Keras:

high-level API for Tensor flow

User-friendly interface for building neural networks.

Supports TensorFlow, Theano and Microsoft Cognitive ToolKit (CNTK) as backend engines.

The core data structures of Keras are layers and models.

Sequential Model: The simplest model, a linear stack of layers (suitable for simple architecture with a single input and output.)

Functional API: for complex architectures, arbitrary graphs of layers (more flexibility in building neural networks with multiple inputs, multiple outputs, shared layers and even loops. These are even suitable for more intricate models(I/P, O/P networks), residual connections and models with branching layers)

Sequential Model

Steps:

1) Define model (stack of layers) (type of connection, units, activations, etc.)
2) Compile the model (loss function, optimiser, metrics, etc.)
3) Model fitting (X_train, Y_train, epochs, batch_size, etc.)
4) Evaluate test
5) Generate predictions

Base Layer Class: Class from which all layers inherit.

Activation Function: applying mathematical functions to neuron outputs, enabling neural networks to capture complex patterns in data by introducing non-linearities.

All built-in activations may also be passed via their string identifier.

Available activations:

a).ReLU (Rectified Linear Unit): $f(x) = \max(0, x)$ (It replaces all negative values in the input with zero, leaving positive values unchanged.)

Advantages:

1) Computationally efficient since it involves simple thresholding
2) Sparse activations (a subset of neurons is activated at a time), which can help with reducing overfitting.
3) Avoids the Vanishing gradient problem, making it suitable for deeper networks.
4) ReLU has a constant gradient for positive inputs, making it easier to optimize using gradient descent algorithms

Disadvantages:

1) **Dying ReLU Problem**: Neurons with negative inputs always output zero causing them to 'dead' and not contributing to the network's learning where the network is inactive.
2) **Unbounded Activation**: ReLU has unbounded activation for positive inputs, which may cause exploding gradients during training.

b). Sigmoid Function: $f(x)= 1/(1+e^{-x})$
Widely used activation, a smooth, S-shaped curve, Squashes input values to 0 or 1.

Used for binary classification tasks, output layer for neural networks fo

Advantages:

1) Easily interpretable.
2) Sigmoid function is smooth and differentiable everywhere, which is beneficial for gradient-based optimisation algorithms during training.

Disadvantages:

1) **Vanishing Gradient:** where gradients can become extremely small during backpropagation, which leads to slow convergence.
2) **Squashed Outputs**: input values to 0 or 1
3) **Not Zero-centered:** The sigmoid function is not zero-centered, which can lead to slower convergence when used in deeper networks and can make optimisation more challenging.

c). Softplus function: $f(x)=\ln(1+e^{x})$

It's a smooth, continuously differentiable function similar to the ReLU but with a smooth transition around zero.

It maps any real-valued number to a positive range. It is an alternative to ReLU.

Advantages:

1) Smooth and differentiable everywhere similar to ReLU

2) Positivity

Disadvantages:

1) Limited Range: While Soft Plus avoids the saturation issues of traditional sigmoid activations, It still has a limited range in the positive direction, which may not capture extreme positive values as effectively as unbounded functions like ReLU.
2) The softplus function involves computing exponential function, which can be computationally expensive compared to simple activation functions like ReLU.

d). Soft sign function: $f(x)= x/(1+|x|)$

The soft sign function is an activation function used in neural networks. It's a smooth, continuous function that maps any real-valued number to a range between -1 and 1.

The soft sign function is designed to provide a smooth, continuous transition between positive and negative values, ensuring that it remains differentiable everywhere.

Advantages:

1) **Smoothness:** The soft sign function is smooth and differentiable everywhere, making it suitable for gradient-based optimization algorithms during training.
2) **Symmetry:** The soft sign function is smooth and differentiable everywhere, making it suitable for gradient-based optimization algorithms during training.
3) **Avoids Saturation:** Soft sign avoids the saturation issues of traditional sigmoid activation functions, ensuring that gradients do not become extremely small for large inputs.

Disadvantages:

1) **Limited Range:** Like many other activation functions, soft sign has a limited range between -1 and 1, which may not capture extreme positive or negative values as effectively as unbounded functions like ReLU.
2) **Computational Complexity:** The soft sign function involves computing the absolute value, which can add computational complexity compared to simpler activation functions like ReLU.

e). Tanh Function (Hyperbolic Tangent): $f(x)= (e^{x}-e^{-x})/ (e^{x}+e^{-x})$

The tanh function maps any real-valued number to a range between -1 and 1, offering a zero-centred alternative to the sigmoid function.

Tanh is commonly used in hidden layers of neural networks, especially in scenarios where zero-centred activations are preferred or where inputs have both positive and negative values.

Advantages:

1) **Zero-centred:** Tanh outputs are zero-centred, which can help with faster convergence during training, especially in scenarios where inputs are centred around zero.

2) **Smoothness:** Tanh is smooth and differentiable everywhere, making it suitable for gradient-based optimization algorithms during training.

3) **Non-linear Mapping**: Tanh introduces non-linearity into the network, allowing it to learn complex patterns and relationships in the data.

Disadvantages:

1) **Saturation**: Tanh suffers from saturation issues, especially for large inputs, where gradients become extremely small, leading to slower learning or vanishing gradient problems.

2) **Computational Complexity**: Tanh involves computing the exponential function, which can add computational complexity compared to simpler activation functions like ReLU.

3) **Limited Range**: Tanh has a limited range between -1 and 1, which may not capture extreme positive or negative values as effectively as unbounded functions like ReLU.

1. **ELU (Exponential Linear Unit)**:
   - **Explanation**: ELU is similar to ReLU but with a smooth transition around zero, preventing "dying" neurons.
   - **Where to Use**: Suitable for hidden layers in deep neural networks.
   - **Advantages**: Helps mitigate vanishing gradient problem, retains zero-centered outputs.
   - **Disadvantages**: Slightly more computationally expensive than ReLU.
2. **PreLU (Parametric Leaky Rectified Linear Unit)**:
   - **Explanation**: Similar to Leaky ReLU, but the slope of the negative part is learned during training.
   - **Where to Use**: Used in CNNs and deep neural networks.
   - **Advantages**: Allows the network to learn the slope of the negative part.
   - **Disadvantages**: Introduces additional parameters to learn.
3. **Exponential Function**:
   - **Explanation**: The exponential function raises the base of Euler's number (e) to the power of the input value.
   - **Where to Use**: Can be used for various mathematical transformations.
   - **Advantages**: Useful for modeling exponential growth or decay.

- **Disadvantages**: Outputs can grow rapidly for large inputs, potentially leading to numerical instability.

4. **Leaky ReLU (Leaky Rectified Linear Unit)**:
   - **Explanation**: Leaky ReLU allows a small, non-zero gradient for negative inputs, preventing dying neurons.
   - **Where to Use**: Hidden layers in deep neural networks.
   - **Advantages**: Addresses the dying ReLU problem.
   - **Disadvantages**: May not perform well on all types of data.
5. **ReLU 6 (Clipped Rectified Linear Unit)**:
   - **Explanation**: Clips ReLU output to a maximum value of 6, preventing unbounded growth.
   - **Where to Use**: Suitable for networks where unbounded activations are undesirable.
   - **Advantages**: Prevents unbounded growth of activations.
   - **Disadvantages**: Loss of information in the clipped region.
6. **SiLU (Sigmoid Linear Unit)**:
   - **Explanation**: Also known as Swish, SiLU applies the sigmoid function to the input.
   - **Where to Use**: Suitable for hidden layers in neural networks.
   - **Advantages**: Smooth gradient, computational efficiency.
   - **Disadvantages**: May not consistently outperform ReLU.
7. **Hard-SiLU (Hard Sigmoid Linear Unit)**:
   - **Explanation**: A hard approximation of the SiLU function.
   - **Where to Use**: Can be used in situations where computational efficiency is crucial.
   - **Advantages**: Simplicity, computational efficiency.
   - **Disadvantages**: Less expressive power compared to full SiLU.
8. **GELU (Gaussian Error Linear Unit)**:
   - **Explanation**: Approximates the cumulative distribution function of the Gaussian distribution.
   - **Where to Use**: Suitable for deep learning models, particularly transformers.
   - **Advantages**: Smooth and continuous, good performance in certain architectures.
   - **Disadvantages**: Computational cost, may not always outperform ReLU variants.
9. **Hard-Sigmoid Function**:
   - **Explanation**: A piecewise linear approximation of the sigmoid function.
   - **Where to Use**: Suitable for models where a less complex activation function is needed.
   - **Advantages**: Simple and efficient computation.
   - **Disadvantages**: Less expressive than full sigmoid, limited range.
10. **Linear Function**:

- **Explanation**: The simplest activation function, outputs the input value as is.
- **Where to Use**: Used in the output layer for regression problems.
- **Advantages**: Simple, preserves input information.
- **Disadvantages**: Lacks non-linearity, may limit model expressiveness.

11. **Mish Function**:

- **Explanation**: Mish function is a smooth, non-monotonic activation function.
- **Where to Use**: Suitable for hidden layers in deep neural networks.
- **Advantages**: Smooth gradient, potentially better performance than ReLU.
- **Disadvantages**: May not consistently outperform other activation functions.

12. **Log Softmax Function**:

- **Explanation**: Applies the natural logarithm to the softmax function output.
- **Where to Use**: Used in the output layer for multi-class classification problems.
- **Advantages**: Stable computation, avoids numerical overflow/underflow.
- **Disadvantages**: Loss of interpretability compared to softmax.

13. **Softmax Function**:

- **Explanation**: Converts a vector of real numbers into a probability distribution.
- **Where to Use**: Used in the output layer for multi-class classification problems.
- **Advantages**: Produces probabilities, useful for classification tasks.
- **Disadvantages**: Sensitive to large input values, may cause vanishing gradients.

Weight Initialiser: It is a method used in neural network training to set the initial values of the weights in the network's layers, it helps the network learn effectively and converge to a good solution during training.

Types of weight initialisers: Random (Normal, uniform), Truncated Normal, Zeros, Glorot or Xavier (uniform, normal), He initialiser (Normal, uniform), Orthogonal Initialiser, Constant, Variance Scaling, Lecun (Normal, Uniform), Identity Initialisier.

Normal Distribution:

Gaussian distribution, bell-shaped curve with the majority of values concentrated around the mean.

Characterized by two parameters: mean and standard deviation

Uniform Distribution:

All values within a specified range are equally likely to occur, constant probability density within its range, values are equally distributed across the range.

Commonly used for weight initialization when a broader range of values is desired, or when no prior knowledge of the distribution of weights is available.

1)Random initialiser (Normal, Uniform): Initializes weights with random values drawn from a specified distribution (normal or uniform)

Suitable for general-purpose deep neural networks when no specific initializer is preferred.

Advantages: Simple to implement, breaks the symmetry, prevents neurons from getting stuck.

Disadvantages: Lack of control over initialization precision, may not be optimal for specific architectures or tasks.

2)Truncated Normal: Likely to random initialiser, but truncates values to beyond the values

Useful when outliers or extreme values need to be avoided.

Advantages: Reduces the likelihood of extreme weights, helpful for stabilizing training.

Disadvantages: Slightly slower initialization compared to standard random normal.

3)Zeros: Initializes all weights to zero.

Rarely used directly; often employed in specific cases such as biases or as a starting point for fine tuning.

Advantages: Simple and straightforward initialization.

Disadvantages: Can lead to symmetry issues and poor convergence if not used carefully.

4)Glorot or Xavier (Uniform, Normal): Initializes weights based on the Xavier initialisation method, using a distribution with zero mean and a variance calculated based on the number of input and output neurons.

Effective for tanh or sigmoid activation functions.

Advantages: Helps maintain variance of activations and gradients across layers, mitigates vanishing/exploding gradients.

Disadvantages: May not perform optimally with different activation functions.

5)He-Initilializer (Normal, Uniform): Similar to Xavier initialisation but scales the variance differently, particularly suitable for deep networks using ReLU activations.

Advantages: Helps prevent vanishing gradients, and facilitates faster convergence.

Disadvantages: Can lead to exploding gradients if not used carefully.

6)Orthogonal Initializer: Initializes weights as orthogonal matrices, useful for maintaining orthogonality in weight matrices which is beneficial for RNN

Advantages: Helps stabilize gradients in RNNs, and maintains matrix orthogonality.

Disadvantages: Higher computational cost compared to other initializers.

7)Constant: Initializes weights with a constant value, rarely used directly for weight initialisation, often used for initializing biases or in specific network layers.

Advantages: Provides control over Initialization values.

Disadvantages: May not be suitable as the sole initializer for weights due to a lack of variety, and slower convergence.

8)Variance Scaling: Scales weights based on the fan-in or fan-out of the layer and used as a general-purpose initializer when specific initialization is not required.

Advantages: Adapts initialization based on layer size, and prevents vanishing/exploding gradients.

Disadvantages: May not provide optimal initialisation for certain architectures or activation functions.

9)LeCun (Normal, Uniform): Initializes weights based on the LeCun initialization method, which scales weights by the square root of the number of inputs. Suitable for networks using tanh activation functions.

Advantages: Helps maintain signal variance through layers, and mitigates vanishing gradients.

Disadvantages; May not be optimal for architectures using other activation functions.

10)Identity Initializer: Initializes weights to an identity matrix, suitable for certain architectures or layers where initialization is desired.

Advantages; Preserves information flow in certain network structures.

Disadvantages: Limited applicability to specific scenarios; not suitable for general purpose initialization.

Layer weight regularizers: regularizers allow you to apply penalties on layer parameters or layer activity during optimisation

Kernel_regularizer: Regularizer to apply a penalty on the layer's kernel

Bias_regularizer: Regularizer to apply a penalty on the layer's bias

Activity_regularizer: Regularizer to apply a penalty on the layer's output

L1 regularizer

L2 regularizer

L1 + L2 regularizer

Layer weight constraints: setting constraints on model parameters during training. There are two arguments where are constraints applied.

a)kernel_constraint: for the main weights matrix

b)bias_constraint: for the bias

Available weight constraints:

1)Non Negative constraint

2)MaxNorm (e.g, max_value = 2)

3)MinMaxNorm (min_value, max_value)

4)UnitNorm-→ enforce that the weights of a model are normalized to have a unit norm (i.e., a norm of 1). This constraint can be particularly useful in situations where you want to prevent the model from becoming too sensitive to small changes in the input data.

Core Layers:

1)Input object: It is not technically a layer, it defines the shape and dtype of the inputs into a model

2)InputSpec object: It is similar to input object but we need to specify constraints.

3)Dense layer: It is a fully connected layer where each neuron is connected to every neuron in the previous layer. It is one of the most common types of layers used in neural networks for learning non-linear relationships in the data.

4)EinsumDense layer: It is a specialised variant of the dense layer that uses the Einstein summation convention for matrix multiplication. It can be useful in scenarios where we need to perform custom tensor operations efficiently.

5)Activation layer: layer applies an element-wise activation function to the output of the previous layer.

6)Embedding layer: layer is used for learning dense vector representations of categorical variables or discrete entities in the input data. It is commonly used in NLP tasks where words or tokens are converted into dense vectors.

7)Masking layer: The layer is used to handle variable-length input sequences by masking out certain timesteps where the input data is padded. It is often used in sequence-to-sequence models or RNNs to handle sequences of varying lengths efficiently.

8)Lambda layer: layer allows customs operations in a model (computations or transformations).

9)Identity layer: The layer simply returns its input without any transformation. It can be useful in certain scenarios for debugging.

Convolution Layers: These convolution layers are fundamental building blocks in deep learning models, allowing neural networks to automatically learn hierarchical representations of data, capturing spatial, temporal, or spatiotemporal patterns depending on the dimensionality of the data. They are widely used in computer vision, natural language processing, and many other fields for feature extraction and representation learning.

1)Conv1D layer: used for processing sequences such as time series or natural language data. It slides a 1D kernel over the input data to extract features.

2)Conv2D layer: used for processing images. It slides a 2D kernel over the input image to detect spatial patterns and features.

3)Conv3D layer: primarily applied in video processing and volumetric data analysis. It slides a 3D kernel over the input volume to extract spatiotemporal features.

4)SeparableConv1D layer: It separates spatial and depth-wise convolutions. It can reduce computational complexity while retaining effectiveness, commonly used in resource-constrained environments or mobile applications.

5)SeparableConv2D layer: similar to SeparableConv1D, often used in image processing tasks to reduce computational cost without significant loss in performance.

6)DepthwiseConv1D layer: Depth-wise separable convolution applies a single convolution filter per input channel, reducing the number of parameters and computational cost compared to traditional convolution. It's useful for lightweight models or tasks with limited computational resources.

7)DepthwiseConv2D layer: Depthwise separable convolution in 2D, where each input channel is convolved separately with a kernel. It's commonly used in mobile and embedded applications where computational efficiency is critical.

8)Conv1DTranspose layer: Also known as ‘deconvolution’ or ‘transposed convolution’, this layer performs the inverse operation of Conv1D. It is used in tasks like up-sampling in autoencoders or generating high-resolution outputs in generative models like GANs.

9)Conv2DTranspose layer: Similar to Conv1DTranspose, Conv2DTranspose performs transposed convolution in two dimensions. It's commonly used in image segmentation, super-resolution, and generating images in generative models.

10)Conv3DTranspose layer: Conv3DTranspose performs transposed convolution in three dimensions, often used in tasks like 3D image reconstruction, volumetric segmentation, and generating volumetric data in 3D generative models.

Pooling Layers: Pooling layers are essential components of CNNs used in deep learning for feature extraction and dimensionality reduction. They are typically inserted between convolution layers to reduce spatial dimensions while retaining important information. They help in capturing the most salient features while discarding irrelevant details. The choice of pooling layer depends on the nature of the input data, the specific task, and the desired trade-off between spatial resolution and computational efficiency.

1)MaxPooling1D layer: Extracting maximum value from each window along a temporal dimension of a 1D input sequence. It’s commonly used in sequence data processing tasks like text classification or time series analysis.

2)MaxPooling2D layer: Similar to MaxPooling1D layer but it operates on 2D input features maps(images). It’s effective for image recognition tasks, reducing spatial dimensions while preserving important features.

3)MaxPooling 3D layer: Similar to the above layers, but it works with 3D input volumes, typically used in video processing or volumetric data analysis.

AveragePooling 1D, 2D, and 3D layers: Instead of taking the maximum value, this layer computes the average value within each window.

GlobalMaxPooling1D layer: Computes the maximum value across the entire temporal dimension of a 1D input sequence. It’s useful when the entire sequence holds important information, such as in sentimental analysis or audio classification. The same concept works for 2D and 3D layers.

GlobalAveragePooling1D layer: Computes the average value across the entire temporal dimension of a 1D input sequence. It's suitable for tasks where global information is more relevant than local features. The same concept works for 2D and 3D layers.

Recurrent Layers: These layers are specifically designed to handle sequential data, where the order of the data points matters. They maintain an internal state that evolves as new inputs are processed, allowing them to capture temporal dependencies within the data.

1)LSTM layer: LSTM stands for Long Short-Term Memory. It's a type of recurrent layer that addresses the vanishing gradient problem in traditional RNNs by introducing gating mechanisms. These gates decide what information to keep or discard from the cell state, allowing LSTMs to effectively capture long-term dependencies in sequential data.

Advantages: Effective in capturing long-term dependencies, overcomes vanishing gradient problem, can handle sequences of variable length.

Disadvantages: Computationally more expensive than simple RNNs, may be prone to overfitting with large datasets.

Limitations: Limited parallelism due to its sequential nature, can struggle with extremely long sequences.

2)LSTM Cell layer: Represents a single LSTM unit within an LSTM layer. It consists of various gates (input, forget, and output gates) and a cell state. The gates control the flow of information into and out of the cell state, allowing the LSTM to regulate the flow of information through time.

3)GRU layer: GRU stands for Gated Recurrent Unit. It's a simplified version of the LSTM architecture where forget and input gates into a single 'update gate'. This reduces the number of parameters in the model compared to LSTM while maintaining comparable performance.

Advantages: Fewer parameters compared to LSTM, computationally more efficient.

Disadvantages: May not perform as well as LSTM in capturing long-term dependencies, less expressive power.

Limitations: holds vanishing gradient problem, may struggle with very long sequences.

4)GRU Cell layer: Similar to the LSTM Cell layer, but represents a single unit within a GRU layer.

5)SimpleRNN layer: It's a basic form of a recurrent layer, where the output at each time step is computed based on the current input and the previous hidden state. However, SimpleRNNs suffer from the vanishing gradient problem and are not capable of capturing long-term dependencies effectively.

6)TimeDistributed layer: This layer applies the same operation to each time step of a sequence independently. Every input should be at least 3D, and the dimension of index one of the first input will be considered to be the temporal dimension.

7)Bidirectional layer: It processes the input sequence in both forward and backward directions. This allows to capture information from past and future contexts simultaneously.

Advantages: Captures information from both past and future contexts, potentially improves performance in sequence modelling tasks.

Disadvantages: Increases computational complexity, and may introduce delays in real-time applications due to processing in both directions.

Limitations: Cannot be used in online learning scenarios, as it requires access to the entire input sequence.

8) ConvLSTM1D/ConvLSTM2D/ConvLSTM3D layers: These layers extend the LSTM architecture by incorporating convolutional operations into recurrent processing. They are particularly useful for tasks involving spatiotemporal data, such as video prediction or weather forecasting.

9)Base RNN layer: This represents a generic RNN layer without any specific architecture like LSTM or GRU.

10) Simple RNN cell layer: Represents a single unit within a SimpleRNN layer.

11)Stacked RNN cell layer: Involves stacking multiple recurrent cells on top of each other to create deeper architectures. This can potentially increase the model's capacity to capture complex patterns in sequential data but also increases computational complexity and training time.

Preprocessing layers: These layers are essential components in the process of preparing the data for machine learning models. These layers are used to transform input data into a format that is suitable for training and interference

1)Text preprocessing:

Text Vectorization layer: This layer spilt the sentence into words which are called tokens and these tokens are converted into numerical vectors. It often includes options for lowercasing text, removing punctuation, and handling out-of-vocabulary tokens. Text Vectorisation is crucial for converting textual data into a format that machine learning models can process effectively.

2)Numerical features preprocessing layers:

a)Normalization layer: This layer scales numerical features to a similar range, typically between 0 and 1, or with a mean of 0 and a standard deviation of 1. Normalization prevents features with larger magnitudes from dominating the training process and helps in gradient-based optimization.

b)Spectral Normalization layer: It is a technique primarily used in deep neural networks to stabilise training by constraining the Lipschitz constant of the weight matrices. It helps in preventing the explosion of weights during training.

c)Discretization layer: Converts continuous numerical features into discrete categories or bins. This can be helpful when working with algorithms that perform better with categorical data or when simplifying the representation of continuous variables.

3) Categorical features preprocessing layers:

a)Category Encoding layer: Encodes categorical features into a numerical format suitable for machine learning algorithms. One-hot encoding and ordinal encoding are common techniques used.

b)Hashing layer: Hashing converts categorical features into a fixed-size representation using a hash function. It's useful when the number of unique categories is large and traditional encoding methods become impractical. For example, if you have a categorical feature like 'City' with hundreds or thousands of unique cities, one-hot encoding would create a binary vector with a dimension for each city, most of which would be zeros. This can lead to memory inefficiency and computational overhead, especially when dealing with large datasets.

In such cases, the hashing layer can be a more efficient alternative. It converts categorical features into a fixed-size representation using a hash function. The hash function maps each category to a specific index in a fixed-size space, regardless of the total number of unique categories. This helps in reducing the dimensionality of the feature space and can be more memory-efficient compared to traditional encoding methods.

c)HashedCrossing layer: HashedCrossing combines categorical features using hashing techniques to generate new features. It's often used in conjunction with hashing layers to create interaction features between categorical variables.

d)StringLookup layer and IntegerLookup layer: These layers map categorical strings or integer indices to embeddings, which are numerical representations suitable for input into neural networks.

4)Image preprocessing layers:

a)Resizing layer: Resizing layers are used to resize images to a specified dimension. It's essential to ensure that images are uniform before feeding them into machine learning models.

b)Rescaling layer: This layer scales pixel values of images to a specified range, often between 0 and 1, to ensure numerical stability during training.

c)CenterCrop layer: CenterCrop layer crops the center portion of images to a specified size, which can be useful for removing irrelevant background information or focusing on the main subject of an image.

5)Image augmentation layers:

a)RandomCrop layer: Randomly crops images to introduce variability and reduce overfitting. It helps the model learn to generalize better to unseen data.

b)RandomFlip layer: flips images horizontally or vertically to augment the training data and make the model more robust to orientations.

c)Random Translation layer: translates images horizontally or vertically to simulate changes in viewpoint or perspective.

d)RandomRotation layer: rotates images by a specified angle to introduce variability and make the model invariant to rotation.

e)RandomZoom layer: zooms into or out of images to simulate changes in scale or perspective.

f)RandomContrast layer and RandomBrightness layer: These layers adjust contrast and brightness randomly to introduce variability in illumination conditions, making the model more robust to different lighting conditions.

