Springboard Capstone Project 2

Classification of Images

(Deep Learning)

Abhishek Kumar

01 September 2020

Contents

1 Introduction

This project is about classifying images. If an image is fed into a machine, it will try to guess what the image is about. For example if the machine has been trained to classify among cats, dogs and birds and when an image is fed to the machine it will correctly say what the image was fed to it.

This project can be used for a variety of purposes. It can be used by govts, local authorities and private companies to see if people are ***wearing masks or not***. For a company dealing with superstore or vending machines, the project could help them maintain the ***count of the stock*** of different brands of the same product, say for example a superstore has 3 brands of ice-cream BR, Amul and Vadilal. Each time an ice cream of a brand is picked its stock count would change real time and this will help the client maintain the stock properly.This project could also help in self driving cars in ***classifying the road sign*** and help them take proper action. This project could also help in the ***medical industry*** by helping to classify between *healthy and unhealthy cells*. The potential of image classification is immense.

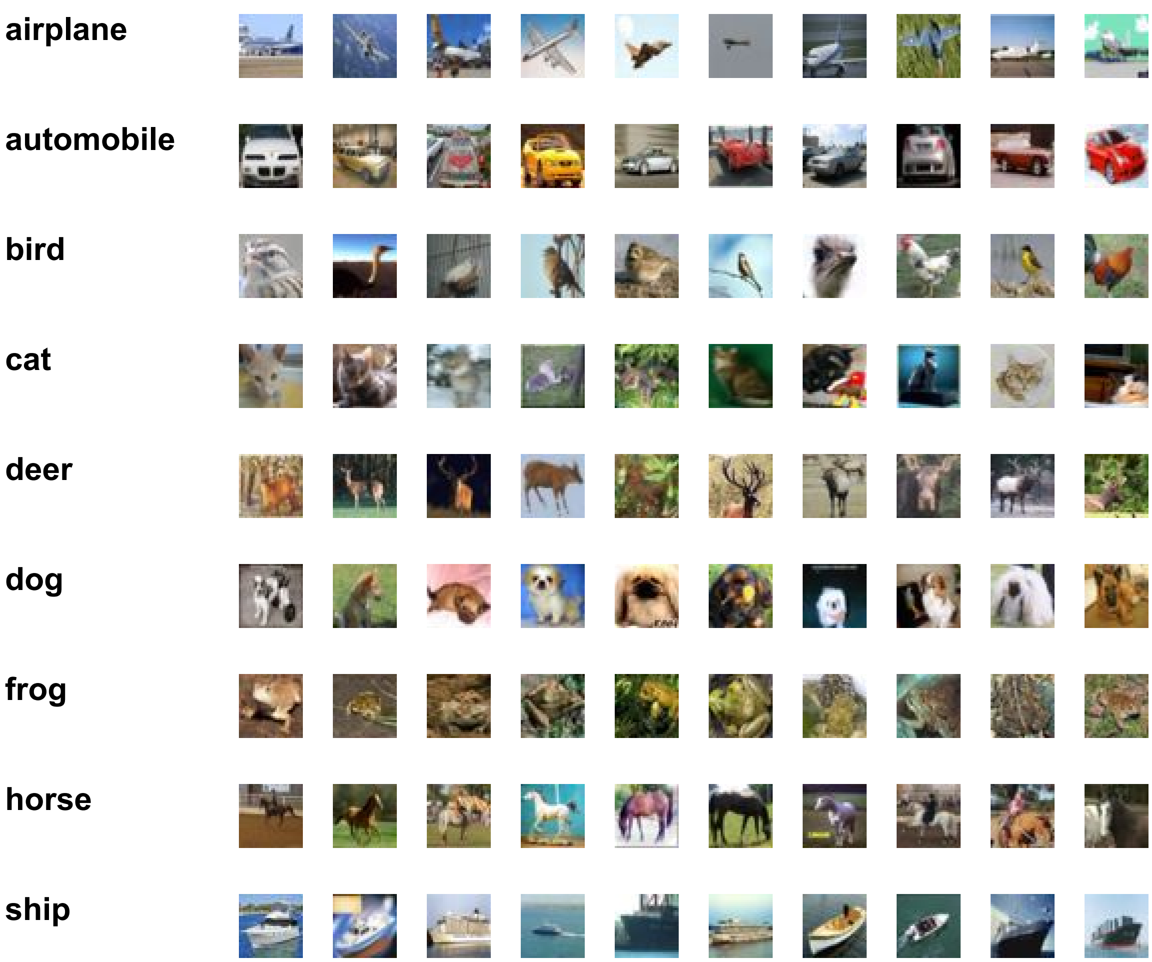
2 Data Acquisition and cleaning

The data set was collected from here. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton (credit).

