Nepal Disaster Images

In [ ]:

Discussion

Looking at the heatmap, the model does not perform as well on 1s as on 0s considering on the diagonal, the 1s is darker. On the error plot, the column for class 0 is quite bright, which tells that many images get missclassified as 0s. The model performs better when only the last layer is fine-tuned with test set accuracy of 85.052% compared to 84.764% when more layers are fine-tuned. This is an indication that the pre-trained NasNet model is similar to our dataset as the higher layers represent more specific object features.

In [ ]:

In [ ]:

The dataset size is not a representative of how well the model performs. The highest test-set accuracies are not necessarily the largest dataset size

In [ ]:

In [ ]:

In [69]:
```
from __future__ import absolute_import, division, print_function, unicode_literals
```

In [70]:
```
import tensorflow as tf

tf.test.gpu_device_name()
```

Out[70]: ''

In [71]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:

In [ ]:

In [72]:
```
import os
```

In [73]:
```
from sklearn.metrics import confusion_matrix
```

In [ ]:

In [ ]:

In [ ]:

Load data

In [74]:
```
train_dir = 'data_nepal_eq/train'
validation_dir = 'data_nepal_eq/validation'
test_dir = 'data_nepal_eq/test'
```

In [ ]:

In [75]:
```
train_damaged_dir = os.path.join(train_dir, 'damaged')  # directory with our training damaged pictur
train_undamaged_dir = os.path.join(train_dir, 'undamaged')  # directory with our training undamaged
validation_damaged_dir = os.path.join(validation_dir, 'damaged')  # directory with our validation da
validation_undamaged_dir = os.path.join(validation_dir, 'undamaged')  # directory with our validatio
test_damaged_dir = os.path.join(test_dir, 'damaged')  # directory with our test damaged pictures
test_undamaged_dir = os.path.join(test_dir, 'undamaged')  # directory with our test undamaged pictur
```

In [ ]:
```
```

Understand the data

Let's look at how many damaged and undamaged images are in the training and validation directory:

In [76]:
```
num_damaged_tr = len(os.listdir(train_damaged_dir))
num_undamaged_tr = len(os.listdir(train_undamaged_dir))

num_damaged_val = len(os.listdir(validation_damaged_dir))
num_undamaged_val = len(os.listdir(validation_undamaged_dir))

num_damaged_ts = len(os.listdir(test_damaged_dir))
num_undamaged_ts = len(os.listdir(test_undamaged_dir))

total_train = num_damaged_tr + num_undamaged_tr
total_val = num_damaged_val + num_undamaged_val
total_test = num_damaged_ts + num_undamaged_ts
```

In [ ]:
```
```

In [77]:
```
print('total training damaged images:', num_damaged_tr)
print('total training undamaged images:', num_undamaged_tr)

print('total validation damaged images:', num_damaged_val)
print('total validation undamaged images:', num_undamaged_val)

print('total test damaged images:', num_damaged_ts)
print('total test undamaged images:', num_undamaged_ts)
print("--")
print("Total training images:", total_train)
print("Total validation images:", total_val)
print("Total test images:", total_test)
```

```
total training damaged images: 6711
total training undamaged images: 4752
total validation damaged images: 2237
total validation undamaged images: 1584
total test damaged images: 2236
total test undamaged images: 1584
--
Total training images: 11463
Total validation images: 3821
Total test images: 3820
```

In [ ]:

In [78]:
```
# set up variables
batch_size = 128
#epochs = 5
#IMG_HEIGHT = 150
#IMG_WIDTH = 150

# VGG19, VGG16, Inception
#IMG_HEIGHT = 224
#IMG_WIDTH = 224

# inceptionV3
#IMG_HEIGHT = 299
#IMG_WIDTH = 299


IMG_HEIGHT = 331
IMG_WIDTH = 331
```

