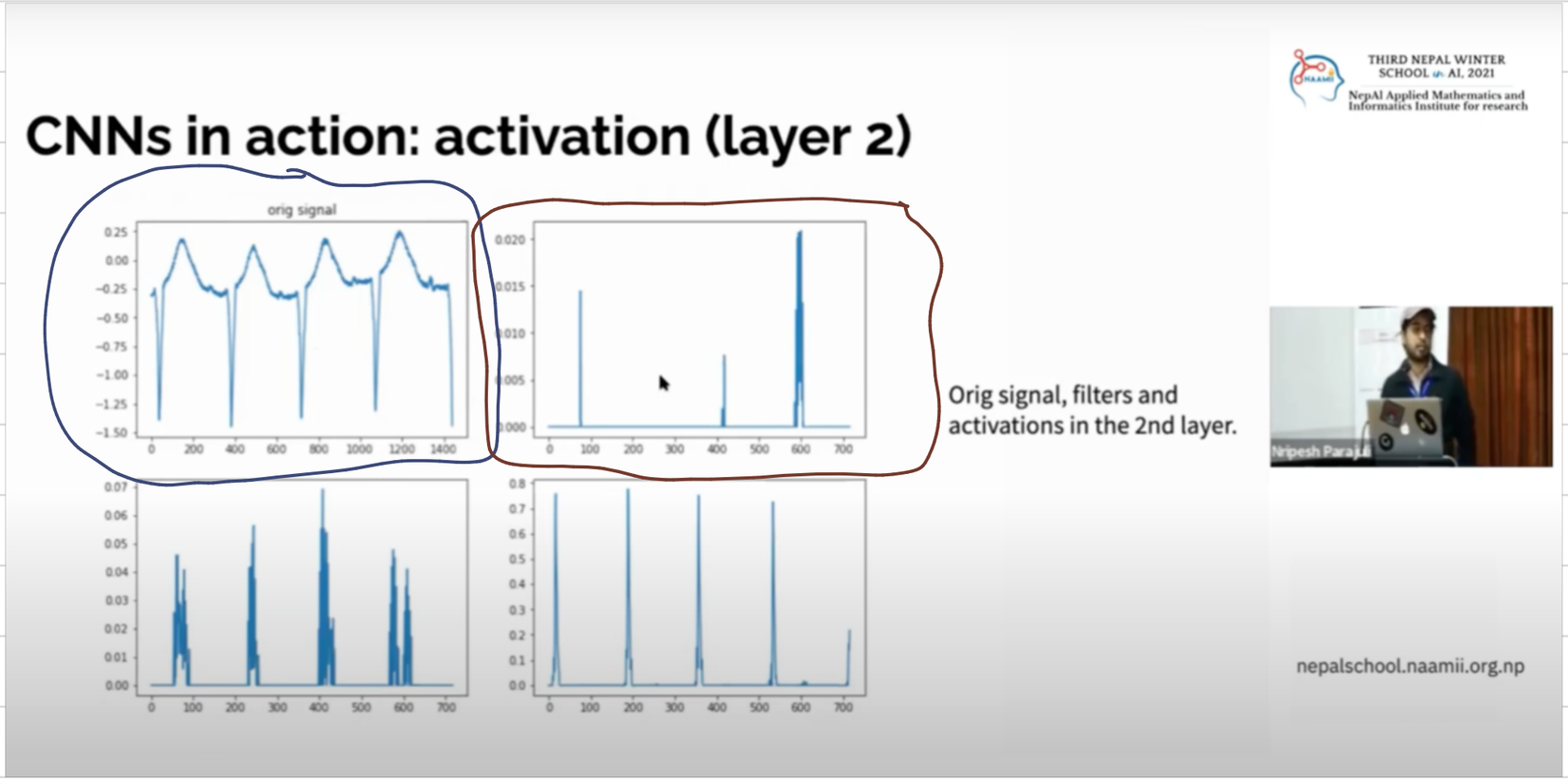
Day 17, Oct-19, 2024.

Continuing CNNs.



(1) Top left:

Original Input Signal, having multiple peaks & troughs. Can be Sensor Signal (Time-Series).

(2) Top Right: and Bottom Graphs.

→ illustrates the activations at different points within the CNN specifically in the second layer

→ Spikes suggest that specific features or patterns in the original signals are detected

→ Activations are sparse, CNNs after filtering - certain patterns (like edges in images or specific frequencies in Image/Audio Signals).