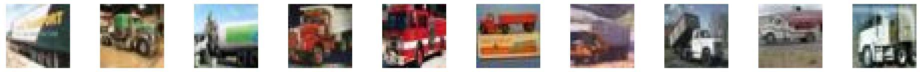
The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:



truck

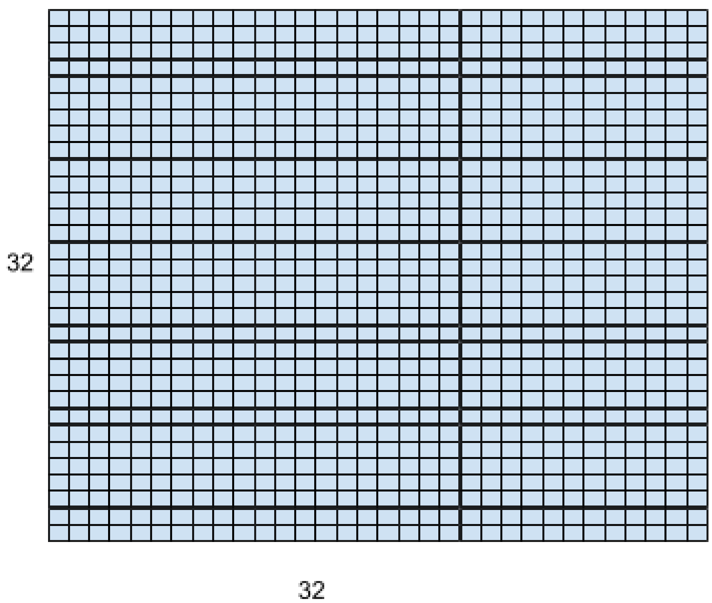


The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

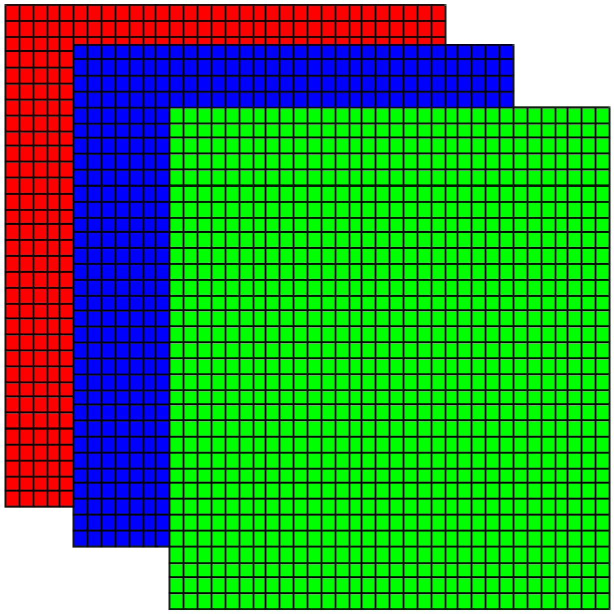
The dataset is clean and not much cleaning is required.

3 Data exploration

Each images are 32 * 32 pixels, what it means is that the height and width of the images are of 32 pixels each.



What does each pixel hold?
Each pixel has numbers which depend on the amount of red, green and blue colors (as we know that red blue and green are the universal colors and all the other colors are produced from combination of these three).



So for each pixel we will have three values.

4 Deep learning Models

After cleaning the data and analysing the images, we can proceed with model building. Here we use convolution neural networks (CNN) and in CNN we will use three different deep learning models, we will use keras along with tensorflow. We will use ***keras tuner***, **transfer learning** and ***image generators***.

4.1 Keras Tuner

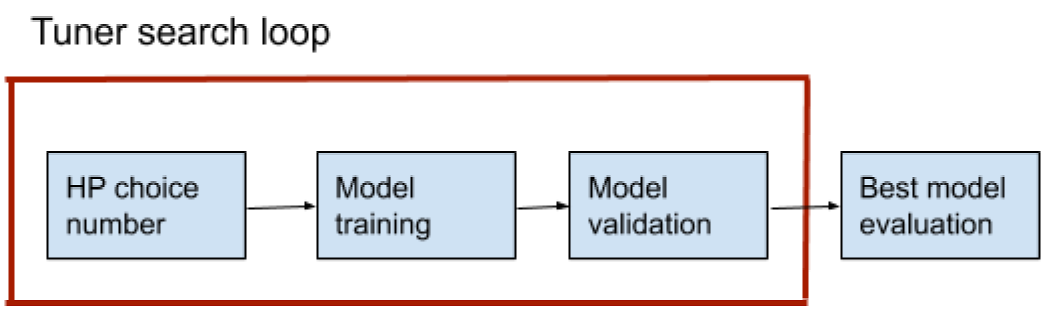
The **Keras Tuner** is a library that helps us pick the optimal set of hyperparameters for your TensorFlow program. The process of selecting the right set of hyperparameters for your machine learning (ML) application is called hyperparameter *tuning* or hypertuning.
It can be comparable to random search.

