IMPLEMENTATION OF FACTORIZED CONVOLUTIONAL NEURAL NETWORKS AND ANALYSIS OF THE COMPUTATION WITH DIFFERENT MODELS

IMPLEMENTATION OF FACTORIZED CONVOLUTIONAL NEURAL NETWORKS AND ANALYSIS OF THE COMPUTATION WITH DIFFERENT MODELS

A thesis submitted to Indian Institute of Technology Kharagpur for B.Tech Project

By

Sayantan Kirtaniya (18EC33002)

Under the guidance of

Professor Swanand Ravindra Khare

(Mathematics and Computing Department)

Department of Electrical and Electronics Engineering

Indian Institute of Technology, Kharagpur

Department of Electrical and Electronics Engineering, Indian Institute of Technology Kharagpur, Kharagpur, India- 721302

CERTIFICATE

This is to certify that the work presented in this thesis is entitled Implementation of **IMPLEMENTATION OF FACTORIZED CONVOLUTIONAL NEURAL NETWORKS AND ANALYSIS OF THE COMPUTATION WITH DIFFERENT MODELS** has been carried out by Sayantan Kirtaniya under my supervision. This is his bonafide work and is worthy of consideration for a B.Tech term project - 2 of the Institute.

Prof.Swanand R Khare
Department of Mathematics and Computing
Indian Institute of Technology Kharagpur
India - 721302.

DECLARATION

I certify that:

1) the work described in this project is original and has been done by me with the support and guidance of my supervisor;

2) I have not submitted the work to any other institute for any other degree or diploma;

3) I have strictly adhered to ethical norms and guidelines while preparing the thesis;

4) I have strictly followed the guidelines presented by the Institute in doing the thesis;

5) Whenever I have used materials (data, models, figures, and text) from other sources, I have given due credit to authors and papers by citing them. Details of the citations are available in the references;

Sayantan Kirtaniya

ACKNOWLEDGEMENT

I am grateful to my supervisor ***Prof. Swanand R Khare*** for guiding me throughout this project. Without his valuable suggestions, I could not have presented this work. Special thanks to ***Tamara G. Kolda*** (Mathematical Consultant MathSci.ai) and ***Ankur Moitra*** (Department of Mathematics at MIT) for their wonderful work in this domain with their lecture videos and research papers, which helped me to understand the ideas and implementation of this area of research.

ABSTRACT

Computation is a big factor when we try to implement the convolutional neural networks so we need so factorization in the layers, CP decomposition can be a technique we can try to implement in the convolutional layers such that the idea of the models helps to get train faster, in this thesis, we are going to impediment wider networks to factorized layers such that the models become efficient and fast, also we are going to analyze the computation time and effect in the accuracy of the model to find out the best solution of our problem statement.

