Machine Learning Digital Assignment
Nikhil Saraogi [21MCA1080]

Part 1: EfficientNet B5

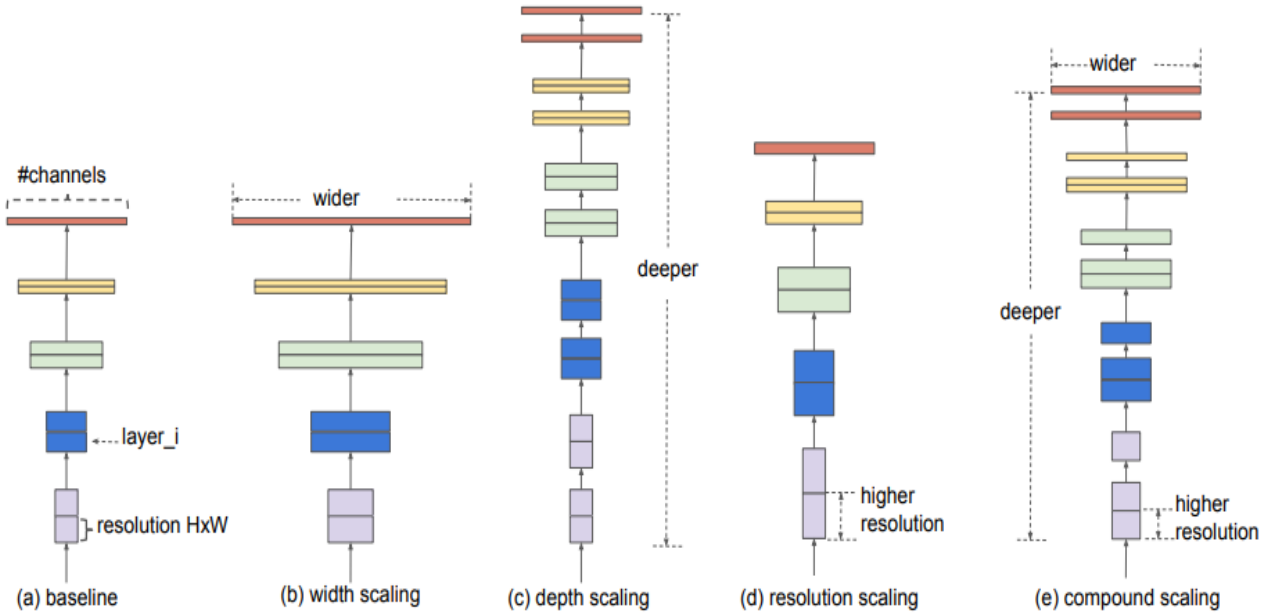
Understand the model and provide a detailed explanation of the Model. Analyse and compare it with other models like AlexNet, GoogLeNet, ResNet .

EfficientNet

EfficientNet model was proposed by Mingxing Tan and Quoc V. Le of Google Research, Brain team in their research paper 'EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks'. This paper was presented in the International Conference on Machine Learning, 2019. These researchers studied the model scaling and identified that carefully balancing the depth, width, and resolution of the network can lead to better performance.
Based on this observation, they proposed a new scaling method that uniformly scales all dimensions of depth, width and resolution of the network. They used the neural architecture search to design a new baseline network and scaled it up to obtain a family of deep learning models, called EfficientNets, which achieve much better accuracy and efficiency as compared to the previous Convolutional Neural Networks.

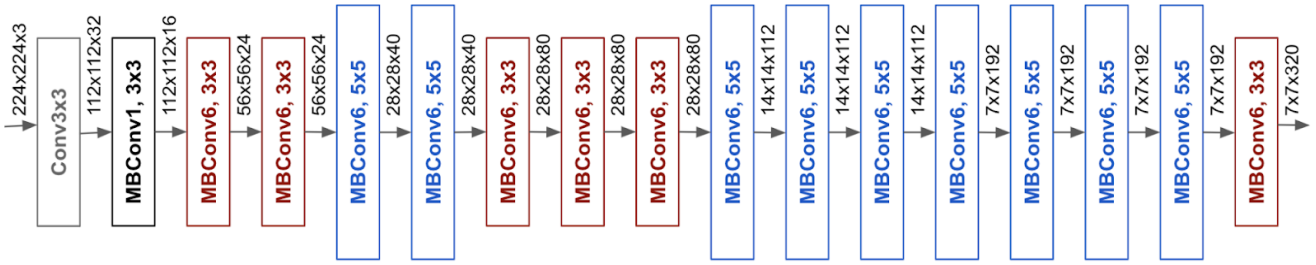
Scaling

The researchers used the compound scaling method to scale the dimensions of the network. The applied grid search strategy to find the relationship between the different scaling dimensions of the baseline network under a fixed resource constraint. Using this strategy, the could find the appropriate scaling coefficients for each of the dimensions to be scaled-up. Using these coefficients, the baseline network was scaled by the desired size. The researchers claimed in their work that this compound scaling method improved the model's accuracy and efficiency.