Hyperparameters are the variables that govern the training process and the topology of an ML model. These variables remain constant over the training process and directly impact the performance of your ML program. Hyperparameters are of two types:

1. **Model hyperparameters** which influence model selection such as the number and width of hidden layers
2. **Algorithm hyperparameters** which influence the speed and quality of the learning algorithm such as the learning rate for Stochastic Gradient Descent

(SGD) and the number of nearest neighbors for a k Nearest Neighbors (KNN) classifier

4.1.1 Hyperparameter tuning with Keras Tuner



First, a tuner is defined. Its role is to determine which hyperparameter combinations should be tested. The library search function performs the iteration loop, which evaluates a certain number of hyperparameter combinations. Evaluation is performed by computing the trained model's accuracy on a held-out validation set. Finally, the best hyperparameter combination in terms of validation accuracy can be tested on a held-out test set.

4.1.2 Search Space definition

To perform hyperparameter tuning, we need to define the search space, that is to say which hyperparameters need to be optimized and in what range. Here, there are already 2 hyperparameters that we can tune.

1 The number of filters for the dense layer.
2 The size of the kernel.

```
filters=hp.Int('conv_1_filter', min_value=32, max_value=128, step=16),
kernel_size=hp.Choice('conv_1_kernel', values = [3,5]),
activation='relu'
```

4.1.3 Architectures comparison

We can tune only one model, but we went for two models to tune to see if the complexity of the models is decreased. Can it give a better result with decreased training time?

A Model with 6 dense layer

```
def build_model(hp):
  model = keras.Sequential([
    keras.layers.Conv2D(
        filters=hp.Int('conv_1_filter', min_value=32, max_value=128, step=16),
        kernel_size=hp.Choice('conv_1_kernel', values = [3,5]),
        activation='relu',
        input_shape=(32,32,3)
    ),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Flatten(),
    keras.layers.Dense(
        units=hp.Int('dense_1_units', min_value=32, max_value=128, step=16),
        activation='relu'
    ),
    keras.layers.Dense(10, activation='softmax')
```

B Model with 5 dense layer (less complex)

```
def build_model(hp):
  model = keras.Sequential([
    keras.layers.Conv2D(
        filters=hp.Int('conv_1_filter', min_value=32, max_value=128, step=16),
        kernel_size=hp.Choice('conv_1_kernel', values = [3,5]),
        activation='relu',
        input_shape=(32,32,3)
    ),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(
        filters=hp.Int('conv_2_filter', min_value=32, max_value=64, step=16),
        kernel_size=hp.Choice('conv_2_kernel', values = [3,5]),
        activation='relu'
    ),
    keras.layers.Dropout(0.50),
    keras.layers.BatchNormalization(),
    keras.layers.Flatten(),
    keras.layers.Dense(
        units=hp.Int('dense_1_units', min_value=32, max_value=128, step=16),
        activation='relu'
    ),
    keras.layers.Dense(10, activation='softmax')
```

4.1.4 Summary

Model 1

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 80)	6080
conv2d_1 (Conv2D)	(None, 24, 24, 48)	96048
dropout (Dropout)	(None, 24, 24, 48)	0
batch_normalization (BatchNo	(None, 24, 24, 48)	192
conv2d_2 (Conv2D)	(None, 20, 20, 48)	57648
dropout_1 (Dropout)	(None, 20, 20, 48)	0
batch_normalization_1 (Batch	(None, 20, 20, 48)	192
conv2d_3 (Conv2D)	(None, 16, 16, 48)	57648
dropout_2 (Dropout)	(None, 16, 16, 48)	0
batch_normalization_2 (Batch	(None, 16, 16, 48)	192
conv2d_4 (Conv2D)	(None, 12, 12, 48)	57648
dropout_3 (Dropout)	(None, 12, 12, 48)	0
batch_normalization_3 (Batch	(None, 12, 12, 48)	192
flatten (Flatten)	(None, 6912)	0
dense (Dense)	(None, 96)	663648
dense_1 (Dense)	(None, 10)	970

