Project

**Class Assessment 1**

**COURSE – INT248 – Advanced Machine**

**Learning**

|  |  |  |  |
| --- | --- | --- | --- |
| r. No. | Registration No | Name of Student | Roll No |
| 1 | **11803997** | **GOVIND GANGA** | **RKM071A17** |
| 2 | **11805727** | **Satyam Chaudhary** | **RKM071A17** |

GitHub Link:- https://github.com/govind1042/Handwritten-character-Recognition

**Submitted To Md. Imran Hussain**

Lovely Professional University

Jalandhar, Punjab, India.



**Handwritten Character Recognition using deep learning**

Handwriting recognition (HWR), also known as Handwritten Text Recognition (HTR), is the ability of a computer to receive and interpret intelligible [handwritten](https://en.wikipedia.org/wiki/Handwriting) input from sources such as [paper](https://en.wikipedia.org/wiki/Paper) documents, [photographs](https://en.wikipedia.org/wiki/Photograph), [touch-screens](https://en.wikipedia.org/wiki/Touch-screen) and other devices. The image of the written text may be sensed "off line" from a piece of paper by optical scanning ([optical character recognition](https://en.wikipedia.org/wiki/Optical_character_recognition)) or [intelligent word recognition](https://en.wikipedia.org/wiki/Intelligent_word_recognition). Alternatively, the movements of the pen tip may be sensed "on line", for example by a pen-based computer screen surface, a generally easier task as there are more clues available. A handwriting recognition system handles formatting, performs correct [segmentation](https://en.wikipedia.org/wiki/Segment_(handwriting)) into characters, and finds the most plausible words.

**What is covered here:**

* Offline Handwritten Recognition
* Understand the detailed architecture of the Handwritten Recognition system.
* How to use the Data Augmentation technique to increase the accuracy and ability to work in real-time.

**Why Deep Learning?**

Diagram, schematic

Description automatically generated

Fig1: Deep Learning vs Machine Learning & Deep Learning performance with Other Algorithm.

Machine Learning needs *early Feature Extraction as features* and performed classification on it. But Deep Learning acts as a “black box” which do feature extraction and classification on its own.

In above figure shows that the main task is to classify the given image as face or non-face. In the case of machine learning, it needs a feature of images such as edges, colour, shape, etc., and performs classification on its own.

But deep learning extracts feature and perform classification on its own. Example of this given image to the Convolutional Neural Networks that every layer of CNN takes the feature and finally Fully Connected Layer perform classification

The main conclusion is Deep Learning self-extracts features with deep neural networks and classifies itself. Compare to traditional Algorithms its performance increase with the Amount of Data.

This article is all for building your own handwritten recognition system with Pytorch and Fastai. It covers detailed intuition about architecture and how I reach the solution and increase accuracy. In the end, I will provide the GitHub repo link where the pre-trained model is provided. Note that you can build a handwritten recognition system in any language where the architecture remains the same. In the end, you can build a handwritten recognition system in your own language provided the dataset to it. Let’s get started!

**Dataset**

**EMNIST(source: Kaggle)**

The EMNIST dataset is a set of handwritten character digits derived from the NIST Special Database 19 and converted to a 28x28 pixel image format and dataset structure that directly matches the MNIST dataset. Further information on the dataset contents and conversion process can be found in the paper available at <https://arxiv.org/abs/1702.05373v1>.

**Format**

There are six different splits provided in this dataset and each are provided in two formats:

* Binary (see emnist*source*files.zip)
* CSV (combined labels and images)

Each row is a separate image

785 columns

* First column = class\_label (see mappings.txt for class label definitions)
* Each column after represents one pixel value (784 total for a 28 x 28 image)

**ByClass** and **ByMerge** datasets

The full complement of the NIST Special Database 19 is available in the ByClass and ByMerge splits. These two datasets have the same image information but differ in the number of images in each class. Both datasets have an uneven number of images per class and there are more digits than letters. The number of letters roughly equate to the frequency of use in the English language.