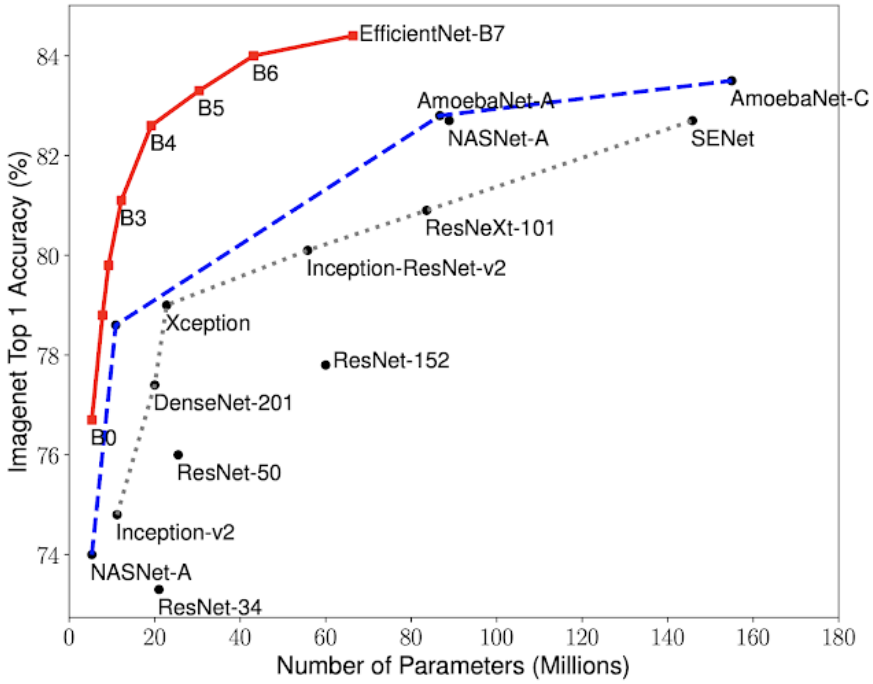


EfficientNet Architecture

The researchers first designed a baseline network by performing the neural architecture search, a technique for automating the design of neural networks. It optimizes both the accuracy and efficiency as measured on the floating-point operations per second (FLOPS) basis. This developed architecture uses the mobile inverted bottleneck convolution (MBConv). The researchers then scaled up this baseline network to obtain a family of deep learning models, called EfficientNets. Its architecture is given in the below diagram.

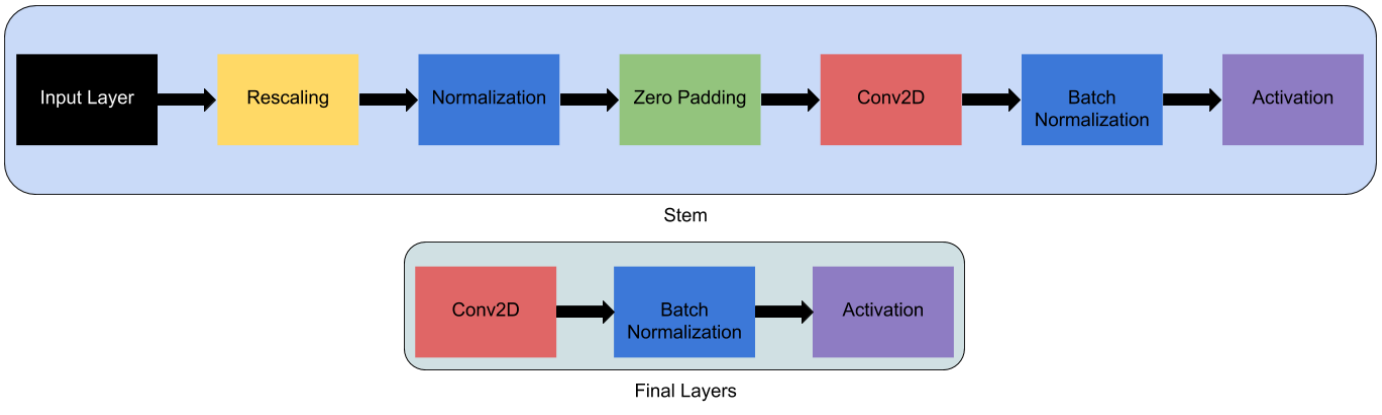


They also presented a comparison of EfficientNet's performance with other powerful transfer learning models when worked on ImageNet dataset. It has been shown that the latest version of EfficientNet that is EfficientNet-B7 has the highest accuracy among all with less number of parameters.