CONTENTS

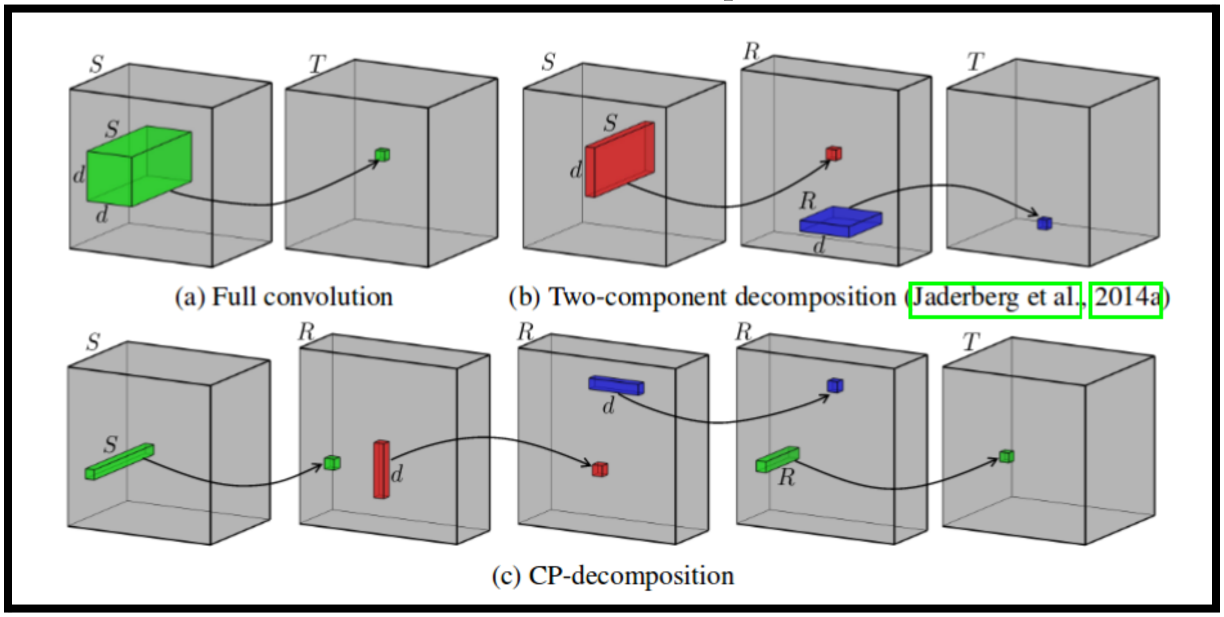
1. Introduction

The main goal of the thesis we are going to target is that we are implementing factorized neural networks in images such that the convolution computation will decrease.

Generally, we can use bottleneck layers to decrease the computation of the neural network. A bottleneck layer's main goal is to minimize the size of the input tensor in a convolutional layer with kernels larger than 1x1 by lowering the number of input channels, or the depth of the input tensor.[1]

The so-called wider convolutional layer is another simple technique to speed up convolutions.

Separable Convolutions in Depth, Instead of convolution simultaneously over all channels of an image, we execute a separate 2D convolution with a depth of channel multiplier on each channel. The intermediate channels in channels * channel multipliers are concatenated and mapped to out channels using a 1x1 convolution. As a result, there are much fewer parameters to train.



Source: V. Lebedev et al, Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition

Numerous publications, including those by V. Lebedev et al., demonstrate various tensor decomposition strategies that dramatically reduce the number of parameters and, as a result, the number of calculations necessary.[2]

TensorTrain decomposition and Tucker are two examples of such techniques. Tensorly is a fantastic package for PyTorch and NumPy that handles all of the low-level implementations for us. Although there is nothing comparable in TensorFlow, there is a TensorTrain aka TT scheme implementation.[4]

2. Conceptual areas and algorithms

Tensor decompositions are a logical technique to apply a low-rank approach to a multidimensional situation. Remember that the low-rank decomposition of a matrix A of dimension n m and rank R is:

$$A(i,j) = \sum_{r=1}^{R} A_1(i,r)A2(j,r)\ ,\ i = \overline{1,n}, j = \overline{1,m},$$

[6]This leads to the concept of variable separation. The canonical polyadic decomposition (CP-decomposition, also known as CANDECOMP/PARAFAC model) is the easiest technique to separate variables in cases of multiple dimensions (Kolda & Bader, 2009). For a d-dimensional array A, of size $n_1 \times \cdots \times n_d$ a CP-decomposition has the following form:

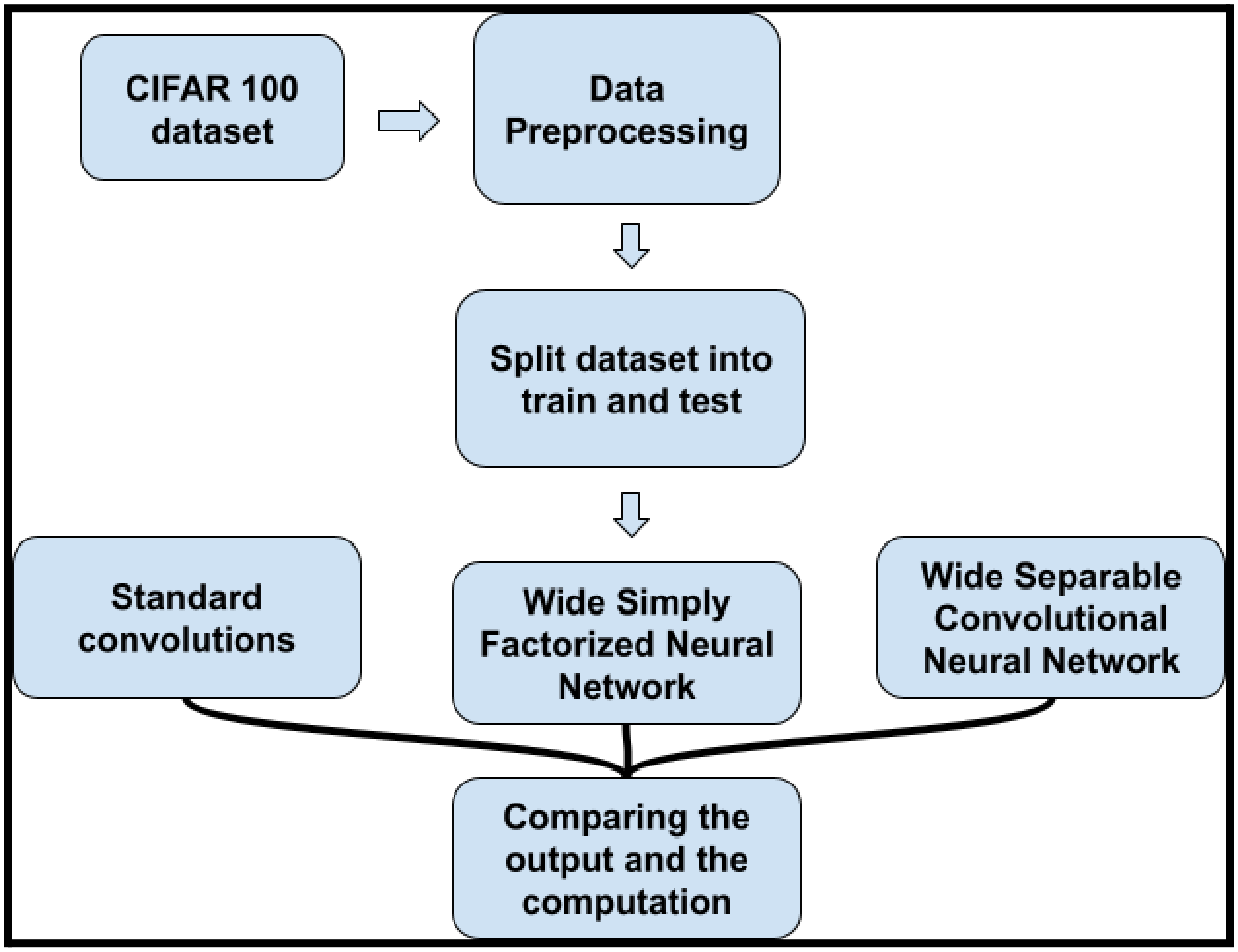
$$A(i_1,\ldots\ldots, i_d) = \sum_{r=1}^{R} A_1(i_1, r)\ldots\ldots A_d(i_d, r)$$

Canonical rank refers to the lowest possible R. The benefit of this decomposition is that we just need to save a little amount of data, and elements $(n_1 \times \cdots \times n_d)R$ instead of the whole tensor with $n_1 \ldots\ldots\ldots n_d$ elements.

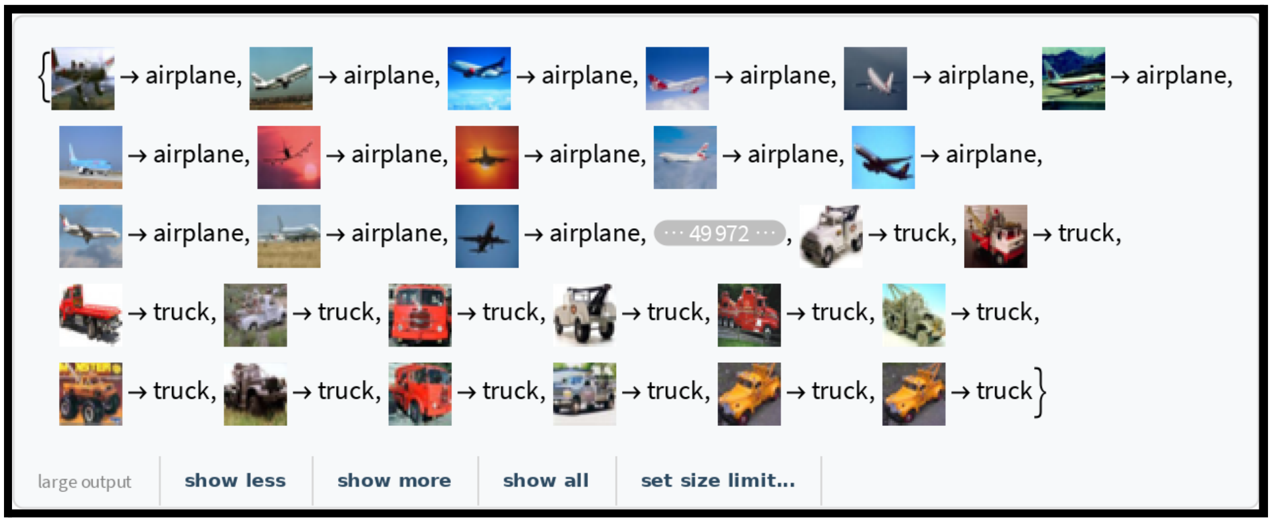
For **library** usage we are going to use the Tensorly - Pytorch and Numpy for the computational purposes.

