

# Multi-Image Classification

# TensorFlow - Keras

EC327 Final Project Documentation On Model 5

Todd Tian, Purna

Model 1 Model 2 and 3 Model 4 Model 5 and PPT Final Doc and Video

Dec 8 Dec 9 Dec 10 Dec 11 Dec 12

# Introduction

Our model employs the use of an open-source platform called TensorFlow which is developed by Google primarily for deep learning applications. TensorFlow provides pre-built functions and advanced operations to ease the task of building different neural networks. This platform provides a comprehensive and flexible ecosystem of tools, community tools and libraries. Additionally, a high level interface, called Keras, is used in TensorFlow to solve machine learning problems. Keras provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity.

# Final Model

## Pretraining on MobileNetV2

For our fifth and final model, we ran our data on **MobileNetV2**, an open source model trained by Google for image recognition. Specifically, MobileNetV2 is an improved convolutional neural network (CNN) algorithm that is based on an *inverted residual structure* where the input and output of the residual blocks are *thin bottleneck layers*, as opposed to how traditional residual models operate. Model 5, unsurprisingly, had the best accuracy rate even on the first epoch. This is because this open source image classifier has been trained on millions of images so that this model would be able to classify virtually any category of images. It’s extremely efficient and able to run through each epoch at a much faster learning rate with no overfitting at all. These models serve as the foundation for image classification that extends to products such as your phone's facial recognition abilities. It is virtually perfect when it comes to static images even if there are several classes.

## Inverted Residual Structure and Bottlenecks

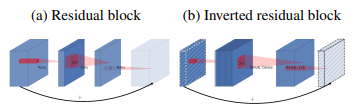
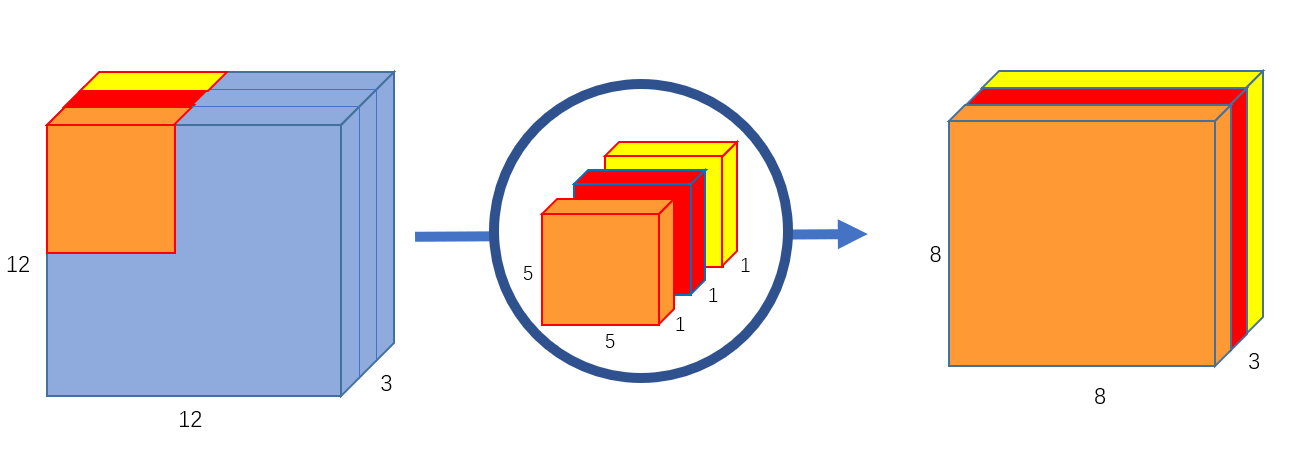
To understand what an inverted residual structure is, it helps to understand the differences between that and a normal residual structure. When running a normal residual structure, the hidden layer first filters the features of the images, then squeezes out the most important features from those images. An **inverted residual structure**, on the other hand, squeezes the features using a 1x1 block, then filters the features in a 3x3 block, and finally squeezes the most important features of the image again using a 1x1 block (**Figure 7**). The initial problem with this, however, is that this structure traditionally runs all blocks with the relu activation function, meaning that the squeeze on the final block does not take into account the negative values of the image. This can be detrimental to the model because negative values that are not taken into account can be important when classifying an image. 

Figure 7

For example, referring to **Figure 8**, it would be better to squeeze the features of the dog along with residuals of the grass, compared to squeezing the image where part of the dog’s head may be missing. As a result, MobileNetV2 implements **thin bottleneck layers**. The bottleneck layers remove the Relu activation function at the last 1x1 block, which guarantees the preservation of the entire object of the image we are trying to classify. As a result, this makes the model much more confident when classifying which 

animal is which by creating a model that is much more flexible across the training and validation sets, and decreases the bias for both datasets substantially.

## Depthwise Conv2D and Batch Normalization

Model 5 also makes use of depthwise Conv2D, unlike the traditional Conv2D discussed earlier, and batch normalization. Unlike Conv2D, where the features of an image are split into one layer, **depthwise Conv2D** (as displayed in **Figure 9**) splits the features of an image into multiple layers within the hidden layer, and returns features in separate layers, each representing specific attributes of the image (E.g. One layer representing texture, a second layer representing color, a third layer representing size, etc.). While this separates the features, the problem with this is the layers become very messy because as you go deeper into the neural network and the number of filters applied increases. As a result, **batch normalization** is needed to renormalize each layer that represents the features before you pass into the next hidden layer. Because the features and inputs of the model are normalized, the implementation of depthwise conv2D and batch normalization allows the model to become much more efficient, requiring the running of less epochs.

# Conclusion

What we’ve taken away from diving into this platform is that computational power is key. Along with model efficiency, it is rather difficult to iterate on a data set when each epoch run takes several hours. During our workflow, we’ve gained a new appreciation and admiration for open source R&D projects and supercomputers in general. This project would’ve been virtually impossible if not for the open source tools developed by the Tensorflow and Keras team. Machine learning is hard but creating machine learning tools is even harder.