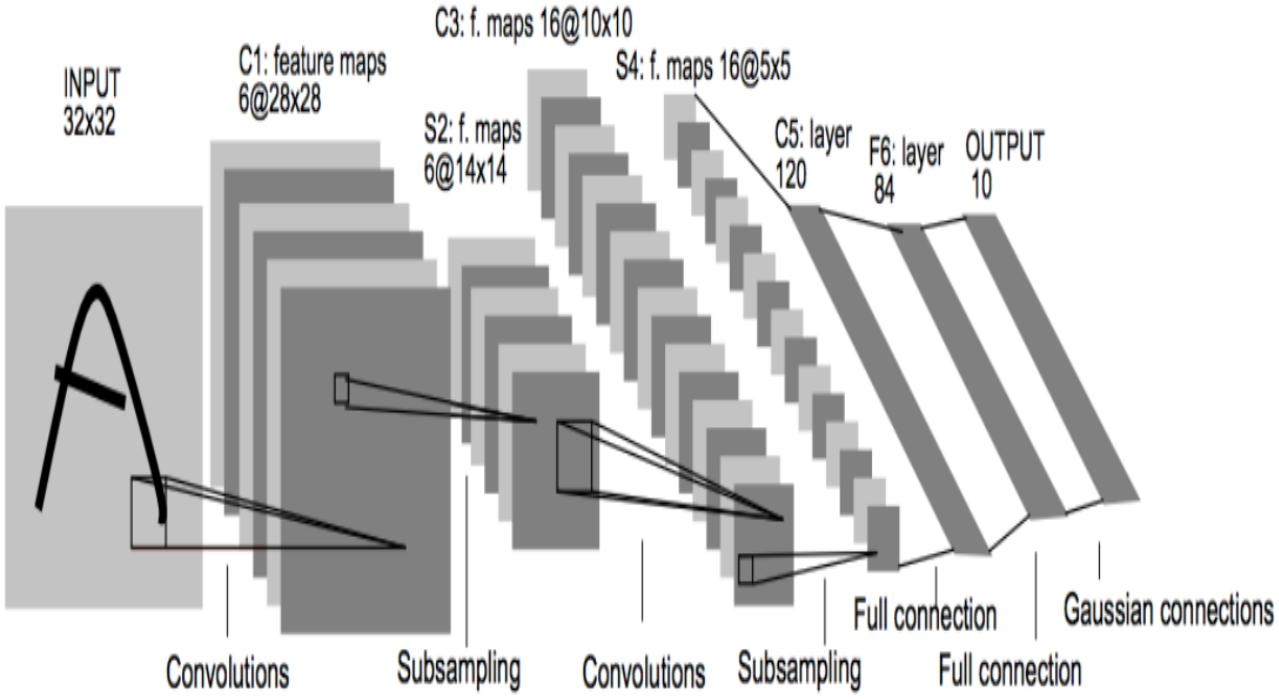


Figure 10: LeNet-5 Architecture. Credit: LeCun et al., 1998 - [Ref.No - 5]

Input The LeNet architecture accepts a 32x32xC image as input, where C is the number of color channels. Since MNIST images are grayscale, C is 1 in this case.

Architecture

Layer 1: Convolutional. The output shape should be 28x28x6.
Activation. Your choice of activation function.
Pooling. The output shape should be 14x14x6.
Layer 2: Convolutional. The output shape should be 10x10x16.
Activation. Your choice of activation function.

Pooling. The output shape should be 5x5x16.

Flatten. Flatten the output shape of the final pooling layer such that it's 1D instead of 3D. The easiest way to do is by using tf.contrib.layers.flatten, which is already imported for you.

Layer 3: Fully Connected. This should have 120 outputs.

Activation. Your choice of activation function.

Layer 4: Fully Connected. This should have 84 outputs.

Activation. Your choice of activation function.

Layer 5: Fully Connected (Logits). This should have 10 outputs.

