CNNs continued

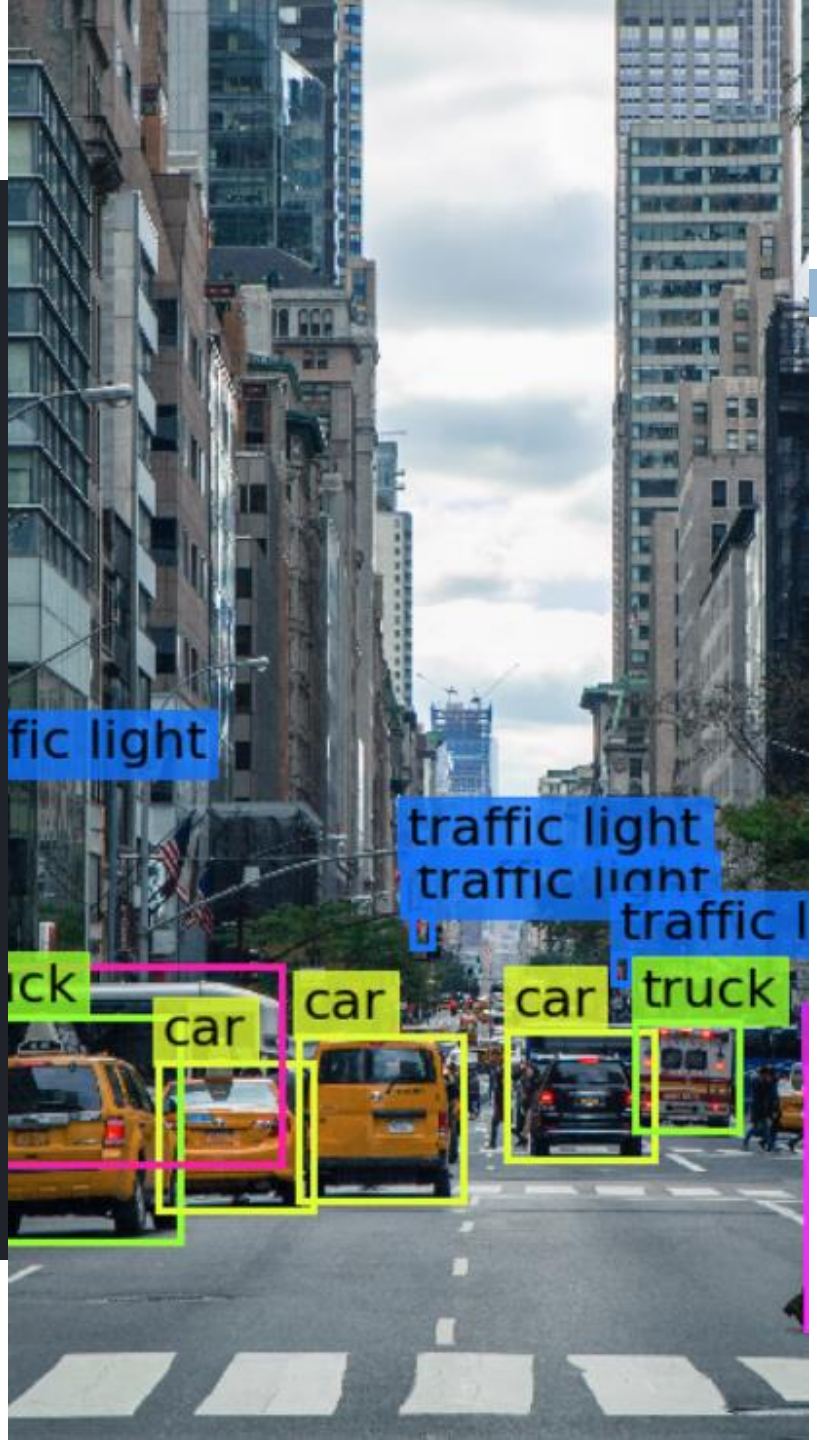
Deep Reinforcement Learning

D: CNNs continued.

J.J. Collins

Dept. of CSIS

University of Limerick



Reading

Based on:

- Chapters 11 and 14 in Aurelien Géron. Hands On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Ed. O'Reilly. 2019

Objectives

Section 1

- ML Challenges
- Weight Initialisation

ML Challenges

Main Challenges of Machine Learning

1. Insufficient Quantity of Data

- Michele Banko and Eric Brill stated in their 2001 paper that their "results demonstrated that we may want to reconsider the trade off between spending time and money on algorithm development versus spending it on corpus development"
- Their results demonstrated that different ML algorithms perform similarly on complex problems of natural language disambiguation once they had enough data.
- Supported by Peter Norvig paper in 2009 entitled The Unreasonable Effectiveness of Data"

Main Challenges of Machine Learning

2. Non Representative Data

- Small sample sizes lead to **sampling noise**
- Large samples can still contain **sampling bias** if the sampling method is flawed.
- Example: US 1936 General Election –Landon v Roosevelt
- Literary Digest polled 10 million people, response rate 24%.
- Predicted that Landon would win with 57% of the vote.
- Actual – Roosevelt won with 62%

Main Challenges of Machine Learning

3. Poor Quality Data

- Data contains significant errors, outliers, and noise.
- Clean the data
  - Consider discarding outliers and fixing errors manually
  - If instances are missing features, you must either discard this feature or fill in the missing values
    - E.g. x% of respondents did not specify age

4. Irrelevant Features

- Garbage In Garbage Out
- Irrelevant Features slow down learning and can degrade performance
- **Feature Engineering:**
  - Feature Selection
  - Feature Extraction: combining exiting features to produce a more useful one, maybe with dimensionality reduction techniques
  - Creating new features requires new data

Main Challenges of Machine Learning

5. **Overfitting the Training Data**

- Model performs well on the training data but it does not generalise well.
- Figure shows a high-degree polynomial life satisfaction model that strongly overfits the training data
  - How well can it generalise? Would you trust its predictions?
  - Predicting a value is an example of regression
- Perhaps a simpler linear model would have sufficed!

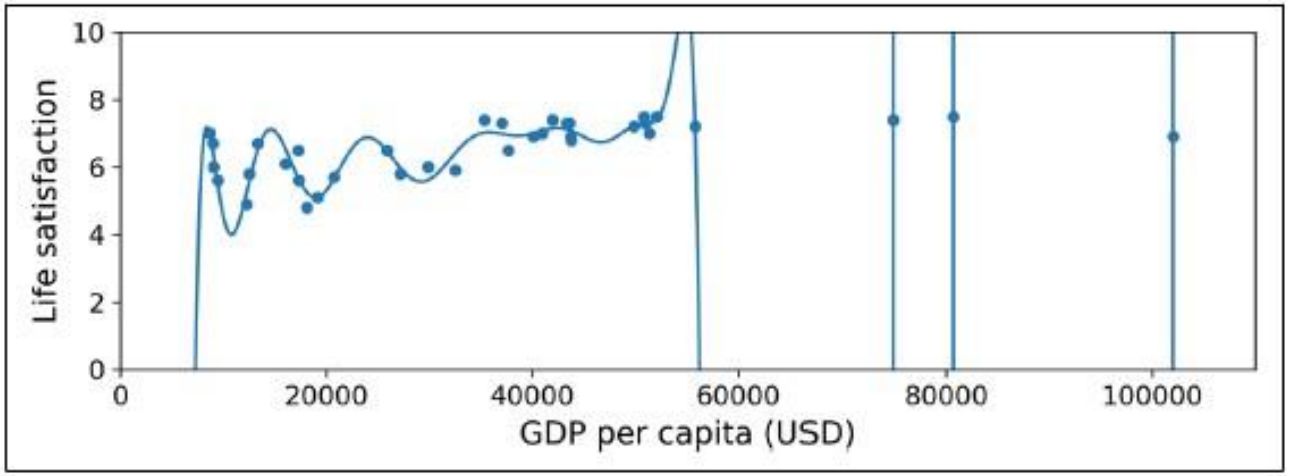


Figure: reprinted with kind permission of Aurélien Géron (2019)

Main Challenges of Machine Learning

- Overfitting happens when the model is too complex relative to the amount and quality of the training data.
  - Simplify the model by using fewer parameters, by reducing the features
  - Gather more training data
  - Reduce the noise in the training data by fixing errors and removing outliers
  - Constrain the model through regularisation

Main Challenges of Machine Learning

6. Underfitting the Training Data

- When the model is too simple to learn the underlying structure of the data
- A linear model of life satisfaction tends to underfit
- Solution
  - Use a more powerful model with more parameters
  - Feature engineering
  - Reduce regularistaion
    - Dropout
    - L1 and L2
    - Minibatch

