FM9528A Banking Analytics

Coursework-3
Deep Learning

Student ID: 251139213

Word Count: 1850

Education and LiDAR Images

Financial inclusion is a method of offering banking and financial services to individuals who are mostly deprived in different areas and it is a key focus of modern regulatory efforts.

The LiDAR images can be used to provide an insight to the living conditions of a particular area and to have some idea of the various deprivation indexes. It is a technology that uses laser pulses to create 3D and elevations renderings of objects and terrains, similar to how radar technology uses sound.

As the LiDAR images provide the high-resolution images of an area so from the images the living conditions of the people in that area can be deciphered. This can also be helpful in knowing deprivation index of education which measures the lack of attainment and skills in the local population that are commonly measured for a geographical area.

The LiDAR images can show the presence or absence of schools or any skill development institute in the area. How the area is developed also gives the indication of the education/ skills of the people living there. The more organised dwellings with good infrastructure indicate better education index.

People living in different geographical location face challenges relating to their education and the challenges may vary due to geographic and demographic differences [1] between the places they reside and also depend on the factors like -

- racial and ethnic populations
- access to transportation
- availability of community resources
- educational funding
- general urbanization of an area

Students living in rural or urban areas, in poverty, or identifying as a minority, are significantly more likely to drop out of high school. Low-income areas and technology gap also affect the quality of education. (Emily Nunez, 2016)

From the LiDAR images, we expect to find similar relations. The images where there are smaller scattered houses can be the rural areas, the images where there will be constructions and medium sized houses can be sub-urban areas and the images where there are big houses can be the urban areas. The education level will wary for all, but it could be the case that urban/sub-urban areas have higher level of education as compared to some of the rural areas.

Neural Networks (VGG16 and ResNet50v2)

Using the LiDAR images as input and education index as output, we calibrate a neural network using two models from different families (ResNet and VGG) from the Keras Application Library.

The data is pre-processed and divided into train-validation-test sets with test set of 20% of data which is appropriate for the given size of input dataset of images.

We found the following datasets:

- Train: Found 23503 images
- Validation: Found 5875 images
- Test: Found 7345 images

We then train our VGG16 model and ResNet50v2 model with dense layers along with two additional layers and the whole model as trainable respectively for better performance as the last layers represent the high dimensional data, being closer to the original features of images.

VGG16 Model

The VGG 16 model is a classic 16-layer model in Deep Learning.

This model was trained over the ImageNet data, we leverage the already-trained weights, and adapt just the last few layers for our purposes.

We start by loading the VGG16 model and by first loading the model on-the-fly using the library. For an efficient storage of the model using a binary format we use h5py package.

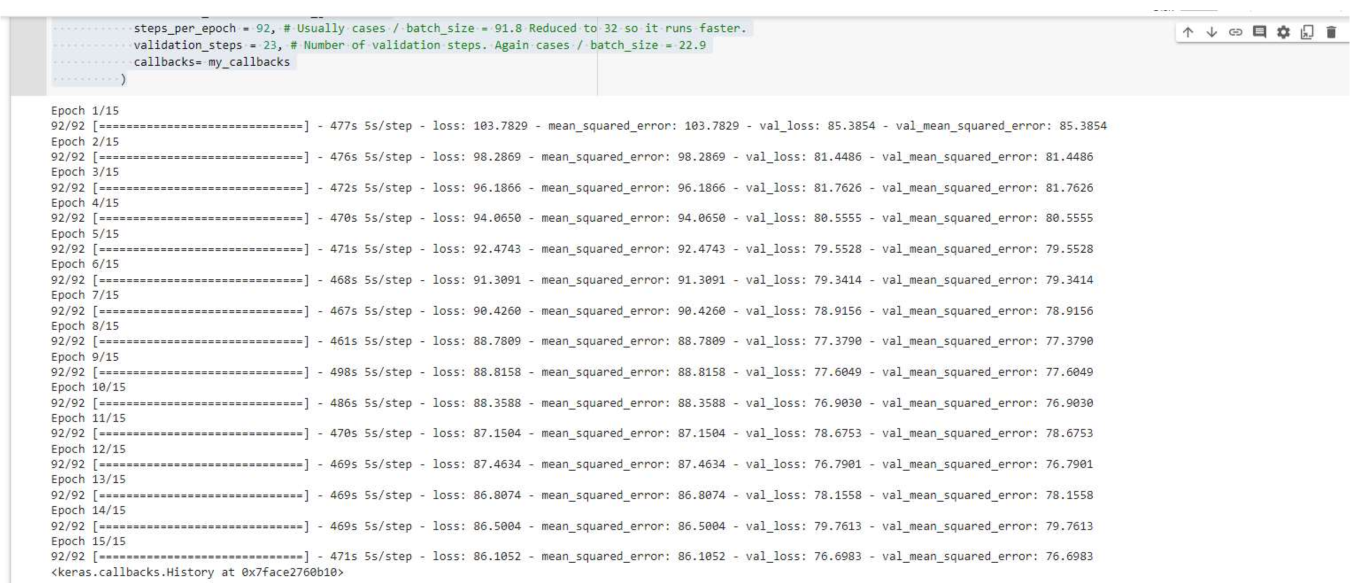
Model Architecture:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 128)               3211392

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 128)               16512

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 1)                 129

=================================================================
Total params: 17,942,721
Trainable params: 7,947,649
Non-trainable params: 9,995,072
_________________________________________________________________
```

Following below parameters are taken for training our model-

1) 3 dense layers with dropout are added for better performance
2) ReLU is used as the activation function because our index has all non-zero positive values.
3) Mean squared error is used as the regression loss
4) Adam optimizer is used with learning rate = 1e-5 and decay = 1e-3/200 for convergence
5) Rescale our image inputs for VGG16 model
6) Shear range = 0 as our images are appropriately clicked by satellite
7) Zoom range = 0.2, to capture 80-120% of closeup
8) Horizontal/Vertical Flip = True as the satellite images maybe flipped
9) Batch size = 256 for the K80 Tesla GPU
10) Use flow_from_dataframe for our data generator to process our input
11) Class_mode = raw, x_col = Images and y_col = Education index values for our regression problem
12) Callbacks for saving and controlling our model -
    Stop training if validation error stays within 0.00001 for three rounds
13) steps_per_epoch = 92 and validation_steps = 23 using formula as set_size/batch_size

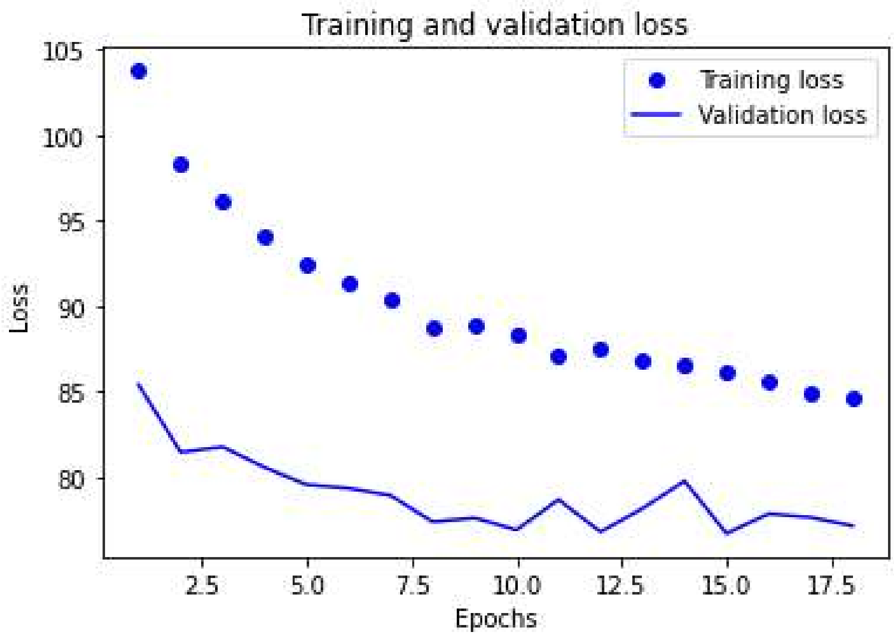
Model training results are as follows -



```
            steps_per_epoch = 92, # Usually cases / batch_size = 91.8 Reduced to 32 so it runs faster.
            validation_steps = 23, # Number of validation steps. Again cases / batch_size = 22.9
            callbacks= my_callbacks
          )

Epoch 1/15
92/92 [==============================] - 477s 5s/step - loss: 103.7829 - mean_squared_error: 103.7829 - val_loss: 85.3854 - val_mean_squared_error: 85.3854
Epoch 2/15
92/92 [==============================] - 476s 5s/step - loss: 98.2869 - mean_squared_error: 98.2869 - val_loss: 81.4486 - val_mean_squared_error: 81.4486
Epoch 3/15
92/92 [==============================] - 472s 5s/step - loss: 96.1866 - mean_squared_error: 96.1866 - val_loss: 81.7626 - val_mean_squared_error: 81.7626
Epoch 4/15
92/92 [==============================] - 470s 5s/step - loss: 94.0650 - mean_squared_error: 94.0650 - val_loss: 80.5555 - val_mean_squared_error: 80.5555
Epoch 5/15
92/92 [==============================] - 471s 5s/step - loss: 92.4743 - mean_squared_error: 92.4743 - val_loss: 79.5528 - val_mean_squared_error: 79.5528
Epoch 6/15
92/92 [==============================] - 468s 5s/step - loss: 91.3091 - mean_squared_error: 91.3091 - val_loss: 79.3414 - val_mean_squared_error: 79.3414
Epoch 7/15
92/92 [==============================] - 467s 5s/step - loss: 90.4260 - mean_squared_error: 90.4260 - val_loss: 78.9156 - val_mean_squared_error: 78.9156
Epoch 8/15
92/92 [==============================] - 461s 5s/step - loss: 88.7809 - mean_squared_error: 88.7809 - val_loss: 77.3790 - val_mean_squared_error: 77.3790
Epoch 9/15
92/92 [==============================] - 498s 5s/step - loss: 88.8158 - mean_squared_error: 88.8158 - val_loss: 77.6049 - val_mean_squared_error: 77.6049
Epoch 10/15
92/92 [==============================] - 486s 5s/step - loss: 88.3588 - mean_squared_error: 88.3588 - val_loss: 76.9030 - val_mean_squared_error: 76.9030
Epoch 11/15
92/92 [==============================] - 470s 5s/step - loss: 87.1504 - mean_squared_error: 87.1504 - val_loss: 78.6753 - val_mean_squared_error: 78.6753
Epoch 12/15
92/92 [==============================] - 469s 5s/step - loss: 87.4634 - mean_squared_error: 87.4634 - val_loss: 76.7901 - val_mean_squared_error: 76.7901
Epoch 13/15
92/92 [==============================] - 469s 5s/step - loss: 86.8074 - mean_squared_error: 86.8074 - val_loss: 78.1558 - val_mean_squared_error: 78.1558
Epoch 14/15
92/92 [==============================] - 469s 5s/step - loss: 86.5004 - mean_squared_error: 86.5004 - val_loss: 79.7613 - val_mean_squared_error: 79.7613
Epoch 15/15
92/92 [==============================] - 471s 5s/step - loss: 86.1052 - mean_squared_error: 86.1052 - val_loss: 76.6983 - val_mean_squared_error: 76.6983
<keras.callbacks.History at 0x7face2760b10>
```

```
# Train!
CBModel.fit(
            train_generator,
            epochs=epochs,
            validation_data=validation_generator,
            steps_per_epoch = 92, # Usually cases / batch_size = 91.8 Reduced to 32 so it runs faster.
            validation_steps = 23, # Number of validation steps. Again cases / batch_size = 22.9
            callbacks= my_callbacks
          )
```

```
Epoch 1/8
92/92 [==============================] - 502s 5s/step - loss: 85.5925 - mean_squared_error: 85.5925 - val_loss: 77.8492 - val_mean_squared_error: 77.8492
Epoch 2/8
92/92 [==============================] - 501s 5s/step - loss: 84.8572 - mean_squared_error: 84.8572 - val_loss: 77.6420 - val_mean_squared_error: 77.6420
Epoch 3/8
92/92 [==============================] - 484s 5s/step - loss: 84.6007 - mean_squared_error: 84.6007 - val_loss: 77.1460 - val_mean_squared_error: 77.1460
Epoch 4/8
```



We stop further training as the training and validation losses seem to converge and remain nearly constant.

Mean Square errors for VGG16:

- Training Set - 86.1052
- Validation Set - 76.6983
- Test Set - 134.588

ResNet50v2 Model

ResNet is one of the most powerful 50-layer deep neural networks model.

We first set the base model (ResNet) to non-trainable (freeze weights), then add the new models. Only top layers are trained for convergence and then we unfreeze the weights to fine-tune. This helps us to adjust the weights for better performance.

Following below parameters are taken for training our model-

1) 3 dense layers with dropout are added for better performance
2) ReLU is used as the activation function because our index has all non-zero positive values.
3) Mean squared error is used as the regression loss
4) Adam optimizer is used with learning rate = 1e-5 and decay = 1e-3/200 for convergence
5) No need to rescale our image inputs for ResNet model as preprocess function handles it
6) Shear range = 0 as our images are appropriately clicked by satellite
7) Zoom range = 0.2, to capture 80-120% of closeup
8) Horizontal/Vertical Flip = True as the satellite images maybe flipped
9) Batch size = 128 for the K80 Tesla GPU
10) Use flow_from_dataframe for our data generator to process our input
11) Class_mode = raw, x_col = Images and y_col = Education index values for our regression problem
12) Callbacks for saving and controlling our model -
    Stop training if validation error stays within 0.00001 for three rounds
13) steps_per_epoch = 184 and validation_steps = 46 using formula as set_size/batch_size

Model Architecture:

```
[ ] base_model.summary()

    Model: "resnet50v2"
    __________________________________________________________________________________________________
     Layer (type)                   Output Shape         Param #     Connected to
    ==================================================================================================
     input_5 (InputLayer)           [(None, 224, 224, 3  0           []
                                    )]

     conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0           ['input_5[0][0]']

     conv1_conv (Conv2D)            (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                                    )

     pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0           ['conv1_conv[0][0]']
                                    )

     pool1_pool (MaxPooling2D)      (None, 56, 56, 64)   0           ['pool1_pad[0][0]']

     conv2_block1_preact_bn (BatchN  (None, 56, 56, 64)  256         ['pool1_pool[0][0]']
     ormalization)

     conv2_block1_preact_relu (Acti  (None, 56, 56, 64)  0           ['conv2_block1_preact_bn[0][0]']
     vation)

     conv2_block1_1_conv (Conv2D)   (None, 56, 56, 64)   4096        ['conv2_block1_preact_relu[0][0]'
                                                                     ]

     conv2_block1_1_bn (BatchNormal  (None, 56, 56, 64)  256         ['conv2_block1_1_conv[0][0]']
     ization)

     conv2_block1_1_relu (Activatio  (None, 56, 56, 64)  0           ['conv2_block1_1_bn[0][0]']
     n)

     conv2_block1_2_pad (ZeroPaddin  (None, 58, 58, 64)  0           ['conv2_block1_1_relu[0][0]']
     g2D)

     conv2_block1_2_conv (Conv2D)   (None, 56, 56, 64)   36864       ['conv2_block1_2_pad[0][0]']

     conv2_block1_2_bn (BatchNormal  (None, 56, 56, 64)  256         ['conv2_block1_2_conv[0][0]']
     ization)

     conv2_block1_2_relu (Activatio  (None, 56, 56, 64)  0           ['conv2_block1_2_bn[0][0]']
     n)

     conv2_block1_0_conv (Conv2D)   (None, 56, 56, 256)  16640       ['conv2_block1_preact_relu[0][0]'
                                                                     ]

     conv2_block1_3_conv (Conv2D)   (None, 56, 56, 256)  16640       ['conv2_block1_2_relu[0][0]']

     conv2_block1_out (Add)         (None, 56, 56, 256)  0           ['conv2_block1_0_conv[0][0]',
                                                                      'conv2_block1_3_conv[0][0]']

     conv2_block2_preact_bn (BatchN  (None, 56, 56, 256)  1024       ['conv2_block1_out[0][0]']
     ormalization)

     conv2_block2_preact_relu (Acti  (None, 56, 56, 256)  0          ['conv2_block2_preact_bn[0][0]']
     vation)

     conv2_block2_1_conv (Conv2D)   (None, 56, 56, 64)   16384       ['conv2_block2_preact_relu[0][0]'

[ ] # Set the layers
```

```
ResModel.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 image_only_input (InputLaye  [(None, 224, 224, 3)]    0
 r)

