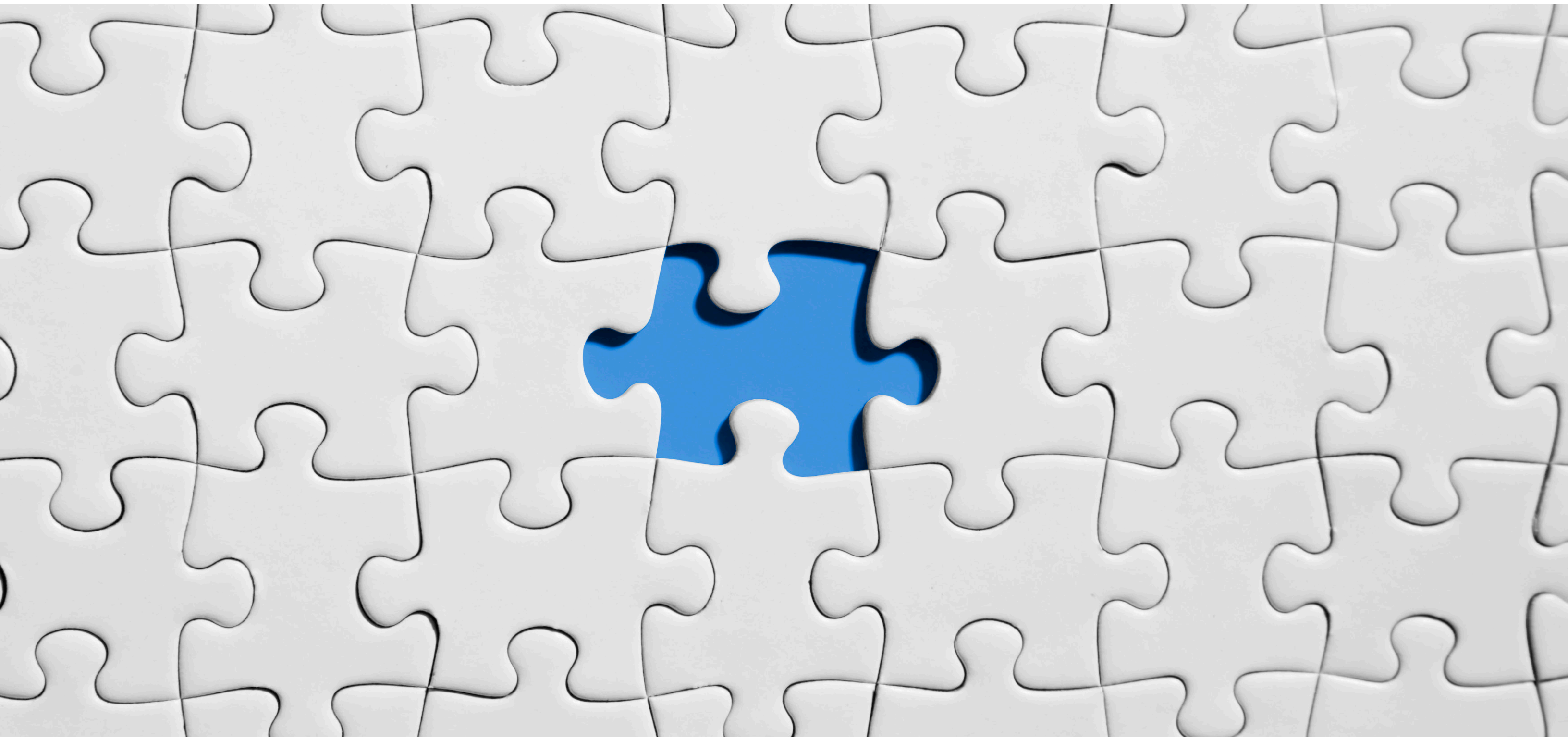
FINE-TUNING DEEP NEURAL NETWORKS



Rajasekhar Nannapaneni

Sr Principal Engineer, Solutions Architect

Dell EMC

Rajasekhar.nannapaneni@dell.com

The Dell Technologies Proven Professional Certification program validates a wide range of skills and competencies across multiple technologies and products.

From Associate, entry-level courses to Expert-level, experience-based exams, all professionals in or looking to begin a career in IT benefit from industry-leading training and certification paths from one of the world's most trusted technology partners.

Proven Professional certifications include:

- Cloud
- Converged/Hyperconverged Infrastructure
- Data Protection
- Data Science
- Networking
- Security
- Servers
- Storage
- Enterprise Architect

Courses are offered to meet different learning styles and schedules, including self-paced On Demand, remote-based Virtual Instructor-Led and in-person Classrooms.

Whether you are an experienced IT professional or just getting started, Dell Technologies Proven Professional certifications are designed to clearly signal proficiency to colleagues and employers.

Learn more at www.dell.com/certification

Abstract

Deep Neural Networks has become increasingly popular and emerged as one of the significant ideas of machine learning in the last decade. Deep Learning is applied to many key problems in various applications which gave efficient results.

In this article, several tuning techniques of deep neural networks are discussed which enhances the performance of the overall deep network. Also, a novel deep neural network is developed and applied on a benchmark dataset “CIFAR-10” to showcase the results and working of these tuning techniques.

The results of this work act as a use case to highlight the need for a world class hardware and infrastructure to apply these techniques in real time on industry level problems and applications.

Table of Contents

Abstract 3
List of Figures 5
Glossary 5
A1.1 Introduction 6
A1.2 Applications of deep learning 6
A1.3 Evolution of Deep learning models 7
B1.1 Working of deep learning models 9
B1.2 Network Internals 10
B1.3 Parameters and Hyperparameters 11
B1.4 Hyperparameter Tuning 11
C1.1 Use Case – CIFAR-10 dataset 13
C1.2 UseCase – Modified ResNet Model 13
C1.2.1 Mathematics behind the modified ResNet model 15
C1.3 Evaluation of the modified ResNet 16
D1.1 Infrastructure needs for deep learning models 17
D1.2 World-class infrastructures 18
D1.3 Conclusion 20
References 21
APPENDIX 22
APPENDIX – Python code for Deep learning – modified ResNet Model for CIFAR-10 22

List of Figures

Figure No.	Figure Caption	Pg.No.
Figure 1	Applications of Deep learning	8
Figure 2	Timeline of Deep learning model evolution	9
Figure 3	ILSVRC Top 5 error % on ImageNet	10
Figure 4	Deep Learning model	11
Figure 5	Visualization of images inside the layers	12
Figure 6	Parameters and Hyperparameters	13
Figure 7	Hyperparameters and their Sensitivity	14
Figure 8	CIFAR-10 dataset	15
Figure 9	Identity and Projection blocks	17
Figure 10	Modified ResNet-18 model	18
Figure 11	Residual Net	18
Figure 12	Model Accuracy and Loss	19
Figure 13	Data Compute Algorithm	21
Figure 14	Infrastructure Requirements	21
Figure 15	Dimensionality per data types	22
Figure 16	Hardware requirements for deep learning	23
Figure 17	Performance of Dell EMC Isilon Ready Solution on ResNet-50	24

Glossary

Abbreviation	Full Form
DL	Deep Learning
RESNET	Residual Network
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
RL	Reinforcement Learning
CIFAR	Canadian Institute for Advanced Research
GPU	Graphics Processing Unit
CPU	Central Processing Unit
FPGA	Field-Programmable Gate Array
ALU	Arithmetic Logic Unit
ML	Machine Learning

A1.1 Introduction

Deep learning has revolutionized the field of machine learning. The advent of deep learning has facilitated efficient solutions for high dimensional problems thus creating newer possibilities and opportunities. Deep learning is being applied across several applications in various domains.

Research in the area of deep learning is very dynamic and progressing at rocket speed every year. Newer models are tested on benchmark datasets to exceed previously set achievements. Section-A of this article highlights some of the fields where deep learning is being applied and will also focus on the evolution of some of the successful deep learning models related to computer vision.

The deep learning models evolved over time have introduced several hyper parameters in each of those networks and it’s important to tune these hyper parameters to obtain higher efficiencies. The hyper parameters involved in the deep learning models and their tuning will be discussed in Section-B.

Section-C explores a deep learning model for a benchmark image dataset called CIFAR-10 which involves fine-tuning techniques and implementation of this model on GPU hardware. This leads to a crucial point on the hardware requirements to run these deep learning models efficiently which will be discussed in Section-D where the need for world class infrastructure is detailed.[4]

A1.2 Applications of deep learning

Deep learning has a wider application in various fields; retail, financial, energy, manufacturing, business, security, telecom, automotive, speech analytics, telecom, etc. In particular, deep learning has outperformed traditional machine learning and human capabilities in the image processing of computer vision.

Some of the image-related applications where deep learning performed extremely well are detection of brain tumor using CT scans, analyzing MRI images, cancer detection based on scan images, optical character recognition, autonomous driving vehicles, manufacturing defect detection through images, etc. Figure 1 gives an overview of deep learning applications.

