DEEP LEARNING IMAGE ANALYSIS FOR DISASTER RECOVERY, A DATAKIND REPORT FOR THE WORLD BANK GFDRR

PATRICK DOUPE

ABSTRACT

We discuss how the World Bank can use machine learning and satellite images to improve disaster relief efforts. We include a review of image analysis with convolutional neural networks. These networks are illustrated with code examples using the `Keras` deep learning library.

1 INTRODUCTION

The capacity for computers to take images and return useful information has grown over recent years. We first trained models detect numbers, then to distinguishing between cats and dogs. Recently, we have started to segmenting images by objects. In this article, we review this literature ask *how can we use images and deep learning to identify areas at risk in a crisis*.

We do this in two ways. First, by presentiting an intuitive understanding of various deep learning models and model types; second, by presenting applications of models using these methods and satellite images to generate insights. The philosophy of this review is not that the reader should expect to know how to build working prototypes, rather to understand them in sufficient detail so that they can better collaborate with trained researchers. For a more detailed treatment of deep learning, try [1].

We also provide `Python` code and links to simple tutorials so that the reader can obtain a feel for how these things work. This language is standard in both research and production of deep learning models.

If `Python` is new to you, we recommend a tutorial by two leading economists [2].

We also present some applications of this, in industry and in the NGO sector. Much of the NGO sector applications use traditional Geographic Information System (GIS) approaches like change analysis. Although a useful tool, this is outside the scope of this report.[1]

1.1 Deep learning Frameworks

In addition to many languages, there are many different deep learning libraries (or frameworks). We focus in this review on `Keras`. We make this choice because of `Keras`' ease of use.

There are many other libraries and frameworks. It is difficult to get hard numbers about relative usage of these because much usage is in industry. Still, there are rough audiences for each library. `TensorFlow` is useful in research and production. There is a high level of boilerplate required and this is not for beginners. In fact, `TensorFlow` recommonds the use of `Keras` for beginners.[2] `PyTorch` markets itself for fast, flexible experimentation. Version 1.0 will provide support for building models in production. Both `TensorFlow` and `PyTorch` are about as fast as each other, and both are faster than `Keras` [3]. Other languages include the popular `Caffe`, Microsoft's `CNTK` and Apache's `MXNet`.

Figure 1 is an example linear regression model with ten explanatory variables. We begin with importing some objects: an `Input` object which defines the shape of the explanatory varables; a `Dense` object which maps the data from the input shape to the output shape; a `Model` object which combines the two. Short of comment lines and importing data, we can run a regression model in under ten lines. Although ten lines is ten times `R`'s `lm` command, turning these ten lines into deep learning is not much more effort.

2 CONVOLUTIONAL NEURAL NETWORKS FOR COMPUTER VISION TASKS

Why don't we use linear regression? An image is a three dimensional matrix of numbers. One dimension is the width (W) of the image, one is the height (H) of the image, and we typically have red, green and blue spectral bands. That is, we have $H \times W \times 3$ numbers in an image.

[1] For a tutorial on environmental change detection, try https://media.asf.alaska.edu/uploads/pdf/qgis_Environmental_Change_detection_v2.pdf

[2] https://www.tensorflow.org/tutorials/

```
from keras.layers import Input
from keras.layers import Dense
from keras.models import Model

# import data
...

# set amount of explanatory (x) variables
inputs = Input(shape=(10,))
# set outcome (y) variable
predictions = Dense(1, activation='linear')(inputs)
# define the model
model = Model(inputs=inputs, outputs=predictions)
# set loss function and optimizer
model.compile(optimizer='rmsprop',
              loss='rmse',
              metrics=['accuracy'])
# starts training
model.fit(X, y)
```

Figure 1: A linear regression model in `Keras`. The input is the explanatory variables as a 10 column vector. Models need to be compiled with loss functions and optimizers. We will explain these and other terms in subsequent sections.

```
...
H = ...           # height of images
W = ...           # width
num_classes =     # number of classes to classify
inputs = Input(shape=(W * H * 3,))
# set outcome (y) variable
predictions = Dense(num_classes,
                    activation='sigmoid')(inputs)
...
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
...
```

Figure 2: Using a linear regresion for object detection. The inputs now have a dimension equal to the height $\times$ width $\times$ 3 (one for red, green and blue channels). We also change the loss function from root mean squared error loss to categorical cross entropy for our classification problem.

We can flatten the three dimensions to one and feed this into a multinomial logistic regression model. The model can be implemented by making the code changes in Figure 2.