By using categorical_crossentropy loss function as it is a multi-class classification problem. Since all the labels carry similar weight we prefer accuracy as performance metric. A popular gradient descent technique called "adam" is used for optimization of the model parameters.

function module used:

keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

Output

```
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
10000/10000 [======================
Test loss: 0.048388700930664615
Test accuracy: 0.9878
```

Figure 11: LeNet Accuracy

Output As shown in the figure 11 LeNet Accuracy is 98.78% which is pretty high, which implies the model is trained well for prediction. It took 243 to 245 seconds to train.

first test- Shown in figure 11 and 12:

For learning rate = 0.001

Obtained Accuracy is: 98.78%

Process time: 244 seconds.

Second test:

For learning rate as 0.01 and momentum as 0.9

Obtained Accuracy is: 96.68%

Process time: 247 seconds.

Third test:

learning rate as 0.02 and momentum as 0.8

Obtained Accuracy is: 97.68%

Process time: 254 seconds.

Hence from all the three tests above first test has the high accuracy.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 250s 4ms/step -
Epoch 2/10
60000/60000 [==============================] - 250s 4ms/step -
Epoch 3/10
60000/60000 [==============================] - 250s 4ms/step -
Epoch 4/10
60000/60000 [==============================] - 251s 4ms/step -
Epoch 5/10
60000/60000 [==============================] - 250s 4ms/step -
Epoch 6/10
60000/60000 [==============================] - 251s 4ms/step -
Epoch 7/10
60000/60000 [==============================] - 250s 4ms/step -
Epoch 8/10
60000/60000 [==============================] - 250s 4ms/step -
Epoch 9/10
60000/60000 [==============================] - 250s 4ms/step -
Epoch 10/10
60000/60000 [==============================] - 250s 4ms/step -
```

Figure 12: LeNet time

2.4 Report the best classification accuracy and the corresponding confusion matrices obtained by the above two classification methods (i.e., LRC (or SVM) versus LeNet). Evaluate and compare their classification performance.

Once the training is completed with the training data-set we can test the models by using the test data-set. will be using the confusion_matrix function to get the result for the confusion matrix and plot the result using matplot library in python - Jupyter.

Classification Accuracy for SVM: A image regarding the svm accuracy is shown in figure 13 with accuracy of 84.27% **Output**

```
Classification report for classifier LinearSVC(C=1.0, class
          intercept_scaling=1, loss='squared_hinge', max_it
          multi_class='ovr', penalty='l2', random_state=Non
          verbose=0):
              precision    recall  f1-score   support

           0       0.95      0.93      0.94       980
           1       0.94      0.98      0.96      1135
           2       0.85      0.88      0.87      1032
           3       0.84      0.85      0.85      1010
           4       0.85      0.92      0.88       982
           5       0.56      0.90      0.69       892
           6       0.84      0.95      0.89       958
           7       0.93      0.86      0.90      1028
           8       0.95      0.37      0.53       974
           9       0.90      0.76      0.82      1009