Total params: 940,458
Trainable params: 940,074
Non-trainable params: 384

Model 2

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 112)	3136
conv2d_1 (Conv2D)	(None, 26, 26, 64)	179264
dropout (Dropout)	(None, 26, 26, 64)	0
batch_normalization (BatchNo	(None, 26, 26, 64)	256
conv2d_2 (Conv2D)	(None, 22, 22, 64)	102464
dropout_1 (Dropout)	(None, 22, 22, 64)	0
batch_normalization_1 (Batch	(None, 22, 22, 64)	256
conv2d_3 (Conv2D)	(None, 18, 18, 64)	102464
dropout_2 (Dropout)	(None, 18, 18, 64)	0
batch_normalization_2 (Batch	(None, 18, 18, 64)	256
flatten (Flatten)	(None, 20736)	0
dense (Dense)	(None, 96)	1990752
dense_1 (Dense)	(None, 10)	970

Total params: 2,379,818
Trainable params: 2,379,434
Non-trainable params: 384

4.1.5 Accuracy

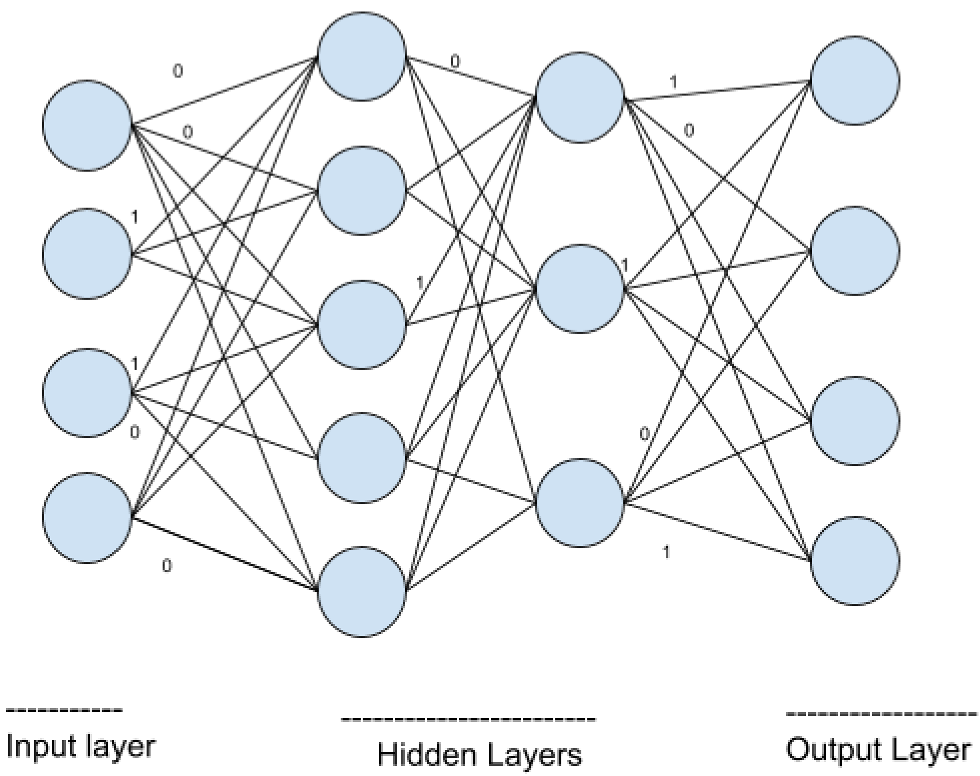
Model 1	64%
Model 2	76.44%

Here we see that a less complex model gave us a better result with decreased training time.