Normalization layers: It is a crucial technique used in machine learning and deep learning to improve the convergence and stability of training algorithms. It involves adjusting the input data or intermediate representations within a model to ensure that they have a consistent scale or distribution. Normalization is particularly important in deep neural networks, where it helps address issues such as vanishing gradients, exploding gradients, and unstable training dynamics.

1)BatchNormalization layer: widely used in deep learning, especially in CNNs and deep forward networks. It operates on mini-batches of data during training and normalizes the activations of each layer by adjusting the mean and variance of the inputs. By normalizing activations within each mini-batch, batch-normalization allows for faster training and enables the use of higher learning rates.

2)LayerNormalization: normalizes the activations of each layer across the feature dimension (i.e., along the channels or neurons) instead of across the batch dimension. It calculates the mean and variance for each feature independently, it is often used in RNNs and transformer architectures, where batch sizes may vary or where batch normalisation may not be suitable due to the sequential nature of the data. It also helps stabilise the training of deep models and improve their generalization performance.

3)UnitNormalisation layer: It is similar to layer normalization but operates at the level of individual instances rather than across the feature dimension. It is commonly used in style transfer networks, image-to-translation tasks, and generative adversarial networks(GANs) to ensure consistency in the learned representations.

4)GroupNormalization layer: Group normalization divides the channels of the input into groups and computes the mean and variance for each group separately. Unlike batch normalization, which operates on mini-batches, group normalisation does not depend on batch size and is less sensitive to batch statistics. It is beneficial when batch sizes are small or when batch normalisation may not be suitable, such as in transfer learning scenarios or in models with limited computational resources.

Regularisation layer: Regularization techniques are crucial in ML and DL to prevent overfitting and improve the generalisation ability of models.

1)Dropout Layer: Dropout is a technique where randomly selected neurons are ignored during the technique. It involves setting a fraction of input units to zero at each update during training, which helps prevent overfitting by reducing the co-adaptation of neurons.

Advantages:

1)Effectively prevents overfitting.

2)Reduces reliance on specific sets of neurons, improving generalization.

Disadvantages:

1)Increases training due to multiple passes.

2)May increase the variability of the learning process, making convergence harder.

Limitations:

Dropout can be computationally expensive and may not always lead to significant improvements in performance.

2)SpatialDropout1D, SpatialDropout2D, and SpatialDropout3D Layer: These layers are variants of dropout specifically designed for spatial data such as images or sequences. Instead of dropping entire neurons, these layers drop entire features(channels) along the spatial dimensions.

Advantages:

1)Particularly effectively for spatial data like images or sequences.

2)Maintains spatial structure within the data.

Limitations:

Limited to spatial data and may not generalize well to other types of data.

3) Gaussian dropout layer: It is a variation of dropout where the dropout mask is sampled from a Gaussian distribution. Which might be helpful in certain scenarios like training generative models.

Advantages:

Provides a smoother regularization effect compared to traditional dropout.

Disadvantages:

Requires additional computation due to sampling from a Gaussian distribution.

4)AlphaDropout Layer: Alpha dropout is a variant of dropout designed specifically for ReLUs. It maintains the mean and variance of the activations closer to the theoretical values during training.

Only for ReLU activations.

5)GaussianNoise Layer: The layer adds Gaussian noise to the input data during training, acting as a form of regularisation.

Advantages: Provides a simple form of regularisation by adding noise to the input.

Disadvantages: May not be as effective as dropout in preventing overfitting.

6)ActivityRegualarisation layer: Activity regularisation penalises the activations of the network, encouraging them to be smaller during training.

Advantages: Directly penalizes large activations, which can help prevent overfitting.

Disadvantages: Might slow down training due to the additional; penalty term.

Use activity regularisation when you want to directly penalize large activations in the network.

Attention Mechanisms:

These have become a pivotal component in modern deep learning architectures, particularly in tasks involving sequential or hierarchical data.

1)Attention Layer: It helps the model focus on relevant parts of the input sequence when generating an output by computing attention weights between a set of queries and a set of key-value pairs, typically used in sequence-to-models. Use attention layers in sequence-to-sequence tasks where capturing dependencies between different parts of the input sequence is crucial.

2)MultiHeadAttention Layer: The model focuses on different parts of the input sequence simultaneously. It computes multiple sets of attention weights in parallel, each with its own set of learned parameters.

May introduce additional complexity and require careful tuning of the number of attention heads.

Increased computational and memory requirements due to the parallel computation of multiple attention heads.

Used in tasks involving long sequences or multiple modalities.

3)GroupQueryAttention Layer: It is a mechanism that groups queries into clusters and computes attention within each cluster separately. It helps in reducing computational complexity compared to standard attention mechanisms while still capturing relevant information.

Advantages:

Reduces computational complexity by grouping queries.

Can be more efficient for large-scale models.

Disadvantages:

May sacrifice some level of fine-grained attention compared to standard attention mechanisms.

Limitations:

Performance may degrade if the grouping strategy is not carefully designed.

4) Additive Attention Layer: It computes attention scores using a feedforward neural network with a single hidden layer. It provides a more flexible attention mechanism compared to standard dot-product attention, allowing the model to learn complex attention patterns.

Reshaping Layers:

These are fundamental components in deep learning architectures that allow the transformation of input data into different shapes or dimensions.

1)Reshape layer: This layer changes the shape of the input tensor without changing its data. It allows the reorganisation of data into different dimensions.

Advantages: Enables flexibility in reshaping input data to match the required input shape of subsequent layers.

Disadvantage: May require careful design to ensure compatibility with subsequent layers.

Limitation: Limited to reshaping input tensors without altering the data itself.

Use the Reshape layer when the input data needs to be reshaped to fit the expected input shape of subsequent layers, such as when transitioning between convolutional and fully connected layers.

2)Flatten Layer: The Flatten layer transforms a multi-dimensional tensor into a 1D vector by collapsing all dimensions except the batch dimension.

Advantage: Simplifies the input for subsequent fully connected layers

Disadvantage: May lose spatial or structural information present in higher-dimensional data.

Limitation: Not suitable for tasks where spatial or structural relationships within the data are important.

Use the Flatten layer when transitioning from convolution layers to fully connected layers in tasks like image classification.

3)RepeatVector layer: Allows the replication of data along a specific axis, useful for sequence-to-sequence tasks.

4)Permute Layer: It is used when the order of dimensions in the input tensor needs to be changed, such as when working with different data formats or reshaping for specific operations.

5) Cropping 1D, Cropping 2D, Cropping 3D Layers: These layers remove specific portions of the input tensor along one or more dimensions. It just focuses on relevant regions like removing padding from images in CNNs.

6)UpSampling 1D, UpSampling 2D, UpSampling 3D Layers: These layers increase the spatial dimensions of the input tensor by inserting repetitions of data along specified axes. Used in such cases like image segmentation or super-resolution tasks.

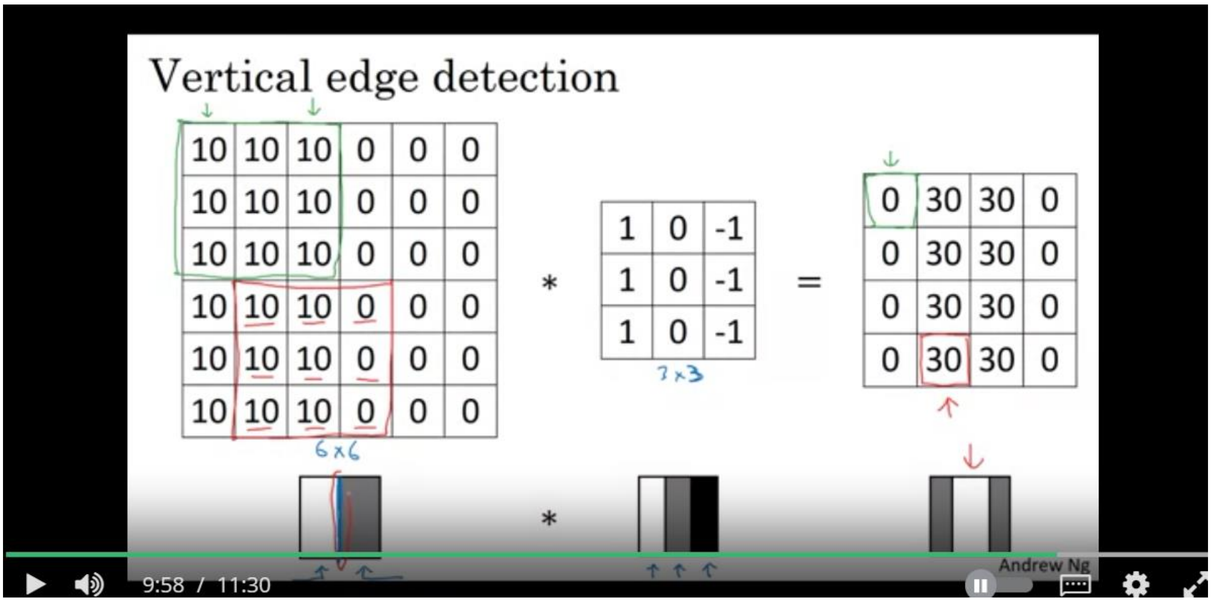
7) ZeroPadding 1D, ZeroPadding 2D, ZeroPadding 3D Layers: These layers add zeros to the edges of the input tensor along specified dimensions. Helps preserve spatial dimensions and prevent information loss, particularly in CNNs

Computer Vision: It is a field of Artificial Intelligence and computer science that focuses on enabling computers to interpret and understand visual information from the real world. It involves tasks like image recognition, object detection, tracking, scene reconstruction, and more.

Essentially, it aims to replicate the human visual system's ability to perceive and understand visual data, allowing machines to analyse and make decisions based on images or videos.

Edge Detection: It is a fundamental technique in computer vision to identify boundaries within images. The edges represent significant changes in intensity or colour within an image.

It is a crucial preprocessing step in many computer vision tasks, including object detection, image segmentation, and feature extraction. It helps in reducing the complexity of subsequent analysis by focusing on the most relevant parts of the image.



Vertical edge detection

$$\underset{6\times 6}{\begin{bmatrix}10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\end{bmatrix}} * \underset{3\times 3}{\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}} = \begin{bmatrix}0&30&30&0\\0&30&30&0\\0&30&30&0\\0&30&30&0\end{bmatrix}$$

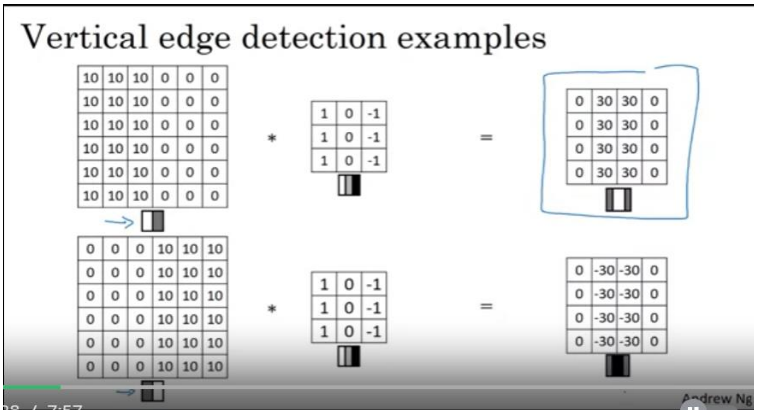
Algorithms for Edge detection:

a)Gradient-based methods: These methods detect edges by identifying abrupt changes in intensity or colour using the gradient of the image. Common gradient-based operators include the Sobel, Prewitt, and Roberts operators.

b)Laplacian of Gaussian (LoG): This method involves convolving the image with a Gaussian filter to smooth it and then applying the Laplacian operator to detect regions of rapid intensity change.

c)Canny edge detector: This is one of the most popular edge detection algorithms. It involves multiple steps including Gaussian smoothing, gradient calculation, non-maximum suppression, and edge tracking by hysteresis.

d)Morphological methods: These techniques involve the use of mathematical operations such as dilation, erosion, and opening /closing to highlight and enhance edges.

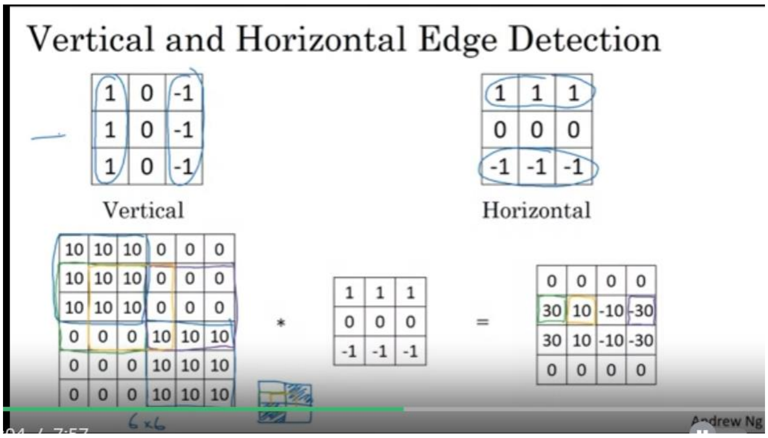


Vertical edge detection examples

$$\begin{bmatrix}10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\end{bmatrix} * \begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix} = \begin{bmatrix}0&30&30&0\\0&30&30&0\\0&30&30&0\\0&30&30&0\end{bmatrix}$$

$$\begin{bmatrix}0&0&0&10&10&10\\0&0&0&10&10&10\\0&0&0&10&10&10\\0&0&0&10&10&10\\0&0&0&10&10&10\\0&0&0&10&10&10\end{bmatrix} * \begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix} = \begin{bmatrix}0&-30&-30&0\\0&-30&-30&0\\0&-30&-30&0\\0&-30&-30&0\end{bmatrix}$$

Here +30 values are in white and -30 values are in darker region, so we can conclude that negative and positive values make a impact on changing colors.



Vertical and Horizontal Edge Detection

$$\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}$$

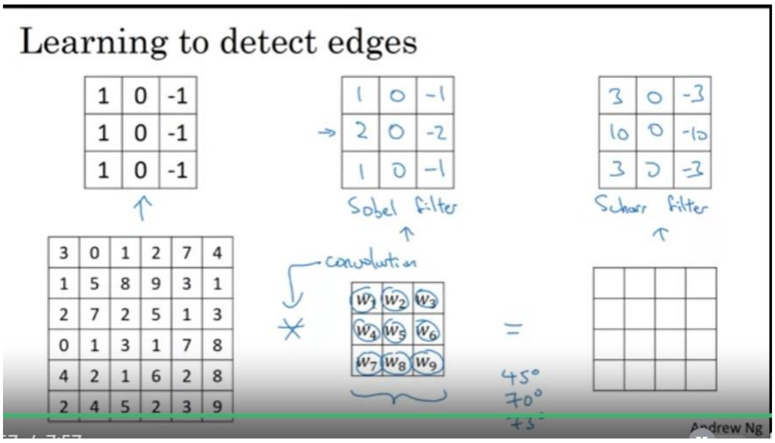
Vertical

$$\begin{bmatrix}1&1&1\\0&0&0\\-1&-1&-1\end{bmatrix}$$

Horizontal

$$\underbrace{\begin{bmatrix}10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\\0&0&0&10&10&10\\0&0&0&10&10&10\\0&0&0&10&10&10\end{bmatrix}}_{6\times6} * \begin{bmatrix}1&1&1\\0&0&0\\-1&-1&-1\end{bmatrix} = \begin{bmatrix}0&0&0&0\\30&10&-10&-30\\30&10&-10&-30\\0&0&0&0\end{bmatrix}$$

Andrew Ng



Learning to detect edges

$$\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}$$

$$\begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}$$

Sobel filter

$$\begin{bmatrix}3&0&-3\\10&0&-10\\3&0&-3\end{bmatrix}$$

Scharr filter

convolution

$$\begin{bmatrix}3&0&1&2&7&4\\1&5&8&9&3&1\\2&7&2&5&1&3\\0&1&3&1&7&8\\4&2&1&6&2&8\\2&4&5&2&3&9\end{bmatrix} * \underbrace{\begin{bmatrix}w_1&w_2&w_3\\w_4&w_5&w_6\\w_7&w_8&w_9\end{bmatrix}} = \begin{bmatrix}\,&\,&\,&\,\\\,&\,&\,&\,\\\,&\,&\,&\,\\\,&\,&\,&\,\end{bmatrix}$$

45°
70°
73°

Andrew Ng

Some of the famous filters are the Sobers filter/operator and the next Scharr filter, In filters, the numbers can be rotational values like 45deg, 50deg, etc.

Common filters/operators are:

1)Sobel operator/filter: gradient type, 3x3 convolution kernel.

Sobel horizontal edge detection kernel:

-1 0 1

-2 0 2

-1 0 1

Sobel vertical edge detection kernel:

-1 -2 -1

0 0 0

1 2 1

2)Prewitt Operator:

Prewitt horizontal edge detection kernel:

-1 0 1

-1 0 1

-1 0 1

Prewitt vertical edge detection kernel:

-1 -1 -1

0 0 0

1 1 1

3)Scharr operator: Scharr operators are similar to Sobel operators but provide better rotational symmetry. They offer improved performance for edge detection, especially for diagonal edges. Scharr operators consist of two 3x3 convolution kernels:

Scharr horizontal edge detection kernel:

-3 0 3

-10 0 10

-3 0 3

Scharr vertical edge detection kernel:

-3 -10 -3

0 0 0

3 10 3

4)Laplacian Operator: The Laplacian operator detects edges by computing the second derivative of the image intensity. It is often used in combination with Gaussian smoothing to reduce noise. The 3x3 Laplacian kernel is:

0 1 0

1 -4 1

0 1 0

Padding: Padding refers to adding extra border pixels to the input images before applying the convolution operation. Padding is a crucial parameter in CNN architectures, and it serves serval purposes:

1)Preserving Spatial Information: Convolution layers in CNNs apply filters (kernels) to input images to extract features. During this process, the dimension of the output feature maps(pixels) may decrease. Padding helps maintain the spatial dimensions of the input and output feature maps. Without padding, the spatial dimensions of the feature maps would shrink with each convolutional layer, which leads to information loss.

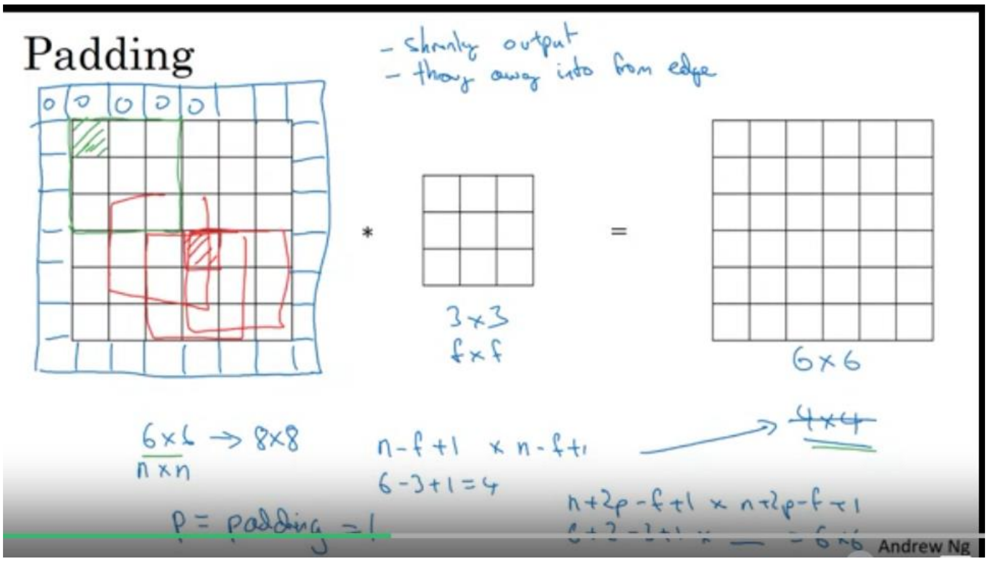
2)without padding, it results in less accurate feature detection.

3)Output size can be controlled with padding.

Padding is typically applied symmetrically around the input image, where the number of extra pixels added to each side (top, bottom, left, and right) is determined by the padding size specified by the user. These are two common types of padding:

a)Valid (No padding): Also known as ‘valid’ convolution, this type of convolution does not use padding. As a result, the spatial dimensions of the output feature maps are smaller than those of input images, depending on the filter size and stride.

b)Same Padding: Input and output feature maps have the same spatial dimensions. This is achieved by adding an equal number of zeros around the input image.



Padding

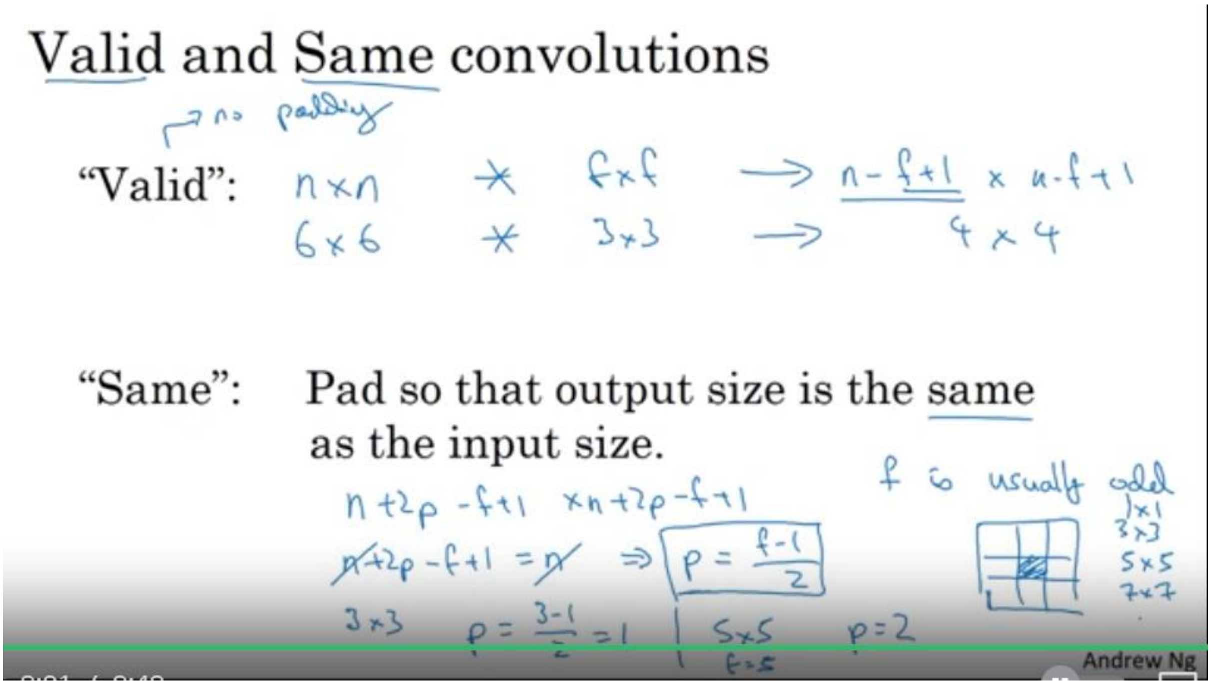
n = input size

f = filter size, and it is usually odd because 1) Input and output feature maps have the same spatial dimensions.

2) maintaining padding as symmetric.

3) one-centered.

p = padd size

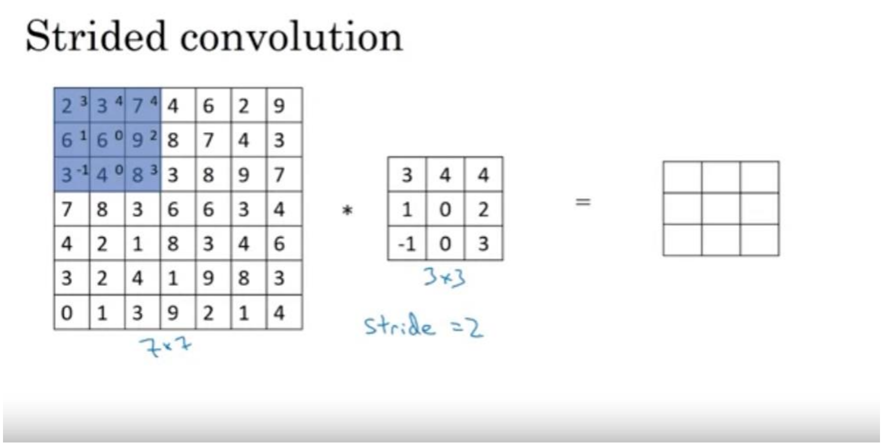


Valid and Same convolutions

"Valid":

"Same": Pad so that output size is the same as the input size.

Strided Convolution: It is a technique used in CNNs for processing input data with a specific stride length. In traditional convolution operations, a filter (also known as a kernel) slides across the input data with a stride of 1, meaning it moves one pixel at a time. However, in strided convolution, the filter moves more than one pixel at a time, which depends on strided length.



