Data Mining

Deep Learning Assignment Alternative, 2019

Henry Chin | s3774122

RMIT University

Classification of Houses and Flats Through Deep Learning

Project Overview

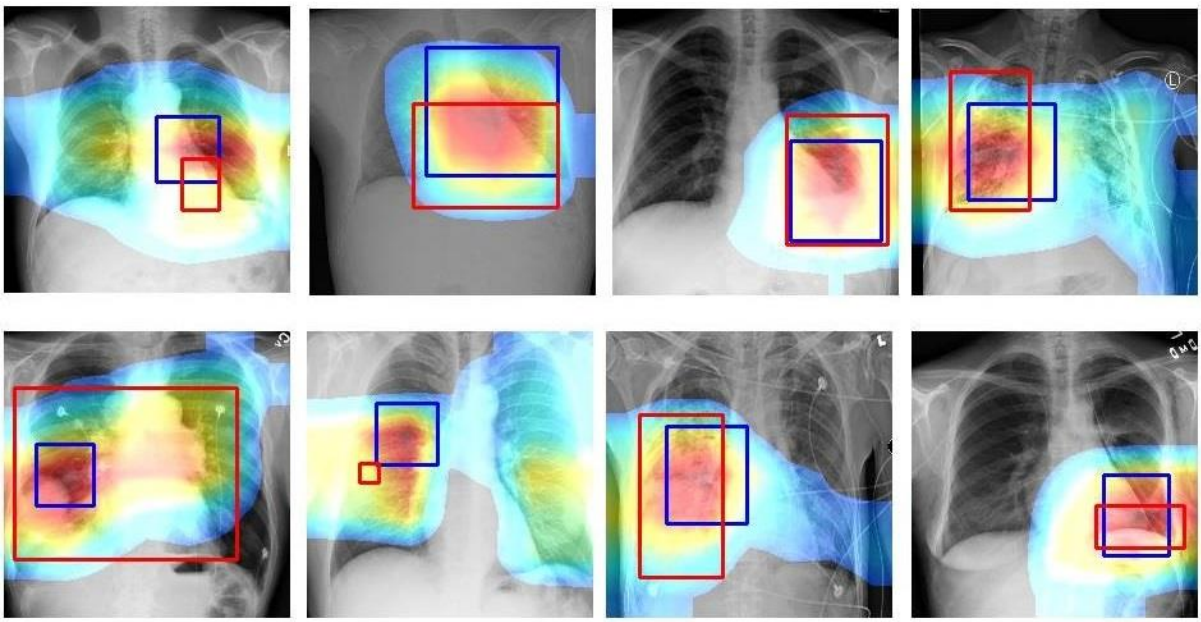
Computationally recognising images of houses and flats is a challenge and a problem known as computer vision. Several attempts have been made within this area, notably the MNIST database for handwriting and fashion items (Mahapatra 2018) but no one has yet applied computer vision to our specific problem.

The goal of this project is to discriminate between houses and flats from 2-dimensional image data.

Problem Statement

In this project, we were given real world dataset and to examine how deep learning, a subset of machine learning algorithms that is well suited to recognising patterns can help us classify images. If the project succeeds, the model can be used by professionals such as surveyors to calculate housing density within a given area in company with other imaging tools like Google Maps or drone mapping.

The method determined is through a deep learning approach. A convolutional neural network (CNN) is used, as the convolutional property simulates human vision and it has been applied successfully to other computer vision problems such as autonomous vehicles, Google Translate application, facial recognition, healthcare, real-time sports tracking and manufacturing.



Visualisation of heat maps with predictions on an X-ray image

Steps needed:

1. Obtain dataset with quality annotations
2. Clean-up and pre-processing of dataset
3. Design architecture for the CNN
4. Training of CNN
5. Testing the model
6. Analysis of results

There were several iterations for steps 3 – 6.

Question and Hypothesis

Attempting to classify a set of 2-D images that represents houses and flats. These series of images were obtained from real estate website. A deep learning model is utilized to address this problem and a CNN is implemented.