 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800

 flatten (Flatten)           (None, 100352)            0

 dense (Dense)               (None, 64)                6422592

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 64)                4160

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 29,991,617
Trainable params: 29,946,177
Non-trainable params: 45,440
_________________________________________________________________
```

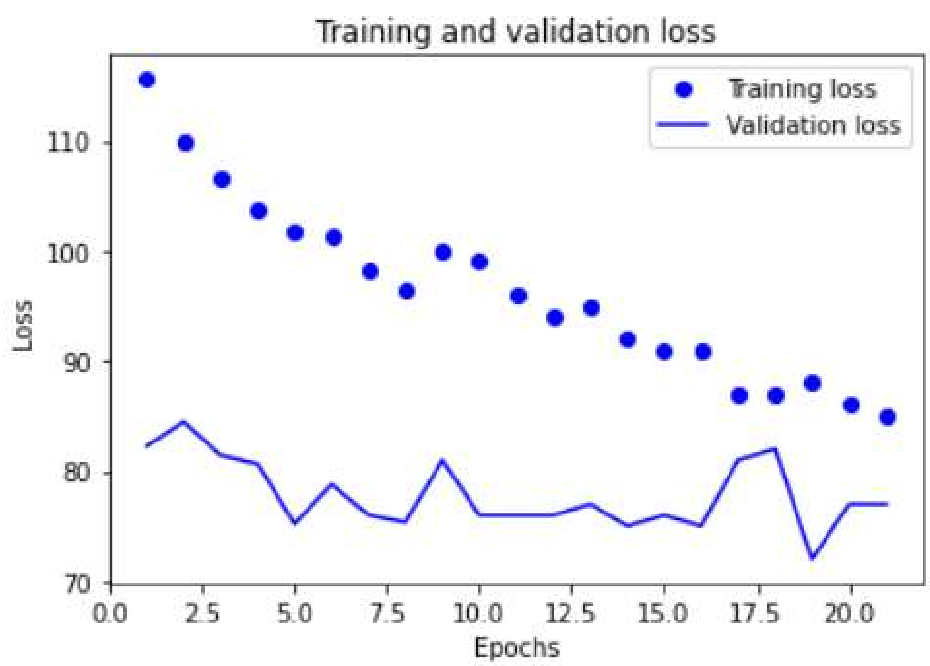
Training results for ResNet50v2 model are as follows-

```
# Number of epochs
epochs = 10

# Train!
ResModel.fit(
                 train_generator,
                 epochs=epochs,
                 validation_data=validation_generator,
                 steps_per_epoch = 184, # Usually cases / batch_size = 10. = 23503/128 = 183.61
                 validation_steps = 46, # Number of validation steps. Again cases / batch_size = 3. = 5875/128 = 45.89
                 callbacks = my_callbacks3

                 )
```

```
Epoch 1/10
184/184 [==============================] - 629s 3s/step - loss: 115.5860 - mean_squared_error: 115.5860 - val_loss: 82.2503 - val_mean_squared_error: 82.2503
Epoch 2/10
184/184 [==============================] - 612s 3s/step - loss: 109.8603 - mean_squared_error: 109.8603 - val_loss: 84.4798 - val_mean_squared_error: 84.4798
Epoch 3/10
184/184 [==============================] - 613s 3s/step - loss: 106.6959 - mean_squared_error: 106.6959 - val_loss: 81.4295 - val_mean_squared_error: 81.4295
Epoch 4/10
184/184 [==============================] - 614s 3s/step - loss: 103.7590 - mean_squared_error: 103.7590 - val_loss: 80.6529 - val_mean_squared_error: 80.6529
Epoch 5/10
184/184 [==============================] - 613s 3s/step - loss: 101.7202 - mean_squared_error: 101.7202 - val_loss: 75.2102 - val_mean_squared_error: 75.2102
Epoch 6/10
184/184 [==============================] - 610s 3s/step - loss: 101.3763 - mean_squared_error: 101.3763 - val_loss: 78.7994 - val_mean_squared_error: 78.7994
Epoch 7/10
184/184 [==============================] - 610s 3s/step - loss: 98.1242 - mean_squared_error: 98.1242 - val_loss: 76.0074 - val_mean_squared_error: 76.0074
Epoch 8/10
184/184 [==============================] - 609s 3s/step - loss: 96.5556 - mean_squared_error: 96.5556 - val_loss: 75.3444 - val_mean_squared_error: 75.3444
<keras.callbacks.History at 0x7f01b19d8d50>
```

```
                  )
```

```
Epoch 1/15
184/184 [==============================] - 636s 3s/step - loss: 100.5641 - mean_squared_error: 100.5641 - val_loss: 81.6307 - val_mean_squared_error: 81.6307
Epoch 2/15
184/184 [==============================] - 570s 3s/step - loss: 99.6896 - mean_squared_error: 99.6896 - val_loss: 76.9703 - val_mean_squared_error: 76.9703
Epoch 3/15
184/184 [==============================] - 565s 3s/step - loss: 96.4278 - mean_squared_error: 96.4278 - val_loss: 76.3962 - val_mean_squared_error: 76.3962
Epoch 4/15
184/184 [==============================] - 572s 3s/step - loss: 94.7535 - mean_squared_error: 94.7535 - val_loss: 76.9819 - val_mean_squared_error: 76.9819
Epoch 5/15
184/184 [==============================] - 568s 3s/step - loss: 95.8063 - mean_squared_error: 95.8063 - val_loss: 77.8694 - val_mean_squared_error: 77.8694
Epoch 6/15
184/184 [==============================] - 565s 3s/step - loss: 92.0491 - mean_squared_error: 92.0491 - val_loss: 75.4635 - val_mean_squared_error: 75.4635
Epoch 7/15
184/184 [==============================] - 566s 3s/step - loss: 91.7978 - mean_squared_error: 91.7978 - val_loss: 76.8055 - val_mean_squared_error: 76.8055
Epoch 8/15
184/184 [==============================] - 566s 3s/step - loss: 91.0338 - mean_squared_error: 91.0338 - val_loss: 75.4134 - val_mean_squared_error: 75.4134
Epoch 9/15
184/184 [==============================] - 564s 3s/step - loss: 87.2379 - mean_squared_error: 87.2379 - val_loss: 81.2928 - val_mean_squared_error: 81.2928
Epoch 10/15
184/184 [==============================] - 564s 3s/step - loss: 87.8208 - mean_squared_error: 87.8208 - val_loss: 82.2583 - val_mean_squared_error: 82.2583
Epoch 11/15
184/184 [==============================] - 570s 3s/step - loss: 88.2591 - mean_squared_error: 88.2591 - val_loss: 72.6616 - val_mean_squared_error: 72.6616
Epoch 12/15
184/184 [==============================] - 568s 3s/step - loss: 86.0876 - mean_squared_error: 86.0876 - val_loss: 77.9364 - val_mean_squared_error: 77.9364
Epoch 13/15
184/184 [==============================] - 566s 3s/step - loss: 85.8031 - mean_squared_error: 85.8031 - val_loss: 77.9824 - val_mean_squared_error: 77.9824
Epoch 14/15
```



We stop training our model further as we observed the near convergence of the training and validation losses.

Mean Square errors for ResNet50v2:

- Training Set - 88.2591
- Validation Set – 72.6616
- Test Set - 137.9188

Model Comparison

We choose mean square error as our parameter to compare our two models as it's a regression model and our data lies between the range of 0-58, so its square value isn't that significant, so won't be an issue for us.

The training error and test error for VGG16 is better than ResNet50v2 whereas the validation error is better for ResNet50v2 model as compared to VGG16.

We choose VGG16 to be better model as it gives a lower error on test set i.e the unseen data for our models.

In general, ResNet50v2 performs better than VGG16 because its faster and has more training parameters but the performance of the model also depends on the computational speed and the resources. Here, VGG16 performs better than ResNet50v2 maybe because of the given dataset of LiDAR images and the parameters we choose based on the limitation of the Google Collab.

GradCAM for VGG16

We visualize the learning, to detect exactly what is happening through GradCAM.

It's a method that allows visualizing how one image activates the neural network. Basically, we look for the direction that the model used to get to its decisions.

We perform GradCAM on VGG16 which gave us a lower test error.

Parameters taken for GradCAM function-

1) We pre-process the image for our VGG model
2) Define the classifier layer names as - `["block5_pool",`
```
"flatten",
"dense",
"dense_1",
"dense_2",]
```
3) Define the last convolutional layer as `"block5_conv3"` based on the model architecture
4) Create the heatmap using GradCAM function
5) Superimpose the heatmap to the image to see the visual learning affect

For representing the whole range of our index, we select the below 10 images on the basis of means and quantiles of our range –

id	education
28676	0.013
28925	0.013
43588	7.029
16726	14.88
41686	29.038
41687	29.038
49990	33.67
40497	45.616
40500	58.976
40501	58.976

The GradCAM output for the selected images is as follows-

Image 28676:

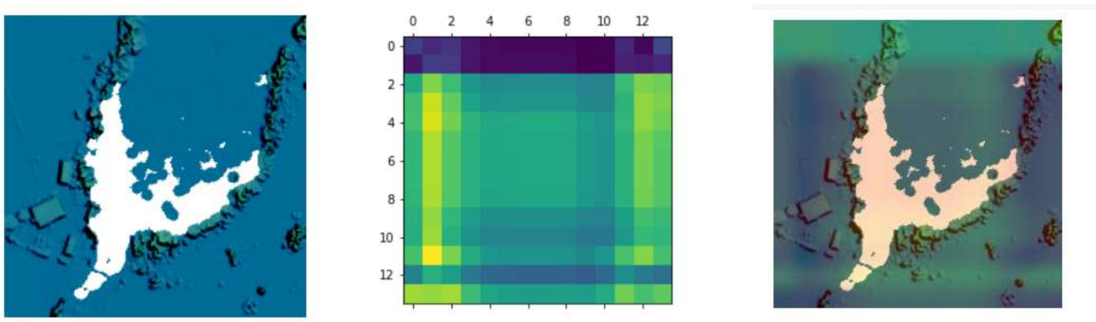


Image 43588:

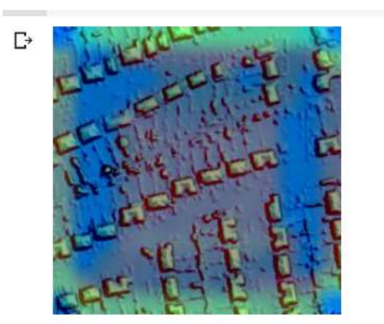


Image 16726:

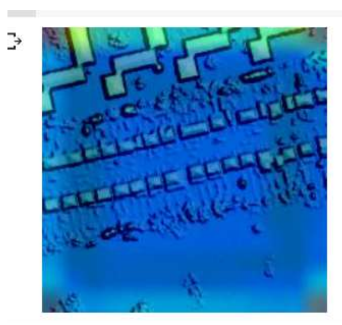


Image 41687:

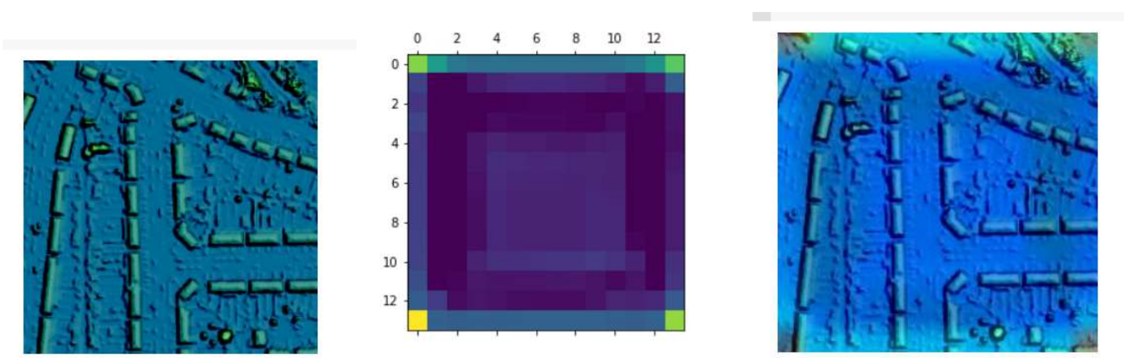
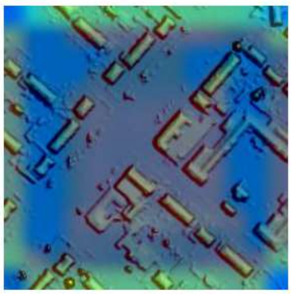
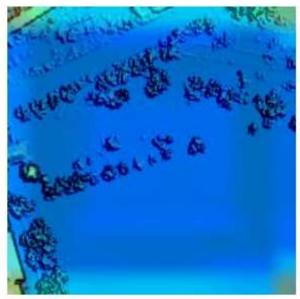
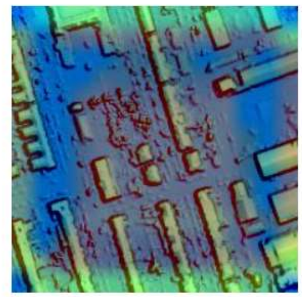


Image 49990: Image 40497: Image 40500:







From the above images we can infer that the model is using the small green boxes or objects as information to make the predictions. These objects can be the buildings or some structures.

We observe that the model tries to make predictions but that isn't accurate and has a high test-error rate of mean squared error of 135 approximately. Our model isn't highly efficient to predict the education level from the LiDAR images which is acceptable as because of the varying challenges in different areas (rural, urban or suburban) we may not be really able to predict the education level of an area accurately from the living spaces or any unlabelled constructions itself.

LiDAR Data – Privacy and Ethical Challenges

LiDAR is a Remote Sensing Method is used to create high resolution representation of an area. A study of the data through these images helps in measuring the Multidimensional Deprivation of that particular area.

The concept that the physical appearance of a human settlement is a reflection of the society that formed it and that people living in comparable physical dwelling conditions have similar social and demographic features underpins the use of remote sensing data to obtain socioeconomic data. Housing conditions and the characteristics of the surrounding environment have been linked to health and general quality of life [3].

Data from remote sensing can also be used to calculate deprivation indices. It could also be beneficial for tracking the quality of life in a given location over time and quantifying changes in its spatial pattern between dates.

As governments use publicly collected data for a variety of objectives, demand for data security and privacy is increasing. Technology that protects privacy and sensitive personal data must be adopted by state entities. When compared to webcams, LiDAR can be quite useful in assisting governments in addressing these privacy concerns. Without collecting facial recognition or other biometric data, LiDAR can monitor municipal areas and follow individuals and vehicles while creating data about their surroundings. Camera-based monitoring systems, on the other hand, record persons facial and other identifying information (Ministry of Housing, Communities & Local Government, September 2019).

Furthermore, the LiDAR image dataset can be used to wrongly categorize people or areas on the basis of the multidimensional deprivation indices. If this information as a sole factor affects the decision-making of any financial institute or any organization then it’s an unethical practice and should be avoided. To overcome this situation, we should consider other sources of information too along with the LiDAR images.

Word Count: 1850

References

1) Emily Nunez, April 2016, The Effects of Geographic Location on Education, https://prezi.com/otn4tcyfjl0s/the-effects-of-geographic-location-on-education/?frame=200f59031000d56383d8ee59438da44912659c4c
2) Daniel Arribas-Bel, Jorge E. Patino, Juan C. Duque (May 2017) , Remote sensing-based measurement of Living Environment Deprivation: Improving classical approaches with machine learning, 10.1371/journal.pone.0176684 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5413026/
3) Ministry of Housing, Communities & Local Government (September 2019), English indices of deprivation 2019: research report https://www.gov.uk/government/publications/english-indices-of-deprivation-2019-research-report

Appendix

Jupyter Notebook Collab link –

https://colab.research.google.com/drive/1S-FRsaRTOZFdVCOAradZ33MJt1fDLujX?usp=sharing

Google Drive link containing all the related files –

https://drive.google.com/drive/folders/18awrTVkOXUTc-C_155520tlFdWX3TSek?usp=sharing

Coursework3_DeepLearning.ipynb

```
!nvidia-smi -L
GPU 0: Tesla K80 (UUID: GPU-b45fbdad-7c98-dd37-b60e-0decfd07ecf5)
```

Deep Learning

Here, we will perform Regression on LiDAR images using VGG16 and ResNet50v2. Later, we apply GradCAM on the better performing model.

Data Preprocessing

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
Mounted at /content/drive
```