Strided convolution

2^{3}	3^{4}	7^{4}	4	6	2	9
6^{1}	6^{0}	9^{2}	8	7	4	3
3^{-1}	4^{0}	8^{3}	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7×7

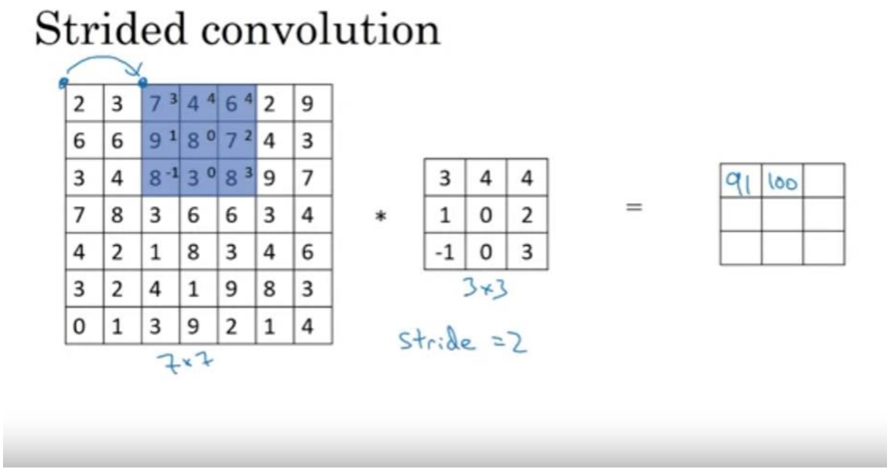
*

3	4	4
1	0	2
-1	0	3

3×3

stride = 2

=



Strided convolution

2	3	7^{3}	4^{4}	6^{4}	2	9
6	6	9^{1}	8^{0}	7^{2}	4	3
3	4	8^{-1}	3^{0}	8^{3}	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7×7

*

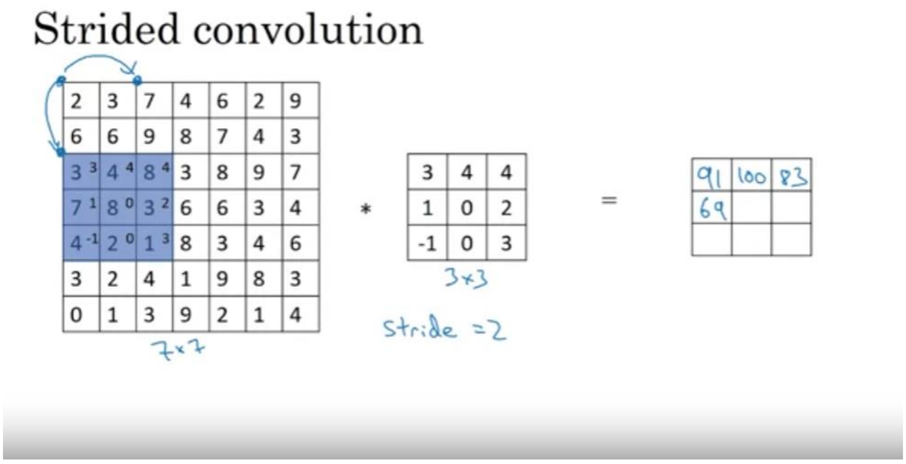
3	4	4
1	0	2
-1	0	3

3×3

stride = 2

=

91	100	



Strided convolution

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3^{3}	4^{4}	8^{4}	3	8	9	7
7^{1}	8^{0}	3^{2}	6	6	3	4
4^{-1}	2^{0}	1^{3}	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7×7

*

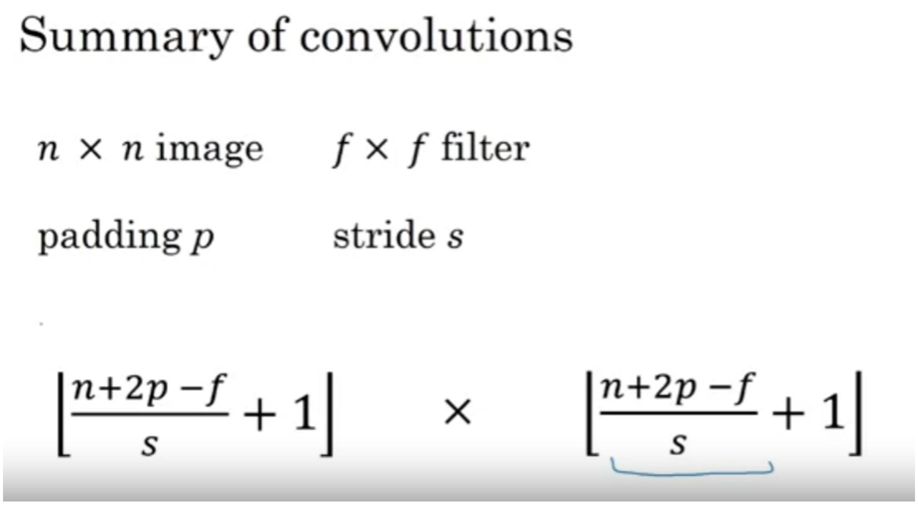
3	4	4
1	0	2
-1	0	3

3×3

stride = 2

=

91	100	83
69		

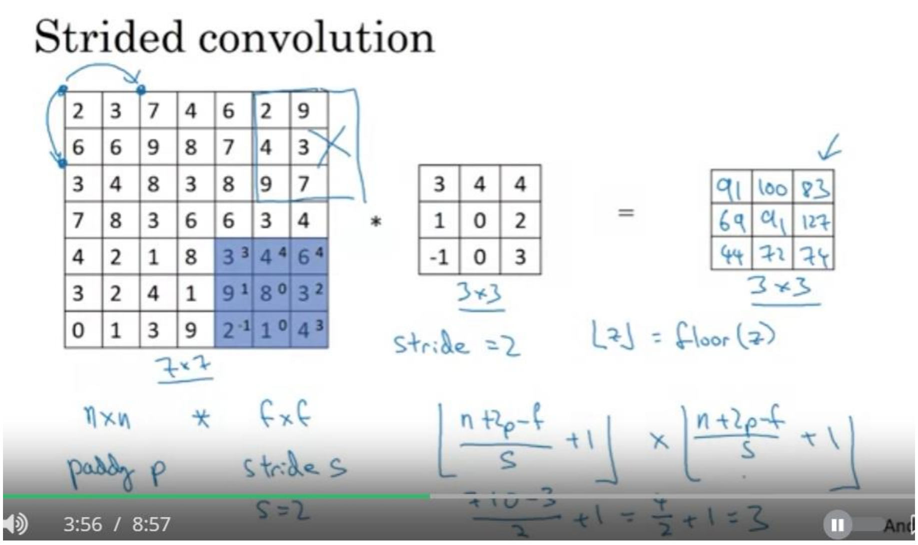


Summary of convolutions

$n \times n$ image $\quad f \times f$ filter

padding p $\quad$ stride s

$$\left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{\underbrace{s}_{}}+1 \right\rfloor$$



Strided convolution

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3^{3}	4^{4}	6^{4}
3	2	4	1	9^{1}	8^{0}	3^{2}
0	1	3	9	2^{-1}	1^{0}	4^{3}

7×7

*

3	4	4
1	0	2
-1	0	3

3×3

stride = 2

=

91	100	83
69	91	127
44	72	74

3×3

$\lfloor z \rfloor$ = floor(z)

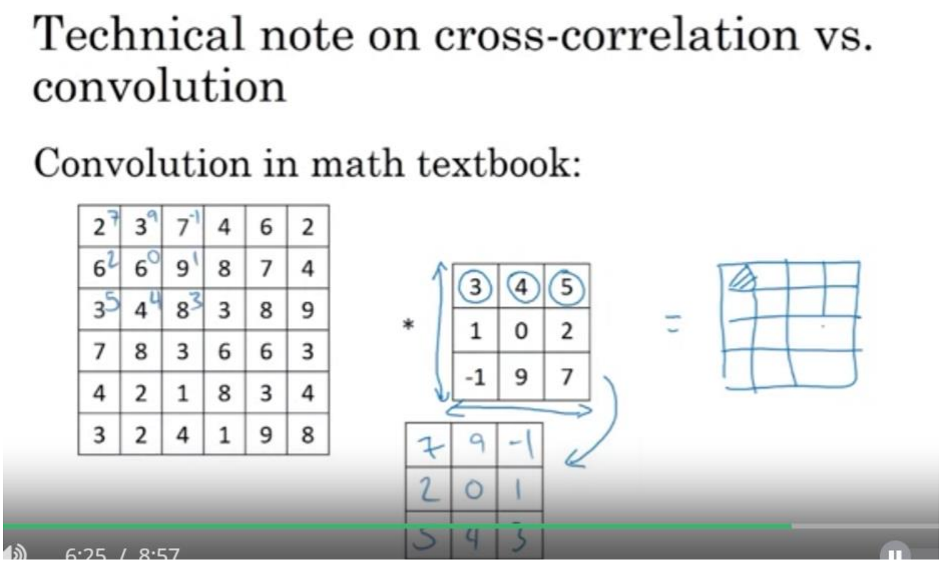
n×n $\quad$ * $\quad$ f×f

padding p $\quad$ stride s

s=2

$$\left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s}+1 \right\rfloor$$

$$\frac{7+0-3}{2}+1 = \frac{4}{2}+1 = 3$$



Technical note on cross-correlation vs. convolution

Convolution in math textbook:

2^{7}	3^{9}	7^{-1}	4	6	2
6^{2}	6^{0}	9^{1}	8	7	4
3^{5}	4^{4}	8^{3}	3	8	9
7	8	3	6	6	3
4	2	1	8	3	4
3	2	4	1	9	8

*

③	④	⑤
1	0	2
-1	9	7

7	9	-1
2	0	1
5	4	3

=

Strided convolution is commonly used in CNNs for several reasons:

a)Dimensionality Reduction

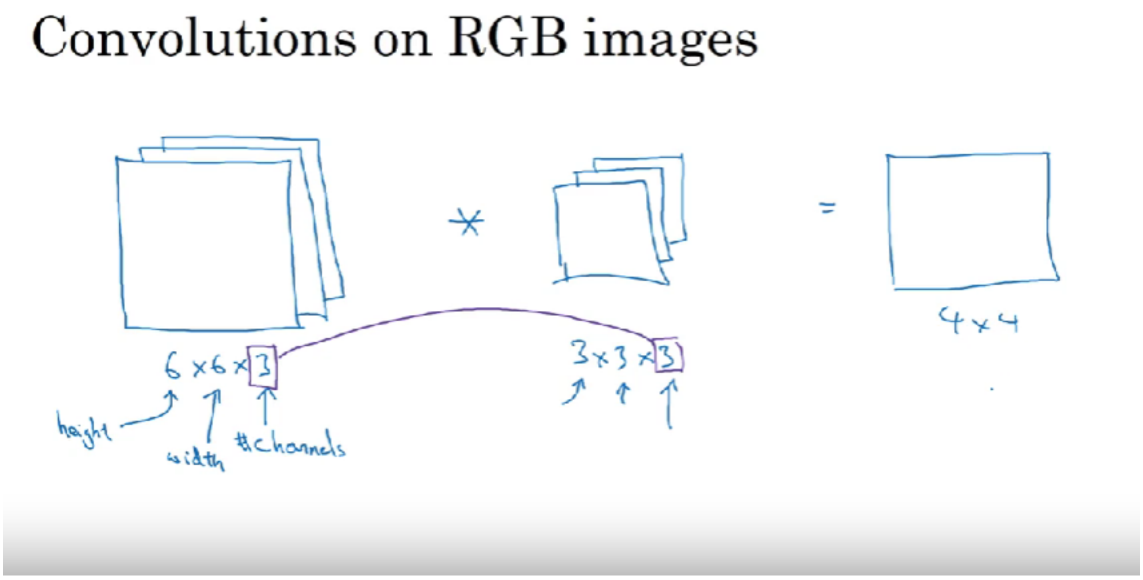
b)Feature Extraction

c) Strided convolution can sometimes replace pooling layers like max pooling, which are used for downsampling, without introducing additional pooling layers. This simplifies the network architecture by capturing important patterns in the data.

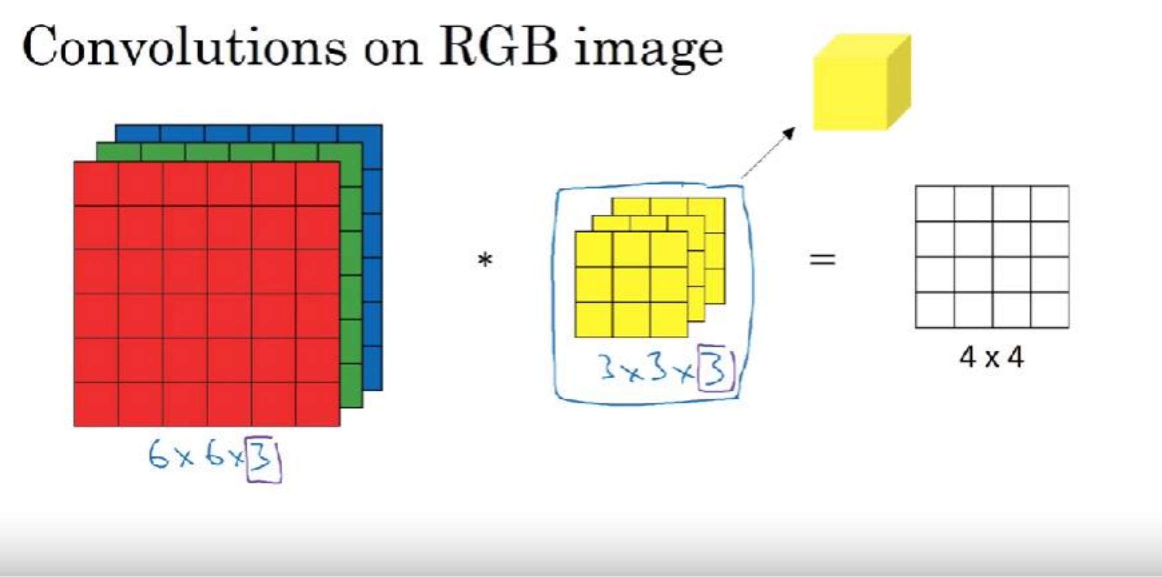
Convolutions Over Volume:

It is commonly known as 3D convolution or volumetric convolution and is an extension of the concept of convolution from 2D images to 3D volumes. It’s widely used in various applications such as video processing, medical imaging, and 3D object detection.

1)Input Volume: Just like in 2D convolution where we work with images, in convolution over volume, we deal with 3D volumes. Imagine these volumes as stacks of images or frames forming a sequence. Each element in this volume represents information across three dimensions: height, width, and depth (or channels).

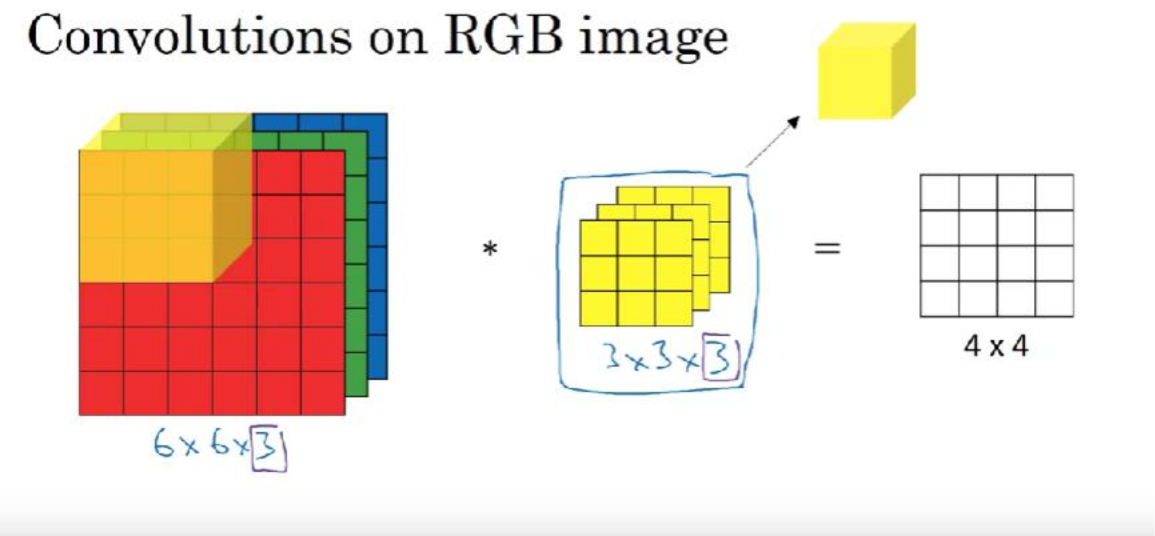


Convolutions on RGB images



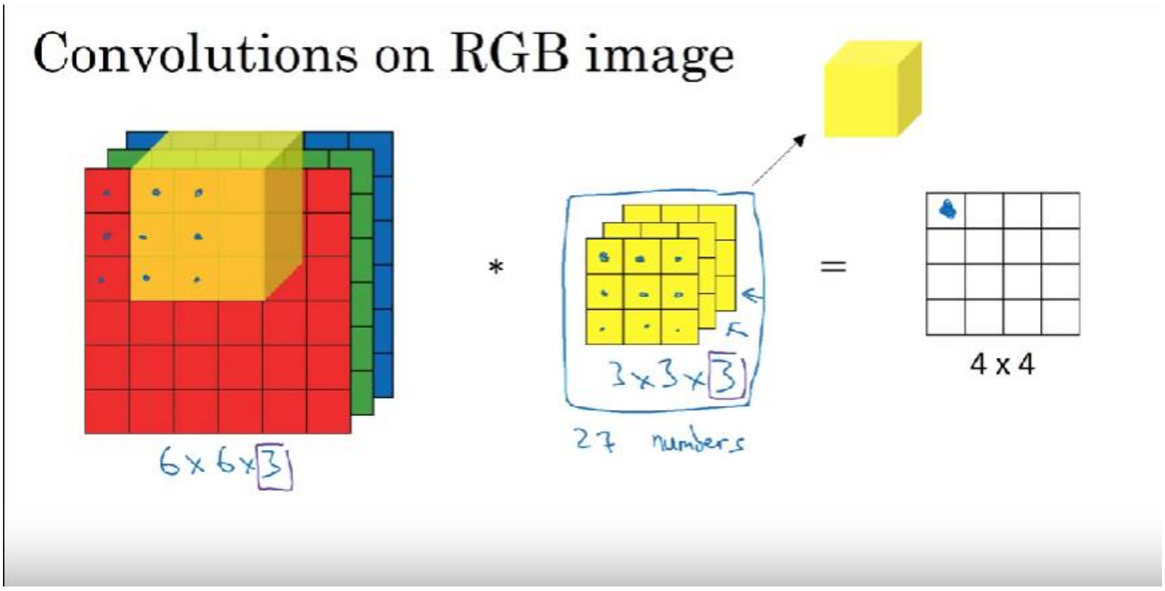
Convolutions on RGB image

2)Convolution Kernel: Similar to 2D convolution, we use convolution kernels (or filters) to extract features from the input volume. These kernels are 3D structures with learnable weights.



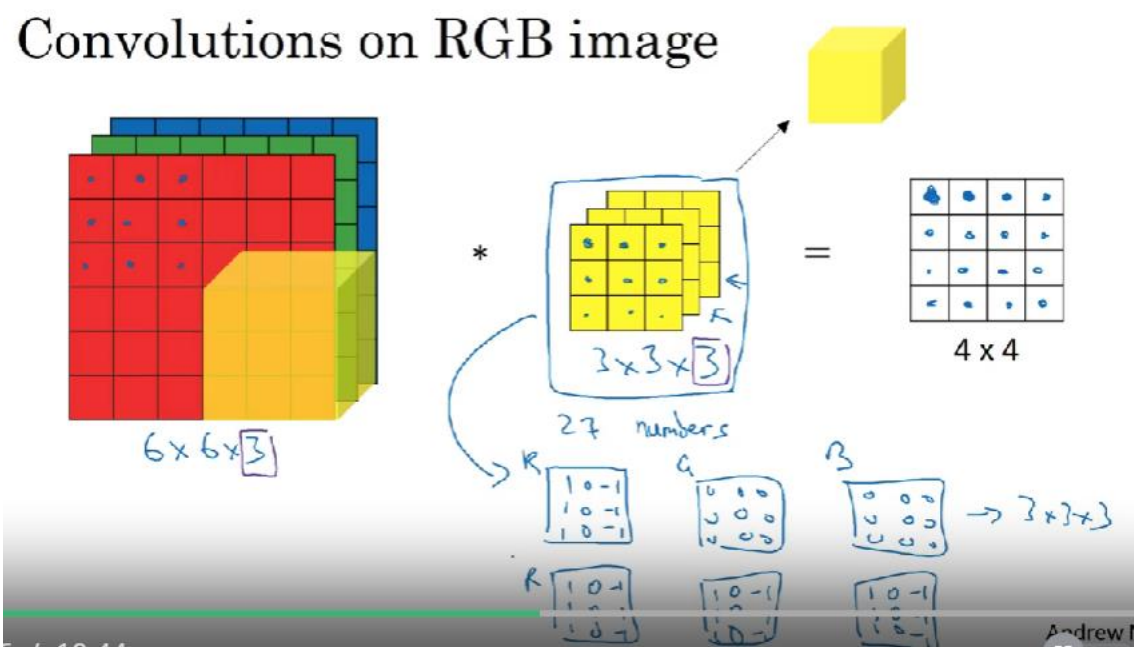
Convolutions on RGB image

3)Convolution Operation: The convolution operation involves sliding the 3D kernel across the input volume. At each position, the kernel convolves with a local region of the input volume, computing element-wise multiplications and then summing them up to produce a single output value.

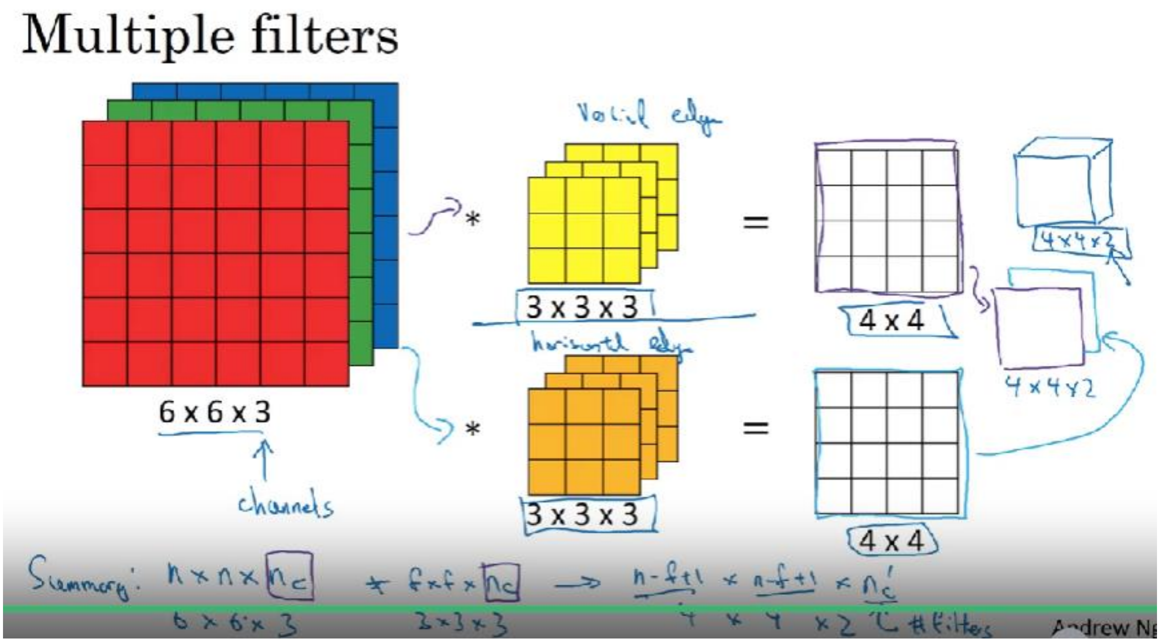


Convolutions on RGB image

4)Strides and Padding: It determines how much the kernel moves along each dimension after each convolution operation, affecting the spatial dimensions of the output volume. Padding can be applied to control the spatial dimensions of the output volume especially at the edges.