4.2 Transfer learning / Pre-trained models

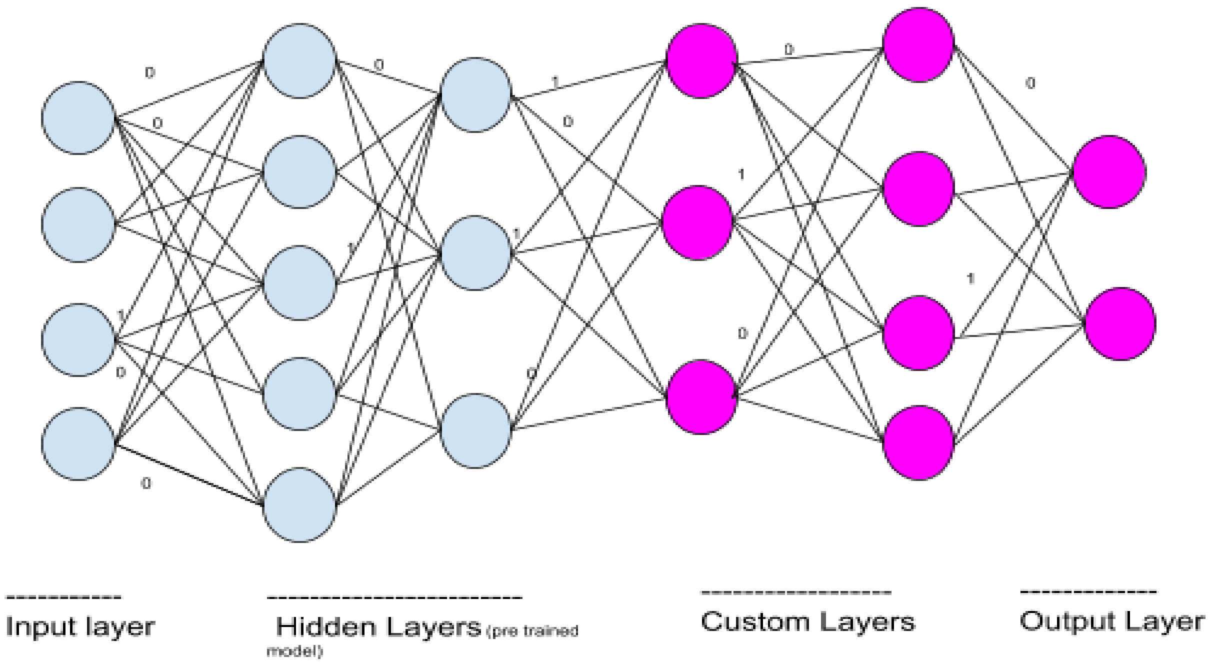
Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge (weights) gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cats could apply when trying to recognize dogs.

4.2.1 How Transfer Learning works



For simplicity weights are only 0 and 1 for demonstration
Assume all the weights are assigned.

Suppose the above diagram is a CNN model of a *state of the art* model, it has been trained and the weights are fixed.



For simplicity weights are only 0 and 1 for demonstration
Assume all the weights are assigned.

We can use the CNN model(state of art) from above. Remove the output layer of the state of art model, add our own custom layers and our output layer, preserving the weights of the previous layers.

4.2.2 Pre trained models used

A ResNet50
B VGG16
C InceptionV3

A ResNET 50

ResNet-50 is a convolutional neural network that is 50 layers deep. We can load a pre trained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.

B VGG16

VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR (Imagenet) competition in 2014. It 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. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output. The 16 in VGG16 refers to it as 16 layers that have weights. The network has an image input size of 224-by-224.

C Inception V3

Inception v3 is a convolutional neural network for assisting in image analysis and object detection, and got its start as a module for Googlenet. It is the third edition of Google's Inception Convolutional Neural Network, originally introduced during the ImageNet Recognition Challenge. Just as ImageNet can be thought of as a database of classified visual objects, Inception helps classification of objects in the world of computer vision. Inception-v3 is a convolutional neural network is 48 layers deep.The network has an image input size of 224-by-224.

4.2.3 Model Summary

A ResNet 50

```
==========================================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
```

The Resnet 50 model has over 23 million parameters.

B VGG16

```
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

The VGG 16 has over 14.7 million parameters.

C Inception V3

```
==================================================================================================
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
__________________________________________________________________________________________________
```

The Inception V3 has over 21.8 million parameters.

4.2.4 Model Architecture

```
model = models.Sequential()
model.add(layers.UpSampling2D((2,2)))
model.add(layers.UpSampling2D((2,2)))
model.add(layers.UpSampling2D((2,2)))
model.add(resnet_conv_base)
model.add(layers.Flatten())
model.add(layers.BatchNormalization())  # batchnormalization for speeind up the math
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer=optimizers.RMSprop(lr=2e-5), loss='binary_crossentropy', metrics=['acc'])

history = model.fit(x_train, y_train, epochs=5, batch_size=20, validation_split=0.2 )
```

Three line used for increasing the size of input image from 32 *32 to 256 * 256.

The pretrained model

Removing the output layer of pre trained model and added more layers

The model architecture remains the same for all the three models. Except for the line in *yellow*, we will set the pre trained model as resnet, vgg16 or inception.

4.2.5 Model Training

A ResNet 50

```
2000/2000 [==============================] - 562s 281ms/step - loss: 0.2558 - acc: 0.4280 - val_loss: 0.1360 - val_acc: 0.7620
Epoch 2/5
2000/2000 [==============================] - 563s 281ms/step - loss: 0.1754 - acc: 0.6522 - val_loss: 0.0860 - val_acc: 0.8567
Epoch 3/5
2000/2000 [==============================] - 562s 281ms/step - loss: 0.1351 - acc: 0.7617 - val_loss: 0.0672 - val_acc: 0.9014
Epoch 4/5
2000/2000 [==============================] - 562s 281ms/step - loss: 0.1094 - acc: 0.8220 - val_loss: 0.0501 - val_acc: 0.9190
Epoch 5/5
2000/2000 [==============================] - 562s 281ms/step - loss: 0.0903 - acc: 0.8652 - val_loss: 0.0430 - val_acc: 0.9297
```

B VGG16