* train: 697,932
* test: 116,323
* total: 814,255

classes: ByClass 62 (unbalanced) / ByMerge 47 (unbalanced)

**Balanced dataset**

The EMNIST Balanced dataset is meant to address the balance issues in the ByClass and ByMerge datasets. It is derived from the ByMerge dataset to reduce mis-classification errors due to capital and lower case letters and also has an equal number of samples per class. This dataset is meant to be the most applicable.

* train: 112,800
* test: 18,800
* total: 131,600

classes: 47 (balanced)

**Letters datasets**

The EMNIST Letters dataset merges a balanced set of the uppercase and lowercase letters into a single 26-class task.

* train: 88,800
* test: 14,800
* total: 103,600

classes: 37 (balanced)

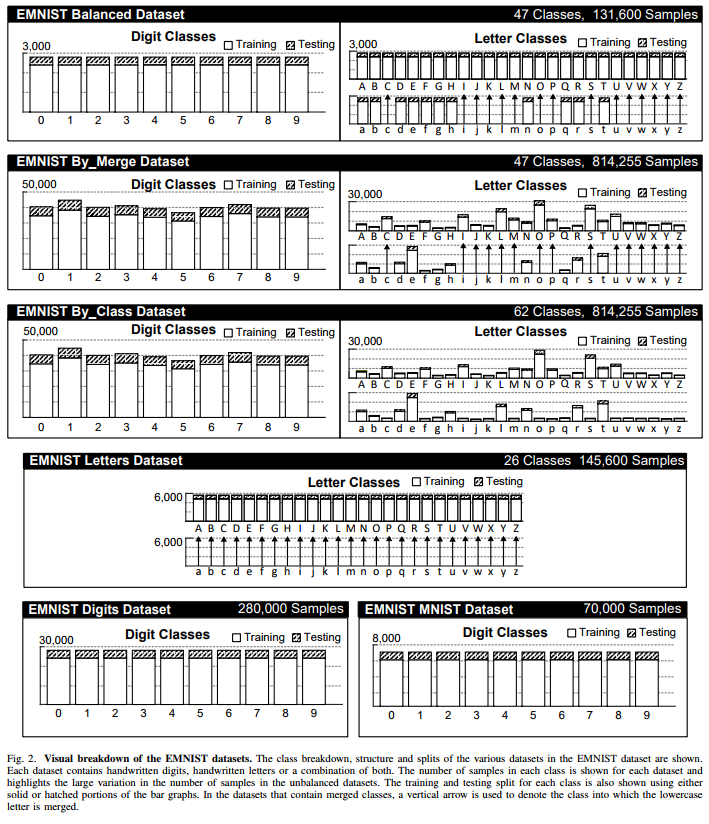
**Digits and MNIST datasets**

The EMNIST Digits and EMNIST MNIST dataset provide balanced handwritten digit datasets directly compatible with the original MNIST dataset.

* train: Digits 240,000 / MNIST 60,000
* test: Digits 40,000 / MNIST 10,000
* total: Digits 280,000 / MNIST 70,000

classes: 47 (balanced)

**Visual breakdown of EMNIST datasets**

Please refer to the EMNIST paper for details on the structure of the dataset <https://arxiv.org/abs/1702.05373v1>.  
  


Acknowledgement

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters.

Dataset retrieved from <https://www.nist.gov/itl/iad/image-group/emnist-dataset>

*Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik  
The MARCS Institute for Brain, Behaviour and Development  
Western Sydney University  
Penrith, Australia 2751*

**Proposed Architectures their results and experimental analysis**

**Convolutional Neural Network**

Convolutional Neural Network (CNNor ConvNet)is a class of **deep neural networks** which is mostly used to do image recognition, image classification, object detection, etc. The advancements in Computer Vision with Deep Learning has been constructed and perfected with time, primarily over one particular algorithm — a**Convolutional Neural Network.** Image classification is the task of taking an input image and outputting a class or a probability of classes that best describes the image. In CNN, we take an image as an input, assign importance to its various aspects/features in the image and be able to differentiate one from another. The pre-processing required in CNN is much lesser as compared to other classification algorithms.