Convolutions on RGB image

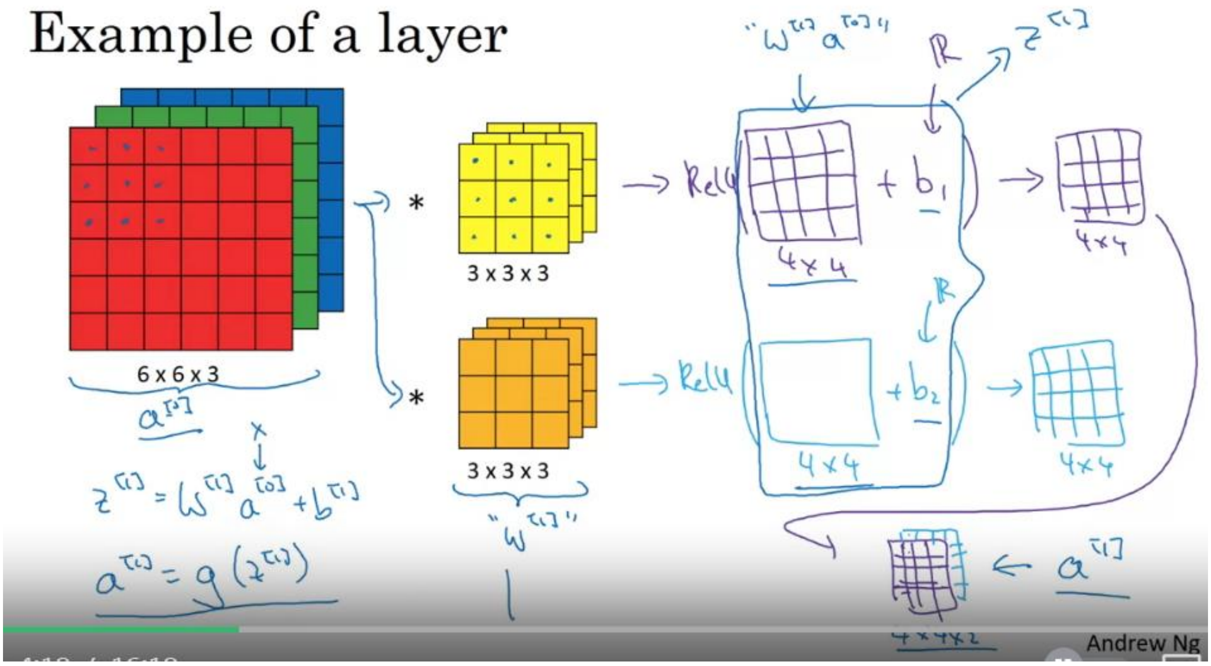


Multiple filters

Summary: $n \times n \times \boxed{n_c} * f \times f \times \boxed{n_c} \longrightarrow \underline{n-f+1} \times \underline{n-f+1} \times \underline{n_c'}$

$6 \times 6 \times 3 \qquad 3 \times 3 \times 3 \qquad 4 \times 4 \times 2$ ← #filters

Andrew Ng

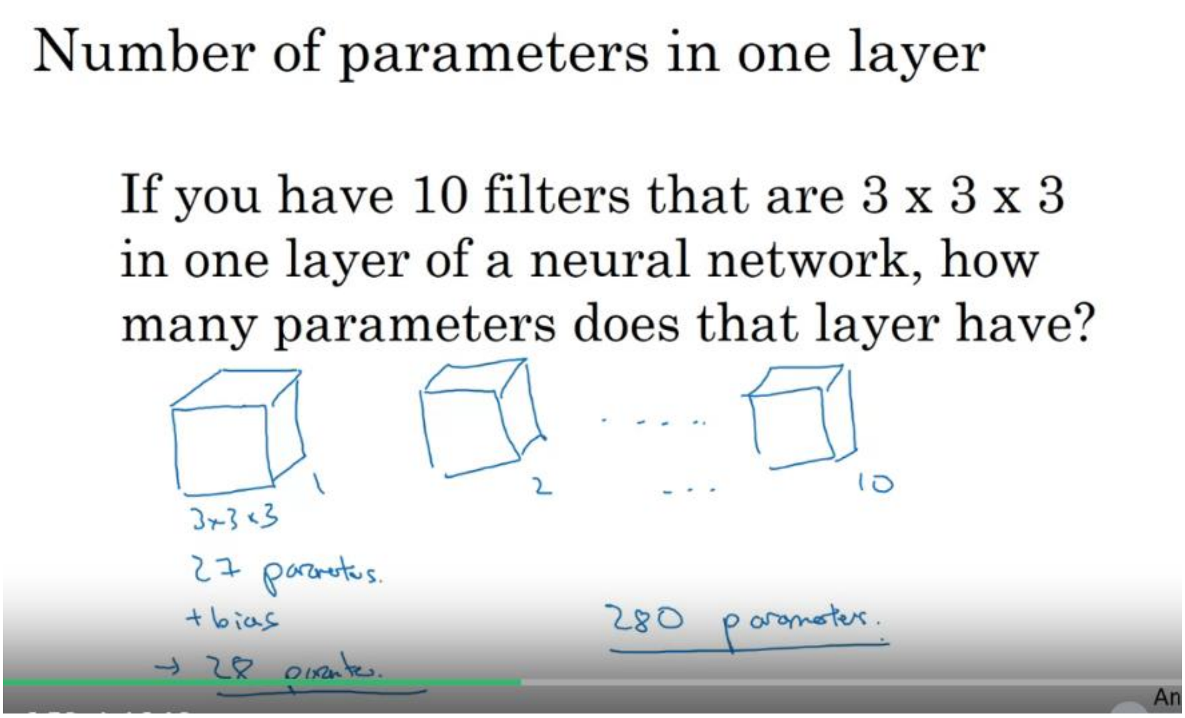


Example of a layer

$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = g(z^{[1]})$

Andrew Ng



Number of parameters in one layer

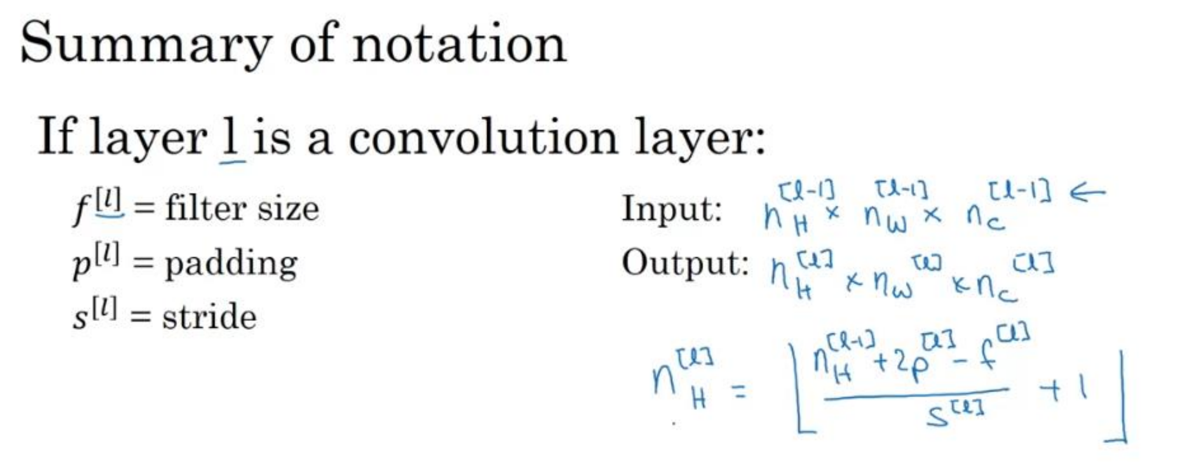
If you have 10 filters that are 3 x 3 x 3 in one layer of a neural network, how many parameters does that layer have?

27 parameters.

+bias

→ 28 parameters.

280 parameters.



Summary of notation

If layer $\underline{l}$ is a convolution layer:

$f^{\underline{[l]}}$ = filter size

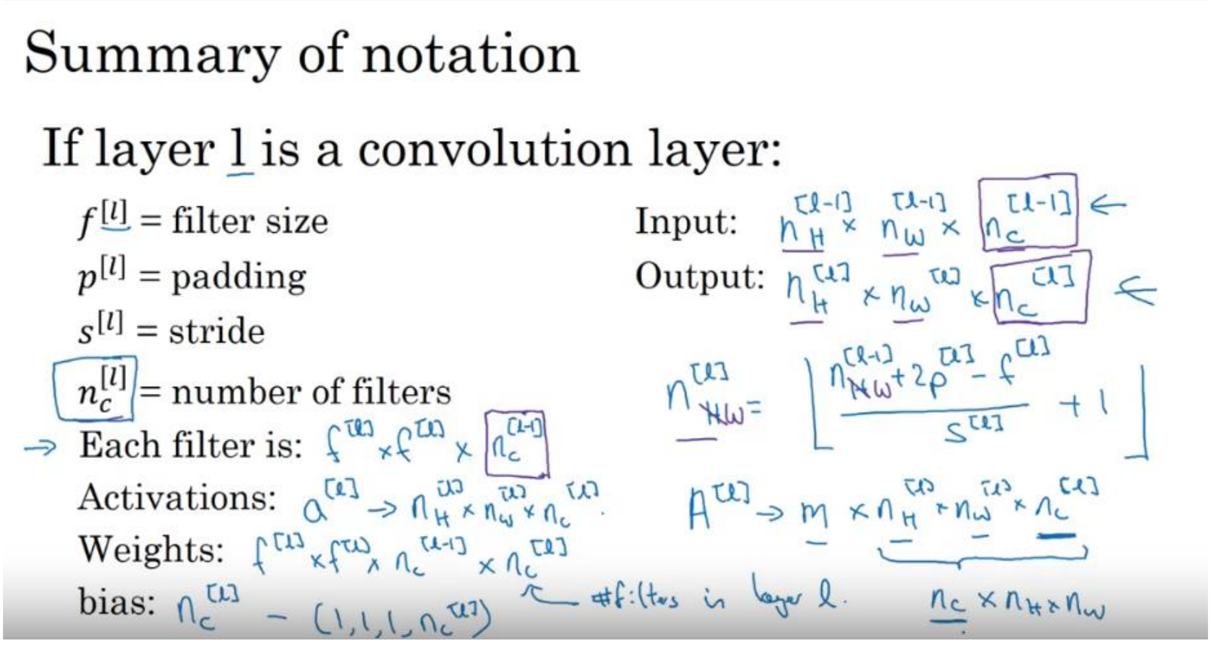
$p^{[l]}$ = padding

$s^{[l]}$ = stride

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$ ←

Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$$n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$



Summary of notation

If layer $\underline{l}$ is a convolution layer:

$f^{\underline{[l]}}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$\boxed{n_c^{[l]}}$ = number of filters

→ Each filter is: $f^{[l]} \times f^{[l]} \times \boxed{n_c^{[l-1]}}$

Activations: $a^{[l]} \to n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$ ↖ #filters in layer l.

bias: $n_c^{[l]}$ — $(1,1,1,n_c^{[l]})$

Input: $\underline{n_H}^{[l-1]} \times \underline{n_W}^{[l-1]} \times \boxed{n_c^{[l-1]}}$ ←

Output: $\underline{n_H}^{[l]} \times \underline{n_W}^{[l]} \times \boxed{n_c^{[l]}}$ ←

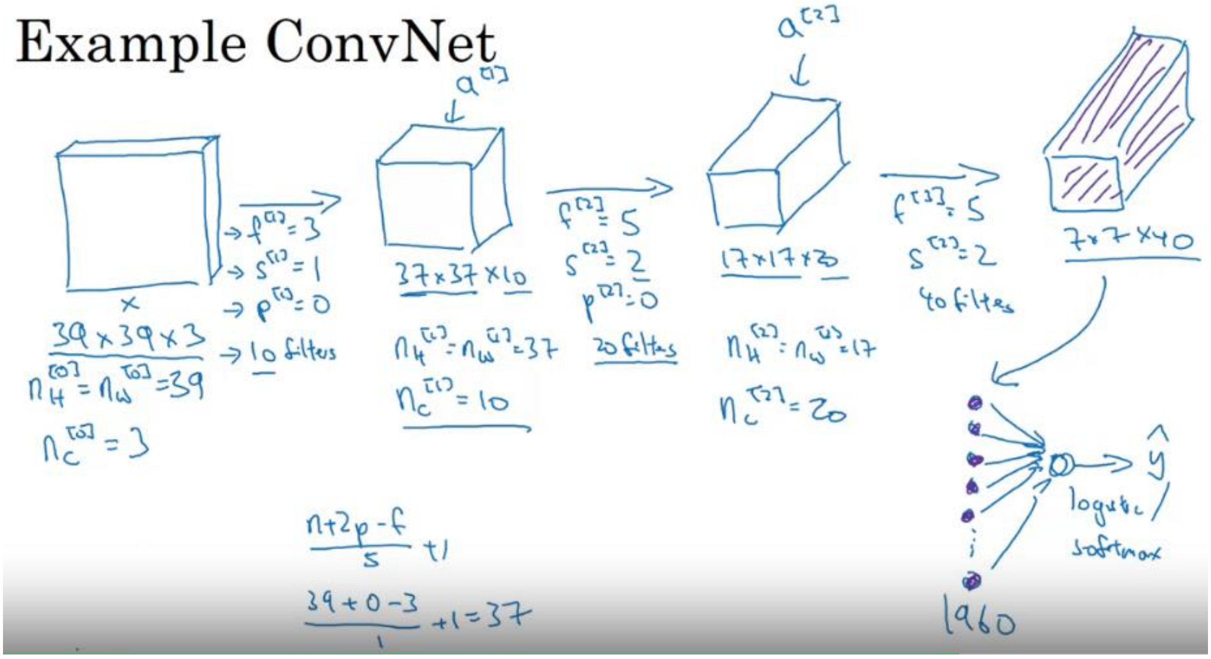
$$\underline{n_{\cancel{H}W}^{[l]}} = \left\lfloor \frac{n_{\cancel{H}W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

$A^{[l]} \to \underline{m} \times \underbrace{\underline{n_H^{[l]}} \times \underline{n_W^{[l]}} \times \underline{n_c^{[l]}}}$

$\underline{n_c} \times n_H \times n_W$

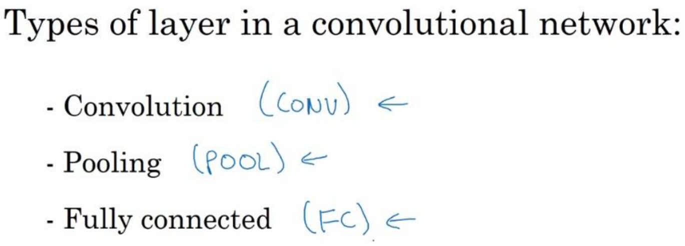
$A^{[1]}$ = batch size Activation = m x n_H x n_w x n_c

Simple Convolutional Network Example:



Example ConvNet

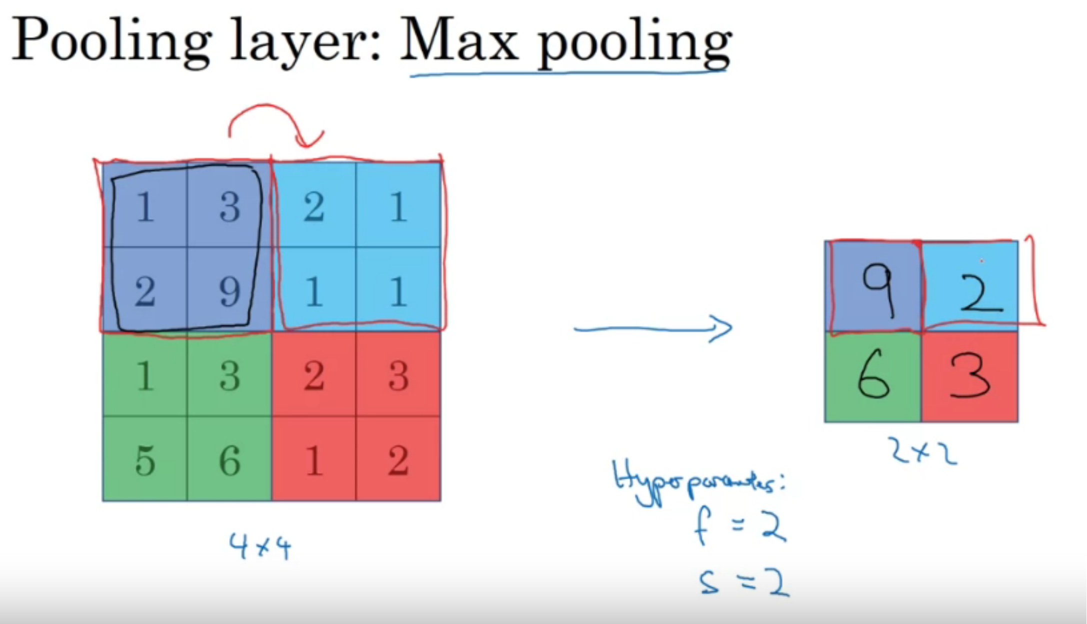
Flatten -→ 7 x 7 x 40 = 1960 features (units or neurons)



Types of layer in a convolutional network:

- Convolution (CONV) ←
- Pooling (POOL) ←
- Fully connected (FC) ←

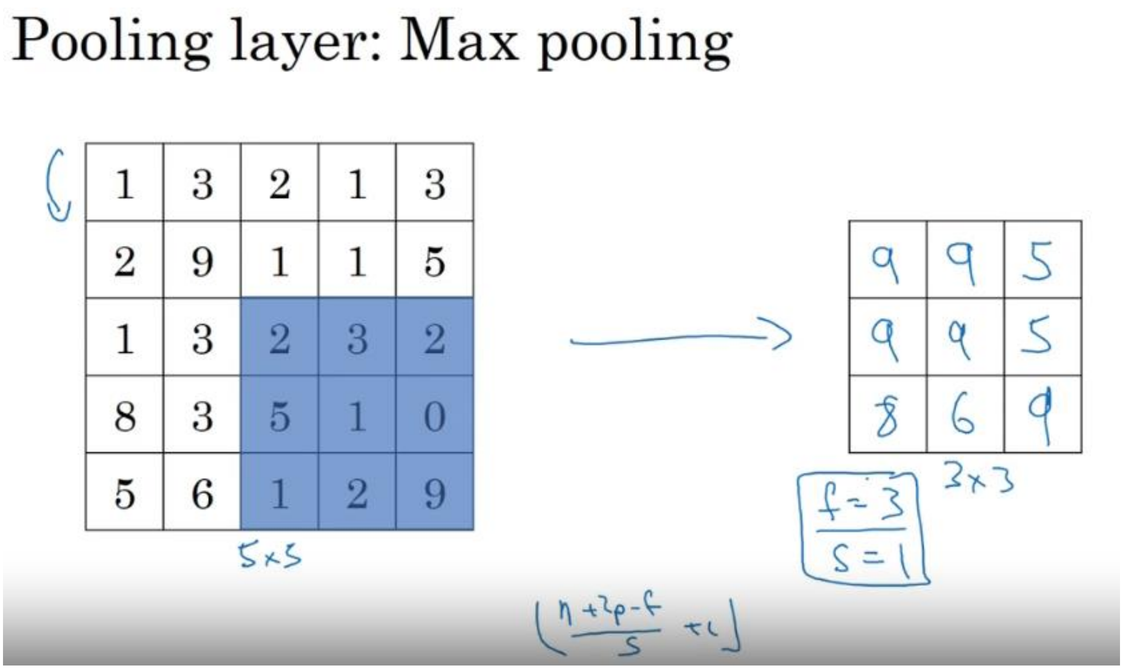
Pooling Layers:



Pooling layer: Max pooling

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

9	2
6	3



Pooling layer: Max pooling

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

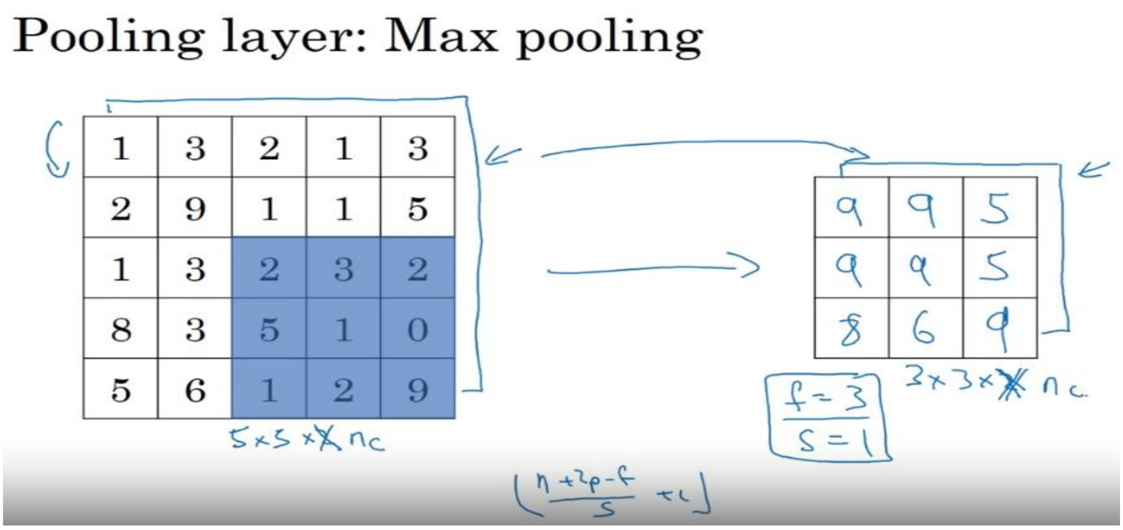
5×5

9	9	5
9	9	5
8	6	9

3×3

$\boxed{\frac{f=3}{s=1}}$

$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$



Pooling layer: Max pooling

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

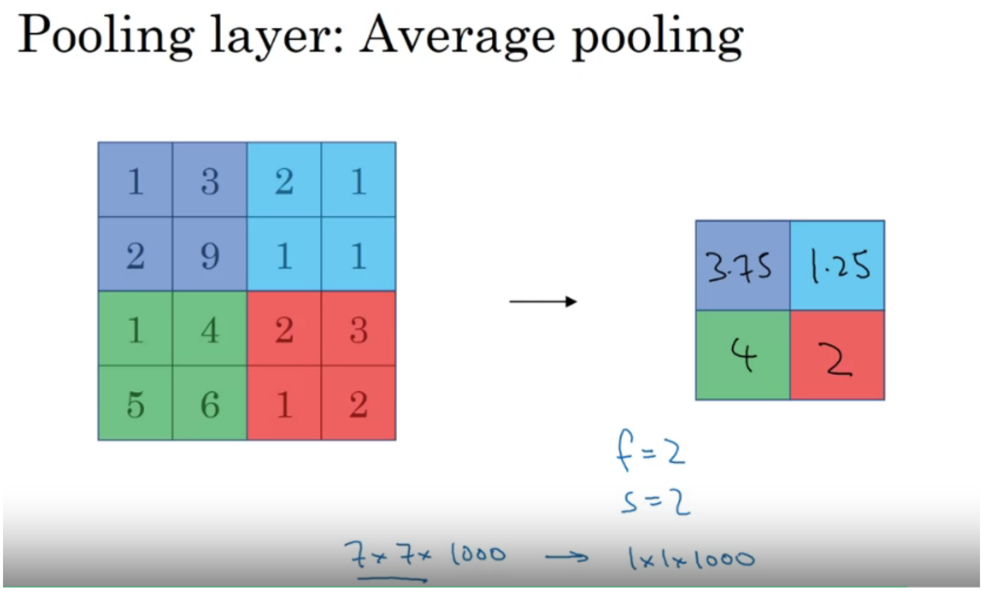
$5 \times 5 \times \cancel{2}\, n_c$

9	9	5
9	9	5
8	6	9

$3 \times 3 \times \cancel{2}\, n_c$

$\boxed{\frac{f=3}{s=1}}$

$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$



Pooling layer: Average pooling

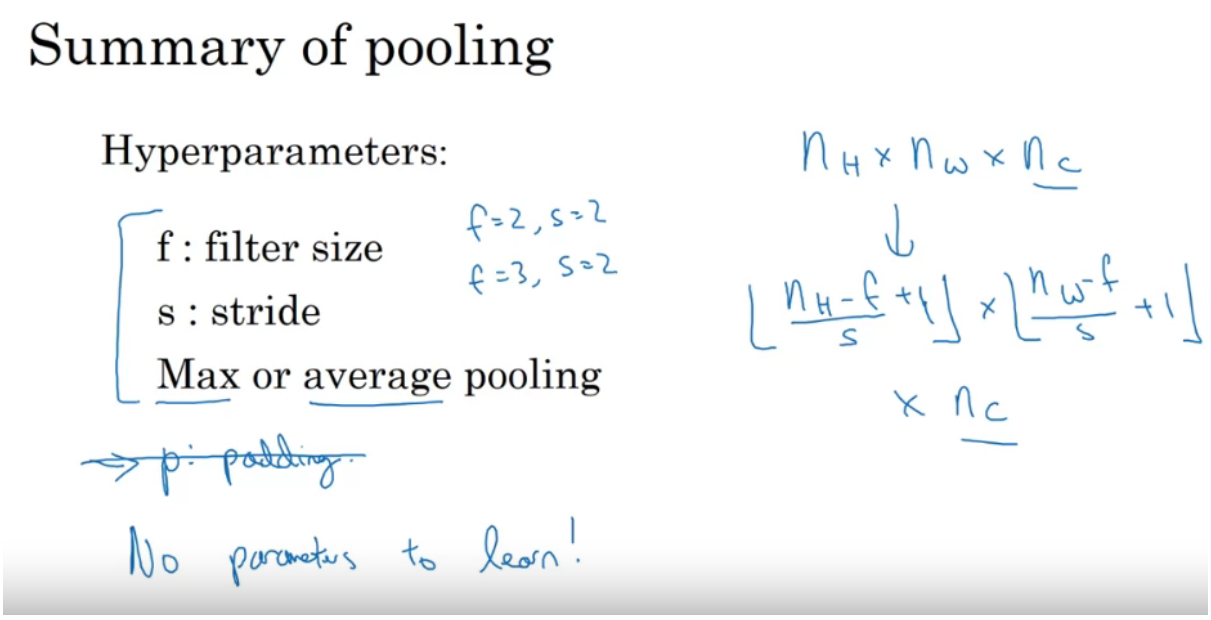
1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2

