Statistical Methods for Machine Learning

Prashant Bahuguna

January 2023

1 Introduction

The project uses pretrained Deep Learning neural networks in TensorFlow2 for the binary classification of cats and dogs. Matrices (accuracy) are obtained as an average of 5 fold cross validation. I use Transfer Learning approach for concluding this project. Transfer Learning also known as Inductive Learning is a Supervised Learning technique (mapping an input to an output based on example of input-output pairs) that re-uses parts of a previously trained model on a new network which may or may not be tasked with the similar problem. Using such pre-trained models reduces the time (greatly) required for training.
Models used:

- InceptionV3
- MobileNetV3Small
- EfficientNetV2B2
- VGG16

2 Neural Networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the brain. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning largely involves adjustments to the synaptic connections that exist between the neurons.

2.1 Architecture for Neural Networks

- Input Layer- As the name suggests, it accepts inputs in several different formats provided by the programmer.
- Hidden Layer- The hidden layer presents in-between input and output layers. It performs all the calculations
- Output Layer
  The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer. The artificial neural network takes input

and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^{n} Wi * Xi + b$$

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

2.2 Modifications in PreTrained Models

All the models listed above used for training went through the same heuristics:

1) Removal of the final dense/classification layer

2) Doing global average pooling in 2D

3) Finally, adding the dense classification layer of rank 2.

Following the exec code with in my script—-

```
# This is the common configuration using weights of the imagenet (explained in next section)
# Input shape is (256, 256)
# Include top = False, executes step 1 above.
config = {"weights": "imagenet", "include_top": False, "input_shape": (256, 256, 3)}

# Model is downloaded using the argument parser arguments based on the string(name of arch)
model = getattr(tf.keras.applications, model_arch)(**config)

# Execution of step 2
average = tf.keras.layers.GlobalAveragePooling2D()

# Execution of step 3
new_layer2 = Dense(2, activation="softmax", name="final_layer")

# Finally the keras Model is initialized as below-
Model(model.input, new_layer2(average(model.output)))
```

2.2.1 Why Imagenet?

The weights of Imagenet were used for the following reasons
1) Imagenet contains 1000 classes, and they share the similar features with the binary category for our classification

2) It is always my first preference to use pre trained weights because then chances of our optimizer getting stuck in local minima is readily reduced.

3 K-Fold Cross Validation

K-fold cross validation is a validation technique for multi-class classification. Results can be validated by distributing the dataset randomly in different groups.

3.1 5-Fold Cross Validation

In this project, I validated my result with 5-Fold Cross Validation by firstly shuffling the dataset and then by putting 1 set for validation and other 4 set for training purposes. Followed by repeating the process for all five sets.

4 CLI For Training

With the given training script, we can train list of models along with list of learning rates and batch sizes. The tensorboard logs and checkpoints (monitoring val loss) are stored automatically.

Batch sizes, Model architectures & learning rates should be a list of int, string and float respectively

```
# Examplary CLI instructions
python train.py --imdir "path/to/the/imagedir" --lr [0.001, 0.0001]
--m ["MobileNetV3Small", "EfficientNetV2B2"] --bs [18, 24] --epoch 5

where

imdir = PATH TO IMAGEDIR. The folder contains images of both Cats and Dogs as separate directories.
However, there are 2 erroneous files(11702.jpb in Dogs and  666.jpg in Cats) and should be filtered

lr = Learning rate
m = Model architecture
bs = Batch size
epoch = No. of epochs to train
```

5 Hyper Parameters

Generally a number of factors determines the choice of learning rate such as the choice of optimizer, depth of architecture, inter-class variance of classification categories. Personally I tested with lr of 0.01, 0.001 and 0.0001. However since the categories were just 2 and understandably simple (due to high contrasting features) it really didn't significantly impact the metrics. For instance, if I start with very low lr, I could achieve the same performances if I train it for more epochs. Little oscillations and trapping in local minima was observed(please refer to logs below) with lr higher than 0.001. Therefore finally, since the lr of 0.0001 is widely empirically accepted to work and I

went with it.

Ideally higher batch size is always better, however, due to constraints such as low GPU memory, I did the maximum batch size that I could use. I could have used Gradient Accumulation for higher batch sizes, however, the results were already better and therefore decided to drop that idea.

Just like batch size, high resolution images does indeed help in capturing granular features. However, due to memory constraints, I decided to go with (256, 256).