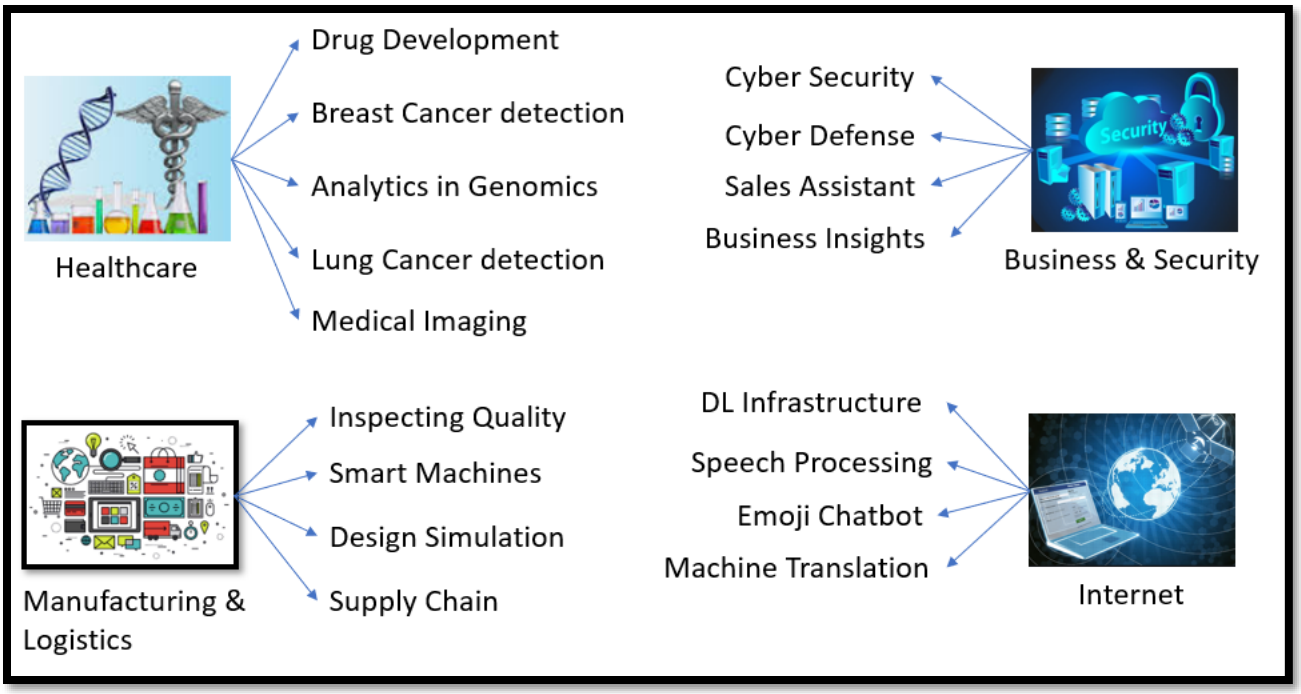


Figure 1: Applications of Deep learning

A1.3 Evolution of Deep learning models

It is beneficial to know how deep learning models have evolved and the various experimentations in history that led to the current state of art models. The idea of artificial intelligence was popularized in the 1940's and it was in 1943 when the artificial neuron was developed which was one of the first step towards neural networks.

The backpropagation algorithm that was proposed by Geoffrey Hinton in 1986 was a key development in the deep learning area and it is used even in the state-of-the-art models today. Yann LeCun is credited with developing one of the breakthrough neural network architectures – LeNet – which has 2 convolution and 3 fully connected layers. This is one of the earliest networks to implement backpropagation algorithm in multi-layer networks with around 60000 parameters. This is also one of the standard networks which has put forth the architecture in layers with a stack of convolution layers.

The timeline detailing the evolution of deep learning models is shown in Figure 2. Observe that Google and Microsoft have come up with state-of-the-art models called GoogleNet (Inception Net) and ResNet.[5]

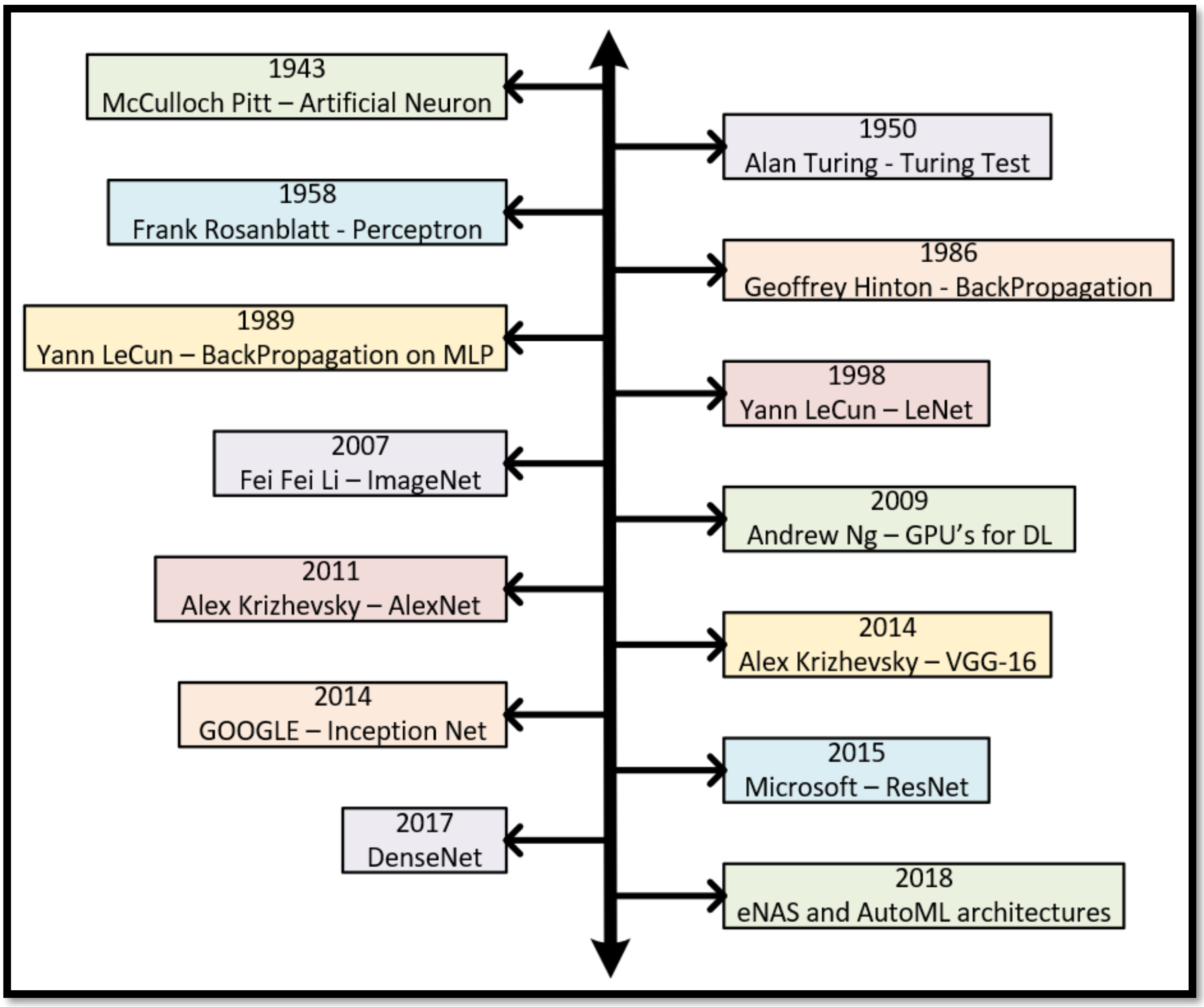

1943
McCulloch Pitt – Artificial Neuron
1950
Alan Turing - Turing Test
1958
Frank Rosanblatt - Perceptron
1986
Geoffrey Hinton - BackPropagation
1989
Yann LeCun – BackPropagation on MLP
1998
Yann LeCun – LeNet
2007
Fei Fei Li – ImageNet
2009
Andrew Ng – GPU's for DL
2011
Alex Krizhevsky – AlexNet
2014
Alex Krizhevsky – VGG-16
2014
GOOGLE – Inception Net
2015
Microsoft – ResNet
2017
DenseNet
2018
eNAS and AutoML architectures

Figure 2: Timeline of Deep learning model evolution

These are multi-layered networks where ResNet by Microsoft is about 152 layers and has surpassed human performance in 2015 for some of the benchmark datasets. The performance indicator in these models is its accuracy to classify the given multiclass image datasets, like CIFAR-10.

The annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) introduces many new models which aim for efficient architectures from an accuracy perspective on the benchmark datasets. Figure 3 shows the top 5 error % obtained by various models in different years. [7]

Evolution of these models over the years not only showcases the improvement in classification accuracies but also the introduction of various techniques and novel architectures. Every model offered something novel to previous models that has made it successful. Until 2011, shallow network models from traditional computer vision had been on the top of the list.