    accuracy                           0.84     10000
   macro avg       0.86      0.84      0.83     10000
weighted avg       0.86      0.84      0.84     10000
```

Figure 13: Support vector machine Accuracy

confusion matrix for SVM: A image regarding the svm - Confusion matrix is shown in figure 14 with accuracy of 84.27% **Output**

Classification Accuracy for LeNet - CNN:

Accuracy obtained : 0.982399986

confusion matrix - CNN:Displayed in figure 15 Output Hence we can conclude that LeNet - CNN performs far better than SVM from the accuracy of 98.23% far greater than 84.27% in terms of classification which is .

2.5 Attach your code at the end of the report.

Code attached at end....

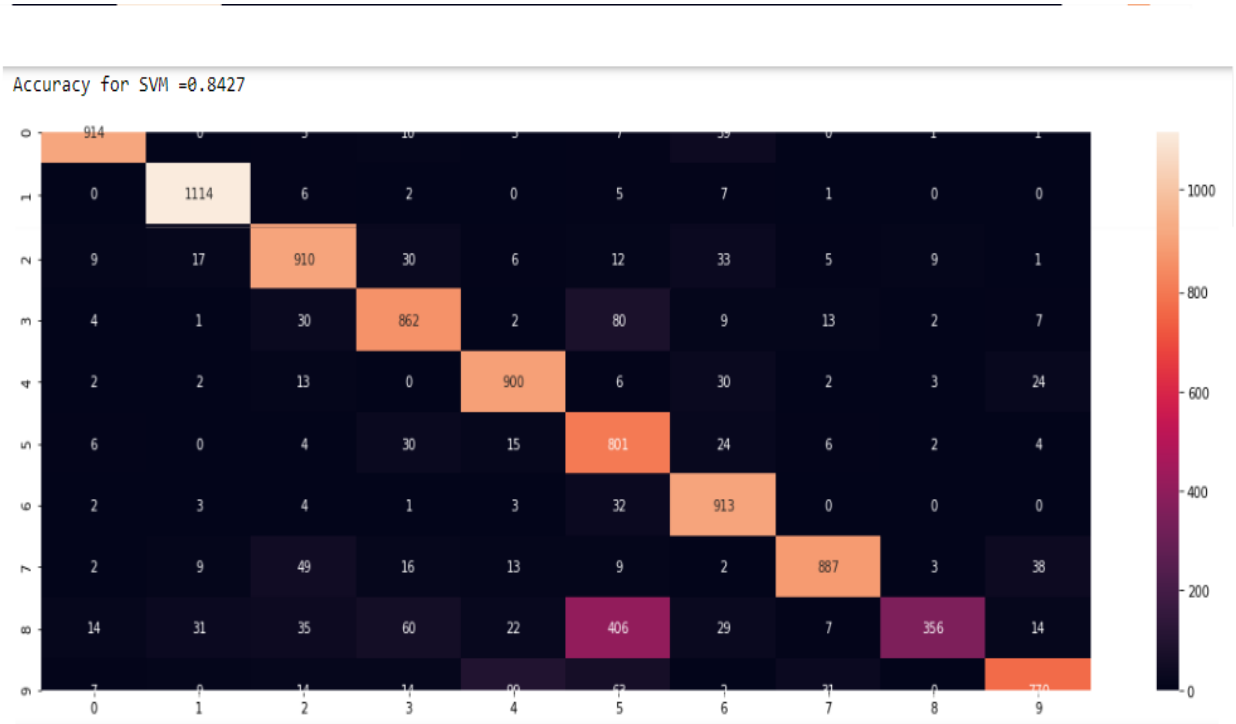


Figure 14: LeNet time

```
[[ 974    0    2    0    0    0    2    2    0    0]
 [   0 1131    0    1    0    1    1    1    0    0]
 [   2    1 1019    1    1    0    1    7    0    0]
 [   0    0    0 1003    0    3    0    2    2    0]
 [   0    0    0    0  977    0    2    1    0    2]
 [   2    0    1    4    0  880    2    1    0    2]
 [   4    2    0    0    1    1  949    0    1    0]
 [   0    4    3    1    0    0    0 1019    0    1]
 [   3    0    0    2    1    1    3    4  959    1]
 [   0    0    0    0   12    3    1    1    1  991]]
```

Figure 15: LeNet Confusion matrix

2.6 References

1.) https://en.wikipedia.org/wiki/MNIST_database
2.) http://yann.lecun.com/exdb/mnist/ 3.) https://scikit-learn.org/stable/modules/generated/sklearn.svm.
4.) https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
5.) https://colab.research.google.com/drive/1CVm50PGE4vhtB5I_a_yc4h5F-itKOVL9scrollTo=1w66ueiLlP
6.) https://en.wikipedia.org/wiki/Convolutional_neural_network

3 Task 3 – Image Classification with a Pre-trained CNN Model

3.1 Describe how you use the pre-trained Inception-BatchNorm network to classify some object images.

Predicting the object in a image by a machine is a very difficult task as the objects sizes varies in each image making it difficult. With a pretrained CNN can able to predict the objects. Steps to be followed are:

1.) download the pre-trained batch inception network from the below link -
http://data.mxnet.io/mxnet/data/Inception.zip

2.) Then load the mxnet and imager package to load and preprocess the images in R.

3.) Load the pre-trained model and the mean image which is used for pre-processing.

4.) Use the below code to load an image from the system or from the imager package:

im < − load.image(system.file(”extdata/parrots.png”, package=”imager”))

plot(im)

The output for the loaded image would be as shown in figure 17:

5.) Preprocessing: before feeding the image to deep network, we need to perform some preprocessing to make the image meet the deep network input requirements, which includes cropping and subtracting the mean shown in image 18.

6.) Defined preprocessing function is used to get the normalized image.

7.) We will now use the predict function to get the probability over classes.

8.) Use the max.col on the transpose of prob to get the class index.

9.) Once it is done read the name of the classes from the synset.txt file.

10.) At last print the predicted name for the image.

11.) And at last print the predicted name for the image.

Figure 16: Display of image in R environment

