Image Classification with Transfer Machine Learning and Fine Tuning

Vikas Kumar

Northeastern University

Dec 2018

Abstract

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. The objective of our project is to use the methodology of Transfer Learning to classify custom images and improve the accuracy by varying the convolutional network of a particular model. In this paper we are considering three image classification pre-trained models on Image Net dataset namely, VGG 16, Inception V3 and Xception model to compare their relative accuracy and maximize their individual accuracies by varying and selecting the best parameters and Fine Tuning the convolutional layer.

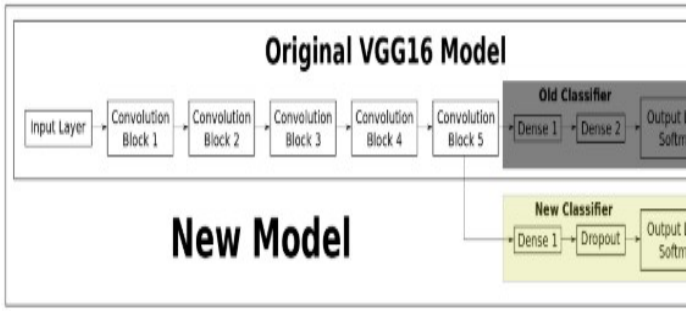
Introduction

Image classification refers to the task of extracting information classes from a multiband raster image. The resulting raster from image classification can be used to create thematic maps.

Dataset

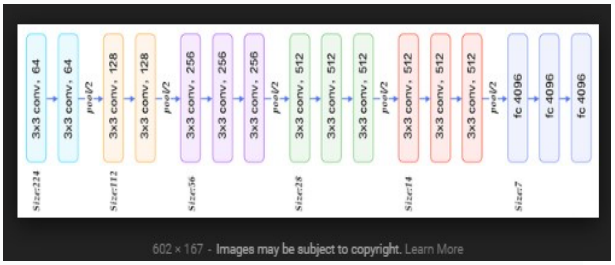
This paper implements three pre trained models of Keras and compares them in terms of Validation accuracy. The dataset used is a Monkey Classification dataset with 10 classes. The pre trained models were trained on ImageNet dataset which is a large visual database designed for use in visual object recognition software research. The pre trained weights can be downloaded from https://github.com/fchollet/deep-learningmodels.

Transfer Learning



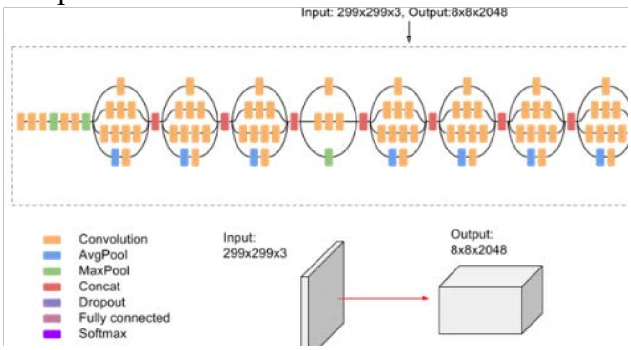
It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision

and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.



Inception V3

We are using Inception since it has been known to attain greater than 78.1% accuracy on the ImageNet dataset. The model itself is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers. Batchnorm is used extensively throughout the model and applied to activation inputs. Loss is computed via Softmax.



VGG 16

VGG16 is a convolutional neural network architecture and excellent vision model. This network is characterized by its simplicity, using only 3×3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fullyconnected layers, each with 4,096 nodes are then followed by a softmax classifier

Xception

We are using Xception since it is an extension of the Inception architecture which replaces the standard Inception modules with depthwise separable convolutions.

Methodolgy

For our project we designed classifiers using above three pre-trained models.

Preprocessing Image Data for Input to Network:

We are using Keras's data-generator for inputting data into the neural network, which will loop over the data for eternity.

Preprocessing Image Data for Train:

We have a small training-set so it helps to artificially inflate its size by making various transformations to the images. We use a builtin data-generator that can make these random transformations

Preprocessing Image Data for Test:

We want to measure exact classification accuracy on test so we do not perform any transformations

Model Building:

Parameters used to build the model:

1. Pre-trained Convolutional layer

```
_________________________________________
Layer (type)                 Output Shap
=========================================
inception_v3 (Model)         (None, 2048
_________________________________________
dense_2 (Dense)              (None, 512)
_________________________________________
dropout_2 (Dropout)          (None, 512)
_________________________________________
dense_3 (Dense)              (None, 10)
=========================================
Total params: 22,857,002
Trainable params: 1,054,218
Non-trainable params: 21,802,784
_________________________________________
```

2. Fully Connected Layer with dropout
3. Fully Connected Layer without dropout

```
model_NoDropOut.summary()
```

```
_________________________________________
Layer (type)                 Outpu
=================================
vgg16 (Model)                (None
_________________________________________
dense_19 (Dense)             (None
_________________________________________
dense_20 (Dense)             (None
=================================
Total params: 14,982,474
Trainable params: 267,786
Non-trainable params: 14,714,688
_________________________________________
```

4. Varying Learning rates
5. Varying Optimzer functions
6. Varying the number of Fully Connected Layers
7. Varying the number of neurons in Fully Connected Layers

