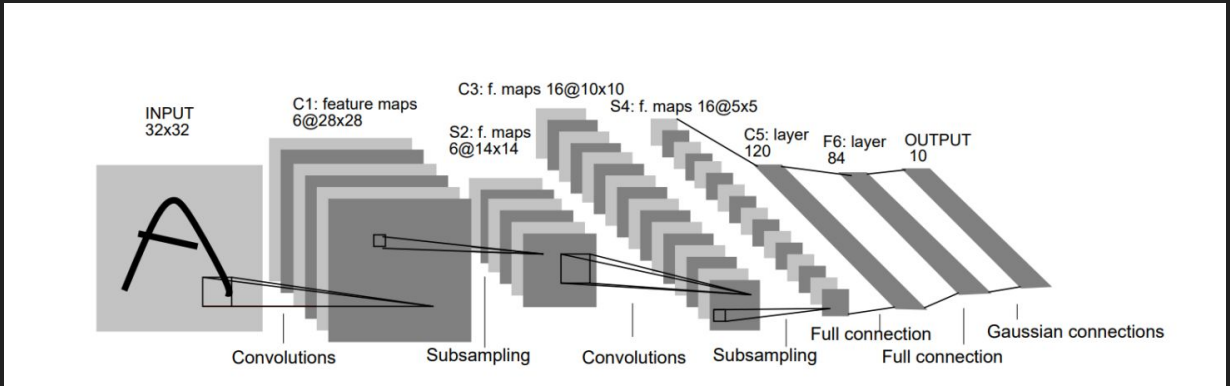
Lecture 7: Convolutional Neural Networks (CNNs)

QMUL Machine Learning society
Manuel Teres

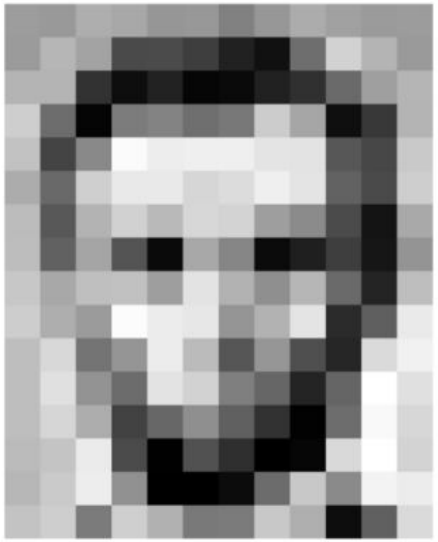
Introduction to CNNs

- Definition: CNNs are a class of deep neural networks designed for image processing.
- Key applications: Image recognition, object detection, medical imaging, etc.
- Structure: Consists of convolutional layers, pooling layers, fully connected layers.



How do we feed data into the model

What you see



Input Image

What you both see

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Input Image + values

What the computer "sees"

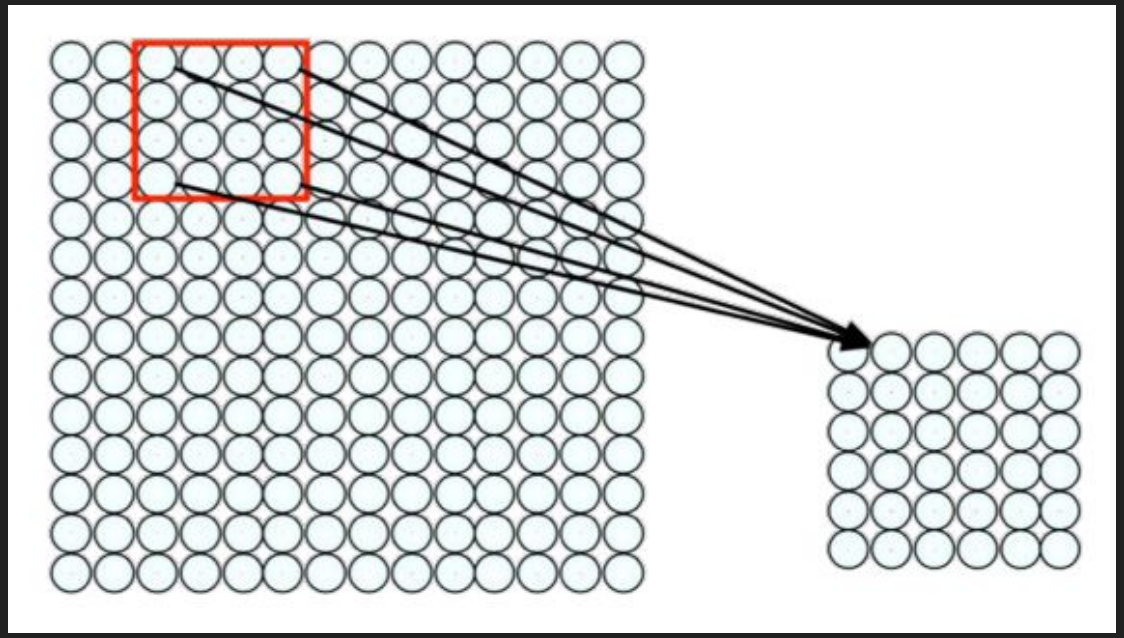
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