Common Things

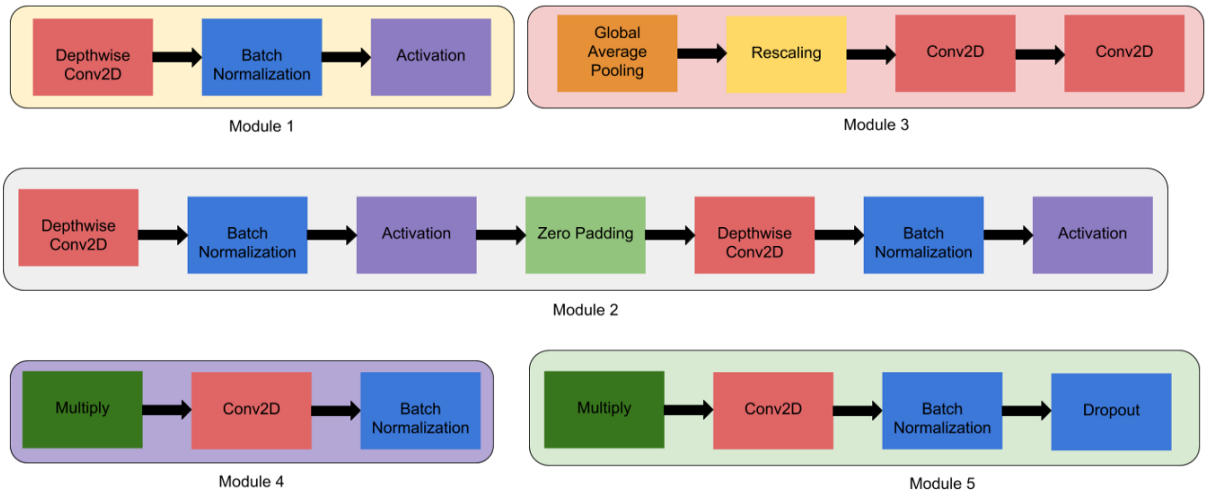
The first thing is any network is its stem after which all the experimenting with the architecture starts which is common in all the eight models and the final layers.



After this each of them contains 7 blocks. These blocks further have a varying number of sub-blocks whose number is increased as we move from EfficientNetB0 to EfficientNetB7. To have a look at the layers of the models in Colab write this code:

```
!pip install tf-nightly-gpu
import tensorflow as tf
IMG_SHAPE = (224, 224, 3)
model0 = tf.keras.applications.EfficientNetB0(input_shape=IMG_SHAPE, include_top=False, weights="imagenet")
tf.keras.utils.plot_model(model0) # to draw and visualize
model0.summary() # to see the list of layers and parameters
```

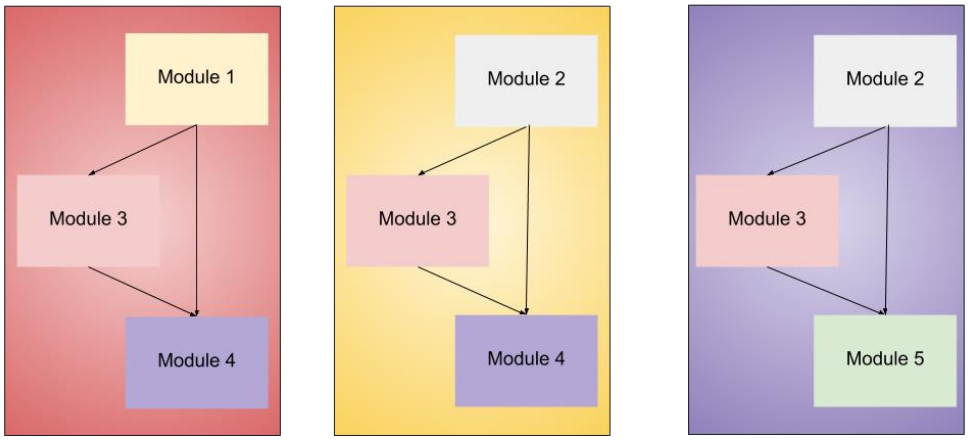
If you count the total number of layers in EfficientNet-B0 the total is 237 and in EfficientNet-B7 the total comes out to 813!! But don't worry all these layers can be made from 5 modules shown below and the stem above.



5 modules we will use to make the architecture.

- **Module 1** — This is used as a starting point for the sub-blocks.
- **Module 2** — This is used as a starting point for the first sub-block of all the 7 main blocks except the 1st one.
- **Module 3** — This is connected as a skip connection to all the sub-blocks.
- **Module 4** — This is used for combining the skip connection in the first sub-blocks.
- **Module 5** — Each sub-block is connected to its previous sub-block in a skip connection and they are combined using this module.