In [ ]:
```

```

Data preparation

In [79]:
```
train_image_generator = ImageDataGenerator(rescale=1./255) # Generator for our training data
validation_image_generator = ImageDataGenerator(rescale=1./255) # Generator for our validation data
test_image_generator = ImageDataGenerator(rescale=1./255) # Generator for our test data
```

In [80]:
```
train_data_gen = train_image_generator.flow_from_directory(batch_size=batch_size,
                                                           directory=train_dir,
                                                           shuffle=True,
                                                           target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                           class_mode='binary')
```

```
Found 11463 images belonging to 2 classes.
```

In [81]:
```
val_data_gen = validation_image_generator.flow_from_directory(batch_size=batch_size,
                                                              directory=validation_dir,
                                                              target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                              class_mode='binary')
```

```
Found 3821 images belonging to 2 classes.
```

In [82]:
```
test_data_gen = test_image_generator.flow_from_directory(batch_size=batch_size,
                                                         directory=test_dir,
                                                         shuffle=True,
                                                         target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                         class_mode='binary')
```

```
Found 3820 images belonging to 2 classes.
```

In [ ]:

In [ ]:

Visualize training images

In [83]:
```
sample_training_images, _ = next(train_data_gen)
```

In [16]:
```
sample_training_images, labels = next(train_data_gen)
```

In [17]:
```
labels
```

Out[17]:
```
array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1.,
       1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1.,
       0., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0.,
       1., 0., 1., 1., 0., 0., 0., 0., 0., 1., 0., 1., 0., 1., 1., 0., 1.,
       1., 1., 1., 0., 1., 0., 1., 1., 1., 0., 0., 1., 0., 0., 1., 0., 1.,
       0., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1.,
       1., 1., 0., 0., 1., 0., 0., 1., 0.], dtype=float32)
```

In [18]:
```
sample_training_images, labels = next(train_data_gen)
```

In [19]:
```
labels
```

Out[19]:
```
array([1., 0., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0.,
       0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0.,
       1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 1., 0., 1., 1., 1., 1., 0.,
       0., 1., 0., 1., 1., 0., 0., 0., 0., 1., 0., 1., 0., 1., 1., 0., 0.,
       0., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0.,
       0., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 0., 0., 0., 1., 1., 0.,
       1., 1., 0., 0., 0., 1., 1., 0., 0.], dtype=float32)
```