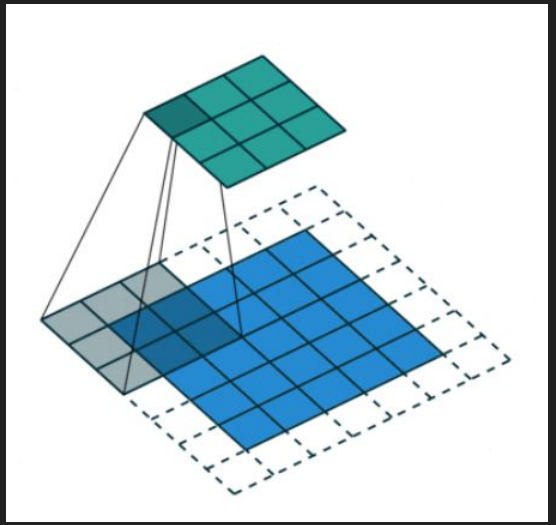
Pixel intensity values
("pix-el"=picture-element)

Convolutional Layers

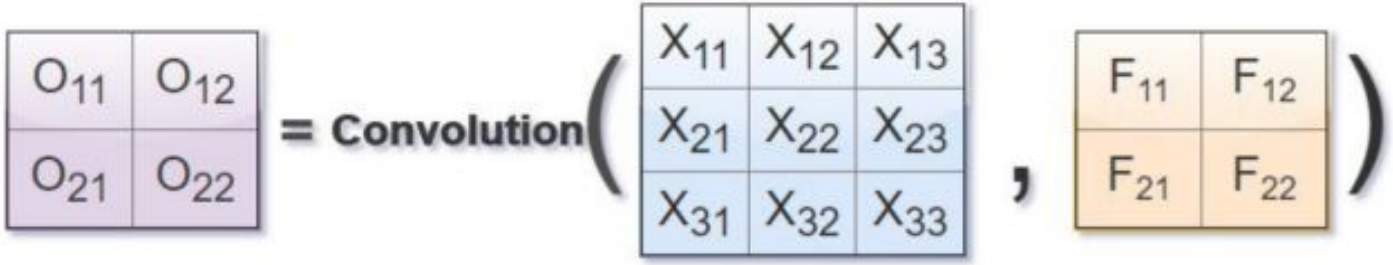
- Purpose: Extracts spatial features from input images.
- Convolution operation:
  - Uses a kernel (filter) to slide over the image.
  - Computes dot product at each position.
  - Produces feature maps.
- Example:
  - 3x3 filter applied to a 5x5 image.
- Activation function: ReLU (Rectified Linear Unit) for non-linearity.
- We usually add padding

Kernel (filter) sliding over the image





Resulting Matrix after applying the filter



Resulting Matrix = Convolution(Initial Data, Filter)