```
plot(im)

preproc.image <- function(im, mean.image) {
  # crop the image
  shape <- dim(im)
  short.edge <- min(shape[1:2])
  xx <- floor((shape[1] - short.edge) / 2)
  yy <- floor((shape[2] - short.edge) / 2)
  croped <- crop.borders(im, xx, yy)
  # resize to 224 x 224, needed by input of the model.
  resized <- resize(croped, 224, 224)
  # convert to array (x, y, channel)
  arr <- as.array(resized) * 255
  dim(arr) <- c(224, 224, 3)
  # subtract the mean
  normed <- arr - mean.img
  # Reshape to format needed by mxnet (width, height, channel, num)
  dim(normed) <- c(224, 224, 3, 1)
  return(normed)
}

normed <- preproc.image(im, mean.img)
prob <- predict(model, X = normed)
dim(prob)

max.idx <- max.col(t(prob))
max.idx

synsets <- readLines("Inception/synset.txt")
print(paste0("Predicted Top-class: ", synsets[[max.idx]]))
```

Figure 17: Preprossing Used in code

3.2 Describe the provided Caltech256 training and test sets.

The Caltech 256 is considered an improvement to its predecessor, the Caltech 101 dataset, with new features such as larger category sizes, new and larger clutter categories, and overall increased difficulty. This is a great dataset to train models for visual recognition: How can we recognize frogs, cell phones, sail boats and many other categories in cluttered pictures? How can we learn these categories in the first place? Can we endow machines with the same ability?[Ref No. 1]

In train there are 257 objects and 30 images in each set with leads to 7710 images in train.

In test there are 257 objects and 50 images in each set with leads to 12,850 images in train.

Totaling the train and test there are 20,560 images.

3.3 Describe how you use the pre-trained Inception-BatchNorm network to extract feature representation for each image in the Caltech256 dataset.

pre-trained Inception Batch Norm model is prepared using MXNet library in R. MXNet is a flexible and efficient deep learning framework and One of the interesting things that a deep learning algorithm can do is classify real world images. we need to perform some preprocessing to make the image meet the deep network input requirements. Preprocessing includes cropping and subtracting the mean. Then we will classify the image using the predict function to get the probability over the classes. Now read the name of the classes from the synset.txt file and print the predicted result for the image. Now the model is pre trained. Once the model is pre trained we will perform feature extraction for Caltech 256 dataset.

Steps followed and code:

1.) looping through all the 7710 images from the training list that we got from the training dataset one by one.