```
2000/2000 [==============================] - 1851s 925ms/step - loss: 0.3354 - acc: 0.2200 - val_loss: 0.2679 - val_acc: 0.3552
Epoch 2/5
2000/2000 [==============================] - 1855s 928ms/step - loss: 0.2807 - acc: 0.3474 - val_loss: 0.2295 - val_acc: 0.4726
Epoch 3/5
2000/2000 [==============================] - 1854s 927ms/step - loss: 0.2528 - acc: 0.4218 - val_loss: 0.2198 - val_acc: 0.4940
Epoch 4/5
2000/2000 [==============================] - 1849s 925ms/step - loss: 0.2318 - acc: 0.4802 - val_loss: 0.2035 - val_acc: 0.5474
Epoch 5/5
2000/2000 [==============================] - 1851s 925ms/step - loss: 0.2151 - acc: 0.5280 - val_loss: 0.2007 - val_acc: 0.5505
```

C Inception V3

```
2000/2000 [==============================] - 1216s 608ms/step - loss: 0.2262 - acc: 0.5177 - val_loss: 0.0822 - val_acc: 0.8714
Epoch 2/5
2000/2000 [==============================] - 1214s 607ms/step - loss: 0.1347 - acc: 0.7740 - val_loss: 0.0494 - val_acc: 0.9271
Epoch 3/5
2000/2000 [==============================] - 1213s 607ms/step - loss: 0.1029 - acc: 0.8497 - val_loss: 0.0385 - val_acc: 0.9418
Epoch 4/5
2000/2000 [==============================] - 1214s 607ms/step - loss: 0.0823 - acc: 0.8910 - val_loss: 0.0353 - val_acc: 0.9456
Epoch 5/5
2000/2000 [==============================] - 1214s 607ms/step - loss: 0.0653 - acc: 0.9215 - val_loss: 0.0315 - val_acc: 0.9521
```

4.2.6 **Model Summary** (after adding more layers)

A ResNet50

