Queen Mary
University of London

DR ADRIAN BEVAN

PRACTICAL MACHINE LEARNING

CNNS – CODING

LECTURE PLAN

- Introduction
- Implementing:
  - conv(olution) layers
  - maxpool layers
  - fully connected layers
- Output

QMUL Summer School: https://www.qmul.ac.uk/summer-school/
Practical Machine Learning QMplus Page: https://qmplus.qmul.ac.uk/course/view.php?id=10006

INTRODUCTION

- The basic building blocks of a convolutional neural network (CNN) have been discussed earlier.
  - Need convolution layers to process regions of images
  - Stride from one region of an image to the next to produce a convolution image of the original feature space.
  - Maxpooling is used for dimensional reduction of convolution images.
  - Fully connected layers translate this final convolution or maxpool layer output into a decision or regression score.
- Deep networks of this type can quickly start to generate hundreds of thousands or millions of HPs that need to be optimised, and here normal CPUs become a tiresome part of the development cycle for a model.

CONVOLUTION/POOL LAYERS

- `Example_MNIST_CNN.py`

- Before proceeding the input image needs to be reshaped from a flat 784 array of pixels to a 28x28 2D array:

```
x_image = tf.reshape(x_, [-1,28,28,1])
```

  - Arguments are: -1=feeding a batch of images, otherwise specify the number of images to reshape; width (in pixels), height (in pixels), number of colour channels.

- TensorFlow provides a `tf.nn.conv2d` function implementation to facilitate construction of a conv layer.

- Also a maxpool function (in `tf.nn.max_pool`)

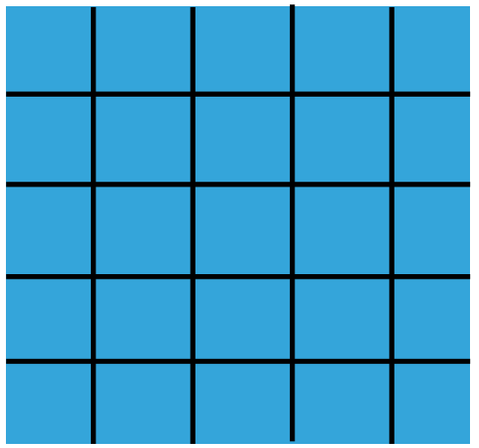
```
# layer 1
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

CONVOLUTION/POOL LAYERS

- `Example_MNIST_CNN.py`

- weight_variables([5,5,1,32]):
  - 5x5 convolution filter
  - depth of 1 (number of input channels)
  - stack of 32 convolution filters (number of output channels)

32 x



Single channel (grey scale) images

- weight_variables([32]):
  - stack of 32 convolution filters (number of output channels)

```
# layer 1
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

CONVOLUTION/POOL LAYERS

- `Example_MNIST_CNN.py`

- 2x2 max-pooling
  - Input image size: 28x28
  - Downsampling a 2x2 array of pixels to a single pixel results in a 14x14 array.

```
# layer 1
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

CONVOLUTION/POOL LAYERS

- `Example_MNIST_CNN.py`

- Input 28x28 pixels, with a depth of 1
- Both conv layers use padding:
  - Expand the input image to retain the same size [28x28] output image.
  - Conv filter is a 5x5 filter, so the padding expands the 28x28 image to a 32x32 image (adding a border of 2 pixels).
  - 1st conv layer has 32 outputs, 2nd conv layer has 64 outputs.
- Maxpool layers then halve the image size:
  - 1st pool layer: [28x28] reduces to a [14x14] image.
  - 2nd pool layer: [14x14] reduces to a [7x7] image.

FULLY CONNECTED LAYERS

- `Example_MNIST_CNN.py`

- The fully connected (FC) layer is just like an MLP.

- # Input nodes = (#pixels in input image) x (#filters)

= (7x7) x 64
(fixed by conv/maxpool config)

- # output nodes = 1024 (arbitrary choice of user)

```
# fully connected layer
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

FULLY CONNECTED LAYERS

`Example_MNIST_CNN.py`

- The fully connected (FC) layer is just like an MLP.
- Reshape the tensor to make sure that it is flat (like an MLP) rather than square like an image.

```
# fully connected layer
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

FULLY CONNECTED LAYERS

- `Example_MNIST_CNN.py`

- The fully connected (FC) layer is just like an MLP.
- Output of this FC layer is a relu function just like the final layer of an MLP.