Weight Initialisation

Weight Initialisation

- Glorot and Bengio suggest that
  1. The variance of the outputs of each layer to be equal to the variance of its inputs,
  2. The gradients to have equal variance before and after flowing through a layer in the reverse direction.
- The layer has an equal number of inputs and outputs
  - The *fan-in* and *fan-out* of the layer,
- The authors proposed an alternative that has proven to work very well in practice:
- Called *Xavier initialization* or *Glorot initialization* after the paper's first author

Weight Initialisation

- The connection weights of each layer must be initialized randomly as follows:
- **Normal distribution** with mean 0 and variance $\sigma^2 = \frac{2}{fan_{avg}}$
- **Uniform distribution** between –r and +r with $r = \sqrt{\frac{3}{fan_{avg}}}$
- Replacing fan_{avg} with fan_{in} gives LeCun's initialisation from the early 1990s.
- Using Glorot initialization can speed up training considerably, and it is one of the tricks of the trade that led to the success of Deep Learning

Initialization	Activation functions	σ^2 (Normal)
Glorot	None, tanh, logistic, softmax	$1 / fan_{avg}$
He	ReLU and variants	$2 / fan_{in}$
LeCun	SELU	$1 / fan_{in}$

Figure: reprinted with kind permission of Aurélien Géron (2019)

Weight Initialisation

- Keras uses Glorot initialization with a uniform distribution.
- You can change this to He initialization by setting kernel_initializer = "he_uniform" or kernel_initializer = "he_normal"
- keras.layers.Dense(10, activation="relu", kernel_initializer="he_normal")

Kaiming He et al., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *Proceedings of the 2015 IEEE International Conference on Computer Vision* (2015): 1026–1034.

Testing and Validation

- How to be confident that a production model will generalise well
- Split data into training set and test set
  - Train model on training set
  - Test on test set for generalisation error (out of band sample error)
- Typical rations are 80% for train, 20% for test
- If the training error is low but generalisation error is high → overfitting
- Not sure which model to use?
  - For example linear model and polynomial model
  - Use Holdout Validation
  - Hold out part of the training set to evaluate several candidate models and select the best one
  - Know as validation set or development set
  - Train multiple models on the reduced training set
  - Select the model that performs best on the validation set
  - Train the best model on the full training set with validation set put back.
  - Evaluate the final model on the test set

Objectives

Section 2:

- ❑ Batch Normalisation
- ❑ Gradient Clipping
- ❑ Transfer Learning
- ❑ Adversarial Data
- ❑ Architectures

Batch Normalisation

Batch Normalisation

- Using weight initialisation strategies such as Glorot and He initialisation, and ReLU type activation can reduce vanishing gradients, more is needed.
- Ioffe and Szegedy proposed Batch Normalisation in 2015.
- Add an operation before or after the activation function in the hidden layers
- It zero centres and normalises each input
- Then scales and shifts the results using 2 parameter vectors per layer
- Adding a BN layer as the very first layer eliminates the need to use a Standard Scaler to normalise the training set.
- It does this by evaluating the means and standard deviations across the current mini-batch.

Batch Normalisation

- BN in Keras in a model with 2 hidden layers

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(300, activation="relu", kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(100, activation="relu", kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(10, activation="softmax")])
```

- Each BN layer has 4 parameters per input
  a) γ is the output scale parameter vector for the layer
  b) β is the output shift parameter vector for the layer
  c) μ is the vector of input means for the minibatch
  d) σ is the vector of input standard deviations
- First BN layer = 4 x (28 x 28) = 4 x 784 = 3136 parameters

Batch Normalisation

- μ and σ are non-trainable
- (3136 + 1200 + 400) /2 = 2368
- BN standard practice, not shown in diagrams,
- Recent paper by Zhang et al. (2019) may change this, used fixed-upadte (fixup) weight initialisation without BN for a network of 10K layers.

```
[ ]  model.summary()

Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_11 (Flatten)         (None, 784)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 784)               3136
_________________________________________________________________
dense_244 (Dense)            (None, 300)               235500
_________________________________________________________________
batch_normalization_1 (Batch (None, 300)               1200
_________________________________________________________________
dense_245 (Dense)            (None, 100)               30100
_________________________________________________________________
batch_normalization_2 (Batch (None, 100)               400
_________________________________________________________________
dense_246 (Dense)            (None, 10)                1010
=================================================================
Total params: 271,346
Trainable params: 268,978
Non-trainable params: 2,368
_________________________________________________________________
```

```
bn1 = model.layers[1]
[(var.name, var.trainable) for var in bn1.variables]
```

```
[('batch_normalization/gamma:0', True),
 ('batch_normalization/beta:0', True),
 ('batch_normalization/moving_mean:0', False),
 ('batch_normalization/moving_variance:0', False)]
```

Gradient Clipping

Gradient Clipping

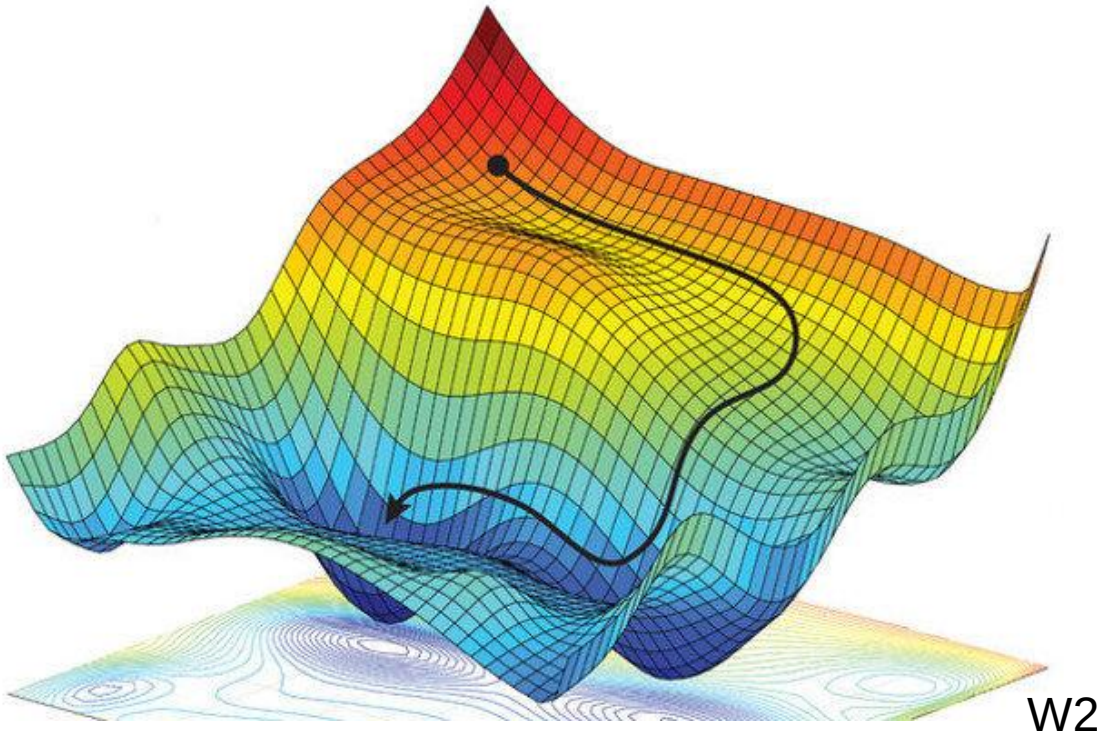
- To fix the exploding gradient problem
- Clips gradients during backprop so that they never exceed a threshold
- Most often used in Recurrent Neural Networks as BN is difficult to use in RNNs.
- Optimiser clips every gradient to a value between -1 and +1.
- Partial derivative of every parameter will be clipped between -1 and +1
- Sample gradient vector [0.9, 100] where 2nd component (axis) is dominant
- ClipValue → [0.9, 1.0] loss of dominant axis information
- ClipNorm → [0.008999, 0.99995] maintains dominant axis information