Practical Numerical Example

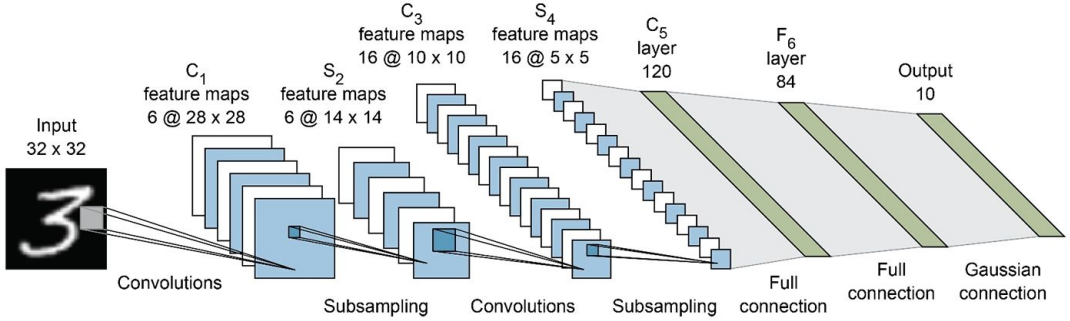
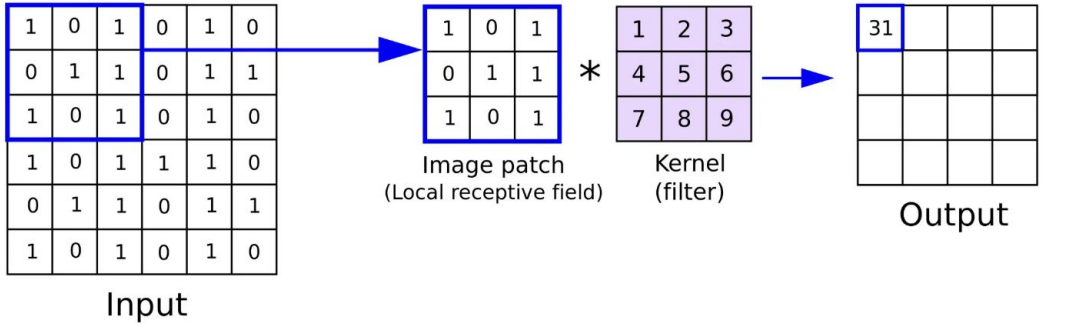