```
# fully connected layer
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

DROPOUT

- `Example_MNIST_CNN.py`

- Dropout is achieved by using the nn.dropout function on the model of the conv/maxpool and FC layer.

```
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

- The keep_prob placeholder is then set when training.

CNN MODEL OUTPUT

`Example_MNIST_CNN.py`

- The model is constructed from the fully connected layer, compactifying the 1024 dimensional inputs to a 10 output decision.

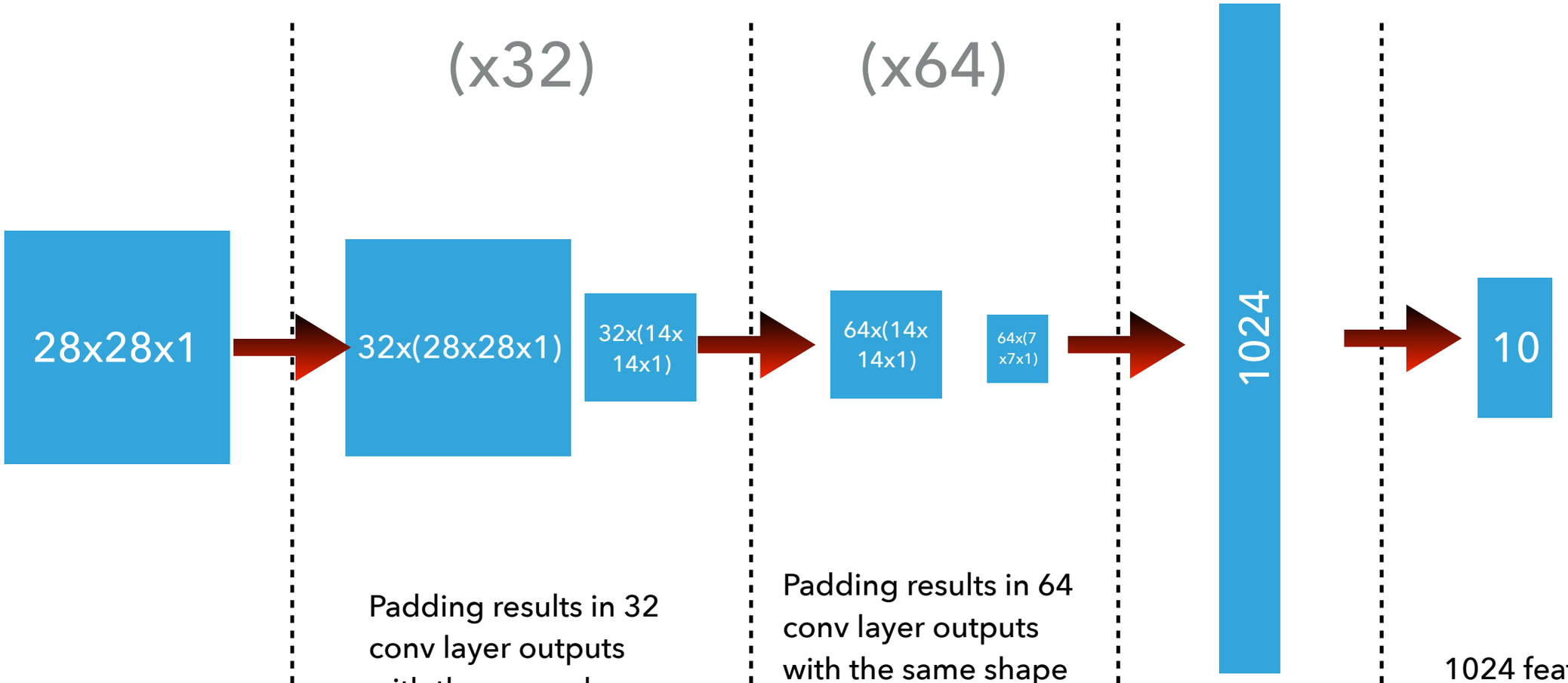
```
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

model = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

- This is achieved with the usual matrix multiplication step we've seen throughout.

CNN MODEL OVERVIEW

- `Example_MNIST_CNN.py`

- input image : conv : pool : conv : pool : FC : output node



28x28x1
(=784 px) image

Padding results in 32 conv layer outputs with the same shape as the input.

Max-pooling reduces the 32 conv layer images to 14x14x1 (=196 px) images

Padding results in 64 conv layer outputs with the same shape as the input (14x14x1).