```
optimizer = keras.optimizers.SGD(clipvalue=1.0)
```

```
optimizer = keras.optimizers.SGD(clipnorm=1.0)
```

Gradient Clipping

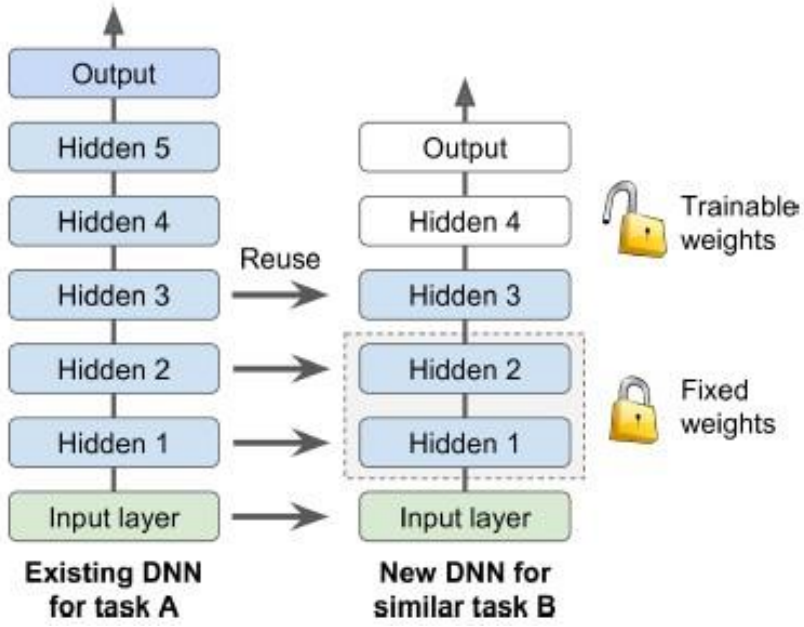
- Shows 2D Error Surface
- What is a dominant axis?



https://www.researchgate.net/publication/325142728_Spatial_Uncertainty_Sampling_for_End-to-End_Control

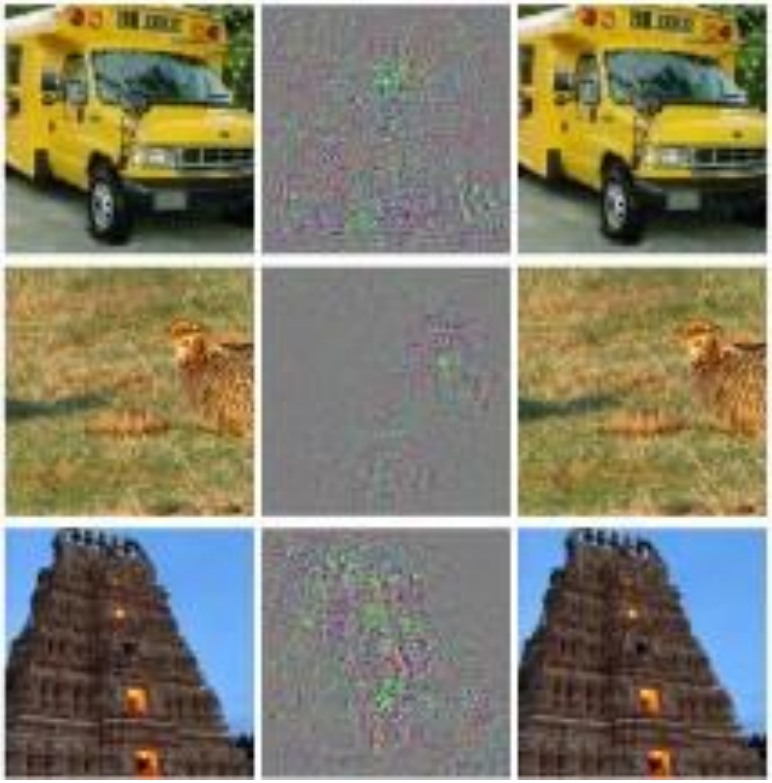
Transfer Learning

- Expensive to train from scratch
- Reuse of pre-trained layers for similar tasks
- Requires significantly less training data
- Freeze lower hidden layers
- Output layer and upper hidden layer need to be retrained/replaced
- May have to drop top hidden lay
- Standard practice in DNNs
- See BAIR Model Zoo
- https://github.com/BVLC/caffe/wiki/Model-Zoo



Adversarial Data

- Not all is rosy in the garden
- Intriguing properties of neural networks, by Szegedy et al.(2013)
- Right: original correctly classified left: adversarial (perturbed) images incorrectly classified.
- What is the middle column illustrating?
- For more see: DAmageNet: A Universal Adversarial Dataset by Chen et al. (2019)
- Just one of many



https://arxiv.org/pdf/1312.6199.pdf

ILSVRC

- 1.2 million images from 1000 categories

Top 5 accuracy

- 2012: 84.7% Krizhevsky, Sutskever and Hinton (KSH)
  - Top 1 Accuracy: 63.3%
- 2013: 88.3%
- 2014: 93.33% GoogleLeNet with 22 layers
  - VGG runner up with 92.7%
- 2015: 96.7% ResNet

- Ground Standard / Ground Truths
- Labelling:
  - "I became very good at identifying breeds of dogs... Based on the sample of images I worked on, the GoogLeNet classification error turned out to be 6.8%... My own error in the end turned out to be 5.1%, approximately 1.7% better. "
  - https://arxiv.org/abs/1409.0575 (2015)

(a) Siberian husky

(b) Eskimo dog

https://arxiv.org/abs/1409.4842

AlexNet

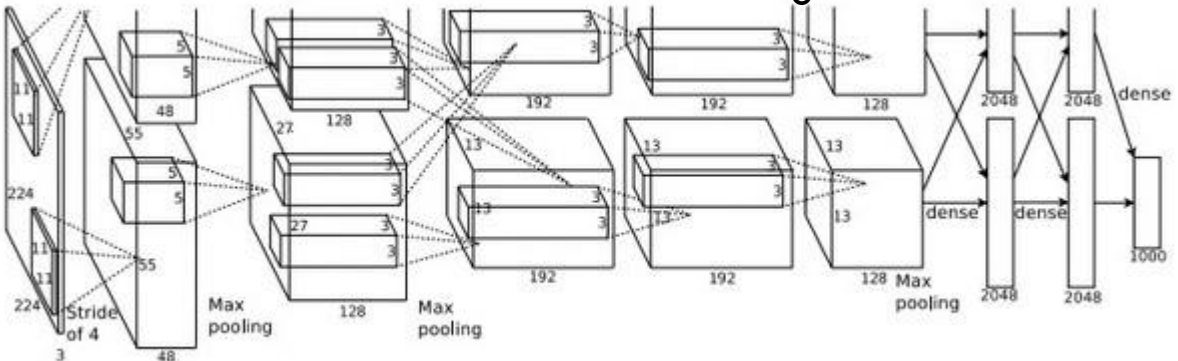
- 2012: 84.7% Krizhevsky, Sutskever and Hinton (KSH)
- Stacked convolutional layers on top of each other
- To reduce overfitting, the authors used two regularization techniques:
  - Dropout with a 50% dropout rate during training to the outputs of layers F9 and F10.
  - *Data augmentation* by randomly shifting the training images by various offsets, flipping them horizontally, and changing the lighting conditions.

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	–	1,000	–	–	–	Softmax
F10	Fully connected	–	4,096	–	–	–	ReLU
F9	Fully connected	–	4,096	–	–	–	ReLU
S8	Max pooling	256	6 × 6	3 × 3	2	valid	–
C7	Convolution	256	13 × 13	3 × 3	1	same	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	same	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	same	ReLU
S4	Max pooling	256	13 × 13	3 × 3	2	valid	–
C3	Convolution	256	27 × 27	5 × 5	1	same	ReLU
S2	Max pooling	96	27 × 27	3 × 3	2	valid	–
C1	Convolution	96	55 × 55	11 × 11	4	valid	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

AlexNet

- 2012:
- Consists of 7 layers of hidden neurons
- First 5 are convolutional layers
- Next 2 are fully connected
- Output is a 1000 unit softmax
- Input contains 3 x 224 x 224
- ImageNet contains images of different sizes
- Rescaled each image so that shorter side had length 256
- Then cropped 256 x 256 from the centre of the rescaled image

- Subsampled "random" 224 x 224 → data augmentation
- First hidden layer uses receptive fields of size 11 x 11 with stride 4 → 96 feature maps
- Split across 2 GPUs, hence 48 shown in diagram
- Maxpooling is 3x3 with overlapping
- Second hidden layer uses receptive fields of size 5 x5 → 256 feature maps
