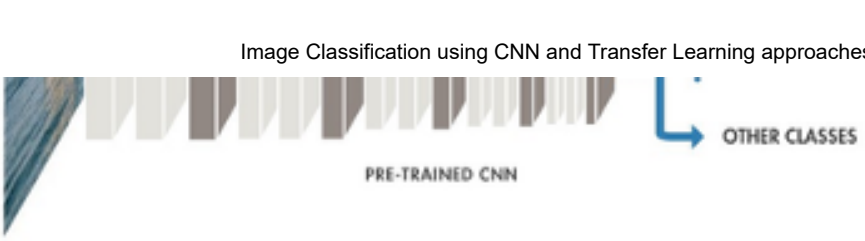
Image Classification using CNN and Transfer Learning approaches

Saket Garodia

Jan 8 · 8 min read

This blog illustrates deep learning using convolutional neural network and Transfer Learning approaches



Supermarkets around the world need to **cluster different fruits** to put them into their right racks and tag the right **prices.** It is not easy especially nowadays when there are a lot of varieties of each fruit which differ very slightly with each other that even the human eye can be confused.

To solve this problem, machine learning and deep learning techniques can be very useful as the development in these techniques especially in the field of neural networks has been immense in the last decade.

In this blog, I will explore deep learning approaches. The dataset and problem has been taken from Kaggle. Here's the link for the same:

https://www.kaggle.com/moltean/fruits/tasks

Approaches Explored:-

1) Building **convolutional layers** along with max-pooling

2) Using T**ransfer learning** approach from VGG16 to start with and then adding some extra convolutional layers along with max-pooling layers.

I used google Colab since it gives a free GPU and allows us to use the drive repository.

The dataset contains about 83000 images across 120 fruits.

I will download the huge dataset using kaggle API into drive and fetch them using Colab.

Let us start:

```
#unzipping training folder

!unzip -uq "/content/gdrive/My Drive/kaggle/Training.zip" -d
"/content/gdrive/My Drive/kaggle"
```

```
#unzipping test folder  !unzip -uq "/content/gdrive/My
Drive/kaggle/Test.zip" -d "/content/gdrive/My Drive/kaggle"

#import necessary libraries

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_files
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import array_to_img, img_to_array,
load_img
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D
from keras.layers import Activation, Dense, Flatten, Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint
from keras import backend as K
```

Now that I have imported all the necessary libraries, I will import the training and test data which are in different folders and them. The training and test folders contain 120 folders each with different fruit images in them.

```
# Loading data and putting them into training and test sets

#locations setting for training and test datasets
train_data='/content/gdrive/My Drive/kaggle/Training'
test_data='/content/gdrive/My Drive/kaggle/Test'

#creates X_train and Y_train using file_names and folders
def get_data(path):
    data = load_files(path)
    files = np.array(data['filenames'])
    targets = np.array(data['target'])
    target_labels = np.array(data['target_names'])
    return files,targets,target_labels

X_train, Y_train, labels = get_data(train_data)
X_test, Y_test,_ = get_data(test_data)
Y_train = np_utils.to_categorical(Y_train, 120)
Y_test = np_utils.to_categorical(Y_test, 120)
```

Now, let us split the dataset into training and validation sets.

```
# splitting train set into training and validation sets

X_train, X_val = train_test_split(X_train, test_size=0.2,
random_state=33)
Y_train, Y_val = train_test_split(Y_train, test_size=0.2,
random_state=33)
```

Now that we have created separate Xs and ys for training and test, let's load the images into array format (using pixel values).

```
#converting images into array to start computation

def convert_image_to_array(files):
    images_as_array=[]
    for file in files:
        images_as_array.append(img_to_array(load_img(file)))
    return images_as_array

X_train = np.array(convert_image_to_array(X_train))
X_val = np.array(convert_image_to_array(X_val))
X_test = np.array(convert_image_to_array(X_test))
```

We will normalize our inputs y using division by 255 since 255 is the maximum possible pixel values. Normalizing inputs helps neural networks run faster else it goes around like a ball in a bowl for long until it reaches the minimum objective point.

```
#nomalizing the pixel values before feeding into a neural network

X_train = X_train.astype('float32')/255
X_val = X_val.astype('float32')/255
X_test = X_test.astype('float32')/255
```

Approach 1 — Customized CNN

Now, let us start with our first approach that is using customized Convolutional Neural Networks. **CNNs** are amazing techniques that help a neural network learn **spatial and related features**. Before CNNs came, spatial information was tough to get learned into a neural network since all the data was fed in a flattened format. CNNs helps the neural

network to learn the relationships between various areas of an image like edges, eyes, etc. The **further deep the neural network goes, the more complex features** are learned.