Input Signals ⟶ Filter ⟶ Activation At different points ⟶ Output.

Convolutional Neural Networks Scope.

→ try CNNs for time-series signal Although we have RNN for the time-series and sequence input.

→ CNNs works for the reliable and repeatable patterns & input data. Images are periodic & have temporal pattern for processing and extraction.

→ CNNs are not meant to do time-series input. They are for Image processing and Computer Vision, Video processing, Speech and Audio processing, AR/VR, Generative Models.

The Convolutional Operation

$$s(t) \Rightarrow (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)\, w(t-a)$$

where $x(t)$ Signal

$w(t) \rightarrow$ Kernel

$t \Rightarrow$ time / position of o/p

$a \Rightarrow$ position or time-shift applied to Kernel

Discrete Convolutional formula.

$$S(i,j) = (K * I)(i,j) \Rightarrow \sum_m \sum_n I(i+m, j+n)\, K(m,n)$$

useful for specific to Images, which have finite known sizes

where $I(i,j) \rightarrow$ input Image

$K(m,n) \rightarrow$ Convolutional Kernel

$m, n \rightarrow$ indices for Kernel

$I(i+m, j+n) \rightarrow$ Input Image pixel at position $(i+m, j+n)$ i,j are local pixels.

for i in range( ):

→ Outer loop iterates

Over the rows of the images starting Index = 1 and ending h-2.

The convolution operation: code

```
def conv_im_kernel(im, kernel):
    # hard-coded for 3 X 3 kernels
    k_h, k_w = kernel.shape
    im_h, im_w = im.shape
    new_im = np.zeros(im.shape)
(1) for i in range(1, im_h - 2):
(2)     for j in range(1, im_w - 2):
(3)         im_ij = im[i : i + 3, j : j + 3]
(4)         new_im[i, j] = np.sum(np.multiply(im_ij, kernel))
    return new_im
```

nepalschool.naamii.org.np

for j in image

→ Inner loop iterates over the columns of the images, starting from index 1 and ending iw - 2, ensures enough space for Kernel 3×3 left and right.

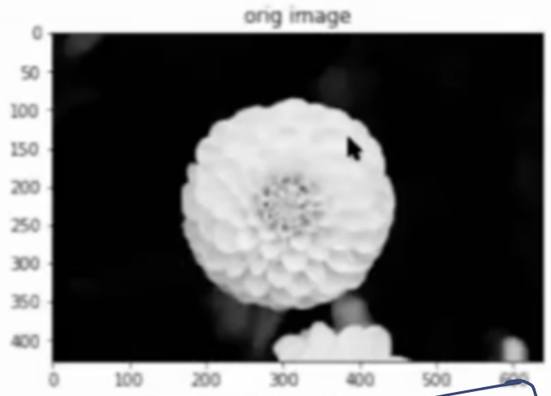
(3) im_ij = _ _ _ → Extracts 3×3 Subregion of the image centered (i,j)
Slice i: i+3 selects 3 rows and 3 columns.

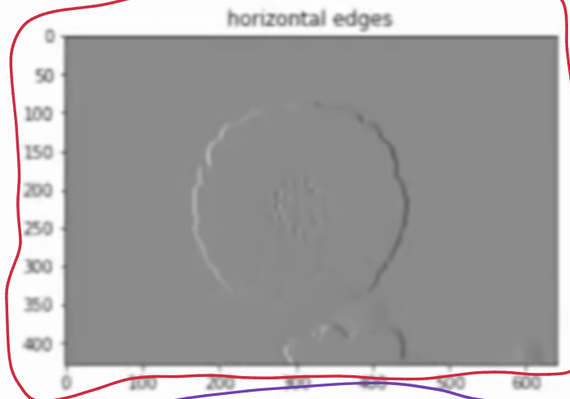
(4) multiplies 3×3 Image element-wise with Kernel & Sum them (Dot product)

