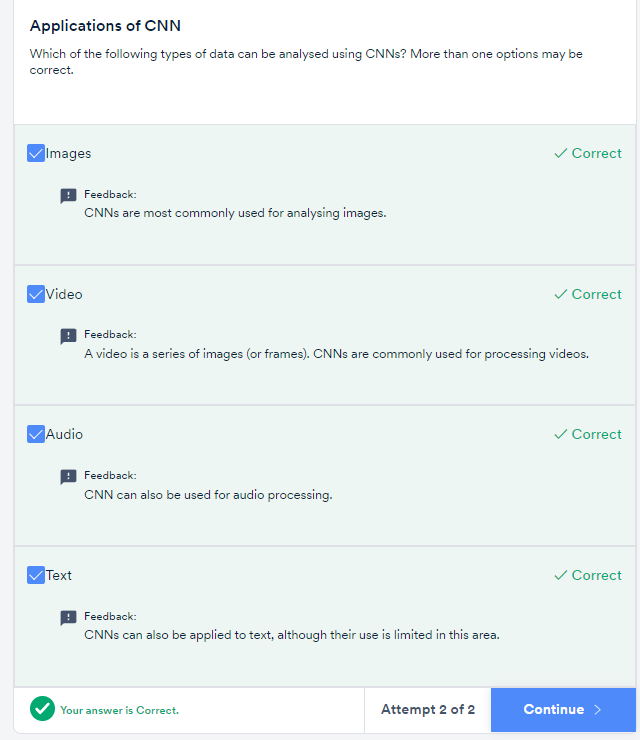
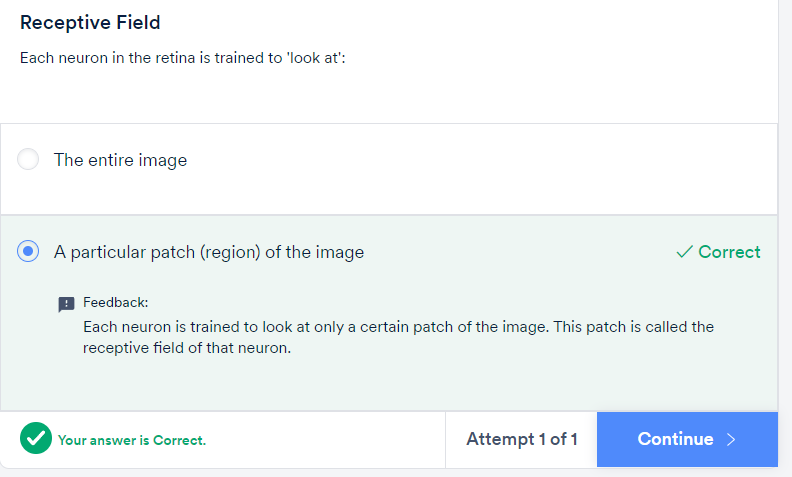
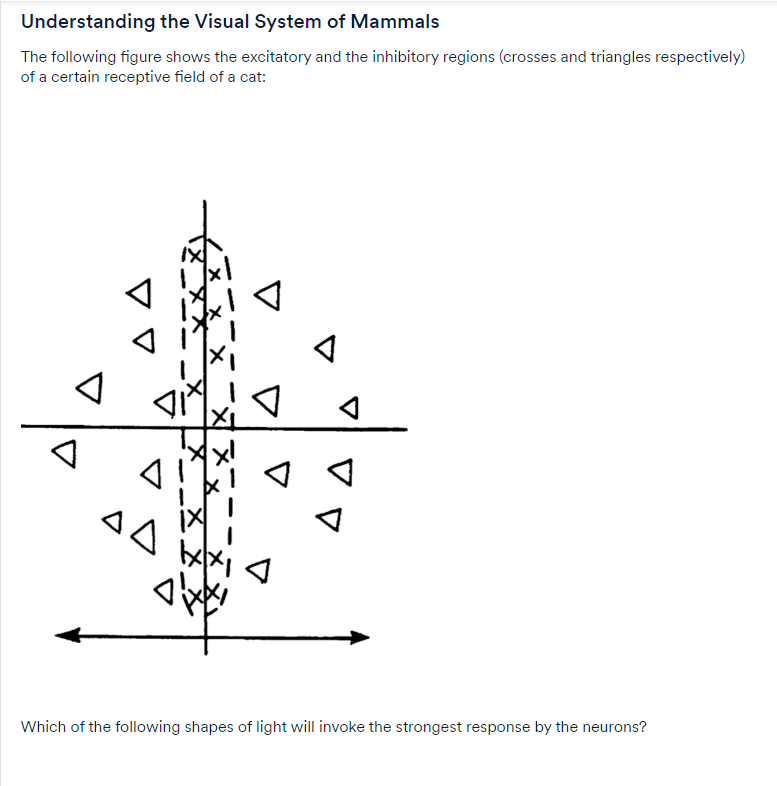
**Convolutional Neural Networks.**



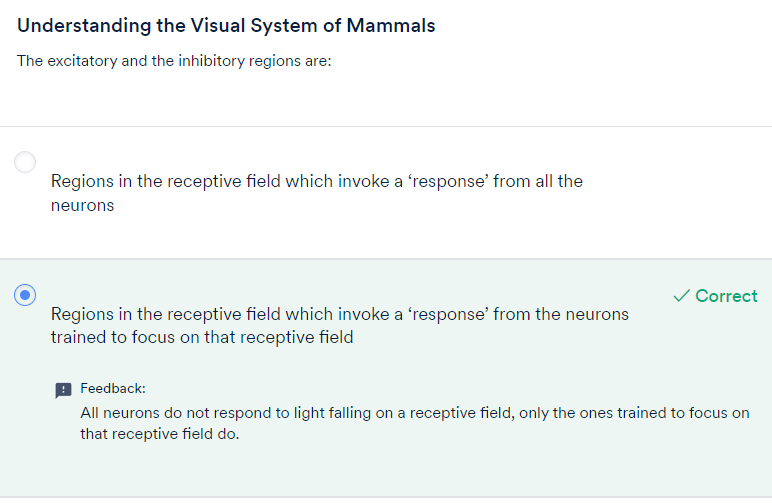
Some of the important observations made in the study were:

* Each neuron in the retina focuses on one part of the image and that part of the image is called the **receptive field of that neuron.**
* There are **excitatory and** **inhibitory regions** in the receptive field. The neurons only ‘fire’ when there is a **contrast between the excitatory and the inhibitory regions.** If we splash light over the excitatory and inhibitory regions together, because of no contrast between them, the neurons don’t ‘fire’ (respond). If we splash light just over the excitatory region, neurons respond because of the contrast.
* The **strength of the response**is proportionalto the summation over only the excitatory region (not inhibitory region). Later, you will study the **pooling layer in CNNs**which corresponds to this observation.









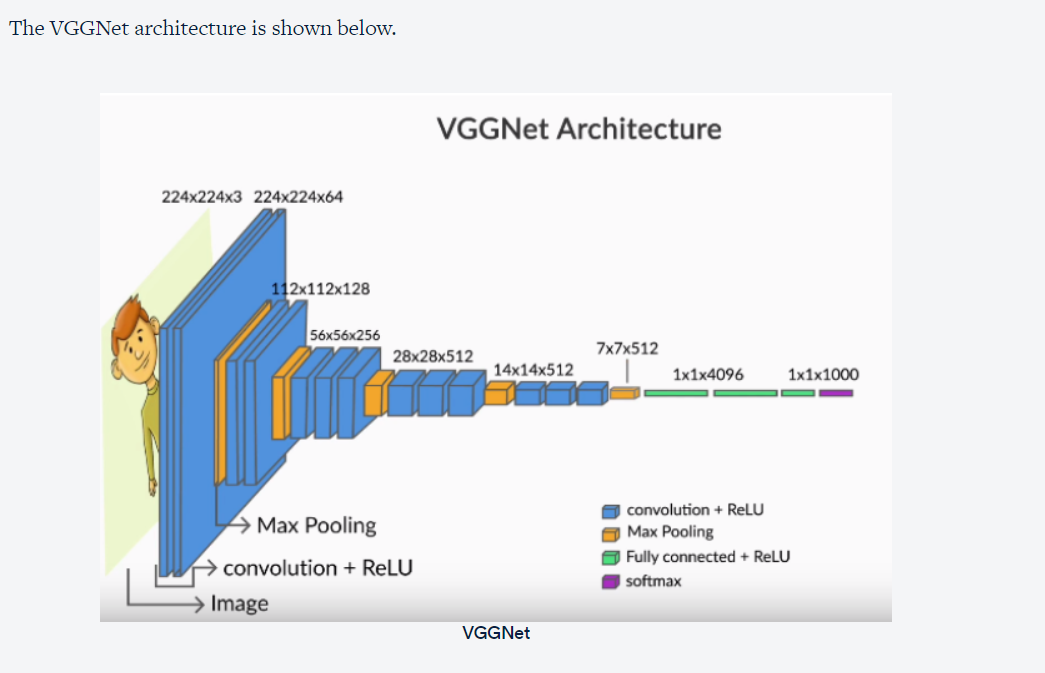
* The **receptive fields of all neurons are almost identical** in shape and size
* There is a **hierarchy in the units:** Units at the initial level do very basic tasks such as picking raw features (such as horizontal edges) in the image. The subsequent units extract more abstract features, such as identifying textures, detecting movement, etc. The layers 'higher' in the hierarchy typically **aggregate the features** in the lower ones.

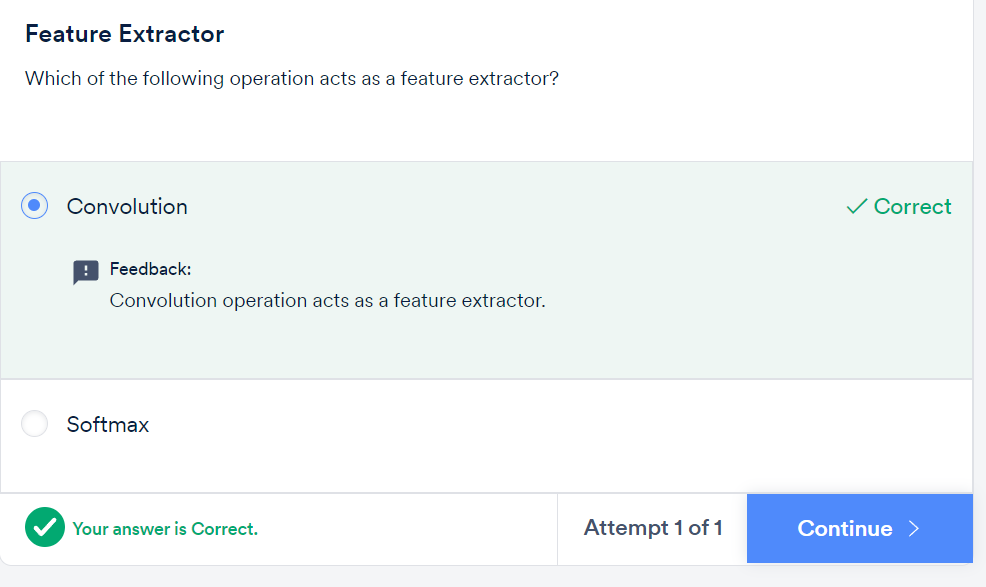
We have already discussed most of the key ideas of the CNN architecture through this paper. Summarising the main points below:

* Each unit, or neuron, is dedicated to its own **receptive field.** Thus, every unit is meant to ignore everything other than what is found in its own receptive field.
* The **receptive field**of each neuron is**almost identical** in shape and size.
* The subsequent layers compute the **statistical aggregate**of the previous layers of units. This is analogous to the 'pooling layer' in a typical CNN.
* Inference or the perception of the image happens at various **levels of abstraction**. The first layer pulls out raw features, subsequent layers pull out higher-level features based on the previous features and so on. Finally, the network gets an overall perception of an image in the last layer.

To summarise, there are three main concepts you will study in CNNs:

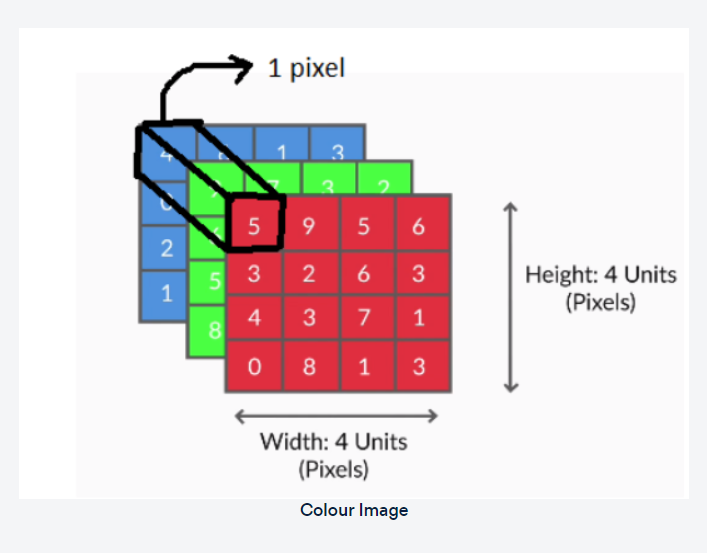
* Convolution, and why it 'shrinks' the size of the input image
* Pooling layers
* Feature maps