3.75	1.25
4	2

$f = 2$

$s = 2$

$\underline{7 \times 7} \times 1000 \rightarrow 1 \times 1 \times 1000$



Summary of pooling

Hyperparameters:

- f : filter size
- s : stride
- Max or average pooling

→ ~~p: padding~~

No parameters to learn!

f=2, s=2

f=3, s=2

$$n_H \times n_W \times n_C$$

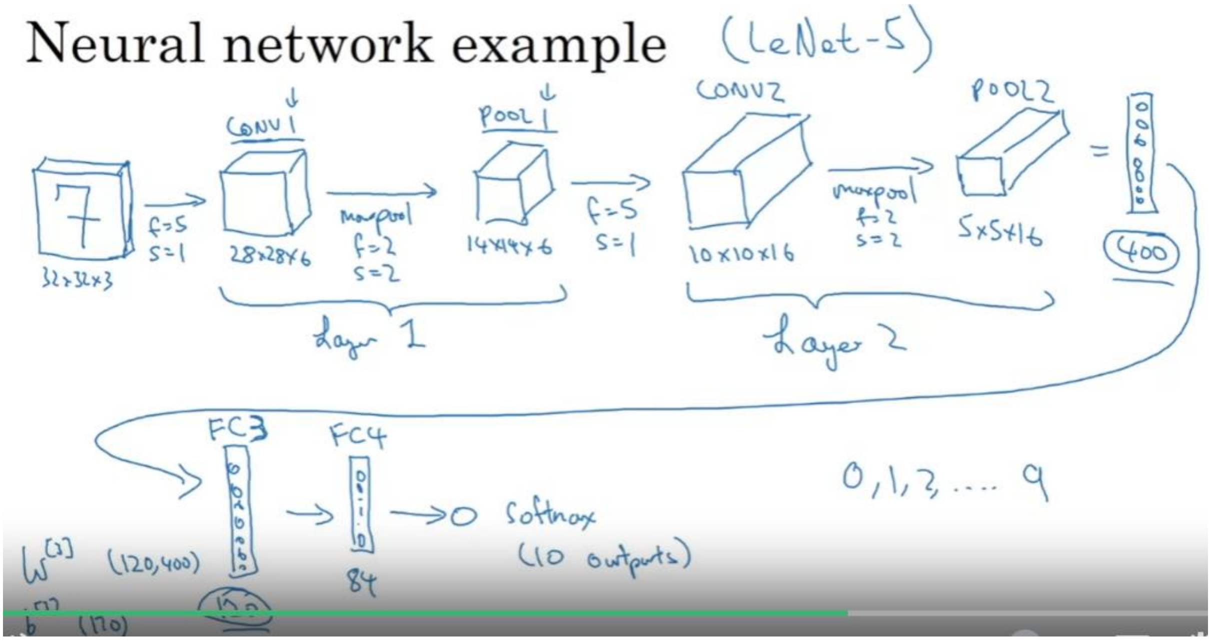
$$\downarrow$$

$$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W - f}{s} + 1 \right\rfloor \times n_C$$

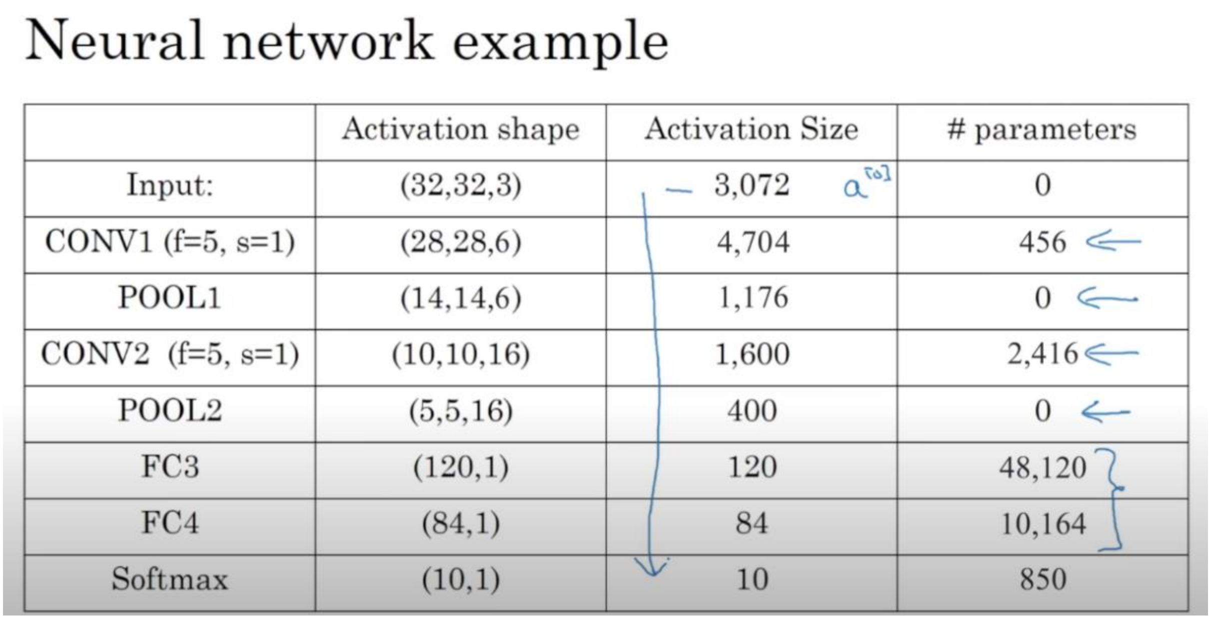
No padding is required.

f = 2, s=2 or f = 3, s = 2 are the common hyperparameters used. No extra parameters to be learned.

Neural Network Example:



Neural network example (LeNet-5)



Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 $a^{[0]}$	0
CONV1 (f=5, s=1)	(28,28,6)	4,704	456
POOL1	(14,14,6)	1,176	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	2,416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,120
FC4	(84,1)	84	10,164
Softmax	(10,1)	10	850

FC = fully connected

Parameters for POOL1 and POOL2 are 0.

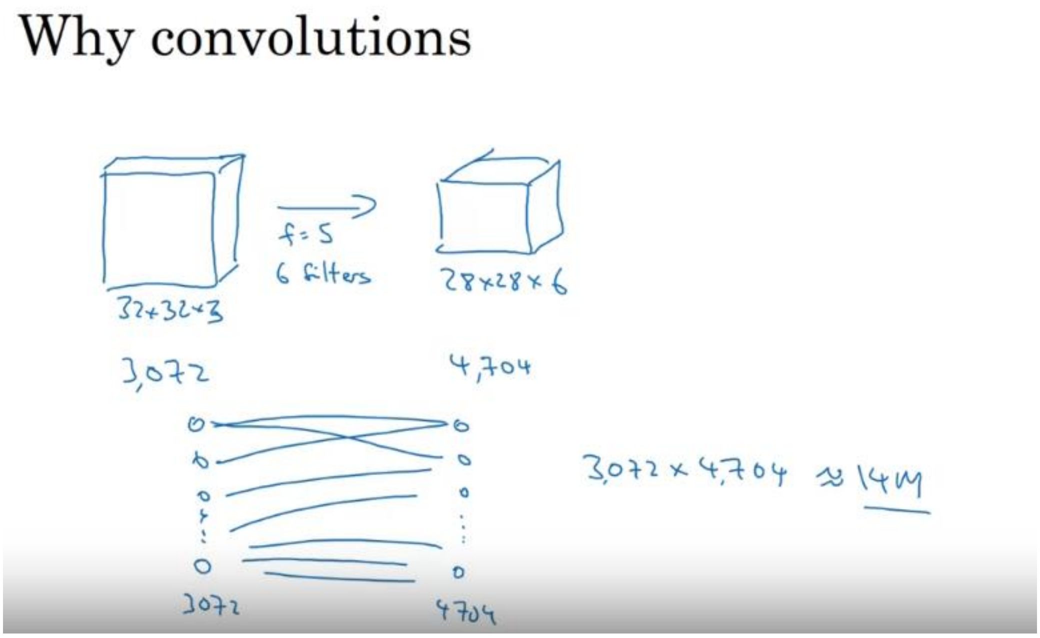
As the Activation shape decreases, the number of channels increases.

Flatten() is applied after POOl2.

Network: Input → CONV1 → POOL1 → CONV2 → POOL2 → FC → FC → softmax/logistic

Why Convolutions? Why not dense layers in CNNs?

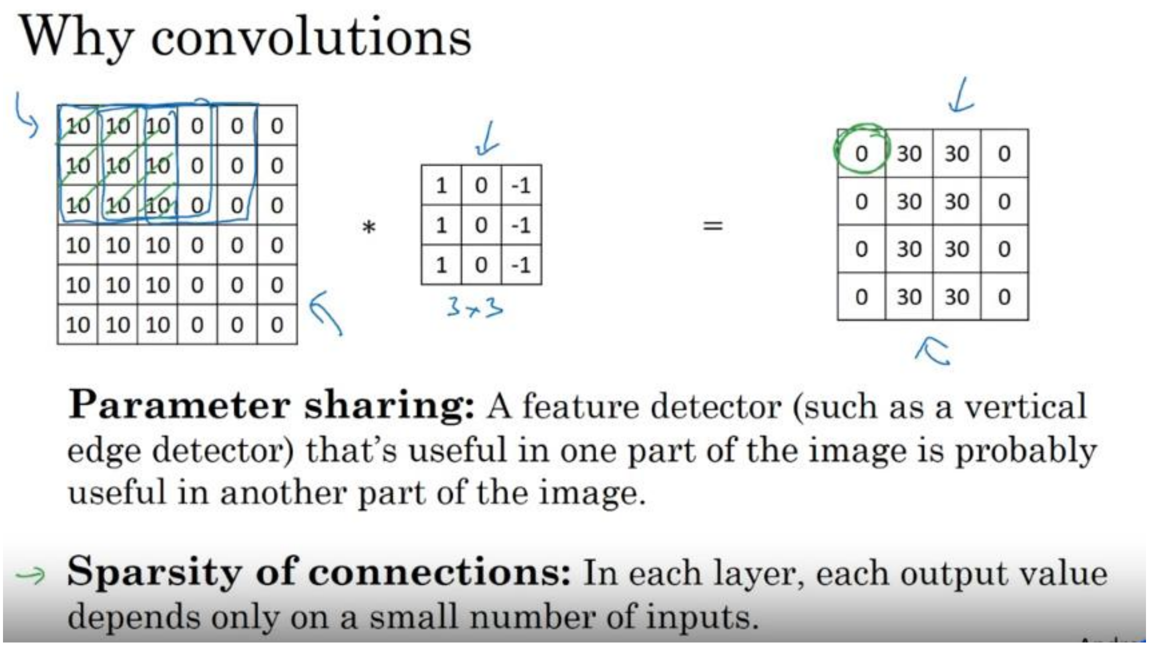
1)Local Connectivity: In images, for instance, neighbouring pixels contain related information, Convolutional filters slide across the input, capturing local patterns and preserving spatial relationships. Dense layers, on the other hand, connect every input neuron to every output neuron, ignoring spatial structure.



Why convolutions

2)Parameter Sharing: Convolutional layers share parameters across different regions of the input. Each filter is applied to the entire input using the same set of weights. This sharing reduces the number of parameters, making CNNs more efficient and less prone to overfitting, especially when data is limited.

In contrast, dense layers have separate parameters for each connection, resulting in a large number of parameters, which can lead to overfitting, especially in high-dimensional data.



Why convolutions

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{3\times3} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

→ **Sparsity of connections:** In each layer, each output value depends only on a small number of inputs.

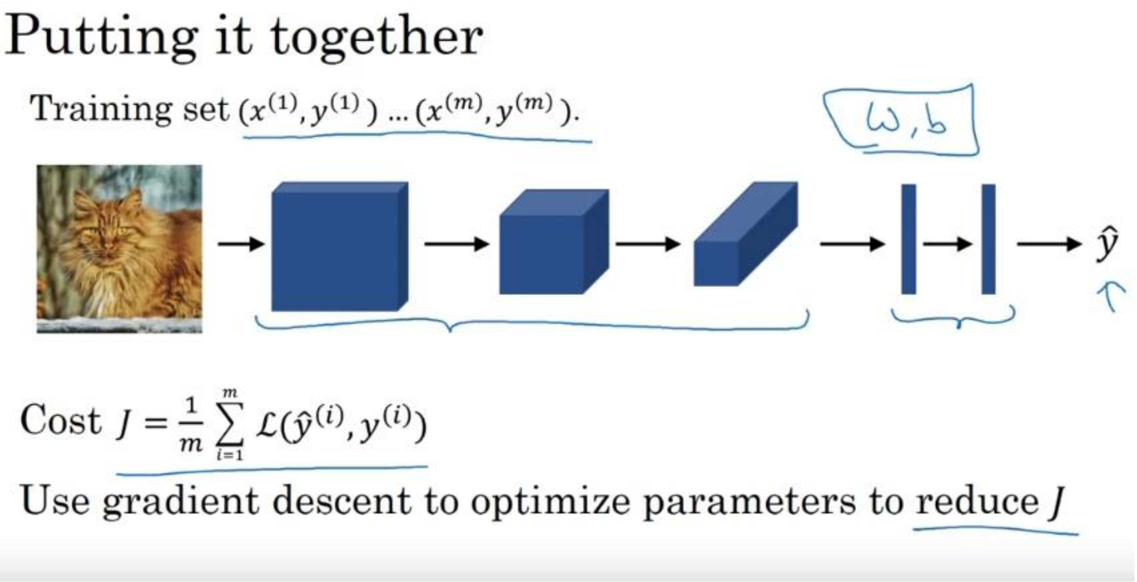
3)Translation Invariance: Convolutional layers are invariant to translations in the input data. This property is crucial for tasks like image recognition where the position of an object within the image shouldn't affect the network's ability to recognise it. Convolution filters learn to detect features regardless of their location in the input.

Dense layers, by design, lack this translation invariance since every input neuron is connected to every output neuron and the spatial arrangement matters.

4)Hierarchical Features Learning: CNNs typically consist of multiple convolutional layers followed by pooling layers, which progressively extract hierarchical features. Lower layers capture low-level features like edges and textures, while higher layers learn more abstract features and object representations.

Dense layers don't naturally capture this hierarchical structure because they treat all input features equally without considering spatial relationships.

5)Computational Efficiency: Convolutional operations are computationally more efficient than dense layers, especially for high-dimensional input data like images. By sharing parameters and local connectivity, convolutions reduce the number of computations required compared to fully connected layers, making CNNs more scalable to larger datasets and deeper architectures.



Putting it together

Training set $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$.

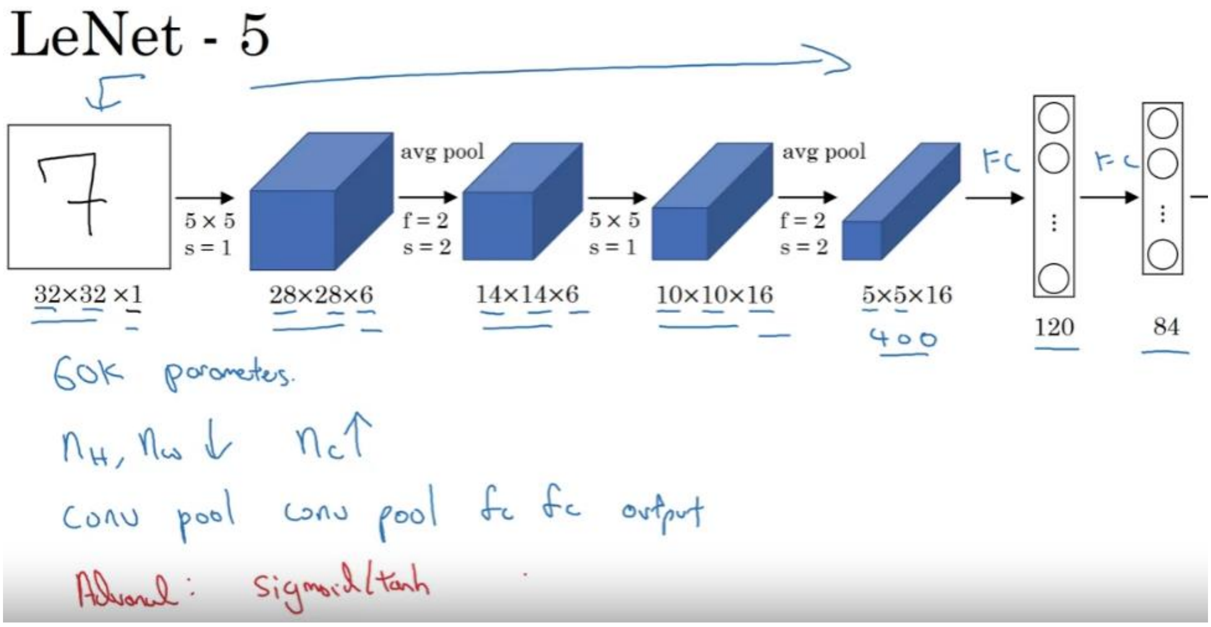
Cost $J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

Use gradient descent to optimize parameters to reduce J

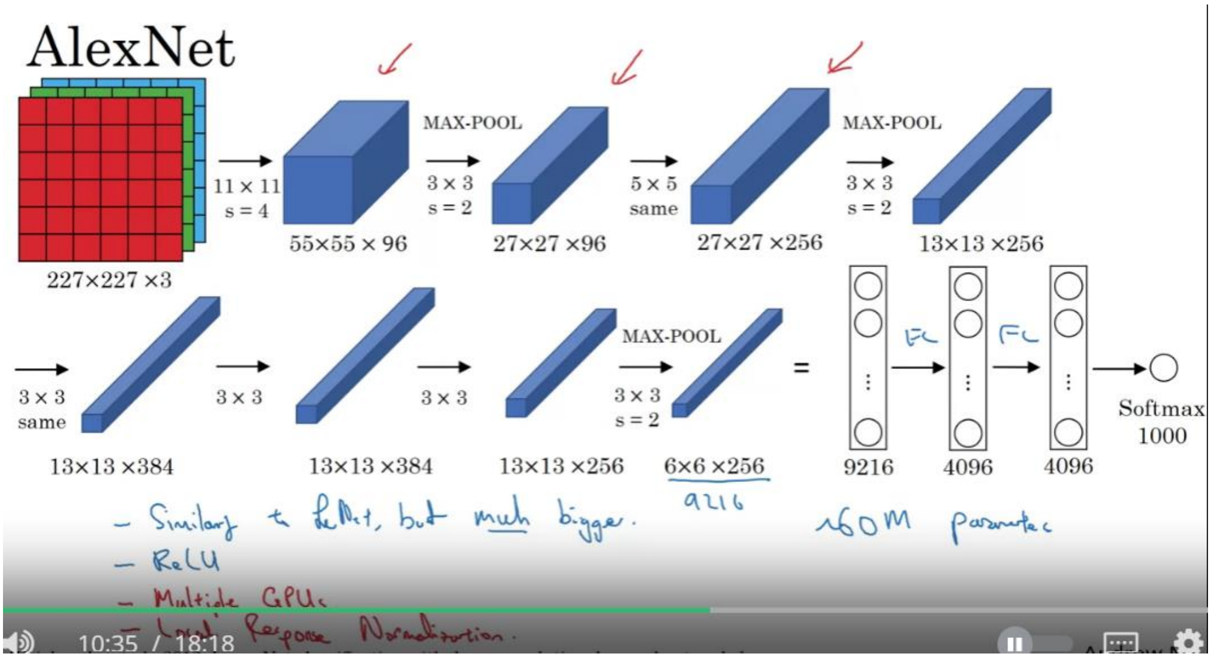
Classic Networks:

Classic networks such as LeNet-5, AlexNet, VGG-16, and VGG-19 are pioneering CNN architectures that played significant roles in advancing the field of deep learning, particularly in the domain of computer vision.

1)LeNet-5: It was one of the earliest CNN architectures designed for handwritten digit recognition. It consists of several convolutional and pooling layers followed by fully connected layers. It can be considered a form of image classification. However, its shallow architecture and limited complexity may not be well-suited for more challenging image classification tasks involving complex scenes and objects.

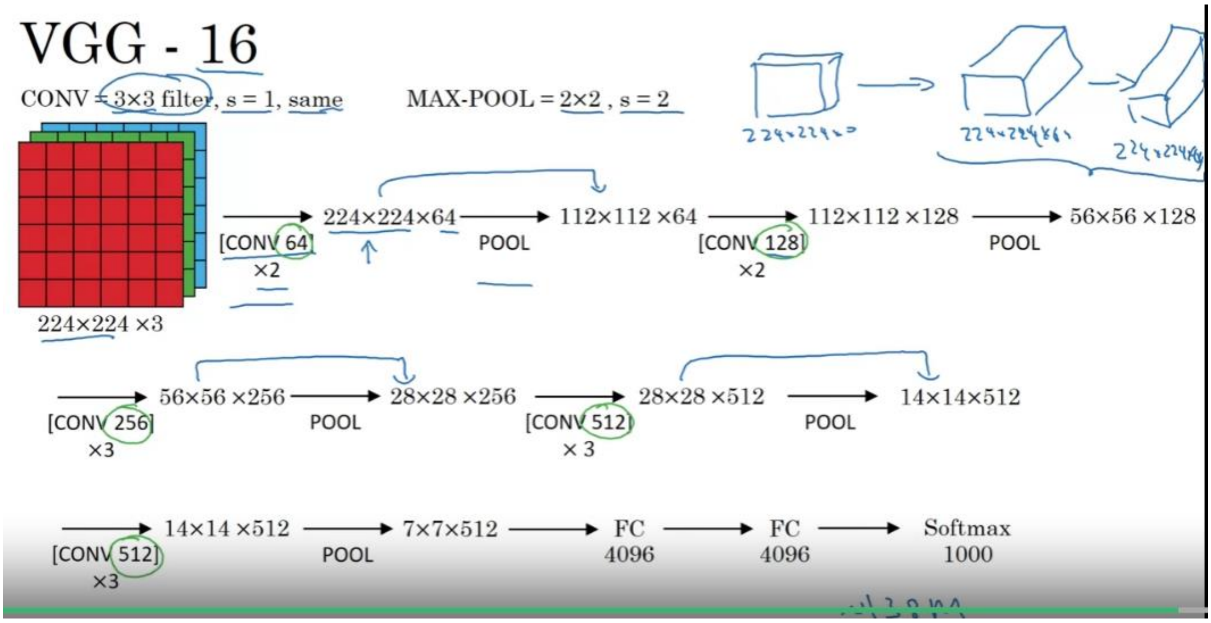


2)AlexNet: It is highly suitable for image classification tasks. Its deep architecture and effective feature-learning capabilities enable it to recognize a wide range of objects and scenes in images.



It achieved breakthrough performance on the ImageNet dataset, demonstrating its effectiveness for large-scale classification tasks.

3)VGG-16 and VGG-19 (Visual Geometry Group): Simple and uniform architecture, making it easy to understand and reproduce. Both VGG – 16 and VGG – 19 consist of multiple convolutional layers with small 3 x 3 filters, followed by max-pooling layers and fully connected layers. Achieved strong performance on image classification tasks, serving as a baseline for comparison. High computational cost due to the large number of parameters.



VGG - 16

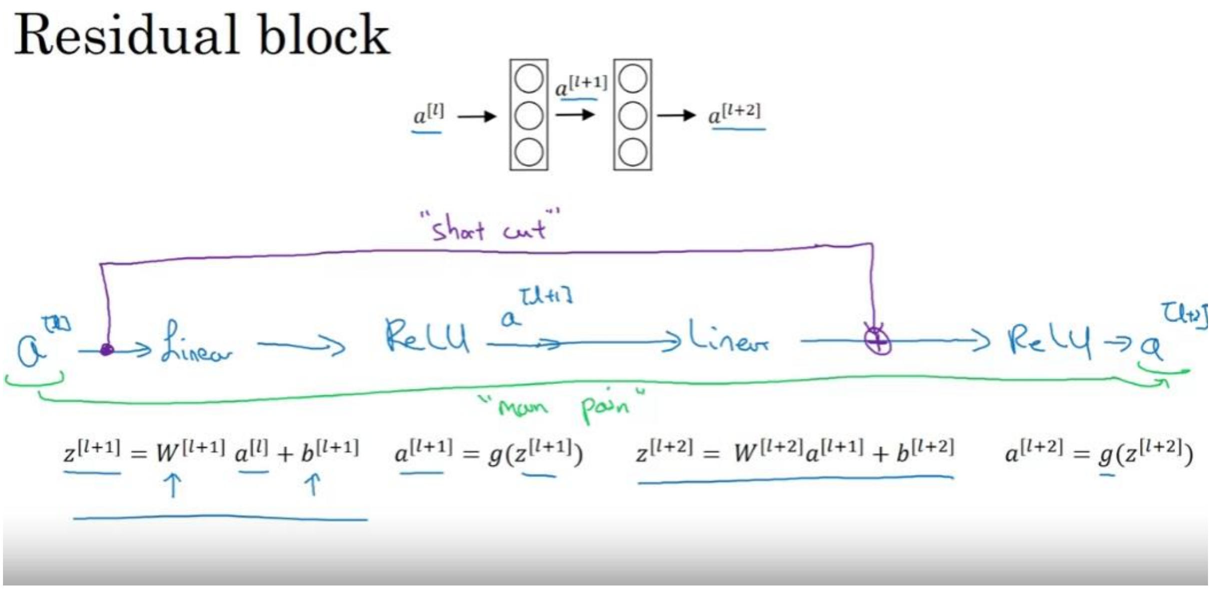
CONV = 3×3 filter, s = 1, same   MAX-POOL = 2×2 , s = 2

224×224×3 → [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] × 3 → 28×28 ×512 → POOL → 14×14×512

→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

ResNet (Residual Networks): It is a deep neural network architecture designed to address the challenge of training very deep neural networks. The main motivation behind ResNet is to overcome the problem of vanishing gradients that occurs when training deep neural networks. As a network gets deeper, it becomes increasingly difficult to train because of the vanishing gradient problem.