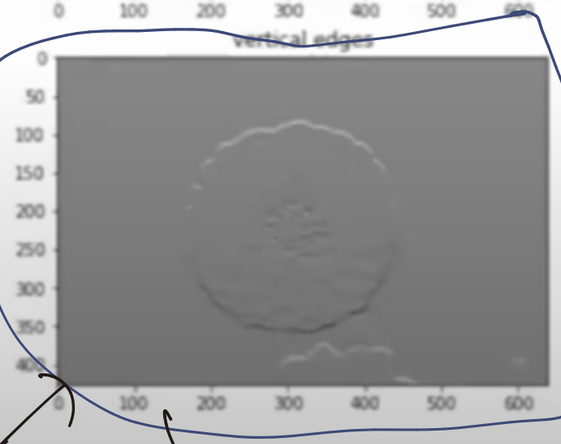
THIRD NEPAL WINTER SCHOOL in AI, 2021

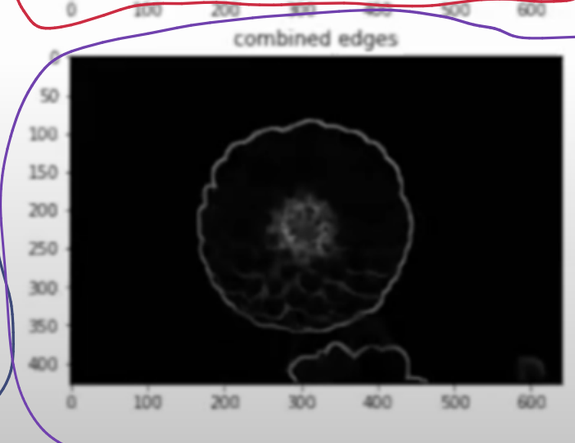
NepAl Applied Mathematics and Informatics Institute for research

Convolution example: edge detection









→ vertical edges.

Vertical & Horizontal edges

Nirpesh Parajuli

Combined Edges

nepalschool.naamii.org.np

Horizontal Edges

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

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

Pooling : Is it differentiable | have Parameter ? → No.

Stride :

Number of pixels (or units) by which the filter or Kernel moves across the Input Image during the Convolutional operation.

(1) Stide value 1: → filters moves one pixel at a time.

(2) Stide value 2: filters moves 2 pixels at a time.

$$\text{O/p Size} \Rightarrow \left[\frac{\text{Input Size} - \text{Kernel Size} + 2 \times \text{padding}}{\text{Stride}}\right] + 1$$

Larger strides → Output Smaller feature Map.

Example:

Input Size = 32

Kernel Size : 5

padding : 0

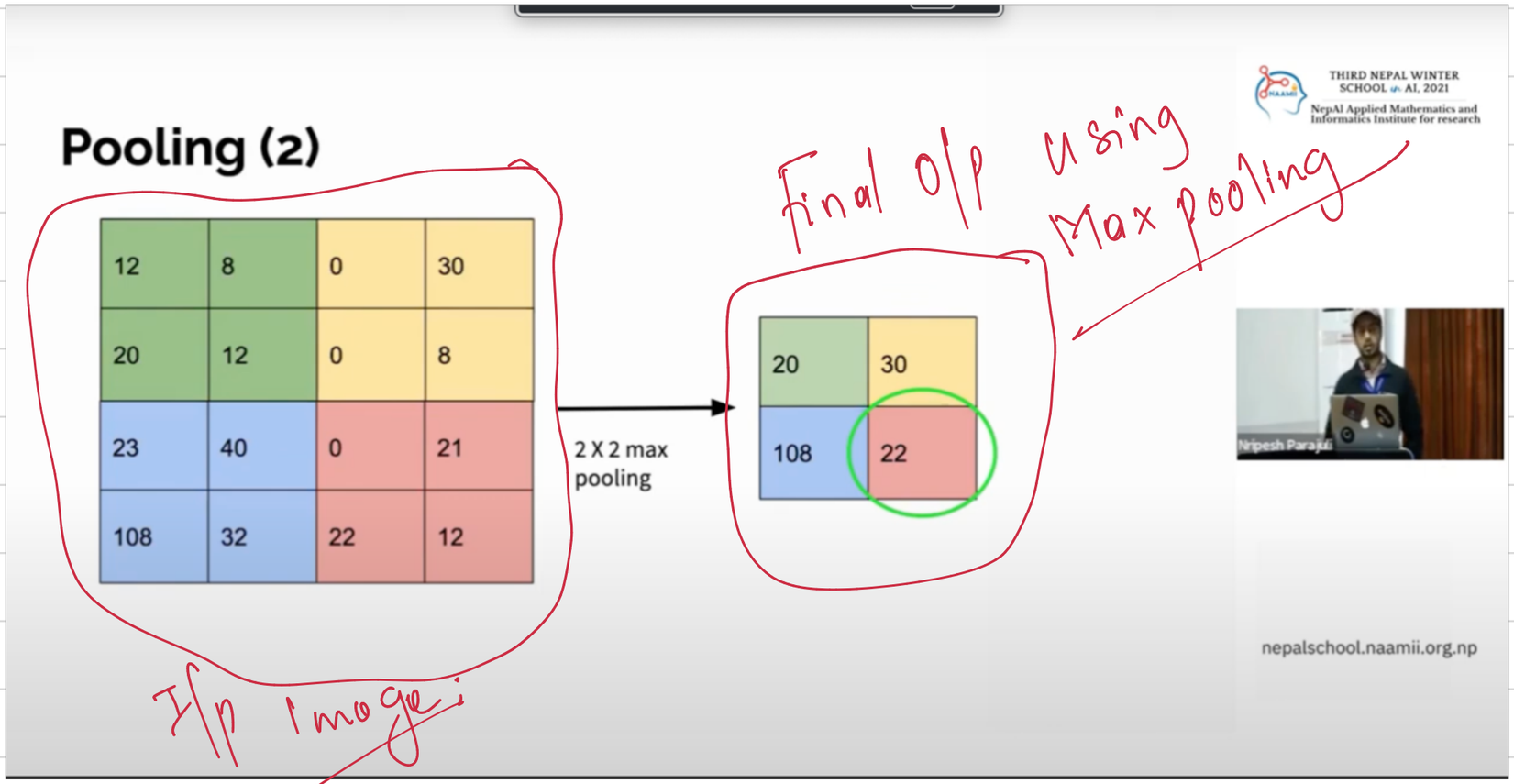
Stides : 2

$$\text{O/p Size} \Rightarrow \left[\frac{32-5+2\times0}{2}\right]+1$$

[O/p Size = 14] Output feature Map Size 14x14.

Translation Invariance: → Ability of ANN to recognize features regardless of their exact position in the input.

- Purpose:
  Max Pooling
  → Reduce Spatial dimensions of the representation
  → Extract dominant features
  → Provide a form of translation invariance.



Example: Among

12	8
20	12

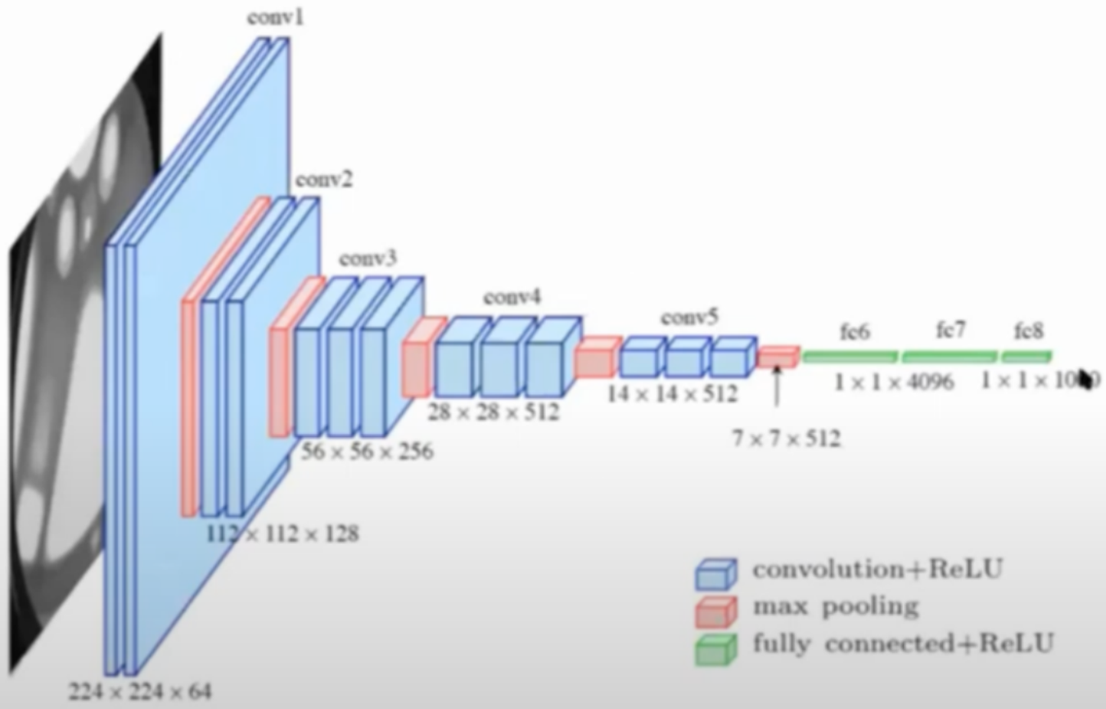
→ 20 is Max, and Similarly for other 2x2 this shows that max pooling allows to detect features regardless of their exact position.

