PROJECT 3: Object classification using Neural networks

Dogs vs Cats classification using Residual Learning and VGG16 for Image Recognition

September 14, 2020

I. Authors:

Student Name	Student ID
Trần Đại Chí	18127070
Phan Tấn Đạt	18127078
Thái Nhật Tân	18127204

II. Abstract:

Dogs and Cats photos are classified using an advanced convolution neural networks. This this project, we will implement a Residual Network and a VGG16 Network which was built and trained from scratch on Google Colab using Tensorflow and Keras module. Moreover, the dataset provided by kaggle was used in training and testing process of this project.

III. Introduction:

Dogs and cats are human's most beloved animals. The internet is full of cats and dogs images which people uploaded to share the cuteness of their favourite pets to the world. This lead to the problem of categorizing these images among trillions of photos, this job can be performed automatically using the applications of neural networks to save human's time.

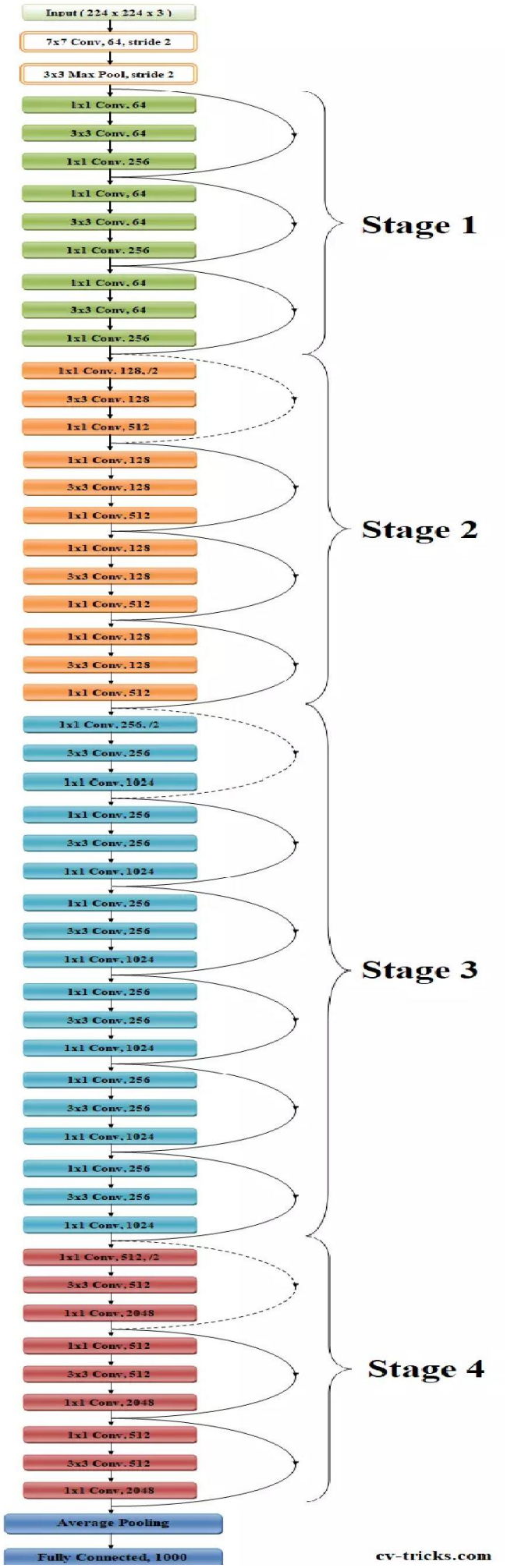
In this project, we built a neural network model in python with the help of tensorflow and keras module to simulate Residual Networks with 50 layers and VGG16 which is are well-known neural networks used as a backbone for many computer vision tasks nowadays.

This is a supervised learning neural network trained using the data set from kaggle on Google Colabs.