Here, as approach 1, we will use 2 X 2 filters and increase the number of layers the deeper we go along with 2 X 2 max-pooling layer which chooses the maximum value at a certain area.

We will use the RELU activation function to remove linearity to learn complex features.

We will use **dropout** regularization which chooses a node using a probability that we will define and it will help prevent overfitting the model.

Finally, a **softmax** unit will be used to classify and find the loss function.

```
#Building model 1 using customized convolutional and pooling layers

model = Sequential()

#input_shape is 100*100 since thats the dimension of each of the
fruit images
model.add(Conv2D(filters = 16, kernel_size = 2,input_shape=
(100,100,3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters = 32,kernel_size = 2,activation=
'relu',padding='same'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters = 64,kernel_size = 2,activation=
'relu',padding='same'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters = 128,kernel_size = 2,activation=
'relu',padding='same'))
model.add(MaxPooling2D(pool_size=2))

# specifying parameters for fully connected layer
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(150))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(120,activation = 'softmax'))
model.summary()
```

Here's the structure of our model:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 100, 100, 16)      208
_________________________________________________________________
activation_1 (Activation)    (None, 100, 100, 16)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 50, 50, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 50, 50, 32)        2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 25, 25, 64)        8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 128)       32896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 150)               691350
_________________________________________________________________
activation_2 (Activation)    (None, 150)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_2 (Dense)              (None, 120)               18120
=================================================================
Total params: 752,910
Trainable params: 752,910
Non-trainable params: 0
_________________________________________________________________
```

In order to speed up our training, we will use **Adam optimizer** instead of Stochastic Gradient Descent. Adam takes advantage of both the momentum and rmsprop by leading us fast to the optimized values.

```
#importing ootimizers

from keras.optimizers import SGD, Adam, RMSprop

optimizer = Adam()
model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])
```

Its time to fit our model. We will use a **batch size of 128** and check for **20 epochs**.

```
# creating a file to save the trained CNN model
checkpointer = ModelCheckpoint(filepath =
'cnn_from_scratch_fruits.hdf5', verbose = 1, save_best_only = True)

# fitting model using above defined layers
CNN_model = model.fit(X_train,Y_train,
        batch_size = 128,
        epochs=20,
        validation_data=(X_val, Y_val),
        callbacks = [checkpointer],
        verbose=2, shuffle=True)
```

Showing last 4 epochs for illustration:

```
Epoch 00017: val_loss did not improve from 0.00005
Epoch 18/20
 - 9s - loss: 0.0164 - acc: 0.9944 - val_loss: 0.0017 - val_acc: 0.9995

Epoch 00018: val_loss did not improve from 0.00005
Epoch 19/20
 - 10s - loss: 0.0205 - acc: 0.9933 - val_loss: 0.0013 - val_acc: 0.9998

Epoch 00019: val_loss did not improve from 0.00005
Epoch 20/20
 - 9s - loss: 0.0204 - acc: 0.9939 - val_loss: 3.5900e-04 - val_acc: 0.9999