In [ ]:
```
```

In [ ]:
```
```

In [20]:
```
# This function will plot images in the form of a grid with 1 row and 5 columns where images are pla
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

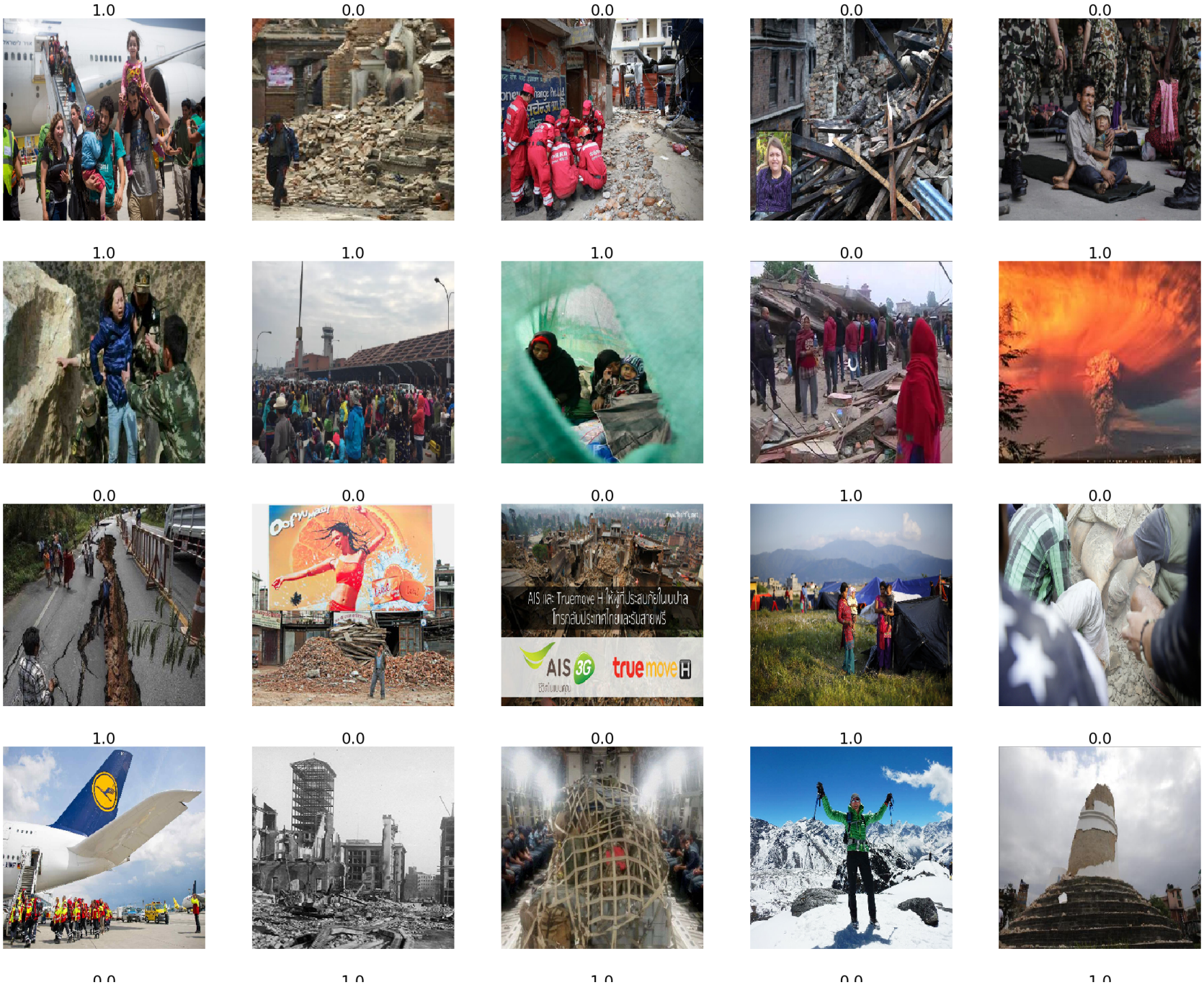
In [21]:
```
plotImages(sample_training_images[:5])
```

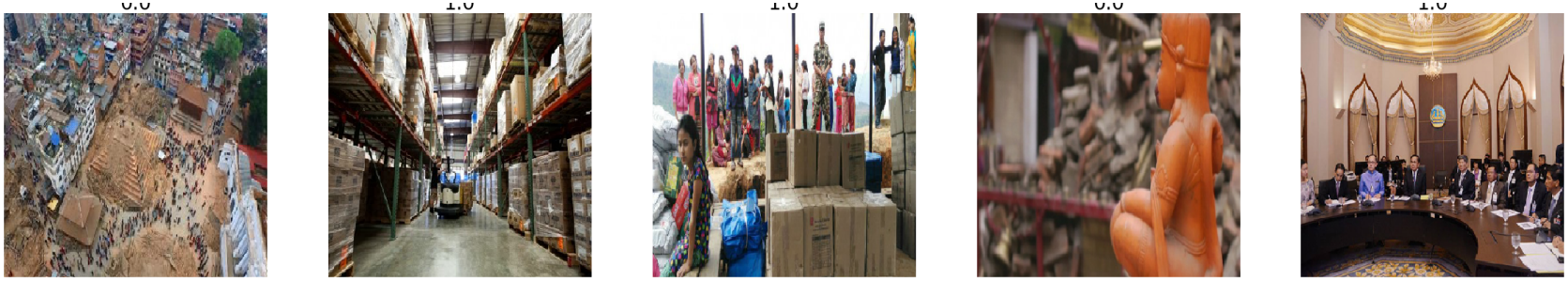
In [ ]:

In [ ]:

In [22]:
```
def show_batch(image_batch, label_batch):
  plt.figure(figsize=(40,40))
  for n in range(25):
      ax = plt.subplot(5,5,n+1)
      plt.imshow(image_batch[n])
      #plt.title(CLASS_NAMES[label_batch[n]==1][0].title())
      plt.title(label_batch[n], fontdict={'fontsize':28})
      plt.axis('off')
```

In [23]:
```
image_batch, label_batch = next(train_data_gen)
show_batch(image_batch, label_batch)
```





In [ ]:

In [24]:
```
#train_data_gen.classes[:50]
```

In [ ]:

In [25]:
```
label_batch
```

Out[25]:
```
array([1., 0., 0., 0., 0., 1., 1., 1., 0., 1., 0., 0., 0., 1., 0., 1., 0.,
       0., 1., 0., 0., 1., 1., 0., 1., 0., 0., 0., 0., 1., 1., 1., 0., 1.,
       1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0.,
       0., 0., 0., 1., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0.,
       0., 0., 0., 1., 1., 0., 0., 1., 0., 0., 1., 1., 0., 1., 0., 0., 1.,
       0., 0., 1., 0., 1., 1., 1., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0.,
       0., 1., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 0., 0., 1., 1., 0.,
       0., 1., 0., 0., 1., 1., 0., 0., 1.], dtype=float32)
```

In [ ]:

In [ ]:

NASNetLarge Pre-trained model

In [26]:
```
from keras.applications.nasnet import NASNetLarge
from keras.preprocessing import image
from keras.applications.nasnet import preprocess_input
```