  - Feature map uses 48 inputs
- Split across 2 GPUs, hence 128 shown in diagram



Getting a Handle on Parameters

HYPOTHETICAL

- Imagine that the Input = image 150x100 x 3
  - 3 channels for RGB
- A convolutional layer with 5x5 kernels (filters) stride 1, outputting 200 feature maps of size 150x100
- Parameters = ((5 x 5 x 3)+ 1) x 200
- = 76 * 200 = 15,200
- Each feature map contains 150 x 100 neurons
- Each neuron computes a sum of 5x5x3 = 75 inputs
- Total computations 75 x (150 x 100) x 200 = 225 million
- 32 bit floating point
- Output = 200 x 150 x 100 x 32 = 96 million bits = 12 MB RAM per training instance
- Training Batch of size 100
- RAM = (12 MB x 100)1.2GB RAM for one convolutional layer
- During training, everything computed in the forward pass needs to be preserved for backprop

Getting a Handle on Parameters

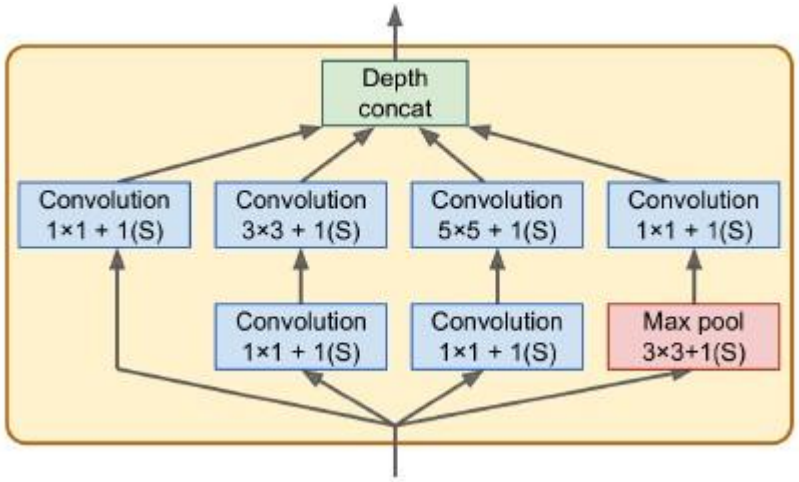
Out of Memory error:

- Reduce minbatch size
- Reduce dimensionality using stride > 1
- Remove some layers
- Use 16 bit floats instead of 32 bit
- Use multiple compute nodes
  - AlexNet distributed across 2 GPUs

Google LeNet Using Inception Modules

GoogleLeNet

- Winner of 22014 ISLRVC
- Top 5 error less than 7%
- Much deeper network than previous CNNs
- Fewer parameters 6 million instead of 60 million in AlexNet
- Inception Modules
- 3x3 + 1 (s) = 3x3 kernel with stride 1 and same padding
- ReLUs
- Outputs same dimensions as inputs
- tf.concat() to stack feature map from top 4 convolutional layers



An Inception Module

GoogleLeNet

1. **Deep neural networks should be "large" in terms of layers and units.**
2. Convolutional neural networks benefit from extracting features at varying scales.
    1. The biological human visual cortex functions by identifying patterns at different scales leading to better performance.
    2. Therefore **multi-scale convnet have the potential to learn more.**
3. Hebbian Principle — *neurons that fire together, wire together.*

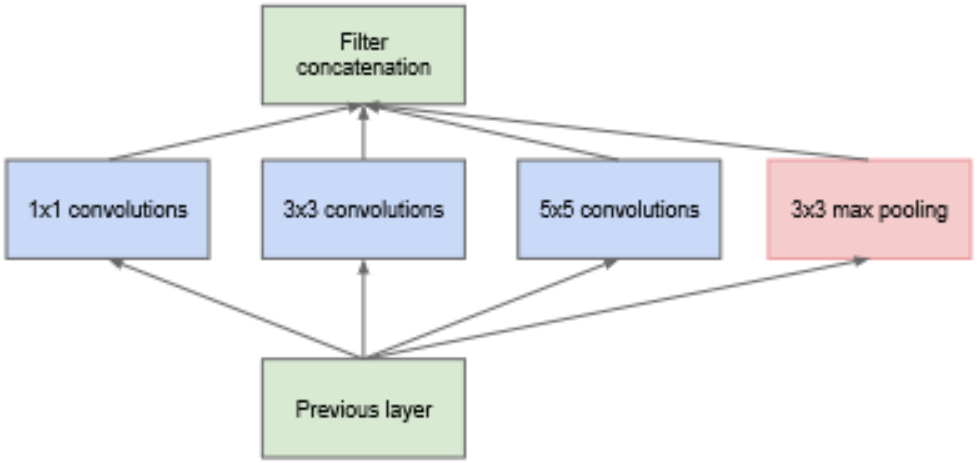
Downisde: computational cost and overfitting

GoogleLeNet: 1x1 Convolutions

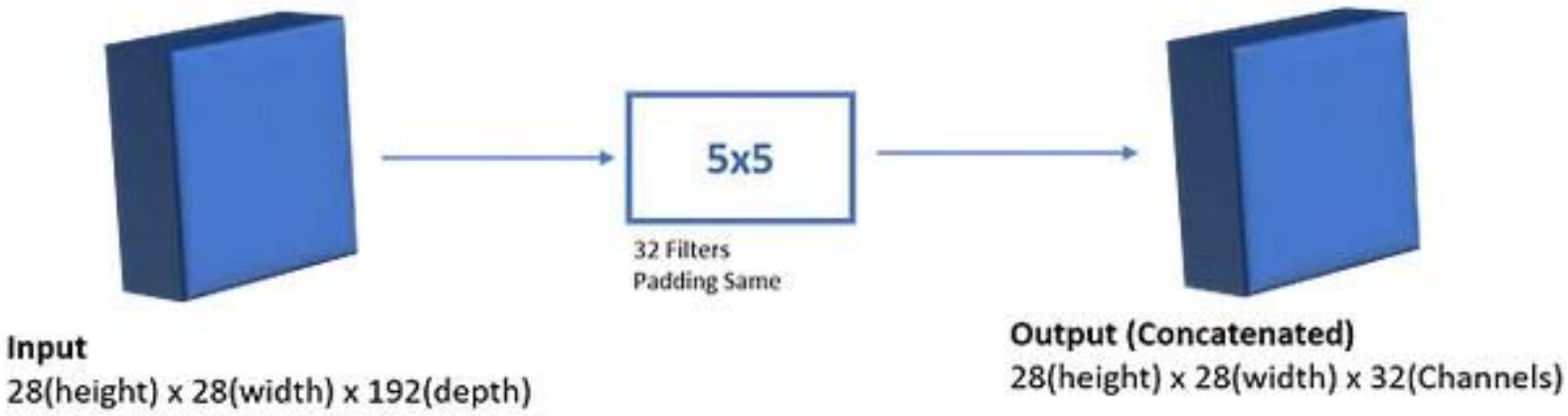
- *Reduce the dimensions of data passing through the network, which provides the additional benefit of an increase in the width and depth of the network.*
- 1x1 convolution sometimes referred to as "*Network In Network*", was introduced in 2013 in this paper by Lin et al.
- 1x1 filter does not learn any spatial patterns that occur within the image
- 1x1 learn patterns across the depth(cross channel) of the image.
- 1x1 provide a method dimension reduction, but they also provide the additional benefit of enabling the network to learn more

GoogleLeNet: Naive Inception

- *Inception Module consists of:*
  - *Input layer*
  - *1x1 convolution layer*
  - *3x3 convolution layer*
  - *5x5 convolution layer*
  - ***Max pooling layer***
  - ***Concatenation layer***

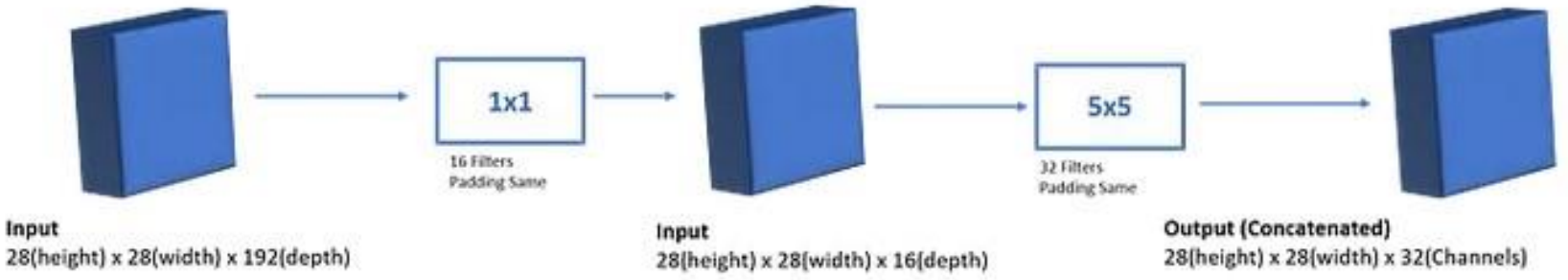


GoogleLeNet: Naive Inception



- Num ops: multiply the number of outputs that are required to be provided (28x28x32), with the number of multipliers needed to work out a single value within the output (5x5x192).
- Num multiplier ops = (output dimensions) * (filter dimensions) * (depth of input channel)
- Num multiplier ops = (28 x 28 x 32) x (5 x 5) x (192)
- = 120, 244, 400

GoogleLeNet



- Num multiplication ops = ((28 x 28 x 16) * (1 x 1 x 192)) + ((28 x 28 x 32) * (5 x 5 x 15))
- Num multiplication ops =(2,408,448)+ (10,976,000)
- = 12,443,648.

GoogleLeNet

- **Benefits of the Inception Module**