Epoch 00020: val_loss did not improve from 0.00005
```

```
#checking testset accuracy

score = model.evaluate(X_test, Y_test)
print('Test accuracy:', score[1])
```

```
20622/20622 [==============================] - 3s 137us/step
Test accuracy: 0.9941324798758607
```

Well, it gave an **amazing result with an accuracy of 0.995** which is tough even for a human eye. Just imagine the power of deep learning and neural networks. Neural

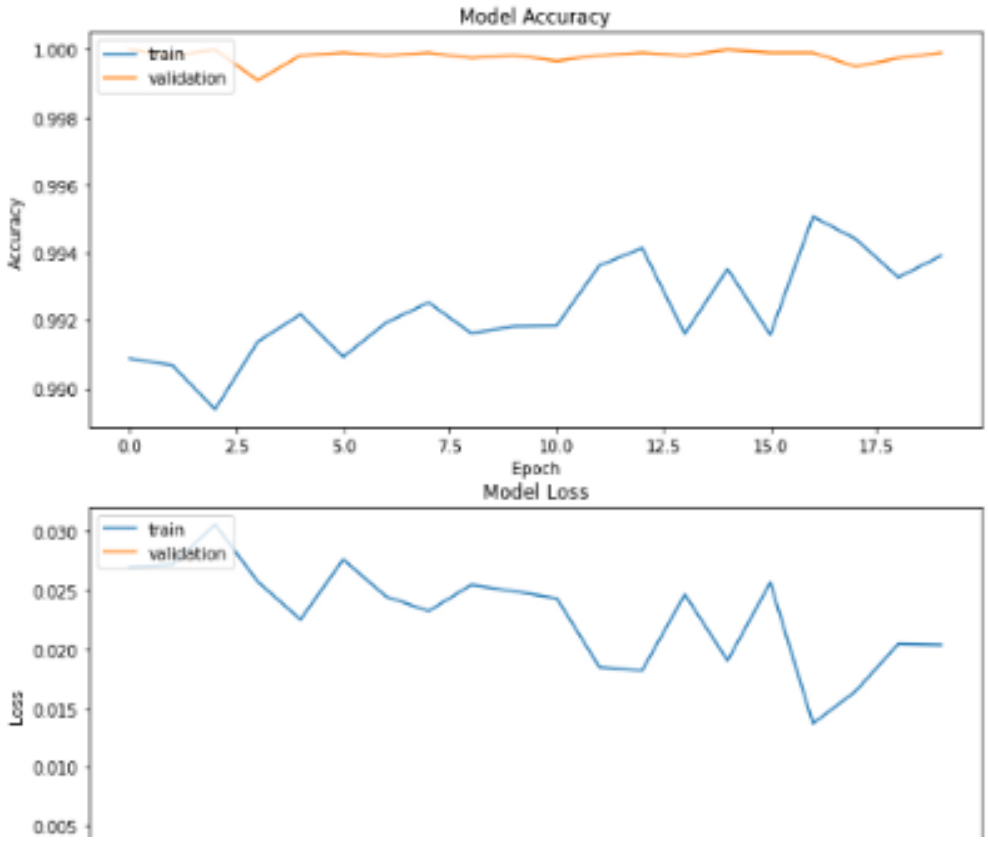
networks with convolutional layers are indeed magical.

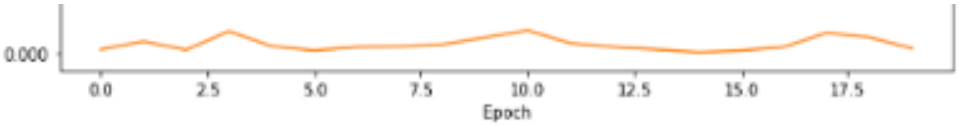
To gain some more excitement, let us visualize the names of the fruits it predicted along with the actual names and the images.

```
# using model to predict on test data
Y_pred = model.predict(X_test)

# Lets plot the predictions of different fruits and check their
original labels

fig = plt.figure(figsize=(20, 15))
for i, idx in enumerate(np.random.choice(X_test.shape[0], size=25,
replace=False)):
    ax = fig.add_subplot(5, 5, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(X_test[idx]))
    pred_idx = np.argmax(Y_pred[idx])
    true_idx = np.argmax(Y_test[idx])
    ax.set_title("{} ({})".format(labels[pred_idx],
labels[true_idx]),
                 color=("green" if pred_idx == true_idx else "red"))
```

Now, let us see how the loss function an accuracy changes as the model trains for 20 epochs

```
#plotting the loss function and accuracy for different epochs

plt.figure(1, figsize = (10, 10))
plt.subplot(211)
plt.plot(CNN_model.history['acc'])
plt.plot(CNN_model.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')