2.) Now, we need to get feature layer symbol out of internals first. Here we use global_pool_output as shown below:

internals = model$symbol$get.internals()

fea_symbol = internals[[match("global_pool_output", internals$outputs$)]]

3.) Then we will rebuild a new model using the feature symbol as mentioned below:

model2 <− list(symbol = fea_symbol,

arg.params = model$arg.params, aux.params = model$aux.params)

class(model2) <− "MXFeedForwardModel"

4.) Then we can do the predict using the new model to get the internal results. You need to set allow.extra.params = TRUE since some parameters are not used this time.

global_pooling_feature <− predict(model2, X = normed, allow.extra.params = TRUE)

dim(global_pooling_feature)

5.) This way we will train the model on the training set. Now we use the call function and sore the

result of the feature for the training dataset in the csv.
6.) After that we will call the SVM function to train the model on SVM and store the result in a variable that will be used for prediction.

The same process is called for each image of the testing dataset. We will loop through all the 12,850 images one by one and extract the features and build the testing data model and predict the testing result. Then we will use the call functions to bind the test extracted features and store the result in the csv. Once this is done we will call the predict function on the trained SVM model to predict the result of the model.

3.4 **Report the best classification accuracy obtained by the LRC or SVM classifier. List the top 10 pairs of classes that are confused most in the confusion matrix.**

The confusion matrix and classification accuracy using SVM model on the Caltech256 dataset is as shown in Figure 18. We can see that accuracy using kernel = radial is 76.86%. To get the top 10 pairs of classes which are confused most can be obtained by loop over all the 257 classes, compare each and every class like a linear searching and compare with a max element which is assigned as 0. Then we will get the class pairs that are more confused from the confusion matrix.

3.5 References :

1.) https://www.kaggle.com/jessicali9530/caltech256
2.) https://datascience.stackexchange.com/questions/17079/how-can-i-make-big-confusion-matrices-easier-to-read
3.) https://mxnet.incubator.apache.org/tutorials/r/classifyRealImageWithPretrainedModel.html
4.) https://github.com/apache/incubator-mxnet/blob/master/R-package/vignettes/classifyRealImageWithP

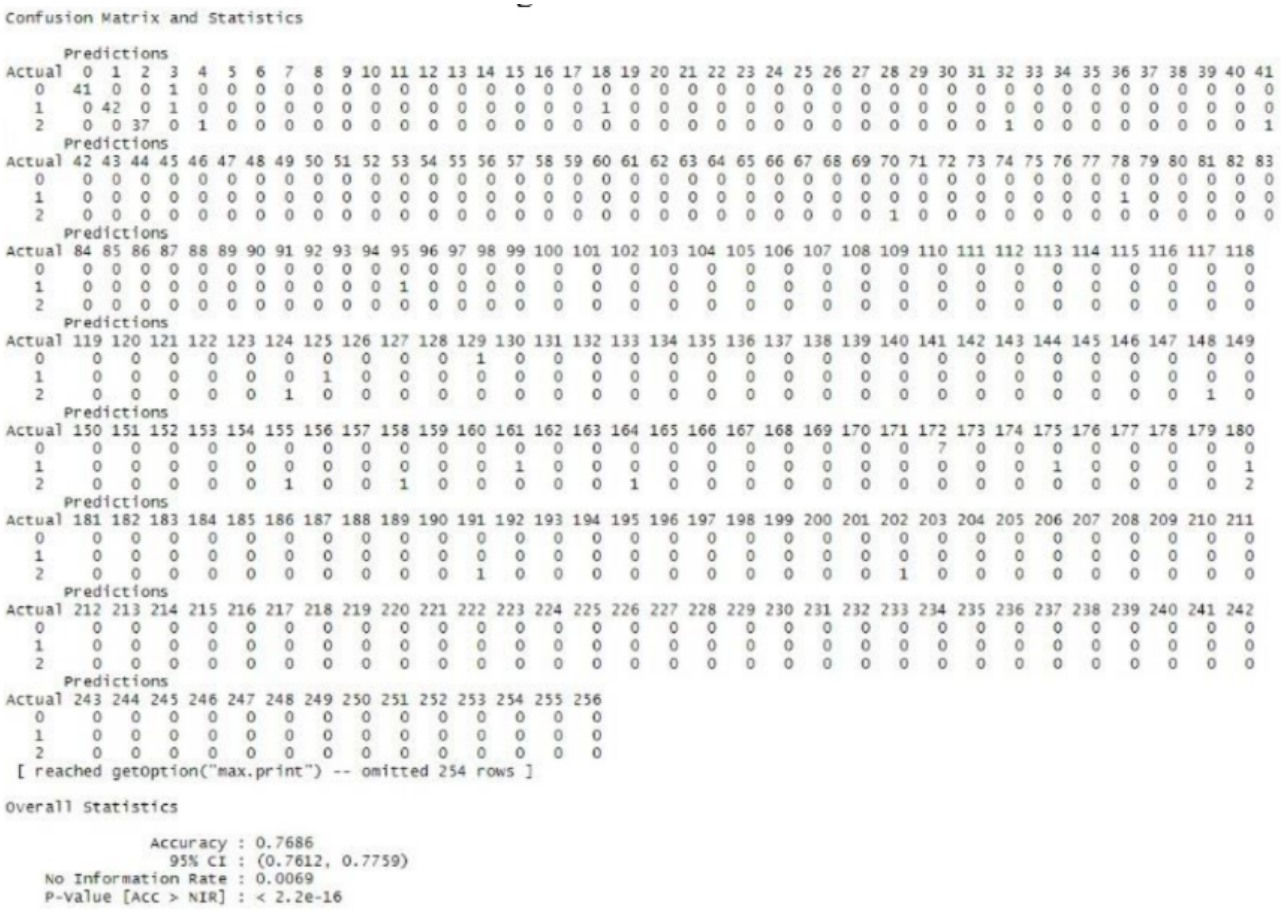
3.6 **Attach your code at the end of the report.**

Code attached at end....

4 Code

4.1 Code for Task - 2

Task 2 is done in python and shown in figures - 19, 20, 21 and 22.