In [ ]:

```
!wget "https://uwoca-my.sharepoint.com/:u:/g/personal/cbravoro_uwo_ca/Ea8hL1Qqz-1DqXPUkFg3_OkBkT_oOJ5EdvwX1YU_afWF1w?download=1" #downloads the data
!mv /content/Ea8hL1Qqz-1DqXPUkFg3_OkBkT_oOJ5EdvwX1YU_afWF1w?download=1 /content/data.tar.gz #rename
!tar xvzf data.tar.gz #decompressed
Streaming output truncated to the last 5000 lines.
LIDAR/LIDAR_44106.png
LIDAR/LIDAR_44923.png
LIDAR/LIDAR_53196.png
LIDAR/LIDAR_8048.png
LIDAR/LIDAR_15465.png
LIDAR/LIDAR_30877.png
LIDAR/LIDAR_2042.png
LIDAR/LIDAR_60609.png
```

In [ ]:

```
import pandas as pd

df_data = pd.read_csv("EmbeddingData_C3_9528.csv")
```

In [ ]:

```
df_data.head()
```

Out[ ]:

```
df_data.education.sort_values() # To see the values and perform data analysis
```

Out[ ]:

```
26191     0.013
22240     0.013
1856      0.013
18918     0.013
26750     0.013
          ...
19555    57.186
18153    57.186
2999     58.976
26879    58.976
35423    58.976
Name: education, Length: 36723, dtype: float64
```

In [ ]:

```
df_index = df_data
df_index['Image'] = df_index.apply(lambda x: "LIDAR_" + str(x["id"])+ ".png", axis = 1)
```

In [ ]:

```
# df_index.drop(["LSOA11CD",   "LSOA11NM",     "SOAC11CD",     "SOAC11NM",        "MSOA11CD",     "MSOA11NM",     "LAD17CD",      "LAD17NM",
#                 "LACCD",       "LACNM",        "income",       "employment",
"health","crime" ,      "barriers",     "living_environment"], axis = 1,
inplace = True)

df_index.head()
```

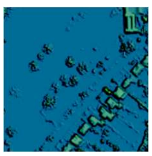
In [ ]:

```
# df_index.sort_values(by=['education'])
```

In [ ]:

```
from IPython.display import Image
Image(filename='/content/LIDAR/LIDAR_28924.png')
```

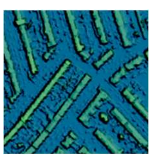
Out[ ]:



In [ ]:

```
Image(filename='/content/LIDAR/LIDAR_40501.png')
```

Out[ ]:



In [ ]:

```
# Dividing the data into train, test sets
from sklearn.model_selection import train_test_split
train, test = train_test_split(df_index,
                               test_size = 0.2,
                               random_state = 251139213)
```

In [ ]:

```
train.head()
```

VGG16

The VGG 16 model is a classic model in Deep Learning. It is a 16 layer model.

This model was trained over the ImageNet data, thus looking to classify among 1000 different types of objects, over a very large database of images. We can leverage these already-trained weights, and adapt just the last few layers for our purposes.

We start by loading the VGG16 model and by first loading the model on-the-fly using the library.

We also need a package that allows for an efficient storage of the model using a binary format. The package is called h5py and also allows for storing your pre-trained models.

In [ ]:

```
import numpy as np
import h5py as h5py
import PIL

# Others
import numpy as np
from sklearn.model_selection import train_test_split

# For AUC estimation and ROC plots
from sklearn.metrics import roc_curve, auc

# Plots
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [ ]:

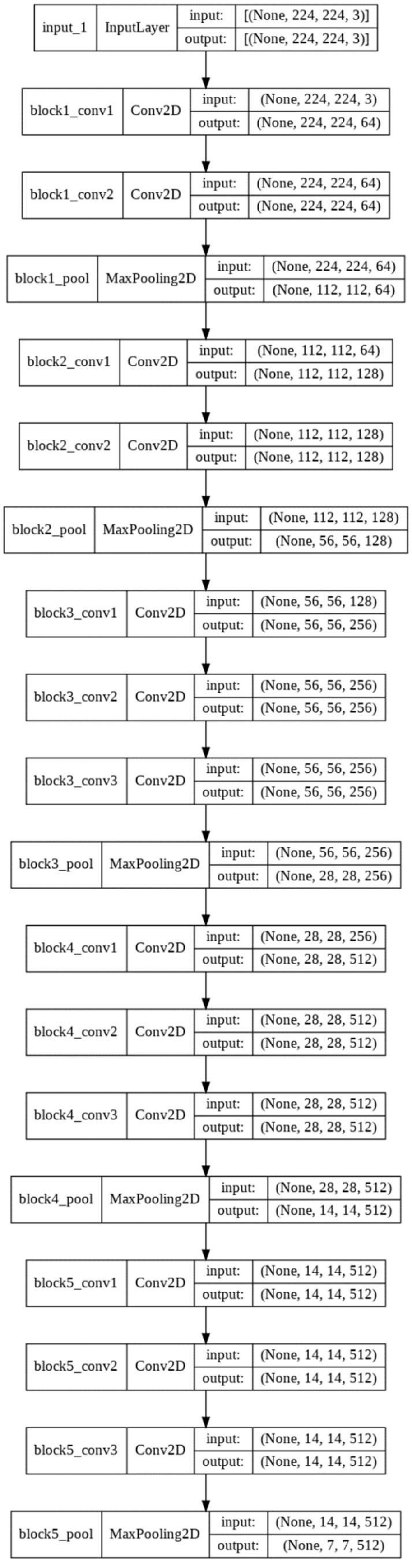
```
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
model = VGG16(weights = 'imagenet',      # The weights from the ImageNet
competition
              include_top = False,       # Do not include the top layer,
which classifies.
              input_shape= (224, 224, 3) # Input shape. Three channels, and
BGR (NOT RGB!!!)
             )
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
58900480/58889256 [==============================] - 1s 0us/step
```

This will download the model and save it to our unoriginally named variable model.

In [ ]:

```
from tensorflow.keras.utils import plot_model
from IPython.display import Image

plot_model(model, show_shapes=True, show_layer_names=True,
to_file='GraphModel.png')
Image(retina=True, filename='GraphModel.png')
```

Out[ ]:


input_1 | InputLayer | input: [(None, 224, 224, 3)] | output: [(None, 224, 224, 3)]
block1_conv1 | Conv2D | input: (None, 224, 224, 3) | output: (None, 224, 224, 64)
block1_conv2 | Conv2D | input: (None, 224, 224, 64) | output: (None, 224, 224, 64)
block1_pool | MaxPooling2D | input: (None, 224, 224, 64) | output: (None, 112, 112, 64)
block2_conv1 | Conv2D | input: (None, 112, 112, 64) | output: (None, 112, 112, 128)
block2_conv2 | Conv2D | input: (None, 112, 112, 128) | output: (None, 112, 112, 128)
block2_pool | MaxPooling2D | input: (None, 112, 112, 128) | output: (None, 56, 56, 128)
block3_conv1 | Conv2D | input: (None, 56, 56, 128) | output: (None, 56, 56, 256)
block3_conv2 | Conv2D | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
block3_conv3 | Conv2D | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
block3_pool | MaxPooling2D | input: (None, 56, 56, 256) | output: (None, 28, 28, 256)
block4_conv1 | Conv2D | input: (None, 28, 28, 256) | output: (None, 28, 28, 512)
block4_conv2 | Conv2D | input: (None, 28, 28, 512) | output: (None, 28, 28, 512)
block4_conv3 | Conv2D | input: (None, 28, 28, 512) | output: (None, 28, 28, 512)
block4_pool | MaxPooling2D | input: (None, 28, 28, 512) | output: (None, 14, 14, 512)
block5_conv1 | Conv2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
block5_conv2 | Conv2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
block5_conv3 | Conv2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
block5_pool | MaxPooling2D | input: (None, 14, 14, 512) | output: (None, 7, 7, 512)

At this point, every single parameter is trainable. We don't need this, as we want to use the parameters that come with the model. We will create a parallel model to store the new trainable layers, and then set all of these layers as untrainable. We will finally add a Dense layer with 128 neurons, plus a Dense layer with two classes.

In [ ]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
```

In [ ]:

```
# Create new model
CBModel = Sequential()

# Copy the layers to our new model. This needs to be done as there is a bug
in Keras.
for layer in model.layers:
    CBModel.add(layer)

# Set the layers as untrainable
for layer in CBModel.layers:
    layer.trainable = False
```

In [ ]:

```
CBModel.summary()
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808   
```

```
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

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

In [ ]:

```
CBModel.layers[15].name
```

Out[ ]:

```
'block5_conv2'
```

In [ ]:

```
# Set layer as trainable.
CBModel.layers[15].trainable = True
CBModel.layers[16].trainable = True
```

In [ ]:

```
# We now add the new layers for prediction.
CBModel.add(Flatten(input_shape=model.output_shape[1:]))
CBModel.add(Dense(128, activation = 'relu'))
CBModel.add(Dropout(0.5))
CBModel.add(Dense(128, activation = 'relu'))
CBModel.add(Dropout(0.5))
CBModel.add(Dense(1, activation = 'relu')) # For Education # All positive
values
```

In [ ]:

```
# How does the model look like?
CBModel.summary()
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080
```

```
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 128)               3211392

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 128)               16512

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 1)                 129

=================================================================
Total params: 17,942,721
Trainable params: 7,947,649
Non-trainable params: 9,995,072
_________________________________________________________________
```

In [ ]:

```
# Compiling the model!
import tensorflow.keras as keras
CBModel.compile(loss=keras.losses.MeanSquaredError(), # MSE for Regression loss
              optimizer=optimizers.Adam(learning_rate=1e-5,            # Learning rate needs to be tweaked for convergence and be small!
                      decay=1e-3 / 200    # Decay of the LR 10^-3 / 1 / 50 / 100 / 200
                      ),
              metrics = [keras.metrics.mean_squared_error],
              )
```

A generator takes images from a directory, and feeds them to the model as needed. **This is necessary to work with big data**. We cannot expect the datasets we work here to fit in memory, so we take the images as needed.

We will first build two image generators (one for testing and one for training), which will generate new samples on the fly using our pictures as input.

We will also conduct **data augmentation**, which are a series of mathematical operations over the datasets to make them search more complex patterns. If you use augmentation, learning will take longer but be more robust. The process to work with this data is the following:

1. Create an [ImageDataGenerator]() object which will process the images and load them as needed.

2. Call the flow_from_dataframe from our generator which will split the data into two parts, one for training and one for validation.

In [ ]:

```
# prepare data augmentation configuration. One for train, one for test.
train_datagen = ImageDataGenerator(
                                  rescale=1./255 ,                          #
NNets like small inputs. Rescale.
                                  shear_range=0,                           #
Shear? As satellite images
                                  zoom_range=0.2,                            #
Zoom? 0.2 means from 80% to 120%
                                  horizontal_flip=True,                      #
Flip horizontally?
                                  vertical_flip=True,                       #
Flip vertically?
                                  preprocessing_function=preprocess_input,   #
VGG expects specific input. Set it up with this function that comes
prepackaged.
                                  validation_split = 0.2                     #
Create a validation cut?
                                  )

test_datagen = ImageDataGenerator(
                                  rescale=1./255 ,                         #
NNets like small inputs. Rescale.
                                  shear_range=0,                           #
Shear?
                                  zoom_range=0,                            #
Zoom? 0.2 means from 80% to 120%
                                  horizontal_flip=False,                   #
Flip horizontally?
                                  vertical_flip=False,                     #
Flip vertically?
                                  preprocessing_function=preprocess_input,#
VGG expects specific input. Set it up with this function that comes
prepackaged.
                                  validation_split = 0                     #
No validation cut for test.
                                  )


# We will use a batch size of 256. Depends on RAM of GPU.
```

```
batch_size = 256


# Train data generator. We point to the training directory!
# train_data_dir = 'IntelClassification/seg_train'

# VGG requires 224 x 224 images.
(img_height, img_width) = (224, 224)

train_generator = train_datagen.flow_from_dataframe(
                                                     dataframe = train,
                                                     directory = 'LIDAR', #
where pics are
                                                     x_col = 'Image',
                                                     y_col = 'education' ,
                                                     target_size=(img_height,
img_width), # What size should they be
                                                     batch_size=batch_size,
# Size of batch
                                                     class_mode='raw',
# Class mode
                                                     subset = 'training',
# What subset to use
                                                     shuffle = True
# Shuffle the data
                                                     )

validation_generator = train_datagen.flow_from_dataframe(
                                                     dataframe = train,
                                                     directory = 'LIDAR', #
where pics are
                                                     x_col = 'Image',
                                                     y_col = 'education' ,
# Where are the pics
                                                     target_size=(img_height,
img_width), # What size should they be
                                                     batch_size=batch_size,
# Size of batch
                                                     class_mode='raw',
# Class mode - raw for regression
                                                     subset = 'validation',
# What subset to use
                                                     shuffle = True
# Shuffle the data
                                                     )


# Test data generator.
# test_data_dir = 'IntelClassification/seg_test'
test_generator = test_datagen.flow_from_dataframe(
                                                   dataframe = test,
                                                   directory = 'LIDAR', #
where pics are
                                                   x_col = 'Image',
```

```
                                                  y_col = 'education' ,
                                                  target_size=(img_height,
img_width),
                                                  batch_size=batch_size, #
Pass images one-by-one
                                                  class_mode='raw',
                                                  shuffle = False # Test set
does NOT shuffle the data.
                                                  )
```

```
Found 23503 validated image filenames.
Found 5875 validated image filenames.
Found 7345 validated image filenames.
```

Now the system is ready to train from the images that we have loaded. We now feed the generators to the model, and ask to train for a certain number of epochs. We found the following datasets:

- Train: Found 23503 images .
- Validation: Found 5875 images.
- Test: Found 7345 images.

Now we can train the model. We will also add a **callback**. Callbacks allow us to stop the training early if we reach convergence, save the model, create temporary plots... anything really. They are fairly powerful and quite necessary when we train big models. We will do two things:

1. Add an EarlyStopping callback to stop training once the validation error stays flat for a couple of epochs.
2. Add a ModelCheckpoint callback that saves the weights of the model with the best performance automatically.

In [ ]:

```
# Define callbacks
import tensorflow as tf
import os
checkpoint_path='/content/drive/MyDrive/FM9528A_Coursework3_251139213/Checkpo
int/CBModel.{epoch:02d}-{val_loss:.2f}.h5'
checkpoint_dir=os.path.dirname(checkpoint_path)
filename = 'Logs.csv'

my_callbacks = [
    # Stop training if validation error stays within 0.00001 for three
rounds.
    tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                     min_delta=0.00001,
                                     patience=3),
    # Save the weights of the best performing model to the checkpoint folder.
    tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                       save_best_only=True,
                                       save_weights_only=True),

    tf.keras.callbacks.CSVLogger(filename, separator = "," , append = True)
]
```

In [ ]:

```
# os.path.dirname('/content/drive/MyDrive/FM9528A_Coursework3_251139213/Checkpoint')
```

Out[ ]:

```
'/content/drive/MyDrive/FM9528A_Coursework3_251139213'
```

In [ ]:

```
# Number of epochs
epochs = 15

# Train!
CBModel.fit(
            train_generator,
            epochs=epochs,
            validation_data=validation_generator,
            steps_per_epoch = 92, # Usually cases / batch_size = 91.8 Reduced to 32 so it runs faster.
            validation_steps = 23, # Number of validation steps. Again cases / batch_size = 22.9
            callbacks= my_callbacks
          )
Epoch 1/15
92/92 [==============================] - 477s 5s/step - loss: 103.7829 - mean_squared_error: 103.7829 - val_loss: 85.3854 - val_mean_squared_error: 85.3854
Epoch 2/15
92/92 [==============================] - 476s 5s/step - loss: 98.2869 - mean_squared_error: 98.2869 - val_loss: 81.4486 - val_mean_squared_error: 81.4486
Epoch 3/15
92/92 [==============================] - 472s 5s/step - loss: 96.1866 - mean_squared_error: 96.1866 - val_loss: 81.7626 - val_mean_squared_error: 81.7626
Epoch 4/15
92/92 [==============================] - 470s 5s/step - loss: 94.0650 - mean_squared_error: 94.0650 - val_loss: 80.5555 - val_mean_squared_error: 80.5555
Epoch 5/15
92/92 [==============================] - 471s 5s/step - loss: 92.4743 - mean_squared_error: 92.4743 - val_loss: 79.5528 - val_mean_squared_error: 79.5528
Epoch 6/15
92/92 [==============================] - 468s 5s/step - loss: 91.3091 - mean_squared_error: 91.3091 - val_loss: 79.3414 - val_mean_squared_error: 79.3414
Epoch 7/15
92/92 [==============================] - 467s 5s/step - loss: 90.4260 - mean_squared_error: 90.4260 - val_loss: 78.9156 - val_mean_squared_error: 78.9156
Epoch 8/15
92/92 [==============================] - 461s 5s/step - loss: 88.7809 - mean_squared_error: 88.7809 - val_loss: 77.3790 - val_mean_squared_error: 77.3790
Epoch 9/15
```