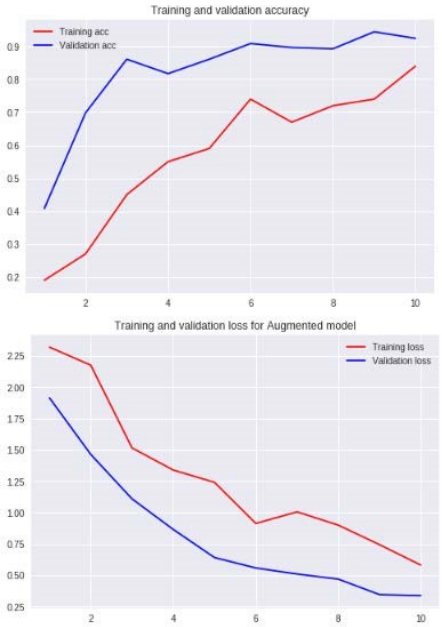
```
model_FC_history = model_FC.fit_generator(train_gen,
                                          steps_per_epoch=5
                                          epochs=10,
                                          validation_data=v
                                          validation_steps=
                                         )
```

9. Fine Tuning

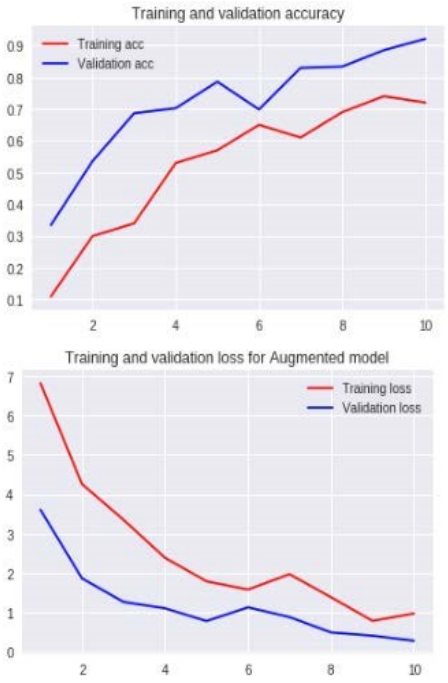
Once the new classifier has been trained we can try and gently fine-tune some of the deeper layers in the VGG16 model as well. We call this Fine-Tuning.We build a base model and validated its accuracy with each varying parameters

Validation Plots:

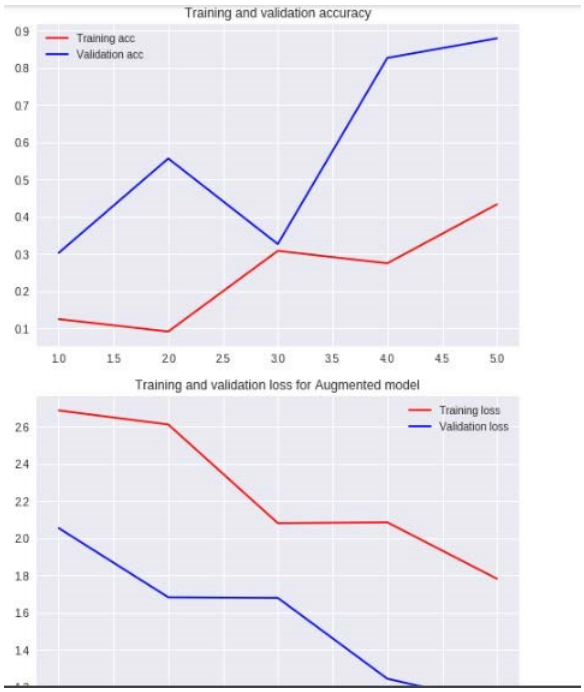
VGG_16



8. No. of Epochs



Inception_V3 accuracy



Xception_V3 accuracy

Results:

Validation accuracy for 3 models

	INCEPTION V3	VGG_16	XCEPTION
Base Model	0.8548	0.9405	0.9484
augmented Model	0.879	0.9206	0.9246
1 Fully Connected layer with dropout	0.8024	0.9405	0.9722
1 Fully Connected layer w/o dropout	0.8024	0.9365	0.8056
2 Fully Connected layer with dropout	0.8105	0.9444	0.9841

Varying Nodes Fine_Tuned	0.6653	
		0.9563

We only fined tuned VGG_16 because of its less complex architecture compared to other two.

Conclusion:

We implemented Transfer Learning and Fine Tuning with pre-trained weights of 3 models. We trained our models on multiple parameters and the performance of all the three models were close with VGG_16 with maximum of 94% accuracy. We can also conclude that the accuracy of the model increases with fine tuning the model

References:

[1]https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/10_Fine-Tuning.ipynb

[2]https://keras.io/applications/

[3]https://towardsdatascience.com/computer-vision-by-andrew-ng-11-lessons-learned7d05c18a6999

[4]http://delivery.acm.org/10.1145/1280000/1273592/p759-raina.pdf?ip=155.33.134.9&id=1273592&acc=ACTIVE%20SERVICE&key=AA86BE8B6928DDC7%2EC2B8A117C7A71F5A%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1544804536_cd39e224c022a17455b90ca620506126

**