Max-pooling reduces the 64 conv layer images to 7x7x1 (=49 px) images

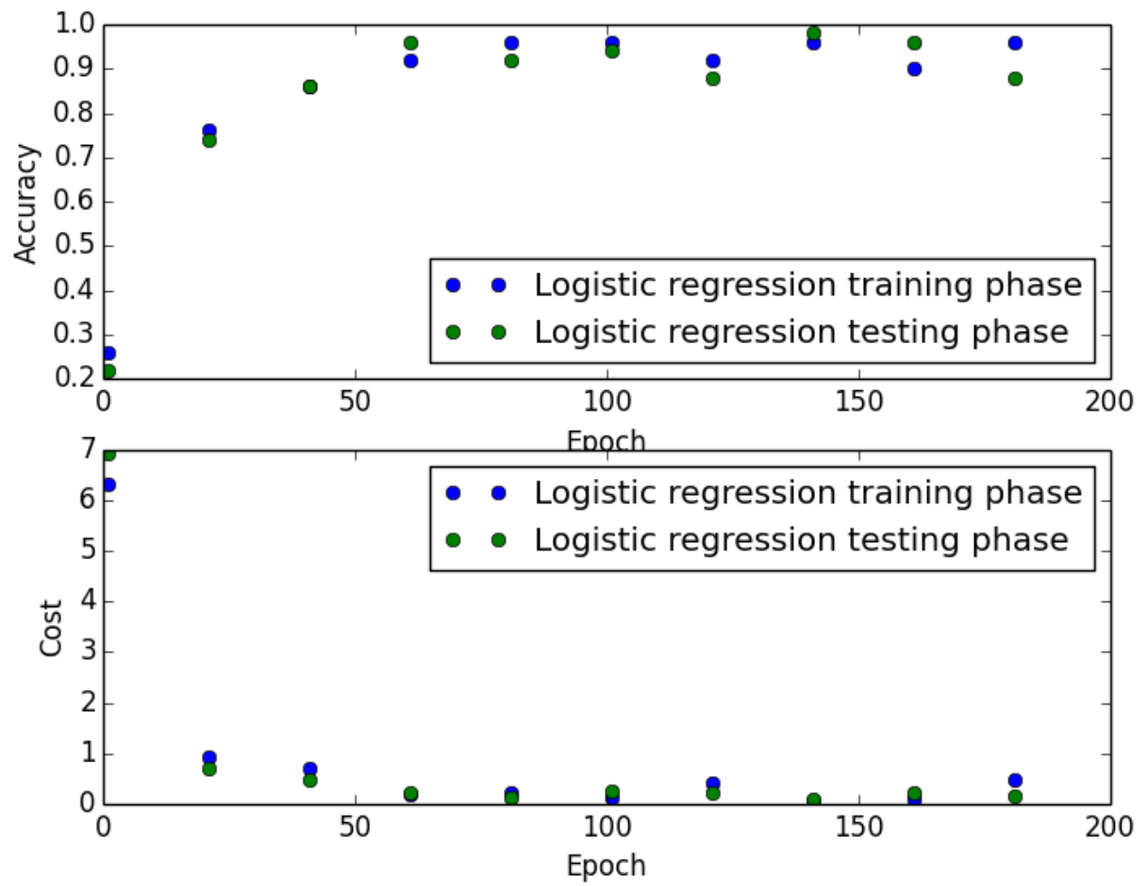
Flatten the 64 7x7x1 images into a single layer with 1024 outputs (reduced from 3136 features)

1024 features of the FC layer are parsed by the output node to provide one of 10 outputs.

OUTPUT

- Track the improvement in test and training sample over the training epochs.

```
step 0, training accuracy 0.26   test accuracy 0.22
step 20, training accuracy 0.76   test accuracy 0.74
step 40, training accuracy 0.86   test accuracy 0.86
step 60, training accuracy 0.92   test accuracy 0.96
step 80, training accuracy 0.96   test accuracy 0.92
step 100, training accuracy 0.96   test accuracy 0.94
step 120, training accuracy 0.92   test accuracy 0.88
step 140, training accuracy 0.96   test accuracy 0.98
step 160, training accuracy 0.9   test accuracy 0.96
step 180, training accuracy 0.96   test accuracy 0.88
```



- Use test, train and validation images to quantify accuracy

Accuracy(Train) = 0.9635 [biased metric*]
Accuracy(Test) = 0.9597
Accuracy(Validation) = 0.9646