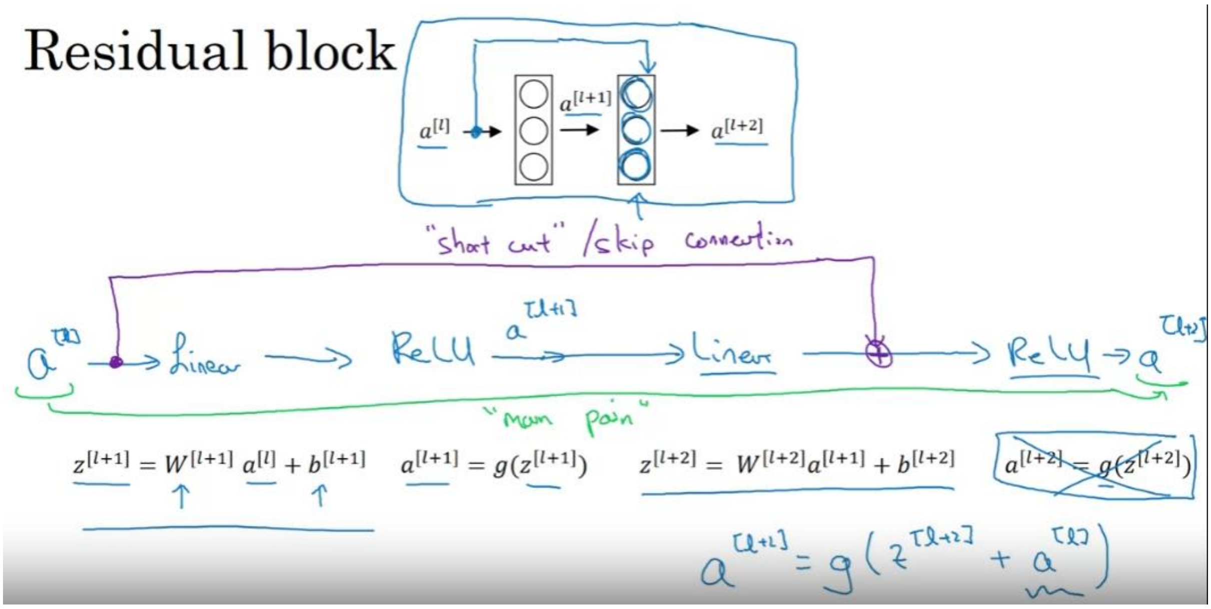


Residual block

$a^{[l]} \rightarrow \ \rightarrow a^{[l+1]} \rightarrow \ \rightarrow a^{[l+2]}$

"short cut"
$a^{[l]}$ → Linear → ReLU $a^{[l+1]}$ → Linear → ⊕ → ReLU → $a^{[l+2]}$
"main path"

$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$ $a^{[l+1]} = g(z^{[l+1]})$ $z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$ $a^{[l+2]} = g(z^{[l+2]})$



Residual block

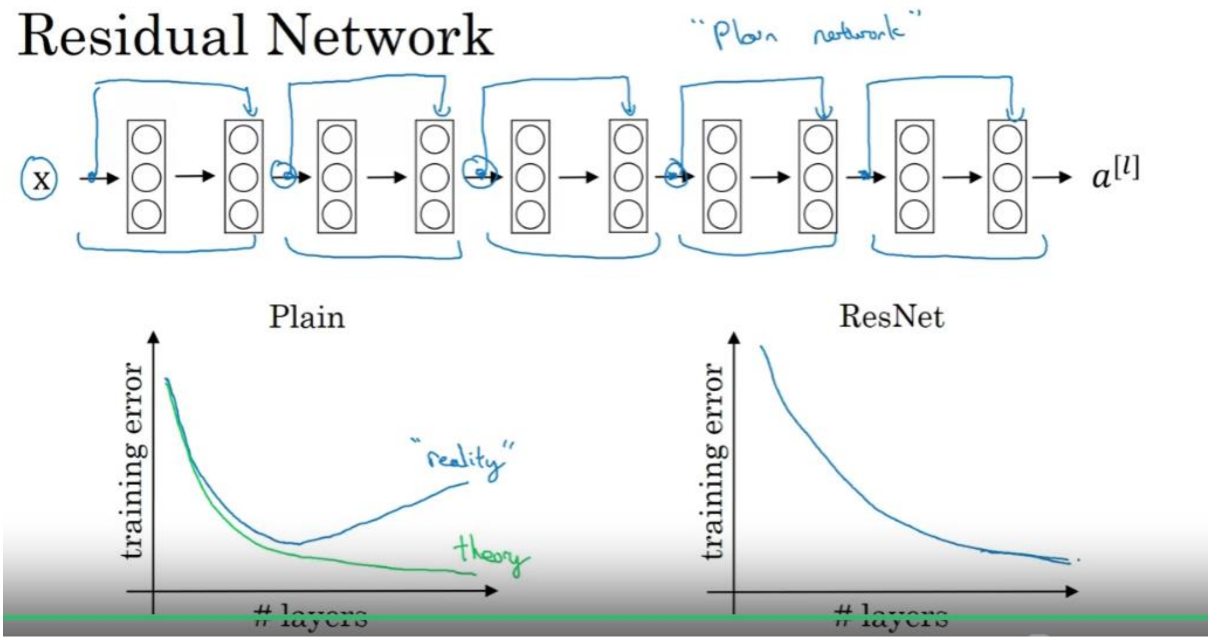
$a^{[l]} \rightarrow \ \rightarrow a^{[l+1]} \rightarrow \ \rightarrow a^{[l+2]}$

"short cut" / skip connection
$a^{[l]}$ → Linear → ReLU $a^{[l+1]}$ → Linear → ⊕ → ReLU → $a^{[l+2]}$
"main path"

$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$ $a^{[l+1]} = g(z^{[l+1]})$ $z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$ $\cancel{a^{[l+2]} = g(z^{[l+2]})}$

$$a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$$

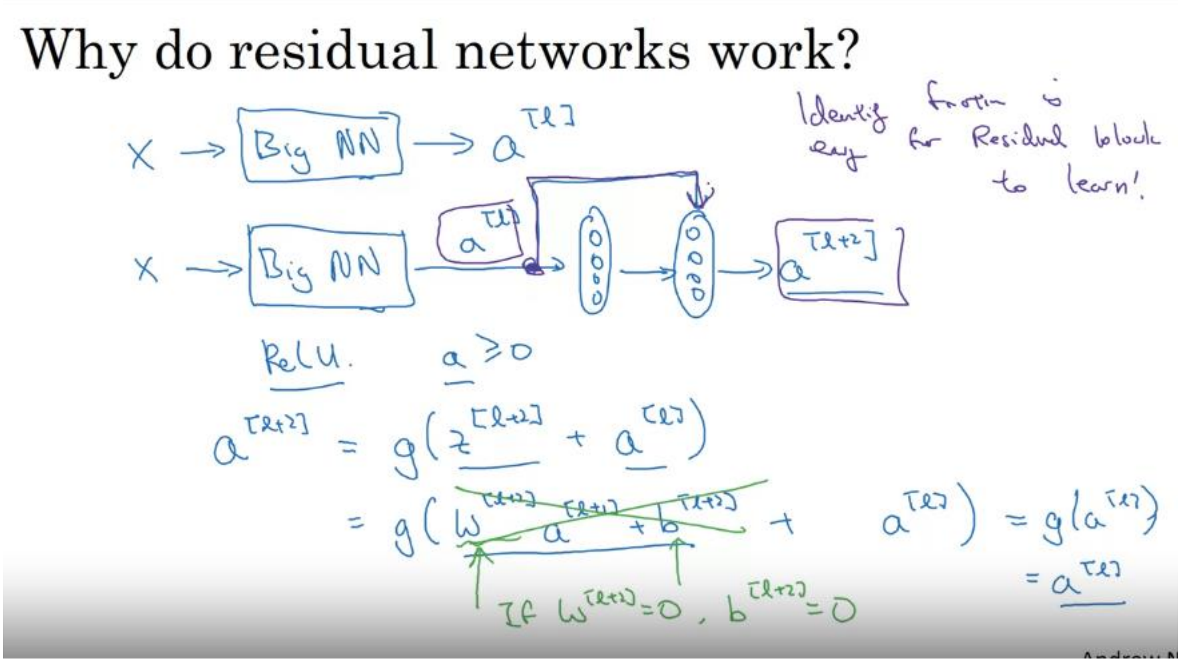
The key innovation of ResNet is the introduction of skip connections, also known as shortcut connections or identity mappings. These connections allow the gradient to bypass one or more layers, providing a direct path from input to output. This addresses the vanishing gradient problem and facilitates the training of very deep networks.



Why ResNet Work?

The basic building block of a ResNet is the residual block. There are two main types of residual blocks:

a)Identity Block: This block is used when the input and output dimensions are the same. It consists of two convolutional layers with batch normalisation and ReLU activation functions, and the input is added directly to the output. This preserves the original information without any change in dimensionality.


Why do residual networks work?

Identity function is easy for Residual block to learn!

ReLU. $a \geq 0$

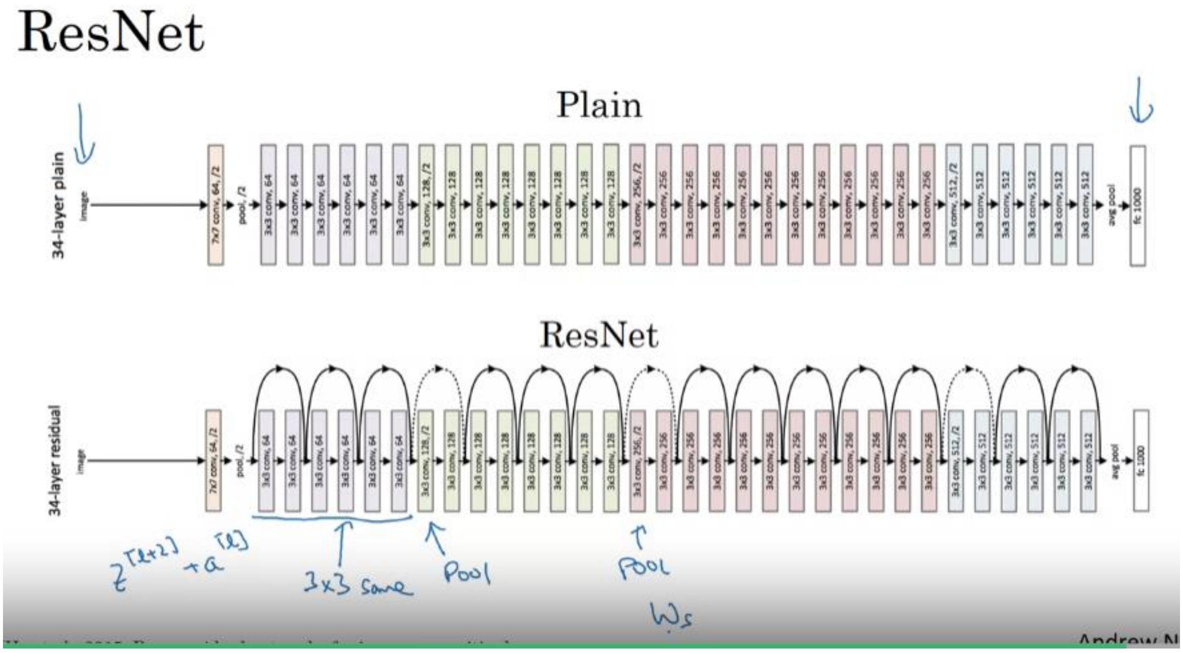
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

$$= g(\cancel{W^{[l+2]} a^{[l+1]} + b^{[l+2]}} + a^{[l]}) = g(a^{[l]}) = a^{[l]}$$

If $W^{[l+2]} = 0$, $b^{[l+2]} = 0$

b)Convolutional Block: This block is used when the input and output dimensions differ. It includes a convolutional layer with a stride of 2 to change the dimensions of the input, followed by batch normalisation and ReLU activation functions. Then, it follows with another convolutional layer to match the dimensions of the shortcut connection.

ResNet

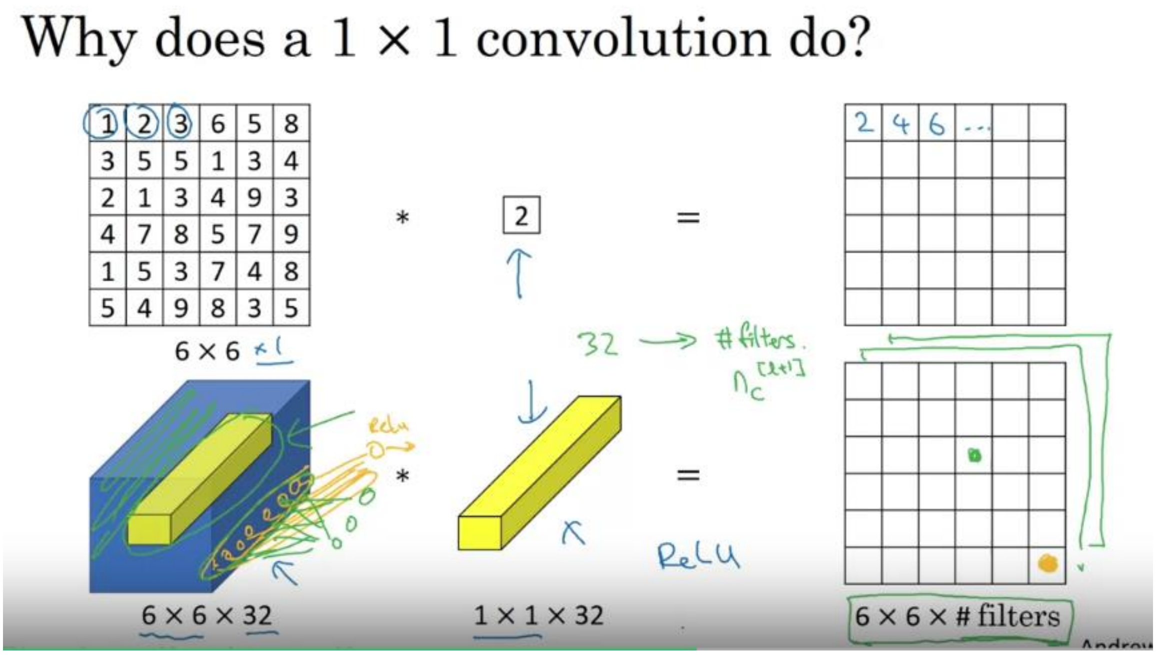


ResNet has been widely used in various computer vision tasks, including:

Image Classification, Object Detection, Semantic Segmentation, Image Super-Resolution, Facial Recognition, Medical Image Analysis, etc.

Networks in Networks and 1 x 1 Convolution: The key idea behind Network in Network is to use micro-neural networks instead of traditional linear filters in convolutional layers. In other words, instead of using a single convolutional filter to extract features, Network in Network employs small neural networks (usually consisting of a few layers) to capture more complex patterns within each receptive field with non-linearities.

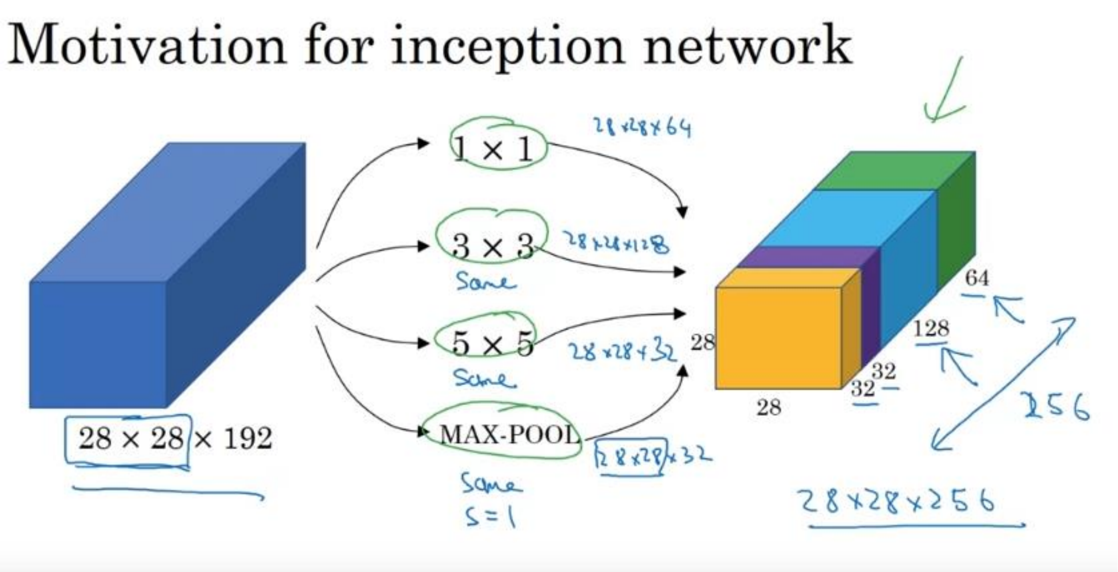
Why does a 1 × 1 convolution do?



1	2	3	6	5	8
3	5	5	1	3	4
2	1	3	4	9	3
4	7	8	5	7	9
1	5	3	7	4	8
5	4	9	8	3	5

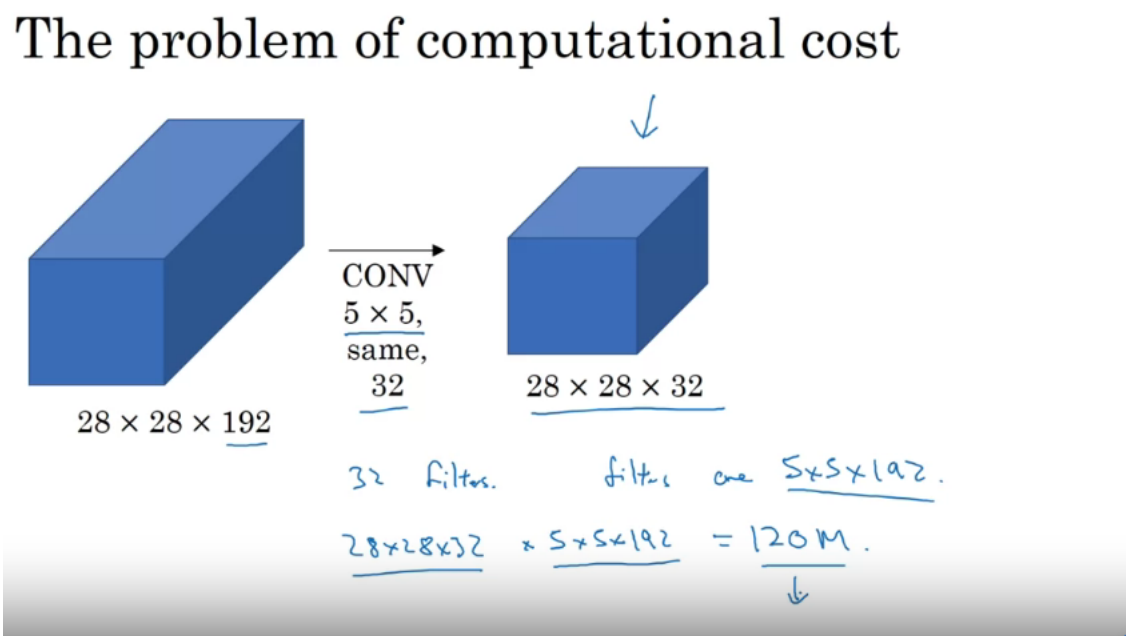
6 × 6 × 1 * 2 = 2 4 6 ...

Inception Network Motivation (GoogLeNet:) Traditional CNN architectures often become computationally expensive as the depth and width of the network increase. Inception aimed to address this issue by designing modules that can feature at multiple spatial scales while maintaining computational efficiency.



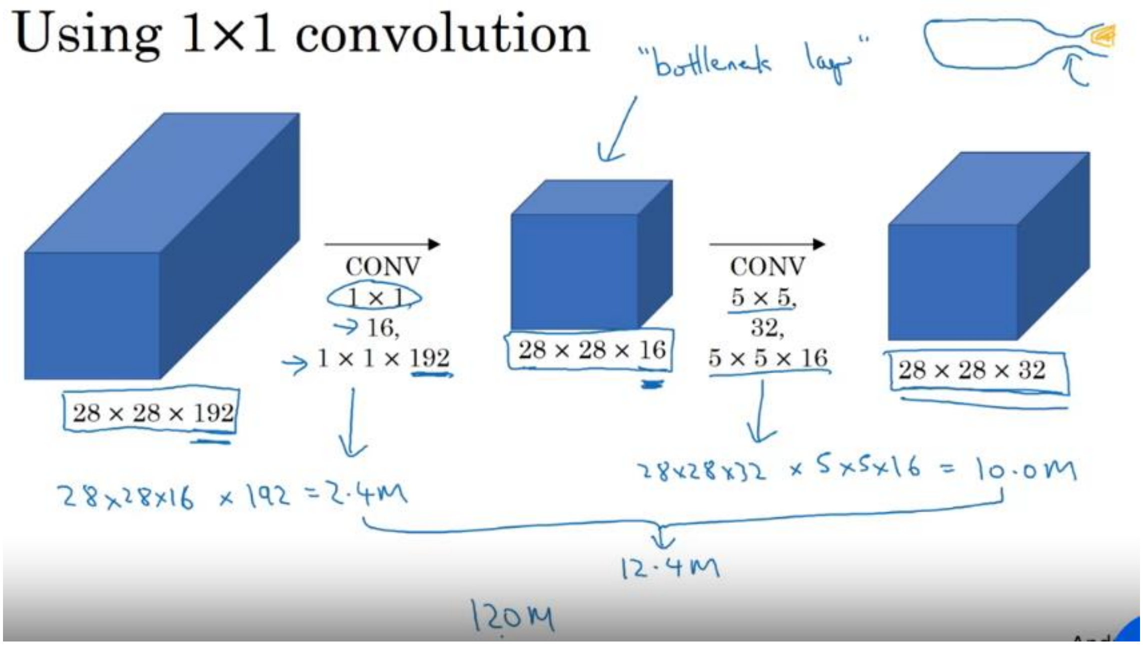
Motivation for inception network

The core idea of the Inception network is the use of the Inception module, which performs convolutions with multiple filter sizes (1 x 1, 3 x 3, 5 x 5) and pooling operations simultaneously. This allows the network to capture features at multiple spatial scales within the same layer, facilitating richer representations.