```
Using TensorFlow backend.
```

In [ ]:

In [27]:
```
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
```

In [ ]:

In [ ]:

In [28]:
```
from keras.callbacks import EarlyStopping
```

In [ ]:

In [29]:
```
#callback = EarlyStopping(monitor='val_loss', patience=7)

callback = EarlyStopping(monitor='val_acc', patience=7, restore_best_weights=True)
```

In [ ]:

In [ ]:

In [30]:
```
base_model = NASNetLarge(weights='imagenet', include_top=False)
```

```
WARNING: Logging before flag parsing goes to stderr.
W1129 19:42:50.394306 140441164433216 deprecation_wrapper.py:119] From /home/demolakstate/anaconda
3/envs/dl/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:66: The name tf.get_defau
lt_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

W1129 19:42:50.395889 140441164433216 deprecation_wrapper.py:119] From /home/demolakstate/anaconda
3/envs/dl/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:541: The name tf.placehol
der is deprecated. Please use tf.compat.v1.placeholder instead.

W1129 19:42:50.403445 140441164433216 deprecation_wrapper.py:119] From /home/demolakstate/anaconda
3/envs/dl/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:4479: The name tf.truncat
ed_normal is deprecated. Please use tf.random.truncated_normal instead.

W1129 19:42:50.435287 140441164433216 deprecation_wrapper.py:119] From /home/demolakstate/anaconda
3/envs/dl/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:190: The name tf.get_defa
ult_session is deprecated. Please use tf.compat.v1.get_default_session instead.

W1129 19:42:50.436106 140441164433216 deprecation_wrapper.py:119] From /home/demolakstate/anaconda
3/envs/dl/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:197: The name tf.ConfigPr
oto is deprecated. Please use tf.compat.v1.ConfigProto instead.

W1129 19:42:50.507803 140441164433216 deprecation_wrapper.py:119] From /home/demolakstate/anaconda
3/envs/dl/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:2041: The name tf.nn.fuse
d_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.

W1129 19:42:50.886631 140441164433216 deprecation_wrapper.py:119] From /home/demolakstate/anaconda
3/envs/dl/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:4267: The name tf.nn.max_
pool is deprecated. Please use tf.nn.max_pool2d instead.