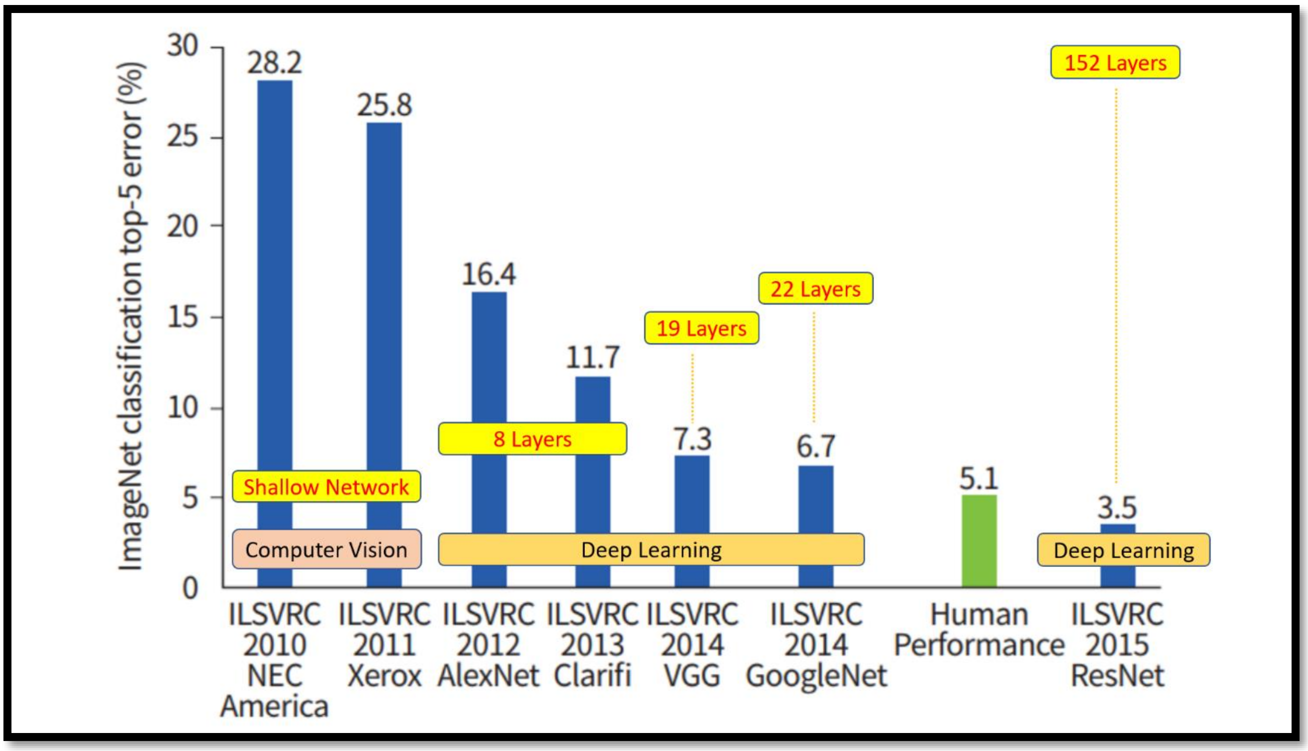


Figure 3: ILSVRC Top 5 error % on ImageNet

The AlexNet in 2012 is one of the first deep learning networks to win ILSVRC. In subsequent years deep learning models have outperformed traditional computer vision models and by 2015 deep learning models have even outperformed human performance in classifying images. [8]

Notice in Figure 3 that as the newer models were introduced, the number of layers kept increasing. This conveys that deep learning models with deeper layers are more successful than the wider layered networks. The introduction of these varied architectures introduces newer hyper parameters and it is equally important to fine tune these hyper parameters to achieve success.

B1.1 Working of deep learning models

Deep learning models are used in supervised, unsupervised learning and reinforcement learning paradigms. In this section, a supervised classification task for image data will be discussed. Any typical deep learning model requires training data for training the network and this training data will be labeled data for supervised learning tasks. One such training data set is CIFAR-10 dataset which contains around 60000 images in 10 different classes.

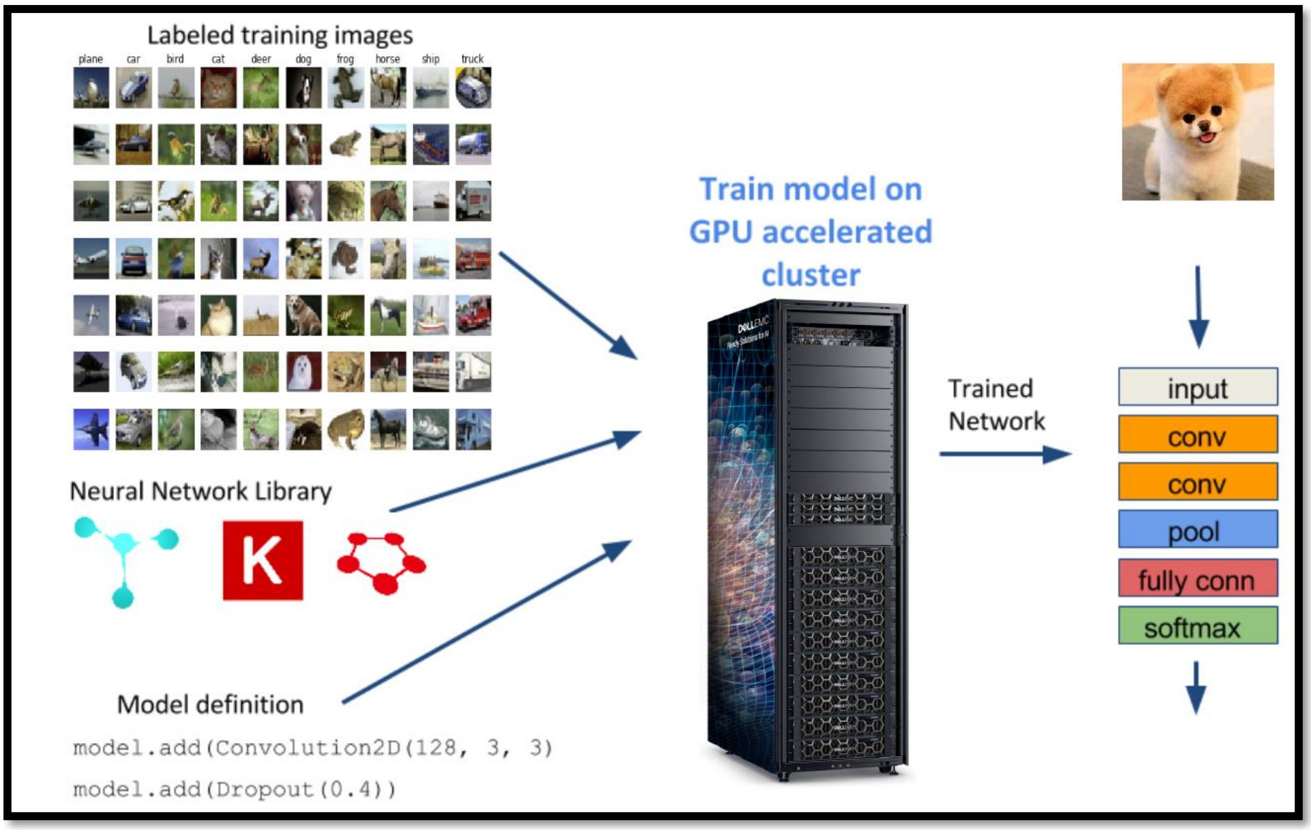


Figure 4: Deep learning model

A neural network library such as Keras can be used to formulate the model definition and the model along with training dataset will be trained on GPU-enabled hardware. Once the model is trained, its accuracy will be evaluated using testing data. This entire set of tasks depicted in Figure 4 require a great deal of parameter tuning in order to optimize the network. Prior to understanding the parameters to be fine-tuned, it's important to understand what happens to the images when traversed in the network.

B1.2 Network Internals

The input images to a deep learning network traverse several layers in the network. The presence of several layers helps the network learn the details in the images. The initial layers of the network capture the edges in the input images and the next set of layers learn the gradients of the images. As the input images progress toward the next layers, the network learns the textures, patterns, parts of objects and eventually, the overall objects itself which helps in classifying the classes.[1]

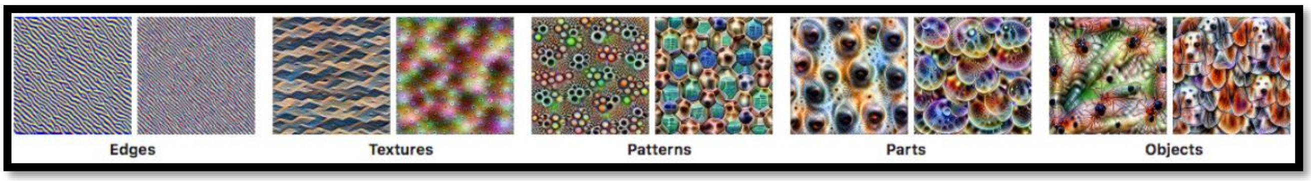


Figure 5: Visualization of images inside the layers

The network first learns low-level features and as the inputs traverse the network, it learns mid-level and high-level features that will make the network learn to classify the classes of images.