```
==================================================================
Total params: 40,899,018
Trainable params: 40,583,370
Non-trainable params: 315,648
__________________________________________________________________
```

B VGG16

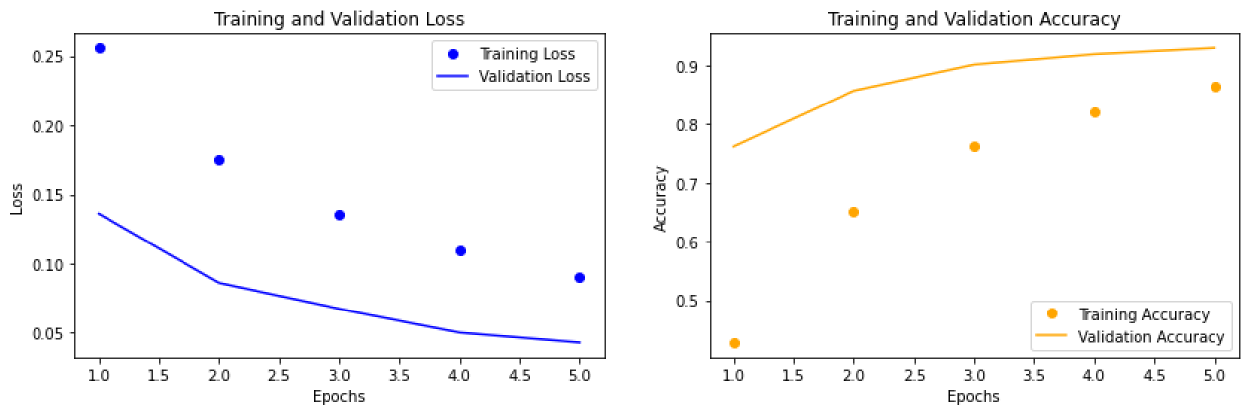
```
==================================================================
Total params: 19,049,866
Trainable params: 18,983,946
Non-trainable params: 65,920
__________________________________________________________________
```

C Inception V3

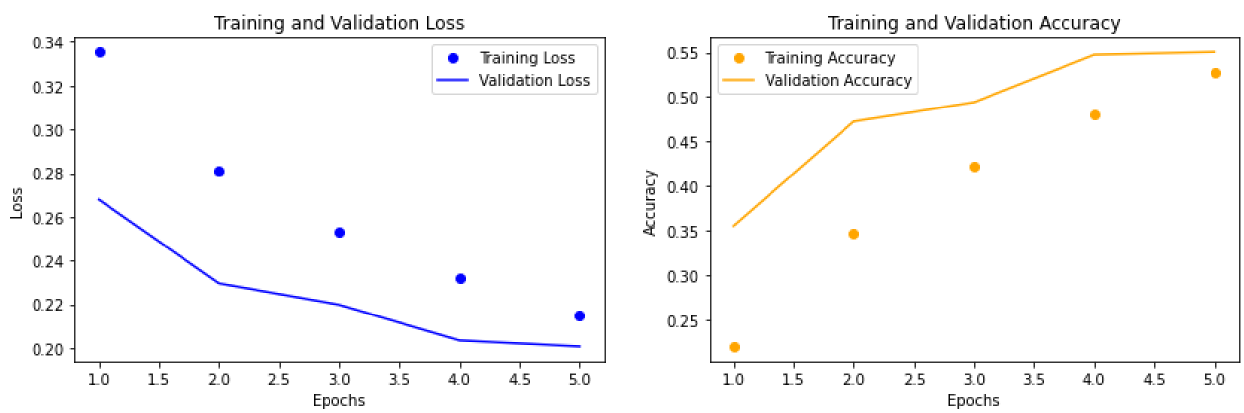
```
==================================================================
Total params: 31,544,682
Trainable params: 31,362,410
Non-trainable params: 182,272
__________________________________________________________________
```

4.2.7 Visualizing loss and accuracy

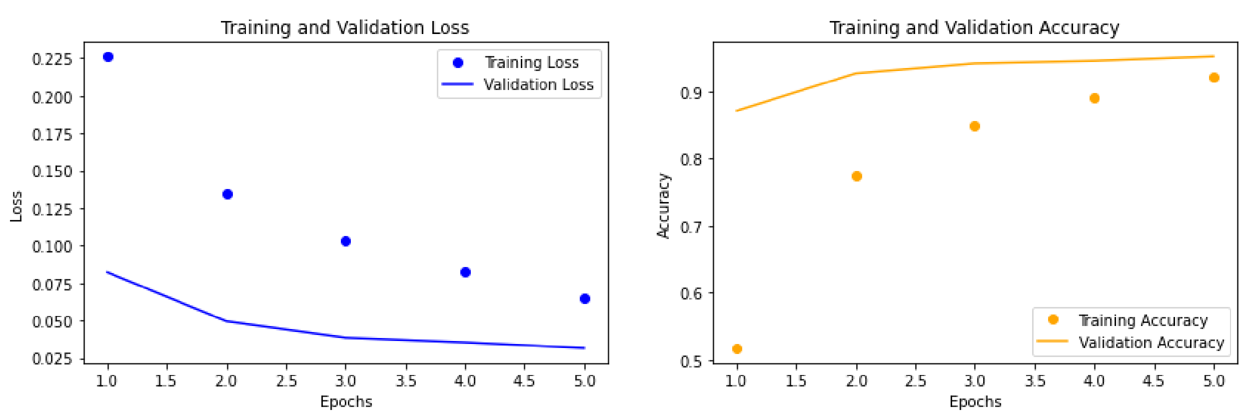
A ResNet50



B VGG16



C Inception V3



4.2.8 Summary

	Parameters (millions)	Parameters after adding more layer (millions)	Train time (approx hour, run on GPU)	Accuracy (%)
ResNet50	23.5	40.9	2.5 hrs	92.40
VGG16	14.7	19.05	2.5 hrs	55.24
Inception V3	21.8	31.5	1.75 hrs	95.10

We can see from the table above that the parameters for the models have increased after adding more layers to the pre trained models. We got great accuracy by using ResNet and Inception V3. But since the training time is less and accuracy is more for the Inception V3 model, we would prefer to use it.

Techniques used to avoid overfitting

To avoid the overfitting we have used dropout and validation data.

4.3 Image Generation

4.3.1 What is image generation?

Image generation means to manipulate an image example flipping, zooming, rotating etc.

Example

Here we can see an image in different forms.

4.3.2 Why do we use image generation

1 for training a small data, by augmentation we increase the number of images.
2 making a model more robust.

4.3.3 Architectures

1 A simple CNN with image generation.
2 Pre trained model with image generation.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape= (32, 32, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# the model so far outputs 3D feature maps (height, width, features)
```

```
model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

```
model = models.Sequential()
model.add(layers.UpSampling2D((2,2)))
model.add(layers.UpSampling2D((2,2)))
model.add(layers.UpSampling2D((2,2)))
model.add(inceptionv3_conv_base)
model.add(layers.Flatten())
model.add(layers.BatchNormalization())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(10, activation='softmax'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