W1129 19:42:51.141192 140441164433216 deprecation_wrapper.py:119] From /home/demolakstate/anaconda
3/envs/dl/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:4271: The name tf.nn.avg_
pool is deprecated. Please use tf.nn.avg_pool2d instead.
```

In [ ]:
```

```

In [31]:
```
# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(2, activation='softmax')(x)

# this is the model we will train
model_NasNet = Model(inputs=base_model.input, outputs=predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

# compile the model (should be done *after* setting layers to non-trainable)
model_NasNet.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy', metrics=['accuracy
```

```
W1129 19:43:25.162454 140441164433216 deprecation_wrapper.py:119] From /home/demolakstate/anaconda
3/envs/dl/lib/python3.7/site-packages/keras/optimizers.py:793: The name tf.train.Optimizer is depre
cated. Please use tf.compat.v1.train.Optimizer instead.
```

In [32]:
```
history_NasNet = model_NasNet.fit_generator(
    train_data_gen,
    steps_per_epoch=total_train // batch_size,
    epochs=15,
    validation_data=val_data_gen,
    validation_steps=total_val // batch_size,
    #callbacks=[callback, cp_callback]
    callbacks=[callback]
)
```

```
W1129 19:43:25.258346 140441164433216 deprecation.py:323] From /home/demolakstate/anaconda3/envs/d
l/lib/python3.7/site-packages/tensorflow/python/ops/math_grad.py:1250: add_dispatch_support.<locals
>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future vers
ion.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where

Epoch 1/15

/home/demolakstate/anaconda3/envs/dl/lib/python3.7/site-packages/PIL/TiffImagePlugin.py:804: UserWa
rning: Corrupt EXIF data.  Expecting to read 4 bytes but only got 0.
  warnings.warn(str(msg))

89/89 [==============================] - 4816s 54s/step - loss: 0.7728 - acc: 0.7580 - val_loss:
0.3286 - val_acc: 0.8618
Epoch 2/15
89/89 [==============================] - 4790s 54s/step - loss: 0.4258 - acc: 0.8126 - val_loss:
0.5798 - val_acc: 0.7717
Epoch 3/15
89/89 [==============================] - 4772s 54s/step - loss: 0.3634 - acc: 0.8427 - val_loss:
0.5773 - val_acc: 0.7834
Epoch 4/15
89/89 [==============================] - 4717s 53s/step - loss: 0.3124 - acc: 0.8669 - val_loss:
0.6310 - val_acc: 0.7882
Epoch 5/15
89/89 [==============================] - 4722s 53s/step - loss: 0.2720 - acc: 0.8849 - val_loss:
0.5169 - val_acc: 0.8478
Epoch 6/15
89/89 [==============================] - 4745s 53s/step - loss: 0.2454 - acc: 0.8959 - val_loss:
0.4794 - val_acc: 0.8535
Epoch 7/15
89/89 [==============================] - 4779s 54s/step - loss: 0.2299 - acc: 0.9068 - val_loss:
0.4415 - val_acc: 0.8708
```

```
Epoch 8/15
89/89 [==============================] - 4729s 53s/step - loss: 0.1914 - acc: 0.9215 - val_loss:
0.9115 - val_acc: 0.7701
Epoch 9/15
89/89 [==============================] - 4733s 53s/step - loss: 0.1745 - acc: 0.9306 - val_loss:
0.5724 - val_acc: 0.8573
Epoch 10/15
89/89 [==============================] - 4736s 53s/step - loss: 0.1549 - acc: 0.9379 - val_loss:
0.5369 - val_acc: 0.8698
Epoch 11/15
89/89 [==============================] - 4773s 54s/step - loss: 0.1486 - acc: 0.9412 - val_loss:
1.0781 - val_acc: 0.7823
Epoch 12/15
89/89 [==============================] - 4865s 55s/step - loss: 0.1408 - acc: 0.9459 - val_loss:
1.4478 - val_acc: 0.7411
Epoch 13/15
89/89 [==============================] - 4864s 55s/step - loss: 0.1207 - acc: 0.9521 - val_loss:
0.8415 - val_acc: 0.8329
Epoch 14/15
89/89 [==============================] - 4829s 54s/step - loss: 0.1116 - acc: 0.9545 - val_loss:
0.8645 - val_acc: 0.8381
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [33]:
```
# Save the entire model to a HDF5 file
#model_NasNet.save('data_all/NasNet_model_ALL.h5')
```