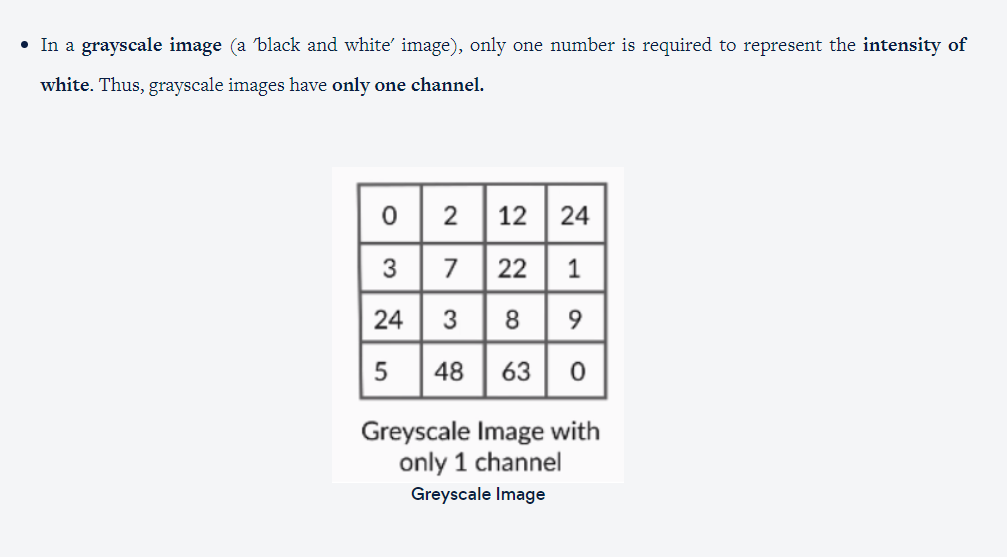


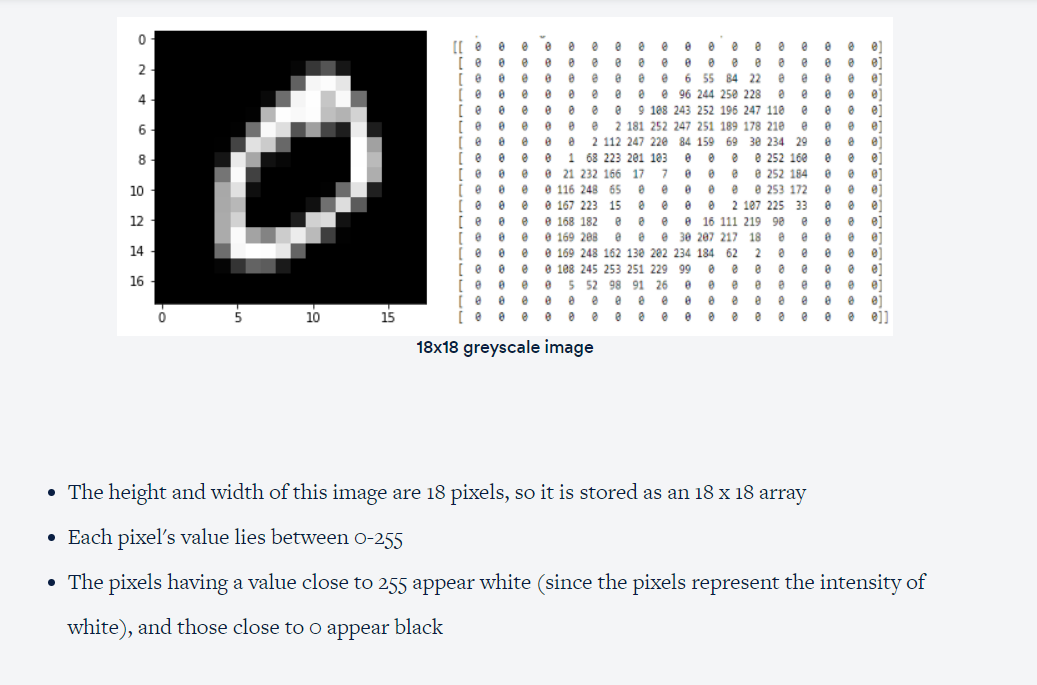


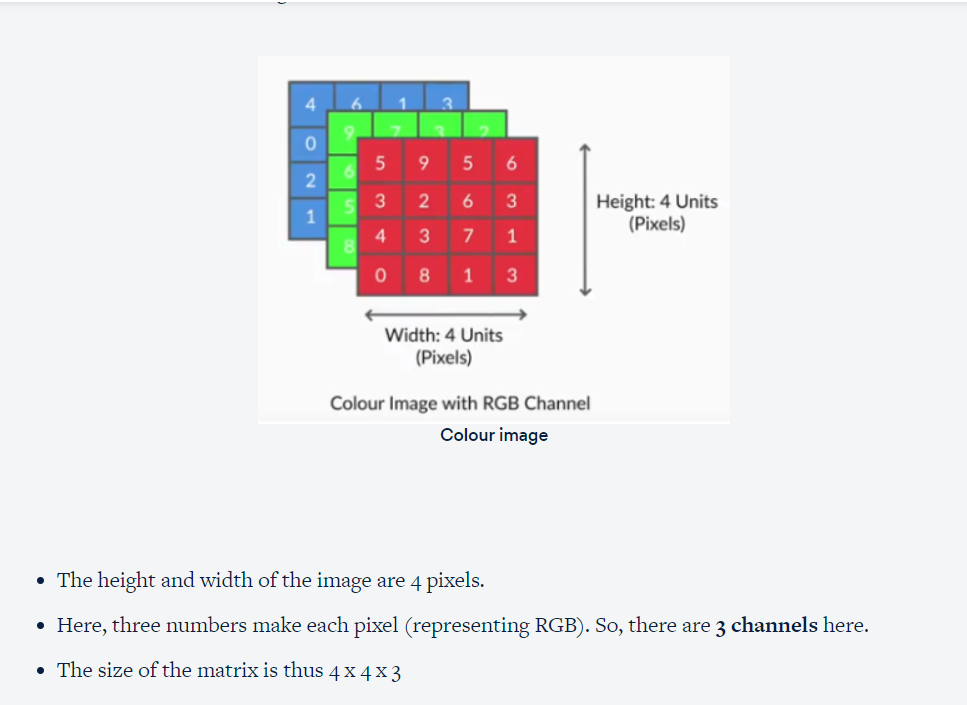


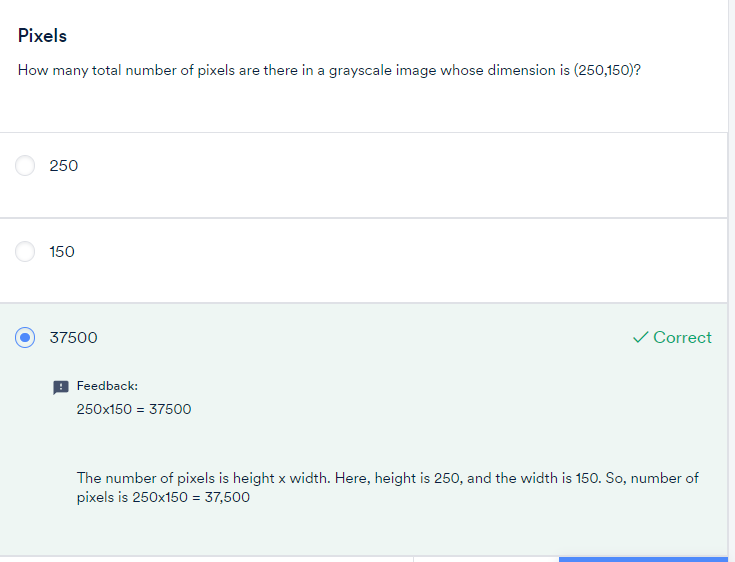
* Images are made up of **pixels**.
* A number between 0-255 represents the **colour intensity** of each pixel.
* Each pixel in a **colour image** is an array representing the intensities of red, blue and green. The red, blue and green layers are called **channels**.

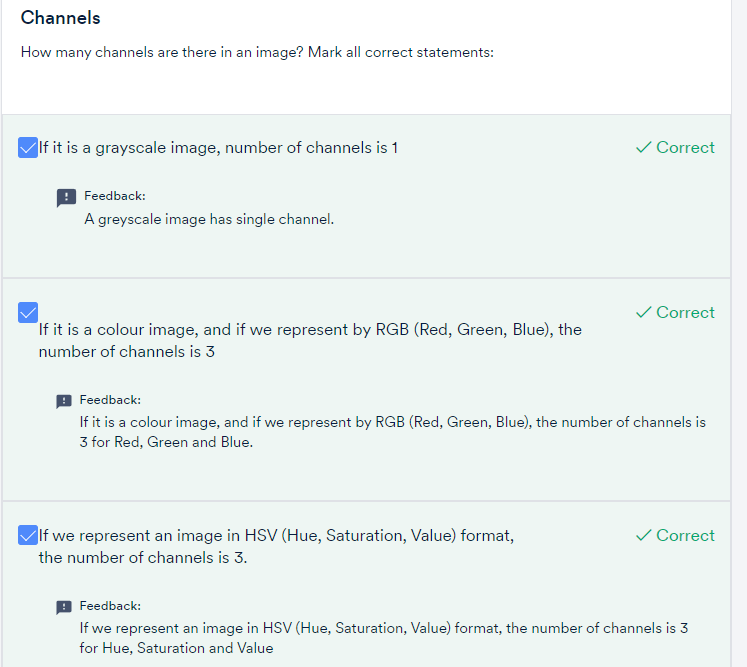


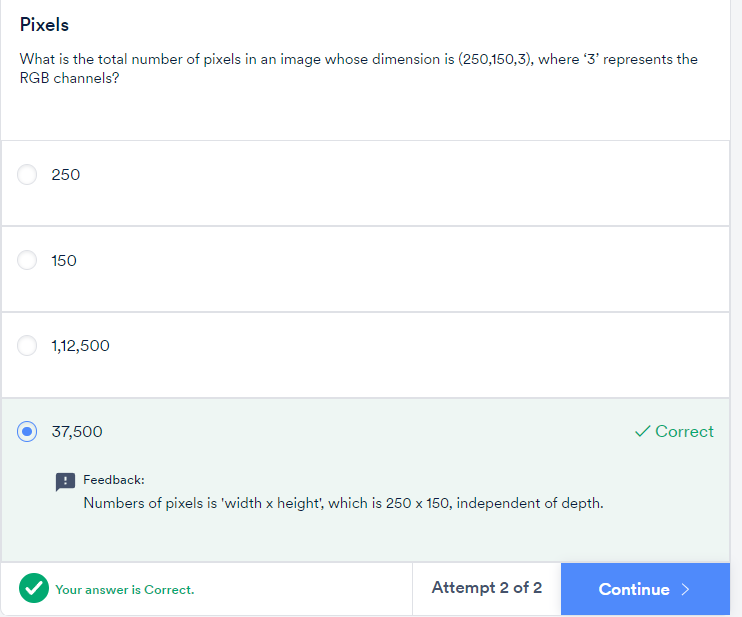


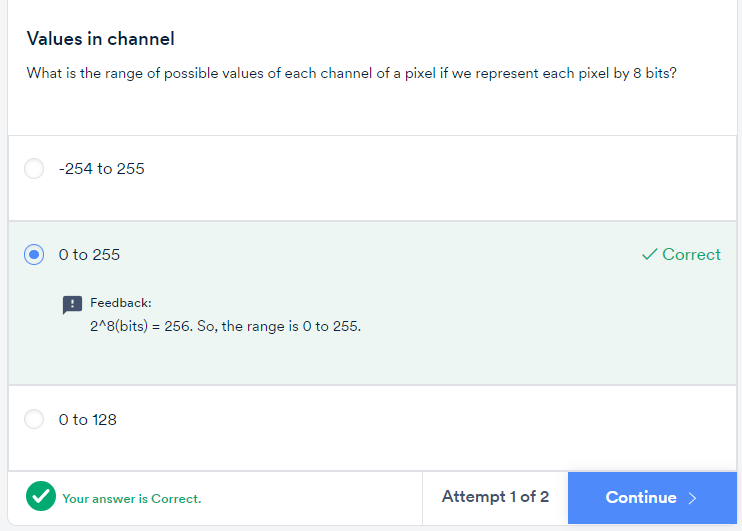


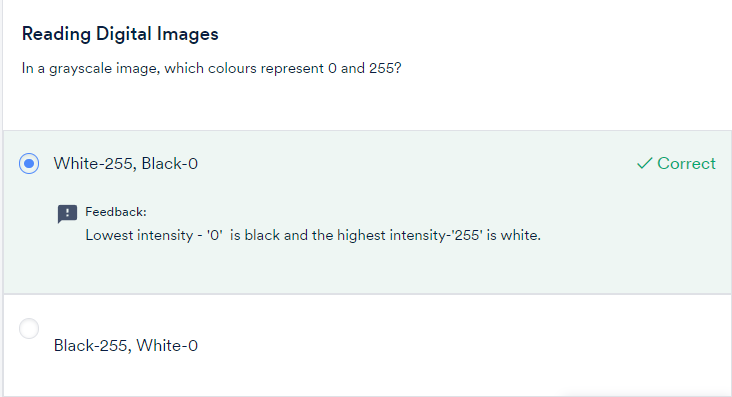


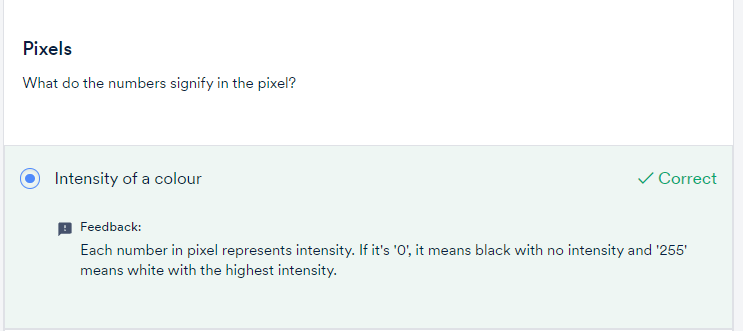












4732891

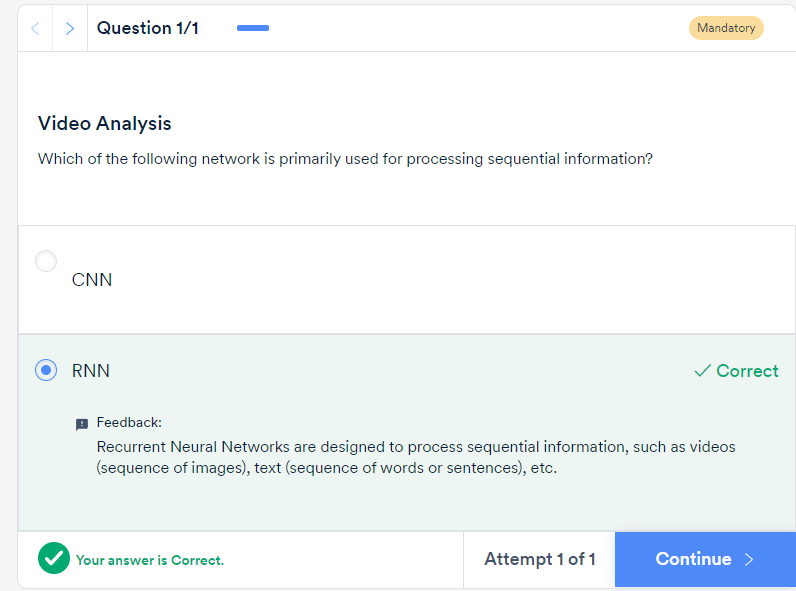
Let's summarise the process of video analysis using a CNN + RNN (Recurrent Neural Network) stack. At this point, you only need to understand that RNNs are good at processing sequential information such as videos (a sequence of images), text (a sequence of words or sentences), etc. You will study RNN in the next module.

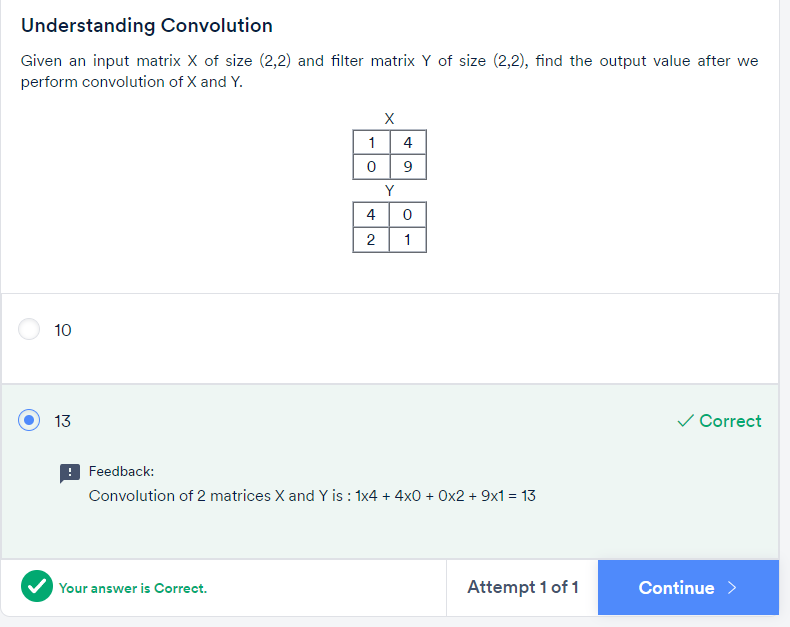
For a **video classification** task, here's what we can do. Suppose the videos are of length 1 minute each. If we extract frames from each video at the rate of **2 frames per second** (FPS), we will have 120 frames (or images) per video. Push each of these images into a convolutional net (such as VGGNet) and **extract a feature vector** (of size 4096, say) for each image. Thus, we have 120 feature vectors representing each video.