IV. Background/Related Work:

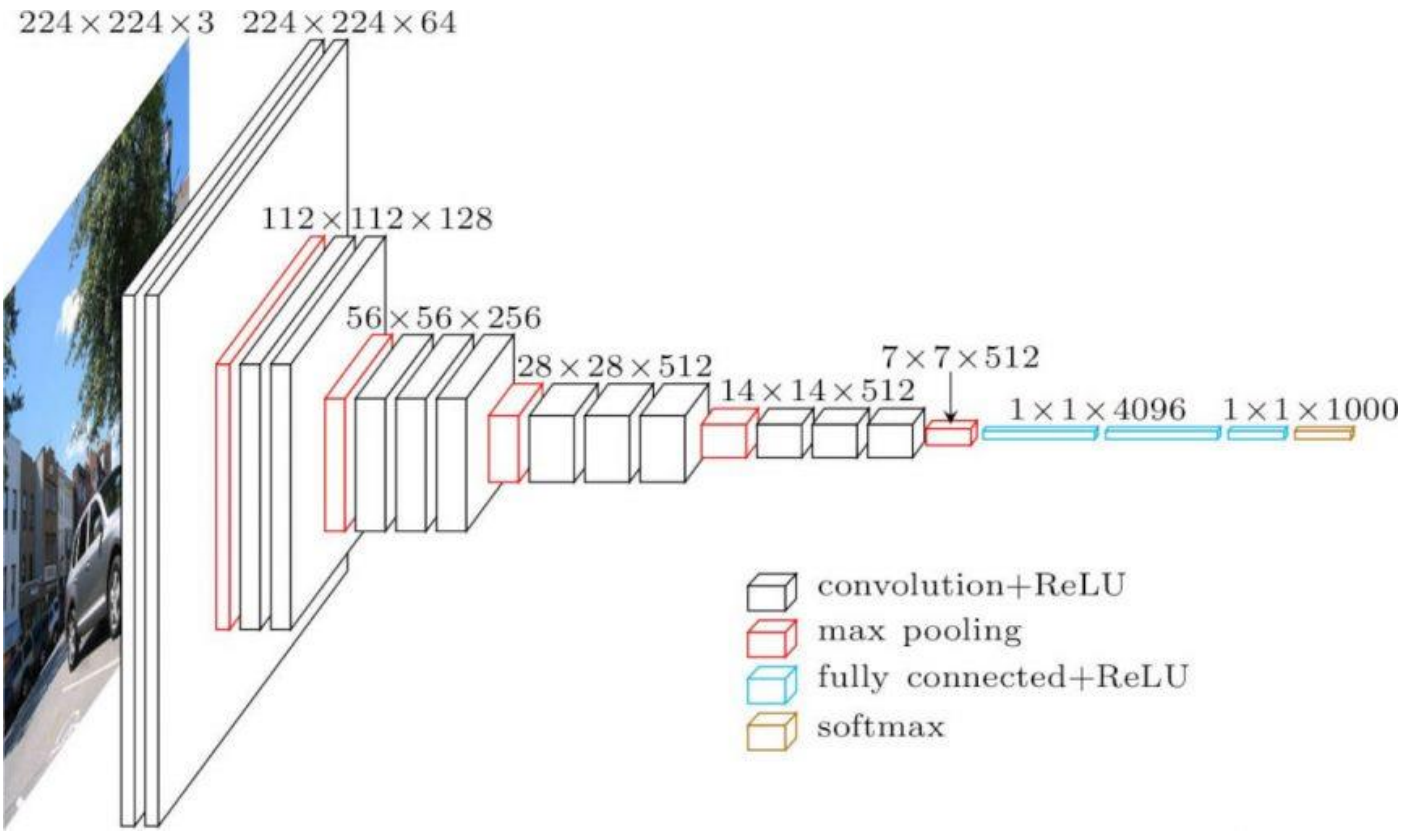
1. Resnet50

– Resnet50's architecture has 4 stages, which is following in this image below:


Input (224 x 224 x 3)
7x7 Conv, 64, stride 2
3x3 Max Pool, stride 2
1x1 Conv, 64
3x3 Conv, 64
1x1 Conv, 256
1x1 Conv, 64
3x3 Conv, 64
1x1 Conv, 256
1x1 Conv, 64
3x3 Conv, 64
1x1 Conv, 256
Stage 1
1x1 Conv, 128, /2
3x3 Conv, 128
1x1 Conv, 512
1x1 Conv, 128
3x3 Conv, 128
1x1 Conv, 512
1x1 Conv, 128
3x3 Conv, 128
1x1 Conv, 512
1x1 Conv, 128
3x3 Conv, 128
1x1 Conv, 512
Stage 2
1x1 Conv, 256, /2
3x3 Conv, 256
1x1 Conv, 1024
1x1 Conv, 256
3x3 Conv, 256
1x1 Conv, 1024
1x1 Conv, 256
3x3 Conv, 256
1x1 Conv, 1024
1x1 Conv, 256
3x3 Conv, 256
1x1 Conv, 1024
1x1 Conv, 256
3x3 Conv, 256
1x1 Conv, 1024
1x1 Conv, 256
3x3 Conv, 256
1x1 Conv, 1024
Stage 3
1x1 Conv, 512, /2
3x3 Conv, 512
1x1 Conv, 2048
1x1 Conv, 512
3x3 Conv, 512
1x1 Conv, 2048
1x1 Conv, 512
3x3 Conv, 512
1x1 Conv, 2048
Stage 4
Average Pooling
Fully Connected, 1000
cv-tricks.com

- We consider the input size as 224x224x3. Resnet architecture performs the initial convolution use 7x7 and max-pool use 3x3
- After that. In stage 1 in network start, it has 3 residual blocks, which contain 3 layers each. Kernels's size use to perform the convolution operation in all 3 layers of the block of stage 1 are 64,64,128.
- The curved arrows refer to the identity connection, the dashed connected arrow shows that convolution operation in the residual block is performed in stride 2. So size of input will reduce half in height and width, but channel width will increase two times.
- When progressing from one stage to other stage, channel width will increase two times, size of input will reduce half
- For each residual function, 3 layers are stacked one over the other. 3 layers are 1x1, 3x3, 1x1 convolutions. The 1x1 convolution layers are reduce and then restore the dimensions. The 3x3 layer is left with smaller input/output dimensions
- Finally, network has an average pooling layer followed by a fully connected layer having 1000 neurons

2. VGG16



- We consider the input size as 224x224x3. Image is passed through a stack of convolutional layers, where the filters were used with a very small

receptive field 3x3. In one configurations, it also utilizes 1x1 convolution filters. The convolution stride is fixed to 1 pixel, layer input is such that the spatial resolution is preserved after convolution, mean that 1 pixel for 3x3 convolution layers.

- Spatial pooling is carried out by 5 max pooling layers, which follow some of the convolution layers. Max pooling is performed over a 2x2 pixel with stride is 2
- 3 Fully connected layers follow a stack of convolutional layers: the first two have 4096 channels each, the third perform 1000 way classification and contains 1000 channels.
- The final layer is the soft max , the configuation of the fully connected layers is the same in all networks

V. Approach:

A. Preprocessing data process:

1. Download the dataset from the link https://www.kaggle.com/c/dogs-vs-cats/data
2. Unzip *dogs-vs-cats.zip* to get *train.zip* and *test1.zip*
3. Unzip *train.zip* and *test1.zip*
4. Randomly divide the images in folder *train* into a folder called *cats_and_dogs_filtered* consist of 2 sub-folder called *train* and *test*, each will have two folders *cats* and *dogs* according to their labels
5. When being used for training, these images will be resizes to 224*224 by default

B. Implementation:

Resnet:

The structure of ResNet-50 in this project consists of a 7x7 convolutional layer, 4 convolutional blocks, 12 identity blocks and a dense layer

1. The model building process:
   - Define identity block.
   - Define convolutional block.
   - Build the structure of ResNet50 without top layer.
   - Load weights without top layer which were downloaded from keras trained by ImageNet from https://github.com/fchollet/deep-learning-models/releases/download/v0.2/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5. The weights of top layer were trained by the dataset downloaded from kaggle as described in **Preprocessing data process**.
   - Add top layer to ResNet50.
   - Setup training attribute from *./cats_and_dogs_filtered/train* and *./cats_and_dogs_filtered/test*.
   - Compile the model.
2. Train ResNet-50 for Cats.Vs.Dogs.
3. Save the best model.
4. Use the best model to predict images

VGG16:

The architecture of VGG16 in this project consists of 16 layers with 2 convolution layer of 64 channel of 3x3 kernel, 2 convolution layer of 128 channel of 3x3 kernel, 3 convolution layer of 256 channel of 3x3 kernel, 6

convolution layer of 512 channel of 3x3 kernel, 3 dense layer and all of them has same padding