In [ ]:

In [ ]:

In [34]:
```
#model_NasNet.load_weights('NasNet/NasNet_model.h5')

#new_model_NasNet = tf.keras.models.load_model('NasNet/NasNet_model.h5')
```

In [ ]:

In [ ]:

Visualize NasNetLarge model

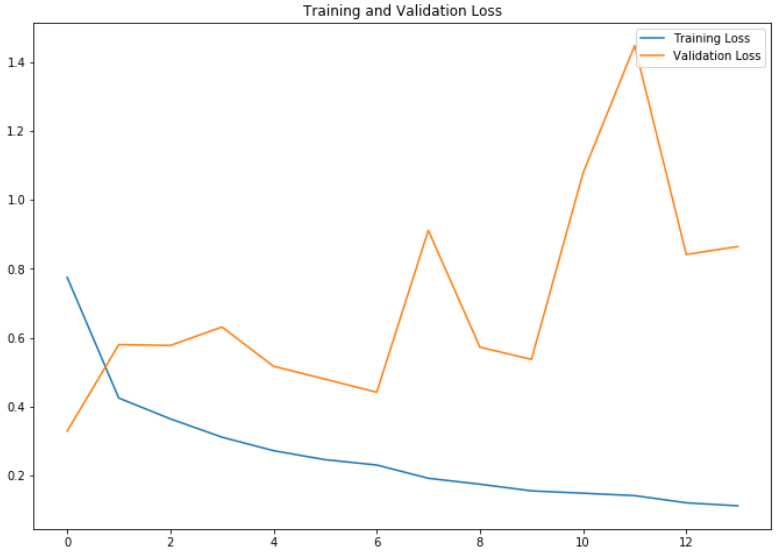
In [36]:
```
acc = history_NasNet.history['acc']
val_acc = history_NasNet.history['val_acc']

loss = history_NasNet.history['loss']
val_loss = history_NasNet.history['val_loss']

epochs_range = range(14)

plt.figure(figsize=(25, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



In [ ]:
```
```

In [ ]:

Evaluation of NasNet model accuracy on test data

Let's compare how the model performs on the test dataset

In [37]:
```
test_loss, test_acc = model_NasNet.evaluate_generator(test_data_gen, verbose=0)
```

In [38]:
```
print('\nTest accuracy:', test_acc)
```

```
Test accuracy: 0.8505235599597711
```

In [39]:
```
print('\nTest loss:', test_loss)
```