These 120 feature vectors, representing a video as a sequence of images, can now be fed sequentially into an RNN which classifies the videos into one of the categories.

The main point here is that a**CNN acts as a feature extractor** for images,and thus, can be used in a variety of ways to process images**.**

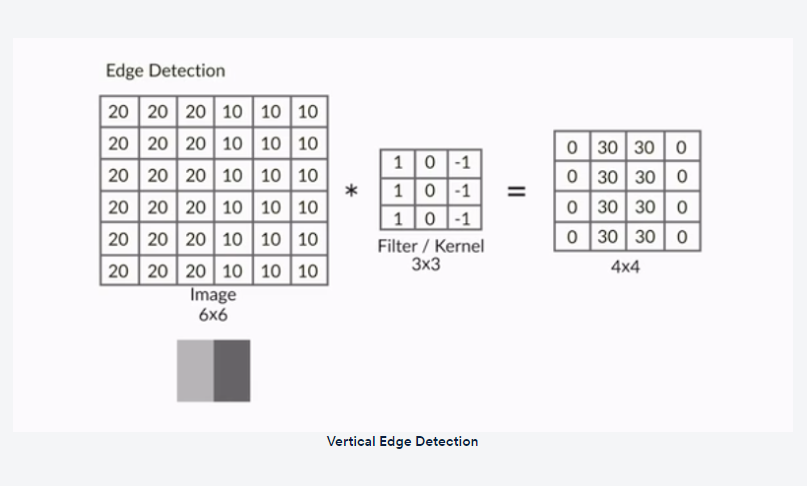
In the next few segments, you will study the main elements of CNNs in detail - convolutions, pooling, feature maps etc.





This was an example of how the convolution operation (using an appropriate filter) detects certain features in images, such as horizontal or vertical edges.

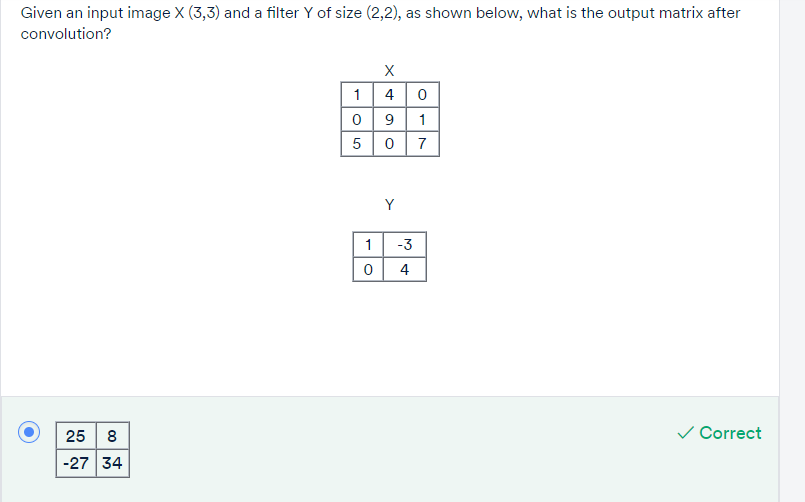
In the convolution output using the first filter, only the middle two columns are nonzero while the two extreme columns (1 and 4) are zero. This is an example of **vertical edge detection**.

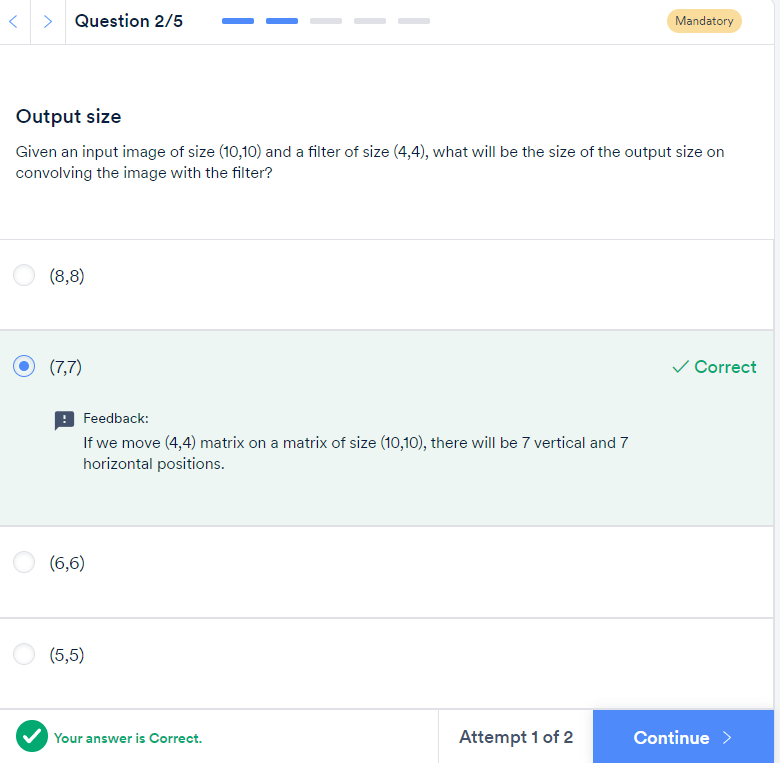


Note that each column of the 4 x 4 output matrix **looks at exactly three columns** of the input image. The values in the four columns represent the amount of change (or gradient) in the intensity of the corresponding columns in the input image along the horizontal direction.

For example the output is 0 (20 - 20 or 10 - 10) in the columns 1 and 4, denoting that there is no change in intensity in the first three and the last three columns of the input image respectively.

On the other hand, the output is 30 (20 - (-10)) in the columns 2 and 3, indicating that there is a gradient in the intensity of the corresponding columns of the input image.

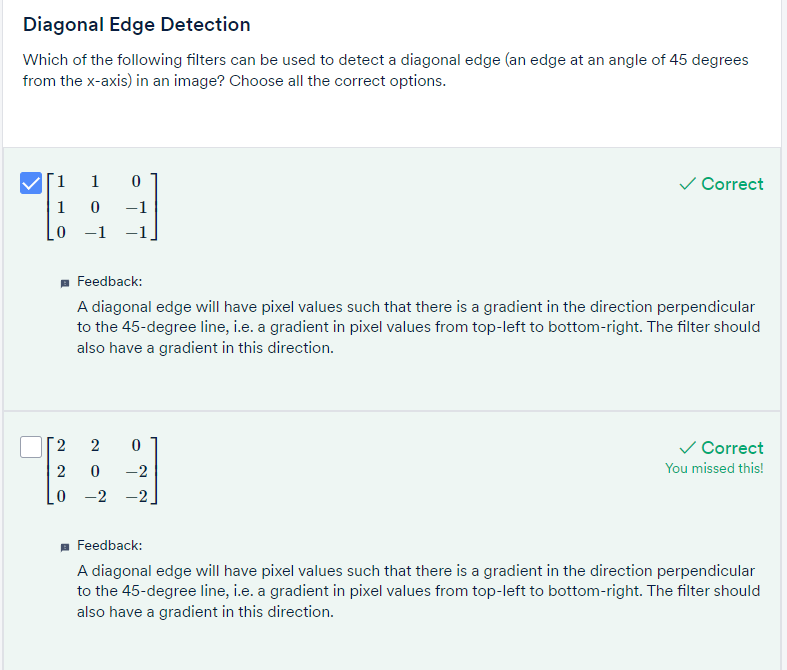




10-4+1







The following are the two most common ways to do padding:

* Populating the dummy row/columns with the pixel values at the edges
* Populating the dummy row/columns with zeros (zero-padding)

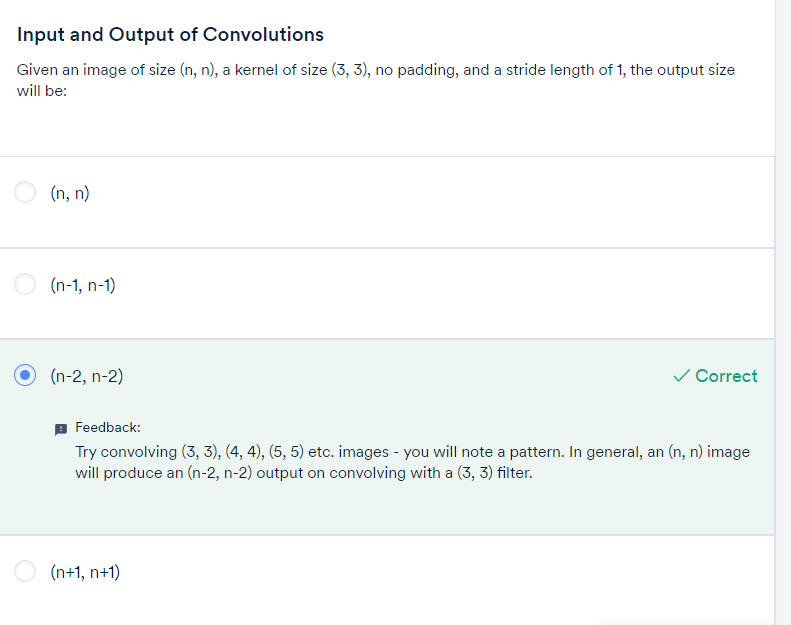
**Notation:**

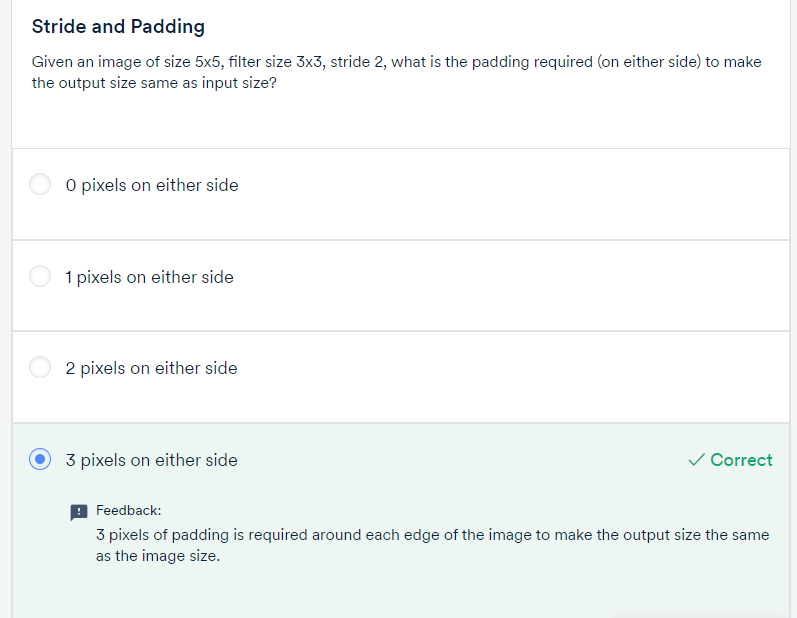
**Padding of 'x' means that 'x units' of rows/columns are added all around the image.**

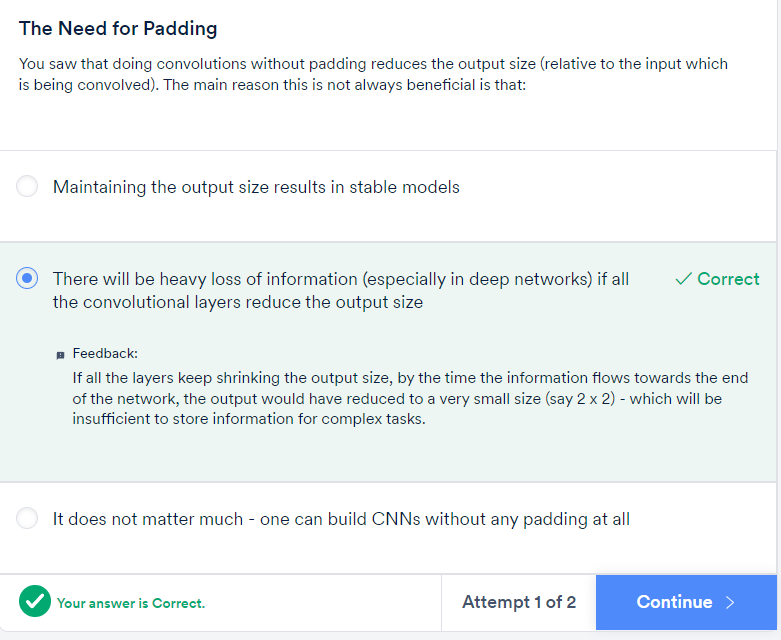
An alternate (less commonly used) way to do convolution is to shrink the filter size as you hit the edges.

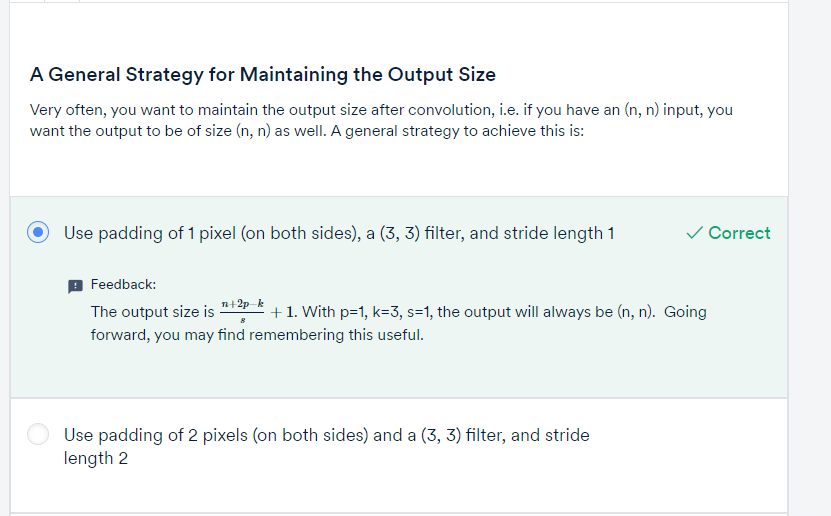
You may have noticed that when you convolve an image **without padding** (using any filter size), the **output size is smaller**than the image (i.e. the output '**shrinks'**). For example. when you convolve a (6, 6) image with a (3, 3) filter and stride of 1, you get an output of (4, 4).