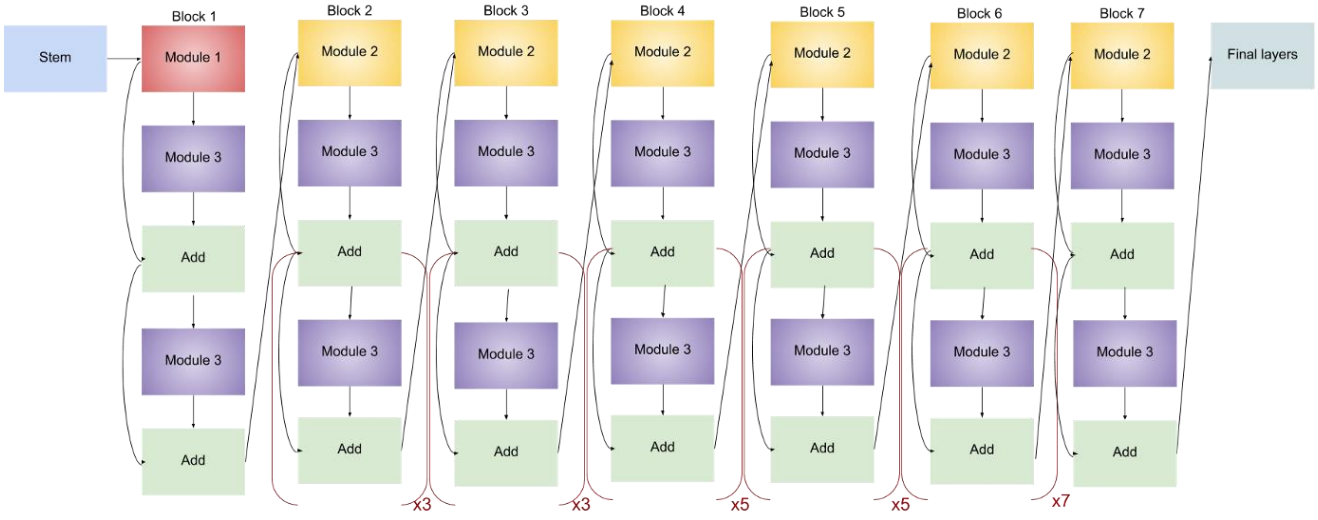
These modules are further combined to form sub-blocks which will be used in a certain way in the blocks.



Sub-block 1 Sub-block 2 Sub-block 3

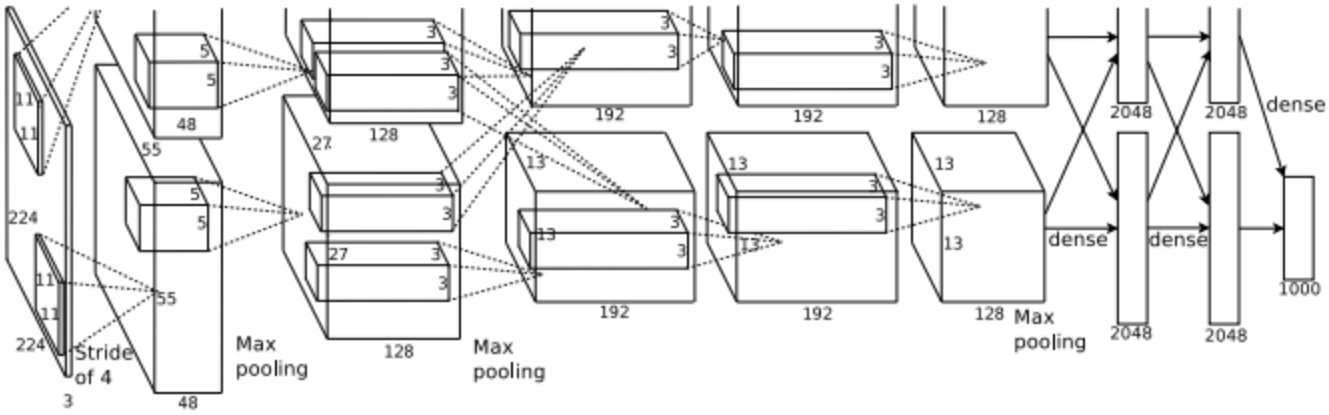
- **Sub-block 1** — This is used only used as the first sub-block in the first block.
- **Sub-block 2** — This is used as the first sub-block in all the other blocks.
- **Sub-block 3** — This is used for any sub-block except the first one in all the blocks.