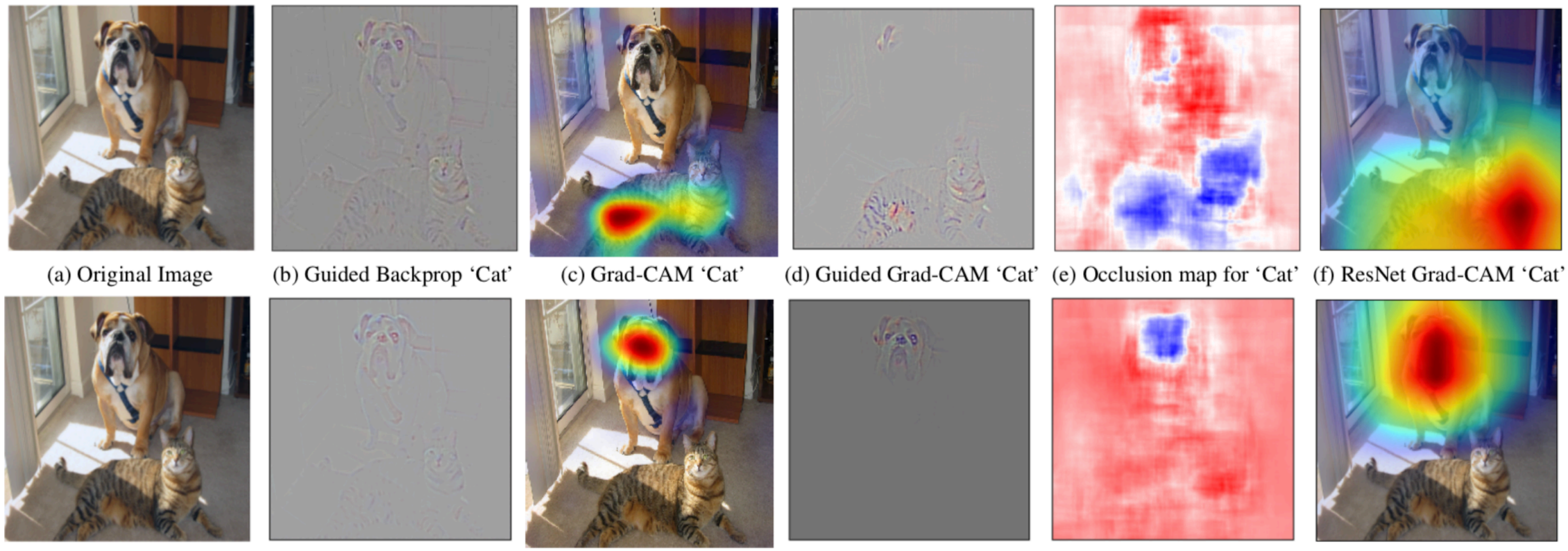
*Remember that this is biased as the training sample is used to configure the model. It is not a good metric to establish model performance.

WHAT IS HAPPENING WHEN MY NUMBERS ARE BEING CLASSIFIED?

- A CNN is a complicated algorithm, and it is easy to loose intuition with what part of the input feature space (image) leads to a given conclusion.
- Can easily lead to a black-box mentality for machine learning practitioners.
- But it is possible to ask the machine why it made a given decision; in this context this amounts to understanding what part of the image led to the classification.
- There are algorithms that allow you to probe this question.
  - Grad Cam and Guided Grad cam are examples of these See Ref. [1].
  - Reflecting on what parts of images lead to a mis-classification of an example can help data scientists engineer a better model.