```
92/92 [==============================] - 498s 5s/step - loss: 88.8158 -
mean_squared_error: 88.8158 - val_loss: 77.6049 - val_mean_squared_error:
77.6049
Epoch 10/15
92/92 [==============================] - 486s 5s/step - loss: 88.3588 -
mean_squared_error: 88.3588 - val_loss: 76.9030 - val_mean_squared_error:
76.9030
Epoch 11/15
92/92 [==============================] - 470s 5s/step - loss: 87.1504 -
mean_squared_error: 87.1504 - val_loss: 78.6753 - val_mean_squared_error:
78.6753
Epoch 12/15
92/92 [==============================] - 469s 5s/step - loss: 87.4634 -
mean_squared_error: 87.4634 - val_loss: 76.7901 - val_mean_squared_error:
76.7901
Epoch 13/15
92/92 [==============================] - 469s 5s/step - loss: 86.8074 -
mean_squared_error: 86.8074 - val_loss: 78.1558 - val_mean_squared_error:
78.1558
Epoch 14/15
92/92 [==============================] - 469s 5s/step - loss: 86.5004 -
mean_squared_error: 86.5004 - val_loss: 79.7613 - val_mean_squared_error:
79.7613
Epoch 15/15
92/92 [==============================] - 471s 5s/step - loss: 86.1052 -
mean_squared_error: 86.1052 - val_loss: 76.6983 - val_mean_squared_error:
76.6983
```

Out[ ]:

```
<keras.callbacks.History at 0x7face2760b10>
```

The model is able to learn quite well! Checking the convergence plot.

In general, if we are seeing much higher validation loss than training loss, then it's a sign that your model is overfitting - it learns "superstitions" i.e. patterns that accidentally happened to be true in your training data but don't have a basis in reality, and thus aren't true in your validation data

If training loss much greater than validation loss. That is underfitting. If training loss much less than validation loss. That is overfitting.

In [ ]:

```
# Saving the model
CBModel.save('/content/drive/MyDrive/FM9528A_Coursework3_251139213/VGG16.h5')

# Loading
# CBModel =
keras.models.load_model('/content/drive/MyDrive/FM9528A_Coursework3_251139213
/VGG16_Adam_Final.h5')
```

In [ ]:

```
l1 = CBModel.history.history['loss']
vl1 = CBModel.history.history['val_loss']
```

In [ ]:

```
# Number of epochs
epochs = 8

# Train!
CBModel.fit(
```

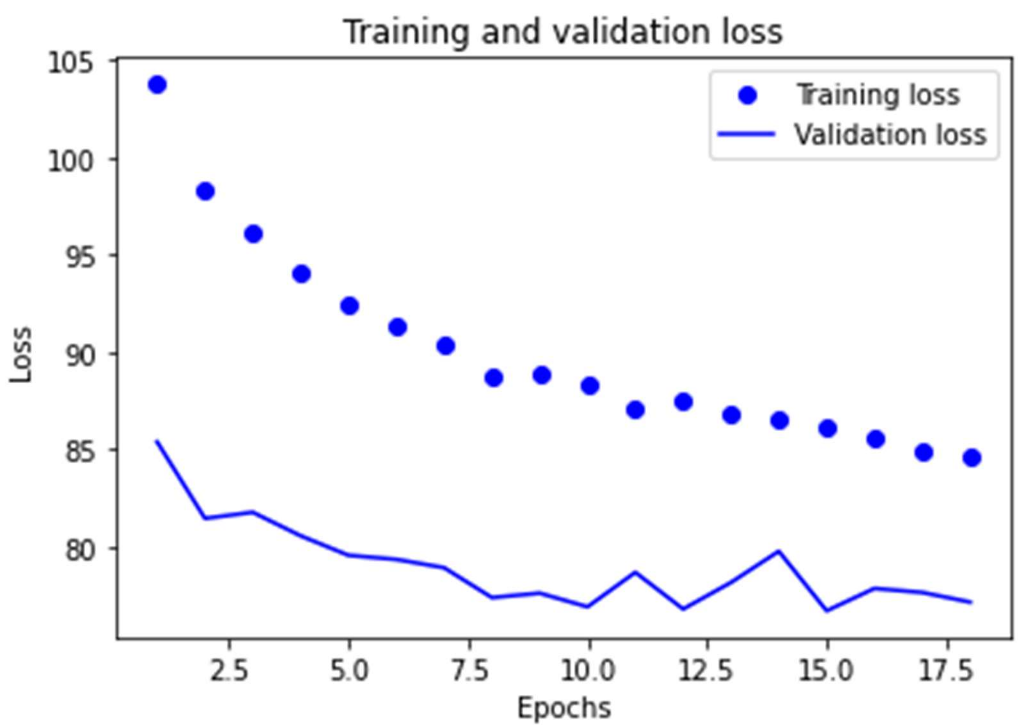
```
            train_generator,
            epochs=epochs,
            validation_data=validation_generator,
            steps_per_epoch = 92, # Usually cases / batch_size = 91.8 Reduced
to 32 so it runs faster.
            validation_steps = 23, # Number of validation steps. Again cases
/ batch_size = 22.9
            callbacks= my_callbacks
          )
```

In [ ]:

```
l2 = CBModel.history.history['loss']
vl2 = CBModel.history.history['val_loss']
```

In [ ]:

```
loss = l1+l2[0:3]
val_loss = vl1 + vl2[0:3]
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [ ]:

```
import tensorflow.keras as keras

CBModel =
keras.models.load_model('/content/drive/MyDrive/FM9528A_Coursework3_251139213
/VGG16.h5')
```

In [ ]:

```
# Load the weights. THIS REQUIRES FIRST CREATING THE LOGIC.

CBModel.load_weights('/content/drive/MyDrive/FM9528A_Coursework3_251139213/Ch
eckpoint/CBModel.15-76.70.h5')
```

In [ ]:

```
# Applying to the test set with a generator.
test_generator.reset()

# Get probabilities
output = CBModel.predict(test_generator)
```

In [ ]:

```
output.reshape(-1)
```

Out[ ]:

```
array([ 7.2357106,  8.524901 , 19.085115 , ..., 16.474632 ,  5.402652 ,
        8.168507 ], dtype=float32)
```

In [ ]:

```
test_generator.labels
```

Out[ ]:

```
array([ 7.639, 12.772, 22.219, ..., 31.733,  0.696, 12.772])
```

In [ ]:

```
def mean_sq_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean((y_true - y_pred)**2 )
```

In [ ]:

```
mse_res = mean_sq_error(test_generator.labels, output)
print(f'The mean squared error over the test is {mse_res}')
The mean squared error over the test is 134.5889583910144
```

ResNet

In [ ]:

```
# Parameters
ImageSize = (224,224)
BatchSize = 128
```

In [ ]:

```
# Import base model. Using ResNet50v2.
from tensorflow.keras.applications.resnet_v2 import ResNet50V2,
preprocess_input

# Import model with input layer
base_model = ResNet50V2(weights = 'imagenet',     # The weights from the
ImageNet competition
                        include_top = False,       # Do not include the top
layer, which classifies.
                        input_shape= (224, 224, 3) # Input shape. Three
channels.
                        )
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
94674944/94668760 [==============================] - 2s 0us/step
94683136/94668760 [==============================] - 2s 0us/step
```

Our process will be to:

1. Set the base model (ResNet) to non-trainable (freeze weights).
2. Add the new model.
3. Train only the top to convergence.
4. Unfreeze the weights to fine-tune.

These steps are needed to properly adjust the weights and are the recommended practice when finetunning more complex models, as explained here. This website also has an example using Xception, another model.

In [ ]:

```
# Set the base model to untrainable.
base_model.trainable = False
```

In [ ]:

```
# Create the full model using the Model API
import tensorflow.keras as keras

# Input layer
inputs = keras.Input(shape=ImageSize + (3,),
                        name = 'image_only_input')

# Add the ResNet model, setting it to be untrainable.
# First we store it on a temporary variable.
x = base_model(inputs, training=False)

# Flatten to make it the same size as the original model
x = Flatten()(x)

# Now we actually add it to a layer. Note the way of writing it.
x = Dense(64, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.5)(x)

# Add final output layer.
outputs = Dense(1, activation='relu')(x)

# Create the complete model object
ResModel = keras.Model(inputs, outputs)
```

In [ ]:

```
# This is what the model looks like now.
ResModel.summary()
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 image_only_input (InputLaye  [(None, 224, 224, 3)]    0         
 r)
```

```
 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800

 flatten_1 (Flatten)         (None, 100352)            0

 dense_3 (Dense)             (None, 64)                6422592

 dropout_2 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 64)                4160

 dropout_3 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 29,991,617
Trainable params: 6,426,817
Non-trainable params: 23,564,800
_________________________________________________________________
```

In [ ]:

```
# Compiling the model! Note the learning rate.
opt = optimizers.Adam(learning_rate=1e-5,            # Learning rate needs to
be tweaked for convergence and be small!
                      decay=1e-3 / 200    # Decay of the LR 10^-3 / 1 / 50 /
100 / 200
                      )
ResModel.compile(loss=keras.losses.MeanSquaredError(), # This is NOT a
classification problem!
                      optimizer=opt,
                      metrics = [keras.metrics.mean_squared_error],
                       )
```

In [ ]:

```
# Define parameters

target_size = (224, 224)
batch_size = 128
DataDir = 'LIDAR'

# Define generators
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
                                  rescale=None,                        #
Inputs are scaled in the preprocessing function
                                  shear_range=0,                         #
Shear?
                                  zoom_range=0.2,                          #
Zoom? 0.2 means from 80% to 120%
                                  horizontal_flip=True,                    #
Flip horizontally?
                                  vertical_flip=True,                    #
Flip vertically?
```

```
                                  preprocessing_function=preprocess_input,  #
ResNet expects specific input. Set it up with this function that comes
prepackaged.
                                  validation_split = 0.2                    #
Create a validation cut?
                                  )

test_datagen = ImageDataGenerator(
                                  rescale=None,                          #
Inputs are scaled in the preprocessing function
                                  shear_range=0,                            #
Shear?
                                  zoom_range=0,                             #
Zoom? 0.2 means from 80% to 120%
                                  horizontal_flip=False,
# Flip horizontally?
                                  vertical_flip=False,                      #
Flip vertically?
                                  preprocessing_function=preprocess_input,  #
VGG expects specific input. Set it up with this function that comes
prepackaged.
                                  )

# Point to the data and **give the targets**. Note the "raw" class_mode
train_generator = train_datagen.flow_from_dataframe(train,
                                                    directory='LIDAR',  #
Look from root directory
                                                    x_col='Image',   # Path
to images
                                                    y_col='education',  #
Target
                                                    target_size=target_size,
# Same as last lab
                                                    batch_size=batch_size,
                                                    shuffle=True,
                                                    class_mode='raw',
                                                    subset='training',
                                                    interpolation="bilinear"
                                                   )

validation_generator = train_datagen.flow_from_dataframe(train,
                                                    directory='LIDAR',
                                                    x_col='Image',
                                                    y_col='education',
                                                    target_size=target_size,
                                                    batch_size=batch_size,
                                                    shuffle=True,
                                                    class_mode='raw',
                                                    subset='validation',
                                                    interpolation="bilinear"
                                                   )

test_generator = test_datagen.flow_from_dataframe(test,
```

```
                                                  directory='LIDAR',
                                                  x_col='Image',
                                                  y_col='education',
                                                  target_size=target_size,
                                                  batch_size=batch_size,
                                                  shuffle=False,
                                                  class_mode='raw',
                                                  interpolation="bilinear"
                                                  )
Found 23503 validated image filenames.
Found 5875 validated image filenames.
Found 7345 validated image filenames.
```

Now let's train! We can easily train this model by calling the fit function and passing the generator. This will **only** train the dense layers, as it is recommended first. It is always a good idea to first give the training parameters somewhere to start from. This is called **model warming up**. We can train the rest of the model in a second round.

You only need to give it a few rounds.

In [ ]:

```
# Define callbacks
import tensorflow as tf
import os
checkpoint_path='/content/drive/MyDrive/FM9528A_Coursework3_251139213/Checkpo
int/ResModel.{epoch:02d}-{val_loss:.2f}.h5'
checkpoint_dir=os.path.dirname(checkpoint_path)
filename = 'Logs2.csv'

my_callbacks2 = [
    # Stop training if validation error stays within 0.00001 for three
rounds.
    tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                     min_delta=0.00001,
                                     patience=3),
    # Save the weights of the best performing model to the checkpoint folder.
    tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                       save_best_only=True,
                                       save_weights_only=True),

    tf.keras.callbacks.CSVLogger(filename, separator = "," , append = True)
]
```

In [ ]:

```
# Number of epochs
epochs = 3

# Train!
ResModel.fit(
                  train_generator,
                  epochs=epochs,
                  validation_data=validation_generator,
                  steps_per_epoch = 184, # Usually cases / batch_size = 10.
                  validation_steps = 46, # Number of validation steps. Again
cases / batch_size = 3.
```

```
                  callbacks = my_callbacks2
                  )
Epoch 1/3
184/184 [==============================] - 496s 3s/step - loss: 148.5081 - mean_squared_error: 148.5081 - val_loss: 102.2973 - val_mean_squared_error: 102.2973
Epoch 2/3
184/184 [==============================] - 464s 3s/step - loss: 125.7656 - mean_squared_error: 125.7656 - val_loss: 97.1081 - val_mean_squared_error: 97.1081
Epoch 3/3
184/184 [==============================] - 453s 2s/step - loss: 121.5963 - mean_squared_error: 121.5963 - val_loss: 95.0181 - val_mean_squared_error: 95.0181
```

Out[ ]:

```
<keras.callbacks.History at 0x7f0224daa2d0>
```

In [ ]:

```
lr1 = ResModel.history.history['loss']
lr_val = ResModel.history.history['val_loss']
```

In [ ]:

```
lr1
```

Out[ ]:

```
[148.50807189941406, 125.76556396484375, 121.59626770019531]
```

In [ ]:

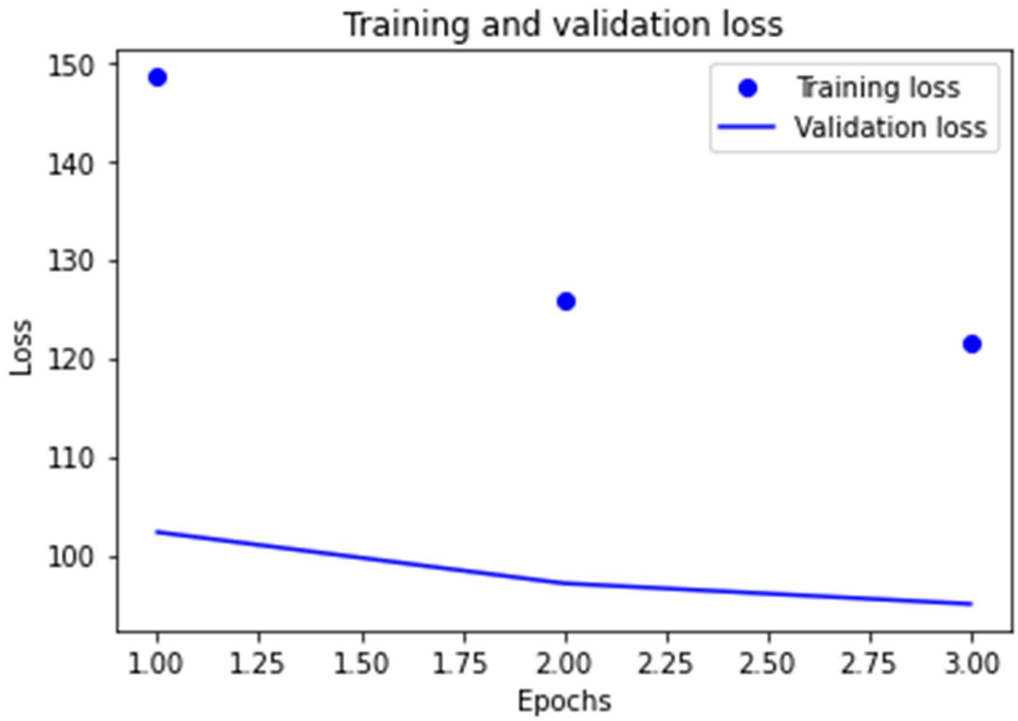
```
lr_val
```

Out[ ]:

```
[102.29725646972656, 97.10811614990234, 95.01813507080078]
```

In [ ]:

```
loss = ResModel.history.history['loss']
val_loss = ResModel.history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [ ]:

```
ResModel.save('/content/drive/MyDrive/FM9528A_Coursework3_251139213/Models/ResModel_warm.h5')
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom mask layers require a config and must override get_config. When loading, the custom mask layer must be passed to the custom_objects argument.
  layer_config = serialize_layer_fn(layer)
```

The model did not learn much, but we are only training the dense layers. Let's try to train now all layers.

In [ ]:

```
# Define callbacks
import tensorflow as tf
import os
checkpoint_path='/content/drive/MyDrive/FM9528A_Coursework3_251139213/Checkpoint/ResModelFe2.{epoch:02d}-{val_loss:.2f}.h5'
checkpoint_dir=os.path.dirname(checkpoint_path)
filename = 'Logs3.csv'

my_callbacks3 = [
    # Stop training if validation error stays within 0.00001 for three rounds.
    tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                     min_delta=0.00001,
                                     patience=3),
    # Save the weights of the best performing model to the checkpoint folder.
    tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                       save_best_only=True,
                                       save_weights_only=True),
```

```
    tf.keras.callbacks.CSVLogger(filename, separator = "," , append = True)
]
```

In [ ]:

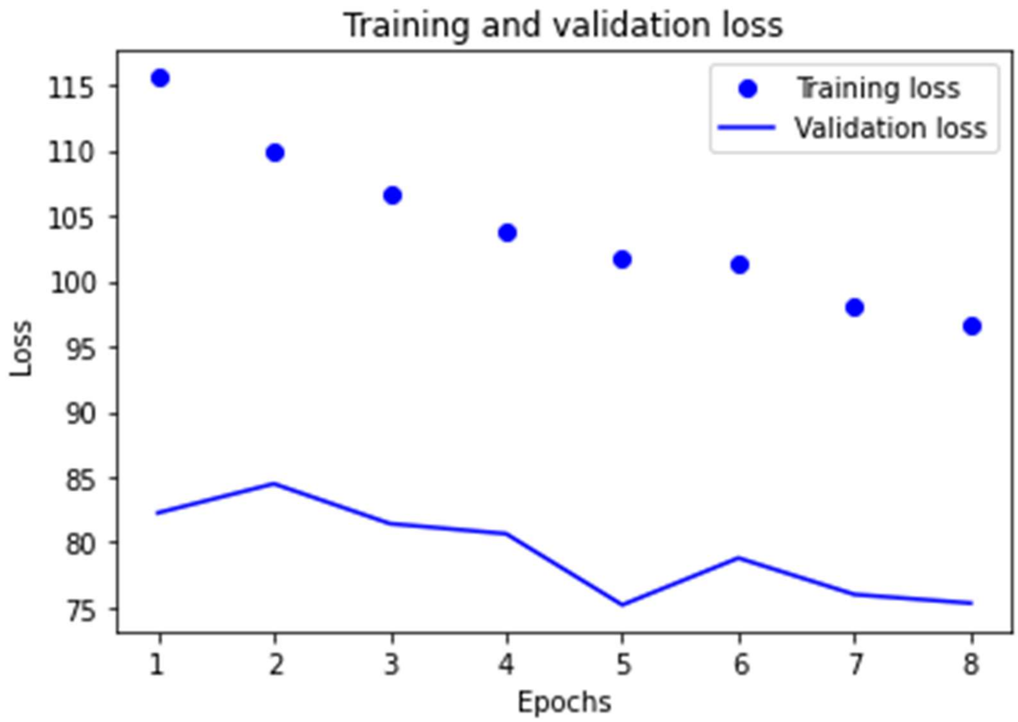
```
base_model.trainable = True

# Recompile as we changed things.
ResModel.compile(loss=keras.losses.MeanSquaredError(), # This is NOT a
classification problem!
                      optimizer=opt,
                      metrics = [keras.metrics.mean_squared_error],

                       )

# Number of epochs
epochs = 5

# Train!
ResModel.fit(
                  train_generator,
                  epochs=epochs,
                  validation_data=validation_generator,
                  steps_per_epoch = 184, # Usually cases / batch_size = 10. =
23503/128 = 183.61
                  validation_steps = 46, # Number of validation steps. Again
cases / batch_size = 3. = 5875/128 = 45.89
                  callbacks = my_callbacks3

                  )
Epoch 1/5
184/184 [==============================] - 617s 3s/step - loss: 100.9013 -
mean_squared_error: 100.9013 - val_loss: 76.4586 - val_mean_squared_error:
76.4586
Epoch 2/5
184/184 [==============================] - 558s 3s/step - loss: 99.0744 -
mean_squared_error: 99.0744 - val_loss: 75.9061 - val_mean_squared_error:
75.9061
Epoch 3/5
184/184 [==============================] - 558s 3s/step - loss: 96.7607 -
mean_squared_error: 96.7607 - val_loss: 80.5025 - val_mean_squared_error:
80.5025
Epoch 4/5
141/184 [=====================>........] - ETA: 1:49 - loss: 96.1709 -
mean_squared_error: 96.1709
```

Keras gives us the full training history of the model, which we can use to track convergence. The following code plots this history.

In [ ]:

```
# Plotting training history.
loss = ResModel.history.history['loss']
val_loss = ResModel.history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
```

```
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [ ]:

```
ResModel.save('/content/drive/MyDrive/FM9528A_Coursework3_251139213/Models/ResModel_FullT.h5')
```

Further training our data

In [ ]:

```
import tensorflow.keras as keras

ResModel =
keras.models.load_model('/content/drive/MyDrive/FM9528A_Coursework3_251139213/ResModel_FullT.h5')
WARNING:tensorflow:Error in loading the saved optimizer state. As a result, your model is starting with a freshly initialized optimizer.
```

In [ ]:

```
base_model.trainable = True

# Recompile as we changed things.
ResModel.compile(loss=keras.losses.MeanSquaredError(), # This is NOT a classification problem!
                      optimizer=opt,
                      metrics = [keras.metrics.mean_squared_error],

                       )

# Number of epochs
```

```
epochs = 15

# Train!
ResModel.fit(
                  train_generator,
                  epochs=epochs,
                  validation_data=validation_generator,
                  steps_per_epoch = 184, # Usually cases / batch_size = 10. = 
23503/128 = 183.61
                  validation_steps = 46, # Number of validation steps. Again 
cases / batch_size = 3. = 5875/128 = 45.89
                  callbacks = my_callbacks3

                  )
Epoch 1/15
184/184 [==============================] - 636s 3s/step - loss: 100.5641 - 
mean_squared_error: 100.5641 - val_loss: 81.6307 - val_mean_squared_error: 
81.6307
Epoch 2/15
184/184 [==============================] - 570s 3s/step - loss: 99.6896 - 
mean_squared_error: 99.6896 - val_loss: 76.9703 - val_mean_squared_error: 
76.9703
Epoch 3/15
184/184 [==============================] - 565s 3s/step - loss: 96.4278 - 
mean_squared_error: 96.4278 - val_loss: 76.3962 - val_mean_squared_error: 
76.3962
Epoch 4/15
184/184 [==============================] - 572s 3s/step - loss: 94.7535 - 
mean_squared_error: 94.7535 - val_loss: 76.9819 - val_mean_squared_error: 
76.9819
Epoch 5/15
184/184 [==============================] - 568s 3s/step - loss: 95.8063 - 
mean_squared_error: 95.8063 - val_loss: 77.8694 - val_mean_squared_error: 
77.8694
Epoch 6/15
184/184 [==============================] - 565s 3s/step - loss: 92.0491 - 
mean_squared_error: 92.0491 - val_loss: 75.4635 - val_mean_squared_error: 
75.4635
Epoch 7/15
184/184 [==============================] - 566s 3s/step - loss: 91.7978 - 
mean_squared_error: 91.7978 - val_loss: 76.8055 - val_mean_squared_error: 
76.8055
Epoch 8/15
184/184 [==============================] - 566s 3s/step - loss: 91.0338 - 
mean_squared_error: 91.0338 - val_loss: 75.4134 - val_mean_squared_error: 
75.4134
Epoch 9/15
184/184 [==============================] - 564s 3s/step - loss: 87.2379 - 
mean_squared_error: 87.2379 - val_loss: 81.2928 - val_mean_squared_error: 
81.2928
Epoch 10/15
184/184 [==============================] - 564s 3s/step - loss: 87.8208 - 
mean_squared_error: 87.8208 - val_loss: 82.2583 - val_mean_squared_error: 
82.2583
Epoch 11/15
```

```
184/184 [==============================] - 570s 3s/step - loss: 88.2591 -
mean_squared_error: 88.2591 - val_loss: 72.6616 - val_mean_squared_error:
72.6616
Epoch 12/15
184/184 [==============================] - 568s 3s/step - loss: 86.0876 -
mean_squared_error: 86.0876 - val_loss: 77.9364 - val_mean_squared_error:
77.9364
Epoch 13/15
184/184 [==============================] - 566s 3s/step - loss: 85.8031 -
mean_squared_error: 85.8031 - val_loss: 77.9824 - val_mean_squared_error:
77.9824
Epoch 14/15
 99/184 [===============>..............] - ETA: 3:43 - loss: 84.6684 -
mean_squared_error: 84.6684
```

In [ ]:

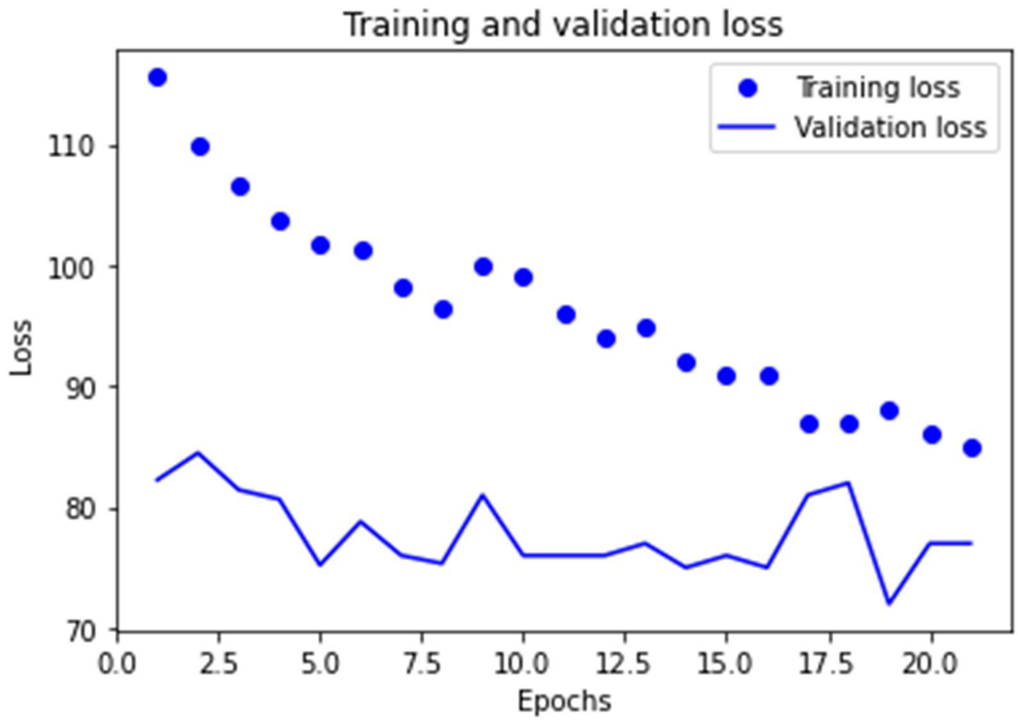
```
lR = [100,99,96,94,95,92,91,91,87,87,88,86,85]
vlR = [81,76,76,76,77,75,76,75,81,82,72,77,77]
```

In [ ]:

```
Loss = [115.58602905273438,
 109.8603286743164,
 106.69586944580078,
 103.75898742675781,
 101.72020721435547,
 101.37633514404297,
 98.12415313720703,
 96.55558013916016]

Val_loss = [82.2503433227539,
 84.47977447509766,
 81.42951965332031,
 80.65293884277344,
 75.21021270751953,
 78.79944610595703,
 76.00736999511719,
 75.34436798095703]
```

In [ ]:

```
# Plotting training history.
loss = Loss + lR
val_loss = Val_loss + vlR
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [ ]:

```
ResModel.save('/content/drive/MyDrive/FM9528A_Coursework3_251139213/ResModel_FullTe2.h5')
```

Let's apply the model to our test data. For this we use the generator we just created.

In [ ]:

```
ResModel.load_weights('/content/drive/MyDrive/FM9528A_Coursework3_251139213/Checkpoint/ResModelFe2.11-72.66.h5')
```

In [ ]:

```
# ResModel.load_weights('/content/drive/MyDrive/FM9528A_Coursework3_251139213/Checkpoint/ResModelF.05-75.21.h5')
# ResModel.load_weights('/content/drive/MyDrive/FM9528A_Coursework3_251139213/Checkpoint/ResModelFe2.08-75.41.h5')
# ResModel.load_weights('/content/drive/MyDrive/FM9528A_Coursework3_251139213/Checkpoint/ResModelFe2.03-76.40.h5')
```

In [ ]:

```
# ResModel.load_weights('/content/drive/MyDrive/FM9528A_Coursework3_251139213/Checkpoint/ResModelFe2.03-76.40.h5')
```

In [ ]:

```
# Applying to the test set with a generator.
test_generator.reset()

# Get probabilities
output = ResModel.predict(test_generator)
```

In [ ]:

```
output.reshape(-1)
```

Out[ ]:

```
array([ 5.28932  , 10.581744 , 15.953831 , ..., 13.104308 ,  6.9865203,
        6.1362624], dtype=float32)
```

In [ ]:

```
def mean_sq_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean((y_true - y_pred)**2 )
```

In [ ]:

```
mse2_res = mean_sq_error(test_generator.labels, output)
print(f'The mean squared error over the test is {mse2_res}')
The mean squared error over the test is 137.9188182606288
```

GradCAM for ResNet50v2

In [ ]:

```
# The explainer. Gotten from https://keras.io/examples/vision/grad_cam/
def make_gradcam_heatmap(
    img_array, model, last_conv_layer_name, classifier_layer_names
):
    from tensorflow import keras
    import tensorflow as tf
    # First, we create a model that maps the input image to the activations
    # of the last conv layer. This layer is located at model.layers[1] as the
    # ResNet model is the first "layer" of the ImageOnlyModel. Modify as
needed.
    last_conv_layer = model.layers[1].get_layer(last_conv_layer_name)
    last_conv_layer_model = keras.Model(model.layers[1].inputs,
last_conv_layer.output)
    print(last_conv_layer)
    print(last_conv_layer_model)

    # Second, we create a model that maps the activations of the last conv
    # layer to the final class predictions
    regression_input = keras.Input(shape=last_conv_layer.output.shape[1:])
    x = regression_input
    for layer_name in classifier_layer_names:
        try:
            x = model.get_layer(layer_name)(x)
        except:
            x = model.layers[1].get_layer(layer_name)(x)
    regression_model = keras.Model(regression_input, x)
    print(regression_model.summary())

    # Then, we compute the gradient of the top predicted class for our input
image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        # Compute activations of the last conv layer and make the tape watch
it
```

```
        last_conv_layer_output = last_conv_layer_model(img_array)
        print("last conv layer output:", last_conv_layer_output)
        tape.watch(last_conv_layer_output)
        # Compute predictions
        top_class_channel = regression_model(last_conv_layer_output)
        print("prediction:", top_class_channel)

    # This is the gradient of the top predicted class with regard to
    # the output feature map of the last conv layer
    grads = tape.gradient(top_class_channel, last_conv_layer_output)
    print("gradients:", grads)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the regression
    last_conv_layer_output = last_conv_layer_output.numpy()[0]
    pooled_grads = pooled_grads.numpy()
    for i in range(pooled_grads.shape[-1]):
        last_conv_layer_output[:, :, i] *= pooled_grads[i]

    print(pooled_grads)

    # The channel-wise mean of the resulting feature map
    # is our heatmap of activation
    heatmap = np.mean(last_conv_layer_output, axis=-1)

    # For visualization purpose, we will also normalize the heatmap between 0
& 1
    # heatmap = np.maximum(heatmap, 0) / np.max(heatmap)
    print(heatmap)

    return heatmap
```

Now we can proceed as normal. Let's get the last convolutional layer and the top of the model.

In [ ]:

```
ResModel.summary()
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 image_only_input (InputLaye  [(None, 224, 224, 3)]    0
 r)

 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800

 flatten (Flatten)           (None, 100352)            0

 dense (Dense)               (None, 64)                6422592

 dropout (Dropout)           (None, 64)                0
```

```
 dense_1 (Dense)             (None, 64)                4160

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 29,991,617
Trainable params: 29,946,177
Non-trainable params: 45,440
_________________________________________________________________
```

In [ ]:

```
base_model.summary()
Model: "resnet50v2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_5 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0
['input_5[0][0]']

 conv1_conv (Conv2D)            (None, 112, 112, 64  9472
['conv1_pad[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0
['conv1_conv[0][0]']
                                )

 pool1_pool (MaxPooling2D)      (None, 56, 56, 64)   0
['pool1_pad[0][0]']

 conv2_block1_preact_bn (BatchN  (None, 56, 56, 64)  256
['pool1_pool[0][0]']
 ormalization)

 conv2_block1_preact_relu (Acti  (None, 56, 56, 64)  0
['conv2_block1_preact_bn[0][0]']
 vation)

 conv2_block1_1_conv (Conv2D)   (None, 56, 56, 64)   4096
['conv2_block1_preact_relu[0][0]'
                                                                 ]

 conv2_block1_1_bn (BatchNormal  (None, 56, 56, 64)  256
['conv2_block1_1_conv[0][0]']
 ization)

 conv2_block1_1_relu (Activatio  (None, 56, 56, 64)  0
['conv2_block1_1_bn[0][0]']
 n)
```

```
 conv2_block1_2_pad (ZeroPaddin  (None, 58, 58, 64)  0
['conv2_block1_1_relu[0][0]']
 g2D)

 conv2_block1_2_conv (Conv2D)   (None, 56, 56, 64)   36864
['conv2_block1_2_pad[0][0]']

 conv2_block1_2_bn (BatchNormal  (None, 56, 56, 64)  256
['conv2_block1_2_conv[0][0]']
 ization)

 conv2_block1_2_relu (Activatio  (None, 56, 56, 64)  0
['conv2_block1_2_bn[0][0]']
 n)

 conv2_block1_0_conv (Conv2D)   (None, 56, 56, 256)  16640
['conv2_block1_preact_relu[0][0]'
                                                            ]

 conv2_block1_3_conv (Conv2D)   (None, 56, 56, 256)  16640
['conv2_block1_2_relu[0][0]']

 conv2_block1_out (Add)         (None, 56, 56, 256)  0
['conv2_block1_0_conv[0][0]',

'conv2_block1_3_conv[0][0]']

 conv2_block2_preact_bn (BatchN  (None, 56, 56, 256)  1024
['conv2_block1_out[0][0]']
 ormalization)

 conv2_block2_preact_relu (Acti  (None, 56, 56, 256)  0
['conv2_block2_preact_bn[0][0]']
 vation)

 conv2_block2_1_conv (Conv2D)   (None, 56, 56, 64)   16384
['conv2_block2_preact_relu[0][0]'
                                                            ]

 conv2_block2_1_bn (BatchNormal  (None, 56, 56, 64)  256
['conv2_block2_1_conv[0][0]']
 ization)

 conv2_block2_1_relu (Activatio  (None, 56, 56, 64)  0
['conv2_block2_1_bn[0][0]']
 n)

 conv2_block2_2_pad (ZeroPaddin  (None, 58, 58, 64)  0
['conv2_block2_1_relu[0][0]']
 g2D)

 conv2_block2_2_conv (Conv2D)   (None, 56, 56, 64)   36864
['conv2_block2_2_pad[0][0]']

 conv2_block2_2_bn (BatchNormal  (None, 56, 56, 64)  256
['conv2_block2_2_conv[0][0]']
 ization)
```

```
 conv2_block2_2_relu (Activatio  (None, 56, 56, 64)  0
['conv2_block2_2_bn[0][0]']
 n)

 conv2_block2_3_conv (Conv2D)   (None, 56, 56, 256)  16640
['conv2_block2_2_relu[0][0]']

 conv2_block2_out (Add)         (None, 56, 56, 256)  0
['conv2_block1_out[0][0]',

'conv2_block2_3_conv[0][0]']

 conv2_block3_preact_bn (BatchN  (None, 56, 56, 256)  1024
['conv2_block2_out[0][0]']
 ormalization)

 conv2_block3_preact_relu (Acti  (None, 56, 56, 256)  0
['conv2_block3_preact_bn[0][0]']
 vation)

 conv2_block3_1_conv (Conv2D)   (None, 56, 56, 64)   16384
['conv2_block3_preact_relu[0][0]'
                                                              ]

 conv2_block3_1_bn (BatchNormal  (None, 56, 56, 64)  256
['conv2_block3_1_conv[0][0]']
 ization)

 conv2_block3_1_relu (Activatio  (None, 56, 56, 64)  0
['conv2_block3_1_bn[0][0]']
 n)

 conv2_block3_2_pad (ZeroPaddin  (None, 58, 58, 64)  0
['conv2_block3_1_relu[0][0]']
 g2D)

 conv2_block3_2_conv (Conv2D)   (None, 28, 28, 64)   36864
['conv2_block3_2_pad[0][0]']

 conv2_block3_2_bn (BatchNormal  (None, 28, 28, 64)  256
['conv2_block3_2_conv[0][0]']
 ization)

 conv2_block3_2_relu (Activatio  (None, 28, 28, 64)  0
['conv2_block3_2_bn[0][0]']
 n)

 max_pooling2d (MaxPooling2D)   (None, 28, 28, 256)  0
['conv2_block2_out[0][0]']

 conv2_block3_3_conv (Conv2D)   (None, 28, 28, 256)  16640
['conv2_block3_2_relu[0][0]']

 conv2_block3_out (Add)         (None, 28, 28, 256)  0
['max_pooling2d[0][0]',
```

```
'conv2_block3_3_conv[0][0]']

 conv3_block1_preact_bn (BatchN  (None, 28, 28, 256)  1024
['conv2_block3_out[0][0]']
 ormalization)

 conv3_block1_preact_relu (Acti  (None, 28, 28, 256)  0
['conv3_block1_preact_bn[0][0]']
 vation)

 conv3_block1_1_conv (Conv2D)   (None, 28, 28, 128)  32768
['conv3_block1_preact_relu[0][0]'
                                                                  ]

 conv3_block1_1_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block1_1_conv[0][0]']
 ization)

 conv3_block1_1_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block1_1_bn[0][0]']
 n)

 conv3_block1_2_pad (ZeroPaddin  (None, 30, 30, 128)  0
['conv3_block1_1_relu[0][0]']
 g2D)

 conv3_block1_2_conv (Conv2D)   (None, 28, 28, 128)  147456
['conv3_block1_2_pad[0][0]']

 conv3_block1_2_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block1_2_conv[0][0]']
 ization)

 conv3_block1_2_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block1_2_bn[0][0]']
 n)

 conv3_block1_0_conv (Conv2D)   (None, 28, 28, 512)  131584
['conv3_block1_preact_relu[0][0]'
                                                                  ]

 conv3_block1_3_conv (Conv2D)   (None, 28, 28, 512)  66048
['conv3_block1_2_relu[0][0]']

 conv3_block1_out (Add)         (None, 28, 28, 512)  0
['conv3_block1_0_conv[0][0]',

'conv3_block1_3_conv[0][0]']

 conv3_block2_preact_bn (BatchN  (None, 28, 28, 512)  2048
['conv3_block1_out[0][0]']
 ormalization)

 conv3_block2_preact_relu (Acti  (None, 28, 28, 512)  0
['conv3_block2_preact_bn[0][0]']
 vation)
```

```
 conv3_block2_1_conv (Conv2D)   (None, 28, 28, 128)  65536
['conv3_block2_preact_relu[0][0]'
                                                                  ]

 conv3_block2_1_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block2_1_conv[0][0]']
 ization)

 conv3_block2_1_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block2_1_bn[0][0]']
 n)

 conv3_block2_2_pad (ZeroPaddin  (None, 30, 30, 128)  0
['conv3_block2_1_relu[0][0]']
 g2D)

 conv3_block2_2_conv (Conv2D)   (None, 28, 28, 128)  147456
['conv3_block2_2_pad[0][0]']

 conv3_block2_2_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block2_2_conv[0][0]']
 ization)

 conv3_block2_2_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block2_2_bn[0][0]']
 n)

 conv3_block2_3_conv (Conv2D)   (None, 28, 28, 512)  66048
['conv3_block2_2_relu[0][0]']

 conv3_block2_out (Add)         (None, 28, 28, 512)  0
['conv3_block1_out[0][0]',

'conv3_block2_3_conv[0][0]']

 conv3_block3_preact_bn (BatchN  (None, 28, 28, 512)  2048
['conv3_block2_out[0][0]']
 ormalization)

 conv3_block3_preact_relu (Acti  (None, 28, 28, 512)  0
['conv3_block3_preact_bn[0][0]']
 vation)

 conv3_block3_1_conv (Conv2D)   (None, 28, 28, 128)  65536
['conv3_block3_preact_relu[0][0]'
                                                                  ]

 conv3_block3_1_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block3_1_conv[0][0]']
 ization)

 conv3_block3_1_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block3_1_bn[0][0]']
 n)
```

```
 conv3_block3_2_pad (ZeroPaddin  (None, 30, 30, 128)  0
['conv3_block3_1_relu[0][0]']
 g2D)

 conv3_block3_2_conv (Conv2D)   (None, 28, 28, 128)  147456
['conv3_block3_2_pad[0][0]']

 conv3_block3_2_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block3_2_conv[0][0]']
 ization)

 conv3_block3_2_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block3_2_bn[0][0]']
 n)

 conv3_block3_3_conv (Conv2D)   (None, 28, 28, 512)  66048
['conv3_block3_2_relu[0][0]']

 conv3_block3_out (Add)         (None, 28, 28, 512)  0
['conv3_block2_out[0][0]',

'conv3_block3_3_conv[0][0]']

 conv3_block4_preact_bn (BatchN  (None, 28, 28, 512)  2048
['conv3_block3_out[0][0]']
 ormalization)

 conv3_block4_preact_relu (Acti  (None, 28, 28, 512)  0
['conv3_block4_preact_bn[0][0]']
 vation)

 conv3_block4_1_conv (Conv2D)   (None, 28, 28, 128)  65536
['conv3_block4_preact_relu[0][0]'
                                                                ]

 conv3_block4_1_bn (BatchNormal  (None, 28, 28, 128)  512
['conv3_block4_1_conv[0][0]']
 ization)

 conv3_block4_1_relu (Activatio  (None, 28, 28, 128)  0
['conv3_block4_1_bn[0][0]']
 n)

 conv3_block4_2_pad (ZeroPaddin  (None, 30, 30, 128)  0
['conv3_block4_1_relu[0][0]']
 g2D)

 conv3_block4_2_conv (Conv2D)   (None, 14, 14, 128)  147456
['conv3_block4_2_pad[0][0]']

 conv3_block4_2_bn (BatchNormal  (None, 14, 14, 128)  512
['conv3_block4_2_conv[0][0]']
 ization)

 conv3_block4_2_relu (Activatio  (None, 14, 14, 128)  0
['conv3_block4_2_bn[0][0]']
 n)
```

```
 max_pooling2d_1 (MaxPooling2D)  (None, 14, 14, 512)  0
['conv3_block3_out[0][0]']

 conv3_block4_3_conv (Conv2D)   (None, 14, 14, 512)  66048
['conv3_block4_2_relu[0][0]']

 conv3_block4_out (Add)         (None, 14, 14, 512)  0
['max_pooling2d_1[0][0]',

'conv3_block4_3_conv[0][0]']

 conv4_block1_preact_bn (BatchN  (None, 14, 14, 512)  2048
['conv3_block4_out[0][0]']
 ormalization)

 conv4_block1_preact_relu (Acti  (None, 14, 14, 512)  0
['conv4_block1_preact_bn[0][0]']
 vation)

 conv4_block1_1_conv (Conv2D)   (None, 14, 14, 256)  131072
['conv4_block1_preact_relu[0][0]'
                                                                 ]

 conv4_block1_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block1_1_conv[0][0]']
 ization)

 conv4_block1_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block1_1_bn[0][0]']
 n)

 conv4_block1_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block1_1_relu[0][0]']
 g2D)

 conv4_block1_2_conv (Conv2D)   (None, 14, 14, 256)  589824
['conv4_block1_2_pad[0][0]']

 conv4_block1_2_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block1_2_conv[0][0]']
 ization)

 conv4_block1_2_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block1_2_bn[0][0]']
 n)

 conv4_block1_0_conv (Conv2D)   (None, 14, 14, 1024  525312
['conv4_block1_preact_relu[0][0]'
                                )                                ]

 conv4_block1_3_conv (Conv2D)   (None, 14, 14, 1024  263168
['conv4_block1_2_relu[0][0]']
                                )

 conv4_block1_out (Add)         (None, 14, 14, 1024  0
['conv4_block1_0_conv[0][0]',
```

```
                                )
'conv4_block1_3_conv[0][0]']

 conv4_block2_preact_bn (BatchN  (None, 14, 14, 1024  4096
['conv4_block1_out[0][0]']
 ormalization)                  )

 conv4_block2_preact_relu (Acti  (None, 14, 14, 1024  0
['conv4_block2_preact_bn[0][0]']
 vation)                        )

 conv4_block2_1_conv (Conv2D)   (None, 14, 14, 256)  262144
['conv4_block2_preact_relu[0][0]'
                                                                ]

 conv4_block2_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block2_1_conv[0][0]']
 ization)

 conv4_block2_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block2_1_bn[0][0]']
 n)

 conv4_block2_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block2_1_relu[0][0]']
 g2D)

 conv4_block2_2_conv (Conv2D)   (None, 14, 14, 256)  589824
['conv4_block2_2_pad[0][0]']

 conv4_block2_2_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block2_2_conv[0][0]']
 ization)

 conv4_block2_2_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block2_2_bn[0][0]']
 n)

 conv4_block2_3_conv (Conv2D)   (None, 14, 14, 1024  263168
['conv4_block2_2_relu[0][0]']
                                )

 conv4_block2_out (Add)         (None, 14, 14, 1024  0
['conv4_block1_out[0][0]',
                                )
'conv4_block2_3_conv[0][0]']

 conv4_block3_preact_bn (BatchN  (None, 14, 14, 1024  4096
['conv4_block2_out[0][0]']
 ormalization)                  )

 conv4_block3_preact_relu (Acti  (None, 14, 14, 1024  0
['conv4_block3_preact_bn[0][0]']
 vation)                        )

 conv4_block3_1_conv (Conv2D)   (None, 14, 14, 256)  262144
['conv4_block3_preact_relu[0][0]'
```

```
                                                                  ]

 conv4_block3_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block3_1_conv[0][0]']
 ization)

 conv4_block3_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block3_1_bn[0][0]']
 n)

 conv4_block3_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block3_1_relu[0][0]']
 g2D)

 conv4_block3_2_conv (Conv2D)   (None, 14, 14, 256)  589824
['conv4_block3_2_pad[0][0]']

 conv4_block3_2_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block3_2_conv[0][0]']
 ization)

 conv4_block3_2_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block3_2_bn[0][0]']
 n)

 conv4_block3_3_conv (Conv2D)   (None, 14, 14, 1024  263168
['conv4_block3_2_relu[0][0]']
                                )

 conv4_block3_out (Add)         (None, 14, 14, 1024  0
['conv4_block2_out[0][0]',
                                )
'conv4_block3_3_conv[0][0]']

 conv4_block4_preact_bn (BatchN  (None, 14, 14, 1024  4096
['conv4_block3_out[0][0]']
 ormalization)                  )

 conv4_block4_preact_relu (Acti  (None, 14, 14, 1024  0
['conv4_block4_preact_bn[0][0]']
 vation)                        )

 conv4_block4_1_conv (Conv2D)   (None, 14, 14, 256)  262144
['conv4_block4_preact_relu[0][0]'
                                                                  ]

 conv4_block4_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block4_1_conv[0][0]']
 ization)

 conv4_block4_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block4_1_bn[0][0]']
 n)

 conv4_block4_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block4_1_relu[0][0]']
 g2D)
```