While this may work OK on some problems, we can do better by exploiting characteristics of image data. Images are different from other datasets. The value of a pixel is highly correlated with neighbouring pixels' values. Pixels of an iris are likely to be the same. Where neighbouring pixels' values are different, this too contains meaning. Continuing our example, a black pixel would indicate a pupil and white the sclera. Flattening data loses this information, Convolutional Neural Networks (CNNs) use this information.

2.1 CNN 101

CNNs exploit the spatial autocorrelation in image data by taking $k \times k \times n$ filters of data to generate features. Here k is some integer (typically between 1 and 11), and n is the depth of the input. For RGB images n=3, but for satellite images with more bands, we can have $n > 3$.

A *filter* starts at the top corner, computes the sum of the elementwise product between the filter and the associated corner of the image (See Figure 3). This generates a single number. The filter is moved to the right and we again take the sum of the elementwise productv. Sweeping across the image we generate a two dimensional matrix. To

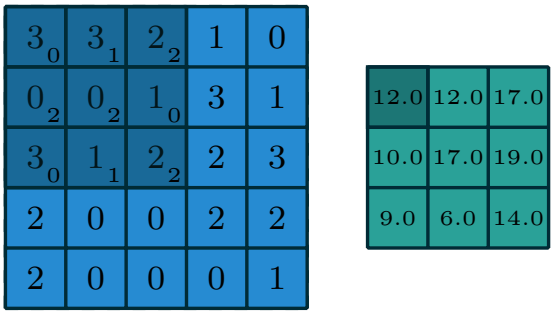

Figure 3: An example of a convolutional operation. The image has 5 pixels length and width. The convolutional filter has size 3×3. The elements are multiplied together and summed to the amount in the top left cell in green. The next step would shift the convolutional layer one pixel to the left. For an RGB image, the filter would be applied to all three layers. Image source: [4]

make the output shape the same as the input shape, the input image can be *pad*ded with zeros. Adding additional filters adds layers to this two dimensional matrix. That is, making the output three dimensional with the depth of output is equal to the number of filters.

To stack layers together for deeper models, there are three additional core components. First, to allow the model to find non linear relationships, models will often include a non linear *activation* function after the elementwise product and sum operation. Common activation functions include the `sigmoid`, `tanh` and `ReLU` functions. The `ReLU` is popular because it is fast and effective. For an output x, the `ReLU` is $\max\{0, x\}$.

Second, to reduce the spatial size of the output, a *pooling* layer reduces a $\ell \times \ell$ patch of the output to a single number. We often using the max operator (See Figure 4). Pooling also reduces the amount of parameters in the model, making it harder for the model to memorise training data. This is known as *overfitting*, where the model works well on training data but has poor performance on new data. Pooling layers act to extract information about what is in the image. The cost of pooling is that we lose some information about where in the image an object is. The bulk of a CNN is stacking these convolution, activation and pooling layers on top of each other.

Last, after stacking these layers on top of each other we flatten the output to a series of dense one dimensional layers. Once we understand these building blocks, coding them is straight forward (see Figure 5).

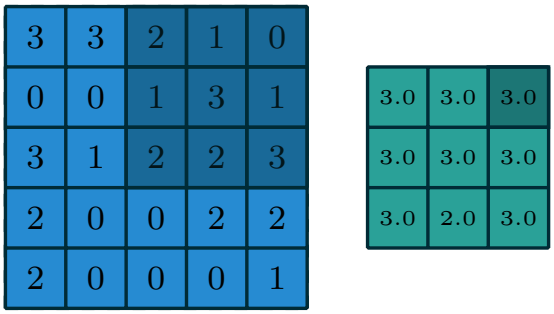

Figure 4: An example of a max pooling operation. The image has 5 pixels length and width. The pooling filter has size 3×3. The maximum value of the image within the window is passed to the output cell in green. The next step would shift the convolutional layer one pixel to the left. Image source: [4]

2.2 Training models

The question is then: how to train these (hundreds of) millions of parameters? The high level answer is that we define a *loss function* of the model error, and try to minimise this function. The *error* is the difference between predictions and actual labels and the *loss* is a differentiable function of the error and the model parameters.

Because the loss function is differentiable, we can use calculus to approximate the minimum. This step is known as `gradient descent`. We take the gradient of the loss function and update the weights by the negative of the gradient (see Figure 6). Since we often train on many thousands (or millions) of images, we cannot compute this in memory. Instead, a small batch of images is used and the model updated. This is called `batch gradient descent`. There is also stochastic gradient descent for batches of size one observation. With enough batches, we obtain good results over time.

We don't have to hard code the `evaluate_gradient` function, we can rely on libraries to do this for us. We saw this in Figures 1 and 2 above. We have the `sgd`, or stochastic gradient descent optimiser and the `rmse` or root mean squared error loss function.