3. Proposed idea

Here is the detailed workflow that we are going to implement in the next part:


CIFAR 100 dataset
Data Preprocessing
Split dataset into train and test
Standard convolutions
Wide Simply Factorized Neural Network
Wide Separable Convolutional Neural Network
Comparing the output and the computation

We are going to take the help with the CIFAR 100 dataset, which has 100 different classes. It has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses.[5]



{→ airplane, → airplane, → airplane, → airplane, → airplane, → airplane, → airplane, → airplane, → airplane, → airplane, → airplane, → airplane, → airplane, → airplane, ··· 49 972 ···, → truck, → truck, → truck, → truck, → truck, → truck, → truck, → truck, → truck, → truck, → truck, → truck, → truck, → truck}

And then we will use a simple convolutional model with standard layers, a wider model with simple factorized layers and wider separable convolutional layer network. And then we will analyze the accuracy and then we are going to understand the time of the computational difference with networks.

We will train each of the models for 5 epochs to understand the computational differences as well as the accuracy of the different models.

4. Implementation of the models

The models we are going to implement are a simple convolutional model with standard layers, a wider model with simple factorized layers and a wider separable convolutional layer network.[7]
Let's look at the models architecture which we have implemented is given here,
The **simple convolutional model** is :

```
Layer (type)                 Output Shape              Param #
=================================================================
==
input_1 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 96)        2688
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 96)        384
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 96)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 96)        83040
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 96)        384
_________________________________________________________________
activation_2 (Activation)    (None, 32, 32, 96)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 96)        83040
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 96)        384
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 96)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 192)       166080
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 192)       768
_________________________________________________________________
activation_4 (Activation)    (None, 16, 16, 192)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 192)       331968
_________________________________________________________________
```

```
batch_normalization_5 (Batch (None, 16, 16, 192)      768
_________________________________________________________________
activation_5 (Activation)    (None, 16, 16, 192)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 192)        331968
_________________________________________________________________
batch_normalization_6 (Batch (None, 8, 8, 192)        768
_________________________________________________________________
activation_6 (Activation)    (None, 8, 8, 192)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 192)        331968
_________________________________________________________________
batch_normalization_7 (Batch (None, 8, 8, 192)        768
_________________________________________________________________
activation_7 (Activation)    (None, 8, 8, 192)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 192)        37056
_________________________________________________________________
batch_normalization_8 (Batch (None, 8, 8, 192)        768
_________________________________________________________________
activation_8 (Activation)    (None, 8, 8, 192)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 8, 8, 100)        19300
_________________________________________________________________
batch_normalization_9 (Batch (None, 8, 8, 100)        400
_________________________________________________________________
activation_9 (Activation)    (None, 8, 8, 100)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 100)              0
_________________________________________________________________
activation_10 (Activation)   (None, 100)              0
=================================================================
==
Total params: 1,392,500
Trainable params: 1,389,804
Non-trainable params: 2,696
```