[1] Selvaraju et al., https://arxiv.org/abs/1610.02391

GRAD CAM

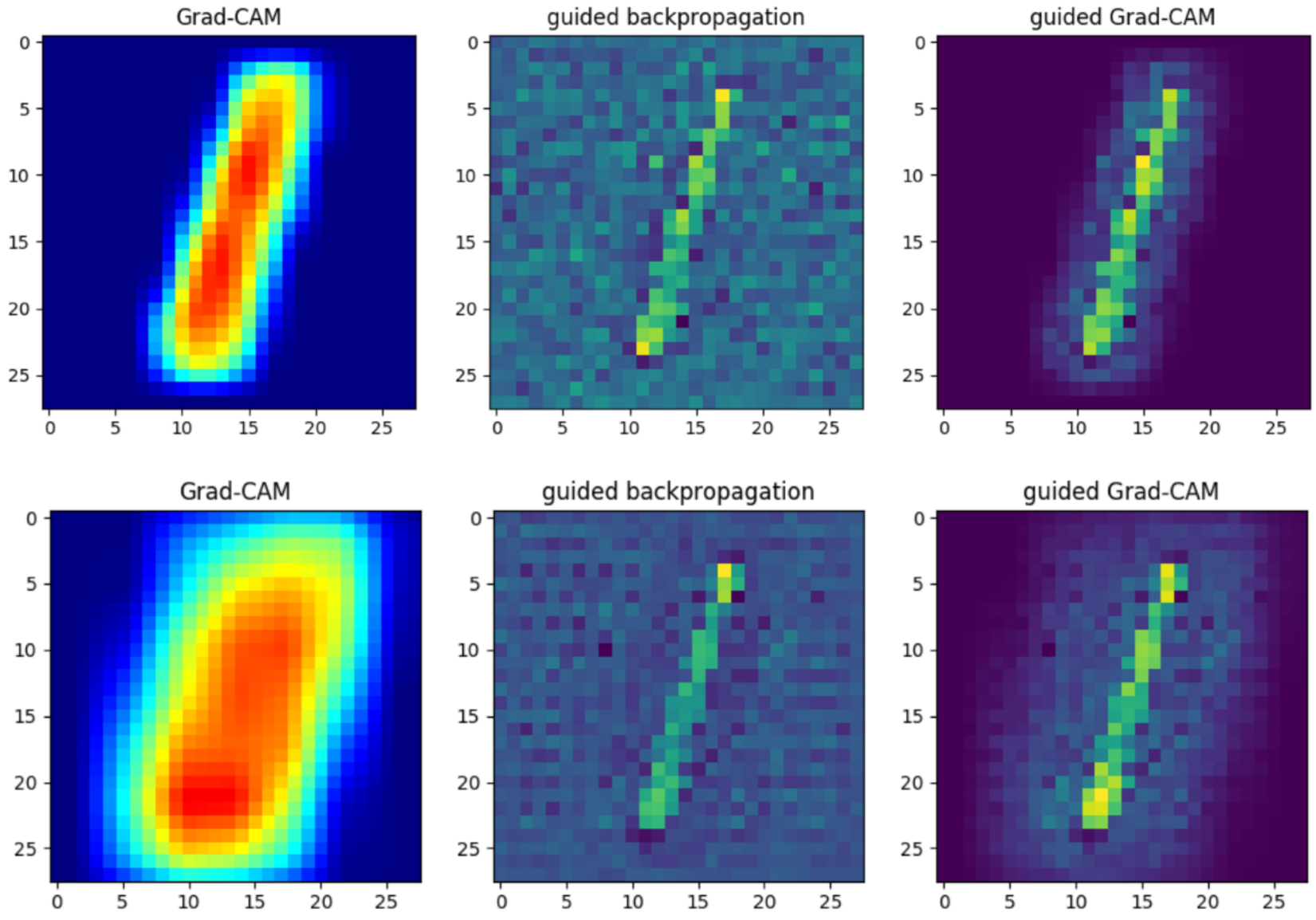


(a) Original Image (b) Guided Backprop 'Cat' (c) Grad-CAM 'Cat' (d) Guided Grad-CAM 'Cat' (e) Occlusion map for 'Cat' (f) ResNet Grad-CAM 'Cat'

- These examples (from Ref [1]) illustrate how different regions (edges for guided back-prop and heat maps from the Grad Cam) have led to a given classification.
- These algorithms provide local explanations of a classification - the field of computer science is actively working on methods for local and global explanations to provide a deeper understanding of models - not just for deep learning.

[1] Selvaraju et al., https://arxiv.org/abs/1610.02391

GRAD CAM: MNIST EXAMPLE



MNIST Example for a handwritten number 1, correctly classified.

- These examples (made by one of our undergraduates: M. Karim) show the use of these techniques in TensorFlow with CNNs that have 3 and 4 conv layers.

SUMMARY

- CNN's are straight forward to implement given the built in functions provided by TensorFlow.
- Some work is required by the user to keep track of the dimensional reduction if there is no padding, to ensure that the shape from one layer to the next is correctly computed.
- The final FC layer is reshaped from an image to a flat layer of nodes, from which the final output model value can be computed.

FURTHER READING

- The example CNN that we use is focussed on a simple interleaving of conv and pool layers, using dropout to enhance HP optimisation, and a FG layer to compactify information ahead of the output node.
- The AI community uses CFAR10 (RGB) images as a benchmark and there are more complicated CNN structures that will provide a better performance than these.
- For example look at the AlexNet and inception CNN models.

https://www.cs.toronto.edu/~kriz/cifar.html

https://www.tensorflow.org/tutorials/images/image_recognition

AlexNet:
https://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

Inception:
https://arxiv.org/abs/1409.4842
https://arxiv.org/abs/1409.4842
https://arxiv.org/abs/1512.00567