Data Exploration

Image dataset obtained had 59,270 images, along with a CSV file that contained tabular data such as property ID, state and region where the property was from and its type.

Sample rows

property-labels

property_id	state	region	property_type
10000149	VIC	other	house
10000657	SA	metro	flat
10000808	SA	metro	house
10001015	VIC	other	house
10001284	SA	metro	house
10001431	SA	metro	house
10001956	SA	metro	house
10002098	SA	metro	house
10002564	SA	other	house

1030748	ACT	other	flat
10307540	VIC	metro	flat
10307630	VIC	metro	flat
10307735	SA	other	house
10307918	SA	other	house
10307995	VIC	metro	house
10308054	VIC	metro	flat
10308330	SA	other	house
10308405	SA	other	house
10308458	VIC	metro	flat

Sample images






Potential data contamination

Upon initial inspection of data, there were a notable number of interior photos. It is approximated that 35% of data images to be interior photos. Unfortunately, attempts at filtering interior photos with published methods, for instance Google Vision and AWS Image Rekognition were unsuccessful. These API offered powerful pre-trained machine learning models that assign labels to images and can be used to classify them into predefined categories. This included object detection, reading printed and handwritten text, detecting facial attributes and general image attributes.

Due to time constrains and limited resources (Google Vision charges a fee per thousand images being classified), we are unable to utilise the API to sort our images entirely. However, we were able to extract predefined labels from some of the dataset.


```
{"responses":[{"labelAnnotations":
[{"mid":"/m/06ht1","description":"Room","score":0.9627714,"topicality":0
.9627714},{"mid":"/m/0c_jw","description":"Furniture","score":0
.958795,"topicality":0.958795},{"mid":"/m/03f6tq","description":"Living
room","score":0.94420177,"topicality":0
.94420177},{"mid":"/m/02rfdq","description":"Interior
design","score":0.9428671,"topicality":0
.9428671},{"mid":"/m/05wrt","description":"Property","score":0
.9275872,"topicality":0
.9275872},{"mid":"/m/01l0mw","description":"Home","score":0
.8954911,"topicality":0
.8954911},{"mid":"/m/03jm5","description":"House","score":0
.8644397,"topicality":0
.8644397},{"mid":"/m/0cgh4","description":"Building","score":0
.8629044,"topicality":0
.8629044},{"mid":"/m/02dgv","description":"Door","score":0
.86171734,"topicality":0.86171734},{"mid":"/m/078n6m","description":"Coffee
table","score":0.8320505,"topicality":0
```


JSON file containing label annotations on one of the images from dataset (Google Vision API)

Data pre-processing

Data pre-processing can be divided into two parts. The first was done during exploratory data analysis. We discovered invalid 'property_type' labels in our CSV file by reason of being labelled as 'BADPROPTYPE' rather than 'house' or 'flat'. There was a total of 128 invalid files and because this number is minimal, we decided to discard them. Data images which were linked to the invalid files were likewise deleted. The primary problem we encountered was that data images was too large and of different sizes.

	0	1	2
count	1000.000000	1000.000000	1000.0
mean	549.033000	773.833000	3.0
std	44.922711	35.328136	0.0
min	298.000000	447.000000	3.0
25%	512.000000	768.000000	3.0
50%	512.000000	768.000000	3.0
75%	600.000000	800.000000	3.0
max	600.000000	800.000000	3.0