Diagram

Description automatically generated

Different layers in a CNN

Importing the processed data frame using load\_pickle function

Table

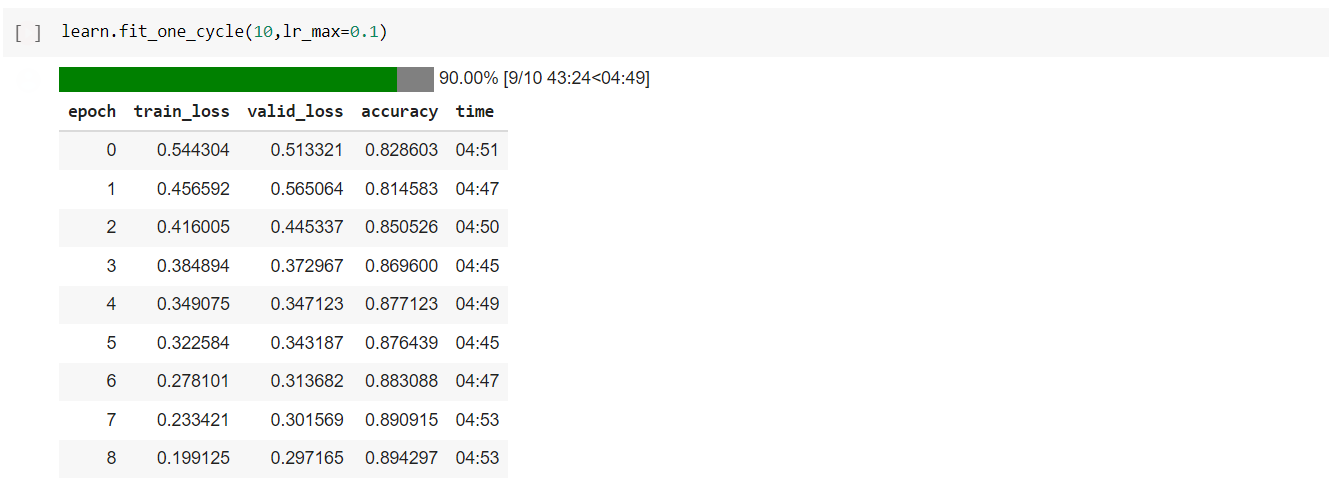
Description automatically generated with low confidence

Graphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated



| **epoch** | **train\_loss** | **valid\_loss** | **accuracy** | **time** |
| --- | --- | --- | --- | --- |
| 0 | 0.544304 | 0.513321 | 0.828603 | 04:51 |
| 1 | 0.456592 | 0.565064 | 0.814583 | 04:47 |
| 2 | 0.416005 | 0.445337 | 0.850526 | 04:50 |
| 3 | 0.384894 | 0.372967 | 0.869600 | 04:45 |
| 4 | 0.349075 | 0.347123 | 0.877123 | 04:49 |
| 5 | 0.322584 | 0.343187 | 0.876439 | 04:45 |
| 6 | 0.278101 | 0.313682 | 0.883088 | 04:47 |
| 7 | 0.233421 | 0.301569 | 0.890915 | 04:53 |
| 8 | 0.199125 | 0.297165 | 0.894297 | 04:53 |

**Residual networks (ResNets)**

When deeper networks are able to start converging, a *degradation*problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly.Using deeper networks is degrading the performance of the model. Microsoft Research paper tries to solve this problem using Deep Residual learning framework.

Solution: Residual Block / Identity block

The idea is that instead of letting layers learn the underlying mapping, let the network fit the residual mapping. So, instead of say H(x), initial mapping*,*let the network fit, F(x) := H(x)-x which gives H(x) := F(x) + x

The approach is to add *a shortcut or a skip connection* that allows information to flow, well just say, more easily from one layer to the next’s next layer, i.e., you bypass data along with normal CNN flow from one layer to the next layer after the immediate next.