- High-performance gain on convolutional neural networks
- Efficient utilisation of computing resource with minimal increase in computation load for the high-performance output of an Inception network.
- Ability to extract features from input data at varying scales through the utilisation of varying convolutional filter sizes.
- 1x1 conv filters learn cross channels patterns, which contributes to the overall feature extractions capabilities of the network.

- Source: https://towardsdatascience.com/deep-learning-googlenet-explained-de8861c82765

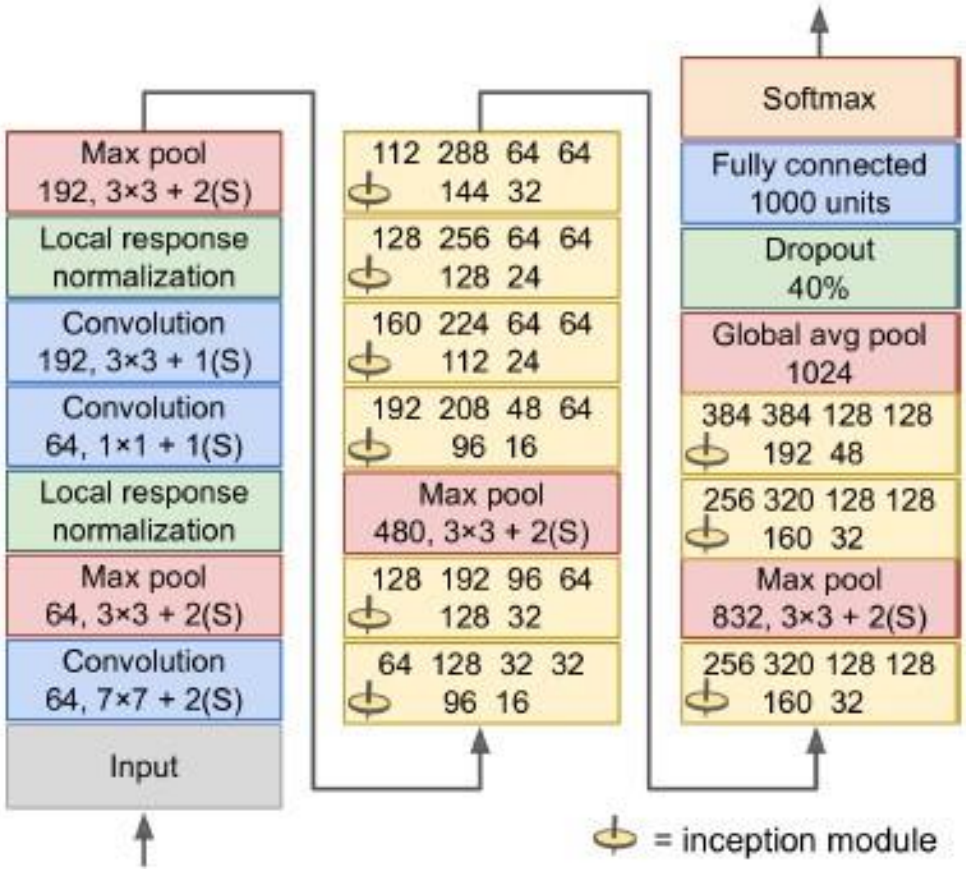
GoogleLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43022.pdf

GoogleLeNet

- The number of kernels in each inception module is a hyperparameter
- 22 layers with a softmax output
- Normalisation layer is to enhance features and reduce unstable gradients
- Maxpooling used for dimensionality reduction to speed up training
- Variants: InceptionV3 and Inception V4



Reprinted with kind permission Géron (2019).

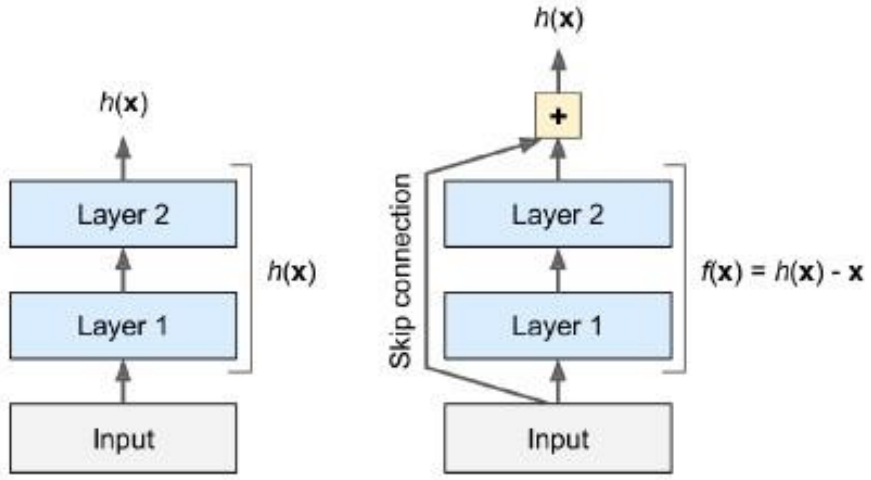
VGGNet

- Runners up in 2014 with 92.7% test accuracy
- Simonyan and Zisseerman from Visual Geometry Group (VGG) at Oxford
- 2 or 3 convolutional layers and a pooling layer repeated
- Up to a total of 16 or 19 convolutional layers
- Finishes with a final dense network with 2 hidden layers and the output layer,
- Uses 3x3 filters.
- https://arxiv.org/pdf/1409.1556.pdf

ResNet

ResNet

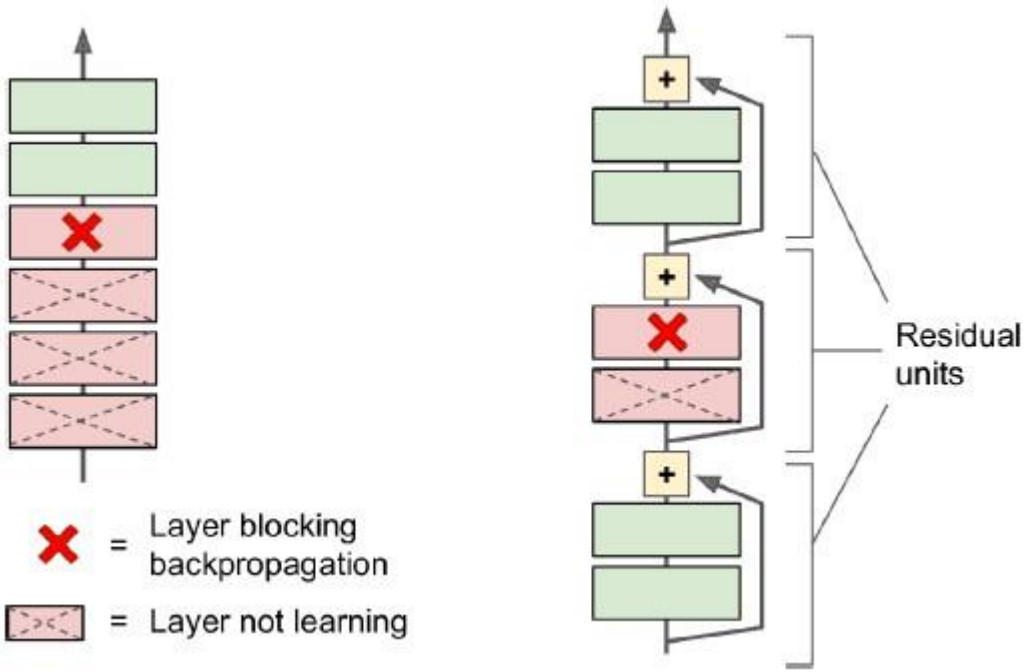
- He et al won ILSVRC 2015 with top 5 error rate of 3.6%
- DNN with 152 layers
- Variants have 34, 50, and 101 – Resnet50, Resnet101
- Deeper models with fewer parameters
- Use of skip (shortcut) connections
- Input into a layer is also added to the outpu of a layer higher up layer in the stack
- Based on neurophysiology. Based on concepts from pyramidal cells in the cerebral cortex.
- Two reasons for skip connections:
  - To avoid the problem of vanishing gradients
  - To mitigate the Degradation (accuracy saturation) problem;
    - Adding more layers to a suitably deep model leads to higher training error.



Left: traditional DNN. Right: skip connection in ResNet.

ResNet

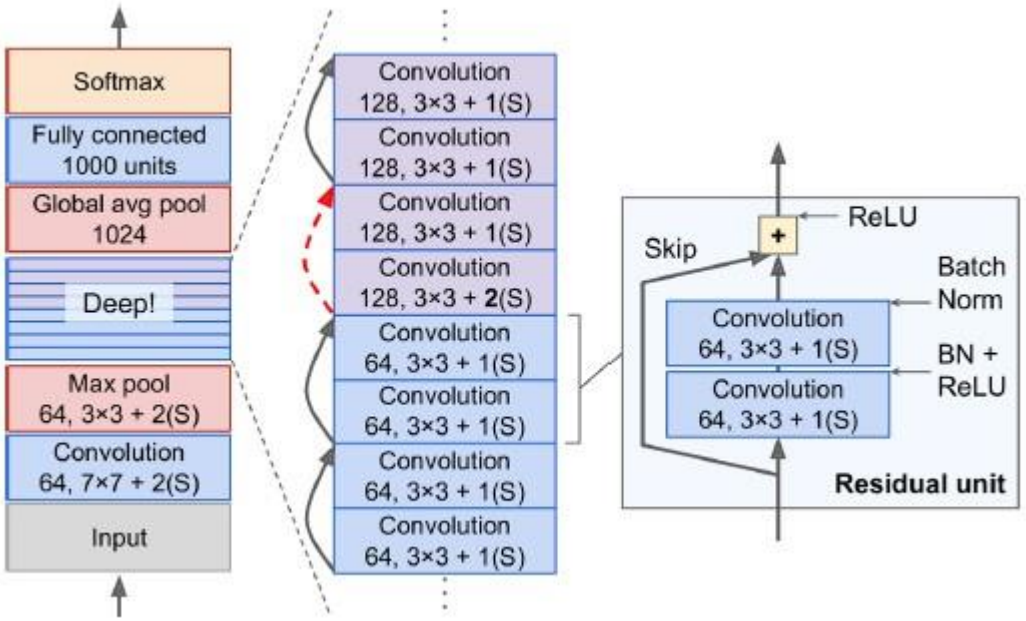
- ResNet speeds up learning
- It allows layers to make progress on learning even if previous layers are blocking due to vanishing gradients



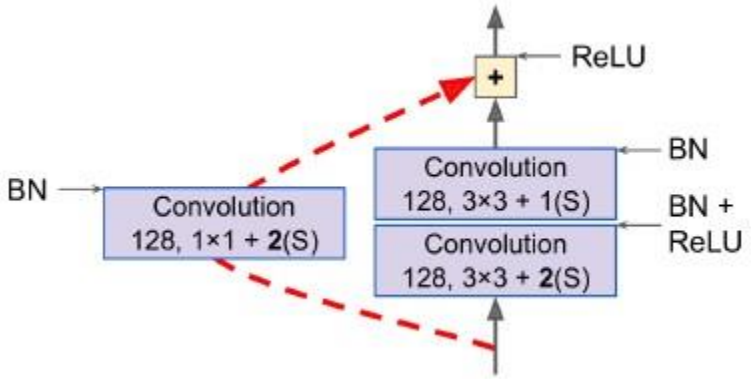
Reprinted with kind permission Géron (2019).

ResNet

- The architecture is starts and ends like GoogleLeNet but with deep stack of residual units in between
- Each residual unit composed of 2 convolutional layers and Batch Normalisation but no pooling layer. ReLUs and 3x3 kernels, stride 1 and same padding
- Number of feature maps doubled every few residual units at the same time as width and height reduced
- Inputs cannot be added to outputs
- Solution: 1x1 with stride 2



Reprinted with kind permission Géron (2019).



ILSVRC

- Top 5 Error
- 2017: < 3% CUI Image Team from the Chinese University of Hong Kong
- 2018: 2.25% Squeeze and Excitation Network (SENet) by Chinese Academy of Science Beijing and VGG Team Oxford

CNNs
continue
d

Objectives

Section 3:

- ResNet-34 Implementation
- Pretrained Models
- Transfer Learning

Preliminaries

- One of the central abstractions in Keras is the layer class.
- A layer encapsulates both a state (the layer's "weights") and a transformation from inputs to outputs (a "call", the layer's forward pass).
- An example of a densely connected layer