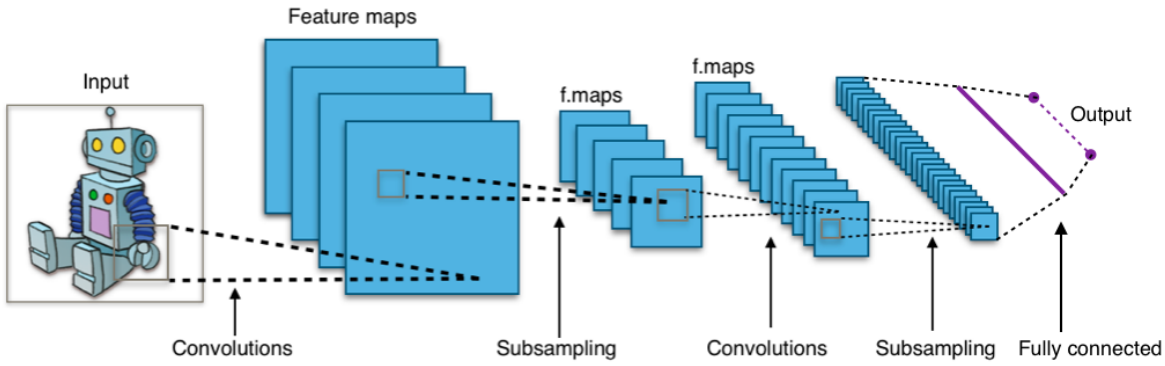
The second pre-processing step was done during training where data is split into training and validation set. 20% of the images of the training set are used for validation. Each base model provided by Keras offers a pre-processing function with specific pre-processing steps for this model. The pre-processing step is applied to an Image Generator which loads the images for training and model evaluation. This is found to be very memory efficient as Jupyter Notebook was not able to handle the large dataset.

Due to images differing in sizes, images had to be rescaled to fit the model. Other solutions were taken cropping images into identical sizes or stretching images to fit the largest image, all of which resulted in information loss. With the range of values for images obtained, it was decided resizing would be the best way forward. It would also be beneficial to measure the model's performance based on different image sizes. A 100% image size (largest size of image data) proved to be too time-consuming during training, so they were ultimately rescaled to 128 x 128 px. Images were of true colours and was converted to grayscale as training time will be reduced further while having little or minimal impact on training accuracy.

Original image (left) and augmented image (right)

Techniques

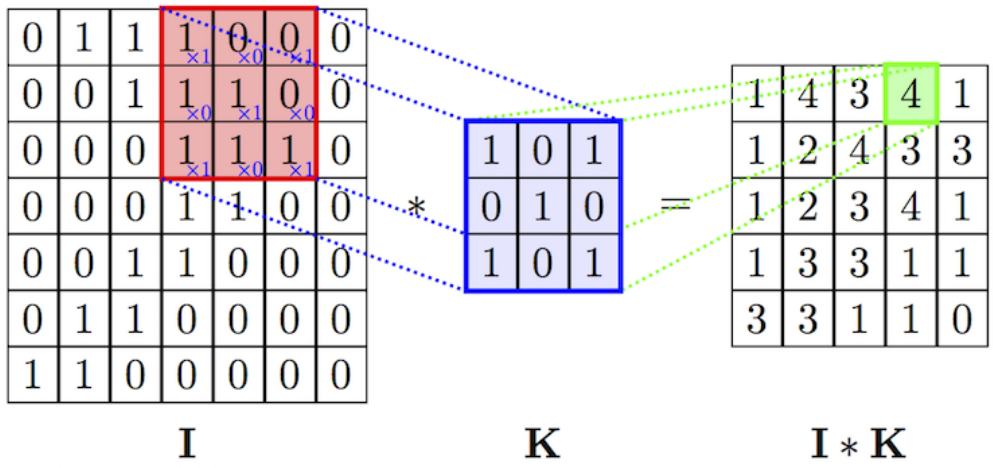
A Convolutional Neural Network (CNN) will be used to solve image classification. They are deep neural networks inspired by biological processes and most commonly applied to analysing visual imagery. It consists of an input and output layer and several hidden layers in between. The hidden layers are typically a convolutional layer followed by a pooling layer.