**A Residual Block:**

Diagram

Description automatically generated

Residual learning: a building block

Two take away from residual block:

Adding additional / new layers would not hurt the model’s performance as regularisation will skip over them if those layers were not useful.

If the additional / new layers were useful, even with the presence of regularisation, the weights or kernels of the layers will be non-zero and model performance could increase slightly.

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application

Description automatically generatedGraphical user interface, application, table

Description automatically generated

| **epoch** | **train\_loss** | **valid\_loss** | **accuracy** | **time** |
| --- | --- | --- | --- | --- |
| 0 | 1.215322 | 0.679338 | 0.781983 | 04:39 |
| 1 | 0.455080 | 0.583010 | 0.800714 | 04:44 |
| 2 | 0.377500 | 0.543482 | 0.818648 | 04:42 |
| 3 | 0.330361 | 0.423815 | 0.854098 | 04:39 |
| 4 | 0.299964 | 0.436084 | 0.847373 | 04:42 |
| 5 | 0.261002 | 0.328505 | 0.881758 | 04:41 |
| 6 | 0.231155 | 0.303908 | 0.890307 | 04:39 |
| 7 | 0.193333 | 0.296462 | 0.895513 | 04:39 |
| 8 | 0.166900 | 0.289916 | 0.898818 | 04:36 |
| 9 | 0.155726 | 0.290979 | 0.899236 | 04:35 |

**Transfer learning**

Transfer learning, used in machine learning, is the reuse of a pre-trained model on a new problem. In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another. For example, in training a classifier to predict whether an image contains food, you could use the knowledge it gained during training to recognize drinks.

In transfer learning, the knowledge of an already trained [machine learning](https://builtin.com/data-science/introduction-to-machine-learning) model is applied to a different but related problem. For example, if you trained a simple classifier to predict whether an image contains a backpack, you could use the knowledge that the model gained during its training to recognize other objects like sunglasses.

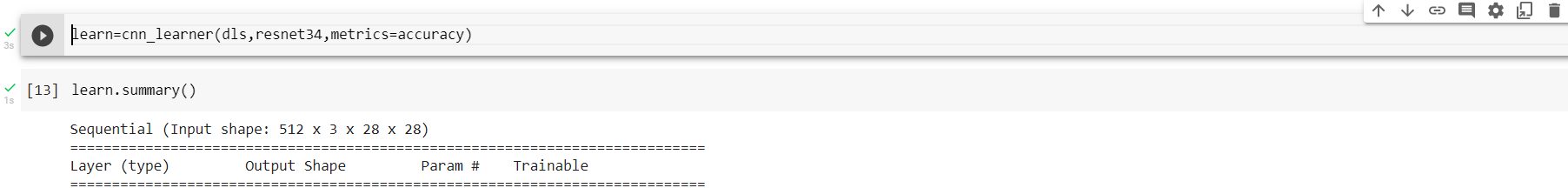
With transfer learning, we basically try to exploit what has been learned in one task to improve generalization in another. We transfer the weights that a network has learned at "task A" to a new "task B."

Table

Description automatically generated with low confidence

Graphical user interface, text, application, email, Teams

Description automatically generated



Sequential (Input shape: 512 x 3 x 28 x 28)