```
1. class Linear(keras.layers.Layer):

2.     def __init__(self, units=32, input_dim=32): (Linear, self).__init__()
3.         w_init = tf.random_normal_initializer()
4.         self.w = tf.Variable( initial_value=w_init(shape=(input_dim, units), dtype="float32"), trainable=True, )
5.         b_init = tf.zeros_initializer()
6.         self.b = tf.Variable( initial_value=b_init(shape=(units,), dtype="float32"), trainable=True)

7.     def call(self, inputs):
8.         return tf.matmul(inputs, self.w) + self.b
```

Preliminaries

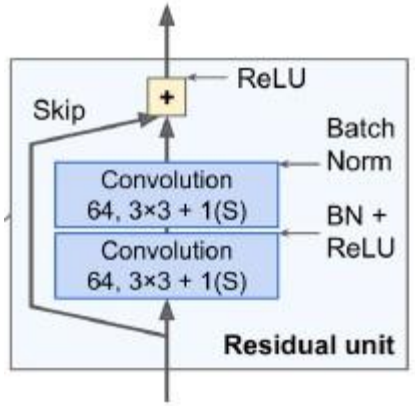
- You use a layer by calling it on some tensor input(s).

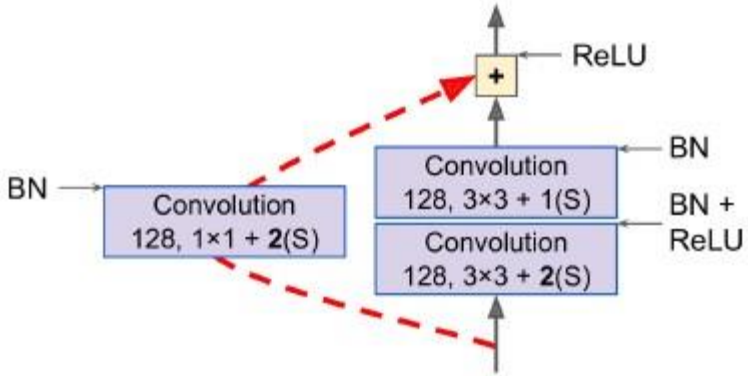
- x = tf.ones((2, 2))
- linear_layer = Linear(4, 2)
- y = linear_layer(x)
- print(y)

```
tf.Tensor(
[[ 0.01103698  0.03099662 -0.1009444   0.10721317]
 [ 0.01103698  0.03099662 -0.1009444   0.10721317]], shape=(2, 4), dtype=float32)
```

Implementing ResNet-34

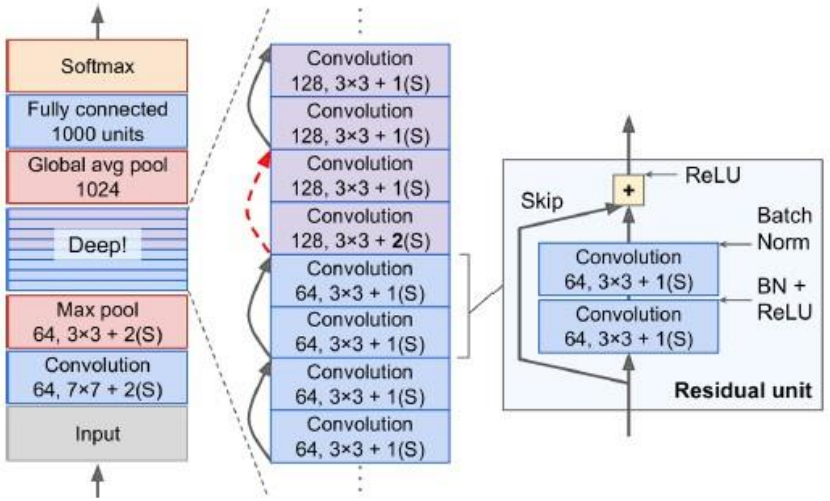

```
1.  DefaultConv2D = partial(keras.layers.Conv2D, kernel_size=3, strides=1,
    padding="SAME", use_bias=False)

2.  class ResidualUnit(keras.layers.Layer):
3.      def __init__(self, filters, strides=1, activation="relu", **kwargs):
4.          super().__init__(**kwargs)
5.          self.activation = keras.activations.get(activation)
6.          self.main_layers = [
7.              DefaultConv2D(filters, strides=strides),
8.              keras.layers.BatchNormalization(),
9.              self.activation,
10.             DefaultConv2D(filters),
11.             keras.layers.BatchNormalization()]
12.         self.skip_layers = [ ]
13.         if strides > 1:
14.             self.skip_layers = [
15.                 DefaultConv2D(filters, kernel_size=1, strides=strides),
16.                 keras.layers.BatchNormalization()]
```

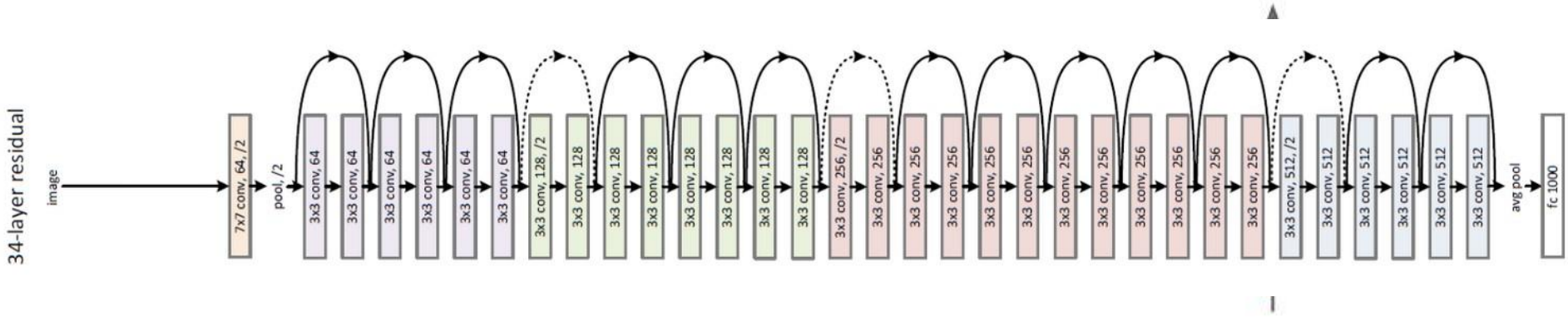




Implementing ResNet-34