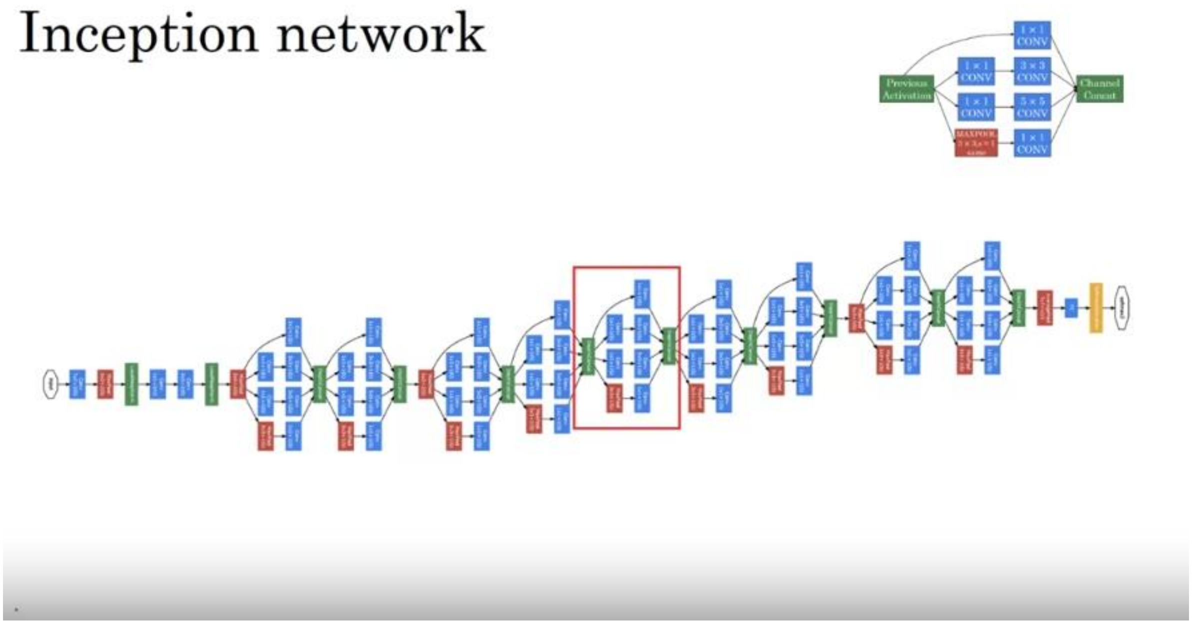


The problem of computational cost

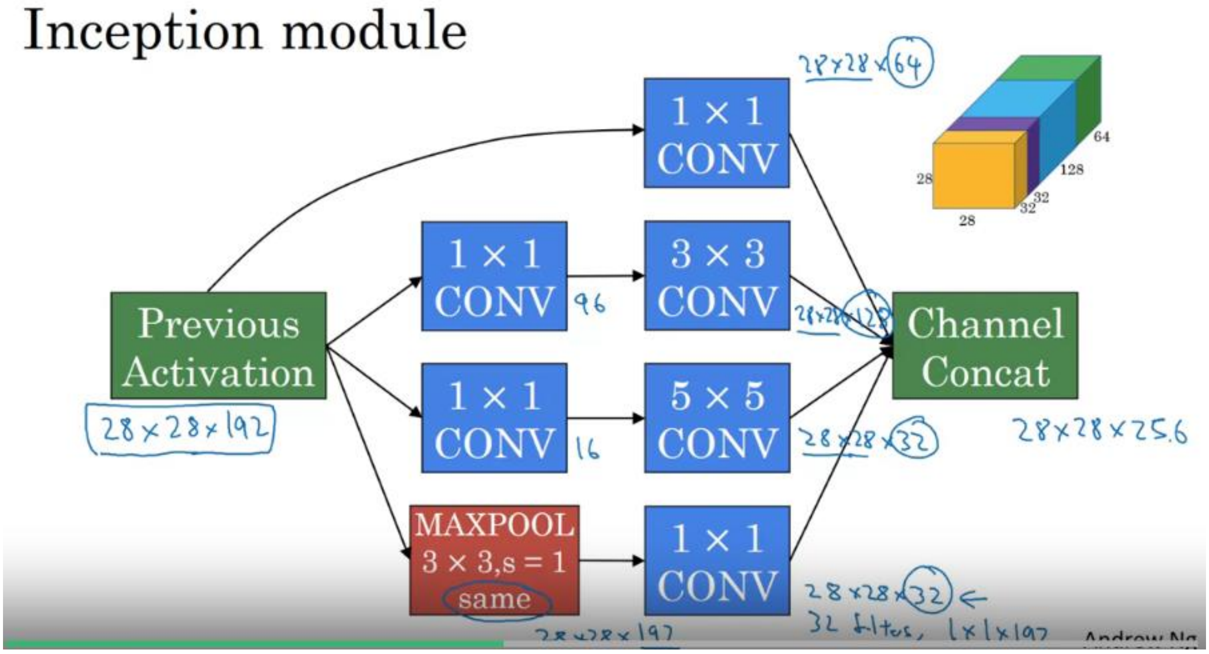
However, the computational costs increase, to manage the computational complexity 1 x 1 (NiN) convolutions are employed for dimensionality reductions before applying larger convolutions which reduces computational complexity.



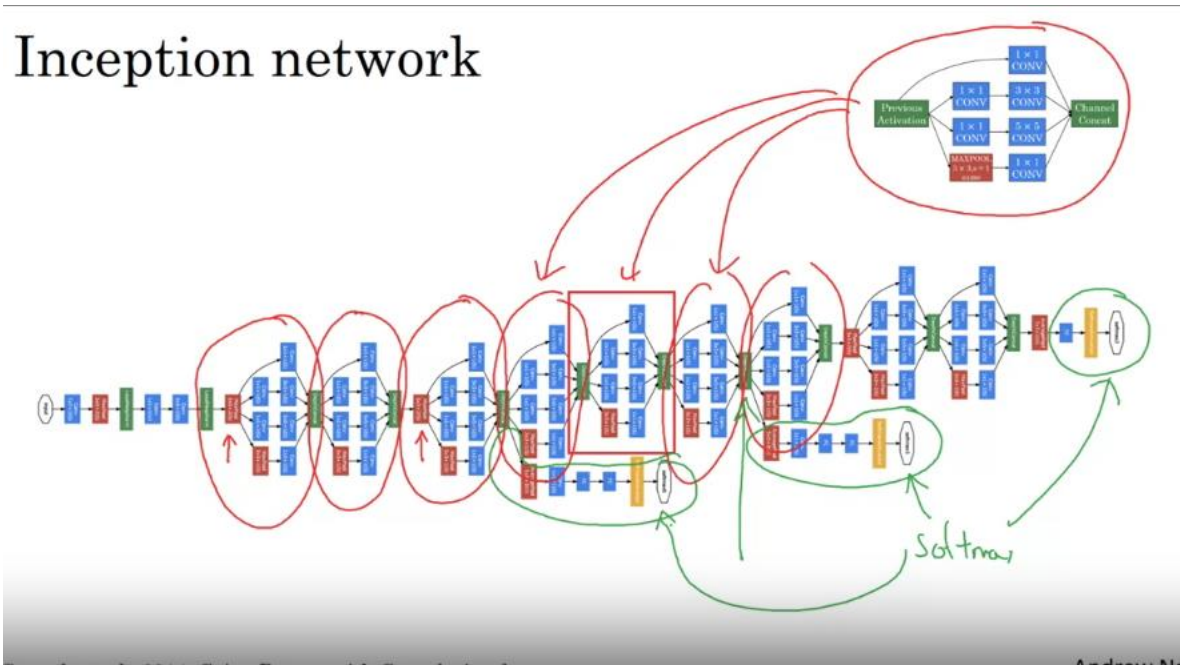
Using 1×1 convolution



Inception network



Inception module



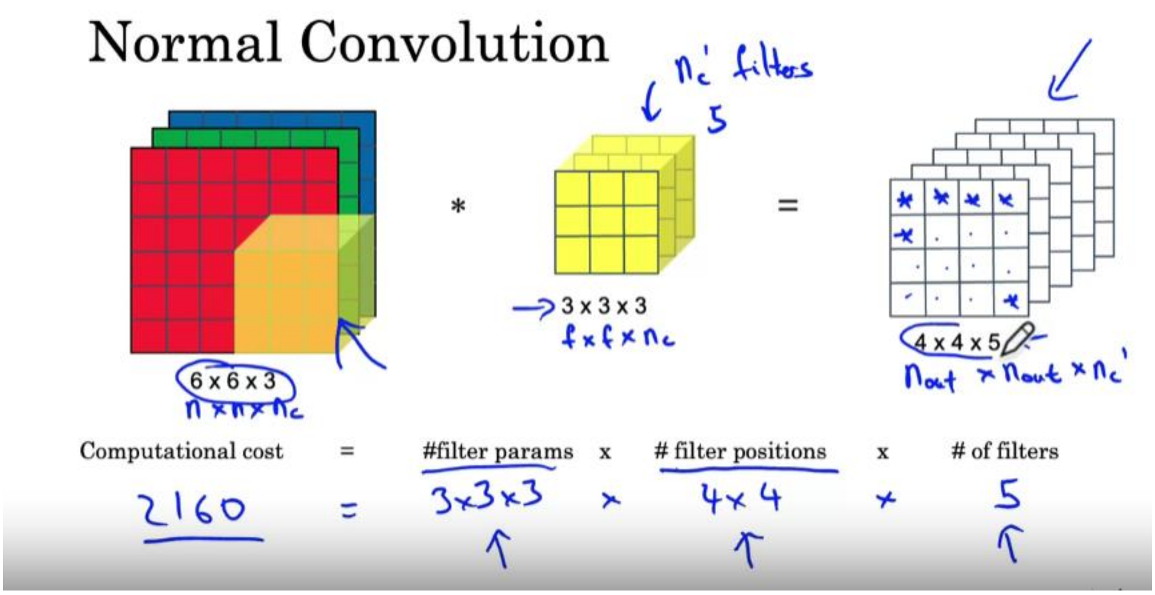
Inception network

(Inception)

Deeper → Deeper → Deeper → Deeper → Deeper → Deeper → Deeper → Deeper → Deeper → Deeper →……..

Inception networks are designed to be relatively deep while maintaining computational efficiency. By carefully designing the inception modules and managing the network's depth, the model can learn complex hierarchical features and achieve state-of-the-art performance on image classification tasks.

MobileNet: It is a class of lightweight deep-neural network architectures specifically designed for efficient deployment on mobile and embedded devices with limited computational resources. Developed by researchers at Google used for image classification, object detection, and semantic segmentation while minimizing model size and computational complexity.

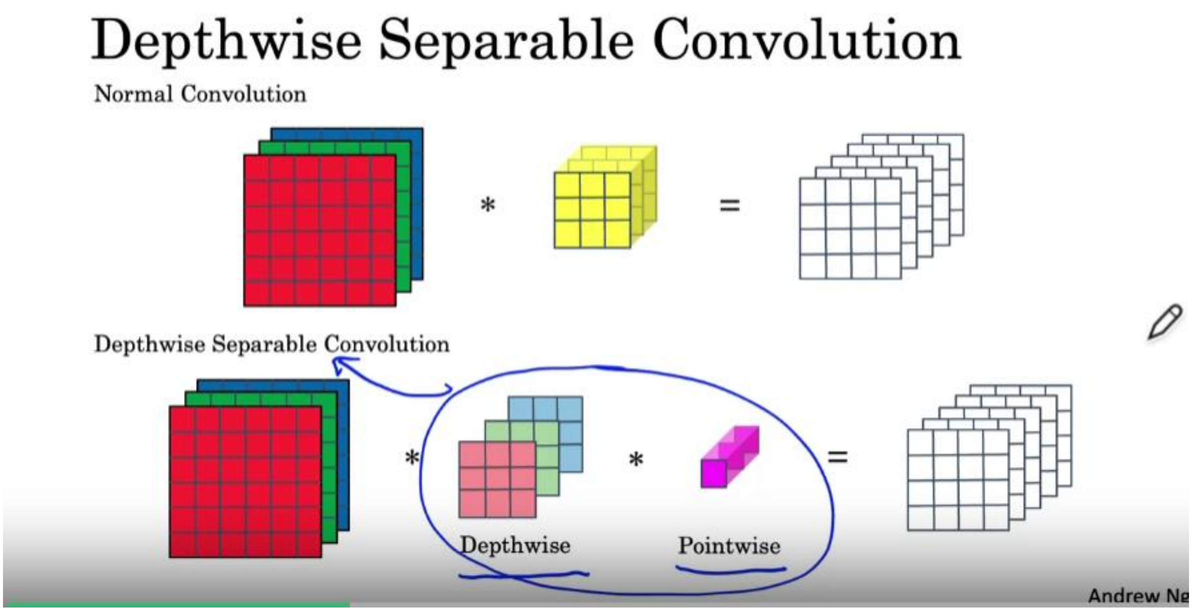


Normal Convolution

Computational cost = #filter params x # filter positions x # of filters

2160 = 3x3x3 x 4x4 x 5

1. Depth-wise Separable Convolution: It is the depth-wise separable convolution operation, which decomposes the standard convolution into separate operations: depth-wise convolution and pointwise convolution.

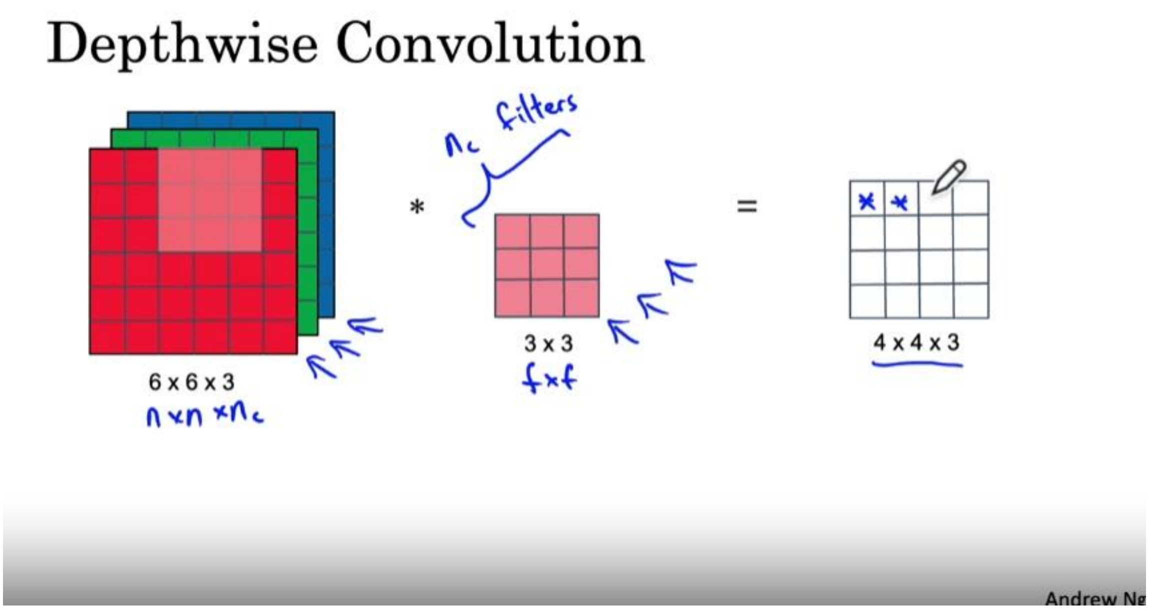


Depthwise Separable Convolution

Normal Convolution

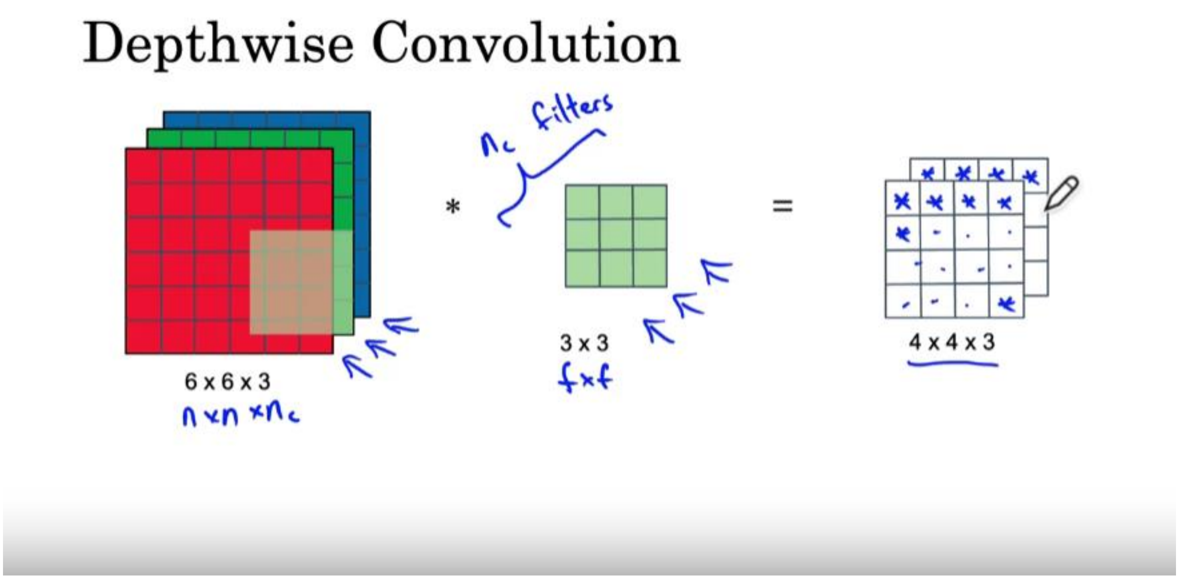
Depthwise Separable Convolution

a. Depth-wise Convolution: Applies a single convolutional filter per input channel, capturing spatial information independently for each channel.

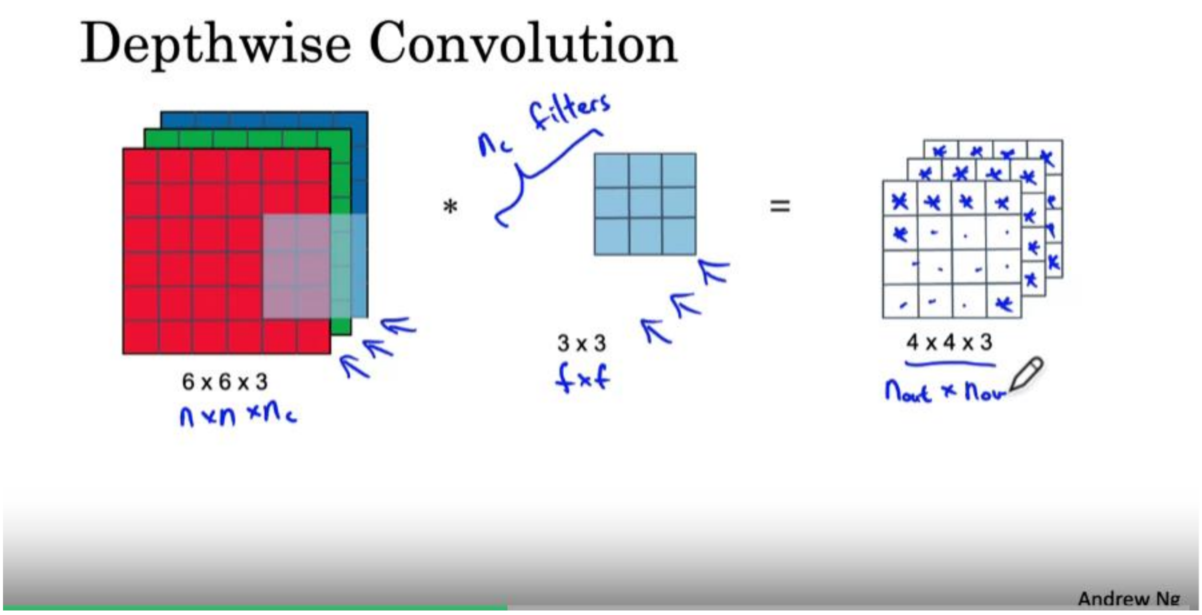


Depthwise Convolution

Andrew Ng

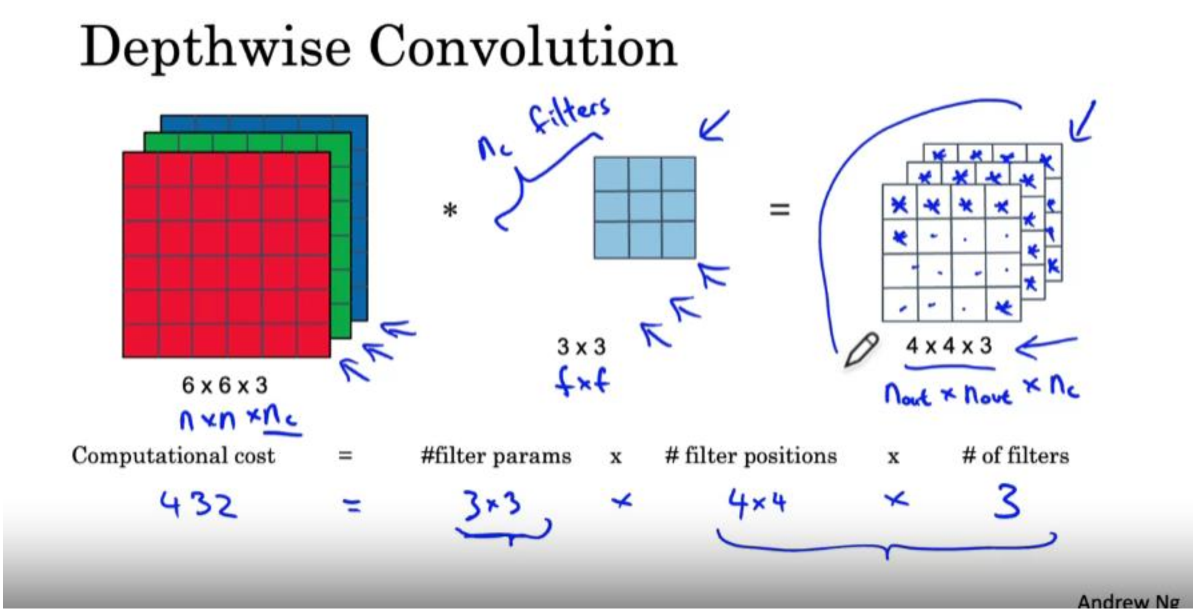


Depthwise Convolution



Depthwise Convolution

Andrew Ng

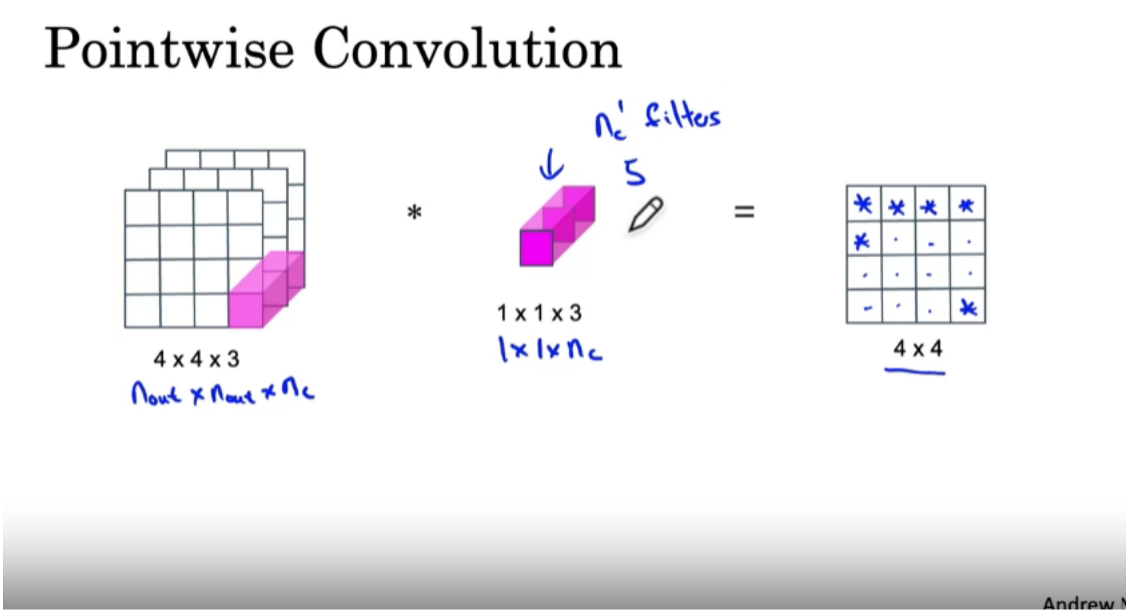


Depthwise Convolution

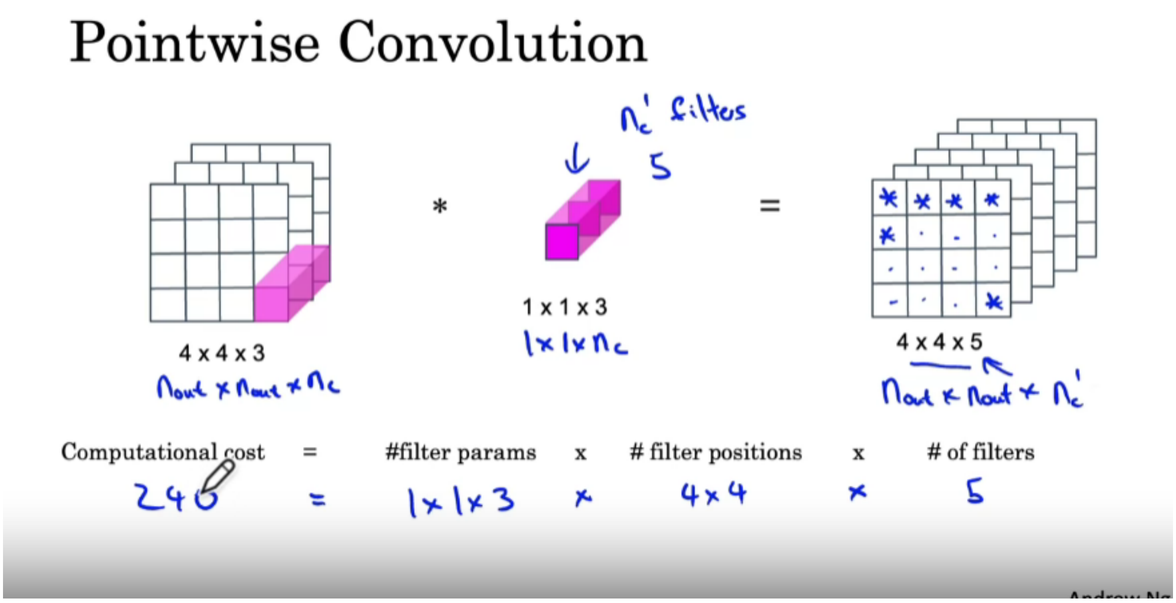
Computational cost = #filter params x # filter positions x # of filters

432 = 3×3 × 4×4 × 3

b. Pointwise Convolution: Combines the outputs of depth-wise convolution using 1 x 1 convolutions, allowing cross-channel information exchange.