Structure of a typical CNN for image classification

Convolutional Layer

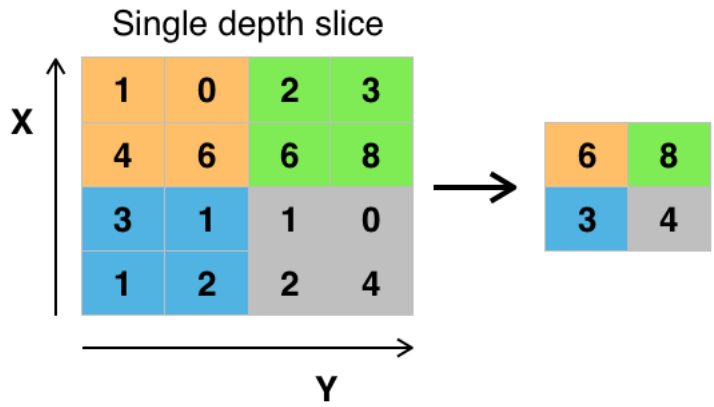
The purpose of the convolutional layer is to extract features from an input image by preserving local spatial signals.



A convolutional operation to extract features

Pooling Layer

Convolutional networks may also include pooling layers. These layers combine outputs of neuron clusters at one layer into a single neuron in the next layer. This is done to reduce memory usage and increase execution speed as well as to prevent overfitting.



MaxPooling layer, that extracts the maximum value in a region to reduce information

Fully Connected Layer

After multiple layers of convolutional and pooling layers, a fully connected layer completes the network. The fully connected layer is a traditional multilayer perceptron responsible for the classification task.

Accuracy

Accuracy is used to compare qualitative results. It is simply the ratio of correct predictions during training.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN}$$

$$TP : TruePositives, TN : TrueNegatives, FN : FalseNegatives, FP : FalsePositive$$

Model Design of the CNN

The model proposes a pattern of convolutional layer with number of filters that increase with the depth of the model, starting with 128 filters for the first layer, 64 filters for the next four layers and finally 32 filters for the last layer. Fourth convolution layer and onwards are followed by max pooling layers and to address overfitting, the dropout method was used between fully connected layers. Finally, output feature maps are flattened and fed to a number of fully connected layers for interpretation and a final prediction.

