#In [1]: Calling tensorflow via keras

```
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense

vgg16 = VGG16(weights='imagenet')

fc2 = vgg16.get_layer('fc2').output
prediction = Dense(output_dim=1, activation='sigmoid', name='logit')(fc2)
model = Model(input=vgg16.input, output=prediction)
print(model.summary())
```

#In [2]: Freezing network

```
import pandas as pd

for layer in model.layers:
    if layer.name in ['fc1', 'fc2', 'logit']:
        continue
    layer.trainable = False

df = pd.DataFrame(([layer.name, layer.trainable] for layer in model.layers),
columns=['layer', 'trainable'])

df.style
```

#In [3]: Initializing Network optimizer

```
from keras.optimizers import SGD

sgd = SGD(lr=1e-4, momentum=0.9)
model.compile(optimizer=sgd, loss='binary_crossentropy',
metrics=['accuracy'])
```

#In [4]: Preprocessing for data augmentation

```
import numpy as np
from keras.preprocessing.image import ImageDataGenerator, array_to_img

def preprocess_input_vgg(x):

    from keras.applications.vgg16 import preprocess_input
    X = np.expand_dims(x, axis=0)
    X = preprocess_input(X)
    return X[0]
```

#In [5]: Training and Validation of the Network

```
train_datagen =
ImageDataGenerator(preprocessing_function=preprocess_input_vgg,
                                    rotation_range=40,
                                    width_shift_range=0.2,
                                    height_shift_range=0.2,
                                    shear_range=0.2,
                                    zoom_range=0.2,
                                    horizontal_flip=True,
                                    fill_mode='nearest')
train_generator =
train_datagen.flow_from_directory(directory='/home/zoro/Desktop/data/train',
                                                        target_size=[224, 224],
                                                        batch_size=16,
                                                        class_mode='binary')
```

```
validation_datagen =
ImageDataGenerator(preprocessing_function=preprocess_input_vgg)
validation_generator =
validation_datagen.flow_from_directory(directory='/home/zoro/Desktop/data/val
',

target_size=[224, 224],
                                                                batch_size=16,

class_mode='binary')
model.fit_generator(train_generator,
                    samples_per_epoch=32,
                    nb_epoch=16,
                    validation_data=validation_generator,
                    nb_val_samples=32);
```

#In [6]: Testing the Network

```
from IPython.display import display
import matplotlib.pyplot as plt

test_datagen =
ImageDataGenerator(preprocessing_function=preprocess_input_vgg)
test_generator =
test_datagen.flow_from_directory(directory='/home/zoro/Desktop/data/test',

target_size=[224, 224],
                                                                batch_size=470,

class_mode='binary')

X_val_sample, _ = next(test_generator)
y_pred = model.predict(X_val_sample)
```

#In [7]: Predicting output for 10 samples

```
nb_sample = 10
for x, y in zip(X_val_sample[:nb_sample], y_pred.flatten()[:nb_sample]):
    s = pd.Series({'Planes': 1-y, 'No_Planes': y})
    axes = s.plot(kind='bar')
    axes.set_xlabel('Class')
    axes.set_ylabel('Probability')
    axes.set_ylim([0, 1])
    plt.show()
    img = array_to_img(x)
    display(img)
```

#In [8]: Checking predicted results

```
abc =y_pred
print(len(abc))
```

#In [9]: Defining class labels from predicted output

```
tmp0=[x[0] for x in abc]
tmp1 =[1-x[0] for x in abc]
y_true_classes=test_generator.classes

y_predict_classes=np.zeros(shape=(470,1))
for i in range(0,len(tmp0)-1):
    if tmp0[i]<=0.5:
        y_predict_classes[i]=0
    else:
        y_predict_classes[i]=1
```

#In [10]: Creating confusion matrix

```
from sklearn.metrics import confusion_matrix
conf_arr=confusion_matrix(y_true_classes,y_predict_classes)
```

#In [11]: Plotting confusion matrix

```
norm_conf = []
for i in conf_arr:
    a = 0
    tmp_arr = []
    a = sum(i, 0)
    for j in i:
        tmp_arr.append(float(j)/float(a))
    norm_conf.append(tmp_arr)

fig = plt.figure()
plt.clf()
ax = fig.add_subplot(111)
ax.set_aspect(1)
res = ax.imshow(np.array(norm_conf), cmap=plt.cm.Spectral,
                interpolation='nearest')

width, height = conf_arr.shape

for x in xrange(width):
    for y in xrange(height):
        ax.annotate(str(conf_arr[x][y]), xy=(y, x),
                    horizontalalignment='center',
                    verticalalignment='center')
```

```
alphabet = '01'
plt.xticks(range(width), alphabet[:width])
plt.yticks(range(height), alphabet[:height])
plt.show()
```