```
 conv4_block4_2_conv (Conv2D)   (None, 14, 14, 256)  589824
['conv4_block4_2_pad[0][0]']

 conv4_block4_2_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block4_2_conv[0][0]']
 ization)

 conv4_block4_2_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block4_2_bn[0][0]']
 n)

 conv4_block4_3_conv (Conv2D)   (None, 14, 14, 1024  263168
['conv4_block4_2_relu[0][0]']
                                )

 conv4_block4_out (Add)         (None, 14, 14, 1024  0
['conv4_block3_out[0][0]',
                                )
'conv4_block4_3_conv[0][0]']

 conv4_block5_preact_bn (BatchN  (None, 14, 14, 1024  4096
['conv4_block4_out[0][0]']
 ormalization)                  )

 conv4_block5_preact_relu (Acti  (None, 14, 14, 1024  0
['conv4_block5_preact_bn[0][0]']
 vation)                        )

 conv4_block5_1_conv (Conv2D)   (None, 14, 14, 256)  262144
['conv4_block5_preact_relu[0][0]'
                                                                  ]

 conv4_block5_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block5_1_conv[0][0]']
 ization)

 conv4_block5_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block5_1_bn[0][0]']
 n)

 conv4_block5_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block5_1_relu[0][0]']
 g2D)

 conv4_block5_2_conv (Conv2D)   (None, 14, 14, 256)  589824
['conv4_block5_2_pad[0][0]']

 conv4_block5_2_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block5_2_conv[0][0]']
 ization)

 conv4_block5_2_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block5_2_bn[0][0]']
 n)
```

```
 conv4_block5_3_conv (Conv2D)   (None, 14, 14, 1024  263168
['conv4_block5_2_relu[0][0]']
                                )

 conv4_block5_out (Add)         (None, 14, 14, 1024  0
['conv4_block4_out[0][0]',
                                )
'conv4_block5_3_conv[0][0]']

 conv4_block6_preact_bn (BatchN  (None, 14, 14, 1024  4096
['conv4_block5_out[0][0]']
 ormalization)                  )

 conv4_block6_preact_relu (Acti  (None, 14, 14, 1024  0
['conv4_block6_preact_bn[0][0]']
 vation)                        )

 conv4_block6_1_conv (Conv2D)   (None, 14, 14, 256)  262144
['conv4_block6_preact_relu[0][0]'
                                                                  ]

 conv4_block6_1_bn (BatchNormal  (None, 14, 14, 256)  1024
['conv4_block6_1_conv[0][0]']
 ization)

 conv4_block6_1_relu (Activatio  (None, 14, 14, 256)  0
['conv4_block6_1_bn[0][0]']
 n)

 conv4_block6_2_pad (ZeroPaddin  (None, 16, 16, 256)  0
['conv4_block6_1_relu[0][0]']
 g2D)

 conv4_block6_2_conv (Conv2D)   (None, 7, 7, 256)    589824
['conv4_block6_2_pad[0][0]']

 conv4_block6_2_bn (BatchNormal  (None, 7, 7, 256)   1024
['conv4_block6_2_conv[0][0]']
 ization)

 conv4_block6_2_relu (Activatio  (None, 7, 7, 256)   0
['conv4_block6_2_bn[0][0]']
 n)

 max_pooling2d_2 (MaxPooling2D)  (None, 7, 7, 1024)  0
['conv4_block5_out[0][0]']

 conv4_block6_3_conv (Conv2D)   (None, 7, 7, 1024)   263168
['conv4_block6_2_relu[0][0]']

 conv4_block6_out (Add)         (None, 7, 7, 1024)   0
['max_pooling2d_2[0][0]',

'conv4_block6_3_conv[0][0]']

 conv5_block1_preact_bn (BatchN  (None, 7, 7, 1024)  4096
['conv4_block6_out[0][0]']
```

```
 ormalization)

 conv5_block1_preact_relu (Acti  (None, 7, 7, 1024)  0
['conv5_block1_preact_bn[0][0]']
 vation)

 conv5_block1_1_conv (Conv2D)   (None, 7, 7, 512)    524288
['conv5_block1_preact_relu[0][0]'
                                                                  ]

 conv5_block1_1_bn (BatchNormal  (None, 7, 7, 512)   2048
['conv5_block1_1_conv[0][0]']
 ization)

 conv5_block1_1_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block1_1_bn[0][0]']
 n)

 conv5_block1_2_pad (ZeroPaddin  (None, 9, 9, 512)   0
['conv5_block1_1_relu[0][0]']
 g2D)

 conv5_block1_2_conv (Conv2D)   (None, 7, 7, 512)    2359296
['conv5_block1_2_pad[0][0]']

 conv5_block1_2_bn (BatchNormal  (None, 7, 7, 512)   2048
['conv5_block1_2_conv[0][0]']
 ization)

 conv5_block1_2_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block1_2_bn[0][0]']
 n)

 conv5_block1_0_conv (Conv2D)   (None, 7, 7, 2048)   2099200
['conv5_block1_preact_relu[0][0]'
                                                                  ]

 conv5_block1_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624
['conv5_block1_2_relu[0][0]']

 conv5_block1_out (Add)         (None, 7, 7, 2048)   0
['conv5_block1_0_conv[0][0]',

'conv5_block1_3_conv[0][0]']

 conv5_block2_preact_bn (BatchN  (None, 7, 7, 2048)  8192
['conv5_block1_out[0][0]']
 ormalization)

 conv5_block2_preact_relu (Acti  (None, 7, 7, 2048)  0
['conv5_block2_preact_bn[0][0]']
 vation)

 conv5_block2_1_conv (Conv2D)   (None, 7, 7, 512)    1048576
['conv5_block2_preact_relu[0][0]'
                                                                  ]
```

```
 conv5_block2_1_bn (BatchNormal  (None, 7, 7, 512)   2048
['conv5_block2_1_conv[0][0]']
 ization)

 conv5_block2_1_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block2_1_bn[0][0]']
 n)

 conv5_block2_2_pad (ZeroPaddin  (None, 9, 9, 512)   0
['conv5_block2_1_relu[0][0]']
 g2D)

 conv5_block2_2_conv (Conv2D)   (None, 7, 7, 512)    2359296
['conv5_block2_2_pad[0][0]']

 conv5_block2_2_bn (BatchNormal  (None, 7, 7, 512)   2048
['conv5_block2_2_conv[0][0]']
 ization)

 conv5_block2_2_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block2_2_bn[0][0]']
 n)

 conv5_block2_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624
['conv5_block2_2_relu[0][0]']

 conv5_block2_out (Add)         (None, 7, 7, 2048)   0
['conv5_block1_out[0][0]',

'conv5_block2_3_conv[0][0]']

 conv5_block3_preact_bn (BatchN  (None, 7, 7, 2048)  8192
['conv5_block2_out[0][0]']
 ormalization)

 conv5_block3_preact_relu (Acti  (None, 7, 7, 2048)  0
['conv5_block3_preact_bn[0][0]']
 vation)

 conv5_block3_1_conv (Conv2D)   (None, 7, 7, 512)    1048576
['conv5_block3_preact_relu[0][0]'
                                                                ]

 conv5_block3_1_bn (BatchNormal  (None, 7, 7, 512)   2048
['conv5_block3_1_conv[0][0]']
 ization)

 conv5_block3_1_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block3_1_bn[0][0]']
 n)

 conv5_block3_2_pad (ZeroPaddin  (None, 9, 9, 512)   0
['conv5_block3_1_relu[0][0]']
 g2D)

 conv5_block3_2_conv (Conv2D)   (None, 7, 7, 512)    2359296
['conv5_block3_2_pad[0][0]']
```

```
 conv5_block3_2_bn (BatchNormal  (None, 7, 7, 512)   2048
['conv5_block3_2_conv[0][0]']
 ization)

 conv5_block3_2_relu (Activatio  (None, 7, 7, 512)   0
['conv5_block3_2_bn[0][0]']
 n)

 conv5_block3_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624
['conv5_block3_2_relu[0][0]']

 conv5_block3_out (Add)         (None, 7, 7, 2048)   0
['conv5_block2_out[0][0]',

'conv5_block3_3_conv[0][0]']

 post_bn (BatchNormalization)   (None, 7, 7, 2048)   8192
['conv5_block3_out[0][0]']

 post_relu (Activation)         (None, 7, 7, 2048)   0
['post_bn[0][0]']

==============================================================================
=====================
Total params: 23,564,800
Trainable params: 23,519,360
Non-trainable params: 45,440
______________________________________________________________________________
_____________________
```

In [ ]:

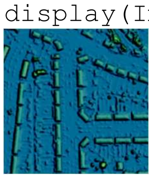
```
# Set the layers.
last_conv_layer_name = "conv5_block3_3_conv"
classifier_layer_names =  ["flatten",
                           "dense",
                           "dropout",
                           "dense_1",
                           "dropout_1",
                           "dense_2"]
```

Now let's load a random image.

In [ ]:

```
# Display
from IPython.display import Image
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
%matplotlib inline

# Get the image in the right size
def get_img_array(img_path, size = (224, 224)):
    import tensorflow as tf
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=size)
    # `array` is a float32 Numpy array of shape (299, 299, 3)
    array = tf.keras.preprocessing.image.img_to_array(img)
```

```
    # We add a dimension to transform our array into a "batch"
    # of size (1, 224, 224, 3)
    array = np.expand_dims(array, axis=0)
    array = preprocess_input(array)
    return array

# Get an image
img_path = '/content/LIDAR/LIDAR_41687.png'
data = get_img_array(img_path)

# Plot it
display(Image(img_path))
```



Let's calculate the GradRAM estimate.

In [ ]:

```
ypred = ResModel.predict(preprocess_input(data))
```

In [ ]:

```
ypred
```

Out[ ]:

```
array([[13.055788]], dtype=float32)
```

In [ ]:

```
# Plot the heatmap!
heatmap = make_gradcam_heatmap(
    data, ResModel, last_conv_layer_name, classifier_layer_names
)

# Display heatmap
plt.matshow(heatmap)
plt.show()
```

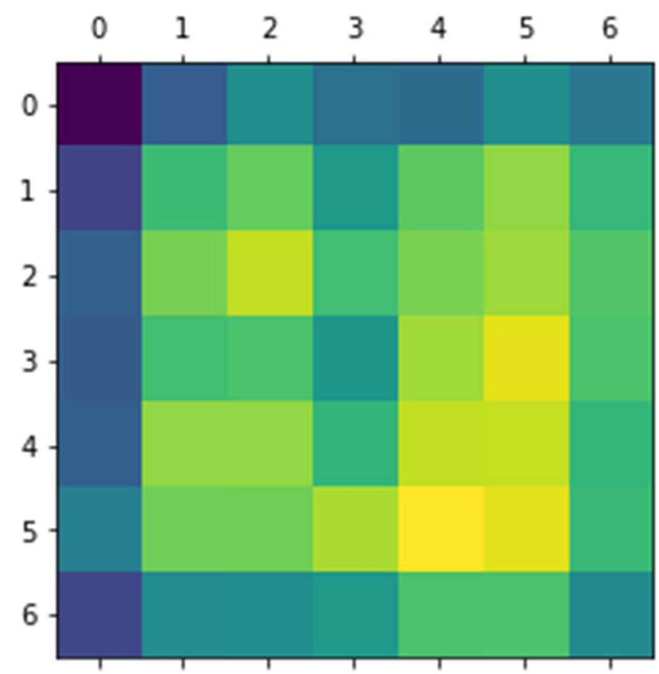
```
<keras.layers.convolutional.Conv2D object at 0x7f2e27577710>
<keras.engine.functional.Functional object at 0x7f2e26eedc50>
Model: "model_41"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_23 (InputLayer)       [(None, 7, 7, 2048)]      0

 flatten (Flatten)           (None, 100352)            0

 dense (Dense)               (None, 64)                6422592

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 64)                4160

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65
```

```
=================================================================
Total params: 6,426,817
Trainable params: 6,426,817
Non-trainable params: 0
_________________________________________________________________
None
last conv layer output: tf.Tensor(
[[[[-4.95564580e-01 -8.55701447e-01 -3.80772352e-01 ... -7.04387307e-01
    -4.58083451e-01 -4.75665033e-01]
   [-3.82043421e-01 -6.06550395e-01 -2.24060223e-01 ... -4.97081757e-01
    -2.10784391e-01 -4.35611784e-01]
   [-2.96298355e-01 -2.19143227e-01 -5.81704155e-02 ... -4.19879287e-01
    -6.66684136e-02 -4.04900908e-01]
   ...
   [-1.54142573e-01 -3.94121110e-01 -7.40435570e-02 ... -2.88498163e-01
    -1.89653441e-01 -3.30726922e-01]
   [-9.54224542e-02 -2.15445161e-01 -5.16480133e-02 ... -3.56693178e-01
    -1.08936496e-01 -2.94976085e-01]
   [-1.51976615e-01 -3.51700723e-01 -1.66533574e-01 ... -5.10419607e-01
    -2.76933342e-01 -3.43488336e-01]]

  [[-3.75018239e-01 -5.58317482e-01 -3.20848227e-01 ... -6.62568748e-01
    -2.09177300e-01 -4.66805279e-01]
   [-3.54437500e-01 -2.48598196e-02  1.64700653e-02 ... -3.91010672e-01
     1.18513554e-01 -2.41792485e-01]
   [-2.99330682e-01  2.55745556e-02  1.03131868e-01 ... -3.04652452e-01
     2.86846720e-02 -4.41023260e-01]
   ...
   [-9.81650054e-02 -9.30884629e-02  2.16373026e-01 ... -2.70678461e-01
    -3.87699418e-02 -2.57151365e-01]
   [-1.84416786e-01 -2.12327719e-01  3.61882001e-01 ... -4.97523248e-01
    -1.39719561e-01  9.19152424e-02]
   [-1.57032743e-01 -2.71744937e-01  5.64760491e-02 ... -3.44674647e-01
    -1.12321310e-01  2.74353866e-02]]

  [[-5.03697217e-01 -4.17797506e-01 -3.20309699e-01 ... -6.81440771e-01
     1.33212477e-01 -5.33723831e-01]
   [-6.61143720e-01 -8.29712301e-02  1.03822134e-01 ... -6.58365071e-01
     5.84970772e-01 -1.07069455e-01]
   [-4.92002457e-01  1.46375284e-01  6.43660203e-02 ... -4.23718393e-01
     5.00325859e-01 -4.52445060e-01]
   ...
   [-2.26827890e-01 -8.80644098e-02  1.95748940e-01 ... -2.82325953e-01
    -1.05296649e-01 -3.28780502e-01]
   [-4.44539309e-01 -2.53253907e-01  4.74628866e-01 ... -5.79513848e-01
    -1.91063762e-01  2.15727746e-01]
   [-2.62258679e-01 -3.13866168e-01  1.54893517e-01 ... -3.77826065e-01
    -7.39969313e-02  2.09710568e-01]]

  ...