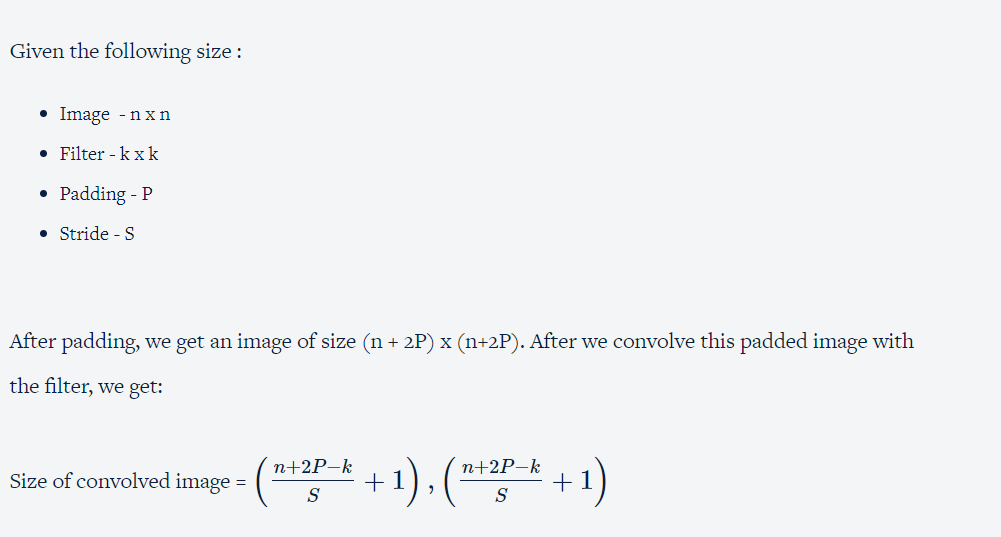
You saw that doing convolutions without padding **reduces the output size**. It is important to note that**only the width and height decrease (not the depth)**when you convolve without padding. The depth of the output depends on the number of filters used



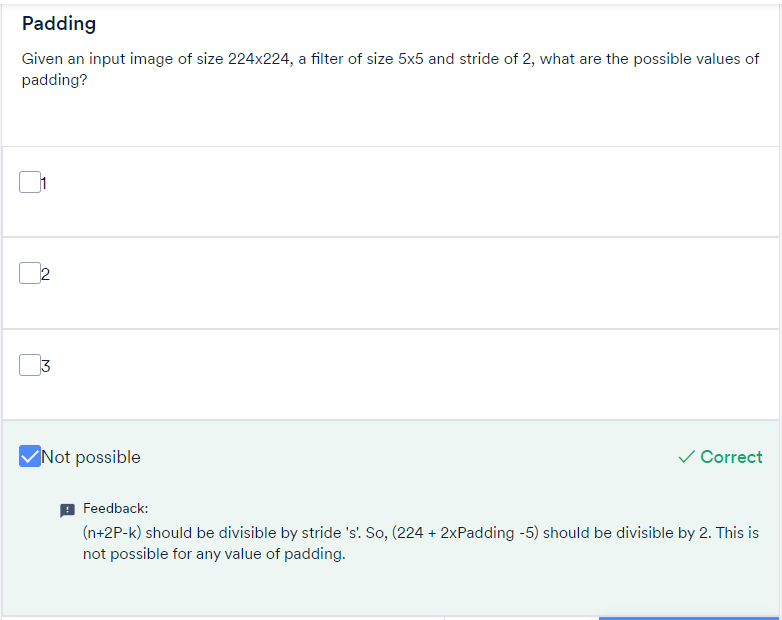


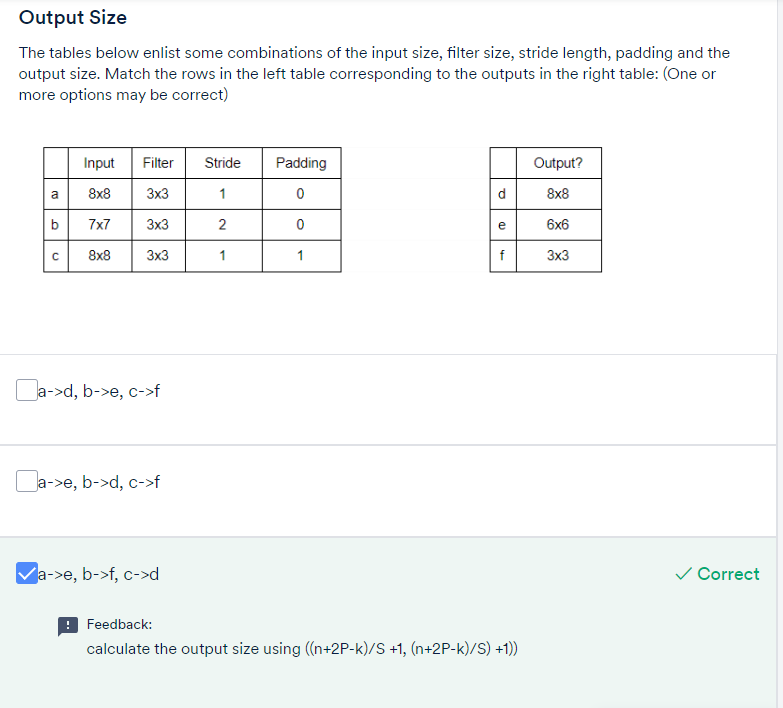








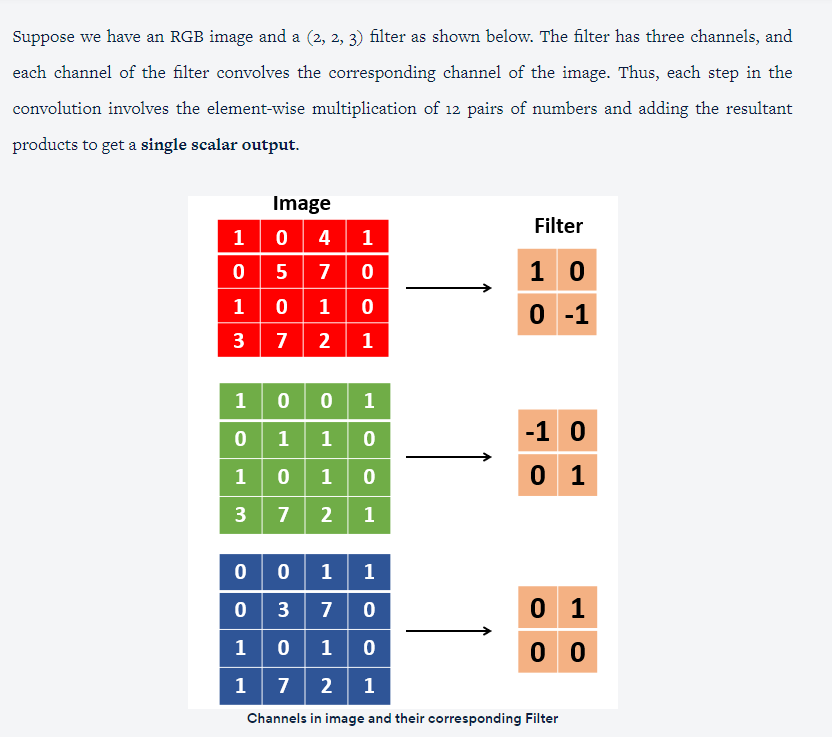


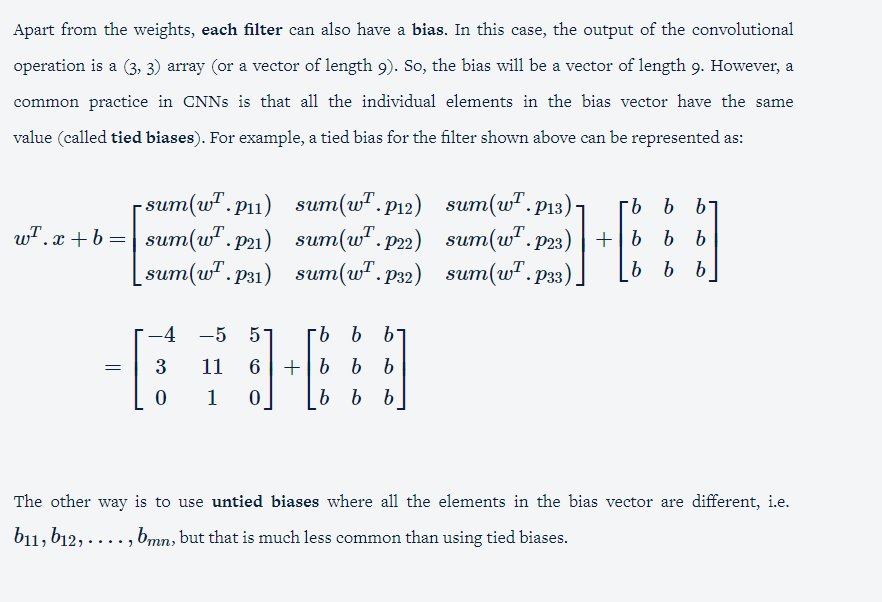


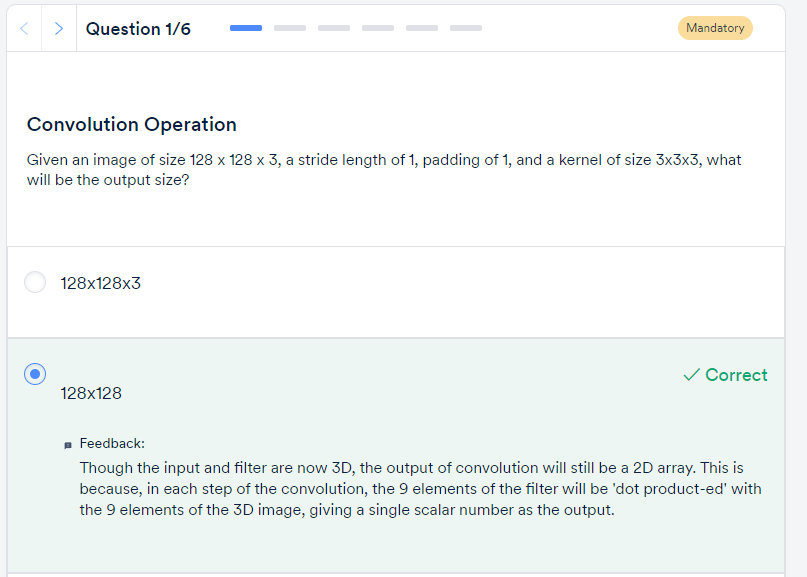


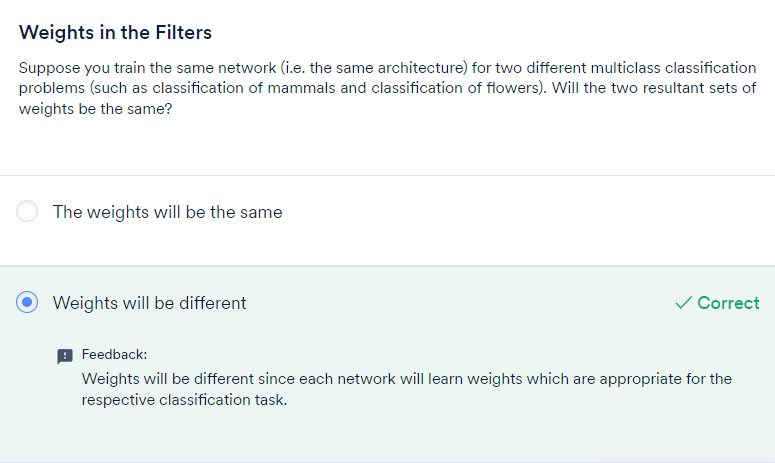
To summarise, you learnt the following:

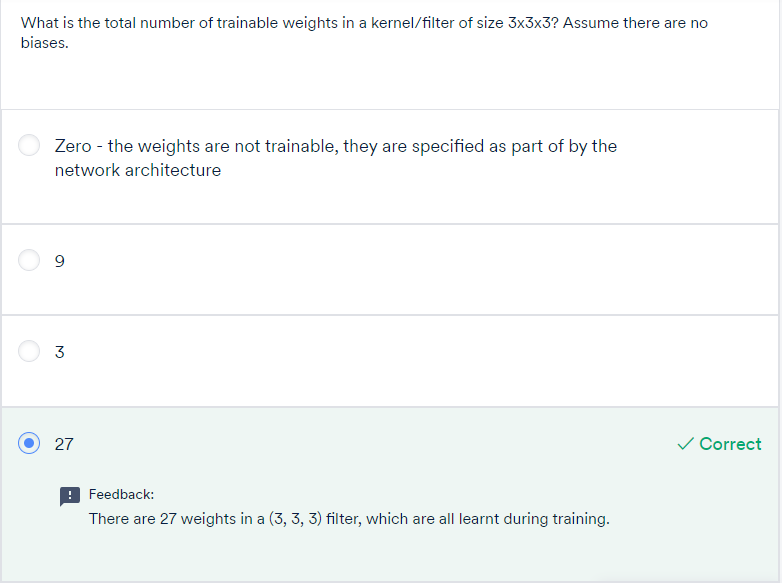
* We use **3D filters** to perform convolution on 3D images. For example: if we have an image of size (224, 224, 3), we can use filters of sizes (3, 3, 3), (5, 5, 3), (7, 7, 3) etc. (with appropriate padding etc.). We can use a filter of any size as long as the number of channels in the filter is the same as that in the input image.
* The**filters are learnt** during training (i.e. duringbackpropagation). Hence, the individual values of the filters are often called **the weights of a CNN.**