There many options for both the optimiser and the loss function. The development of optimisers is an ongoing area of research (e.g. [6]). Despite the choice, default options will be fine for experimentation. Loss functions are selected by the problem at hand. For regression, root mean squared error loss functions are most common. For classification, categorical cross entropy is most common.

```
    # load libraries
from keras.models import Model, Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Flatten, Input
from keras.layers import LSTM, Embedding, Dense

model = Sequential()
    # add first convolutional layer with
    # 64 filters of size 3 x 3
    # and padding to retain image size
    # and ReLU activation function
model.add(Conv2D(64, (3, 3),
          activation='relu',
          padding='same',
          input_shape=(224, 224, 3)))
    # add second convolutional layer with
    # 64 filters of size 3 x 3
    # and ReLU activation function
    # no padding
model.add(Conv2D(64, (3, 3),
          activation='relu'))
    # add pooling
model.add(MaxPooling2D((2, 2))
model.add(Flatten())
model.add(Dense(4096, activation='sigmoid')(inputs)
model.add(Dense(10, activation='sigmoid')(inputs)
```

Figure 5: A simple CNN in`Keras`. This model has two convolutional layers with ReLU activation functions, a max poolution layer and a dense layer to predict 10 classes.

```
while True:
  weights_grad = evaluate_gradient(loss_fun,
                                   data,
                                   weights)
    # perform parameter update
  weights += - step_size * weights_grad
```

Figure 6: Pseudo code for basic gradient descent. First, we calculate the gradient of the loss function with respect to the model weights. We then use this to modify the model weights. Source: [5]

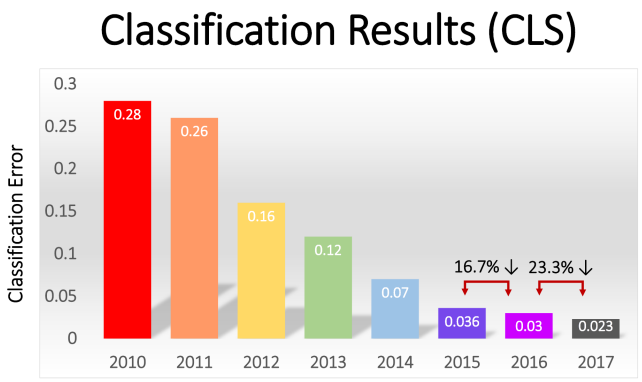

Figure 7: ImageNet classification results, 2010–2017. We see that the error rate dropped rapidly over 2011–2015. Source: Image Net Overivew[3]

These are the simple building blocks for building an object classifier. We now turn to the literature to understand some important details.

3 LITERATURE

The above section is a quick introduction to convolutional neural networks. We go deeper in this section, focusing on key results in image classification, object detection and segmentation. These problems nest each other: to detect an object, a machine needs to output what the object is (classification) and where it is. To segment and image, the machine needs to output where the object is (detection and classification) and where the boundaries of these objects are.

3.1 Object classification

Object classification models tell us what is in an image. The output is a label, for instance a cat or a building. Recent years have seen rapid evolution in models for classification. We focus on a four key papers that have defined progress in this field (see Figure 7).

We begin with AlexNet, which won the ImageNet Classification challenge in 2012 [9]. AlexNet features three innovations which are still in use today. To begin, AlexNet was the first model to 'go deep', with five convolutional layers. Second, the model used the `ReLU` activation function (discussed above). Last, to prevent overfitting, AlexNet used dropout. `Dropout` worked by randomly setting the value of some parameters to zero during training. This way, the model could not rely on specific parameters to predict object classes. Instead,

```
model.add(Dropout(0.5))
```

Figure 8: Adding dropout to a model is simple. Here, we set half of the weights to zero at each training iteration.

the model has to spread this information across parameters. We can easily add innovations to a model (E.g., Figure 8).

VGG Net [7] came second in the main image image classification contest in 2014 (ILSVRC). Although not the best performing model, it has a simple architecture and is perfect for beginners. The architecture is a repeating set of convolutional layers to extract spatial information, activation layers for non linearities and max pooling to reduce information.

VGG net is widely used as a *feature extractor*. VGG net can extract features by taking an arbitrary image and setting the output at the flattening stage. That is, we remove the fully connected layers (or 'top') of the model and reduce the image to a vector. The vector retains some high dimensional representation of the image that can be used in a standard classifier, for example a logistic regression model. The original VGG net weights are used, and no training is undertaken. This common practice can be used to train models on very few observations.