Architecture of EfficientNet-B5



AlexNet (2012)

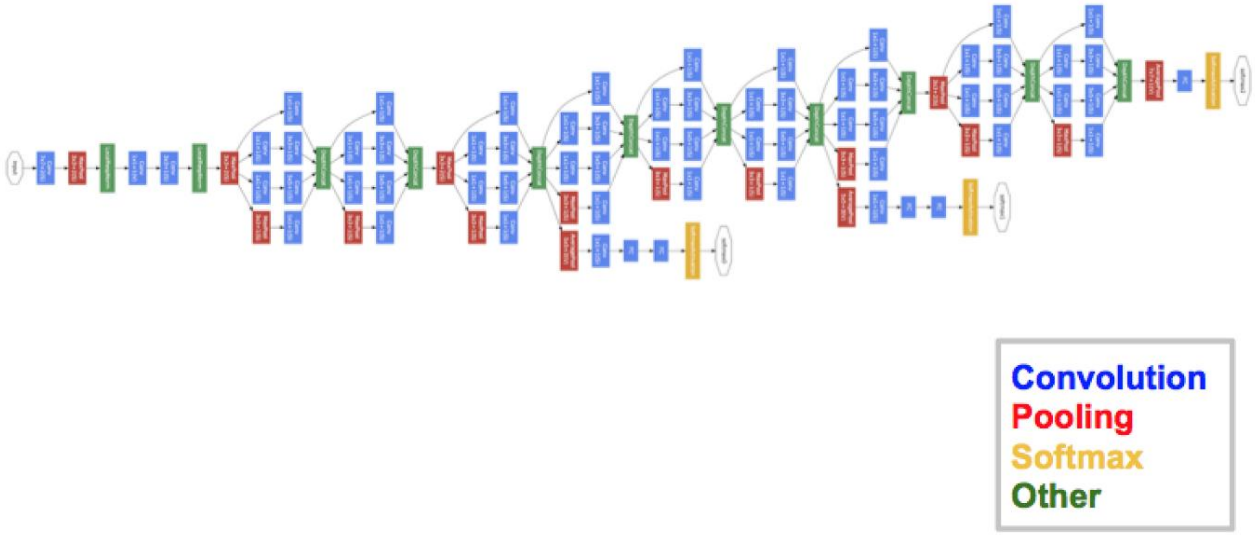
In 2012, AlexNet significantly outperformed all the prior competitors and won the challenge by reducing the top-5 error from 26% to 15.3%. The second place top-5 error rate, which was not a CNN variation, was around 26.2%.



The network had a very similar architecture as LeNet by Yann LeCun et al but was deeper, with more filters per layer, and with stacked convolutional layers. It consisted 11x11, 5x5,3x3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum. It attached ReLU activations after every convolutional and fully-connected layer. AlexNet was trained for 6 days simultaneously on two Nvidia Geforce GTX 580 GPUs which is the reason for why their network is split into two pipelines. AlexNet was designed by the SuperVision group, consisting of Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever.

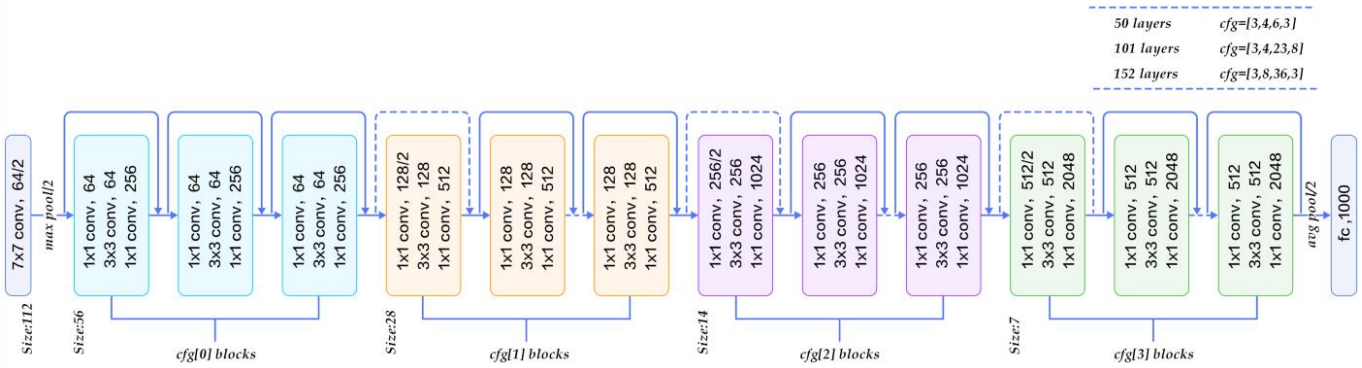
GoogLeNet/Inception(2014)