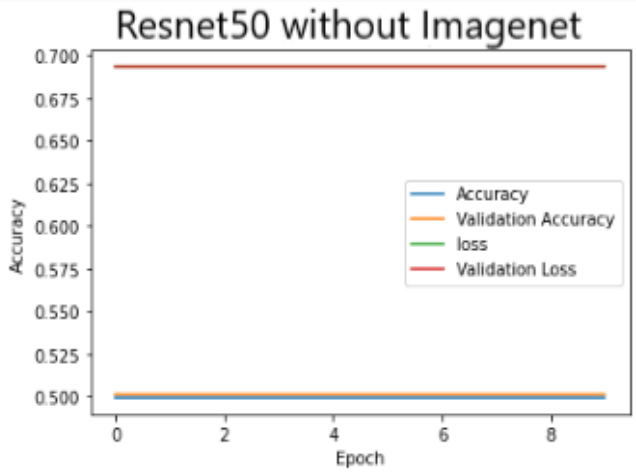
1. Build the architecture of VGG16
2. Load weights (imagenet) for VGG16 from this link https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels.h5
3. We will load model and make loaded layers as not trainable then compile the model
4. Make a call back to save our model with best val_accuracy
5. Using library opencv to read image and resize all of them to same size (224, 224) with 3 channels and load model to predict our image is dog or cat

VI. Experiment:

In the first try of training both model from scratch, using only the data set from kaggle, the validation accuracy is very low and will require many epochs to increase it as shown in the figure below:

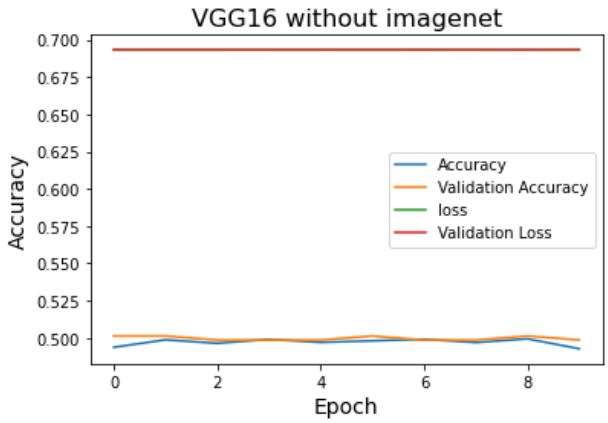
Resnet50:

```
Epoch 1/10
187/187 [==============================] - 106s 567ms/step - loss: 0.6931 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 2/10
187/187 [==============================] - 106s 564ms/step - loss: 0.6931 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 3/10
187/187 [==============================] - 106s 566ms/step - loss: 0.6931 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 4/10
187/187 [==============================] - 108s 575ms/step - loss: 0.6931 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 5/10
187/187 [==============================] - 108s 579ms/step - loss: 0.6931 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 6/10
187/187 [==============================] - 108s 580ms/step - loss: 0.6931 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 7/10
187/187 [==============================] - 109s 580ms/step - loss: 0.6931 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 8/10
187/187 [==============================] - 108s 577ms/step - loss: 0.6931 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 9/10
187/187 [==============================] - 108s 578ms/step - loss: 0.6931 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 10/10
187/187 [==============================] - 109s 580ms/step - loss: 0.6931 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5013
```



VGG16:

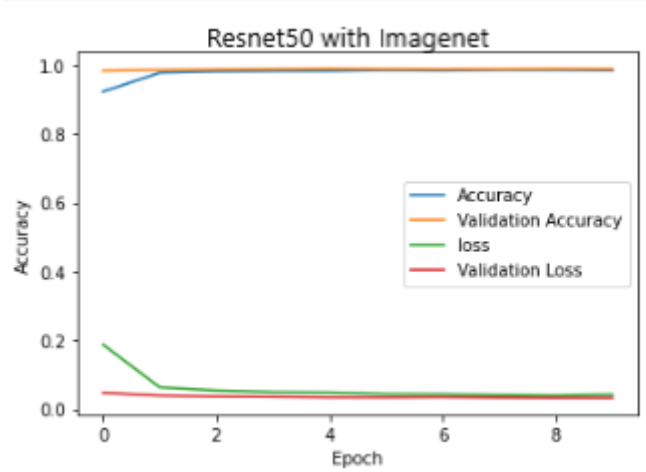
```
Epoch 1/10
293/293 [==============================] - 179s 612ms/step - loss: 0.6932 - accuracy: 0.4938 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 2/10
293/293 [==============================] - 179s 612ms/step - loss: 0.6932 - accuracy: 0.4988 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 3/10
293/293 [==============================] - 179s 611ms/step - loss: 0.6932 - accuracy: 0.4964 - val_loss: 0.6931 - val_accuracy: 0.4987
Epoch 4/10
293/293 [==============================] - 179s 611ms/step - loss: 0.6932 - accuracy: 0.4991 - val_loss: 0.6931 - val_accuracy: 0.4987
Epoch 5/10
293/293 [==============================] - 179s 612ms/step - loss: 0.6932 - accuracy: 0.4971 - val_loss: 0.6931 - val_accuracy: 0.4987
Epoch 6/10
293/293 [==============================] - 179s 611ms/step - loss: 0.6932 - accuracy: 0.4981 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 7/10
293/293 [==============================] - 179s 611ms/step - loss: 0.6932 - accuracy: 0.4992 - val_loss: 0.6931 - val_accuracy: 0.4987
Epoch 8/10
293/293 [==============================] - 179s 611ms/step - loss: 0.6932 - accuracy: 0.4970 - val_loss: 0.6931 - val_accuracy: 0.4987
Epoch 9/10
293/293 [==============================] - 179s 612ms/step - loss: 0.6932 - accuracy: 0.4994 - val_loss: 0.6931 - val_accuracy: 0.5013
Epoch 10/10
293/293 [==============================] - 179s 611ms/step - loss: 0.6932 - accuracy: 0.4928 - val_loss: 0.6931 - val_accuracy: 0.4987
```



To increase the validation accuracy of both model, the transfer learning method is applied by loading weights without top layer trained by ImageNet from which were downloaded from https://github.com/fchollet/deep-learning-models/releases/download/v0.2/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5. This will significantly increase the validation accuracy and improvement can be seen clearly after each epochs

Resnet50:

```
Epoch 1/10
187/187 [==============================] - 110s 588ms/step - loss: 0.1873 - accuracy: 0.9222 - val_loss: 0.0475 - val_accuracy: 0.9827
Epoch 2/10
187/187 [==============================] - 108s 580ms/step - loss: 0.0631 - accuracy: 0.9772 - val_loss: 0.0393 - val_accuracy: 0.9860
Epoch 3/10
187/187 [==============================] - 108s 577ms/step - loss: 0.0539 - accuracy: 0.9808 - val_loss: 0.0367 - val_accuracy: 0.9868
Epoch 4/10
187/187 [==============================] - 108s 575ms/step - loss: 0.0496 - accuracy: 0.9820 - val_loss: 0.0354 - val_accuracy: 0.9876
Epoch 5/10
187/187 [==============================] - 107s 573ms/step - loss: 0.0485 - accuracy: 0.9824 - val_loss: 0.0339 - val_accuracy: 0.9887
Epoch 6/10
187/187 [==============================] - 107s 574ms/step - loss: 0.0439 - accuracy: 0.9845 - val_loss: 0.0337 - val_accuracy: 0.9879
Epoch 7/10
187/187 [==============================] - 107s 570ms/step - loss: 0.0437 - accuracy: 0.9834 - val_loss: 0.0344 - val_accuracy: 0.9875
Epoch 8/10
187/187 [==============================] - 107s 571ms/step - loss: 0.0410 - accuracy: 0.9853 - val_loss: 0.0333 - val_accuracy: 0.9878
Epoch 9/10
187/187 [==============================] - 108s 575ms/step - loss: 0.0427 - accuracy: 0.9838 - val_loss: 0.0325 - val_accuracy: 0.9879
```

