Deep Learning – Project 3 (Part-B)

CLASSIFICATION OF DERMATOLOGY DISEASES	
Submitted To	Dr. Usman Sadiq
Submitted By	Hussam-Ul-Hussain
Roll No	18L-1827
Date	17-12-2018

PART 1

I tried the new variation of **AlexNet** model to train over the given dataset. And, I got very low training accuracy and validation accuracy. As you know, we have **class imbalance** problem in our dataset. So, to tackle this problem, I generated some images using **data augmentation**, for every class to make them equal to class 6.

It takes too much time to save augmented images from **Google Colab** to github or google drive. So, I used PyCharm to do data augmentation and saved new images in my local drive. And then I uploaded dataset, after augmentation, on google drive. Here is the code used for data augmentation:

aug.py

for im in imlist

```
22  labels = []
23  batch_size = 32
24  epochs = 10
25
26  datagen = ImageDataGenerator(
27          rotation_range=40,
28          width_shift_range=0.2,
29          height_shift_range=0.2,
30          shear_range=0.2,
31          zoom_range=0.2,
32          horizontal_flip=True,
33          fill_mode='nearest')
34  diseaseFilesDirectory='/'
35
36  # for i in range(7):
37  i = 7
38  path=os.getcwd()+diseaseFilesDirectory+str(i)+'/'
39  print(path)
40  allfiles=os.listdir(path)
41  imlist=[filename for filename in allfiles if filename[-4:] in [".jpg",".JPG"]]
42  for im in imlist:
43      imarr=np.array(Image.open(path+im),dtype=np.float)
44      imarr = imarr.reshape((1,)+imarr.shape)  # this is a Numpy array with shape (1, 3, 150, 150)
45      i = 0
46      for batch in datagen.flow(imarr, batch_size=1,save_to_dir=path+'GeneratedData', save_prefix='', save_format='jpg'):
47          i += 1
48          if i > 38:
49              break  # otherwise the generator would loop indefinitely
```

Figure 1 - Data Augmentation

Google is providing free GPU on Google Colab for research purposes. So, I used Python3 and GPU on Google Colab to train my model.

This is the model I used in Part 1 to classify dermatology diseases.