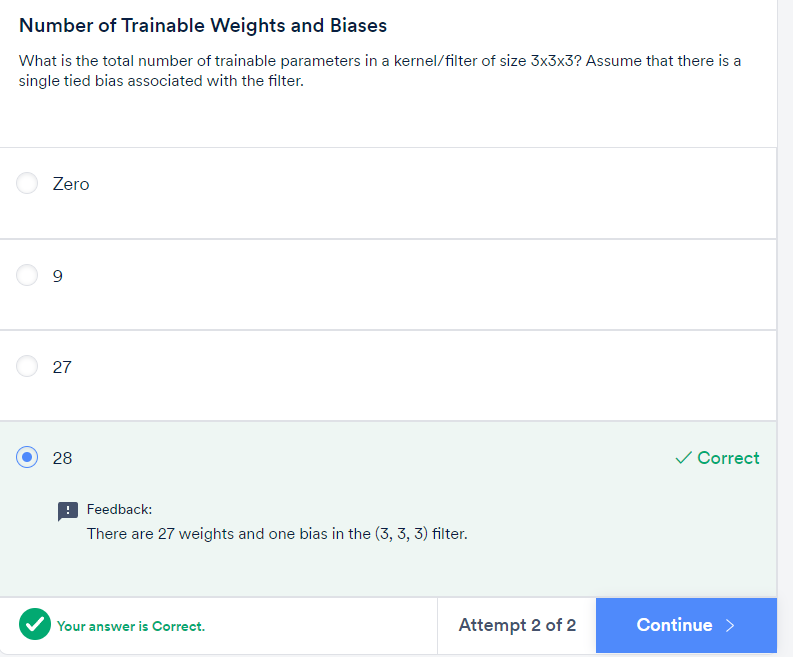


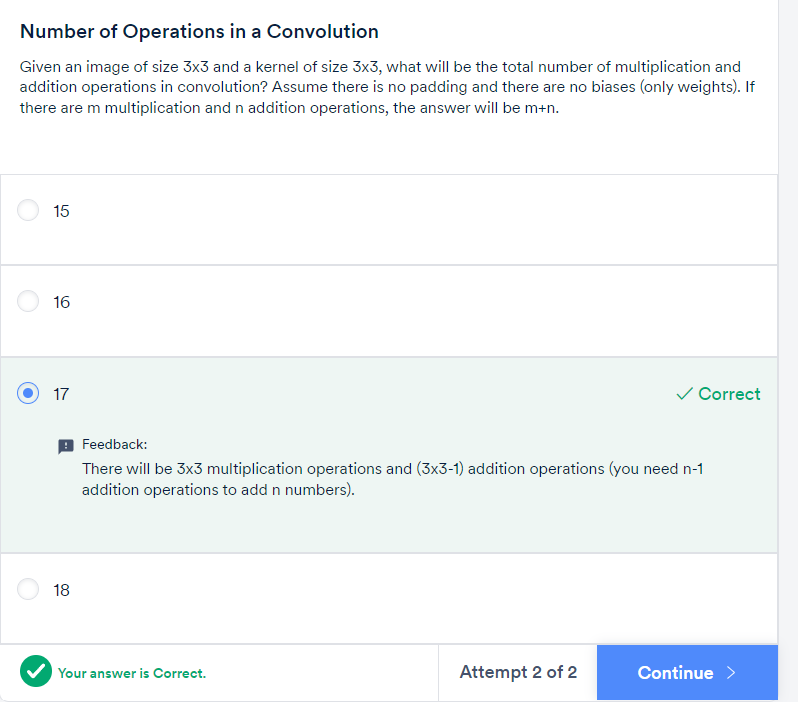












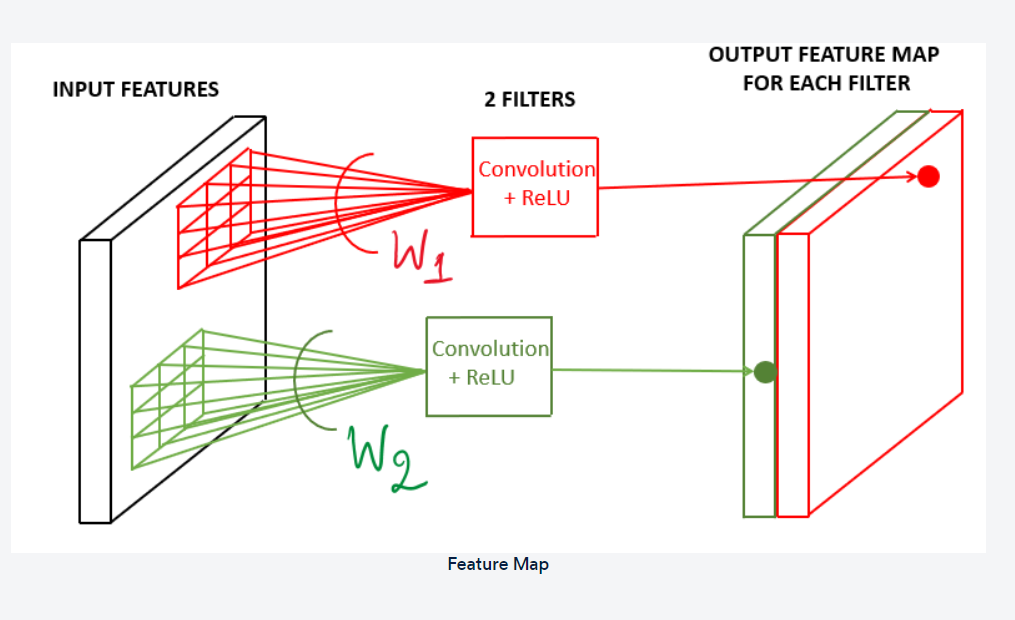


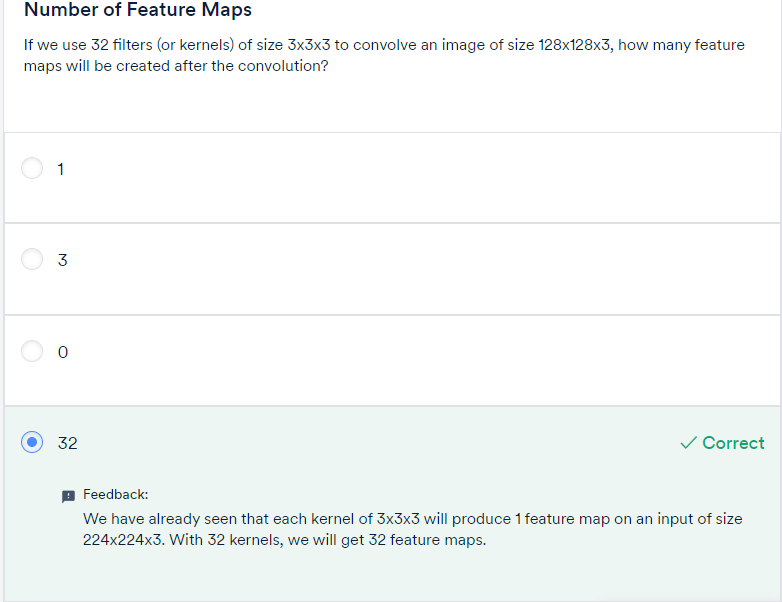
FEATURE MAP:

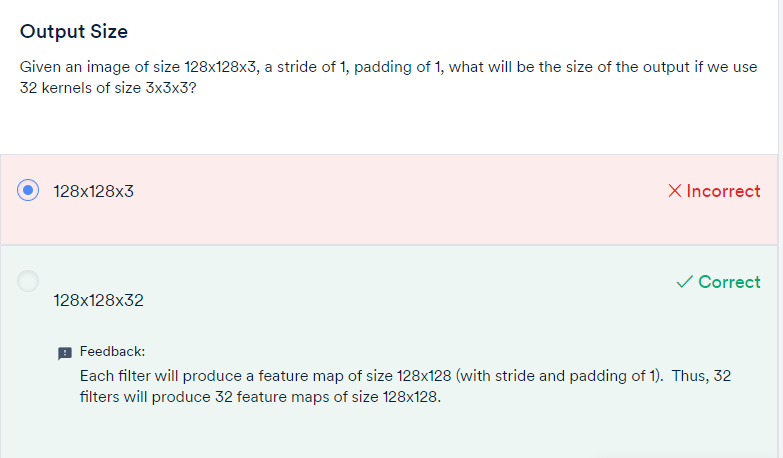
Let's summarise the important concepts and terms discussed above:

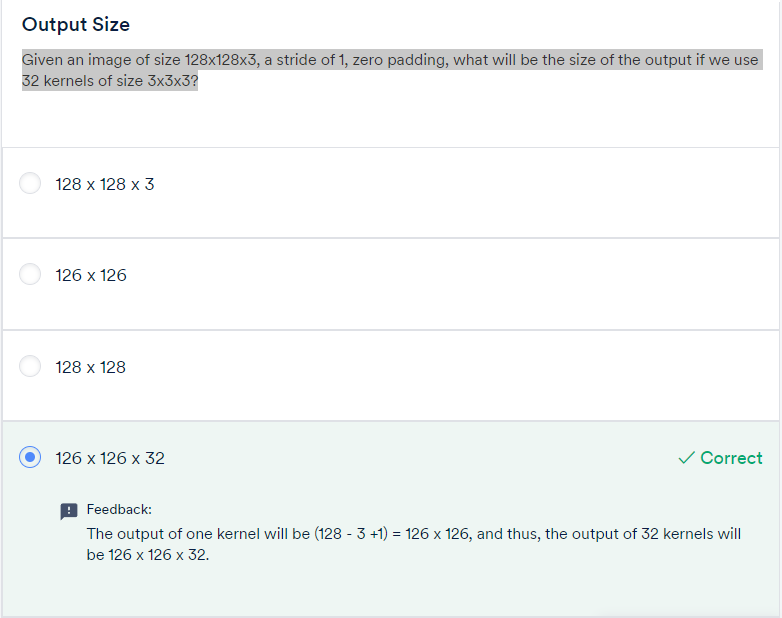
* A **neuron**is basically a filter whose weights are learnt during training. For example, a (3, 3, 3) filter (or neuron) has 27 weights. Each neuron looks at a particular region in the input (i.e. its 'receptive field').
* A **feature map** is a collection of multiple **neurons** each of whichlooks at **different regions** of the input with the **same weights**. All neurons in a feature map extract the same feature (but from different regions of the input). It is called a 'feature map' because it is a mapping of where a certain feature is found in the image.

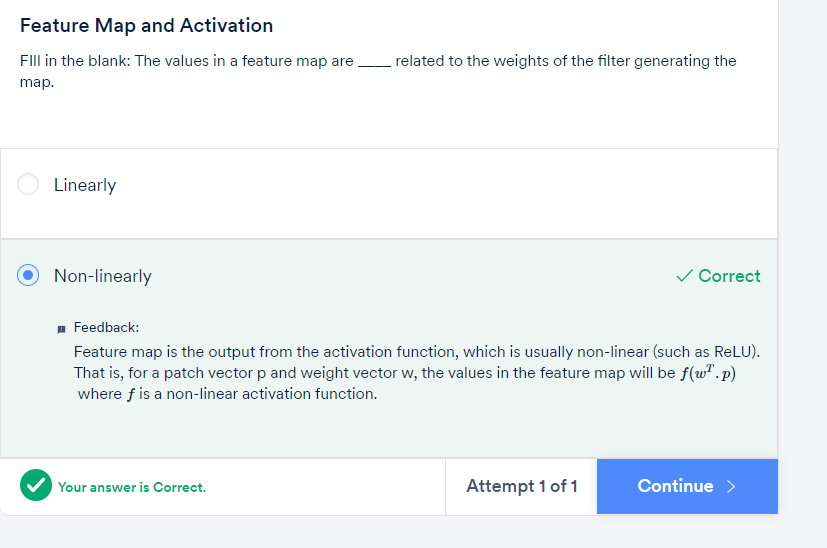
The figure below shows two neurons in a feature map (the right slab) along with the regions in the input from which the neurons extract features.









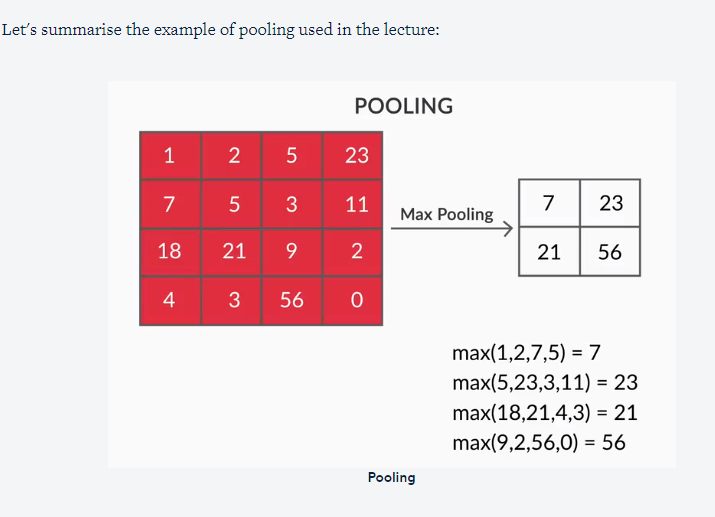


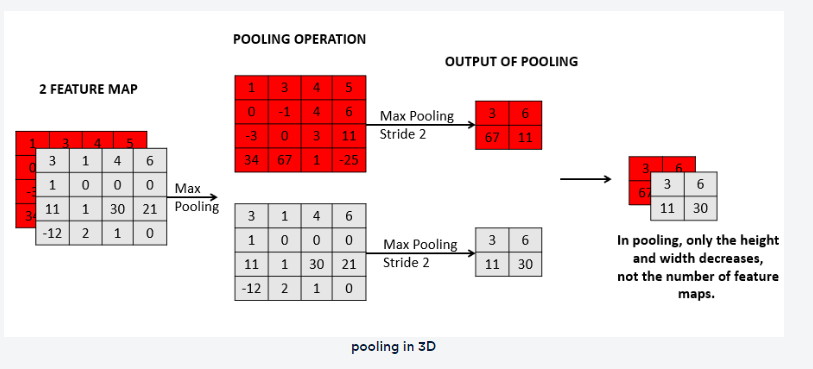
POOLING:

Pooling tries to figure out whether a particular region in the image has the feature we are interested in or not. It essentially looks at larger regions (having multiple patches) of the image and captures an **aggregate statistic** (max, average etc.) of each region. In other words, it makes the network **invariant to local transformations.**

The two most popular aggregate functions used in pooling are 'max' and 'average'. The intuition behind these are as follows:

* **Max pooling**: If any one of the patches says something strongly about the presence of a certain feature, then the pooling layer counts that feature as 'detected'.
* **Average pooling**: If one patch says something very firmly but the other ones disagree,  the pooling layer takes the average to find out.

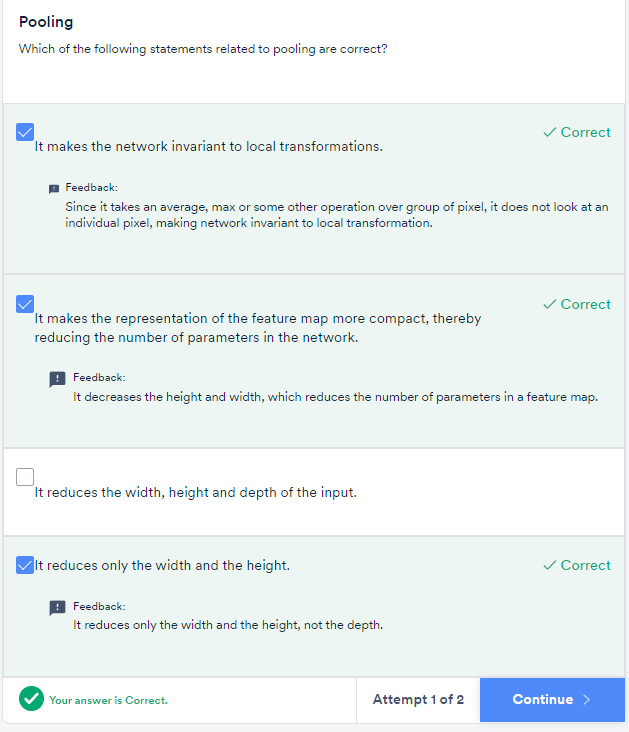




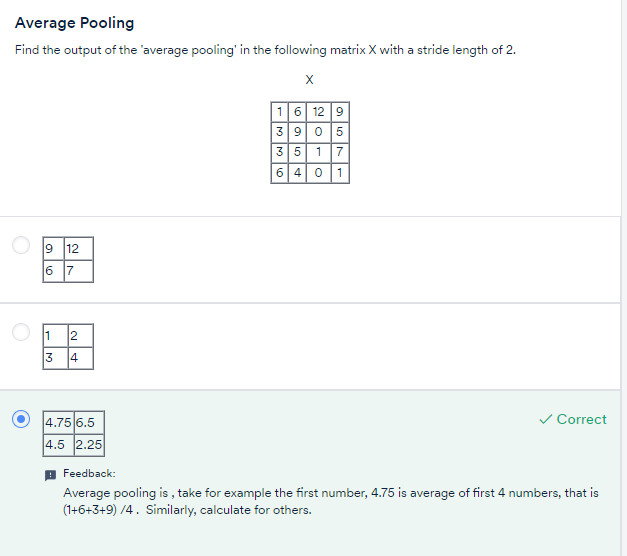
You can observe that pooling operates on each feature map independently. It reduces the size (width and height) of each feature map, but the number of feature maps remains constant.

Pooling has the advantage of making the representationmore compact by**reducing the spatial size** (height and width) of the feature maps, thereby reducing the number of parameters to be learnt. On the other hand, it also **loses a lot of information**, which is often considered a potential disadvantage. Having said that, pooling has empirically proven to improve the performance of most deep CNNs.

Can we design a network without pooling? **Capsule networks** were designed to address some of these potential drawbacks of the conventional CNN architecture. The paper on Capsule networks  is provided below.