```
17.     def call(self, inputs):
18.         Z = inputs
19.         for layer in self.main_layers:
20.             Z = layer(Z)
21.             skip_Z = inputs
22.             for layer in self.skip_layers:
23.                 skip_Z = layer(skip_Z)
24.             return self.activation(Z + skip_Z)
```

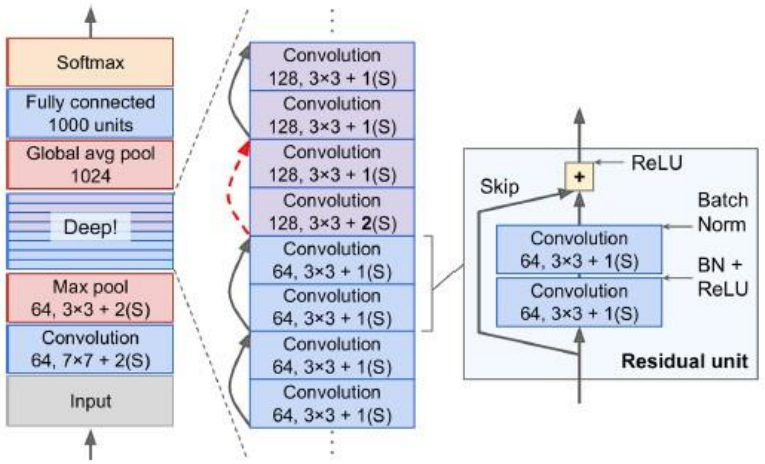




Implementing ResNet-34

- ResNet-34 is the ResNet with 34 layers (only counting the convolutional layers and the fully connected layer)
  - 17 containing 3 residual units that output 64 feature maps, 4 RUs with 128 maps, 6 RUs with 256 maps, and 3 RUs with 512 maps.

```
model = keras.models.Sequential()
model.add(DefaultConv2D(64, kernel_size=7, strides=2,  input_shape=[224, 224, 3]))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation("relu"))
model.add(keras.layers.MaxPool2D(pool_size=3, strides=2, padding="SAME"))
prev_filters = 64
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    model.add(ResidualUnit(filters, strides=strides))
    prev_filters = filters
model.add(keras.layers.GlobalAvgPool2D())
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation="softmax"))
```



ResNet-34

- Less than 40 LOCs to implement ResNet architecture in Keras

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 112, 112, 64)	9408
batch_normalization (BatchNo	(None, 112, 112, 64)	256
activation (Activation)	(None, 112, 112, 64)	0
max_pooling2d (MaxPooling2D)	(None, 56, 56, 64)	0
residual_unit (ResidualUnit)	(None, 56, 56, 64)	74240
residual_unit_1 (ResidualUni	(None, 56, 56, 64)	74240
residual_unit_2 (ResidualUni	(None, 56, 56, 64)	74240
residual_unit_3 (ResidualUni	(None, 28, 28, 128)	230912
residual_unit_4 (ResidualUni	(None, 28, 28, 128)	295936
residual_unit_5 (ResidualUni	(None, 28, 28, 128)	295936
residual_unit_6 (ResidualUni	(None, 28, 28, 128)	295936
residual_unit_7 (ResidualUni	(None, 14, 14, 256)	920576
residual_unit_8 (ResidualUni	(None, 14, 14, 256)	1181696
residual_unit_9 (ResidualUni	(None, 14, 14, 256)	1181696
residual_unit_10 (ResidualUn	(None, 14, 14, 256)	1181696
residual_unit_11 (ResidualUn	(None, 14, 14, 256)	1181696
residual_unit_12 (ResidualUn	(None, 14, 14, 256)	1181696
residual_unit_13 (ResidualUn	(None, 7, 7, 512)	3676160
residual_unit_14 (ResidualUn	(None, 7, 7, 512)	4722688
residual_unit_15 (ResidualUn	(None, 7, 7, 512)	4722688
global_average_pooling2d (Gl	(None, 512)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 10)	5130

Total params: 21,306,826
Trainable params: 21,289,802
Non-trainable params: 17,024

Using Pretrained Models

- In general, you won't have to implement standard models like GoogLeNet or ResNet manually, since pretrained networks are readily available with a single line of code in the keras.applications package.
- For example, you can load the ResNet-50 model, pretrained on ImageNet, with the following line of code:

```
model = keras.applications.resnet50.ResNet50(weights="imagenet")
```

- ResNet-50 model expects 224 × 224-pixel images (other models may expect other sizes, such as 299 × 299)
  - use TensorFlow's tf.image.resize() function to resize the images we loaded earlier:

```
images_resized = tf.image.resize(images, [224, 224])
```

- The tf.image.resize() will not preserve the aspect ratio. If this is a problem, try cropping the images to the appropriate aspect ratio before resizing. Both operations can be done in one shot with

```
tf.image.crop_and_resize().
```

Using Pretrained Models

- Each model provides a preprocess_input() function that you can use to preprocess your images.
  - These functions assume that the pixel values range from 0 to 255, so we must multiply them by 255 (since earlier we scaled them to the 0–1 range):

```
inputs = keras.applications.resnet50.preprocess_input(images_resized * 255)
```

- Other vision models are available in keras.applications, including several ResNet variants, GoogLeNet variants like Inception-v3 and Xception, VGGNet variants, and MobileNet and MobileNetV2 (lightweight models for use in mobile applications).

- Now we can use the pretrained model to make predictions:

```
Y_proba = model.predict(inputs)
```

Using Pretrained Models

- If you want to display the top *K* predictions, including the class name and the estimated probability of each predicted class, use the decode_predictions() function.
- For each image, it returns an array containing the top *K* predictions, where each prediction is represented as an array containing the class identifier, its name, and the corresponding confidence score:

```
1. top_K = keras.applications.resnet50.decode_predictions(Y_proba, top=3)
2. for image_index in range(len(images)):
3.     print("Image #{}".format(image_index))
4.     for class_id, name, y_proba in top_K[image_index]:
5.     print(" {} - {:12s} {:.2f}%".format(class_id, name, y_proba * 100))
```

```
Image #0
  n03877845 - palace       42.87%
  n02825657 - bell_cote    40.57%
  n03781244 - monastery    14.56%
```

Transfer Learning

- Train a model to classify pictures of flowers, reusing a pretrained Xception model.

- Load the data set

```
1. import tensorflow_datasets as tfds
2. dataset, info = tfds.load("tf_flowers", as_supervised=True,
   with_info=True)
3. dataset_size = info.splits["train"].num_examples # 3670
4. class_names = info.features["label"].names # ["dandelion", "daisy",
   ...]
5. n_classes = info.features["label"].num_classes # 5
```

Transfer Learning

- The Data Set
  - https://www.tensorflow.org/tutorials/load_data/images


```
[4]  image_count = len(list(data_dir.glob('*/*.jpg')))
     print(image_count)

     3670
```

```
[11] class_names = train_ds.class_names
     print(class_names)

     ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

Transfer Learning

- There is only a "train" dataset, no test set or validation set, so we need to
- Split the training set.
- The TF Datasets project provides an API for this.
  - Use first 10% of the dataset for testing, the next 15% for validation, and the remaining 75% for training:

6.
```
test_set_raw, valid_set_raw, train_set_raw =
tfds.load("tf_flowers", split=["train[:10%]",
"train[10%:25%]", "train[25%:]"], as_supervised=True)
```

Transfer Learning

- Next we must preprocess the images. The CNN expects 224 × 224 images, so the data must be resized.
- The images must also be passed through Xception's preprocess_input() function:

```
def preprocess(image, label):
    resized_image = tf.image.resize(image, [224, 224])
    final_image = keras.applications.xception.preprocess_input(resized_image)
    return final_image, label
```

- Applied to all 3 sets

```
batch_size = 32
train_set = train_set.shuffle(1000)
train_set = train_set.map(preprocess).batch(batch_size).prefetch(1)
valid_set = valid_set.map(preprocess).batch(batch_size).prefetch(1)
test_set = test_set.map(preprocess).batch(batch_size).prefetch(1)
```

- Data Augmentation
  - tf.image.random_crop() to randomly crop the images, use
  - tf.image.random_flip_left_right() to randomly flip the images horizontally.