6 Augmentations

I used the following augmentations:

- Random horizontal flip - We can begin by randomly applying horizontal flip augmentation to our dataset and seeing how individual images will look after the transformation. This is achieved by passing

  `horizontal_flip=True`

  as an argument to the ImageDataGenerator class.

```
image_gen = ImageDataGenerator(rescale=1./255, horizontal_flip=True)

train_data_gen = image_gen.flow_from_directory(batch_size=BATCH_SIZE,
                                               directory=train_dir,
                                               shuffle=True,
                                               target_size=(IMG_SHAPE,IMG_SHAPE))
```

- Rotation - The rotation augmentation will randomly rotate the image up to a specfied number of degrees. Here, we'll set it to 10.

```
image_gen = ImageDataGenerator(rescale=1./255, rotation_range=10)

train_data_gen = image_gen.flow_from_directory(batch_size=BATCH_SIZE,
                                               directory=train_dir,
                                               shuffle=True,
                                               target_size=(IMG_SHAPE, IMG_SHAPE))
```

- Re-scaling/ Applying Zoom - We can also apply Zoom augmentation to our dataset, zooming images up to 50% randomly.

```
image_gen = ImageDataGenerator(rescale=1./255, zoom_range=0.5)
```

```
train_data_gen = image_gen.flow_from_directory(batch_size=BATCH_SIZE,
                                               directory=train_dir,
                                               shuffle=True,
                                               target_size=(IMG_SHAPE, IMG_SHAPE))
```

While optical augmentations could also be used, I decided to limit to just the ones above.

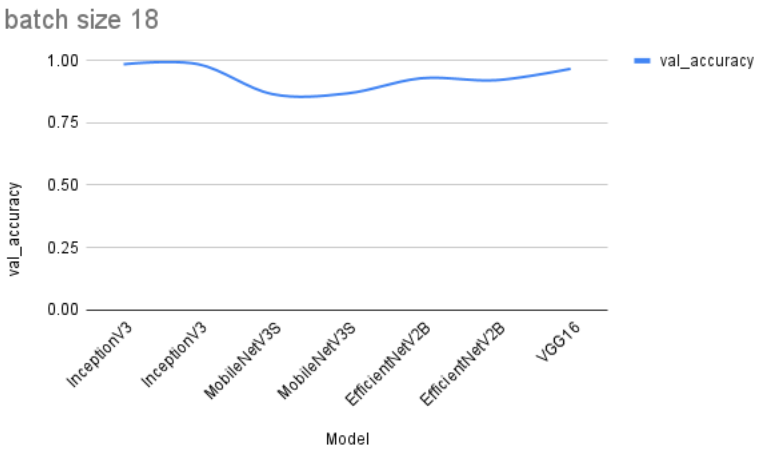
7 Inference

Model checkpoint is released here: download here
After downloading the model, follow the instructions

```
python inference.py path/to/image path/to/model
```

8 Metrics

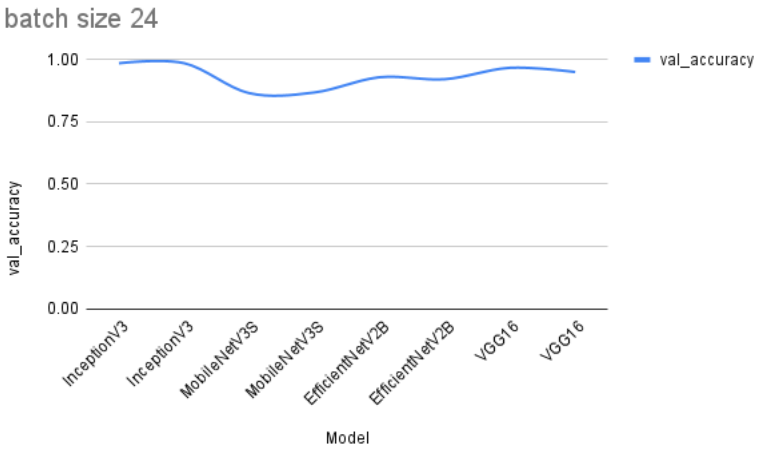
To save training time from all possible combinations, all the results below were computed for the learning rate of 0.0001. Also, they were trained just for 4 epochs with fixed image sizes of (256 x 256). As a rule of thumb, it is always preferable to use shallower networks unless absolutely necessary since bigger neural networks are harder to train.



Architecture	Batch$_{S}ize$	Accuracy
InceptionV3	18	0.9859
MobileNetV3Small	18	0.8658
EfficientNetV2B2	18	0.9298
VGG16	18	0.96778

Along with them, with a batch size of 24, the metrics are -

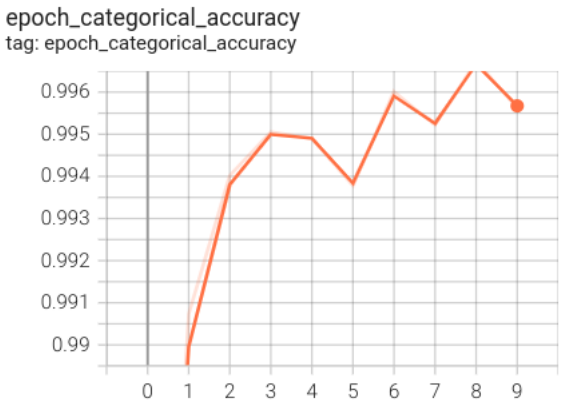
Architecture	Batch_{size}	Accuracy
InceptionV3	24	0.9862
MobileNetV3Small	24	0.8687
EfficientNetV2B2	24	0.9219
VGG16	24	0.9508