To summarise, a **typical CNN layer (or unit)** involves the following **two components** in sequence:

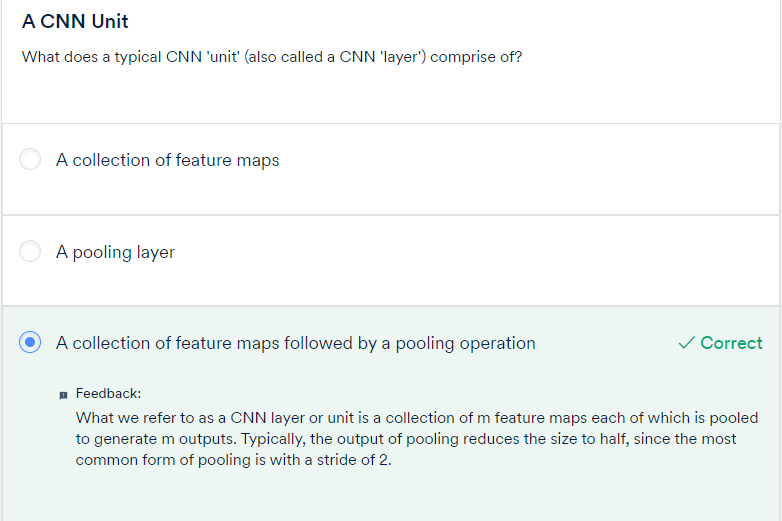
1. We start with an original image and do convolutions using multiple filters to get multiple feature maps.
2. A pooling layer takes the statistical aggregate of the feature maps

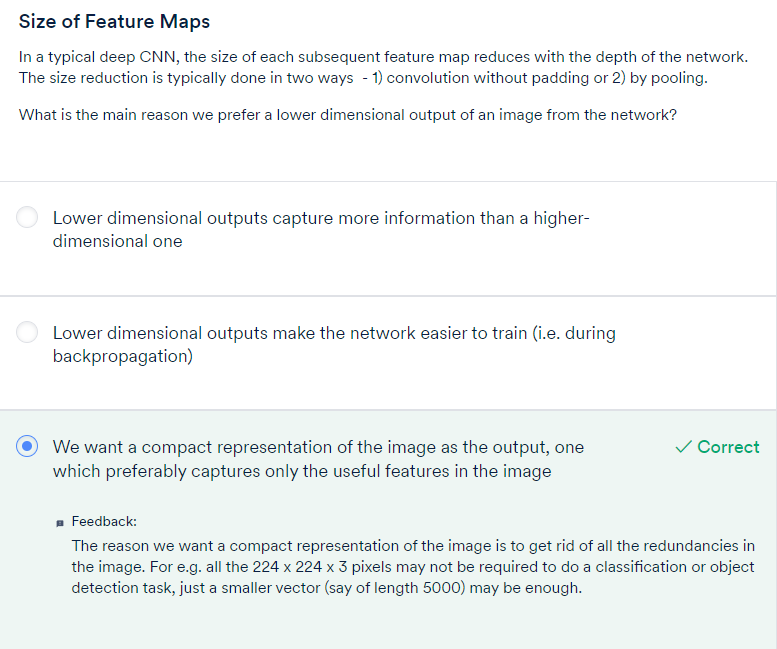
Typically, deep CNNs have **multiple such CNN units** (i.e. feature map-pooling pairs) arranged sequentially. The following lecture will discuss this in detail.

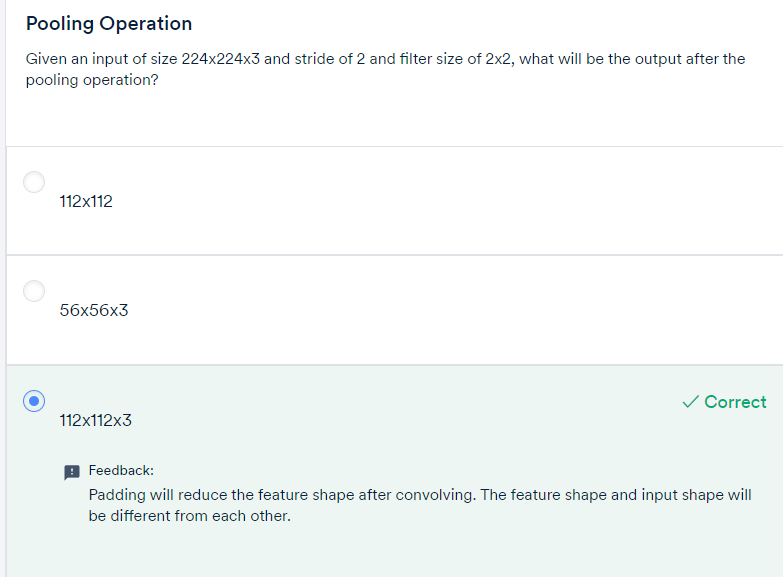
To summarise, a typical CNN has the following sequence of CNN layers:

1. We have an input image which is convolved using multiple filters to create **multiple feature maps**
2. Each feature map, of size (c, c), is **pooled** to generate a (c/2, c/2) output (for a standard 2 x 2 pooling).
3. The above pattern is called **a CNN layer** **or** **unit**. Multiple such CNN layers are stacked on top of one another to create deep CNN networks.

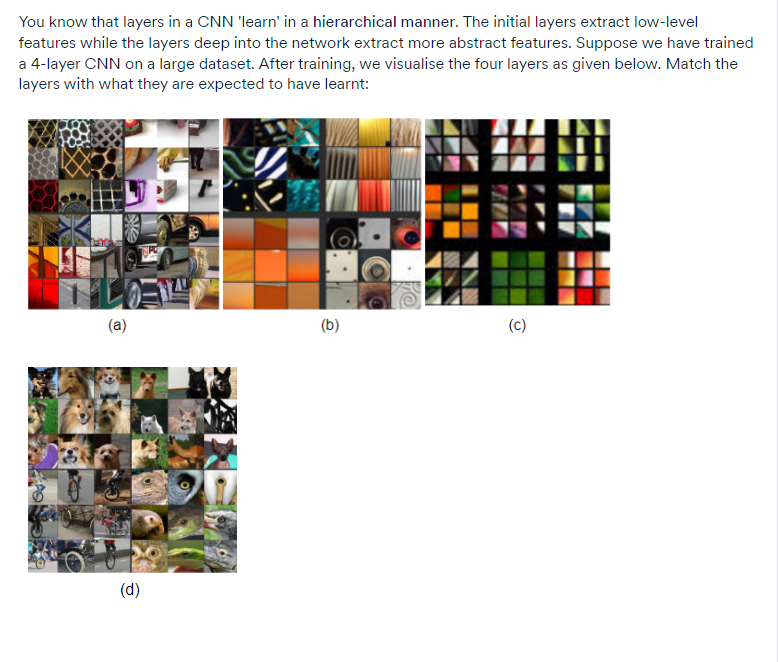
Note that pooling reduces only the height and the width of a feature map, not the depth (i.e. the number of channels). For example, if you have m feature maps each of size (c, c), the pooling operation will produce m outputs each of size (c/2, c/2).

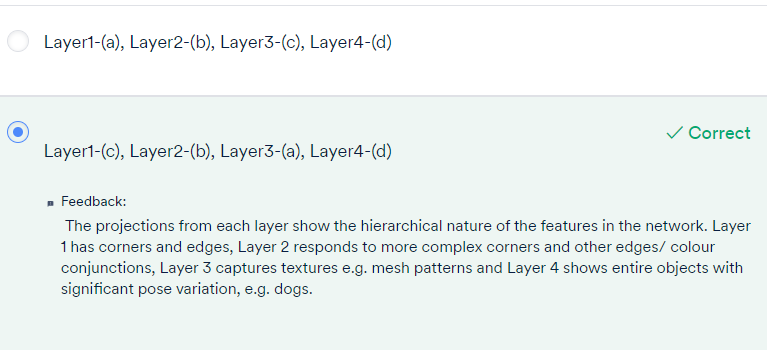


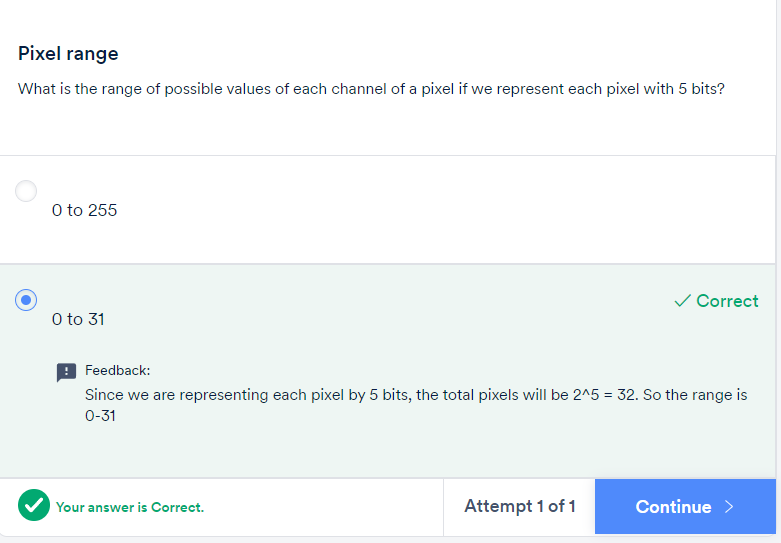


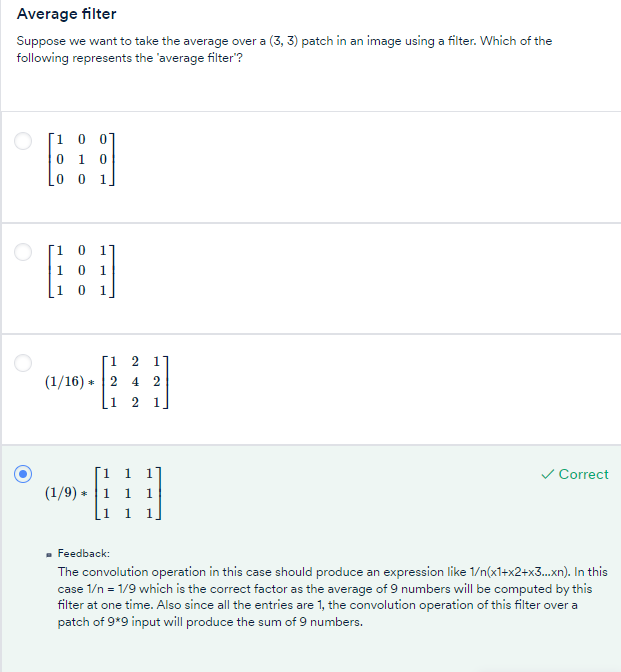


GRADED Questions:

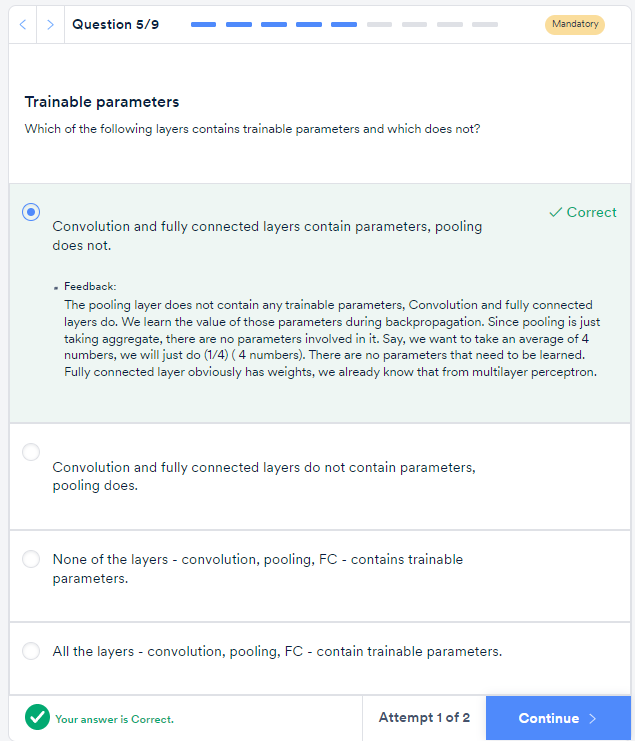


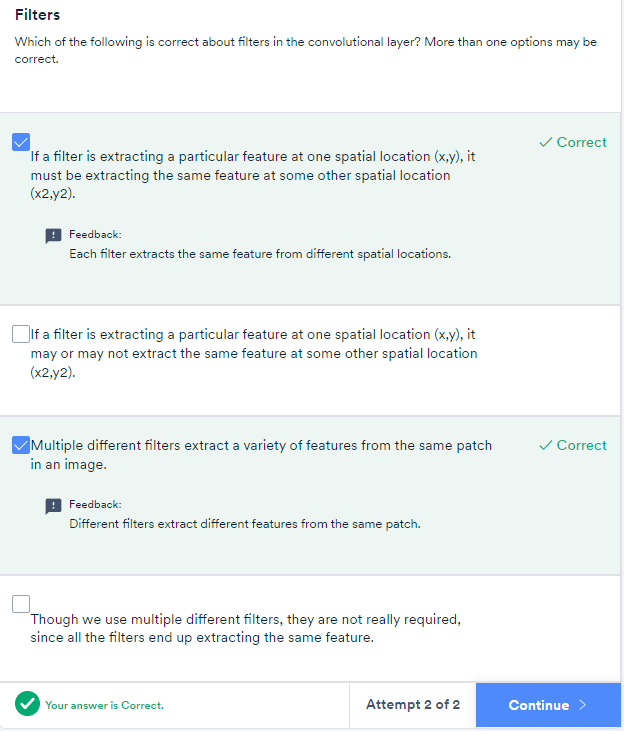


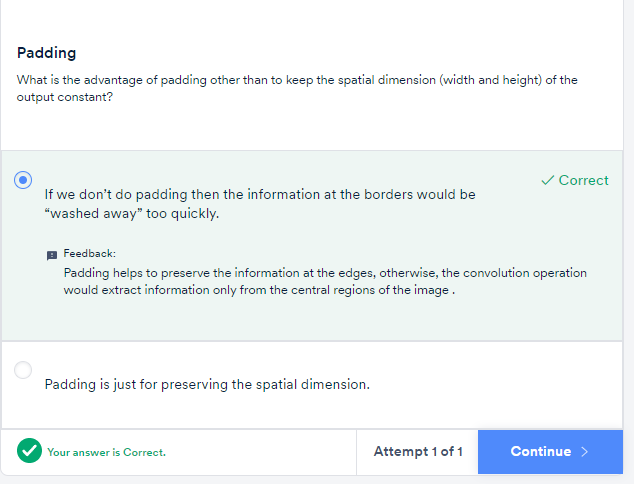


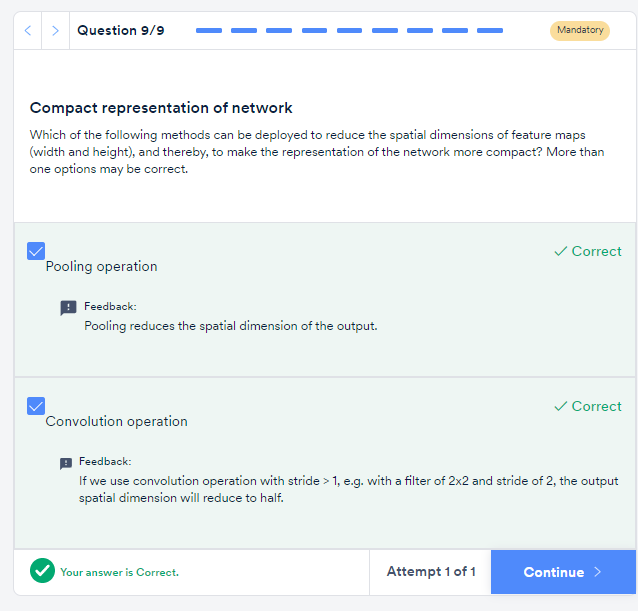


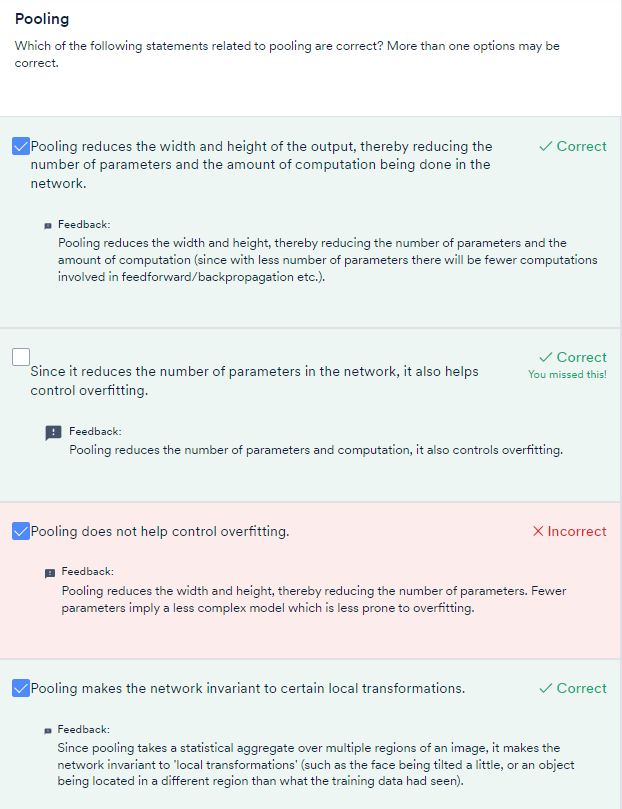


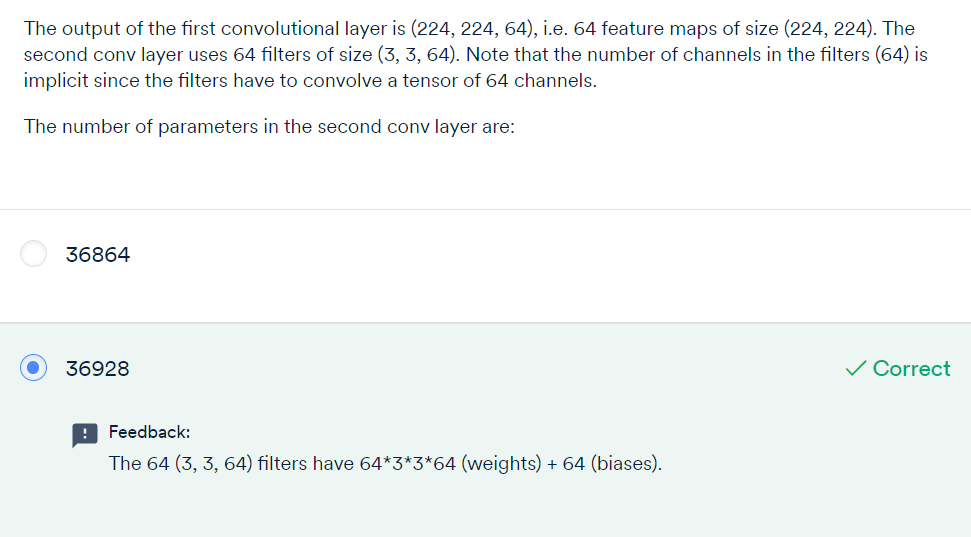


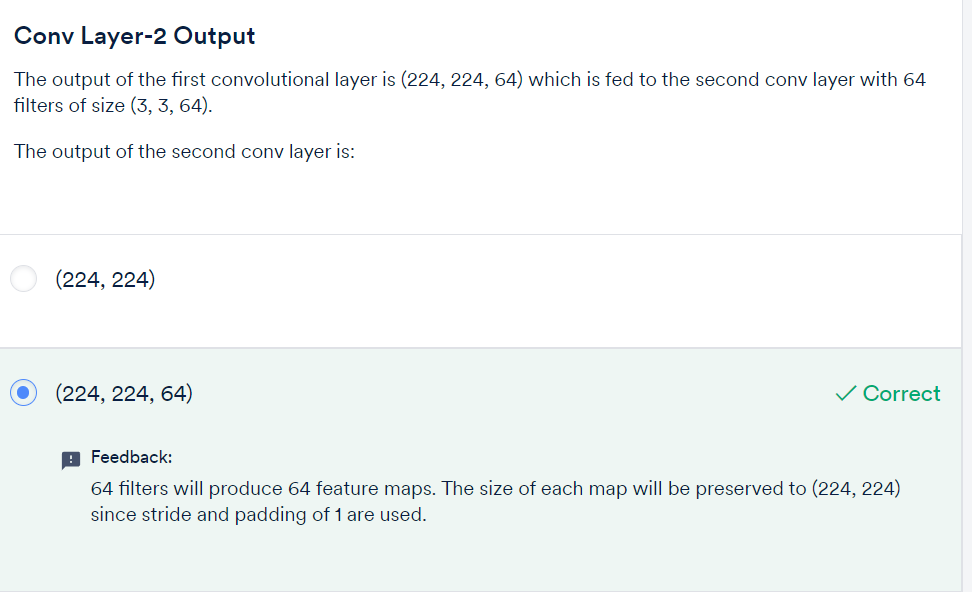


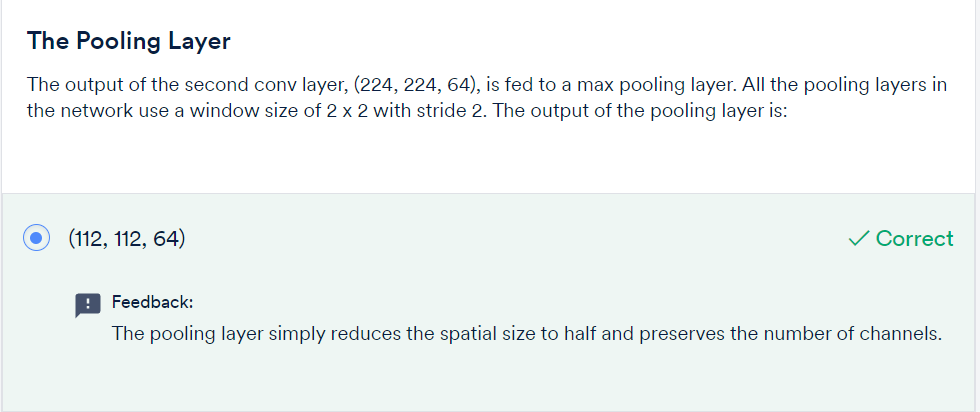


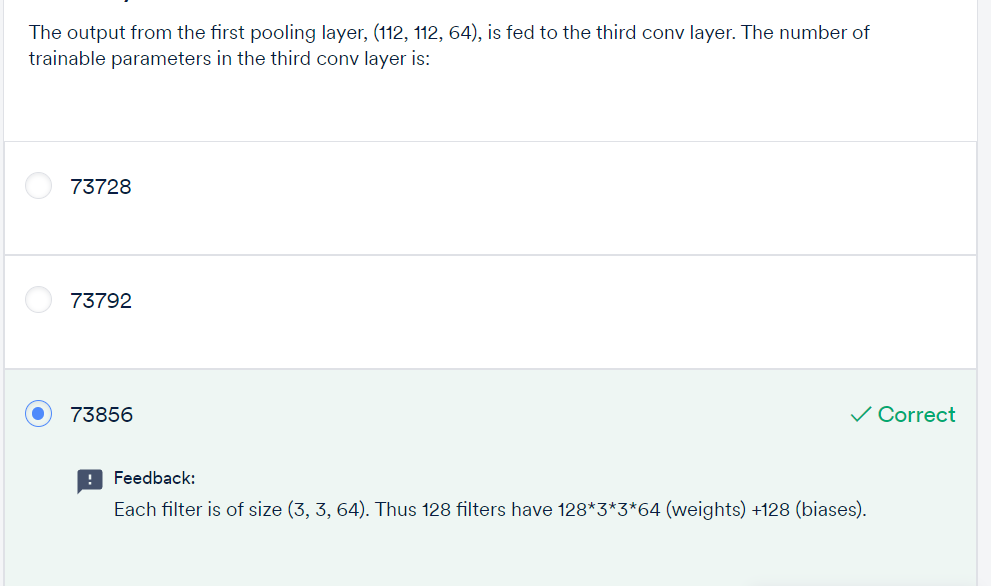


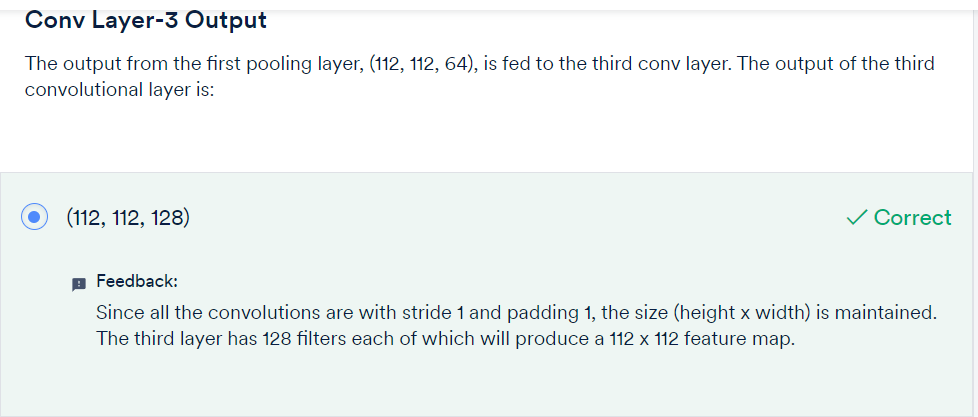


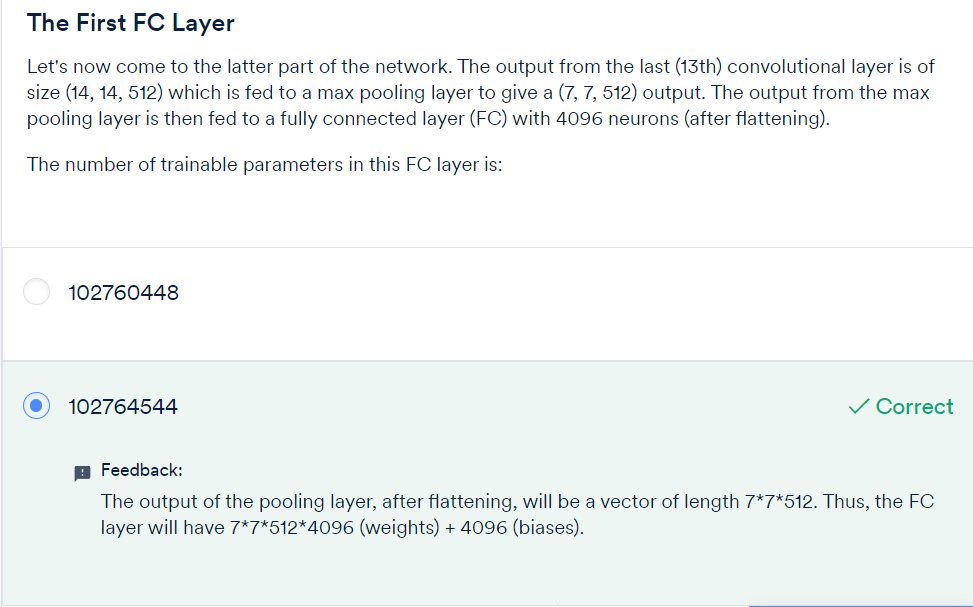


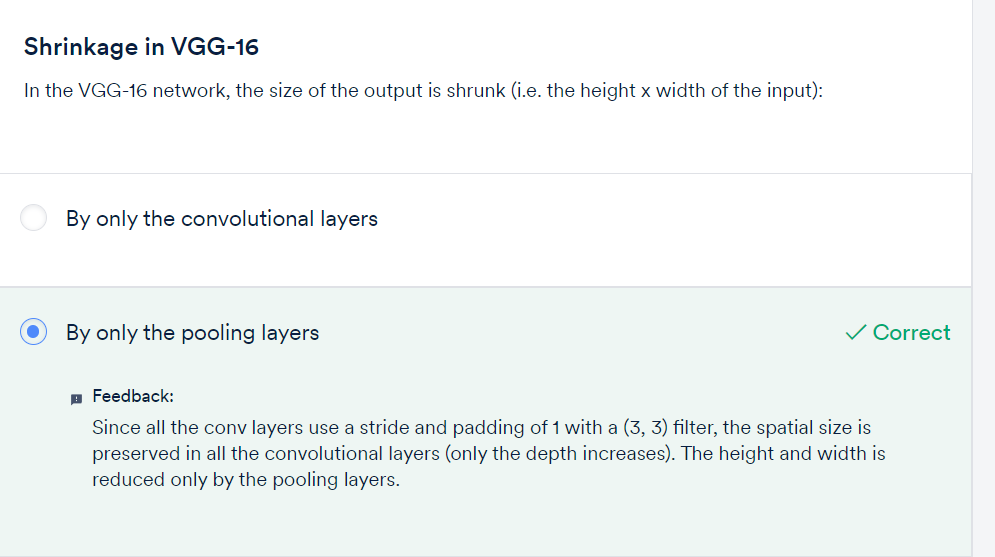




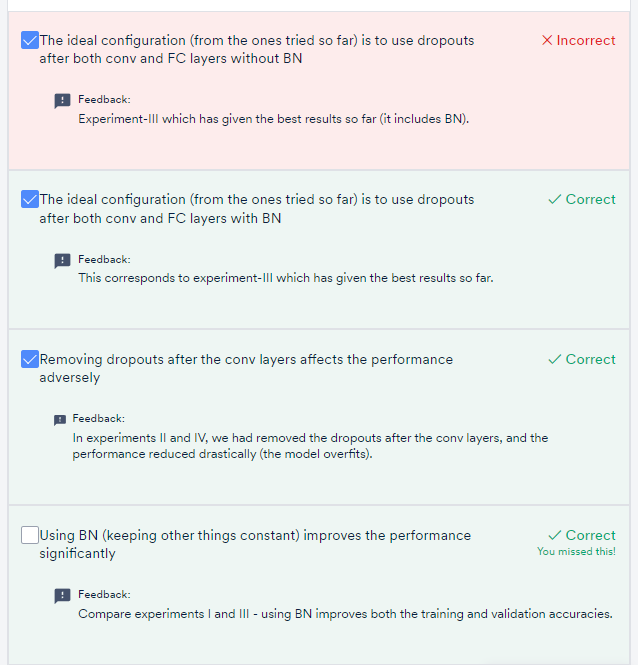








* **Experiment - I** (Use dropouts after conv and FC layers, no BN):
  + Training accuracy =  84%, validation accuracy  =  79%
* **Experiment - II** (Remove dropouts from conv layers, retain dropouts in FC, use BN):
  + Training accuracy =  98%, validation accuracy  =  79%
* **Experiment - III** (Use dropouts after conv and FC layers, use BN):
  + Training accuracy =  89%, validation accuracy  =  82%
* **Experiment - IV** (Remove dropouts from conv layers, use L2 + dropouts in FC, use BN):
  + Training accuracy = 94%, validation accuracy = 76%.