```
Confusion Matrix and Statistics

          Predictions
Actual   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
     0  41  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     1   0 42  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     2   0  0 37  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  1
          Predictions
Actual 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
     0   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     1   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0
     2   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0
          Predictions
Actual 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
     0   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     1   0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     2   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
          Predictions
Actual 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
     0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     1   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     2   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
          Predictions
Actual 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   7   0   0   0   0   0   0   0   0
     1   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   1
     2   0   0   0   0   0   1   0   0   1   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2
          Predictions
Actual 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     2   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0
          Predictions
Actual 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
          Predictions
Actual 243 244 245 246 247 248 249 250 251 252 253 254 255 256
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     2   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 [ reached getOption("max.print") -- omitted 254 rows ]

Overall Statistics

               Accuracy : 0.7686
                 95% CI : (0.7612, 0.7759)
    No Information Rate : 0.0069
    P-Value [Acc > NIR] : < 2.2e-16
```

Figure 18: SVM Caltech256 dataset Confusion matrix and Accuracy

```python
# Task - 2 Venkata Sandeep Kumar Karamsetty - 6228975 - vskk033
import numpy
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
from sklearn import metrics
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
from keras import optimizers
import itertools
import numpy as np
import pandas as pd
import seaborn as sns
import os
from sklearn.linear_model import LogisticRegressionCV

K.set_image_dim_ordering('th')

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

# load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# reshape to be [samples][pixels][width][height]
X_train_CNN = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
X_test_CNN = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')

# normalize inputs from 0-255 to 0-1
X_train_CNN = X_train_CNN / 255
X_test_CNN = X_test_CNN / 255

# one hot encode outputs
y_train_CNN = np_utils.to_categorical(y_train)
y_test_CNN = np_utils.to_categorical(y_test)
```

Figure 19: Task 2 - One

```
num_classes = y_test_CNN.shape[1]

#Reshaping to a vector to process for SVM and LRC
trainSamples, trainWidth, trainHeight = X_train.data.shape
testSamples, testWidth, testHeight = X_test.data.shape

X_train = X_train.reshape((trainSamples,trainWidth*trainHeight))
X_test = X_test.reshape((testSamples,testWidth*testHeight))


# Create a classifier: a support vector classifier and LRC

svmClassifier = LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
             intercept_scaling=1, loss='squared_hinge', max_iter=1000,
            multi_class='ovr', penalty='l2', random_state=None, tol=0.0001, verbose=0)

# We learn the digits on the now
svmClassifier.fit(X_train, y_train)

# Now predict the value of the digit on the second half:
expected = y_test
predicted = svmClassifier.predict(X_test)

print("Classification report for classifier %s:\n%s\n"
      % (svmClassifier, metrics.classification_report(expected, predicted)))

# compute the confusion matrix
cmSVM = confusion_matrix(expected, predicted)

df = pd.DataFrame(cmSVM, index=range(num_classes) )
fig = plt.figure(figsize=(20,7))
heatmap = sns.heatmap(df, annot=True, fmt="d")

#print Accuracy
print("Accuracy for SVM ={}".format(metrics.accuracy_score(expected, predicted)))