Recap of CNN's Architecture.

ReLU [0, ∞)

THIRD NEPAL WINTER SCHOOL in AI, 2021
NepAl Applied Mathematics and Informatics Institute for research

Understanding the network diagrams



This is a typical CNN with some FC layers towards the end:

- Input image - (224 X 224 X 3)
- First layer is convolution + ReLU - padded to have same shape - 64 channels.
- (2 X 2) maxpooling applied after this -> new size 112 X 112
- Repeated 4 more times.
- The 2D features are vectorized eventually.
- 3 fully connected layers are added.

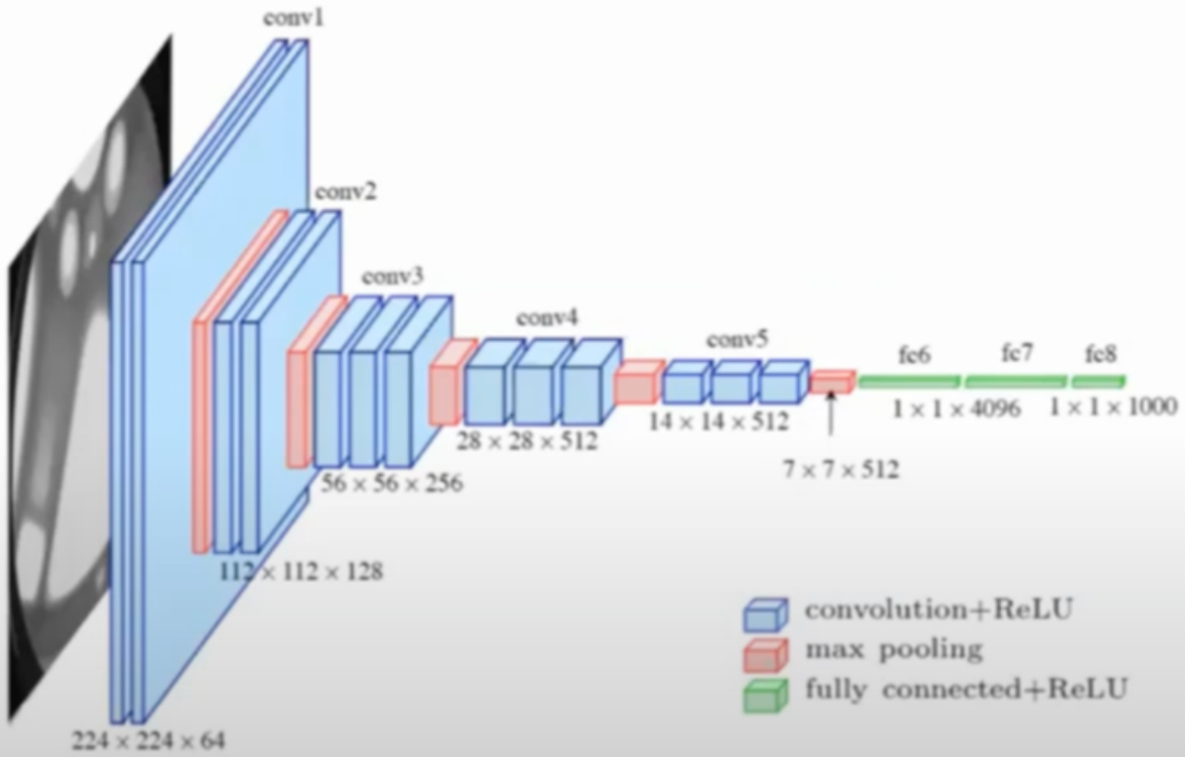
nepalschool.naamii.org.np

Src: https://www.researchgate.net/figure/Fig-A1-The-standard-VGG-16-network-architecture

∴ Decreasing Spatial Space and Increasing Features Map (Extract Rich Features)