The winner of the ILSVRC 2014 competition was GoogLeNet(a.k.a. Inception V1) from Google. It achieved a top-5 error rate of 6.67%! This was very close to human level performance which the organisers of the challenge were now forced to evaluate. As it turns out, this was actually rather hard to do and required some human training in order to beat GoogLeNets accuracy. After a few days of training, the human expert (Andrej Karpathy) was able to achieve a top-5 error rate of 5.1%(single model) and 3.6%(ensemble). The network used a CNN inspired by LeNet but implemented a novel element which is dubbed an inception module. It used batch normalization, image distortions and RMSprop. This module is based on several very small convolutions in order to drastically reduce the number of parameters. Their architecture consisted of a 22 layer deep CNN but reduced the number of parameters from 60 million (AlexNet) to 4 million.

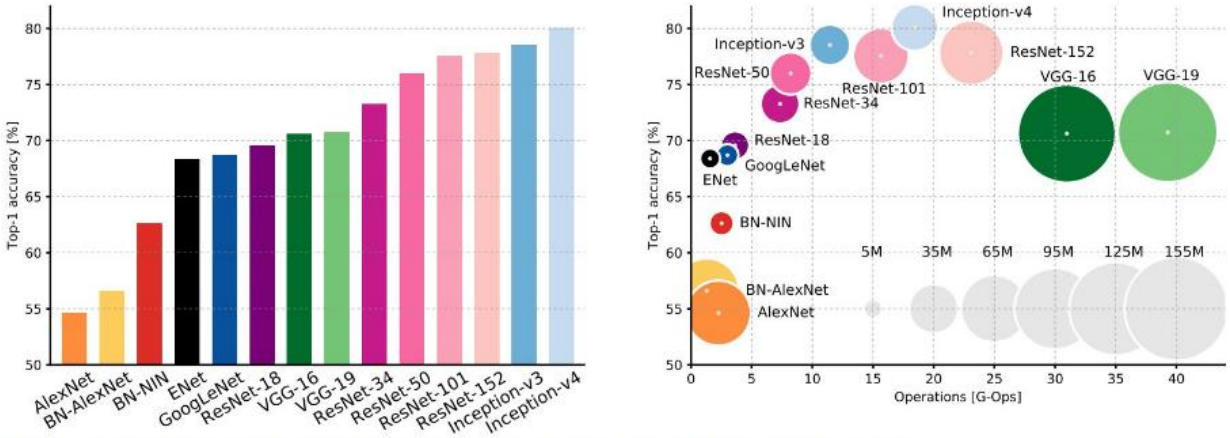


ResNet(2015)

At last, at the ILSVRC 2015, the so-called Residual Neural Network (ResNet) by Kaiming He et al introduced anovel architecture with “skip connections” and features heavy batch normalization. Such skip connections are also known as gated units or gated recurrent units and have a strong similarity to recent successful elements applied in RNNs. Thanks to this technique they were able to train a NN with 152 layers while still having lower complexity than VGGNet. It achieves a top-5 error rate of 3.57% which beats human-level performance on this dataset.



AlexNet has parallel two CNN line trained on two GPUs with cross-connections, GoogleNet has inception modules ,ResNet has residual connections.



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Several comparisons can be drawn:

- AlexNet and ResNet-152, both have about 60M parameters but there is about a 10% difference in their top-5 accuracy. But training a ResNet-152 requires a lot of computations (about 10 times more than that of AlexNet) which means more training time and energy required.
- VGGNet not only has a higher number of parameters and FLOP as compared to ResNet-152 but also has a decreased accuracy. It takes more time to train a VGGNet with reduced accuracy.
- Training an AlexNet takes about the same time as training Inception. The memory requirements are 10 times less with improved accuracy (about 9%)

Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

Part 2: Implementation

In this experiment, I will implement the EfficientNet on multi-class image classification on the CIFAR-10 dataset. To implement it as a transfer learning model, I have used the EfficientNet-B5 version as B6 and B7 does not support the ImageNet weights when using Keras. The CIFAR-10 dataset is a publically available image data set provided by the Canadian Institute for Advanced Research (CIFAR). It consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 50000 training images and 10000 test images in this dataset.