Models went deeper over the next few years, but a puzzle arose: performance did not rise – infact sometimes performance dropped – with additional layers. This was a puzzle because models could simply pass on the same output through multiple layers rather than having degraded performance. This insight resulted in the residual learning framework that passes both the input x and the output of a layer $f(x)$ to the next layer. With these skip connections the model can choose to discard elements of the modified layer output if elements do not help in prediction. An additional innovation is the lack of fully connected layers at the end of the model. These innovation allowed ResNet to have 152 layers and have lower model complexity [10]. ResNet won ILSVRC 2015.

The current state of the art involves creating a model out of an ensemble of models. The winner of the 2017 ImageNet challenge used ensembles and an innovation called 'Squeeze and excitation' [11]. Squeeze-and-excitation networks (SENets) weight the information across image channels. That is, if the red image channel helps with prediction more (say, for fire trucks), the model will learn to use channel to make predictions. SENets lowered the ImageNet classification error rate by almost 25 per cent.

Detecting trucks from cats is a good start, and research is progressing on more challenging problems. There is still a lot of room to

```
1  from keras.applications.vgg16 import VGG16
2  from keras.preprocessing import image
3  from keras.applications.vgg16 import preprocess_input
4  import numpy as np
5
6  model = VGG16(weights='imagenet', include_top=True)
7
8  img_path = 'remote_area_example.jpg'
9  img = image.load_img(img_path, target_size=(224, 224))
10 x = image.img_to_array(img)
11 x = np.expand_dims(x, axis=0)
12 x = preprocess_input(x)
13
14 features = model.predict(x)
```

Figure 9: Using VGG16 in `Keras`. On line 6 we import the model architecture and model weights.

improve for aerial or satellite imagery. For instance, there is evidence that the number of satellite bands does not matter for building detection [12]. One needs only layers from the red, green and blue spectra. One could see SENets exploiting the 5-10 bands of non visual information that comprise satellite images.

3.1.1 *VGG net in `Keras`*

In Figure 9 we download the model on line 6. In this block we download the whole model so that we can use the model to predict what's in the image. If we want to extract features, we can change the argument `include_top` to be `False`. Rather than a list of object class weights, the output of `model.predict()` will be a 4096 dimensional vector. We can store these for use in another model. To date (July 2018), `Keras` has ten image classifier models that can be used in the same way.[4]

3.2 Object detection

The next step after identifying if an object is in an image is pointing out where the object is. The challenge is to "detect a *potentially large number of object instances with varying sizes in the same image* using a limited amount of computing resources." [13, their emphasis]. Models that solve this problem will be useful for detecting where key sites — like damaged roads — are.

[4] https://keras.io/applications/

A brute force, basic model is to slide a classifier across an image. Since this model will have to run multiple times over a single image the model will be very slow.

An early innovation is to treat localisation as a regression problem [13]. This model uses a small standard convolutional neural network, but instead of a softmax classifier layer, the model uses a regression layer to output a binary mask: 1 for inside a bounding box, 0 for outside. One problem was that most of the image is outside the bounding box, so a model can learn to only output zeros. The authors increase the weights of non zero outputs to overcome this problem. Overall, there are five networks: one for box predictions, four others for where the top, bottom, left, right of the box.

As with many of the models to come, the model in [13] is pre-trained on a classification task. A *pre-trained* model is one that is trained for an additional task, typically where we have lots of data. The model has then incorporated the capacity to interpret images. This model is then trained with new data on the task it is required to do, here object classification.

The next innovtion came with regions with CNN features (RCNN) [14, 15]. RCNN improved the then state of the art by more than 30%. Their idea is to extract some two thousand bounding boxes in a preprocessing step, run a classifier through these boxes and combine them at the end. They also use supervised pre training on a large dataset using the VGG net architecture. RCNN works, but is multi stage, expensive to run and slow.

Additional innovations increased the speed of this model. Fast-RCNN [15] shares features across object proposals rather than recalculating features for each proposal. The model also features a multi task loss function for classification and a four dimensional regression for the bounding box. Faster RCNN [16] overcomes the bottleneck of region proposals with a 'Regional Proposal Network' (RPN). The RPN takes *anchors*, or fixed points in the image, and first classifies whether there is any object there. Second, the anchor bounding box is adjusted for a better bounding box fit. FasterRCNN is trainable end to end. This means we can use gradient descent to train the model all at once, rather than connect different pieces.