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

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 128)     1280
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 126, 126, 128)     147584
_________________________________________________________________
dropout_1 (Dropout)          (None, 126, 126, 128)     0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 126, 126, 64)      73792
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 124, 124, 64)      36928
_________________________________________________________________
dropout_2 (Dropout)          (None, 124, 124, 64)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 124, 124, 64)      36928
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 122, 122, 64)      36928
_________________________________________________________________
dropout_3 (Dropout)          (None, 122, 122, 64)      0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 122, 122, 64)      36928
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 120, 120, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 60, 60, 64)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 60, 60, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 60, 60, 64)        36928
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 58, 58, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 19, 19, 64)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 19, 19, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 19, 19, 32)        18464
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 17, 17, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 32)          0
_________________________________________________________________
dropout_6 (Dropout)          (None, 5, 5, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               410112
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
=================================================================
Total params: 920,002
Trainable params: 920,002
Non-trainable params: 0
_________________________________________________________________
```

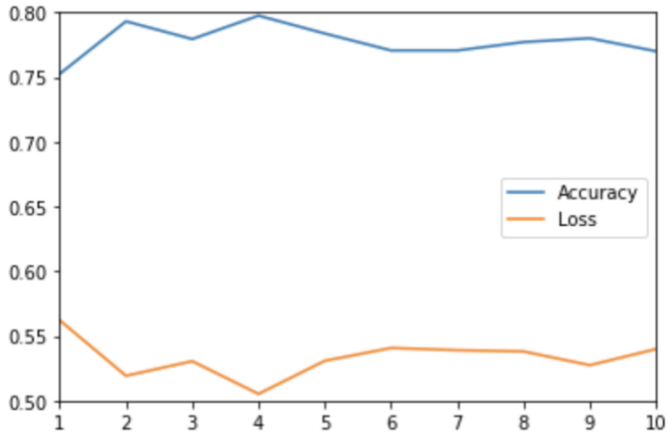
Refinement

Training is done iteratively. The model is initially trained with a thousand samples with default values of parameters. Parameters used for model refinement were values of filters in Keras Conv2D, pooling size and number of epochs. Dimensions of samples were initially kept at 125 x 125px. Then, the model is trained with more samples at higher resolution and parameters are further fine-tuned. Test accuracy is then compared until a sufficient model accuracy was reached.

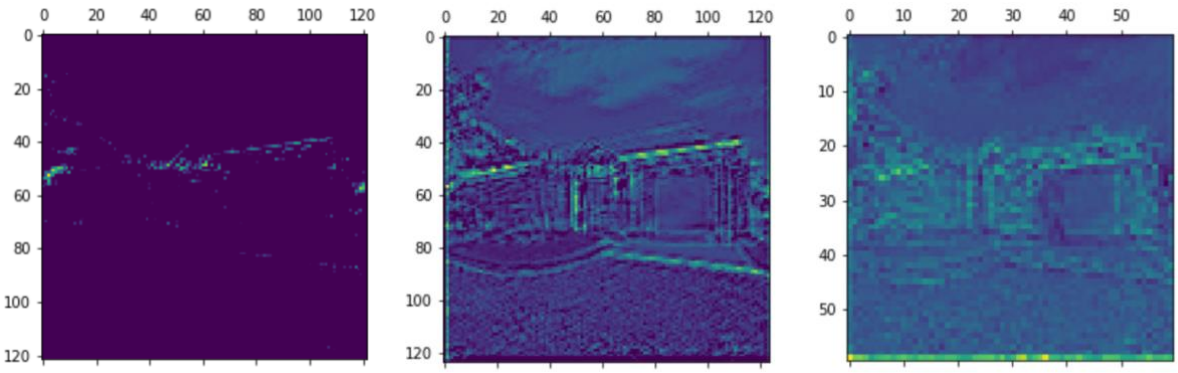
Results

The model had a success rate of 77.51 % in discriminating between houses and flats on a dataset of 59142 images. To our understanding, this is the first application of a CNN to this specific image recognition problem.

To investigate model performance, we obtained accuracy and loss per epoch as preliminary metrics of quality. Instead of a steady increase in accuracy and a steady decrease in loss as would be expected from a well-trained model, we observed that accuracy and loss were unstable over epochs.



Activation maps were used to investigate the cause of these inaccuracies. This method allows visualisation of the changes in state of the activation function of the network per layer. Several layers were extracted for a test sample and plotted. These imply that at least for this image, the outline of the house is the focus of the model. However, no subtle distinguishing features which might further inform the model were observed within the scope of available information.



Justification and Improvement

We believe large amounts of interior photos could have affected training and if they were to be removed, there is a possibility of achieving better outcomes. Image augmentation can be performed to increase dataset, although not every type of augmentation can be used, e.g. vertical flipping would not make sense for this specific data. Another improvement for the project would be to do training on Google's AutoML, comparing and testing results and further adjusting our model accordingly. Images should also be at higher resolution or as vector format if possible.

Future Direction and Discussion

Due to time and resource constrains, there were several methods that were not implemented. Firstly, we considered data augmentation to further increase dataset size as this might have improved data accuracy. Images can be flipped, shifted, skewed, zoomed and with added random brightness augmentation. By artificially increasing number of training data, it helps reduce overfitting and could improve generalization.

We also considered an adversarial attack to make the model more robust. Poisoning of data images is an attack to purposely manipulate the data which degrades the classification performance of learning algorithms. With this, we are able to design model architectures that are more resilient against these threats (Munoz-Gonzales et al. 2017, p. 37).

Furthermore, additional real-world data for testing would improve our model further. It might have been possible to obtain images from other real estate websites. A third category of data that is not either houses or flats but shows empty land can be used as control. This might have increased accuracy of the model. There is also a possibility of interior photos in our dataset increasing model accuracy due to subtle features that we might have overlooked although more testing must be conducted to prove this.

When obtaining subjective metrics for assessing model performance, the existing activation maps could be supplemented with saliency and occlusion maps as well. Viewing data from multiple perspectives may assist in identifying the causes for a model's poor performance.

Potential future works is the opportunity to classify houses based on regions such as houses in the state of Victoria or in Sydney. This can be expanded to other parts of the world such as China, where their housing architecture differs significantly than in Australia.

CONCLUSION

Our model achieved a final accuracy of 0.7751 with 59,142 images. While not ideal, this model represents a first step to developing a more sophisticated model for discriminating between houses and flats. There is significant room for improvement both by data pre-processing and by refining other aspects of the mode.

References

Mahapatra 2018, *A simple 2D CNN for MNIST digit recognition,* Towards Data Science, Medium, viewed 25 September 2019, < https://towardsdatascience.com/a-simple-2d-cnn-for-mnist-digit-recognition-a998dbc1e79a>

Munoz-Gonzalez, L, Biggio, B, Demontis, A, Paudice, A, Wongrassamee, V, C. Lupu, Roli F 2017, 'Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization', *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security,* vol 1, pp. 27-38.

Appendix

In [22]:
```
!pip install --upgrade pip
!pip install --upgrade google-cloud-storage
!pip install --upgrade google-cloud-vision
!pip install --upgrade keras
!pip install matplotlib
```

...

In [7]:
```
!gsutil ls gs://houseflat
!gsutil cp gs://houseflat/datamodifiedname.zip datamodifiedname.zip
!unzip datamodifiedname.zip
```

...

In [39]:
```
%matplotlib inline