```
Test loss: 0.4843735194331064
```

In [ ]:

In [ ]:

Make predictions on test data

Let's make predictions on some images

In [40]:
```
test_data_gen.class_indices
```

Out[40]:
```
{'damaged': 0, 'undamaged': 1}
```

In [41]:
```
predictions = model_NasNet.predict_generator(test_data_gen)
```

In [42]:
```
true_labels = test_data_gen.classes
```

In [43]:
```
predictions[0]
```

Out[43]:
```
array([0.01543148, 0.9845685 ], dtype=float32)
```

In [44]:
```
np.argmax(predictions[0])
```

Out[44]:
```
1
```

In [45]:
```
test_data_gen.classes[0]
```

Out[45]:
```
0
```

In [46]:
```
predictions[-1]
```

Out[46]:
```
array([0.6558108, 0.3441893], dtype=float32)
```

In [47]:
```
np.argmax(predictions[-1])
```

Out[47]:
```
0
```

In [48]:
```
test_data_gen.classes[-1]
```

Out[48]:
```
1
```

In [ ]:
```

```

In [49]:
```
test_data_gen.classes
```

Out[49]:
```
array([0, 0, 0, ..., 1, 1, 1], dtype=int32)
```

In [ ]:
```

```

In [ ]:
```

```

Confusion Matrix

In [84]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:

In [ ]:

In [ ]:

In [50]:
```
y_true = true_labels
y_pred = np.array([np.argmax(x) for x in predictions])
```

In [51]:
```
y_pred
```

Out[51]:
```
array([1, 0, 0, ..., 0, 0, 0])
```

In [ ]:

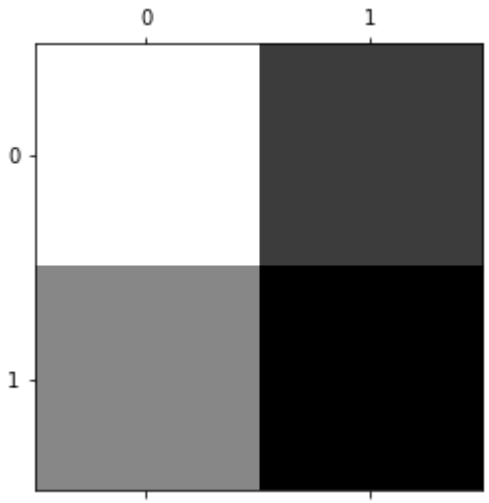
In [ ]:

In [52]:
```
cm = confusion_matrix(y_true, y_pred)
```

In [53]:
```
print(cm)
```

```
[[1470  766]
 [1035  549]]
```

In [85]:
```
plt.matshow(cm, cmap=plt.cm.gray)
plt.show()
```



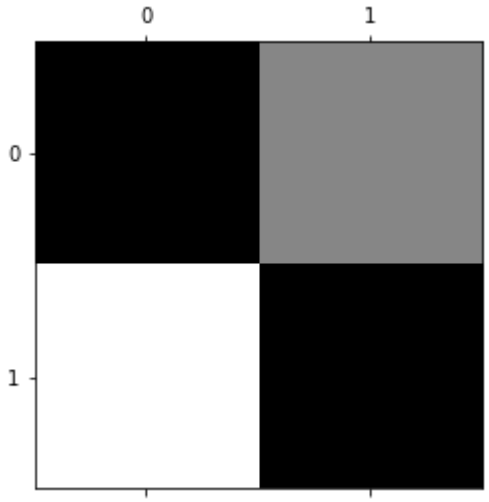
In [ ]:

Plot on Errors

In [86]:
```
row_sums = cm.sum(axis=1, keepdims=True)
norm_cm = cm / row_sums
```

In [ ]:

In [87]:
```
np.fill_diagonal(norm_cm, 0)
plt.matshow(norm_cm, cmap=plt.cm.gray)
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

Confusion Matrix Heat Map

In [54]:
```
#!pip install seaborn
```

In [55]:
```
#import seaborn as sb
```

In [56]:
```
#heat_map = sb.heatmap(cm, annot=True)
#sb.set(font_scale=1)
#plt.show()
```

Classification report of NasNet on test set

In [57]:
```
from sklearn.metrics import classification_report
```

In [58]:
```
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.59      0.66      0.62      2236
           1       0.42      0.35      0.38      1584

    accuracy                           0.53      3820
   macro avg       0.50      0.50      0.50      3820
weighted avg       0.52      0.53      0.52      3820
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Fine-tuning more layers in NASNetLarge

In [59]:
```
from keras.callbacks import EarlyStopping
```

In [60]:
```
#checkpoint_path = "NasNet/cp_2.ckpt"
#checkpoint_dir = os.path.dirname(checkpoint_path)
```

In [61]:
```
# Create a callback that save the model's weights
#cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path, save_weights_only=True,
```

In [ ]:

In [ ]:

In [62]:
```
#callback = EarlyStopping(monitor='val_loss', patience=7)

callback = EarlyStopping(monitor='val_acc', patience=7, restore_best_weights=True)
```

In [ ]:

In [63]:
```
for i, layer in enumerate(base_model.layers):
    print(i, layer.name)
```