Despite these innovations, detection was still slow. The You Only Look Once (YOLO) series of models [17] increased the speed of detection.[5] YOLO works by dividing an image into a grid. For each grid cell, there are a fixed number of bounding boxes. For each bounding box, the model generates five predictions: four spatial components that modify the shape of a candidate bounding box and a measure of confidence. The model then applies a classifier. The model uses

[5] For a video showing the speed at which YOLO can work, see http://www.youtube.com/watch?v=NM6lrxy0bxs

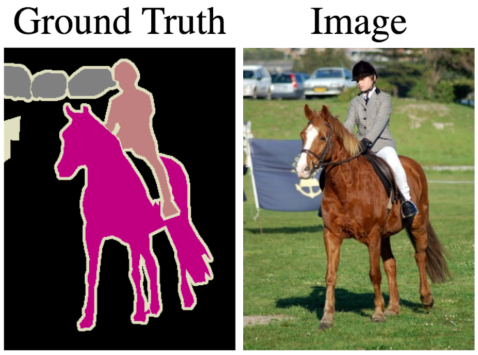

Figure 10: An example of an image segmentation problem. The model must predict the horse, the rider, the three cars and the boundaries between them. Source: [21]

Google's LeNet as a base model and shows good performance in new domains. YOLO version 2 improved performance through pretraining the model on the ImageNet classification task.

Because of this and the model's speed, YOLO might be useful for a first pass over large areas of satellite images. One limitation of YOLO is that it is not the best for identifying many tightly connected objects. To overcome this and other limitations, a recent paper modifies YOLO for satellite images [18]. First, they use upsampling and multiple models at different scales to capture small images.[6] For instance, one classifier for airports and another for airplanes. Second, they define a new network architecture with a denser final output for smaller bounding boxes.

3.3 Image segmentation

The final task we investigate is image segmentation. The challenge here is to outline the boundary of the object (see Figure 10). In one sense this is a simple extension of classification: but one prediction per pixel rather than one prediction per picture. Segmentation would be useful in tracing out road paths and distinguishing regions with damaged properties versus non damaged.

With segmenation, there is a tension between semantics and location. *Semantics* is the question of what, *location* asks where. Local information helps answer the former and global information helps answer the latter. This need for two types of information gave rise to complicated models prior to the era of fully end to end differentiable models. This tension also explains the innovations beyond basic CNNs used for classification.

[21] developed the first end to end trained CNN for segmentation. [21]'s insight was to think of a fully connected layer as a 1 x 1 convolu-

[6] Upsampling is discussed below

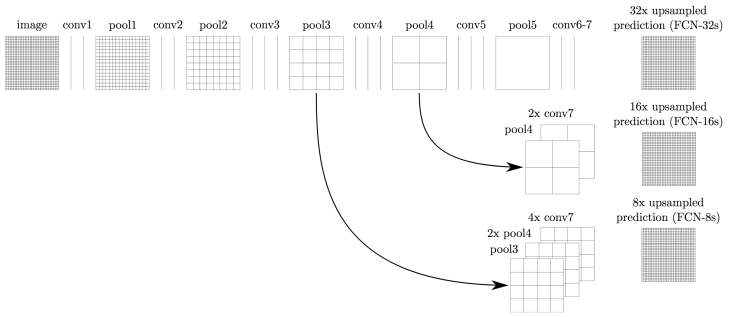

Figure 11: Skip architecture for segmentation. Output from the Pool3 and Pool4 are combined with outputs of Conv7 for more refined location predictions. Source: [21]

tional layer that covers the whole image. Using a standard CNN (e.g. VGG net) did not work well as by the time you've gotten to predicting a pixel, you've lost a lot of the information on the surrounding pixels. The authors get around this by bringing intermediate layers back into the prediction (Figure 11). This allowing the model to combine coarser semantic information and finer appearance information.

The output of this model was still too coarse. The solution was to bring in more skips. Many models do this [22, 23].[7] U-Net [23] uses a CNN to *encode*, or extract features. A *decoder* network then converts these features into a per pixel classification. Because the feature is a small dense representation, the decoder network must *upsample* – or 'blow up' – the image from the dense representation. A benefit of UNets is that we can use any pretrained network to encode the data. UNets are commonly used in the data science community. Segnet uses a similar architecture, but uses less computational space to do so.

An alternative attack on using both local and global information is to discard max pooling and use components that compress class information while retaining location info [24, 25]. These components, called *atrous* convolutions, generate low to mid resolution outputs (Figure 12). Models using atrous convolutions require an additional step to refine the segmentation. Conditional Random Fields (CRF) are used most often.