B1.3 Parameters and Hyperparameters

The deep learning network design consists of several building blocks such as layers, optimizers, activation functions, etc. The model has several parameters in the form of weights which depends on the complexity of the model. The training of the network involves changing the weights of the model via backpropagation algorithm to minimize the loss. [9]

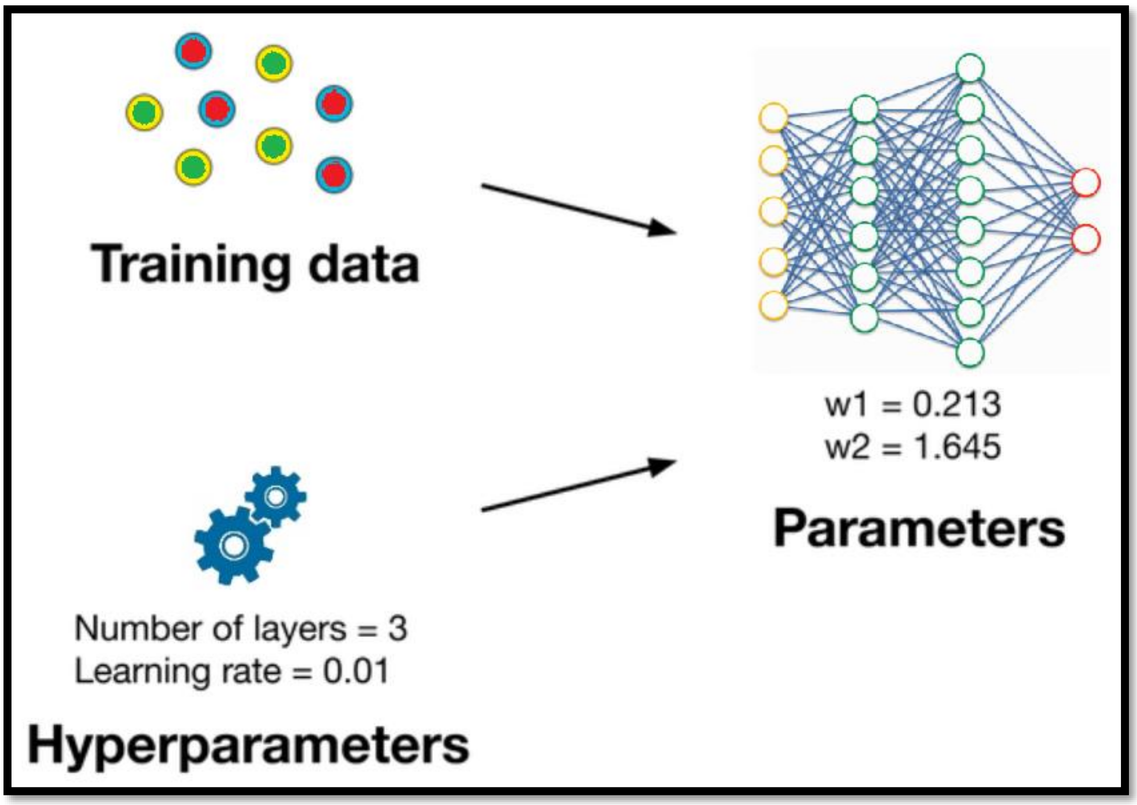


Figure 6: Parameters and Hyperparameters

Apart from the parameters of the network, there are several hyperparameters of the network which dictate its performance.

B1.4 Hyperparameter Tuning

It is important to tune the hyperparameters of a deep learning network as optimizing these hyperparameters would enhance the overall network performance. Some of the hyperparameters are mentioned in this section.

Layers: Number of layers is decided based on the way edges/gradients -> textures -> patterns -> parts of objects -> objects are obtained.

Receptive Field: Depending on the size of input images, one needs to estimate the receptive field needed to obtain edges, textures, patterns, parts of objects and objects.

MaxPooling: Helps reduce the number of layers required in a network by reducing the dimensionality by 75%.

SoftMax: Softmax tries to squish the output between the required range.

Image Normalization: Ensures that all the pixel intensities are within the required range 0-1.

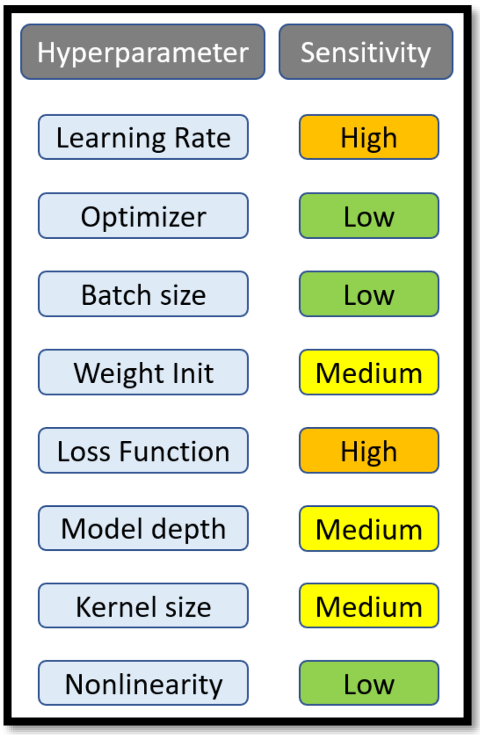
Batch Normalization: Ensures that every feature obtained through a kernel has the same weightage after each convolution layer.

DropOut: Dropouts will resolve overfitting issues by randomly turning off some of the kernels. This will allow the network to learn new features.

Batch Size: The batch size usually depends on the complexity/quantity of images and is also dependent on hardware. Greater batch size at constant learning rate will make the training faster but requires powerful hardware.

Learning Rate: The rate at which the network learns can be tweaked using learning rate.

Adam vs SGD Optimizer: The choice of optimizer impacts the learning rate.



Hyperparameter	Sensitivity
Learning Rate	High
Optimizer	Low
Batch size	Low
Weight Init	Medium
Loss Function	High
Model depth	Medium
Kernel size	Medium
Nonlinearity	Low

Figure 7: Hyperparameters and their Sensitivity

Kernel Size: Kernels are unique individual basic building blocks of an image. These are also called feature extractor as they constitute unique features of an image. Some examples are vertical edges or horizontal edges.

LR schedule: As the number of epoch increase, the network would have learnt more than at the state it started and hence it requires lower learning rate comparative to beginning of training. Thus, an adaptive learning rate would enhance training quality.

Each of the hyperparameters has its own sensitivity towards the performance of the deep learning network, as depicted in Figure 7.

C1.1 Use Case – CIFAR-10 datasetThe hyperparameters tuning and working can be observed in this section. A modified ResNet model has been developed which is trained on CIFAR-10 dataset shown in Figure 8. This dataset is one of the benchmark datasets that has images of 10 different classes. It has 60000 images of size 32x32 pixels each.

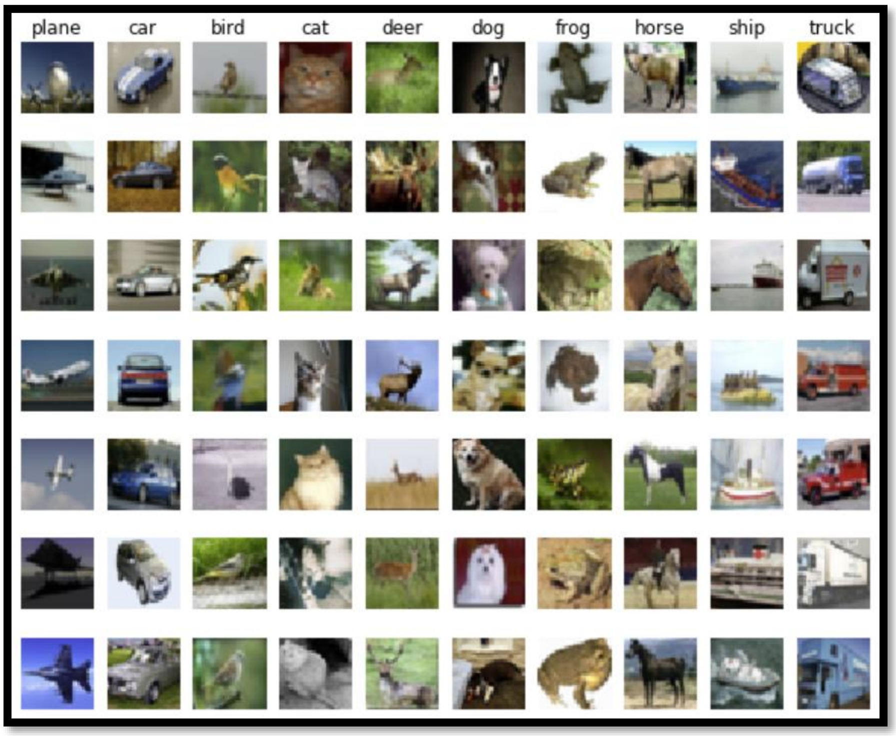


Figure 8: CIFAR-10 dataset

The CIFAR-10 dataset is divided into 50000 images for training and 10000 images for testing the model.

C1.2 UseCase – Modified ResNet Model

ResNet model was originally developed by Microsoft team and it has created a breakthrough in the classification accuracies in ILSVRC challenge 2015 which exceeded human performance.

ResNet is more a deep layered model than a wide layered model. While, In theory, any function can be represented by a huge single layer, this leads to overfitting the data as per the universal approximation theorem. Thus, most successful models are deeper than wide and hence, the modified ResNet model developed in this section is also deeper.