============================================================================

Layer (type) Output Shape Param # Trainable

============================================================================

512 x 64 x 14 x 14

Conv2d 9408 False

BatchNorm2d 128 True

ReLU

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

512 x 64 x 7 x 7

MaxPool2d

Conv2d 36864 False

BatchNorm2d 128 True

ReLU

Conv2d 36864 False

BatchNorm2d 128 True

Conv2d 36864 False

BatchNorm2d 128 True

ReLU

Conv2d 36864 False

BatchNorm2d 128 True

Conv2d 36864 False

BatchNorm2d 128 True

ReLU

Conv2d 36864 False

BatchNorm2d 128 True

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

512 x 128 x 4 x 4

Conv2d 73728 False

BatchNorm2d 256 True

ReLU

Conv2d 147456 False

BatchNorm2d 256 True

Conv2d 8192 False

BatchNorm2d 256 True

Conv2d 147456 False

BatchNorm2d 256 True

ReLU

Conv2d 147456 False

BatchNorm2d 256 True

Conv2d 147456 False

BatchNorm2d 256 True

ReLU

Conv2d 147456 False

BatchNorm2d 256 True

Conv2d 147456 False

BatchNorm2d 256 True

ReLU

Conv2d 147456 False

BatchNorm2d 256 True

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

512 x 256 x 2 x 2

Conv2d 294912 False

BatchNorm2d 512 True

ReLU

Conv2d 589824 False

BatchNorm2d 512 True

Conv2d 32768 False

BatchNorm2d 512 True

Conv2d 589824 False

BatchNorm2d 512 True

ReLU

Conv2d 589824 False

BatchNorm2d 512 True

Conv2d 589824 False

BatchNorm2d 512 True

ReLU

Conv2d 589824 False

BatchNorm2d 512 True

Conv2d 589824 False

BatchNorm2d 512 True

ReLU

Conv2d 589824 False

BatchNorm2d 512 True

Conv2d 589824 False

BatchNorm2d 512 True

ReLU

Conv2d 589824 False

BatchNorm2d 512 True

Conv2d 589824 False

BatchNorm2d 512 True

ReLU

Conv2d 589824 False

BatchNorm2d 512 True

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

512 x 512 x 1 x 1

Conv2d 1179648 False

BatchNorm2d 1024 True

ReLU

Conv2d 2359296 False

BatchNorm2d 1024 True

Conv2d 131072 False

BatchNorm2d 1024 True

Conv2d 2359296 False

BatchNorm2d 1024 True

ReLU

Conv2d 2359296 False

BatchNorm2d 1024 True

Conv2d 2359296 False

BatchNorm2d 1024 True

ReLU

Conv2d 2359296 False

BatchNorm2d 1024 True

AdaptiveAvgPool2d

AdaptiveMaxPool2d

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

512 x 1024

Flatten

BatchNorm1d 2048 True

Dropout

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

512 x 512

Linear 524288 True

ReLU

BatchNorm1d 1024 True

Dropout

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

512 x 47

Linear 24064 True

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Total params: 21,836,096

Total trainable params: 568,448

Total non-trainable params: 21,267,648

Optimizer used: <function Adam at 0x7f21401c1320>

Loss function: FlattenedLoss of CrossEntropyLoss()

Model frozen up to parameter group #2

Callbacks:

- TrainEvalCallback

- Recorder

- ProgressCallback

Graphical user interface, application

Description automatically generated

| **epoch** | **train\_loss** | **valid\_loss** | **accuracy** | **time** |
| --- | --- | --- | --- | --- |
| 0 | 2.124947 | 1.380753 | 0.606788 | 04:58 |