Mask RCNN [26] extends faster RCNN by adding a branch to the model for pixel level classification. Since the output of the CNN is smaller than the ground truth image, some form of *upsampling* or increasing the size of output images is required. Mask RCNN achieves this by using interpolation. Say our image size is 512 x 512, we are proposing a region in the top left 25 x 25 pixel corner, and feature map is size 30 x 30. Then our region of interest is $25 \times 30 \,/\, 512 \approx 1.46$

[7] If you like, you can always combine the two `https://github.com/ykamikawa/Seg-UNet`

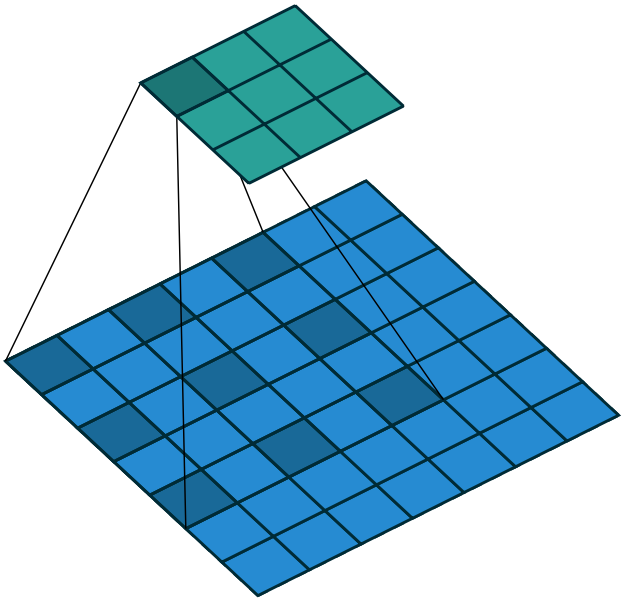

Figure 12: Dilation operation Source: [4]

pixels of the feature map. A naive implementation would use integer division for a region proposal of 1 pixel. Mask RCNN uses bilinear interpolation to estimate what the 1.46th pixel would look like.

4 ANALYSIS WITH SATELLITE IMAGES

We now look some practical applications with satellite images. There are three branches of knowledge we can look at. First, there is the academic literature. Second, data science competitions. Last, we have industry.

4.1 Applications in academia

Much of the academic literature takes a pre-trained CNN or common architecture and trains the model on the task at hand. Two papers have focused on mapping poverty [27, 28]. [27] use satellite images, land use maps and household survey data to map poverty in Mexico. They trained a model using RGB bands and near infrared band. Because pretrained models use the three RGB bands, the model had to be retrained from scratch to exploit all satellite bands. Around fifty per cent of variation in local poverty can be predicted through satellite images and a CNN alone. This increases to around sixty per cent when they include land use cover as a feature. It is not clear from the paper how they include land use cover information maps. It does not

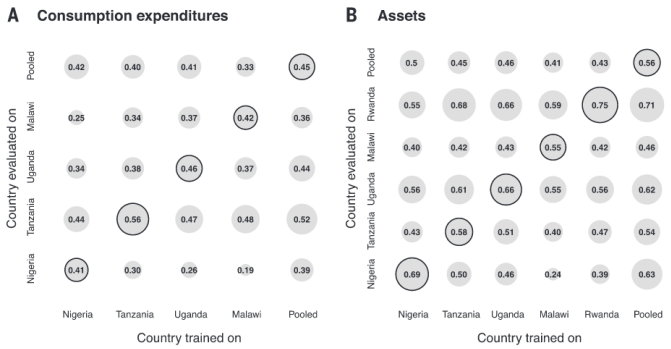

Figure 13: Understanding how well models trained on satellite images from one country generalise to another country. Source: [28]

seem to be a feature layer, perhaps it was used as a regression input with the CNN output, see (1).

$$\text{composite prediction} = \alpha + \beta \text{CNN output} + \phi \text{land use} \tag{1}$$

[28] estimate poverty across Sub Saharan Africa. Since ground truth values are difficult to come by, they use night lights as a proxy. They fine tune a pretrained CNN (VGG 8 layer) to predict night lights. These features are fed into a ridge regression model (LASSO) to estimate poverty taken from household surveys. The satellite data is taken from the Google Static Maps API. Pixels are approximately 1km square. The model's transfer learning is promising. Figure 13 shows the average R^2 for models trained on one country and evaluated each country.

A similar strand of research maps population [29, 30]. [29] map population in Kenya and Tanzania using a VGG-net style architecture. The authors exploit all eight landsat bands by retraining a model from scratch. [30] also train their own model. They use a two step procedure to obtain raw estimates for population. The first gets coarse values at the pixel level. These are combined at the county level. The results are directly interpretable as population estimates. Facebook [31] are also estimating population. This seems to be a building segmentation map. Local population density is estimated by taking the known population and spreading this population amongst the area inferred from the building segmentation map. In addition, Facebook use segmenting tools to identify roads [32].

4.2 Applications in the data science community

In the past few years, data science competitions have been organised around small, high resolution datasets. The two SpaceNet chal-

lenges asked participants to segment roads, buildings and water bodies, amongst others.

In the first edition, the test challenge was to predict buildings in Rio de Janiero, Brazil. This is particularly challenging because buildings in Rio are small and densely built. Four of the top five entries used CNNs. Second place used SSD, third MNC whereas forth and fifth trained their own model. Fifth place in particular placed a large emphasis on image augmentation. The winner used a sequence of well trained random forest models.