The modified ResNet model developed has similarities from ResNet-18 model released by Microsoft. The similarities include the presence of identity block and projection blocks as illustrated in Figure 9. The identity blocks have skip level connections which help to provide varied receptive fields towards the deeper layers. [3]

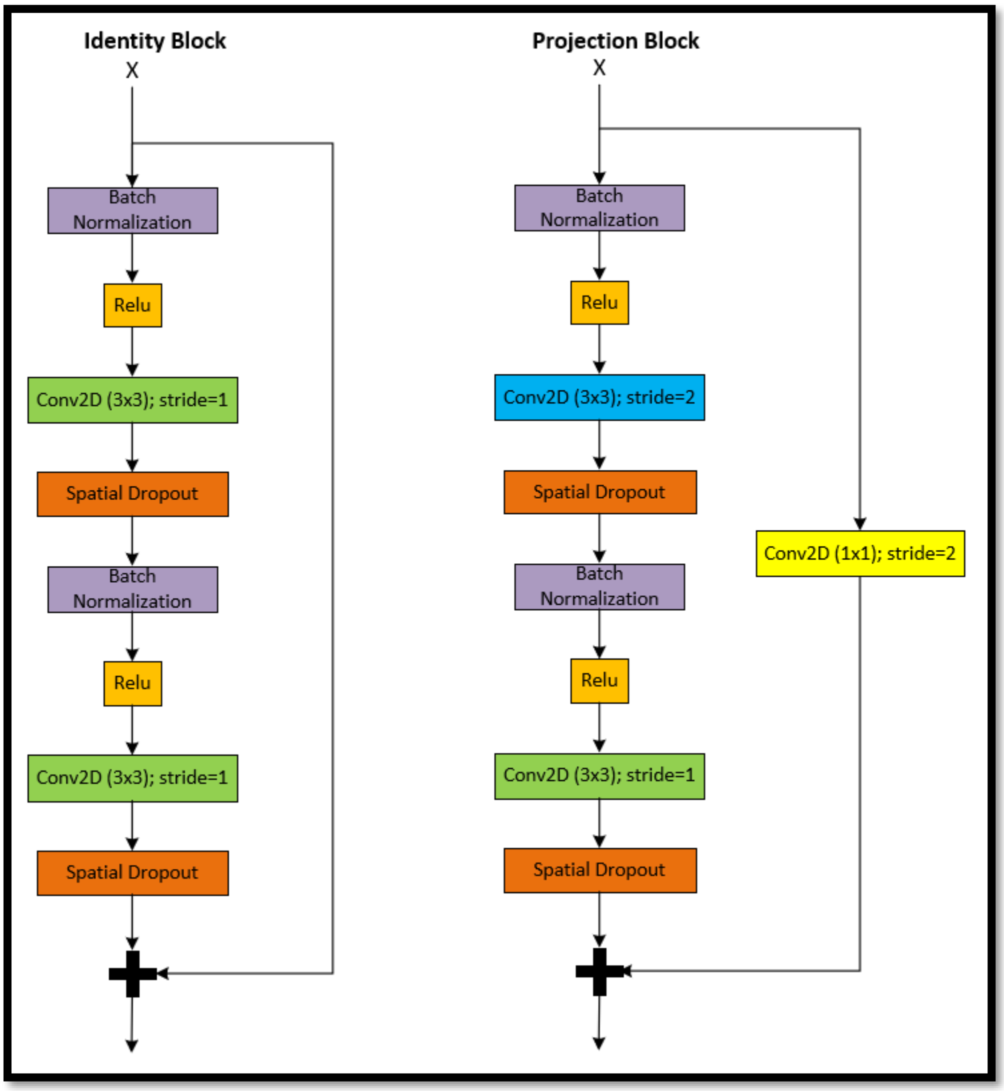


Figure 9: Identity and Projection blocks

The 32x32x3 color images were padded with zeros to conserve the features on the edge of the images and then passed through a series of convolution layers. The input features also pass through a series of identity and projection blocks for extracting edges, gradients, textures, parts of objects and objects thus providing varied receptive fields to deeper layers for better classification.[3]

Activation function Relu is used throughout and towards the end, SoftMax function is used to squish the output between a defined range. Figure 10 illustrates the overall modified ResNet developed. The model developed is quite powerful in extracting the features of input images and it uses the backpropagation algorithm.

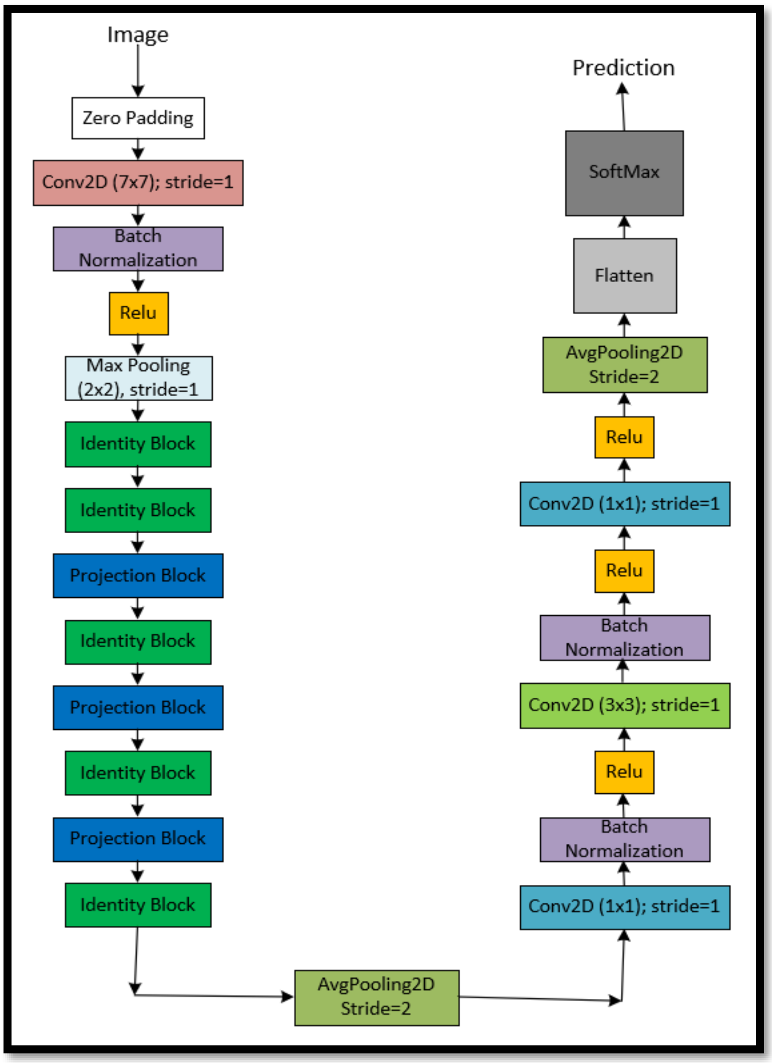


Figure 10: Modified ResNet-18 model

C1.2.1 Mathematics behind the modified ResNet model

Figure 11 depicts the residual network which form the foundation of the ResNet.

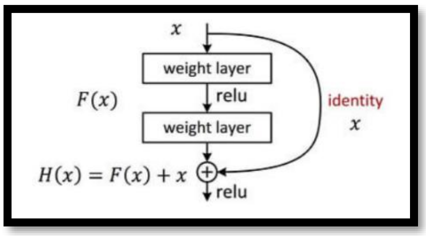


Figure 11: Residual Net

Input x is processed between 2 convolution layers and F(x) is obtained. Finally, the output of the residual network H(x) is obtained by addition of x and F(x).

$$F(x) = \text{Output} - \text{Input} = H(x) - x \tag{1}$$

$$H(x) = F(x) + x \tag{2}$$

The backpropagation in ResNet for one layer is computed by (3)

$$x_{l+1} = \lambda_l x_l + \mathcal{F}(x_l, W_l) \tag{3}$$

$$x_L = \left(\prod_{i=l}^{L-1} \lambda_i\right) x_l + \sum_{i=l}^{L-1} \mathcal{F}(x_i, W_i) \tag{4}$$

The backpropagation in ResNet for L layers from i-th layer is given by (4).

C1.3 Evaluation of the modified ResNet

The modified ResNet is implemented in Google Collaboratory which provides GPU infrastructure to train the network. Figure 12 shows the model accuracy and loss for the training and validation data for 50 epochs run. The training accuracy reached 96.4% and validation accuracy reached up to 88.19%. The code for the model is partially mentioned in the Appendix.

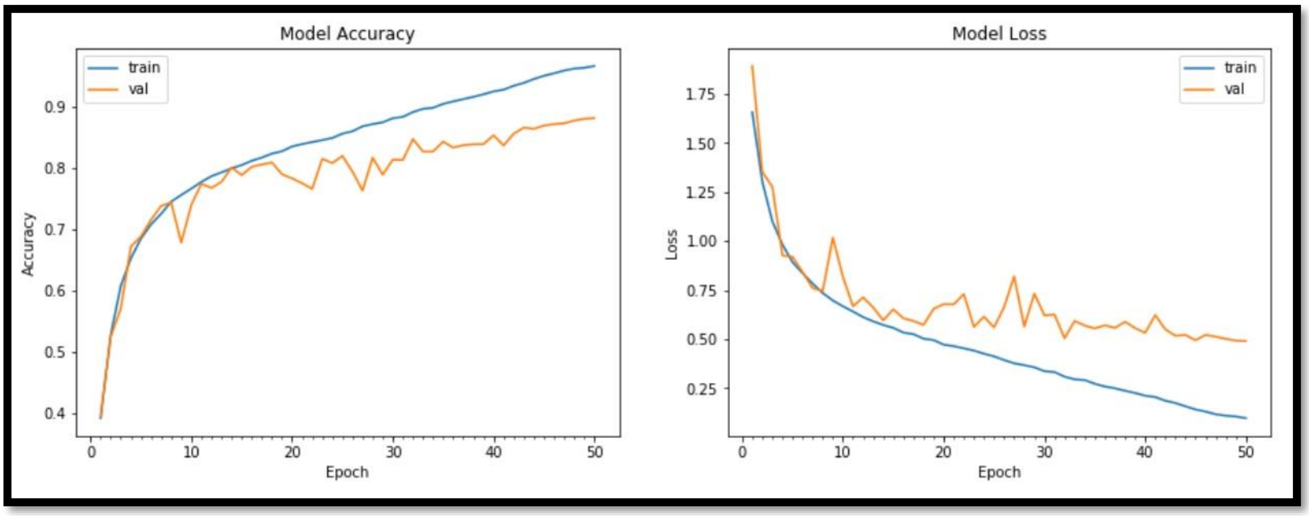


Figure 12: Model Accuracy and Loss

The reason GPU infrastructure is needed is because the network has approximately 3 million parameters to be trained which would need several parallel operations to take place.