```
18 activation_6
19 separable_conv_2_bn_reduction_left1_stem_1
20 separable_conv_2_bn_reduction_right1_stem_1
21 separable_conv_1_pad_reduction_right2_stem_1
22 activation_8
23 reduction_add_1_stem_1
24 separable_conv_1_reduction_right2_stem_1
25 separable_conv_1_pad_reduction_right3_stem_1
26 activation_10
27 separable_conv_1_bn_reduction_right2_stem_1
28 separable_conv_1_reduction_right3_stem_1
29 separable_conv_1_reduction_left4_stem_1
30 activation_7
31 separable_conv_1_bn_reduction_right3_stem_1
32 separable_conv_1_bn_reduction_left4_stem_1
33 reduction_pad_1_stem_1
34 separable_conv_2_reduction_right2_stem_1
35 activation_9
36 activation_11
37 reduction_left2_stem_1
```

In [ ]:

In [64]:
```
# we chose to train  blocks, i.e. we will freeze
# the first 249 layers and unfreeze the rest:
for layer in model_NasNet.layers[:1031]:
   layer.trainable = False
for layer in model_NasNet.layers[1031:]:
   layer.trainable = True
```

In [ ]:
```

```

In [65]:
```
# we need to recompile the model for these modifications to take effect
# we use SGD with a low learning rate
from keras.optimizers import SGD
model_NasNet.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='sparse_categorical_crossentropy',
```

In [66]:
```
# we train our model again (this time fine-tuning the top 2 inception blocks
# alongside the top Dense layers
history_NasNet_2 = model_NasNet.fit_generator(
    train_data_gen,
    steps_per_epoch=total_train // batch_size,
    epochs=10,
    callbacks=[callback],
    validation_data=val_data_gen,
    validation_steps=total_val // batch_size
)
```

```
Epoch 1/10
39/89 [============>.................] - ETA: 32:51 - loss: 0.1835 - acc: 0.9211

/home/demolakstate/anaconda3/envs/dl/lib/python3.7/site-packages/PIL/TiffImagePlugin.py:804: UserWa
rning: Corrupt EXIF data.  Expecting to read 4 bytes but only got 0.
  warnings.warn(str(msg))

89/89 [==============================] - 4811s 54s/step - loss: 0.1748 - acc: 0.9243 - val_loss: 0.
4849 - val_acc: 0.8610
Epoch 2/10
89/89 [==============================] - 4771s 54s/step - loss: 0.1620 - acc: 0.9309 - val_loss: 0.
5037 - val_acc: 0.8543
Epoch 3/10
89/89 [==============================] - 4775s 54s/step - loss: 0.1569 - acc: 0.9361 - val_loss: 0.
5166 - val_acc: 0.8527
Epoch 4/10
89/89 [==============================] - 4796s 54s/step - loss: 0.1529 - acc: 0.9383 - val_loss: 0.
5504 - val_acc: 0.8408
Epoch 5/10
89/89 [==============================] - 4810s 54s/step - loss: 0.1493 - acc: 0.9401 - val_loss: 0.
5271 - val_acc: 0.8432
Epoch 6/10
89/89 [==============================] - 4776s 54s/step - loss: 0.1468 - acc: 0.9411 - val_loss: 0.
5561 - val_acc: 0.8394
Epoch 7/10
89/89 [==============================] - 4753s 53s/step - loss: 0.1445 - acc: 0.9437 - val_loss: 0.
5541 - val_acc: 0.8416
Epoch 8/10
89/89 [==============================] - 4778s 54s/step - loss: 0.1464 - acc: 0.9428 - val_loss: 0.
5544 - val_acc: 0.8327
```

In [ ]:

```
```

In [ ]:

```
```

In [67]:

```
test_loss, test_acc = model_NasNet.evaluate_generator(test_data_gen, verbose=0)
```

In [68]:

```
print('\nTest accuracy:', test_acc)
```

```
Test accuracy: 0.8476439788703519
```

In [ ]:

```
```