In the second edition, additional cities were added. These cities (Las Vegas, Paris, Shanghai and Khartoum) all have a different built infrastructure. The winner used U-Net. To use all the spectral bands, the model had to be trained from scratch. This model did struggle with small and L shaped buildings. The second and third places used models based on the chained random forests models that won the first edition.

Last, we note that there is a new competition that is worth keeping your eyes on`http://deepglobe.org/challenge.html`.

4.3 Applications in industry

Many industries use satellite data to generate information. We do not have access to proprietry methods; however, understanding applications may assist in generating ideas. Common applications include crop monitoring and forecasting,[8] water cover and use,[9] car park occupancy rates (consumer demand) and oil reserves[10]. Of note is the insurance industry, who use satellite images for predicting claims amounts, and identifying damaged or unaffected assets. As one would expect, the intelligence community and associated research groups are doing work too.[11]

4.4 NGO applications

NGOs have tended to use traditional GIS technolgoies — like change detection — to identify damaged buildings. For instance, HRW used before and after photos to identify burned buildings in Rohingya villages throughout Burma [35]. Amnesty International used satellite images to identify buildings and villages attacked by Boko Haram in

[8] `https://telluslabs.com/`

[9] `https://www.vandersat.com/`

[10] `http://www.orbitalinsight.com/`

[11] For instance, CosmiQ (`http://www.cosmiqworks.org/space-30/`) have developed a time series model to map infrastructure after a hurricane [33] and a tool to create your own time series from satellite imagery `https://github.com/CosmiQ/CometTS`

Nigeria [36]. Other applications include identifying the 2011 Oil Spill in Niger [37] and mass graves in Afghanistan [38]. A key aspect of these is that the locations were roughly known prior to analysis. For more applications, see [39].

5 SATELLITE DATA

There are many sources of satellite imagery.[12] Satellite imagery is expensive, but free or limited options are available. For isntance, both LANDSAT and the Copernicus mission images are free of charge and available through the USGS Earth Explorer[13] and the Scientific Data Hub[14] or the Google Earth Engine.[15] LANDSAT has the benefit of a forty year history of images, although at 30m (15m panchromatic) the resolution is course. The Copernicus Programme's Sentinel satellites have a resolution of 10m. At these resolutions, individual buildings will be difficult to isolate.

We know that image resolution matters [42]. It is difficult with the naked eye to locate a 10 square meter building with 10 square meter pixels. For the most part, higher resolution imagery typically costs money. For humanitarian and research purposes, small datasets can be provided. For instance, Planet and Satellogic have researcher and humanitarian access [43, 44]. I would not be surprised if other providers matched this offer. In addition, Google maps and Bing Maps have APIs that allows users do download a limited amount of undated high resolution images. The images have a watermark which can easily be cropped out. Table 1 presents an overview of high resolution image providers.

Table 1: A list of satellite imagery providers

Provider	Free	Description
Planet	Limited	3–5m imagery anywhere in the world
Satellogic	Limited	30 bands, 30m resolution
UrtheCast	N	0.75m resolution, video available
Digital Globe	N	Global coverage, 30 cm resolution
Bing	Y	High resolution, undated
Google	Y	High resolution, undated

[12] The following URLs provide a good list as of July 2018: https://gisgeography.com/free-satellite-imagery-data-list/
[13] https://earthexplorer.usgs.gov/
[14] https://scihub.copernicus.eu/dhus/
[15] https://earthengine.google.com/

5.1 Labelled data

Training models requires labelled data. One option is to do this by hand or outsource to services like Mechanical Turk.[16] For drone data or specific objects to detect, this is a good option. For buildings, roads or waterways, there are a few datasets of use. Some of these are down, but may still be available through other researchers, or asking the organisers.

First, there is the Spacenet challenge data.[17] This data appears to be down at present. The data consists of very high resolution (30–50cm) images of five cities. The data contains labels of buildings, trees, cards, roads and waterways. More labelled urban environments can be found in Urban Environment dataset.[18] For labels, there are two good sources. First, the Urban Atlas project.[19] This project is used in the Urban Environment dataset. A second option is OpenStreetMap.[20] These maps are created by volunteers, much like wikipedia. Your mileage may vary with building accuracy.

6 CONCLUSION

Deep learning is made rapid progress in the ability of computers to extract information from images. In addition, recent advances in satellite technology have allowed companies to provide timely, high quality satellite images. These images can provide large amounts of information for understanding how nature and society is changing on the ground. We can harness two developments to understand what is happening in the immediate aftermath of a disaster.