Image source

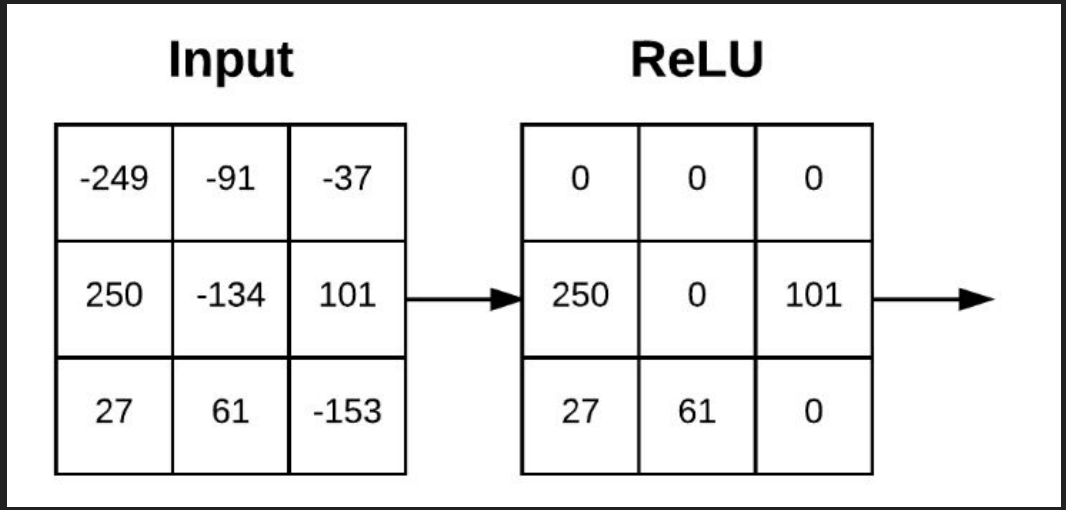
The convolutional layer

iq.opengenus.org



Use of an activation function to introduce non-linearity

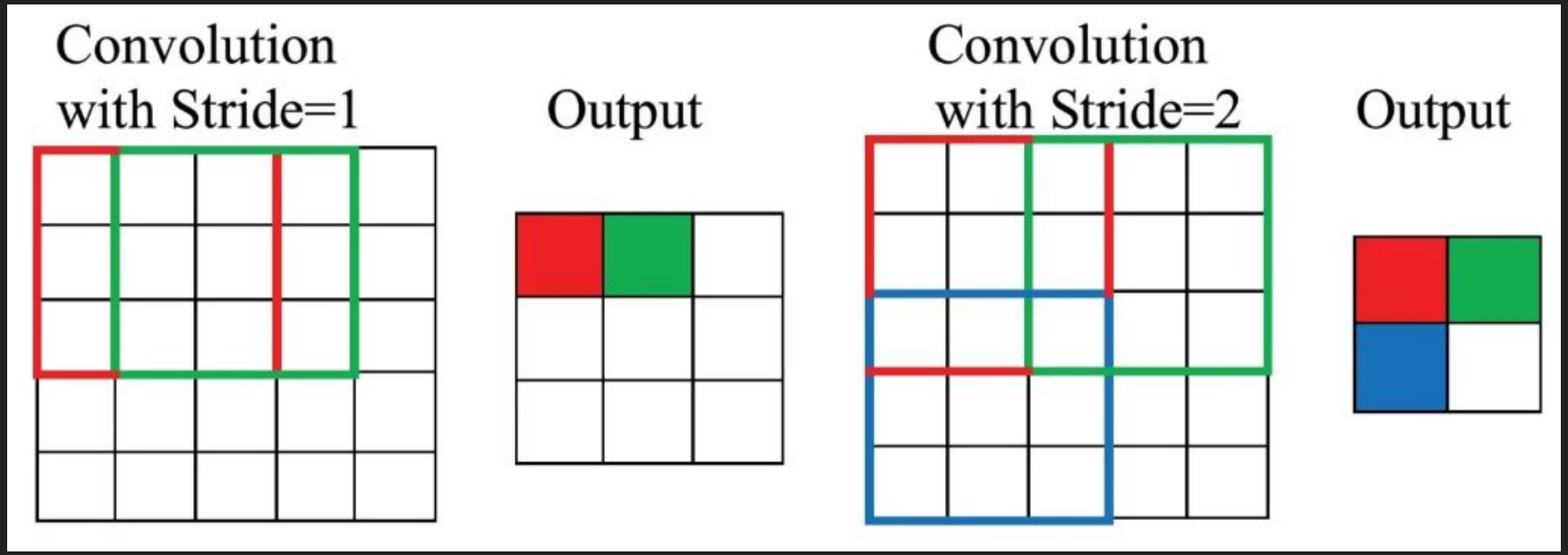
Recall that ReLU makes negative inputs = 0.



Stride and Padding

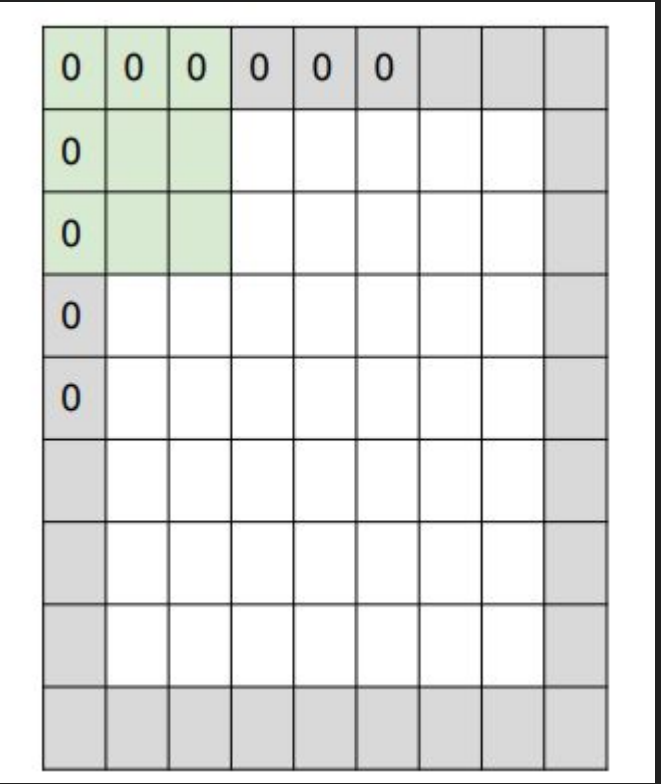
- Stride: Defines the step size of the filter.
  - Larger stride reduces spatial dimensions.
  - “How many steps does it take the filter to go through the image”
- Padding:
  - Zero-padding preserves spatial dimensions.
  - Used to avoid excessive size reduction.
- Example: 5x5 input with a 3x3 filter and stride 2.