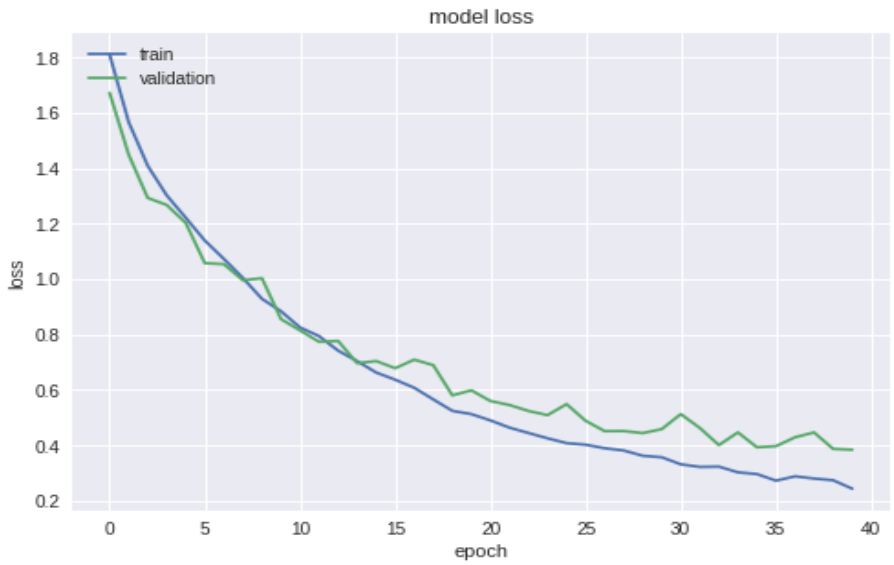
```
model = Sequential()
model.add(Conv2D(kernel_size=(3,3),filters=128,input_shape=(150, 150, 1),activation="relu",padding="valid"))
model.add(Conv2D(kernel_size=(3,3),filters=64,activation="relu",padding="same"))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(kernel_size=(3,3),filters=32,activation="relu",padding="same"))
model.add(Conv2D(kernel_size=(3,3),filters=32,activation="relu",padding="same"))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(kernel_size=(3,3),filters=16,activation="relu",padding="same"))
model.add(Conv2D(kernel_size=(3,3),filters=16,activation="relu",padding="same"))
model.add(Conv2D(kernel_size=(3,3),filters=8,activation="relu",padding="same"))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dropout(0.75))
model.add(Dense(100,activation="relu"))
model.add(Dense(7,activation='softmax'))
model.summary()

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(lr=0.0005),
              metrics=['accuracy'])

X = model.fit(data,labels,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          shuffle=True,
          validation_split=0.2)

plt.plot(X.history['loss'])
plt.plot(X.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

And my training accuracy is 91% and validation accuracy is 87%. Validation loss is 0.24 and validation loss is 0.38.



PART 2

For transfer learning I tried ResNet50 and GoogleNet (InceptionV3) models to train.

ResNet50:

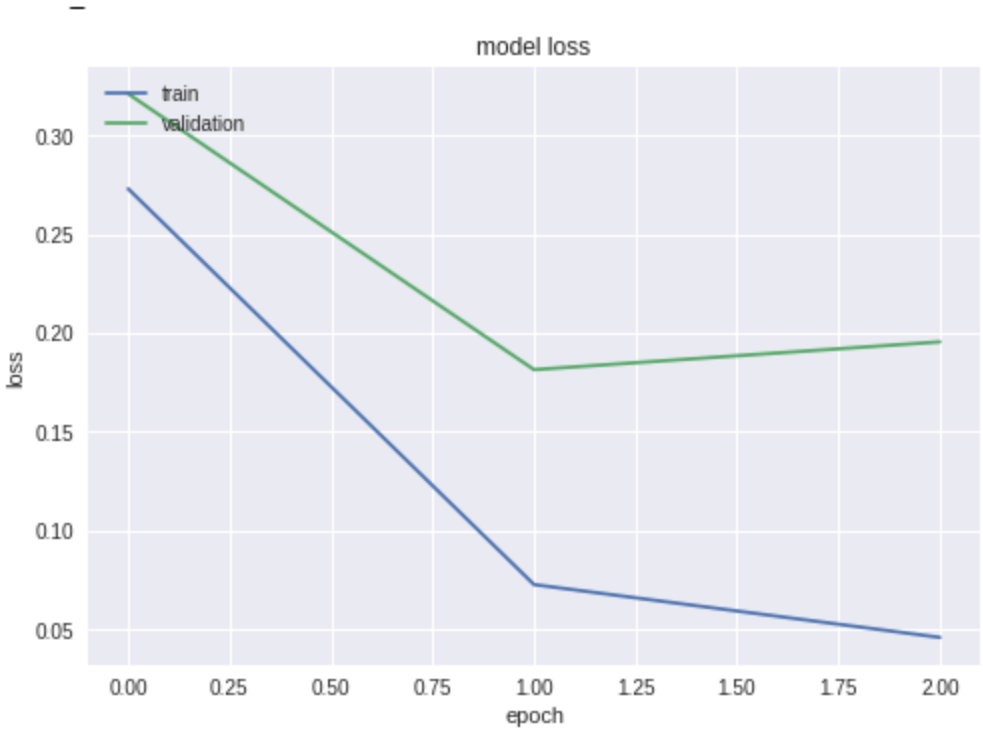
On ResNet model, the first 6 layers were fixed and all other layers were set to non-trainable. And **include_top** attribute was set to FALSE.

```
model = keras.applications.resnet50.ResNet50(include_top=False, weights='imagenet', input_shape=(150, 150,
 3))
```

```
for layer in model.layers[6:]:
    layer.trainable = False
```

After that, I added Flatten, Dropout and Dense Layer on the top of the model. And I got 98% training accuracy and 94% validation accuracy only on 3 epochs.