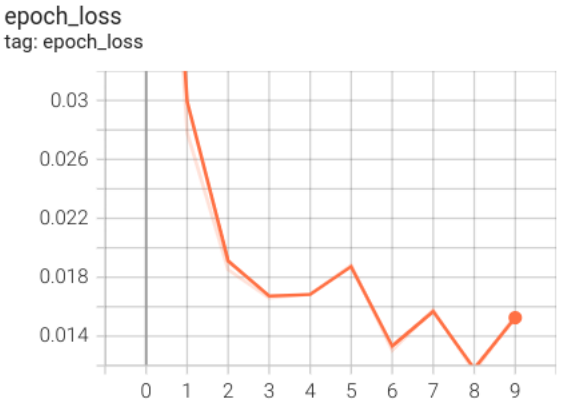
The criterion for saving the checkpoints has been validation loss

9 Observation On Results

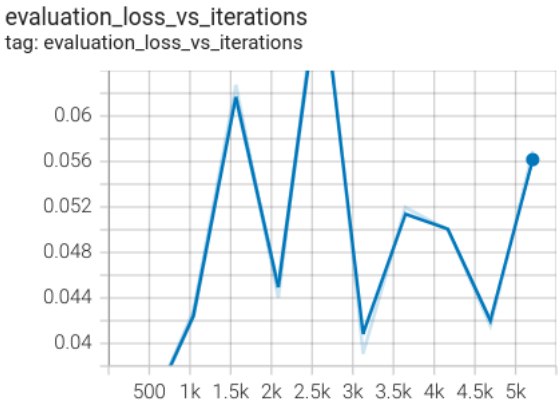
Within just 5 epochs, the best 5 fold accuracy was achieved by InceptionV3 model which is 98.62. At the moment, no scheduler was used since the number of epochs were already too less. During the training, logs are automatically saved as a callback.
Train Accuracy over 10 epochs



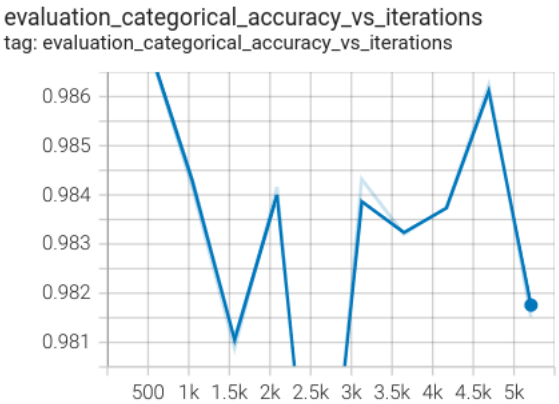
Train Loss over 10 epochs



Val Loss over 10 epochs



Val Accuracy over 10 epochs



Use of even lower learning rate such as 0.00001 could have been even better because we can see, there is little oscillation in the validation metrics. However, since the criterion for saving the checkpoints was val loss, the model released has the least val loss over all the epochs.

10 Conclusion and Further developments

- Since the inter class variance is quite high(due to distance features of the binary classes), shallow models were able to train themselves with high accuracy.
- Current images were just (256, 256) to accommodate them in my GPU, however, higher image sizes (512, 512) etc will assuredly lead to better results.
- The model can be further improved with higher batch sizes, high number of epochs along with the inclusion of more robust augmentations and finally the addition of scheduler. Since, I trained it for just 4 epochs, scheduler was a bit unwanted, however, I would incorporate if I train the model for more epochs.

11 Appendix

In this section we have supplementary material that is not an essential part of the text itself but which may be helpful in providing a more comprehensive understanding of the problem or it is information that is too cumbersome to be included in the body of the paper.

11.1 Requirements

The project contains a requirements.txt script that includes all the essential libraries that are required to run the project efficiently.
To install requirements-

```
pip install -r requirements.txt
```

11.2 Models used

1. InceptionV3
Inception v3 is a convolutional neural network architecture from the Inception family that makes several improvements including using Label Smoothing, Factorized 7 x 7 convolutions, and the use of an auxiliary classifier to propagate label information lower down the network (along with the use of batch normalization for layers in the side head). The key building block is an Inception Module.

- Parameters - 24 Million

2. MobileNetV3Small
MobileNetV3 is a convolutional neural network that is designed for mobile phone CPUs. The network design includes the use of a hard swish activation and squeeze-and-excitation modules in the MBConv blocks.

- Parameters - 3 Million

3. EfficientNetV2B2
EfficientNetV2 is a type convolutional neural network that has faster training speed and better parameter efficiency than previous models. To develop these models, the authors use a combination of training-aware neural architecture search and scaling, to jointly optimize training speed. The models were searched from the search space enriched with new ops such as Fused-MBConv.

- Parameters - 10.2 Million

4. VGG16
VGG16 is a convolution neural net (CNN) which is considered to be one of the excellent vision model architecture till date. Most unique thing about VGG16 is that instead of having a large number of hyper-parameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2.The 16 in VGG16 refers to it has 16 layers that have weights.

- Parameters - 138 Million

11.3 References

- Models Description -paperswithcode

12 Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.