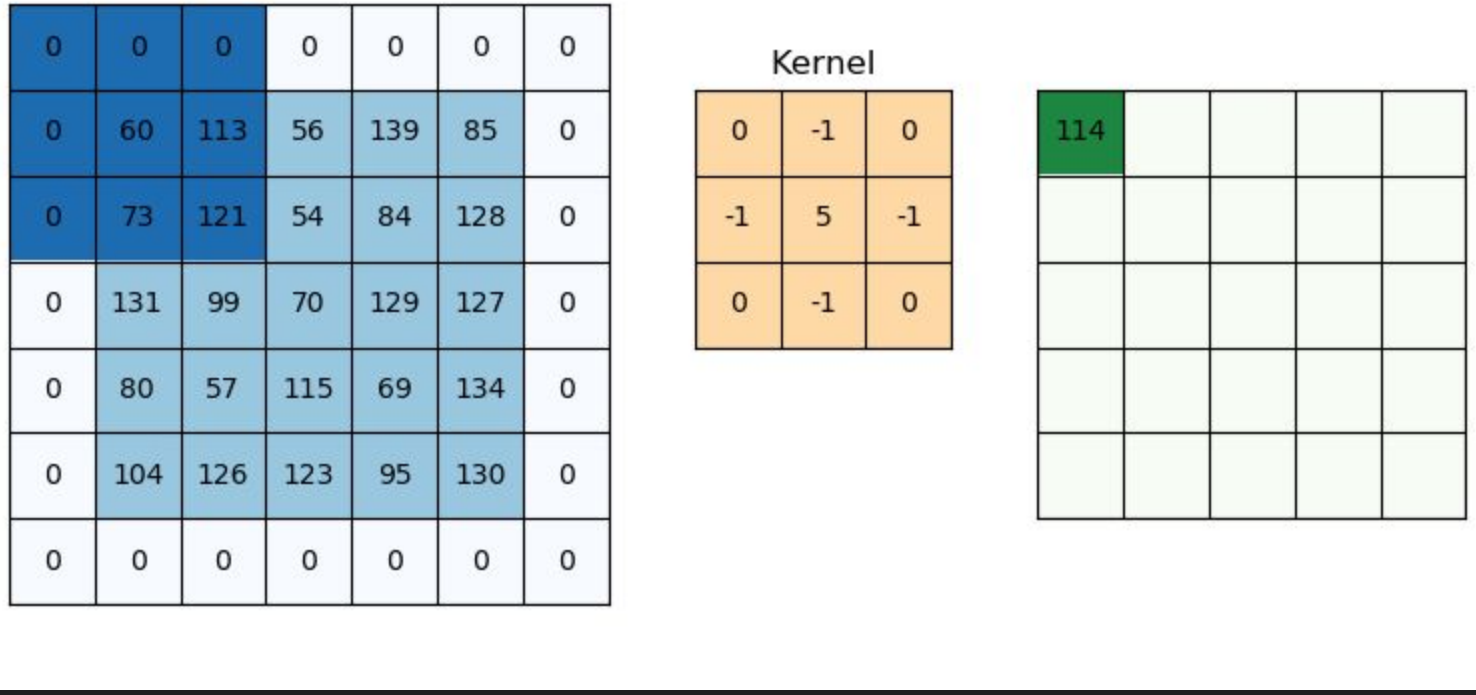
Stride


Convolution
with Stride=1
Output
Convolution
with Stride=2
Output

Padding

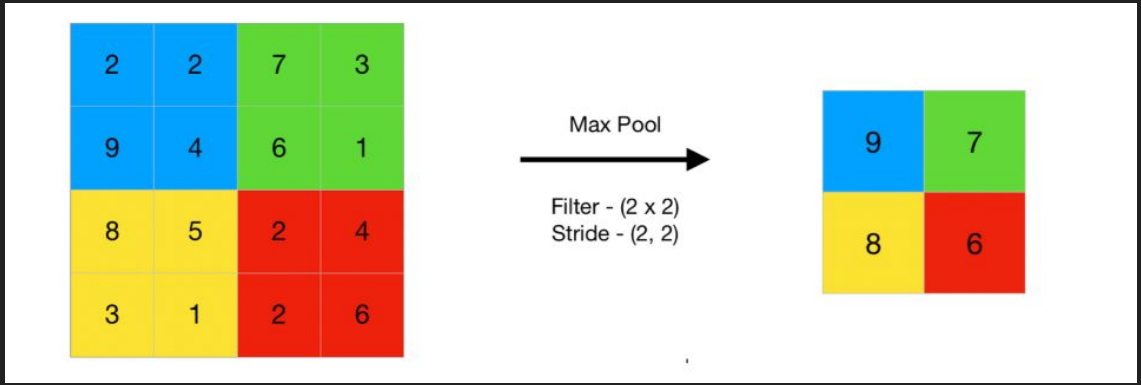
It is common to add 0s to the edges of the images. Otherwise, we are downsampling the image and losing




0 0 0 0 0 0 0
0 60 113 56 139 85 0
0 73 121 54 84 128 0
0 131 99 70 129 127 0
0 80 57 115 69 134 0