In the first step, we will download the data set and import the required libraries-

```
#Keras library for CIFAR dataset
from keras.datasets import cifar10

#Downloading the CIFAR dataset
(x_train,y_train),(x_test,y_test)=cifar10.load_data()
#importing other required libraries
import numpy as np
import pandas as pd
from sklearn.utils.multiclass import unique_labels
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
import itertools
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from keras import Sequential
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import SGD,Adam
from keras.callbacks import ReduceLROnPlateau
from keras.layers import Flatten,Dense,BatchNormalization,Activation,Dropout
from tensorflow.keras.utils import to_categorical
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
```

After importing the libraries, we will download the dataset and preprocess

```
[3] #Train-validation-test split
    from sklearn.model_selection import train_test_split
    x_train,x_val,y_train,y_val=train_test_split(x_train,y_train,test_size=.3)

    #Dimension of the CIFAR10 dataset
    print((x_train.shape,y_train.shape))
    print((x_val.shape,y_val.shape))
    print((x_test.shape,y_test.shape))
```

```
((24500, 32, 32, 3), (24500, 1))
((10500, 32, 32, 3), (10500, 1))
((10000, 32, 32, 3), (10000, 1))
```

```
[4] #Onehot Encoding the labels.
    from sklearn.utils.multiclass import unique_labels
    from tensorflow.keras.utils import to_categorical

    #Since we have 10 classes we should expect the shape[1] of y_train,y_val and y_test to change from 1 to 10
    y_train=to_categorical(y_train)
    y_val=to_categorical(y_val)
    y_test=to_categorical(y_test)

    #Verifying the dimension after one hot encoding
    print((x_train.shape,y_train.shape))
    print((x_val.shape,y_val.shape))
    print((x_test.shape,y_test.shape))
```

I will use the learning rate annealer in this experiment. The learning rate annealer decreases the learning rate after a certain number of epochs if the error rate does not change. Here, through this technique, we will monitor the validation accuracy and if it seems to be a plateau in 3 epochs, it will reduce the learning rate by 0.01.

```
[6] #Learning Rate Annealer
    from keras.callbacks import ReduceLROnPlateau
    lrr= ReduceLROnPlateau(   monitor='val_acc',   factor=.01,   patience=3,  min_lr=1e-5)
```

In the next step, need to install the efficient net and import it using the following way.

```
) !pip install keras_efficientnets
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting keras_efficientnets
  Downloading keras_efficientnets-0.1.7-py2.py3-none-any.whl (15 kB)
Requirement already satisfied: scikit-learn>=0.21.2 in /usr/local/lib/python3.7/dist-packages (from keras_efficientnets) (1.0.2)
Requirement already satisfied: keras>=2.2.4 in /usr/local/lib/python3.7/dist-packages (from keras_efficientnets) (2.8.0)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from keras_efficientnets) (1.7.3)
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.21.2->keras_efficientnets) (1.21.6)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.21.2->keras_efficientnets) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.21.2->keras_efficientnets) (3.1.0)
Installing collected packages: keras-efficientnets
Successfully installed keras-efficientnets-0.1.7
```

```
] from tensorflow.keras.applications.efficientnet import EfficientNetB5
```

Here, I will define the EfficientNet-B5 using the following code snippet

```
#Defining the model
base_model = EfficientNetB5(include_top=False, weights="imagenet", input_shape=(32,32,3),classes=y_train.shape[1])

#Adding the final layers to the above base models where the actual classification is done in the dense layers

model= Sequential()
model.add(base_model)
model.add(Flatten())

#Model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb2 (Functional)  (None, 1, 1, 1408)       7768569

 flatten_1 (Flatten)         (None, 1408)              0

=================================================================
Total params: 7,768,569
Trainable params: 7,700,994
Non-trainable params: 67,575
```

Checking the final model summary

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb2 (Functional)  (None, 1, 1, 1408)       7768569

 flatten_1 (Flatten)         (None, 1408)              0

 dense_5 (Dense)             (None, 1024)              1442816

 dense_6 (Dense)             (None, 512)               524800

 dense_7 (Dense)             (None, 256)               131328

 dense_8 (Dense)             (None, 128)               32896

 dense_9 (Dense)             (None, 10)                1290

=================================================================
Total params: 9,901,699
Trainable params: 9,834,124
Non-trainable params: 67,575
_________________________________________________________________
```

To train the model, I will define below the number of epochs, the number of batches and the learning rate.

```
batch_size= 100
epochs=4
learn_rate=.001
```

I will define the Stochastic Gradient Descent as the optimizer.