#LRC without Cross Validation
lrcClassifier = LogisticRegression(solver = 'lbfgs',multi_class='ovr',random_state = 1)
lrcClassifier.fit(X_train, y_train)

lrcPredicted = lrcClassifier.predict(X_test)

print("Classification report for classifier %s:\n%s\n"
```

Figure 20: Task 2 - two

```
print("Classification report for classifier %s:\n%s\n"
      % (lrcClassifier, metrics.classification_report(expected, lrcPredicted)))

#Compute Confusion Matrix
cmLRC = confusion_matrix(expected, lrcPredicted)

dfLRC = pd.DataFrame(cmLRC, index=range(num_classes) )
fig = plt.figure(figsize=(20,7))
heatmap = sns.heatmap(dfLRC, annot=True, fmt="d")

#Print accuracy
print("Accuracy for LRC ={}".format(metrics.accuracy_score(expected, lrcPredicted)))

#LRC with Cross Validation
lrcClassifierCV = LogisticRegressionCV(solver = 'lbfgs',multi_class='multinomial',rand
lrcClassifierCV.fit(X_train, y_train)

lrcPredictedCV = lrcClassifierCV.predict(X_test)

print("Classification report for classifier %s:\n%s\n"
      % (lrcClassifierCV, metrics.classification_report(expected, lrcPredicted)))

#Compute Confusion Matrix
cmLRCCV = confusion_matrix(expected, lrcPredictedCV)

dfLRCCV = pd.DataFrame(cmLRCCV, index=range(num_classes) )
fig = plt.figure(figsize=(20,7))
heatmap = sns.heatmap(dfLRCCV, annot=True, fmt="d")

#Print accuracy
print("Accuracy for LRCCV ={}".format(metrics.accuracy_score(expected, lrcPredictedCV)

# CNN model
def larger_model():
    # create model
    model = Sequential()
    model.add(Conv2D(30, (5, 5), input_shape=(1, 28, 28), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(15, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
```

Figure 21: Task 2 - three

```
    model.add(Dense(50, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accurac
    return model

# build the model
model = larger_model()
# Fit the model
k=model.fit(X_train_CNN, y_train_CNN, validation_data=(X_test_CNN, y_test_CNN), epochs
# Final evaluation of the model
scores = model.evaluate(X_test_CNN, y_test_CNN, verbose=0)
print("Large CNN Error: %.2f%%" % (100-scores[1]*100))

print('Test loss:', scores[0])
print('Test accuracy:', scores[1])

# Predict the values from the validation dataset
ypred_onehot = model.predict(X_test_CNN)
# Convert predictions classes from one hot vectors to labels: [0 0 1 0 0 ...] --> 2
ypred = np.argmax(ypred_onehot,axis=1)
# Convert validation observations from one hot vectors to labels
ytrue = np.argmax(y_test_CNN,axis=1)


#Classification Report
print("Classification report for classifier %s:\n%s\n"
      % (model, metrics.classification_report(ytrue, ypred)))

#Compute Confusion Matrix
cmCNN = confusion_matrix(ytrue, ypred)

dfCNN = pd.DataFrame(cmCNN, index=range(num_classes) )
fig = plt.figure(figsize=(20,7))
heatmap = sns.heatmap(dfCNN, annot=True, fmt="d")

#Plot for Accuracy and Epochs
plt.plot(k.history['acc'])
plt.plot(k.history['val_acc'])
plt.legend(['Training','Test'])
plt.title('Accuracy')
plt.xlabel('Epochs')
```

Figure 22: Task 2 - four

4.2 Code for Task - 3

Task 3 is done in R and shown in figure 23, 24 and 25.