0 104 126 123 95 130 0
0 0 0 0 0 0 0
Kernel
0 -1 0
-1 5 -1
0 -1 0
114

Pooling Layers

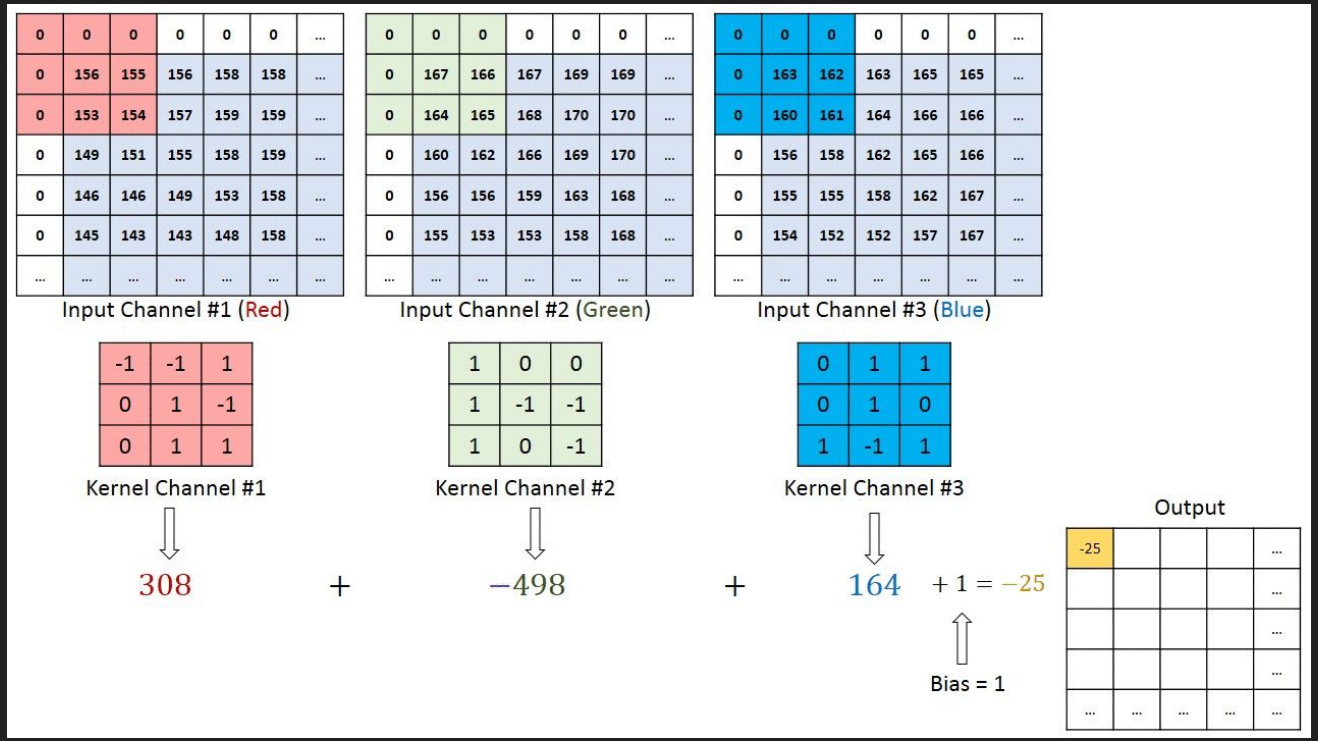
- Purpose: Reduces spatial dimensions while retaining important features.
- Types:
  - Max pooling: Takes the maximum value from a region.
  - Average pooling: Takes the average value.
- Benefit: Reduces overfitting and computation cost.