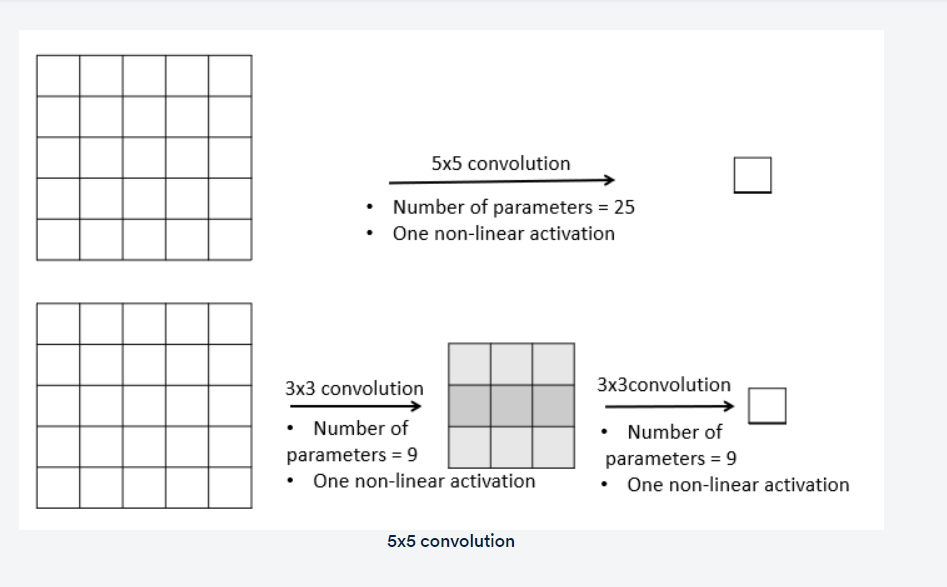
**CNN Architectures and Transfer Learning:**

* 1. **VGGNet**
  2. **GoogleNet**
  3. **ResNet**
  4. **AlexNet**
* The **depth** of the state-of-the-art neural networks has been**steadily increasing** (from AlexNet with 8 layers to ResNet with 152 layers).
* The developments in neural net architectures were made possible by **significant advancements in infrastructure**. For example, many of these networks were trained on multi GPUs in a distributed manner.
* Since these networks have been trained on millions of images, they are good at **extracting generic features** from a large variety of images. Thus, they are now commonly being used as commodities by deep learning practitioners around the world.

**Comprehension - Effective Receptive Field**

The key idea in moving from AlexNet to VGGNet was to **increase the depth** of the network by using **smaller filters.**Let's understand what happens when we use a smaller filter of size (3, 3) instead of larger ones such as (5, 5) or (7, 7).

Consider the example below. Say we have a 5 x 5 image, and in two different convolution experiments, we use two different filters of size (5, 5) and (3, 3) respectively.

****

In the first convolution, the (5, 5) filter produces a feature map with a single element (note that the convolution is followed by a non-linear function as well). This filter has 25 parameters.

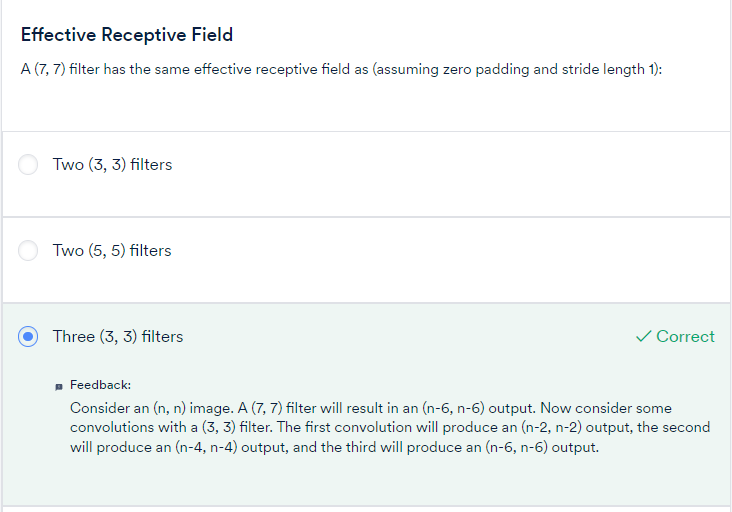
In the second case with the (3, 3) filter, two successive convolutions (with stride=1, no padding) produce a feature map with one element.

We say that the stack of two (3, 3) filters has the same **effective receptive field**as that of one (5, 5) filter.  This is because both these convolutions produce the same output (of size 1 x1 here) whose receptive field is the same 5 x 5 image.

Notice that with a smaller (3, 3) filter, we can make a deeper network with **more non-linearities**and**fewer parameters**. In the above case:

* The (5, 5) filter has 25 parameters and one non-linearity
* The (3, 3) filter has 18 (9+9) parameters and two non-linearities.

Since VGGNet had used smaller filters (all of 3 x 3) compared to AlexNet (which had used 11 x 11 and 5 x 5 filters), it was able to use a higher number of non-linear activations with a reduced number of parameters.

****

**GoogleNet:**

To summarise, some important features of the GoogleNet architecture are as follows:

* Inception modules stacked on top of each other, total 22 layers
* Use of 1 x 1 convolutions in the modules
* Parallel convolutions by multiple filters (1x1, 3x3, 5x5)
* Pooling operation of size (3x3)
* No FC layer, except for the last softmax layer for classification
* Number of parameters reduced from 60 million (AlexNet) to 4 million

**ResNET:**

Thus, the **key motivator** for the ResNet architecture was the observation that, empirically, adding more layers was not improving the results monotonically.  This was counterintuitive because a network with n + 1 layers should be able to learn at least what a network with n layers could learn, plus something more.

Thus, the **skip connection mechanism** was the key feature of the ResNet which enabled the training of very deep networks. Some other key features of the ResNet are summarised below. You are also encouraged to read the detailed results in the ResNet paper provided at the bottom of this page:

* ILSVRC’15 classification winner (3.57% top 5 error)
* 152 layer model for ImageNet
* Has other variants also (with 35, 50, 101 layers)
* Every 'residual block' has two 3x3 convolution layers
* No FC layer, except one last 1000 FC softmax layer for classification
* Global average pooling layer after the last convolution
* Batch Normalization after every convolution layer
* SGD + momentum (0.9)
* No dropout used

Transfer Learning:

Thus, **transfer learning** is the practice of reusing the skills learnt from solving one problem to learn to solve a new, related problem

To summarise, some practical reasons to use transfer learning are as follows:

* Data abundance in one task and data crunch in another related task,
* Enough data available for training, but lack of computational resources.

Let’s revisit the example of **document summarisation**. If you want to do document summarisation in some other language, such as Hindi, you can take the following steps:

* Use word embeddings in English to train a document summarisation model (assuming a significant amount of data in English is available)
* Use word embeddings of another language such as Hindi (where you have a data crunch) to tune the English summarisation model

Thus, the initial layers of a network extract the basic features, the latter layers extract more abstract features, while the last few layers are simply discriminating between images.

In other words, the initial few layers are able to **extract generic representations of an image** and thus can be used for any general image-based task

# Practical Implementation of Transfer Learning

There are two main ways of using pre-trained nets for transfer learning:

* Freeze the (weights of) initial few layers and training only a few latter layers
* Retrain the entire network (all the weights) initialising from the learned weights

Thus, you have the following two ways of training a pre-trained network:

1. ‘**Freeze**’ the initial layers, i.e. use the same weights and biases that the network has learnt from some other task, remove the few last layers of the pre-trained model, add your own new layer(s) at the end and **train only the newly added layer(s).**
2. **Retrain** all the weights **starting (initialising) from the weights and biases** that the net has already learnt. Since you don't want to unlearn a large fraction of what the pre-trained layers have learnt. So, for the initial layers, we will choose a low learning rate.

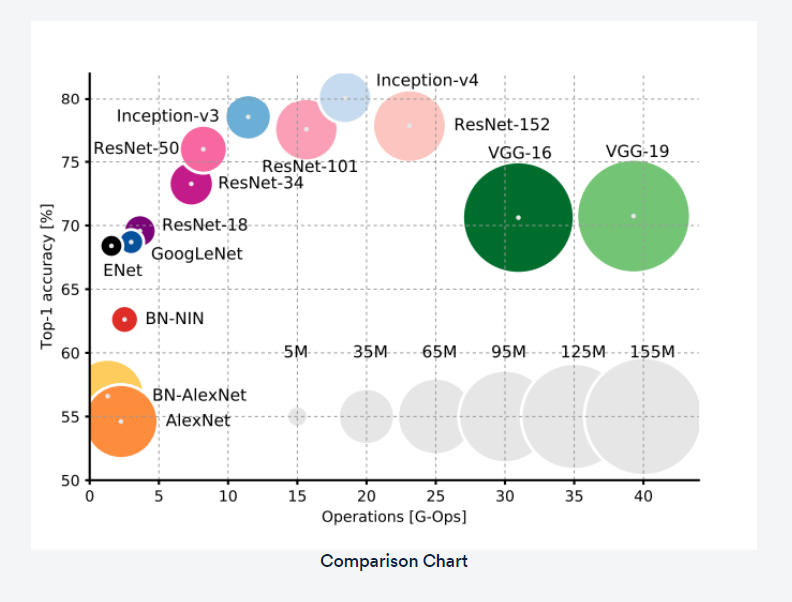
To summarise:

* If the task is a lot similar to that of the pre-trained model had learnt from, you can use most of the layers except the last few layers which you can retrain
* If you think there is less similarity in the tasks, you can use only a few initial trained weights for your task.

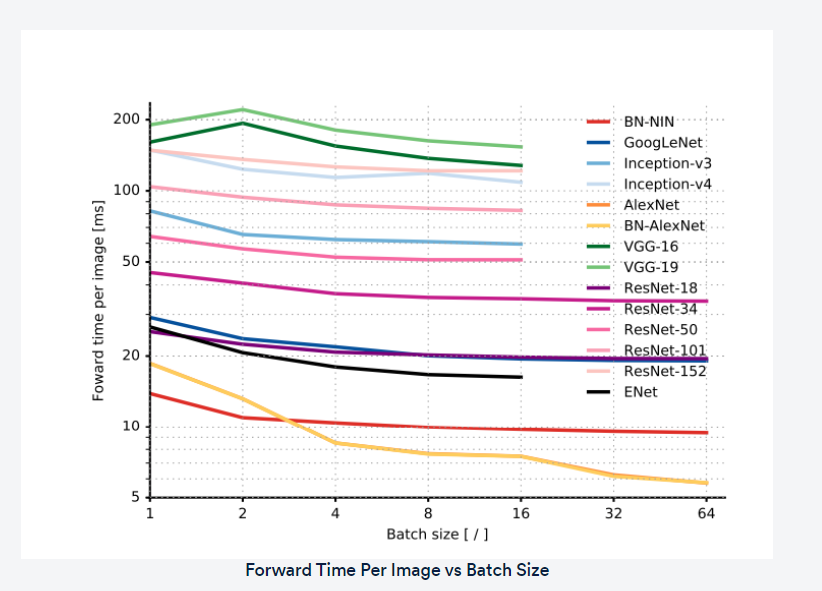
An important point to notice here is that although the VGGNet (VGG-16 and VGG-19) is used widely, it is by far the most expensive architecture — both in terms of the number of operations (and thus **computational time**) and the number of parameters (and thus **memory requirement**).

To summarise, some key points we have discussed are:

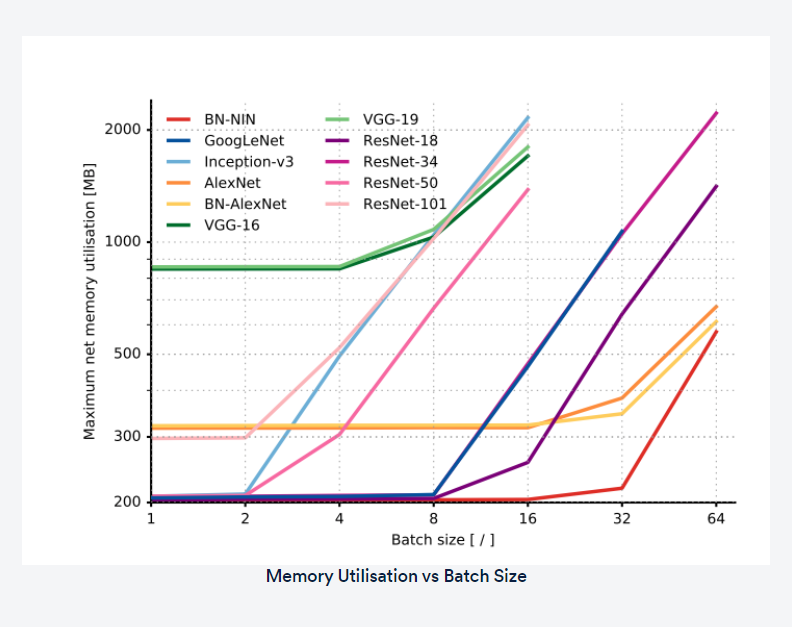
* Architectures in a particular cluster, such as **GoogleNet, ResNet-18 and ENet,** are very attractive since they have **small footprints**(both memory and time) as well as pretty good accuracies. Because of low-memory footprints, they can be used on **mobile devices,**and because the number of operations is small, they can also be used in **real time inference**.
* In some ResNet variants (ResNet-34,50,101,152) and Inception models (Inception-v3,v4), there is a **trade-off** between model accuracy and efficiency, i.e. the inference time and memory requirement.



* There is a **marginal decrease** in the (forward) inference time per image with the batch size. Thus, it might not be a bad idea to use a large batch size if you need to.



* Up to a certain batch size, most architectures use a constant memory, after which the consumption increases linearly with the batch size.



Let's conclude the important points from the latter part of the paper:

* Accuracy and inference time are in a hyperbolic relationship: a little increment in accuracy costs a lot of computational time.
* Power consumption is independent of batch size and architecture.
* The number of operations in a network model can effectively estimate inference time.
* ENet is the best architecture in terms of parameters space utilisation.

Q&A

**Summary**

In this session, you compared thearchitectures of some popular networks which had achieved state-of-the-art results in ImageNet:**AlexNet, VGGNet, GoogleNet and ResNet**.

Until the VGGNet, most of the major innovations had appeared in the form of increased depth, smaller filters, etc. In 2014, GoogleNet introduced an unconventional idea in the form of the **Inception module**, which performs multiple parallel convolutions (1 x 1, 3 x 3, 5 x 5, pooling etc.) on the input. This enabled GoogleNet to increase both the depth and the 'width' of the network (it has 22 layers with multiple inception modules stacked one over another). In the quest for training deeper networks, the **ResNet** team introduced another novel idea - **skip connections,**which enabled training extremely deep networks by 'by-passing the additional layers if they do not learn anything useful, else keeping them'.

Since these models have already been trained on millions of images, and therefore are **good at extracting generic features**, they are well-suited to solve other computer vision problems (with no or little re-training). This is the main idea of **transfer learning.**

In **transfer learning**, a pre-trained network can be repurposed for a new task depending on how much the new task differs from the original one. In two transfer learning experiments, we 1) trained a ResNet-50 by freezing the original weights and adding only a few FC layers, and 2) re-trained the last few layers of ResNet-50. The latter model gave us a boost in accuracy.

Finally, we compared various popular CNN architectures in terms of metrics (other than accuracy) which are important considerations for deployment (**inference time, memory requirements** etc.). We compared the architectures along metrics such as the number of parameters (proportional to memory)**,**operations involved in a feed-forward (proportional to inference time), accuracy, power consumption etc. We saw that some of the oldest architectures (AlexNet) are not suited for most tasks, some architectures are extremely accurate but have very high memory footprints, and that there are some clear **trade-offs** between accuracy and efficiency (computational time and memory).