import os
import glob
import matplotlib
import pandas as pd
import matplotlib.image as mpimg
import numpy as np
from matplotlib import pyplot as plt
from keras import models
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint
from PIL import Image
from sklearn.model_selection import train_test_split
```

```
/jet/var/python/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of
issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(floa
t).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [3]:
```
check = os.listdir('/jet/prs/workspace/datamodifiedname')
check
```

...

In [55]:
```
images = os.listdir('/jet/prs/workspace/datamodifiedname')
images = ["".join(['/jet/prs/workspace/datamodifiedname/', filename]) for filename in images]

mylist = []

for image in images:
    img = image.load_img(image)
    img = image.img_to_array(img)
    mylist.append(img)
```

In [57]:
```
mylist_values = pd.DataFrame([x.shape for x in mylist])
```

In [58]:
```
mylist_values.describe()
```

...

In [4]:
```
labels = pd.read_csv('/jet/prs/workspace/property-labels.csv')
labels.set_index('property_id', inplace = True)
labels = pd.DataFrame(labels['property_type'].replace("flat", 0).replace("house", 1))
labels.shape

less_labels = labels
house = less_labels[less_labels["property_type"] == 1]
flat = less_labels[less_labels["property_type"] == 0]
labels_house_flat = pd.concat([house, flat])
labels_house_flat.shape
```

Out[4]:
```
(59142, 1)
```

In [5]:
```
labels_house_flat.reset_index(inplace=True)
sortedlabels_house_flat = labels_house_flat.sort_values("property_id")
#sortedlabels_house_flat.reset_index(inplace=True)
sortedlabels_house_flat
```

...

In [6]:
```
sortedlabels_house_flat['property_id'] = sortedlabels_house_flat['property_id'].astype(str)
sortedlabels_house_flat['property_id'] = sortedlabels_house_flat['property_id'] + '.jpeg'
sortedlabels_house_flat
```

...

In [7]:
```
sortedlabels_house_flat['property_type'] = sortedlabels_house_flat['property_type'].astype(str)
sortedlabels_house_flat['property_type']
```

...

In [8]:
```
#sortedlabels_house_flat.reset_index(inplace=True)
sortedlabels_house_flat
```

...