Grayscale vs Colour Images

- Grey Scale:
  - Single-channel (1D) input.
  - Faster computation, less data.
- Colour (RGB):
  - 3-channel (Red, Green, Blue) input.
  - More complex feature extraction.
- MNIST data set: 28x28 grayscale image vs. 28x28x3 RGB image.

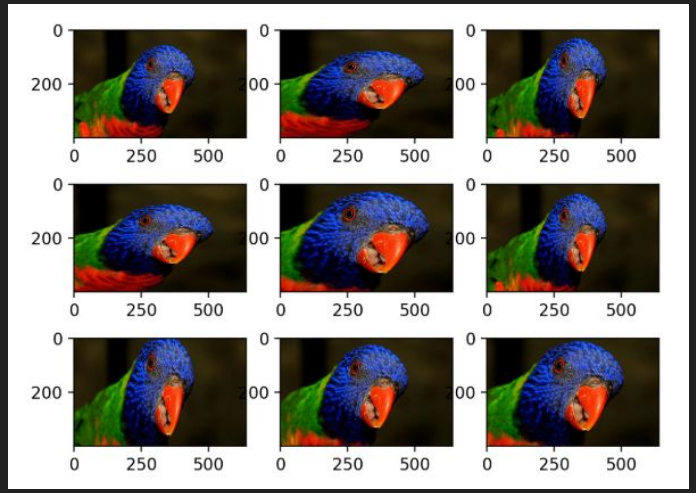
Colour RGB: 3-channel (Red, Green, Blue) input