The model which is: **a wider model with simple factorized layers**

____

```
Layer (type)              Output Shape           Param #
=================================================================
==
input_5 (InputLayer)        (None, 32, 32, 3)        0
_________________________________________________________________
conv2d_91 (Conv2D)          (None, 32, 32, 192)      1920
_________________________________________________________________
conv2d_92 (Conv2D)          (None, 32, 32, 192)      110784
_________________________________________________________________
batch_normalization_70 (Batc (None, 32, 32, 192)      768
_________________________________________________________________
activation_79 (Activation)   (None, 32, 32, 192)      0
_________________________________________________________________
conv2d_93 (Conv2D)          (None, 16, 16, 96)       165984
_________________________________________________________________
batch_normalization_71 (Batc (None, 16, 16, 96)       384
_________________________________________________________________
activation_80 (Activation)   (None, 16, 16, 96)       0
_________________________________________________________________
conv2d_95 (Conv2D)          (None, 16, 16, 384)      110976
_________________________________________________________________
conv2d_96 (Conv2D)          (None, 16, 16, 384)      442752
_________________________________________________________________
batch_normalization_72 (Batc (None, 16, 16, 384)      1536
_________________________________________________________________
activation_81 (Activation)   (None, 16, 16, 384)      0
_________________________________________________________________
conv2d_97 (Conv2D)          (None, 8, 8, 128)        442496
_________________________________________________________________
batch_normalization_73 (Batc (None, 8, 8, 128)        512
_________________________________________________________________
activation_82 (Activation)   (None, 8, 8, 128)        0
_________________________________________________________________
conv2d_98 (Conv2D)          (None, 8, 8, 256)        295168
_________________________________________________________________
batch_normalization_74 (Batc (None, 8, 8, 256)        1024
_________________________________________________________________
activation_83 (Activation)   (None, 8, 8, 256)        0
```

```
_________________________________________________________________
conv2d_99 (Conv2D)         (None, 8, 8, 100)        25700
_________________________________________________________________
batch_normalization_75 (Batc (None, 8, 8, 100)        400
_________________________________________________________________
activation_84 (Activation)   (None, 8, 8, 100)        0
_________________________________________________________________
global_average_pooling2d_10  (None, 100)            0
_________________________________________________________________
activation_85 (Activation)   (None, 100)            0
=================================================================
==
Total params: 1,600,404
Trainable params: 1,598,092
Non-trainable params: 2,312
```

The architecture of **separable convolutional layer network** is :

```
_________________________________________________________________
Layer (type)              Output Shape            Param #
=================================================================
==
input_4 (InputLayer)        (None, 32, 32, 3)        0
_________________________________________________________________
separable_conv2d_23 (Separab (None, 32, 32, 192)       795
_________________________________________________________________
batch_normalization_55 (Batc (None, 32, 32, 192)       768
_________________________________________________________________
activation_62 (Activation)   (None, 32, 32, 192)       0
_________________________________________________________________
conv2d_68 (Conv2D)         (None, 16, 16, 96)       165984
_________________________________________________________________
batch_normalization_56 (Batc (None, 16, 16, 96)       384
_________________________________________________________________
activation_63 (Activation)   (None, 16, 16, 96)       0
_________________________________________________________________
separable_conv2d_24 (Separab (None, 16, 16, 384)       38112
_________________________________________________________________
batch_normalization_57 (Batc (None, 16, 16, 384)       1536
_________________________________________________________________
activation_64 (Activation)   (None, 16, 16, 384)      0
_________________________________________________________________
conv2d_70 (Conv2D)           (None, 8, 8, 128)        442496
_________________________________________________________________
batch_normalization_58 (Batc (None, 8, 8, 128)        512
_________________________________________________________________
activation_65 (Activation)   (None, 8, 8, 128)        0
_________________________________________________________________
conv2d_71 (Conv2D)           (None, 8, 8, 256)        295168
_________________________________________________________________
batch_normalization_59 (Batc (None, 8, 8, 256)        1024
_________________________________________________________________
activation_66 (Activation)   (None, 8, 8, 256)        0
_________________________________________________________________
conv2d_72 (Conv2D)           (None, 8, 8, 100)        25700
_________________________________________________________________
batch_normalization_60 (Batc (None, 8, 8, 100)        400
_________________________________________________________________
activation_67 (Activation)   (None, 8, 8, 100)        0
_________________________________________________________________
global_average_pooling2d_8 ( (None, 100)              0
_________________________________________________________________
activation_68 (Activation)   (None, 100)              0
=================================================================
Total params: 972,879
Trainable params: 970,567
Non-trainable params: 2,312
```

Lets us look at the results of each of the models and observe the output as well as the discussion of the results.