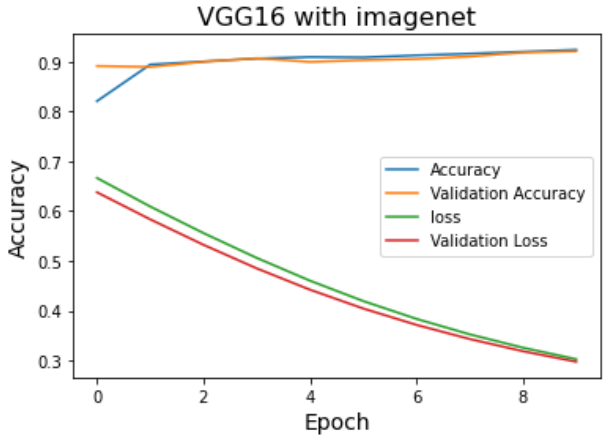


The Hierarchical Data Format version 5 file (.h5) of this training process:

https://drive.google.com/file/d/1QjCmciX476oMC9FdZp3USRG6p3XTC7UL/view?usp=sharing

VGG16:

```
Epoch 1/10
293/293 [==============================] - 178s 609ms/step - loss: 0.6663 - accuracy: 0.8205 - val_loss: 0.6376 - val_accuracy: 0.8910
Epoch 2/10
293/293 [==============================] - 178s 609ms/step - loss: 0.6090 - accuracy: 0.8940 - val_loss: 0.5832 - val_accuracy: 0.8893
Epoch 3/10
293/293 [==============================] - 178s 609ms/step - loss: 0.5557 - accuracy: 0.9001 - val_loss: 0.5322 - val_accuracy: 0.8994
Epoch 4/10
293/293 [==============================] - 179s 611ms/step - loss: 0.5058 - accuracy: 0.9061 - val_loss: 0.4848 - val_accuracy: 0.9067
Epoch 5/10
293/293 [==============================] - 181s 617ms/step - loss: 0.4602 - accuracy: 0.9090 - val_loss: 0.4420 - val_accuracy: 0.8993
Epoch 6/10
293/293 [==============================] - 179s 611ms/step - loss: 0.4192 - accuracy: 0.9084 - val_loss: 0.4043 - val_accuracy: 0.9027
Epoch 7/10
293/293 [==============================] - 178s 609ms/step - loss: 0.3835 - accuracy: 0.9125 - val_loss: 0.3714 - val_accuracy: 0.9051
Epoch 8/10
293/293 [==============================] - 178s 609ms/step - loss: 0.3524 - accuracy: 0.9159 - val_loss: 0.3430 - val_accuracy: 0.9102
Epoch 9/10
293/293 [==============================] - 178s 609ms/step - loss: 0.3256 - accuracy: 0.9197 - val_loss: 0.3185 - val_accuracy: 0.9180
Epoch 10/10
293/293 [==============================] - 180s 615ms/step - loss: 0.3026 - accuracy: 0.9236 - val_loss: 0.2974 - val_accuracy: 0.9207
```



The Hierarchical Data Format version 5 file (.h5) of this training process:

https://drive.google.com/file/d/115iJI86BvOH5QqnUCnFCf-kSA4f2tCWB/view?usp=sharing

VII. Conclusion:

From the graphs of Resnet50 and VGG16 with Imagenet, it is clear that Resnet50 is better with lower *loss* and *validation loss* from the first epoch. At the 10th epoch, *validation accuracy* and *accuracy* of Resnet50 is nearly 1. Meanwhile, VGG16's *validation accuracy* and *accuracy* was only 9.2 at max. This mean that VGG16 will need more epochs to train in order to reach high *validation accuracy* comparing to Resnet50.

VIII. Demo video:

VGG16 +Local web demo: https://youtu.be/kszk4Ne8MN0

Resnet: https://youtu.be/Lyry_49IOiQ

IX. References:

https://www.kaggle.com/suniliitb96/tutorial-keras-transfer-learning-with-resnet50

https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

https://adventuresinmachinelearning.com/introduction-resnet-tensorflow-2/?fbclid=IwAR3TUos4fBGRqNYh3e5F1rZW62cXla7ZqOZdzR5VxK01E0dTodSNRSEltPY

https://en.wikipedia.org/wiki/Residual_neural_network

https://cv-tricks.com/keras/understand-implement-resnets/

https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c

https://neurohive.io/en/popular-networks/vgg16/