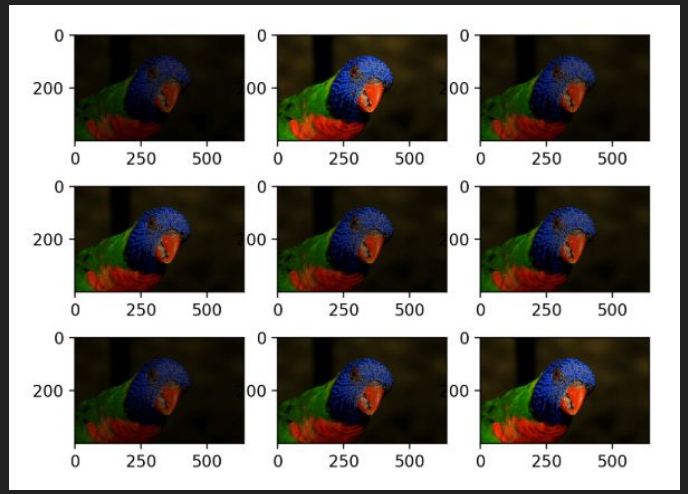


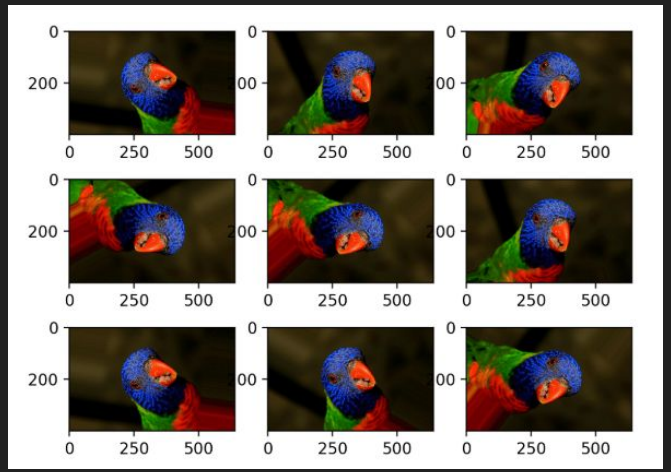
Data Augmentation

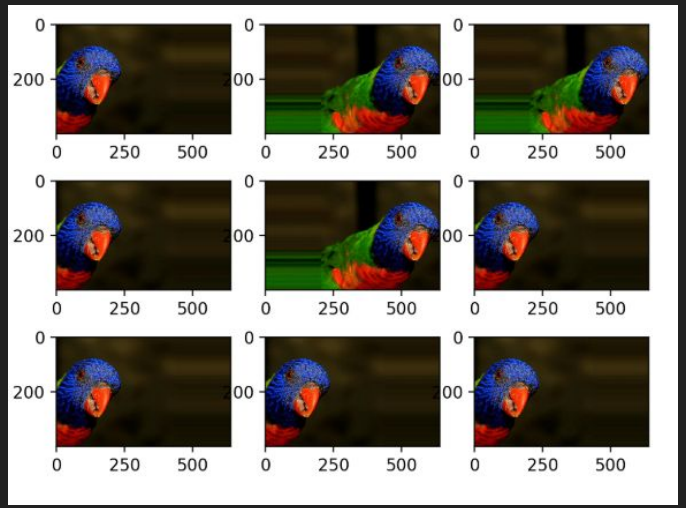
Purpose: Increases dataset diversity to improve generalisation.

- Techniques:
  - Rotation, flipping, zooming, brightness adjustment.
  - Random cropping and noise addition.
- Example: Original vs. augmented images.
- Benefit: Reduces overfitting and improves model robustness.


0
200
0 250 500


0
200
0 250 500


0
200
0 250 500


0
200
0 250 500

Training Parameters in CNNs

Forward Pass:

- Input data (images) is passed through the network.
- Each layer performs operations (convolution, activation, pooling).
- Outputs are generated for each layer.

Loss Calculation:

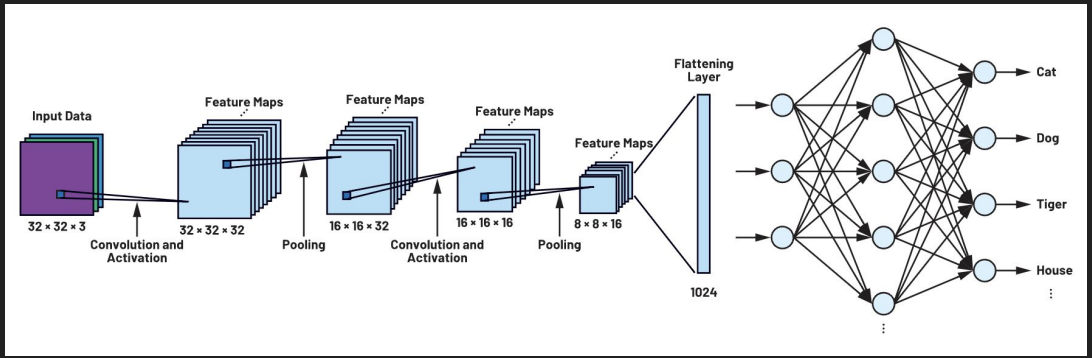
- A loss function computes the error between predicted output and true labels.
- The goal is to minimise this loss through training.

Backward Pass:

- Backpropagation calculates gradients of the loss with respect to the network parameters.
- Gradients indicate how to adjust the parameters (weights and biases) to reduce the error.

Optimizer:

- Optimisers like SGD or Adam adjust the parameters based on the gradients.



Parameters Trained in Convolutional Neural Networks (CNNs)

Weights:

- The weights are the kernels (filters) applied to input data.

Biases:

- Bias terms added to the output of the convolution before activation.

Fully Connected Layer Parameters:

- Weights and biases of the fully connected (dense) layers at the end of the network.

Pytorch Implementation

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.cnn_layers = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=12, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(in_channels=12, out_channels=24, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(kernel_size=2)
        )
        self.fc_layers = nn.Sequential(
            nn.Linear(in_features=24 * 7 * 7, out_features=64), nn.ReLU(), nn.Dropout(p=0.2),
            nn.Linear(in_features=64, out_features=10)
        )

    def forward(self, x):
        x = self.cnn_layers(x)  # Pass through convolutional layers
        x = x.view(x.size(0), -1)  # Flatten the tensor to 1D
        return self.fc_layers(x)  # Pass through fully connected layers
```