This report has attempted to provide a high level overview to deep learning for satellite imagery. We presented the ideas behind models that can tell us what is in an image and where in the image an object is. We connected these ideas to `Python` code, so that the reader can understand how to implement these in practice. Although there is a little more to building models than in this report, these code fragments provide the basis for understanding the vast array of available blog posts, tutorials and MOOCs. Last, we present applications and data sources.

[16] https://www.mturk.com/
[17] https://spacenetchallenge.github.io/datasets/datasetHomePage.html
[18] https://github.com/adrianalbert/urban-environments/tree/master/dataset-collection
[19] https://www.eea.europa.eu/data-and-maps/data/copernicus-land-monitoring-service-urban-atlas
[20] https://www.openstreetmap.org

REFERENCES

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[2] Quantitative Economics. Introduction to python – quantitative economics.

[3] Wojciech Rosinski. Deep learning frameworks speed comparison, 2017.

[4] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016.

[5] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition.

[6] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[8] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.

[12] Adam Van Etten. Panchromatic to multispectral: Object detection performance as a function of imaging bands, 2017.

[13] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors,

Advances in Neural Information Processing Systems 26, pages 2553–2561. Curran Associates, Inc., 2013.

[14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 580–587, Washington, DC, USA, 2014. IEEE Computer Society.

[15] R. Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, Dec 2015.

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149, June 2017.

[17] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[18] Adam Van Etten. You only look twice: Rapid multi-scale object detection in satellite imagery. *CoRR*, abs/1805.09512, 2018.

[19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[20] Gabriela Csurka, Diane Larlus, Florent Perronnin, and France Meylan. What is a good evaluation measure for semantic segmentation?. In *BMVC*, volume 27, page 2013. Citeseer, 2013.

[21] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[22] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[24] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE*

Transactions on Pattern Analysis and Machine Intelligence, 40(4):834–848, April 2018.

[25] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015.

[26] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, Oct 2017.

[27] Boris Babenko, Jonathan Hersh, David Newhouse, Anusha Ramakrishnan, and Tom Swartz. Poverty mapping using convolutional neural networks trained on high and medium resolution satellite images, with an application in mexico. *arXiv preprint arXiv:1711.06323*, 2017.

[28] Neal Jean, Marshall Burke, Michael Xie, W. Matthew Davis, David B. Lobell, and Stefano Ermon. Combining satellite imagery and machine learning to predict poverty. *Science*, 353(6301):790–794, 2016.

[29] Patrick Doupe, Emilie Bruzelius, James Faghmous, and Samuel G. Ruchman. Equitable development through deep learning: The case of sub-national population density estimation. In *Proceedings of the 7th Annual Symposium on Computing for Development*, ACM DEV '16, pages 6:1–6:10, New York, NY, USA, 2016. ACM.

[30] Caleb Robinson, Fred Hohman, and Bistra Dilkina. A deep learning approach for population estimation from satellite imagery. In *Proceedings of the 1st ACM SIGSPATIAL Workshop on Geospatial Humanities*, GeoHumanities'17, pages 47–54, New York, NY, USA, 2017. ACM.

[31] Tobias Tiecke Andreas Gros. Connecting the world with better maps. https://code.facebook.com/posts/1676452492623525/connecting-the-world-with-better-maps/, 2016. [Online; accessed 19-May-2018].

[32] İlke Demir, Forest Hughes, Aman Raj, Kaunil Dhruv, Suryanarayana Murthy Muddala, Sanyam Garg, Barrett Doo, and Ramesh Raskar. Generative street addresses from satellite imagery. *ISPRS International Journal of Geo-Information*, 7(3), 2018.

[33] Jacob Shermeyer. Assessment of electrical and infrastructure recovery in Puerto Rico following hurricane Maria using a multi-source time series of satellite imagery. 2018.

[34] Satellite imagery for human rights monitoring.

[35] Human Rights Watch. Burma: Massive destruction in rohingya villages, 2016.

[36] Nigeria, 2015.

[37] By Christoph Koettl. (s)hell in the niger delta: Satellite images document oil spills, 2011.

[38] Aaas contributes satellite image analysis to investigation of purported mass-grave in afghanistan | aaas - the world's largest general scientific society, 2013.

[39] Human rights applications of remote sensing | aaas - the world's largest general scientific society, 2014.

[40] Robert Simmon. Making sense of satellite data, an open source workflow: Accessing data.

[41] AAAS. High-Resolution Satellite Imagery Ordering and Analysis Handbook, 2018.

[42] Adam Van Etten. The satellite utility manifold; object detection accuracy as a function of image resolution, 2017.

[43] Planets. Planet — disaster datasets.

[44] Satellogic. Satellogic | commitment to science.