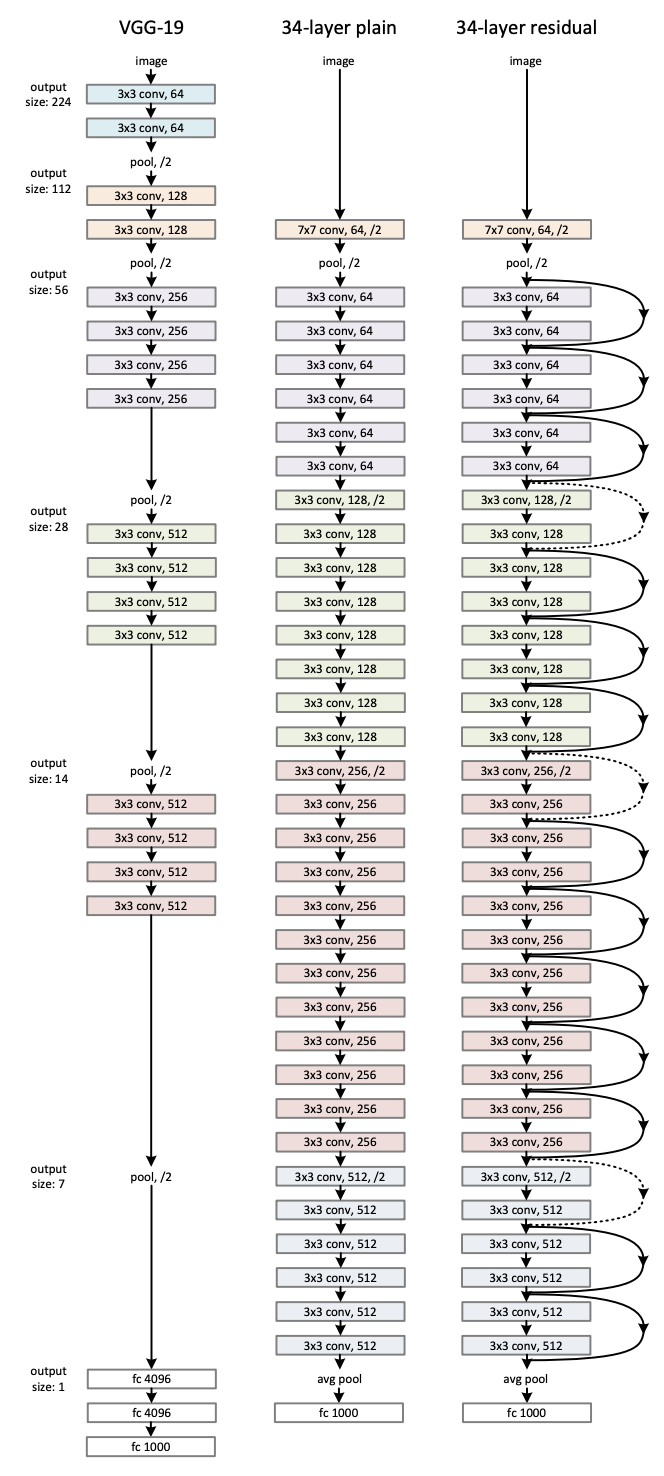
 60.00% [6/10 30:22<20:15]

| **epoch** | **train\_loss** | **valid\_loss** | **accuracy** | **time** |
| --- | --- | --- | --- | --- |
| 0 | 0.824589 | 0.616511 | 0.803979 | 05:03 |
| 1 | 0.511828 | 0.441725 | 0.853450 | 05:04 |
| 2 | 0.398020 | 0.400982 | 0.863610 | 05:06 |
| 3 | 0.328871 | 0.381027 | 0.870365 | 05:01 |
| 4 | 0.268165 | 0.377792 | 0.874089 | 05:01 |
| 5 | 0.219534 | 0.379722 | 0.877440 | 05:04 |

What is Resnet34?

Resnet34 is a 34-layer convolutional neural network that can be utilized as a state-of-the-art image classification model. This is a model that has been pre-trained on the ImageNet dataset--a dataset that has 100,000+ images across 200 different classes. However, it is different from traditional neural networks in the sense that it takes residuals from each layer and uses them in the subsequent connected layers (similar to residual neural networks used for text prediction).

Restnet34 Architecture

Below. on the right-hand side, is Resnet34's architecture where the 34 layers and the residuals from one layer to another are visualized.  


References:-

[www.Kaggle.com](http://www.Kaggle.com) (source of data)

<https://www.nist.gov/itl/iad/image-group/emnist-dataset> (dataset is derived from)

[www.fastai.com](http://www.fastai.com) (high level API library)

[www.pytorch.com](http://www.pytorch.com) (framework for deep learning)

[www.medium.com](http://www.medium.com) (documentation and writing support)