```
_________________________________________________________________________________________________
Total params: 28,708,519
Trainable params: 5,130,407
Non-trainable params: 23,578,112
_________________________________________________________________________________________________
Train on 7948 samples, validate on 3407 samples
Epoch 1/3
7948/7948 [==============================] - 103s 13ms/step - loss: 0.2731 - acc: 0.9190 - val_loss: 0.321
1 - val_acc: 0.9111
Epoch 2/3
7948/7948 [==============================] - 96s 12ms/step - loss: 0.0728 - acc: 0.9805 - val_loss: 0.1815
- val_acc: 0.9475
Epoch 3/3
7948/7948 [==============================] - 96s 12ms/step - loss: 0.0460 - acc: 0.9869 - val_loss: 0.1956
- val_acc: 0.9492
```



It seems like its overfitting in graph, but it isn't. Because, its training and validation accuracy, both started from 91%. So, within that range, its not overfitting the model.

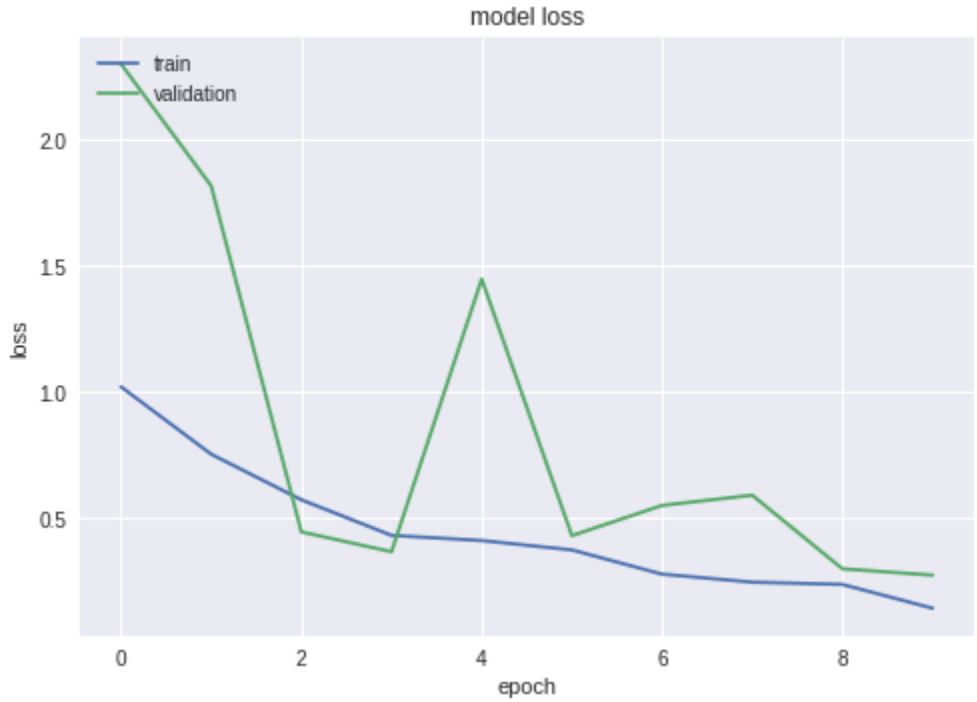
GoogleNet (Inception V3)

On GoogleNet, when I froze first 6 layers just like ResNet50 and made all other layers untrainable, I got very bad results. The validation accuracy wasn't going above 65%.
After that, I set only last 5 layers trainable and all previous layers non-trainable and my validation accuracy started to improve.

```
model = keras.applications.inception_v3.InceptionV3(include_top=False, weights='imagenet', input_shape=(15
0, 150, 3))
```

```
# Freeze the layers which you don't want to train. Here I am freezing the first 5 layers.
for layer in model.layers[:5]:
    layer.trainable = False
```

At the end, after training this model for 10 epochs, I got 95% training accuracy and 91.7% validation accuracy.



I still don't know why its oscillating so much but it kept decreasing the training and validation loss.

PART 3

DGAN

I got **dgan** code from github, which was designed for mnist dataset.
Here is the link to original code: https://github.com/eriklindernoren/Keras-GAN/blob/master/dcgan/dcgan.py

I modified this code for the given dataset. I am unable to change it for 3 channels. But it does generate similar images for given dataset in grayscale.

These are the images generated by the code. Because these images are of 28*28 size so some of them lost most the information.

image_3950.png