  [[-3.92643303e-01 -5.63990176e-01 -1.18541107e-01 ... -4.57109183e-01
    -2.05394462e-01 -2.68065900e-01]
   [-4.36585754e-01 -1.78396910e-01  2.11933747e-01 ... -3.24235260e-01
     2.58099288e-01  8.18234608e-02]
   [-3.89386743e-01  3.32374461e-02 -2.60546356e-02 ...  4.99773165e-03
     3.53361696e-01 -2.62739837e-01]
```

```
    ...
    [-2.92979151e-01  1.45394076e-02  1.36742756e-01 ...  2.33751401e-01
      1.39734432e-01 -2.57683367e-01]
    [-4.94089782e-01 -1.21766090e-01  4.04208094e-01 ...  1.74358130e-01
      7.08106458e-02  6.01879619e-02]
    [-4.07510668e-01 -3.48699540e-01  1.27564669e-01 ...  3.24877985e-02
     -7.42986500e-02  1.07487977e-01]]

   [[-2.14869902e-01 -4.26719666e-01 -4.19756360e-02 ... -3.00215244e-01
     -2.32824296e-01 -3.32093649e-02]
    [-2.84705698e-01 -2.22916260e-01  2.26705641e-01 ... -3.10444981e-01
      5.89935668e-02  2.91723549e-01]
    [-4.36442286e-01 -1.82217464e-01  1.63852558e-01 ... -2.69977480e-01
      3.04996341e-01 -8.59212279e-02]
    ...
    [-3.39290828e-01  4.23270017e-02 -6.19579367e-02 ... -7.43299897e-04
      5.66681385e-01 -2.89863527e-01]
    [-4.54450816e-01 -2.94491112e-01  4.02487628e-02 ... -1.23565160e-01
      4.26278472e-01 -2.71118671e-01]
    [-3.58619958e-01 -3.64736497e-01 -1.23830207e-01 ... -4.30041701e-02
      1.26385540e-01 -1.70594379e-01]]

   [[-3.07996511e-01 -5.54213941e-01 -1.66326031e-01 ... -3.67385983e-01
     -3.85930121e-01 -1.22371398e-01]
    [-2.77022809e-01 -2.67603219e-01 -3.87712009e-02 ... -3.07787955e-01
     -1.09299913e-01  5.99676743e-02]
    [-4.09199357e-01 -2.65462250e-01 -1.44499242e-01 ... -2.09625348e-01
      7.97343105e-02 -3.91349383e-02]
    ...
    [-3.35809380e-01 -1.64965808e-01 -2.34629750e-01 ... -2.18302354e-01
      3.37575406e-01 -2.87553340e-01]
    [-3.14959705e-01 -3.93341780e-01 -1.31505370e-01 ... -1.45671144e-01
      2.98044205e-01 -4.39494371e-01]
    [-2.60136783e-01 -4.31345731e-01 -2.16281950e-01 ... -3.10819566e-01
      1.36962486e-02 -3.88927728e-01]]]], shape=(1, 7, 7, 2048),
dtype=float32)
prediction: tf.Tensor([[3.9638216]], shape=(1, 1), dtype=float32)
gradients: tf.Tensor(
[[[[ 2.5807275e-03 -4.0366611e-04  6.1413394e-03 ...  4.6219877e-03
      5.3425306e-03  1.4770329e-03]
    [-7.3079793e-03 -2.4799202e-03 -1.2160714e-03 ... -5.8021173e-03
     -1.6079048e-03  8.4384484e-04]
    [-2.1847868e-03 -7.2255178e-04 -2.1194387e-03 ...  2.1491228e-03
      5.4530408e-03  4.2355028e-03]
    ...
    [ 7.7539077e-03 -4.6021962e-03  7.9346951e-03 ... -4.9210680e-03
      1.1566026e-03 -3.0182744e-03]
    [-5.0834301e-03  6.4595770e-03 -5.3339079e-04 ... -3.1154838e-03
      1.6639971e-03 -3.4763804e-03]
    [ 5.3743785e-03  4.1206796e-03 -5.8787162e-03 ...  2.1354877e-05
     -2.2620354e-03 -4.4258228e-03]]

   [[-3.1639275e-04  2.9523273e-03 -1.3807417e-03 ... -2.4961014e-03
      4.8689926e-03  1.7020124e-03]
    [ 4.9288422e-03 -3.1184888e-04 -1.1595301e-03 ... -1.4833346e-03
      6.2291529e-03 -5.0981934e-03]
    [ 8.8973027e-03 -1.6781280e-03  3.4484983e-05 ... -1.6009265e-03
```

```
   3.7164143e-03  1.5850749e-03]
  ...
  [-2.1374244e-03  4.5334329e-03 -4.4900109e-03 ... -4.3249046e-03
    1.2240866e-03  1.3504170e-03]
  [ 7.9435529e-05 -2.7126404e-03  1.3356025e-02 ... -3.1608969e-03
    1.5337392e-03 -2.6238300e-03]
  [-4.7897091e-03  6.6276318e-03 -2.6608557e-03 ... -1.2372086e-02
   -8.8964647e-04  5.1777007e-04]]

 [[-2.9682992e-03 -5.7148654e-03 -1.2031866e-03 ... -8.8785053e-04
    2.2034491e-03 -1.8814974e-03]
  [-2.8838739e-03 -3.7314431e-04 -5.4545384e-03 ... -1.9465787e-03
    2.7566950e-03  5.5330424e-03]
  [ 3.0856442e-03  6.0330704e-03  5.2308859e-03 ... -8.9703361e-03
   -3.8592394e-03 -4.0125567e-03]
  ...
  [-4.3626409e-03  2.7272245e-03 -1.5829626e-04 ...  1.2155904e-03
   -5.0993580e-03  7.6629026e-03]
  [-6.4592119e-03  9.2875995e-03  4.6602899e-04 ...  1.2901532e-03
   -3.2318528e-03 -1.0327973e-02]
  [ 5.1463223e-03 -1.3478741e-03  6.6189435e-03 ...  9.7203627e-03
   -5.3613656e-03 -2.0927875e-03]]

 ...

 [[ 6.0011949e-03  8.2584182e-03  3.3545659e-03 ...  2.2355109e-03
    2.1219980e-03 -4.3618036e-03]
  [ 7.5484430e-03  1.9820309e-03  1.3391922e-03 ...  7.7421777e-04
   -2.4773669e-03  2.1868146e-03]
  [ 4.5338408e-03  9.1232304e-03  4.5616711e-03 ...  3.1883882e-03
   -8.2503790e-03  4.6895347e-03]
  ...
  [ 2.3150819e-04 -5.1091975e-03 -1.3392224e-03 ...  1.1809753e-03
    1.4907750e-04  1.3361837e-03]
  [-2.9663972e-03  8.7175341e-03 -4.1727284e-03 ... -7.2226198e-03
   -1.2542635e-02 -3.1244240e-04]
  [-8.0269814e-04  7.3089689e-04 -3.2105041e-03 ... -4.7596951e-04
    3.4707438e-03 -2.6811040e-03]]

 [[-4.9010101e-03 -5.6737848e-04  1.1213834e-04 ... -3.1454051e-03
   -4.8394396e-04  3.4169476e-03]
  [-1.8927548e-03  5.9616566e-03  2.4724077e-03 ... -2.6743619e-03
    3.7348415e-03  2.9198455e-03]
  [ 1.3142738e-03 -1.0421942e-04 -6.1056344e-03 ... -3.1211968e-03
   -7.6391036e-03 -1.6688684e-03]
  ...
  [ 8.3969105e-03 -5.5893390e-03  4.4509135e-03 ...  1.0128791e-03
   -4.1594049e-03  1.1734758e-03]
  [-6.3510470e-06  6.5942854e-04 -1.5204854e-04 ... -3.5311773e-03
   -6.1899195e-03  6.3140364e-03]
  [-1.1467957e-03  1.0183704e-02 -1.4063719e-02 ... -1.0810541e-03
    4.3691951e-03 -1.2811791e-03]]

 [[ 8.3765024e-03  4.5625851e-03  5.2861799e-03 ... -1.6974127e-03
    1.7284924e-03 -8.6963159e-04]
  [ 5.9742383e-03  4.4978438e-03  5.1452657e-03 ...  3.2817072e-03
    7.2007757e-03 -5.0381208e-03]
```

```
   [ 5.5776309e-04 -1.9490951e-03 -2.5135318e-03 ... -1.7766559e-03
    -4.1005574e-03 -3.2981099e-03]
   ...
   [-8.8872155e-03 -7.4286209e-03  9.7149983e-05 ... -6.8429182e-04
    -3.1225993e-03 -9.2680370e-03]
   [-5.1028235e-03  8.1721507e-03  7.0665200e-04 ... -5.3492031e-04
    -9.3890009e-03 -2.3157620e-03]
   [ 6.0484717e-03  1.6192036e-03 -5.2598082e-03 ...  5.6809345e-03
     5.5641411e-03 -6.3047436e-04]]]], shape=(1, 7, 7, 2048), dtype=float32)
[-0.00027804  0.00194855  0.00011103 ... -0.00048219 -0.0004909
 -0.00051791]
[[-1.5276695e-04 -7.4662130e-05 -2.3229713e-05 -5.7361325e-05
  -6.1093240e-05 -2.4606446e-05 -4.8598296e-05]
 [-9.9714249e-05  2.5175519e-05  4.6150970e-05 -1.2867698e-05
   4.2015625e-05  6.3506879e-05  2.0637266e-05]
 [-7.2411080e-05  5.3494885e-05  8.3052590e-05  2.8242517e-05
   5.4995624e-05  6.9294081e-05  3.8012768e-05]
 [-7.6514421e-05  2.8393308e-05  3.3549641e-05 -1.4406305e-05
   7.1363749e-05  9.6174190e-05  3.3005730e-05]
 [-7.2885508e-05  6.4236738e-05  6.3657499e-05  1.8273960e-05
   8.2953513e-05  8.4420899e-05  1.9642886e-05]
 [-4.0655188e-05  4.9903894e-05  5.1235991e-05  7.4077325e-05
   1.0708331e-04  9.5012670e-05  2.4446936e-05]
 [-9.7253054e-05 -2.7691567e-05 -2.4398956e-05 -1.3380743e-05
   3.2584478e-05  3.3051194e-05 -3.0754149e-05]]
```

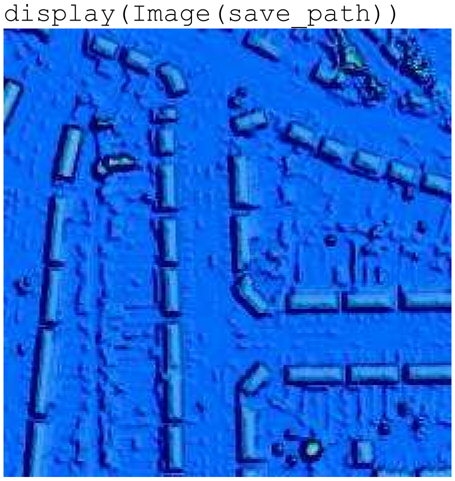


And finally let's superimpose the heatmap of the image.

In [ ]:

```
# We load the original image
img = keras.preprocessing.image.load_img(img_path)
img = keras.preprocessing.image.img_to_array(img)

# We rescale heatmap to a range 0-255
```

```
heatmap = np.uint8(255 * heatmap)

# We use jet colormap to colorize heatmap
jet = cm.get_cmap("jet")

# We use RGB values of the colormap
jet_colors = jet(np.arange(256))[:, :3]
jet_heatmap = jet_colors[heatmap]

# We create an image with RGB colorized heatmap
jet_heatmap = keras.preprocessing.image.array_to_img(jet_heatmap)
jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
jet_heatmap = keras.preprocessing.image.img_to_array(jet_heatmap)

# Superimpose the heatmap on original image
superimposed_img = jet_heatmap * 0.4 + img
superimposed_img = keras.preprocessing.image.array_to_img(superimposed_img)

# Save the superimposed image
save_path = "Img_Example.jpg"
superimposed_img.save(save_path)

# Display Grad RAM
display(Image(save_path))
```



GradCAM for VGG

We will visualize the learning, to detect exactly what is happening.

Its a method that allows visualizing how one image activates the neural network. Basically we will look for the direction that the model used to get to its decisions.

In [ ]:

```
# Imports
import numpy as np
import tensorflow as tf
from tensorflow import keras
```

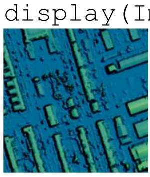
```
# Display
from IPython.display import Image
import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline
```

In [ ]:

```
# Get the image in the right size
def get_img_array(img_path, size = (224, 224)):
    import tensorflow as tf
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=size)
    # `array` is a float32 Numpy array of shape (299, 299, 3)
    array = tf.keras.preprocessing.image.img_to_array(img)
    # We add a dimension to transform our array into a "batch"
    # of size (1, 224, 224, 3)
    array = np.expand_dims(array, axis=0)
    array = preprocess_input(array)
    return array
```

In [93]:

```
# Get an image
img_path = '/content/LIDAR/LIDAR_40500.png'
data = get_img_array(img_path)

# Plot it
display(Image(img_path))
```



In [94]:

```
# The explainer. Gotten from https://keras.io/examples/vision/grad_cam/
def make_gradcam_heatmap(
    img_array, model, last_conv_layer_name, classifier_layer_names
):
    from tensorflow import keras
    import tensorflow as tf
    # First, we create a model that maps the input image to the activations
    # of the last conv layer
    last_conv_layer = model.get_layer(last_conv_layer_name)
    last_conv_layer_model = keras.Model(model.inputs, last_conv_layer.output)

    # Second, we create a model that maps the activations of the last conv
    # layer to the final class predictions
    regression_input = keras.Input(shape=last_conv_layer.output.shape[1:])
    x = regression_input
    for layer_name in classifier_layer_names:
        x = model.get_layer(layer_name)(x)
    classifier_model = keras.Model(regression_input, x)

    # Then, we compute the gradient of the top predicted class for our input
image
    # with respect to the activations of the last conv layer
```

```
    with tf.GradientTape() as tape:
        # Compute activations of the last conv layer and make the tape watch
it
        last_conv_layer_output = last_conv_layer_model(img_array)
        tape.watch(last_conv_layer_output)
        # Compute class predictions
        preds = classifier_model(last_conv_layer_output)
        top_pred_index = tf.argmax(preds[0])
        top_class_channel = preds[:, top_pred_index]

    # This is the gradient of the top predicted class with regard to
    # the output feature map of the last conv layer
    grads = tape.gradient(top_class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted
class
    last_conv_layer_output = last_conv_layer_output.numpy()[0]
    pooled_grads = pooled_grads.numpy()
    for i in range(pooled_grads.shape[-1]):
        last_conv_layer_output[:, :, i] *= pooled_grads[i]

    # The channel-wise mean of the resulting feature map
    # is our heatmap of class activation
    heatmap = np.mean(last_conv_layer_output, axis=-1)

    # For visualization purpose, we will also normalize the heatmap between 0
& 1
    heatmap = np.maximum(heatmap, 0) / np.max(heatmap)
    return heatmap
```

In [95]:

```
# Print the predictions
preds = CBModel.predict(preprocess_input(data/255))
print(preds)
[[6.3750825]]
```

In [ ]:

```
CBModel.summary()
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856     
```

```
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 128)               3211392

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 128)               16512

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 1)                 129

=================================================================
Total params: 17,942,721
Trainable params: 7,947,649
Non-trainable params: 9,995,072
_________________________________________________________________
```

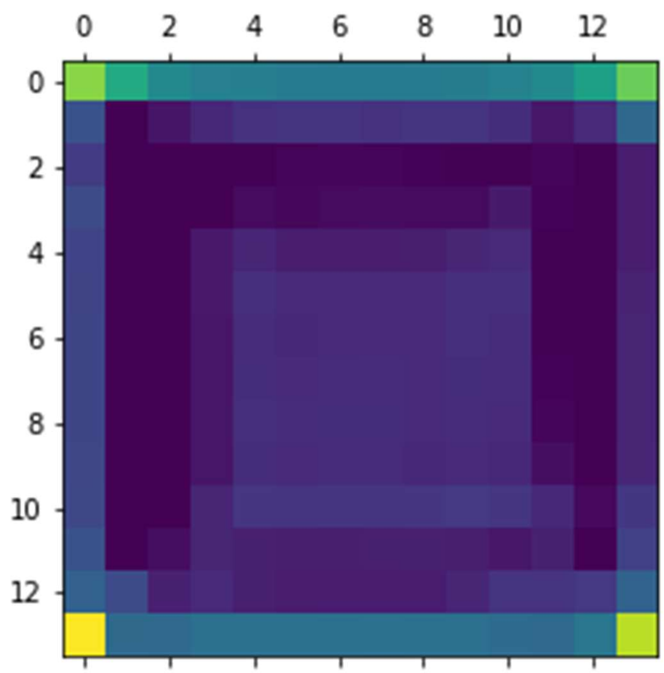
In [96]:

```
# Set the layers.
last_conv_layer_name = "block5_conv3"
classifier_layer_names =  ["block5_pool",
                           "flatten",
                           "dense",
                           "dense_1",
                           "dense_2",]

# classifier_layer_names =  ["block5_pool","flatten",
#                             "dense",
```

```
#                              "dropout",
#                              "dense_1",
#                              "dropout_1",
#                              "dense_2",]
```

In [97]:

```
# Plot the heatmap!
heatmap = make_gradcam_heatmap(
    preprocess_input(data/255), CBModel, last_conv_layer_name,
classifier_layer_names
)

# Display heatmap
plt.matshow(heatmap)
plt.show()
```



Now, to really visualize what's going on, we will superimpose the heatmap over the input. The following code does just that.

In [99]:

```
# We load the original image
img = keras.preprocessing.image.load_img(img_path)
img = keras.preprocessing.image.img_to_array(img)

# We rescale heatmap to a range 0-255
heatmap = np.uint8(255 * heatmap)

# We use jet colormap to colorize heatmap
jet = cm.get_cmap("jet")

# We use RGB values of the colormap
jet_colors = jet(np.arange(256))[:, :3]
jet_heatmap = jet_colors[heatmap]
```

```python
# We create an image with RGB colorized heatmap
jet_heatmap = keras.preprocessing.image.array_to_img(jet_heatmap)
jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
jet_heatmap = keras.preprocessing.image.img_to_array(jet_heatmap)


# Superimpose the heatmap on original image
superimposed_img = jet_heatmap * 0.4 + img
superimposed_img = keras.preprocessing.image.array_to_img(superimposed_img)

# Save the superimposed image
save_path = "Img_Example.jpg"
superimposed_img.save(save_path)

# Display Grad CAM
display(Image(save_path))
```