```r
#Task 3- Venkata Sandeep Kumar Karamsetty - 6228975 - vskk033
#cran <- getOption("repos")
#cran["dmlc"] <- "https://s3-us-west-2.amazonaws.com/apache-mxnet/R/CRAN/"
#options(repos = cran)
#install.packages("mxnet",dependencies = T)
#install.packages('imager')
library(mxnet)
require(imager)
require(e1071)
library(tensorflow)
library(keras)
library(pROC)


setwd("/")

#Use the model loading function to load the model into R and preprocess the data
model = mx.model.load("C:/Users/venka/Desktop/semester - 3/Big data Analytics/Assignment 3/Inception/Inception
mean.img = as.array(mx.nd.load("C:/Users/venka/Desktop/semester - 3/Big data Analytics/Assignment 3/Inception/

#Load and Preprocess the Image
im <- load.image(system.file("extdata/parrots.png", package="imager"))
plot(im)

#Before feeding the image to the deep network, we need to perform some preprocessing to make the image meet th
preproc.image <- function(im, mean.image) {
  # crop the image
  shape <- dim(im)
  if (shape[4] != 3){
    im <- add.color(im)
    shape <- dim(im)
  }
  short.edge <- min(shape[1:2])
  xx <- floor((shape[1] - short.edge) / 2)
  yy <- floor((shape[2] - short.edge) / 2)
  cropped <- crop.borders(im, xx, yy)
  # resize to 224 x 224, needed by input of the model.
  resized <- resize(cropped, 224, 224)
  # convert to array (x, y, channel)
  arr <- as.array(resized) * 255
  dim(arr) <- c(224, 224, 3)
  # subtract the mean
  normed <- arr - mean.img
  # Reshape to format needed by mxnet (width, height, channel, num)
  dim(normed) <- c(224, 224, 3, 1)
  return(normed)
}

# load caltech images directory - test and train dataset
train_imageData <- flow_images_from_directory("datasets/train", generator = image_data_generator(), target_siz
test_imageData <- flow_images_from_directory("datasets/test", generator = image_data_generator(), target_size

#Extracting feature from training dataset
trainFeatures <- list()
y_train <- list()
i<-1
```

Figure 23: Task 3 - one

```
#Extracting feature from training dataset
trainFeatures <- list()
y_train <- list()
i<-1
while(i<=length(train_imageData$filenames))
{
  # load image one by one and process it
  k <- paste("datasets/train/",as.list(train_imageData$filenames[i]),sep = "")
  img <- load.image(k)
  normedDataset = preproc.image(img, mean.img)

  internals = model$symbol$get.internals()
  fea_symbol = internals[[match("global_pool_output", internals$outputs)]]

  trainModel <- list(symbol = fea_symbol,
                     arg.params = model$arg.params,
                     aux.params = model$aux.params)

  class(trainModel) <- "MXFeedForwardModel"

  trainFeatures[[i]] <- predict(trainModel, X = normedDataset, allow.extra.params = TRUE)

  print(i)
  i=i+1
}

d = do.call(rbind, trainFeatures)
#write the result to csv
write.csv(d,"train.csv")
#train the model using SVM
svmModel<-svm(x=d,as.factor(train_imageData$classes),scale=F, kernel="radial", gamma=0.001, cost=10)

#Extracting feature from testing dataset
testFeatures <- list()
i<-1
while(i<=length(test_imageData$filenames))
{
  # load image one by one and process it
  k <- paste("datasets/test/",as.list(test_imageData$filenames[i]),sep = "")
  img <- load.image(k)
  normedDataset1 = preproc.image(img, mean.img)

  internals = model$symbol$get.internals()
  fea_symbol = internals[[match("global_pool_output", internals$outputs)]]

  testModel <- list(symbol = fea_symbol,
                    arg.params = model$arg.params,
                    aux.params = model$aux.params)

  class(testModel) <- "MXFeedForwardModel"

  testFeatures[[i]] <- predict(testModel, X = normedDataset1, allow.extra.params = TRUE)

  print(i)
  i=i+1
}
```

Figure 24: Task 3 - Two

```
110 d1 = do.call(rbind, testFeatures)
111 #write the result to csv
112 write.csv(d1,"test.csv")
113 #Predict the test result
114 svmPredictions <- predict(svmModel,d1)
115 #predictions
116
117 mean(svmPredictions==test_imageData$classes)
118
119 temp <- table("Actual" = test_imageData$classes, "Predictions" = svmPredictions)
120
121 #Draw confusion matrix
122 cm<-confusionMatrix(temp)
123 cm
124 #write CM result
125 write.csv(as.matrix(cm),"conf.csv")
126
```

Figure 25: Task 3 - Three