Total params: 2,857,930; Trainable params: 2,854,218; Non-trainable params: 3,712

The model training took around 76 minutes due to the utilization of GPU infrastructure. To train with CPU architectures would have taken several hours. This use case showcases the need for a GPU-enabled infrastructures to train deep learning models.

D1.1 Infrastructure needs for deep learning models

The deep learning model developed in the previous section was trained on image data and that required GPU-enabled infrastructure during training. GPU's provide high compute densities compared to CPU's and one of the primary reasons for the use of GPU's for deep learning models is that they are built for parallel operations. The deep learning models require a lot of matrix multiplications which can be performed through several arithmetic-logic units (ALU's) parallelly in a GPU. The presence of more ALU's in GPU's is the reason for parallel operations. In addition, GPU's provide high throughput and high tolerance to latencies. Thus, GPU-enabled hardware is pivotal to handle large datasets in deep learning models.

Figure 13 depicts that as the network deepens, the number of parameters also increases rapidly and thus requires more computational resources.

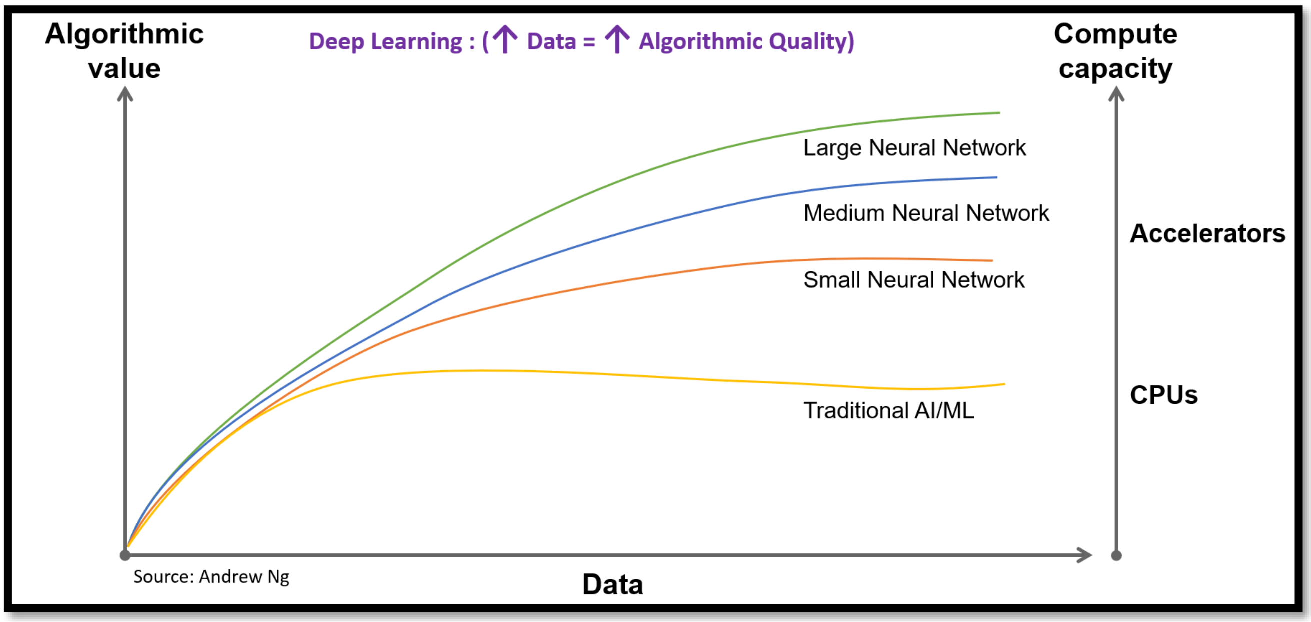


Figure 13: Data Compute Algorithm

It can be observed that large datasets are usually unstructured, mostly image/video data and tend to have high capacity and processing power requirements. Figure 14 illustrates the capacity and performance requirements for business intelligence, machine learning and deep learning models.[6]

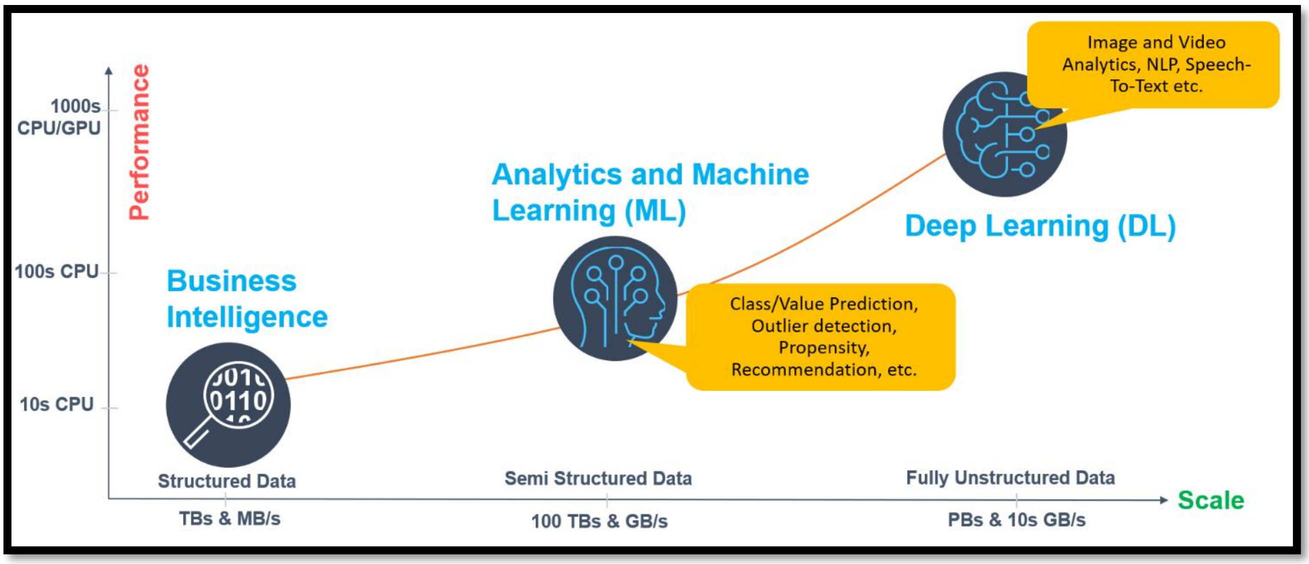
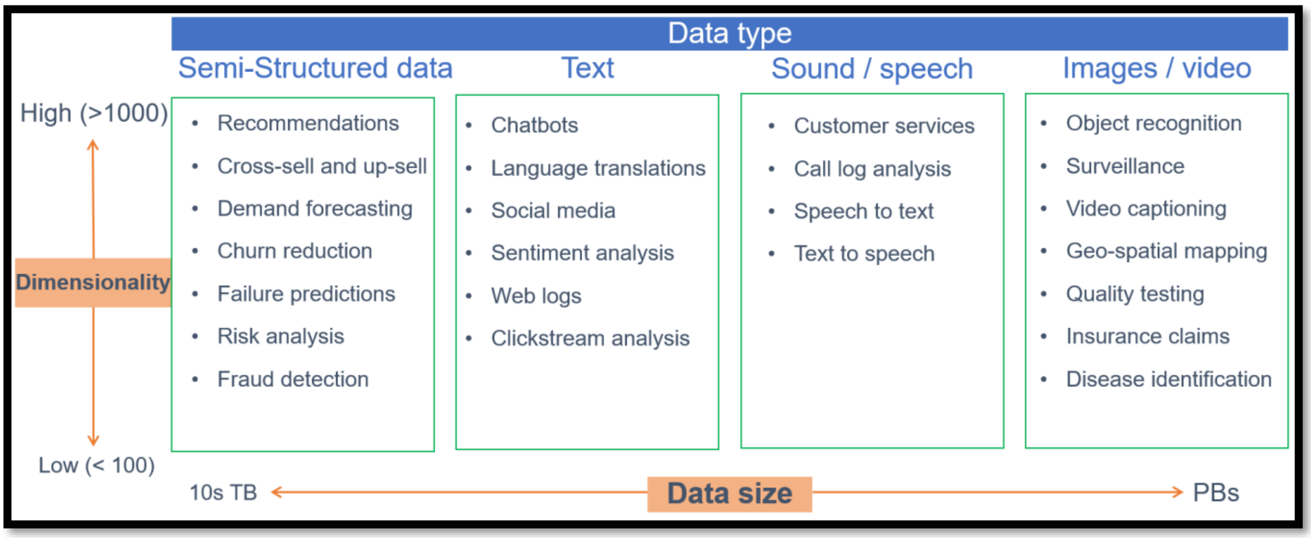