4.3.4 Image generation used (same for both the models)

```
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    shear_range=0.2,
    zoom_range=0.2,)
```

featurewise_center	Boolean. Set input mean to 0 over the dataset, feature-wise.
featurewise_std_normalization	Boolean. Divide inputs by std of the dataset, feature-wise.
width_shift_range	Float, 1-D array-like or int ● float: fraction of total width, if < 1, or pixels if >= 1. ● 1-D array-like: random elements from the array. ● int: integer number of pixels from interval (-width_shift_range, +width_shift_range) ● With width_shift_range=2 possible values are integers [-1, 0, +1], same as with width_shift_range=[-1, 0, +1], while with width_shift_range=1.0 possible values are floats in the interval [-1.0, +1.0).
height_shift_range	Float, 1-D array-like or int float: fraction of total height, if < 1, or pixels if >= 1. 1-D array-like: random elements from the array.

	int: integer number of pixels from interval (-height_shift_range, +height_shift_range) With height_shift_range=2 possible values are integers [-1, 0, +1], same as with height_shift_range=[-1, 0, +1], while with height_shift_range=1.0 possible values are floats in the interval [-1.0, +1.0).
horizontal_flip	Boolean. Randomly flip inputs horizontally.
shear_range	Float. Shear Intensity (Shear angle in counter-clockwise direction in degrees)
zoom_range	Float or [lower, upper]. Range for random zoom. If a float, [lower, upper] = [1-zoom_range, 1+zoom_range].

4.3.5 Model Summary

Model 1

```
Epoch 1/5
1563/1562 [==============================] - 48s 31ms/step - loss: 0.2742 - accuracy: 0.9013 - val_loss: 1.6525 - val_accuracy: 0.8707
Epoch 2/5
1563/1562 [==============================] - 42s 27ms/step - loss: 0.2437 - accuracy: 0.9075 - val_loss: 1.8681 - val_accuracy: 0.8607
Epoch 3/5
1563/1562 [==============================] - 42s 27ms/step - loss: 0.2313 - accuracy: 0.9117 - val_loss: 1.6885 - val_accuracy: 0.8728
Epoch 4/5
1563/1562 [==============================] - 42s 27ms/step - loss: 0.2237 - accuracy: 0.9151 - val_loss: 1.7462 - val_accuracy: 0.8654
Epoch 5/5
1563/1562 [==============================] - 42s 27ms/step - loss: 0.2200 - accuracy: 0.9173 - val_loss: 2.1007 - val_accuracy: 0.8501
```

Model 2 (image gen with pre trained model)

```
Epoch 1/5
1563/1562 [==============================] - 1283s 821ms/step - loss: 0.3262 - accuracy: 0.1440 - val_loss: 2.7607 - val_accuracy: 0.1000
Epoch 2/5
1563/1562 [==============================] - 1277s 817ms/step - loss: 0.2741 - accuracy: 0.2870 - val_loss: 2.7593 - val_accuracy: 0.1000
Epoch 3/5
1563/1562 [==============================] - 1278s 818ms/step - loss: 0.2508 - accuracy: 0.3745 - val_loss: 0.5656 - val_accuracy: 0.0927
Epoch 4/5
1563/1562 [==============================] - 1279s 819ms/step - loss: 0.2338 - accuracy: 0.4437 - val_loss: 2.0864 - val_accuracy: 0.1051
Epoch 5/5
1563/1562 [==============================] - 1281s 819ms/step - loss: 0.2165 - accuracy: 0.5096 - val_loss: 1.8154 - val_accuracy: 0.0978
<tensorflow.python.keras.callbacks.History at 0x7f9a9b36cda0>
```

when we used image generation, we could see that the **model 2 has overfitted**. So, it can be concluded that image generation is good for small datasets.

5 Comparison (of all the above models)

Models		Accuracy(%)
Keras Tuner	With 6 CNN layers	64.00
	With 5 CNN layers	76.44
Transfer Learning	ResNet50	92.40
	VGG16	55.24
	Inception V3	95.10
Image Generation	Basic CNN	85.00
	With transfer learning	overfitted

The **best mode**l is **Inception V3 with custom layers**, which gives us an accuracy of 95%.

6 Future Work

In future we can add more images to our dataset and even try out new deep learning algorithms such as capsule neural networks and more.

7 Conclusion

The above models can be deployed by government authorities, superstore owners, vending machine owners and also in the medical industry for classification of images.

References

1 My Mentor Navaneesh Gangala
2 Springboard Course
3 https://keras-team.github.io/keras-tuner/

4
https://keras.io/api/applications/#:~:text=Keras%20Applications%20are%20deep%20learning,They%20are%20stored%20at%20~%2F.

5 https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Thank you