# plotting model loss
plt.subplot(212)
plt.plot(CNN_model.history['loss'])
plt.plot(CNN_model.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```





Approach 2: Using Transfer Learning for the base layer and adding more convolutional and pooling layers

In this approach, we will use transfer learning to prepare our base layer. **VGG16** is a neural network architecture that was trained on the **imagenet** dataset to classify 1000 different images and we will use the weights already trained on VGG16 for our approach 2.

```
#importing vgg16

#Part 2 using transfer learning

#importing vgg16 architecture which is trained on Imagenet

from keras.applications.vgg16 import VGG16

vgg_model = VGG16(input_shape=[100,100,3], weights='imagenet',
include_top=False)

#We will not train the layers imported.

for layer in vgg_model.layers:
   layer.trainable = False

#summary of the imported vgg model  vgg_model.summary()
```

Here's the summary of how vgg16 looks:

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 100, 100, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 100, 100, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 100, 100, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 50, 50, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 50, 50, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 50, 50, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 25, 25, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 25, 25, 256)       295168
```

```
block3_conv2 (Conv2D)        (None, 25, 25, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 25, 25, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 12, 12, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 12, 12, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 12, 12, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 12, 12, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 6, 6, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 3, 3, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

We will add one convolutional layer with 1024 filters to the vgg model followed by a max-pooling layer and dense layers and fit the model on the fruits dataset and follow the same procedure as above.

```
#adding some layers to the vgg_model imported and again fitting the
model to check the performance

transfer_learning_model = Sequential()

transfer_learning_model.add(vgg_model)


transfer_learning_model.add(Conv2D(1024, kernel_size=3,
padding='same'))

transfer_learning_model.add(Activation('relu'))

transfer_learning_model.add(MaxPooling2D(pool_size=(2, 2)))
transfer_learning_model.add(Dropout(0.3))

transfer_learning_model.add(Flatten())
transfer_learning_model.add(Dense(150))
transfer_learning_model.add(Activation('relu'))
transfer_learning_model.add(Dropout(0.4))
transfer_learning_model.add(Dense(120,activation = 'softmax'))
transfer_learning_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
vgg16 (Model)                (None, 3, 3, 512)         14714688
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 1024)        4719616
_________________________________________________________________
activation_3 (Activation)    (None, 3, 3, 1024)        0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 1, 1, 1024)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 1, 1, 1024)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 150)               153750
_________________________________________________________________
activation_4 (Activation)    (None, 150)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_4 (Dense)              (None, 120)               18120
=================================================================
Total params: 19,606,174
Trainable params: 4,891,486
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
from keras.optimizers import SGD, Adam, RMSprop

optimizer = Adam()
transfer_learning_model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])

#fitting the new model


checkpointer = ModelCheckpoint(filepath = 'transfer_learning.hdf5',
verbose = 1, save_best_only = True)

# running
transfer_learning_cnn = transfer_learning_model.fit(X_train,Y_train,
        batch_size = 128,
        epochs=20,
        validation_data=(X_val, Y_val),
        callbacks = [checkpointer],
        verbose=2, shuffle=True)
```

Last 4 epochs:

```
Epoch 00017: val_loss improved from 0.00073 to 0.00059, saving model to transfer_learning.hdf5
Epoch 18/20
 - 35s - loss: 0.0271 - acc: 0.9912 - val_loss: 6.7652e-04 - val_acc: 0.9999

Epoch 00018: val_loss did not improve from 0.00059
Epoch 19/20
 - 35s - loss: 0.0330 - acc: 0.9898 - val_loss: 3.7352e-04 - val_acc: 0.9999

Epoch 00019: val_loss improved from 0.00059 to 0.00037, saving model to transfer_learning.hdf5
Epoch 20/20
 - 35s - loss: 0.0295 - acc: 0.9905 - val_loss: 4.8138e-04 - val_acc: 0.9999
```

```
Epoch 00020: val_loss did not improve from 0.00037
```