```
sgd=SGD(lr=learn_rate,momentum=.9,nesterov=False)
```

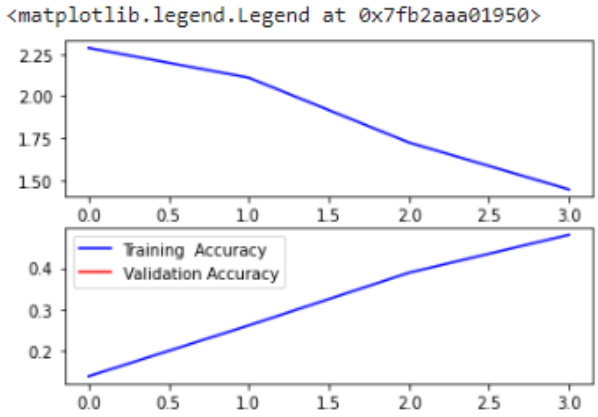
I will compile and train the model

```
#Compiling the model
model.compile(optimizer=sgd,loss='categorical_crossentropy',metrics=['accuracy'])

#Training the model
model.fit_generator(train_generator.flow(x_train, y_train, batch_size = batch_size), epochs = epochs, steps_per_
```

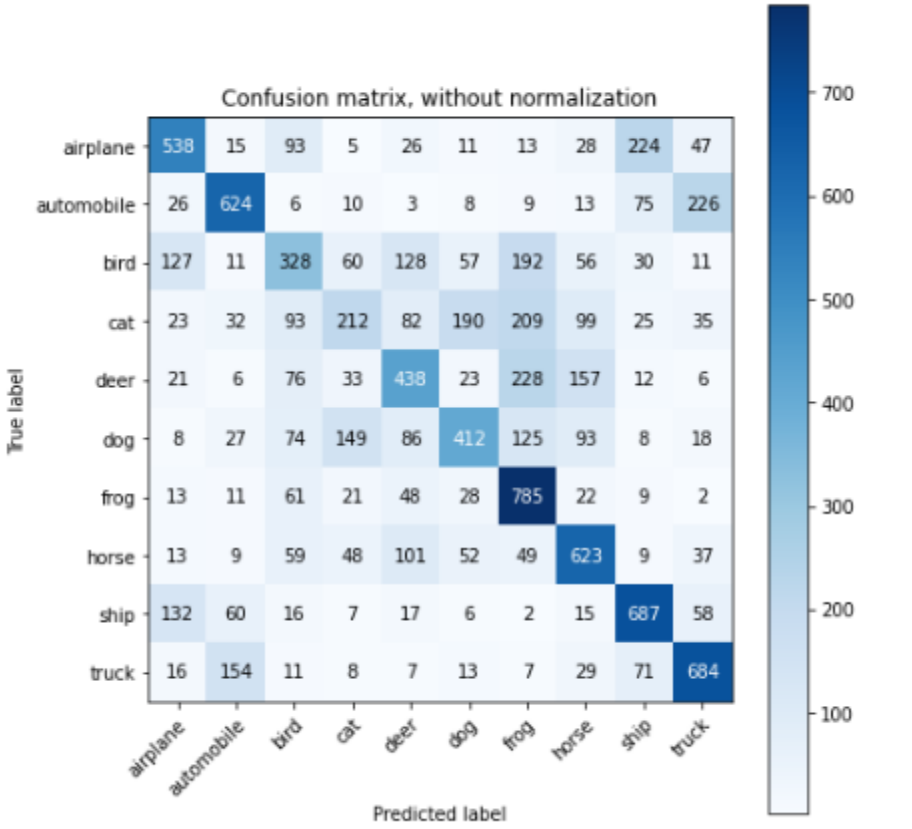
```
Epoch 1/4
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: UserWarning: `Model.fit_generator` is deprecated
  """
245/245 [==============================] - ETA: 0s - loss: 2.2878 - accuracy: 0.1383WARNING:tensorflow:Your inpu
WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available
245/245 [==============================] - 505s 2s/step - loss: 2.2878 - accuracy: 0.1383 - val_loss: 2.2403 - v
Epoch 2/4
245/245 [==============================] - ETA: 0s - loss: 2.1101 - accuracy: 0.2611WARNING:tensorflow:Learning
245/245 [==============================] - 453s 2s/step - loss: 2.1101 - accuracy: 0.2611 - lr: 0.0010
Epoch 3/4
245/245 [==============================] - ETA: 0s - loss: 1.7236 - accuracy: 0.3884WARNING:tensorflow:Learning
245/245 [==============================] - 443s 2s/step - loss: 1.7236 - accuracy: 0.3884 - lr: 0.0010
Epoch 4/4
245/245 [==============================] - ETA: 0s - loss: 1.4436 - accuracy: 0.4802WARNING:tensorflow:Learning
245/245 [==============================] - 462s 2s/step - loss: 1.4436 - accuracy: 0.4802 - lr: 0.0010
<keras.callbacks.History at 0x7fb2b2d83690>
```

After successful training, I will visualize its performance.



Confusion Matrix

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb2af797c10>
```



Accuracy

```
#The average accuracy score in classifying the unseen test data will be obtained now.

#Classification accuracy
from sklearn.metrics import accuracy_score
acc_score = accuracy_score(y_true, y_pred)
print('Accuracy Score = ', acc_score)
```

```
Accuracy Score =  0.5331
```