Figure 14: Infrastructure Requirements

Capacity and performance requirements also depend on the dimensionality of the problem and the data size as shown in Figure 15. One can relate that the traditional machine learning problems would be addressing low dimensional and low data size problems while the high dimensional and high data size problems are addressed by deep learning models.[2]



Data type

Semi-Structured data	Text	Sound / speech	Images / video
Recommendations	Chatbots	Customer services	Object recognition
Cross-sell and up-sell	Language translations	Call log analysis	Surveillance
Demand forecasting	Social media	Speech to text	Video captioning
Churn reduction	Sentiment analysis	Text to speech	Geo-spatial mapping
Failure predictions	Web logs		Quality testing
Risk analysis	Clickstream analysis		Insurance claims
Fraud detection			Disease identification

Figure 15: Dimensionality per data types

D1.2 World-class infrastructures

The Google Collaboratory that was leveraged in training the modified ResNet model in Section C provides GPU access for a duration of 12 hours and hence, serves the purposes for some academicians. However, industrial and real-time use cases and applications needs to depend on a robust world-class infrastructure as the hardware requirements are unique and huge as seen in Figure 16.[6]

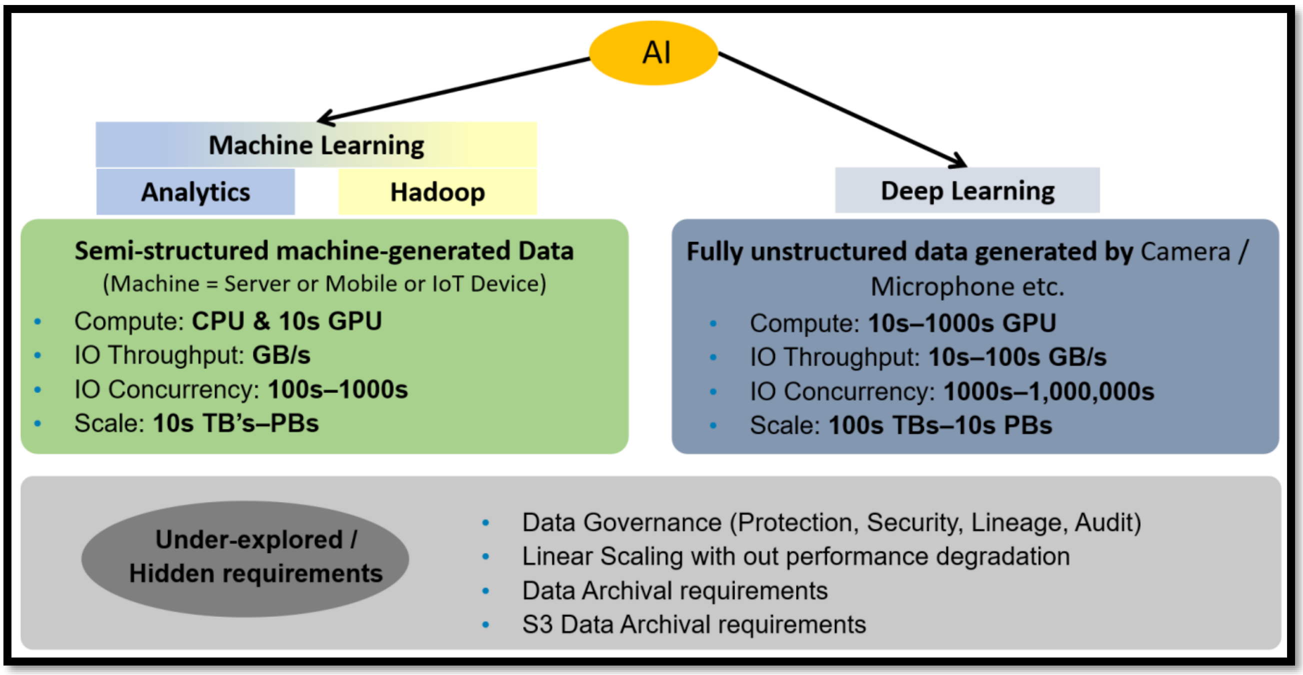

AI
Machine Learning
Analytics
Hadoop
Deep Learning
Semi-structured machine-generated Data
(Machine = Server or Mobile or IoT Device)
Compute: CPU & 10s GPU
IO Throughput: GB/s
IO Concurrency: 100s–1000s
Scale: 10s TB's–PBs
Fully unstructured data generated by Camera / Microphone etc.
Compute: 10s–1000s GPU
IO Throughput: 10s–100s GB/s
IO Concurrency: 1000s–1,000,000s
Scale: 100s TBs–10s PBs
Under-explored / Hidden requirements
Data Governance (Protection, Security, Lineage, Audit)
Linear Scaling with out performance degradation
Data Archival requirements
S3 Data Archival requirements

Figure 16: Hardware requirements for deep learning

From a business perspective one can leverage heterogenous hardware including compute, networking and storage for running deep learning models. However, we find that the ready solutions for deep learning from some vendors are performing well.

For example, the Dell EMC Ready Solution for deep learning delivered tremendous results on benchmark ImageNet dataset as shown in Figure 17. These ready solutions include Dell EMC PowerEdge C4140 servers with 4 Tesla V100 GPU's. The solution also contains Dell EMC Isilon F800 storage hardware which comes with 4 nodes. Thus, Dell EMC Ready Solutions for AI are the best fit for training large datasets on deep learning models.[2]

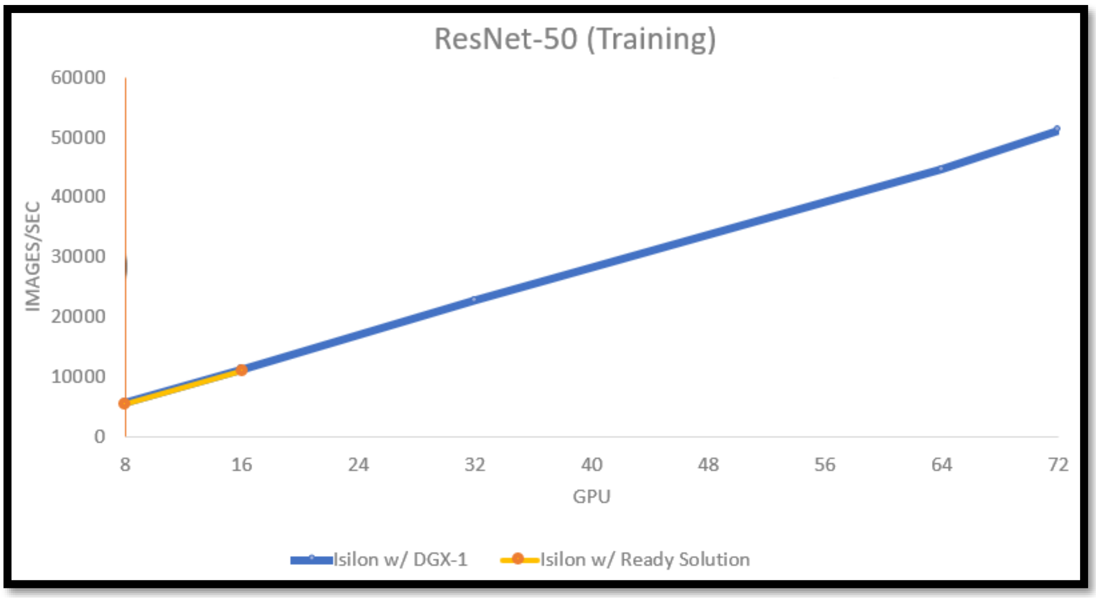


Figure 17: Performance of Dell EMC Isilon Ready Solution on ResNet-50

D1.3 Conclusion

The tremendous opportunities that deep learning enables in various fields will continue to grow as more breakthrough models are created. Once the models are created, various fine-tuning techniques discussed in this article could help achieve expected performance and results.

The modified ResNet model developed in this article trained on benchmark dataset CIFAR-10 proves the necessity of a world-class hardware for deep learning applications. The growing need of deep learning applications will in turn create greater need for GPU- or FPGA-enabled hardware to meet performance and capacity demands.

A comparison is drawn between the performance of separate hardware components and Dell EMC Ready Solution. Designed to readily plug-in for training deep learning models in real-time, the scalable, robust and easily configurable Dell EMC Ready Solution clearly demonstrates its ability perform better in meeting business needs.

References

[1] “Canziani, Alfredo & Paszke, Adam & Culurciello, Eugenio. (2016). An Analysis of Deep Neural Network Models for Practical Applications.”

[2] "Dell EMC Isilon Artificial Intelligence", “Delltechnologies.com, 2020. [Online]. Available: https://www.delltechnologies.com/en-af/solutions/artificial-intelligence/ai-with-isilon/index.htm. [Accessed: 27- Jan- 2020].”

[3] “He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Identity Mappings in Deep Residual Networks. 9908. 630-645. 10.1007/978-3-319-46493-0_38.”

[4] “Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).”

[5] “LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.”

[6] "Ready Solutions for Artificial Intelligence (AI)", “Delltechnologies.com, 2020. [Online]. Available: https://www.delltechnologies.com/en-in/solutions/data-analytics/machine-learning/ready-solutions-for-ai.htm#scroll=off. [Accessed: 27- Jan- 2020].”

[7] “Russakovsky, Olga & Deng, J. & Su, Hao & Krause, J. & Satheesh, Sanjeev & Ma, S. & Huang, Z. & Karpathy, A. & Khosla, A. & Bernstein, M. (2015). ImageNet Large Scale Visual Recognition Challenge. Int. J. Comput. Vis. 115. 1-42.”