In [9]:
```
folder = '/jet/prs/workspace/datamodifiedname'
```

In [10]:
```
train_datagen = ImageDataGenerator(rescale = 1./255, validation_split=0.2)

train_generator = train_datagen.flow_from_dataframe(
    dataframe=sortedlabels_house_flat,
    directory=folder,
    x_col='property_id',
    y_col='property_type',
    target_size=(128, 128),
    batch_size=32,
    color_mode='grayscale',
    class_mode='binary',
    subset='training',
    validate_filenames=False)

validation_generator = train_datagen.flow_from_dataframe(
    dataframe=sortedlabels_house_flat,
    directory=folder,
    x_col='property_id',
    y_col='property_type',
    target_size=(128, 128),
    batch_size=32,
    color_mode='grayscale',
    class_mode='binary',
    subset='validation',
    validate_filenames=False)
```

In [11]:
```
model = Sequential()
# model.add(Conv2D(64, kernel_size=(3, 3),activation='relu',input_shape=(128,128,1)))
# model.add(Conv2D(64, (3, 3), activation='relu'))
# model.add(MaxPooling2D(pool_size=(3, 3)))
# model.add(Dropout(0.5))
# model.add(Flatten())
# model.add(Dense(128, activation='relu'))
# model.add(Dropout(0.5))
# model.add(Dense(2, activation='sigmoid'))

model.add(Conv2D(128, (3, 3), padding='same', activation='relu',input_shape=(128,128,1)))
model.add(Conv2D(128, (3, 3), activation='relu'))
#model.add(MaxPooling2D(pool_size=(1, 1)))
model.add(Dropout(0.5))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
#model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
#model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.5))

model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))
```

In [12]:
```
model.compile(loss='sparse_categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 128)     1280
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 126, 126, 128)     147584
_________________________________________________________________
dropout_1 (Dropout)          (None, 126, 126, 128)     0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 126, 126, 64)      73792
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 124, 124, 64)      36928
_________________________________________________________________
dropout_2 (Dropout)          (None, 124, 124, 64)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 124, 124, 64)      36928
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 122, 122, 64)      36928
_________________________________________________________________
dropout_3 (Dropout)          (None, 122, 122, 64)      0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 122, 122, 64)      36928
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 120, 120, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 60, 60, 64)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 60, 60, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 60, 60, 64)        36928
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 58, 58, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 19, 19, 64)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 19, 19, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 19, 19, 32)        18464
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 17, 17, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 32)          0
_________________________________________________________________
dropout_6 (Dropout)          (None, 5, 5, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               410112
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
=================================================================
Total params: 920,002
Trainable params: 920,002
Non-trainable params: 0
_________________________________________________________________
```

In [41]:
```
checkpointer = ModelCheckpoint(filepath='/jet/prs/workspace/best_weights.hdf5', monitor ='acc', verbose=1, save_best_on
```

In [49]:
```
history = model.fit_generator(train_generator,
                                  steps_per_epoch = 1400,
                                  epochs = 10,
                                  callbacks=[checkpointer],
                                  validation_data = validation_generator,
                                  validation_steps = 50)
model.save('/jet/prs/workspace/my_model.h5')
```

...

In [43]:
```
#from keras.models import load_model
#model = load_model('/jet/prs/workspace/my_model.h5')
import pickle

with open('/jet/prs/workspace/history.cpickle', 'wb') as history_path:
    pickle.dump(history.history, history_path)

with open('/jet/prs/workspace/history.cpickle', 'rb') as history_path:
    past = pickle.load(history_path)

past
```

...

In [ ]:
```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

In [44]:
```
from keras.models import load_model
model = load_model('/jet/prs/workspace/my_model.h5')
layer_outputs = [layer.output for layer in model.layers]
```

Out[44]:
```
<keras.engine.sequential.Sequential at 0x7f9e269fa710>
```

In [45]:
```
img_path = '/content/datamodifiedname/4639666.jpeg'
img = image.load_img(img_path, target_size=(128, 128), color_mode='grayscale')
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)
img_tensor /= 255.
plt.imshow(img_tensor[0])
plt.show()
print(img_tensor.shape)
```

...

In [ ]:
```
first_layer_activation = activations[6]
print(first_layer_activation.shape)
```

In [ ]:
```
plt.matshow(first_layer_activation[0, :, :, 4], cmap='viridis')
```

Acknowledgements

Evolutionary Computing and Machine Learning (ECML) Group at RMIT