Transfer Learning

- Load an Xception model, pretrained on ImageNet.
- Exclude the top of the network by setting include_top=False: this excludes the global average pooling layer and the dense output layer.
- We then add our own global average pooling layer, based on the output of the base model, followed by a dense output layer with one unit per class, using the softmax activation function.
- Finally, we create the Keras Model:

```
16. base_model = keras.applications.xception.Xception(weights="imagenet",
17. include_top=False)
18. avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
19. output = keras.layers.Dense(n_classes, activation="softmax")(avg)
20. model = keras.Model(inputs=base_model.input, outputs=output)
```

Transfer Learning

- Freeze the weights of the pretrained layers, at least at the beginning of training:

```
for layer in base_model.layers:
layer.trainable = False
```

- Compile and run on a GPU

```
optimizer = keras.optimizers.SGD(lr=0.2, momentum=0.9,
decay=0.01)
model.compile(loss="sparse_categorical_crossentropy",
optimizer=optimizer,
metrics=["accuracy"])
history = model.fit(train_set, epochs=5, validation_data=valid_set)
```

- After training the model for a few epochs, its validation accuracy should reach about 75–80% and stop making much progress.

Transfer Learning

- Unfreezee some/all of the bottom layers.
- Compile and run

```
27.  for layer in base_model.layers:
28.      layer.trainable = True
29.  optimizer = keras.optimizers.SGD(lr=0.01, momentum=0.9, decay=0.001)
30.  model.compile(...)
31.  history = model.fit(...)
```

- This time we use a much lower learningrate to avoid damaging the pretrained weights
- Should eventually reach an accuracy of 95%.

CNNs

Foundational Architectures (Early Innovations)

1. **LeNet-5:** One of the earliest CNNs, foundational for digit recognition.
2. **AlexNet:** Popularized CNNs for image classification, winning ImageNet 2012.

Deeper and More Complex (Accuracy Focus)

3. **VGGNet:** Emphasized depth with small 3x3 convolutions, various versions (VGG16, VGG19).
4. **GoogLeNet (Inception):** Introduced Inception modules for efficient feature extraction at multiple scales.
5. **ResNet:** Addressed vanishing gradients with residual connections, enabling very deep networks.
6. **DenseNet:** Extreme version of ResNet, densely connecting layers for feature reuse.
7. **ResNeXt:** Improved ResNet with cardinality (number of parallel paths) for better performance.

CNNs

Mobile-Friendly (Efficiency Focus)

8. MobileNet: Designed for mobile and embedded devices, using depthwise separable convolutions.

9. ShuffleNet: Further optimized MobileNet with channel shuffle operations.

10. SqueezeNet: Reduced parameters with squeeze-and-excitation blocks.

Object Detection

11. YOLO (You Only Look Once): Single-stage object detection, known for speed.

12. SSD (Single Shot MultiBox Detector): Another fast object detection architecture.

13. Faster R-CNN: Two-stage detector with region proposal network, high accuracy.

CNNs

Segmentation

14. **U-Net:** Popular for biomedical image segmentation, with encoder-decoder structure.
15. **Mask R-CNN:** Extends Faster R-CNN for instance segmentation (pixel-level object masks).
16. **DeepLab**

Recent and Notable

16. **EfficientNet:** Focuses on scaling CNNs efficiently (width, depth, resolution).
17. **RegNet:** Designed for network design, finding optimal architectures through search.
18. **Vision Transformer (ViT):** While not strictly CNNs, they are relevant, using attention mechanisms for image tasks.
19. **YOLOv8:** The latest iteration of the YOLO object detection family.
20. **ConvNeXt:** Aims to bridge the gap between CNNs and Transformers, achieving strong performance.
21. **Xception:** extends Google LeNet Inception architecture.

CNNs

Important Notes:

- This list is not exhaustive, does not include hybrids, and new architectures are constantly being developed.
- The "best" architecture depends on the specific task, dataset, and computational resources.
- Many architectures build upon previous ones, incorporating ideas and improvements.

Summary

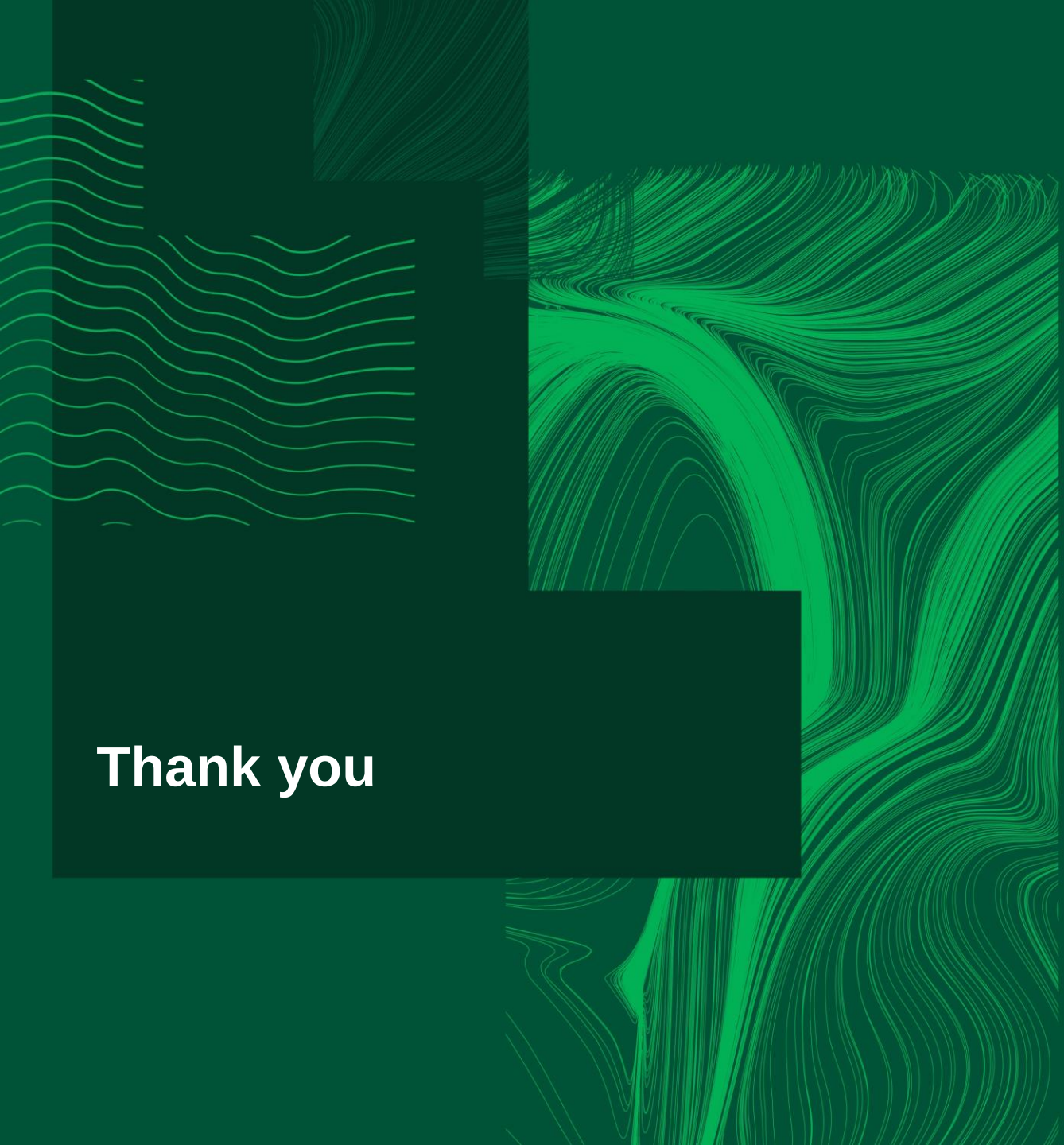
- Introduced ML Challenges
- And specific to ANNs/CNNs
  - Weight Normalisation
  - Batch Normalisation
  - Gradient Clipping
  - Inception and ResNet architectures
  - Code for ResNet34
  - Pre-trained Models and Transfer Learning.

Homework

Study these slides carefully

Chapters 11 and 14 in Aurelien Géron. Hands On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Ed. O'Reilly. 2019

Thank you



UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

University of Limerick,
Limerick, V94 T9PX,
Ireland.

Ollscoil Luimnigh,
Luimneach,
V94 T9PX, Éire.

+353 (0) 61 202020

ul.ie