[8] “Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.”

[9] “Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1-9.”

APPENDIX

APPENDIX – Python code for Deep learning – modified ResNet Model for CIFAR-10

```
"import time
import numpy as np
import keras
import keras.utils
import matplotlib.pyplot as plt
% matplotlib inline
np.random.seed(2017)
from keras import backend as K
from keras.models import Sequential, Model
from keras.layers.convolutional import Convolution2D, MaxPooling2D, SeparableConv2D, Conv2D
from keras.layers import Activation, Flatten, Dense, Dropout, Input, concatenate, ZeroPadding2D, Dropout, AveragePooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers import SpatialDropout2D, Add, GlobalAveragePooling2D
from keras.utils import np_utils
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping, LearningRateScheduler
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import log_loss, roc_auc_score, accuracy_score
from keras.losses import binary_crossentropy
from keras.metrics import binary_accuracy
from keras.callbacks import *
from keras.optimizers import Adam, SGD
from keras import regularizers
from keras.callbacks import Callback
import cv2
from keras.datasets import cifar10
import tflearn
from tflearn.data_augmentation import ImageAugmentation"

"# GRADED FUNCTION: identity_block


def identity_block(X, filters, stage, block):
  #weight_decay = 5e-4
  # defining name basis
  conv_name_base = 'res' + str(stage) + block + '_branch'
  bn_name_base = 'bn' + str(stage) + block + '_branch'


  # Retrieve Filters
  F1, F2 = filters
  # Save the input value. You'll need this later to add back to the main path.
  X_shortcut = X"
  "# First component of main path
  X = BatchNormalization(axis = 3, name = bn_name_base + '1a')(X)
```

```
    X = Activation('relu')(X)
    #X = Conv2D(filters = F1, kernel_size = (3, 3), strides = (1,1), padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), 
name = conv_name_base + '2a')(X)
    X = Conv2D(filters = F1, kernel_size = (3, 3), strides = (1,1), padding = 'same',  name = conv_name_base + '2a')(X)
    X=SpatialDropout2D(0.1)(X)
    # Second component of main path (≈3 lines)
    X = BatchNormalization(axis = 3, name = bn_name_base + '1b')(X)
    X = Activation('relu')(X)
    #X = Conv2D(filters = F2, kernel_size = (3, 3), strides = (1,1), padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), 
name = conv_name_base + '2b')(X)
    X = Conv2D(filters = F2, kernel_size = (3, 3), strides = (1,1), padding = 'same',  name = conv_name_base + '2b')(X)
    X=SpatialDropout2D(0.1)(X)
   # Final step: Add shortcut value to main path, and pass it through a RELU activation (≈2 lines)
    X = Add()([X, X_shortcut])
    return X

# GRADED FUNCTION: projection_block
def projection_block(X, filters, stage, block):
    weight_decay = 5e-4
    # defining name basis
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'
   # Retrieve Filters
    F1, F2 = filters
   # Save the input value. You'll need this later to add back to the main path.
    X_shortcut = X

    # First component of main path
    X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
    X = Activation('relu')(X)
    #X = Conv2D(filters = F1, kernel_size = (3, 3), strides = (2,2), padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), 
name = conv_name_base + '2a')(X)
    X = Conv2D(filters = F1, kernel_size = (3, 3), strides = (2,2), padding = 'same', name = conv_name_base + '2a')(X)
    X=SpatialDropout2D(0.1)(X)
    # Second component of main path (≈3 lines)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
  X = Activation('relu')(X)
    #X = Conv2D(filters = F2, kernel_size = (3, 3), strides = (1,1), padding = 'same', kernel_regularizer=regularizers.l2(weight_decay), 
name = conv_name_base + '2b')(X)
    X = Conv2D(filters = F2, kernel_size = (3, 3), strides = (1,1), padding = 'same', name = conv_name_base + '2b')(X)”
   “X=SpatialDropout2D(0.1)(X)
    # Third component outside main path
    X_shortcut = Conv2D(filters = F1, kernel_size = (1, 1), strides = (2,2), padding = 'same', name = conv_name_base +
'2c')(X_shortcut)
    # Final step: Add shortcut value to main path, and pass it through a RELU activation (≈2 lines)
    X = Add()([X, X_shortcut])
    return X

x0=Input(shape=(32, 32, 3))
x1=ZeroPadding2D((3,3))(x0)
x1 = Conv2D(32, (7, 7), strides = (1, 1), name = 'conv1',padding='same')(x1)
x1=BatchNormalization()(x1)
x1= Activation('relu')(x1)
x1 = MaxPooling2D((2, 2), strides=(1, 1), name = 'pooling1')(x1)
x2 = identity_block(x1, [32,32], stage=1, block='a')
x3 = identity_block(x2, [32,32], stage=1, block='b')
#x3=Conv2D(filters = 64, kernel_size = (1, 1))(x3)
x3 = projection_block(x3, [64,64], stage=2, block='a')
x4 = identity_block(x3, [64,64], stage=2, block='b')
#x4=Conv2D(filters = 32, kernel_size = (1, 1))(x4)
x4 = projection_block(x4, [128,128], stage=3, block='a')
x5 = identity_block(x4, [128,128], stage=3, block='b')
#x5=Conv2D(filters = 32, kernel_size = (1, 1))(x5)
x5 = projection_block(x5, [256,256], stage=4, block='a')
x6 = identity_block(x5, [256,256], stage=4, block='b')
#x6=Conv2D(64, (3, 3), activation='relu')(x6)
#x6=BatchNormalization()(x6)
#x6=Conv2D(10, (1, 1), padding="same",activation='relu')(x6)
#x6=MaxPooling2D((2, 2),padding="same")(x6)
x6 = AveragePooling2D(strides=(2,2))(x6)
x6=Conv2D(filters = 64, kernel_size = (1, 1), strides=(1,1),padding="same")(x6)
x6=BatchNormalization()(x6)
x6 = Activation('relu')(x6)
x6=Conv2D(64, (3, 3), padding="same")(x6)
x6=BatchNormalization()(x6)
x6 = Activation('relu')(x6)
x6=Conv2D(10, (1, 1), padding="same",activation='relu')(x6)
x6 = AveragePooling2D(strides=(2,2))(x6)
seq = Flatten()(x6)
predictions = Activation('softmax')(seq)
model = Model(inputs=x0, outputs=predictions, name='ResNet')
model.summary()”

“from IPython.display import SVG
```

```
from keras.utils.vis_utils import model_to_dot
SVG(model_to_dot(model).create(prog='dot', format='svg'))
batch_size = 128
epochs=50

model.compile(loss='categorical_crossentropy', optimizer=SGD(lr=0.3, momentum=0.9, nesterov=False), metrics=['accuracy'])
datagen_cut = ImageDataGenerator(zoom_range=0.0,horizontal_flip=True, preprocessing_function=get_random_eraser(p=0.5,
s_l=0.02, s_h=0.4, r_1=0.3, r_2=1/0.3,  v_l=0, v_h=1, pixel_level=False))

sched = OneCycle(min_lr=0.07, max_lr=0.9, min_mtm = 0.75, max_mtm = 0.9, annealing_stage=0.08, annealing_rate=0.009,
      training_iterations=np.ceil(((X_train.shape[0]*epochs)/(batch_size))))

mcp_save = ModelCheckpoint('.mdl_wts.hdf5', save_best_only=True, monitor='val_acc', verbose=1)
# train the model
start = time.time()
# Train the model
model_info = model.fit_generator(datagen_cut.flow(X_train, Y_train, batch_size = 128),  samples_per_epoch = Y_train.shape[0],
nb_epoch = 50,  validation_data = (X_test, Y_test), callbacks=[sched,mcp_save])
end = time.time()
```

output:

```
Epoch 1/50
390/390 [==============================] - 95s 243ms/step - loss: 1.6571 - acc: 0.3909 - val_loss: 1.8933 - val_acc: 0.3944
Epoch 00001: val_acc improved from -inf to 0.39440, saving model to .mdl_wts.hdf5
Epoch 2/50
390/390 [==============================] - 91s 234ms/step - loss: 1.3039 - acc: 0.5268 - val_loss: 1.3534 - val_acc: 0.5242
Epoch 00002: val_acc improved from 0.39440 to 0.52420, saving model to .mdl_wts.hdf5
Epoch 3/50
390/390 [==============================] - 91s 233ms/step - loss: 1.1004 - acc: 0.6072 - val_loss: 1.2767 - val_acc: 0.5703
Epoch 00003: val_acc improved from 0.52420 to 0.57030, saving model to .mdl_wts.hdf5
...........
Epoch 00048: val_acc improved from 0.87320 to 0.87780, saving model to .mdl_wts.hdf5
Epoch 49/50
390/390 [==============================] - 91s 233ms/step - loss: 0.1061 - acc: 0.9641 - val_loss: 0.4929 - val_acc: 0.8805
Epoch 00049: val_acc improved from 0.87780 to 0.88050, saving model to .mdl_wts.hdf5
Epoch 50/50
390/390 [==============================] - 91s 233ms/step - loss: 0.0975 - acc: 0.9671 - val_loss: 0.4907 - val_acc: 0.8819
Epoch 00050: val_acc improved from 0.88050 to 0.88190, saving model to .mdl_wts.hdf5"
```