5. Results

For the model 1 : **simple convolutional model** , the results are shown below,

```
Epoch 1/5
1562/1562 [==============================] - 36s 23ms/step - loss:
3.9801 - categorical_accuracy: 0.1236 - top_k_categorical_accuracy: 0.3421
Epoch 2/5
1562/1562 [==============================] - 32s 20ms/step - loss:
3.5620 - categorical_accuracy: 0.2003 - top_k_categorical_accuracy: 0.4800
Epoch 3/5
1562/1562 [==============================] - 33s 21ms/step - loss:
3.3145 - categorical_accuracy: 0.2480 - top_k_categorical_accuracy: 0.5488
Epoch 4/5
1562/1562 [==============================] - 32s 21ms/step - loss:
3.1350 - categorical_accuracy: 0.2808 - top_k_categorical_accuracy: 0.5931
Epoch 5/5
1562/1562 [==============================] - 32s 20ms/step - loss:
2.9940 - categorical_accuracy: 0.3082 - top_k_categorical_accuracy: 0.6263
Accuracy: 0.31860977564102566, Top5 Accuracy 0.6302083333333334 and Loss
2.879887064297994.
```

For the model 2 : **a wider model with simple factorized layers** , the results are shown below,[8]

```
Epoch 1/5
1562/1562 [==============================] - 52s 33ms/step - loss:
3.9616 - categorical_accuracy: 0.1298 - top_k_categorical_accuracy: 0.3542
Epoch 2/5
1562/1562 [==============================] - 49s 31ms/step - loss:
3.5735 - categorical_accuracy: 0.2049 - top_k_categorical_accuracy: 0.4832
Epoch 3/5
1562/1562 [==============================] - 49s 31ms/step - loss:
3.3399 - categorical_accuracy: 0.2505 - top_k_categorical_accuracy: 0.5497
Epoch 4/5
1562/1562 [==============================] - 49s 31ms/step - loss:
```

```
3.1804 - categorical_accuracy: 0.2816 - top_k_categorical_accuracy: 0.5894
Epoch 5/5
1562/1562 [==============================] - 49s 31ms/step - loss:
3.0554 - categorical_accuracy: 0.3078 - top_k_categorical_accuracy: 0.6207
Accuracy: 0.29253611556982345, Top5 Accuracy 0.5900882825040128 and
Loss 3.0202260208742193.
```

For the model 3 : **separable convolutional layer network**, the results are shown below,

```
Epoch 1/5 1562/1562 [==============================] - 30s 19ms/step -
loss: 4.0293 - categorical_accuracy: 0.1147 - top_k_categorical_accuracy: 0.3230
Epoch 2/5 1562/1562 [==============================] - 28s 18ms/step -
loss: 3.7112 - categorical_accuracy: 0.1781 - top_k_categorical_accuracy: 0.4411
Epoch 3/5 1562/1562 [==============================] - 28s 18ms/step -
loss: 3.5124 - categorical_accuracy: 0.2183 - top_k_categorical_accuracy: 0.5028
Epoch 4/5 1562/1562 [==============================] - 29s 18ms/step -
loss: 3.3756 - categorical_accuracy: 0.2446 - top_k_categorical_accuracy: 0.5386
Epoch 5/5 1562/1562 [==============================] - 28s 18ms/step -
loss: 3.2749 - categorical_accuracy: 0.2637 - top_k_categorical_accuracy: 0.5626
Accuracy: 0.27879213483146065, Top5 Accuracy 0.5764446227929374 and
Loss 3.1429175584312428.
```

Let's look at the three results side by side,

Models	AVG. Time of computation	Accuracy	TOP 5 Accuracy	Loss
simple convolutional model	33s 21ms/step	0.318	0.630	2.879
A wider model with simple factorized layers	49s 31ms/step	0.292	0.590	3.020

separable convolutional layer network	28s 18ms/step	0.278	0.576	3.142

- By analyzing the model results we can say that the computational time has decreased by a good amount but the accuracy have decreed by a vault of 0.14, which may be affect the model robustness.
- The loss and Top 5 accuray for the test set are also decreased for the wider simple and wider factorized models, so we have partially made the model better based on its computation time.
- There is a large possibility that the model accuracy will get better because of the increase in the higher number of epochs.

6. Future work

- In the future we can try to implement more models such that we can get better outputs and the models become more robust.
- We use these models in the different medical datasets as there is a very less number of factorized layers implemented in the areas of neural networks.
- We can make ensemble models from the outcomes of the different models and make the output with better accuracy.

7. Reference

Full citations information and papers information:

[1] F. Chollet, Xception: Deep Learning with Depthwise Separable Convolutions
[2] V. Lebedev et al, Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition
[3] C. Szegedy et al, Rethinking the Inception Architecture for Computer Vision
[4] S. Zagoruyko and N. Komodakis, Wide Residual Networks
[5] Speeding-up convolutional neural networks using fine-tuned cp-decomposition
[6] Factorized Convolutional Neural Networks
[7] Factorized Convolution Kernels in Image Processing
[8] TENSORFLOW TOOLS