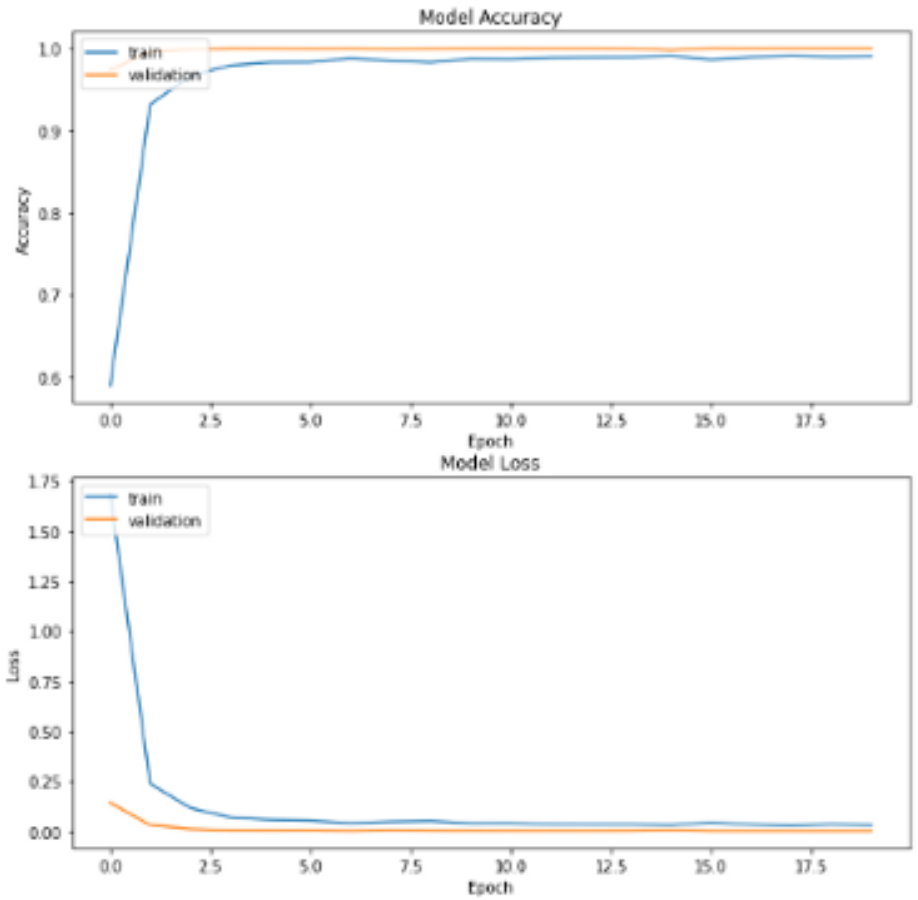
```
#score of the new model built using transfer learning

score = transfer_learning_model.evaluate(X_test, Y_test)
print('Test accuracy:', score[1])
```

```
20622/20622 [==============================] - 14s 670us/step
Test accuracy: 0.9765299194861009
```

We can see that the accuracy decreased a bit but just imagine we didn't even build a complex layer as we did in our 1st approach. We just used the weights from vgg16 and added 1 layer and even then the accuracy of ~0.98 is not bad.


```
#plotting curves for the transfer learning model

plt.figure(1, figsize = (10, 10))
```

```
plt.subplot(211)
plt.plot(transfer_learning_cnn.history['acc'])
plt.plot(transfer_learning_cnn.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')

# plotting model loss
plt.subplot(212)
plt.plot(transfer_learning_cnn.history['loss'])
plt.plot(transfer_learning_cnn.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```



What all more can be done to try improving accuracy in approach 1 if we have great computational power?

1) **Data Augmentation**:- We can increase our training set by using augmentation techniques like rotating images, cropping images, etc which can lead to a larger training set and hence may lead to better accuracy.

2) **More complex layers**: We can try building more complex layers like for example training all the vgg layers again which might lead to a better accuracy

3) **Hyperparameter tuning**: We can try using different regularization techniques suing a grid of parameters for momentum parameters, regularization parameters, rmsprop parameters, learning rates, etc which might lead to a better result.

4)**Increasing number of epochs**: We can run for more number of epochs trying different batch sizes.

I hope you gained some intuition on CNNs and transfer learning. Do clap if you like it.

Thank you and do post some feedback.

Machine Learning