Layer 1:
64 - filters
(3x3) → Kernel
x3 → RGB
+64 ⇒ bias

Counting number of parameters



- Layer 1:
  - 64 X (3 X 3) X 3 + 64 = 1792
  - 64 X (3 X 3) X 64 + 64 = 36,928
- Layer 2:
  - 128 X (3 X 3) X 64 + 128 = 73,856
  - 128 X (3 X 3) X 128 + 128 = 147,584
- .....
- FC6 -> 7 x 7 x 512 X 1024 = 102,760,448
- FC7 -> 4096 X 1000 = 25,088,000
- Total = ~138 millions

2 Convolutional Operation

THIRD NEPAL WINTER SCHOOL on AI, 2021
NepAl Applied Mathematics and Informatics Institute for research

nepalschool.naamii.org.np

Src: https://www.researchgate.net/figure/Fig-A1-The-standard-VGG-16-network-architecture

Layer 2:
① 128 → Filters for features | x (3 x 3) → Kernel | x 64 → Channel of Layer 1 | +128 → bias |

Layer 1.

O/p channels → 64 × (3×3) × 3 + 64 ⇒ 1792 ← bias
Kernel size → (3×3)
RGB → ×3

64 × (3×3) × 64 + 64 ⇒ 36,928 ← Total parameters.

- 64 → no. of filters or O/p channels
- (3×3) → Kernel size
- 64 → Channel of layer1
- 64 → bias

Layer 2:

128 × (3×3) × 64 + 128 ⇒ 73,856 ← Total parameters

- 128 → no. of filters
- (3×3) → Kernel size
- 64 → Input from previous o/p channel layers
- 128 → bias

FC6 (fully Connected 6):

$7 \times 7 \times 512 \times 4096 = 102,760,448$

$7 \times 7 \times 512$ → Size of Input feature Map

4096 ⇒ no. of neurons in this Layer No Separate bias term shown, but typically there would be 4096 bias term.

FC7 (Fully Connected 7):

$4096 \times 1000 \Rightarrow 4,096,000$

- 4096 → no. of Input neurons from FC6
- 1000 → no. of O/p neurons (would 1000 bias terms).

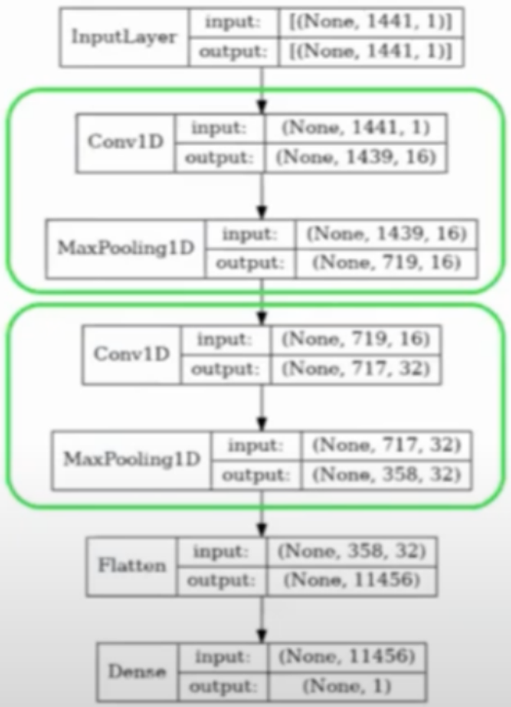
Total → ~ 138 millions (Approx).

Coding Implementation part:

THIRD NEPAL WINTER SCHOOL in AI, 2021

NepAl Applied Mathematics and Informatics Institute for research

Implementation: what does it look like?



A simple CNN in code

```python
from tensorflow import keras
from tensorflow.keras import layers

#create model
cnn_model = keras.Sequential(
    [
        keras.Input(shape=(X_train.shape[1], 1)),
        layers.Conv1D(16, kernel_size=3, activation="relu"),
        layers.MaxPool1D(2),
        layers.Conv1D(32, kernel_size=3, activation="relu"),
        layers.MaxPool1D(2),
        layers.Flatten(),
        layers.Dense(
            1, activation="sigmoid",
        ),
    ]
)

cnn_model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[
        metrics.BinaryAccuracy(),
    ]
)

cnn_model.fit(
    X_train_exp, y_train,
    validation_data=(X_test_exp, y_test),
    epochs=8,
    callbacks=[model_checkpoint_callback],
)
```

nepalschool.naamii.org.np

Hyperparameters are not in the training dataset (Eg: number of layers, batch size, K in KNN)