Pointwise Convolution



Pointwise Convolution

4 x 4 x 3
$n_{out} \times n_{out} \times n_c$
*
n_c' filters
5
1 x 1 x 3
$1 \times 1 \times n_c$
=
4 x 4 x 5
$n_{out} \times n_{out} \times n_c'$

Computational cost = #filter params x # filter positions x # of filters

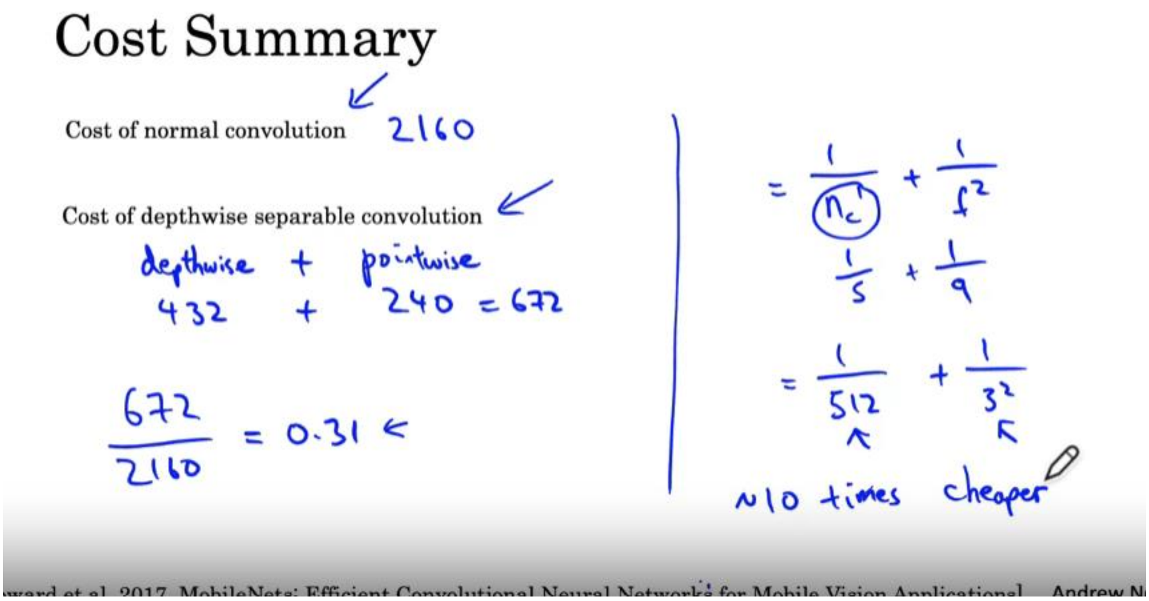
$240 = 1 \times 1 \times 3 \times 4 \times 4 \times 5$

Andrew Ng

2. Efficiency:

a. Reduced Parameters: Depth-wise separable convolutional significantly reduces the number of parameters compared to traditional convolutional layers, leading to smaller models.

b. Computational Efficiency: By decoupling spatial and cross-channel information processing, MobileNet achieves higher computational efficiency, making it suitable for resource-constrained devices.



Cost Summary

Cost of normal convolution 2160

Cost of depthwise separable convolution

depthwise + pointwise

432 + 240 = 672

$\frac{672}{2160} = 0.31$

$= \frac{1}{n_c'} + \frac{1}{f^2}$

$\frac{1}{5} + \frac{1}{9}$

$= \frac{1}{512} + \frac{1}{3^2}$

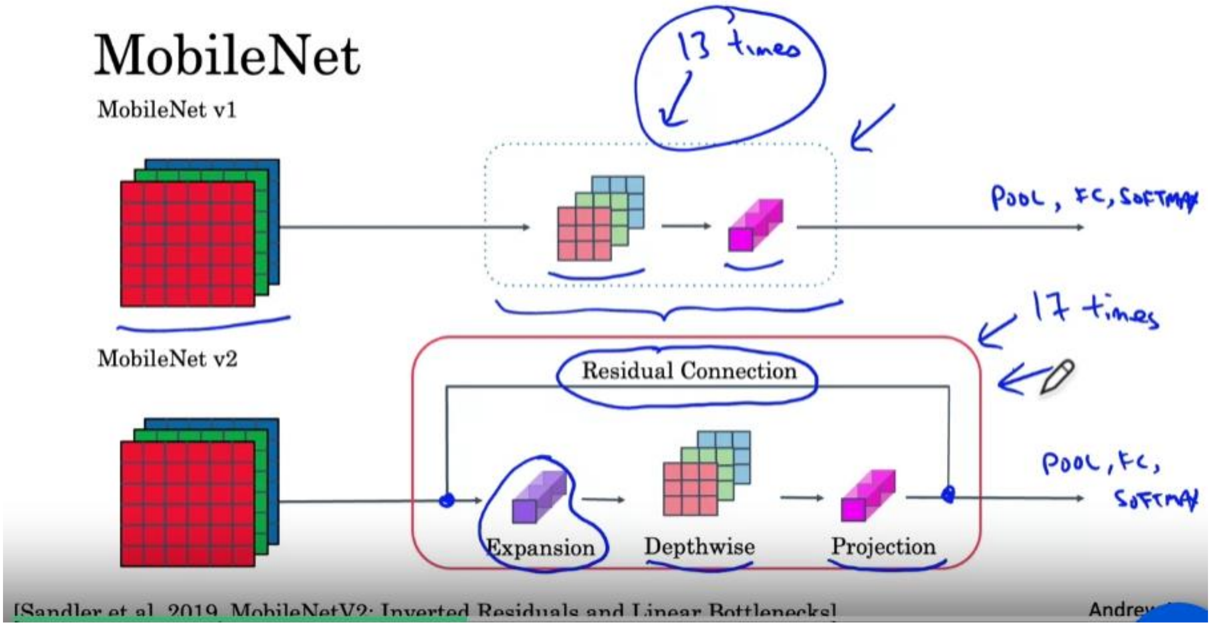
~10 times cheaper

[Howard et al. 2017, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications] Andrew Ng

3. Architecture Variants:

MobileNet V1: The original MobileNet architecture introduced by Google, focuses on balancing model size, latency, and accuracy.

MobileNet V2: Improved version with inverted residuals and linear bottlenecks, achieving better performance and efficiency by introducing residual connections and nonlinearities within depth-wise separable convolution blocks.



MobileNet

MobileNet v1

MobileNet v2

Residual Connection

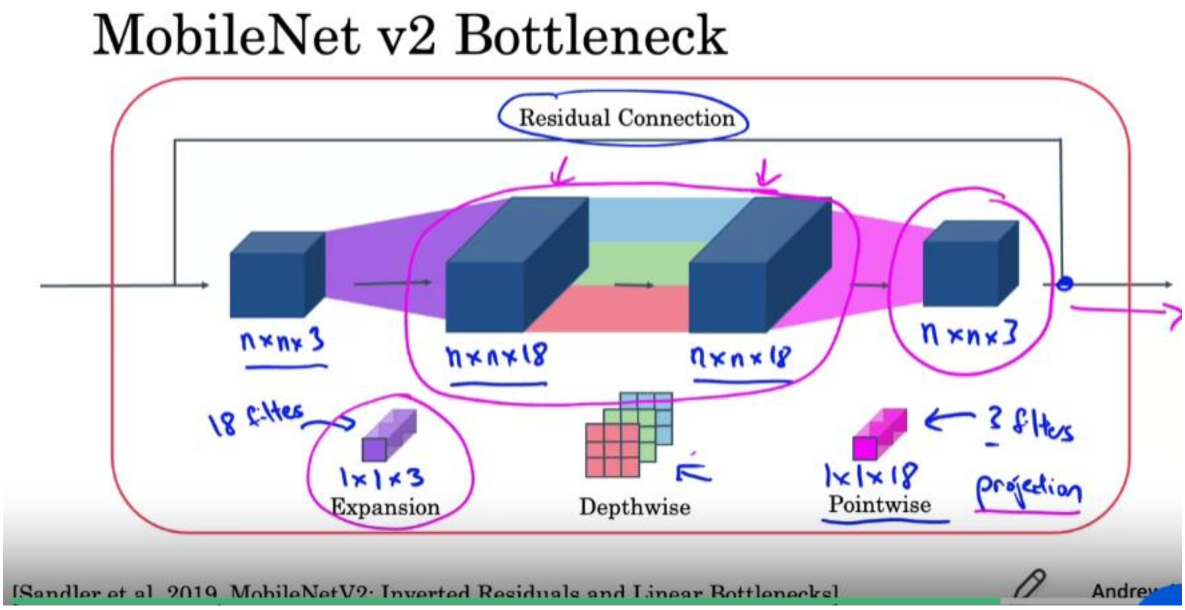
Expansion

Depthwise

Projection

[Sandler et al. 2019. MobileNetV2: Inverted Residuals and Linear Bottlenecks]

Andrew



MobileNet v2 Bottleneck

Residual Connection

Expansion

Depthwise

Pointwise

[Sandler et al. 2019. MobileNetV2: Inverted Residuals and Linear Bottlenecks]

Andrew

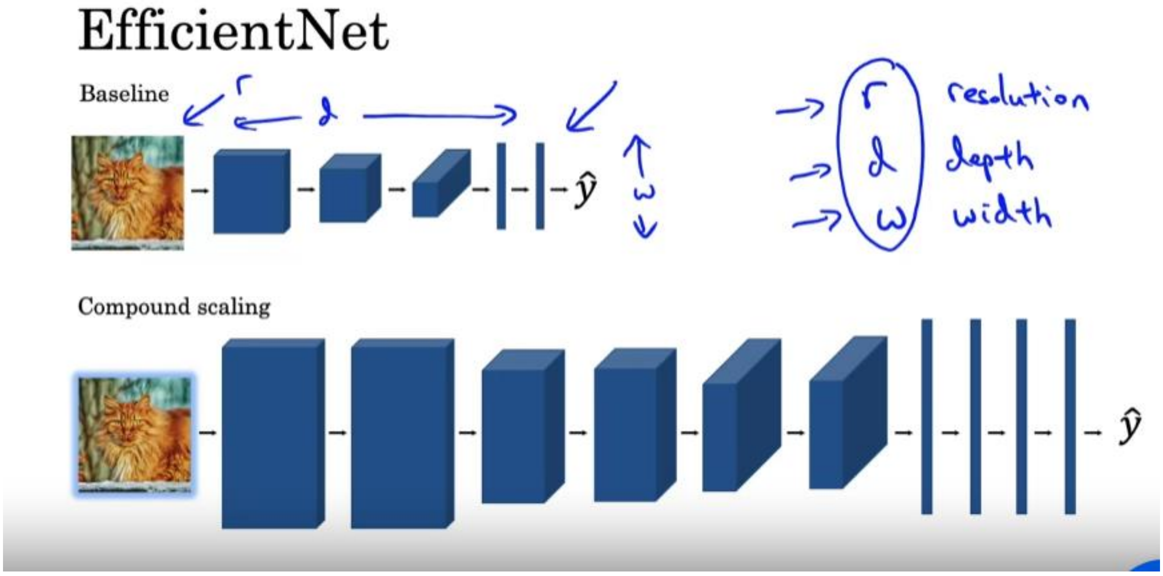
MobileNet V3: Further optimization for speed and accuracy, including a lightweight neural architecture search (NAS) strategy, activation improvements, and network width adjustment techniques.

EfficientNet: It is designed to achieve state-of-the-art performance on image classification tasks while maintaining efficiency in terms of model size and computational complexity. EfficientNet introduces compound scaling, a method that uniformly scales network depth, width, and resolution to achieve optimal trade-offs between model size and accuracy.

Depth Scaling: Increasing network depth adds more layers, allowing the model to capture more complex features and hierarchies.

Width Scaling: Increasing network width involves adding more channels to each layer, which enhances the capacity of the model to learn and represent features.

Resolution Scaling: Increasing input image resolution provides more detailed information to the network, potentially improving classification accuracy.



EfficientNet

Model Variants:

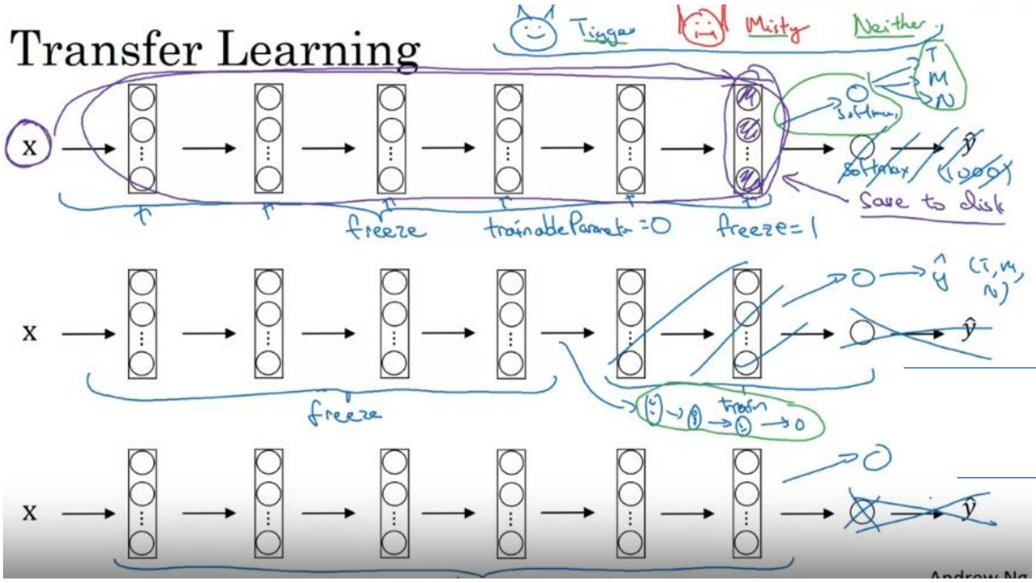
EfficientNet-B0 to B7: EfficientNet provides a spectrum of model variants ranging from B0 to B7, with increasing depth, width, and resolution. B0 is the smallest and least computationally intensive variant, while B7 is the largest and most resource-demanding variant.

EfficientNet-Lite: A lightweight variant of EfficientNet optimized for deployment on mobile and edge devices with limited computational resources. EfficientNet-Lite models achieve a good balance between accuracy and efficiency by reducing parameters and operations.

Applications:

Image Classification: EfficientNet models excel in image classification tasks, achieving state-of-the-art accuracy on benchmark datasets such as ImageNet with relatively compact model sizes.

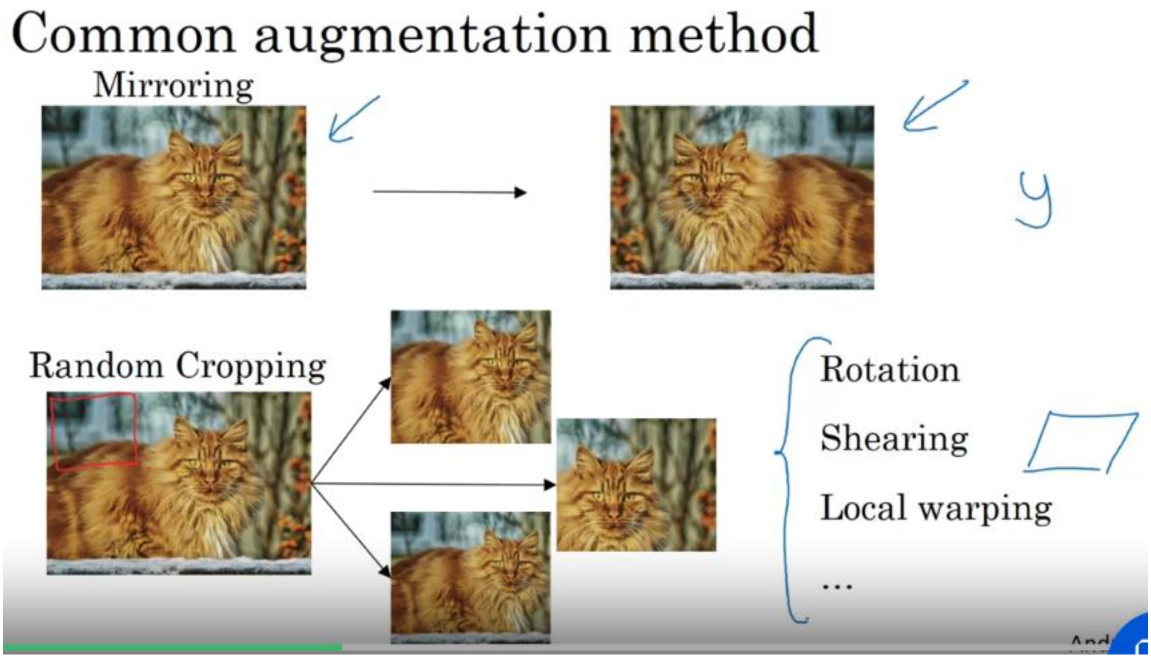
Object Detection and Segmentation: EfficientNet-based architectures serve as backbone networks for object detection and segmentation frameworks, providing high-quality feature representations for downstream tasks such as bounding box regression and instance segmentation.



Transfer Learning

For big data set

For small data



Common augmentation method

Mirroring

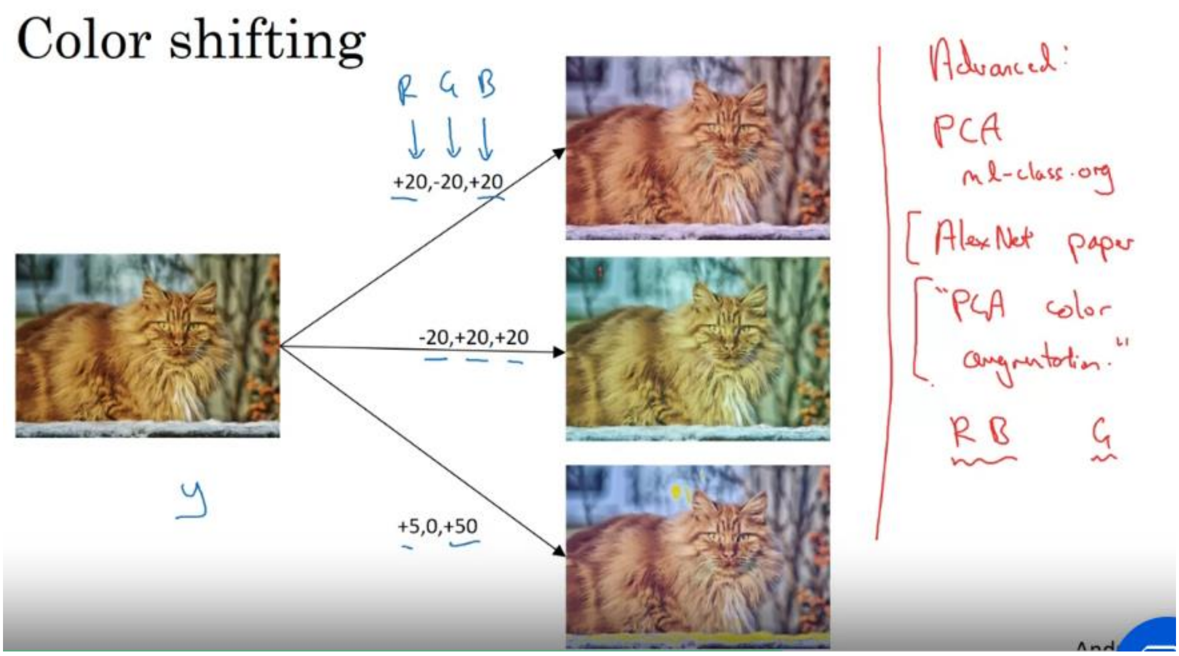
Random Cropping

Rotation

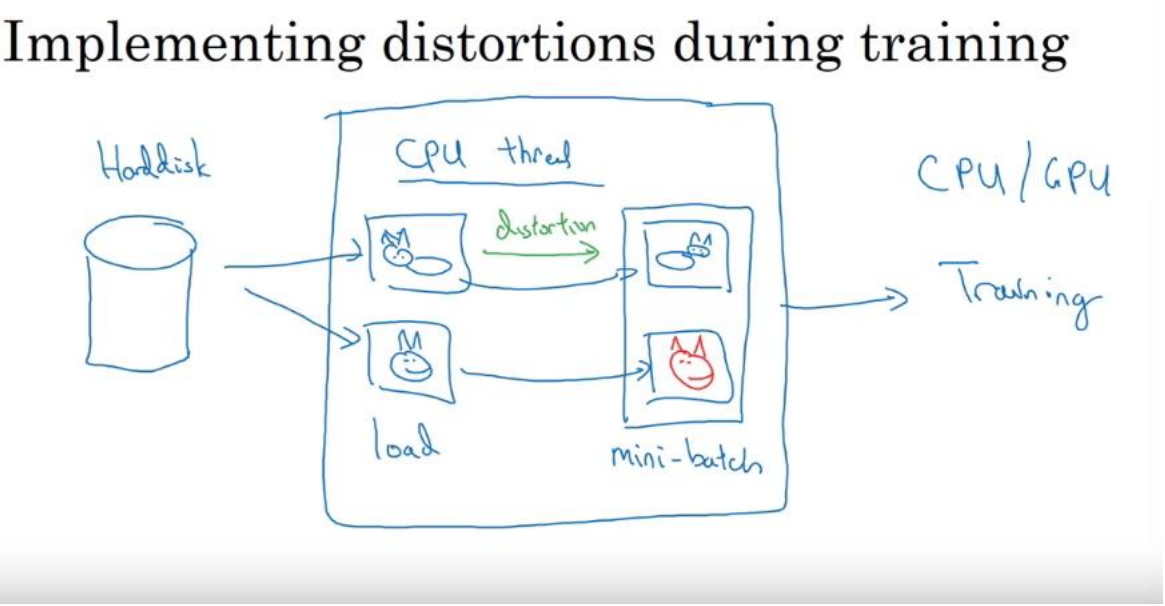
Shearing

Local warping

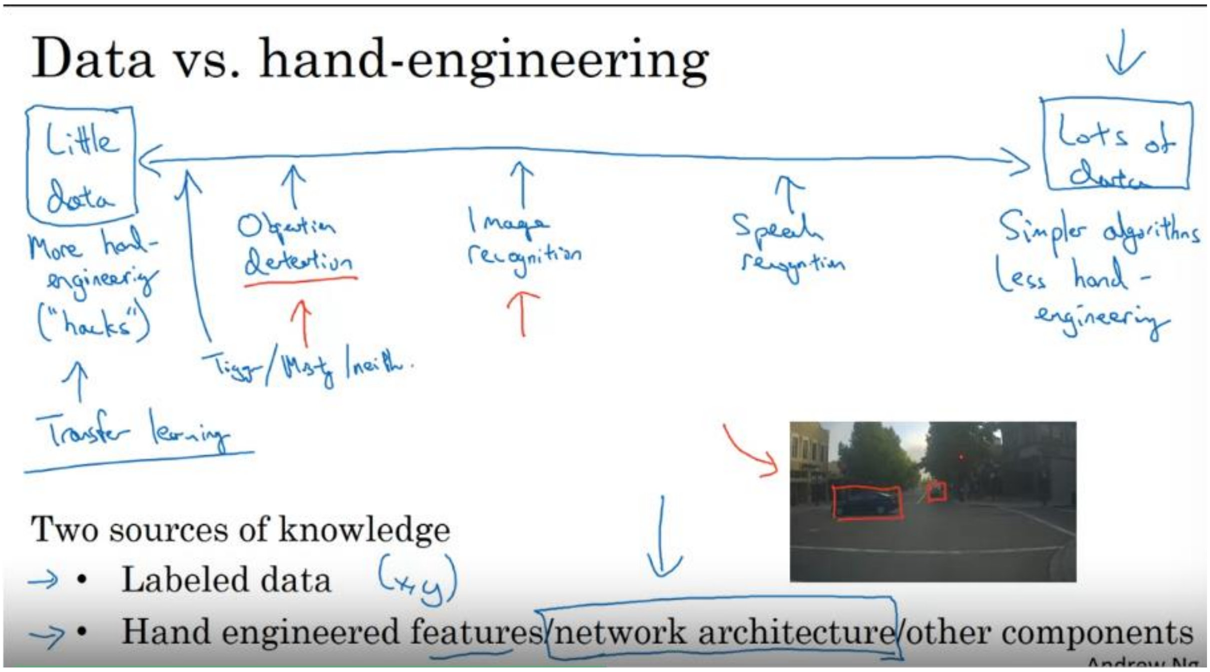
...



Color shifting



Implementing distortions during training



Data vs. hand-engineering

Two sources of knowledge

- Labeled data (x,y)
- Hand engineered features/network architecture/other components

Andrew Ng