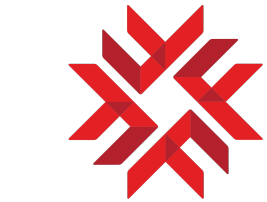
Fundamentals of CNNs II

Course:
INFO-6152 Deep Learning with Tensorflow & Keras 2



Developed by:
Mohammad Noorchenarboo

September 15, 2025

Current Section

The Pooling Layer

In medical imaging or weather prediction, we often deal with huge images containing millions of pixels. **Question:** How can we reduce the size of these images while still keeping the most relevant features for deep learning models?

Concept: Pooling

Pooling is a **downsampling operation** applied to feature maps.
For a region R of size $m \times n$, the pooling function P produces one output value:

$$F'(i,j) = P\big(\{F(u,v) \mid (u,v) \in R(i,j)\}\big)$$

Common pooling types:

- **Max pooling:** $P(R) = \max(R)$
- **Average pooling:** $P(R) = \frac{1}{|R|} \sum_{(u,v)\in R} F(u,v)$

Output size formula:

$$H_{out} = \frac{H - m + 2p}{s} + 1, \quad W_{out} = \frac{W - n + 2p}{s} + 1$$

The Pooling Layer

TensorFlow: Max Pooling Layer

```
import tensorflow as tf

# MaxPooling after a Conv layer
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3),
                           activation='relu', input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2), strides=2)
])

model.summary()
# Expected: Feature map reduced in size by pooling
```

The Pooling Layer

Real-World and Numerical Examples

- **Medical imaging:** reduces resolution but keeps essential tumor boundaries.
- **Speech recognition:** compresses spectrograms while retaining sound patterns.
- **Weather prediction:** shrinks satellite images while preserving storm shapes.
- **Numerical example:**

Input 4×4 matrix: $\begin{bmatrix} 1 & 3 & 2 & 4 \\ 5 & 6 & 1 & 2 \\ 2 & 4 & 0 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$

Apply 2×2 max pooling with stride 2: $\begin{bmatrix} 6 & 4 \\ 4 & 4 \end{bmatrix}$

The Pooling Layer

Common Mistakes

- Choosing pooling windows that are too large $\rightarrow$ lose too much detail.
- Stacking many pooling layers $\rightarrow$ excessive information loss.

Caveats

- Pooling reduces dimensions but also discards some spatial precision.
- Modern CNNs often replace pooling with strided convolutions for better feature preservation.

Get more info $\rightarrow$ DeepLearning.AI: Pooling Explained

Summary: The Pooling Layer

Concept	Key Point
Pooling definition	Downsampling feature maps by summarizing local regions
Max pooling	Selects maximum value from region
Average pooling	Computes average value from region
Output size formula	$H_{out} = \frac{H-m+2p}{s} + 1, \quad W_{out} = \frac{W-n+2p}{s} + 1$
Benefits	Reduces dimensions, prevents overfitting, increases efficiency
Caveats	May lose spatial detail; can be replaced by strided convolutions

Current Section

Global Pooling

In medical imaging, after extracting features from an X-ray using deep CNNs, we often need to reduce the entire feature map into a single value per channel to make a classification decision (e.g., healthy vs. cancer). **Question:** How can we shrink a whole feature map into one number while preserving key information?

Concept: Global Pooling

Global Pooling reduces each entire feature map into a single value.
Types:

- **Global Average Pooling (GAP):**

$$y_c = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} F_c(i,j)$$

- **Global Max Pooling (GMP):**

$$y_c = \max_{i,j} F_c(i,j)$$

Here $F_c(i,j)$ is the activation at location (i,j) in channel c.
Key property: Output dimension depends only on the number of channels, not spatial size.

Global Pooling

TensorFlow: Global Average Pooling Example

```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu',
                           input_shape=(28,28,1)),
    tf.keras.layers.GlobalAveragePooling2D()
])

model.summary()
# Expected: Output shape is (None, 32)
# Each of 32 feature maps reduced to 1 number
```

Real-World Examples

- **Image classification:** replaces fully connected layers, reducing parameters.
- **Medical imaging:** each channel summarizes one diagnostic pattern.
- **MobileNets / EfficientNet:** rely on GAP to achieve lightweight architectures.

Global Pooling

Common Mistakes

- Confusing global pooling with normal pooling (global pooling always outputs 1×1 per channel).
- Using global pooling when fine spatial detail is needed (e.g., segmentation).

Caveats

- Removes all spatial information — cannot recover positions of features.
- Works best for classification but not for tasks like detection or segmentation.

Global Pooling as Taking a Final Exam Grade

Imagine a student's grades across many assignments (pixels). - **Global Average Pooling:** compute the average grade — overall performance. - **Global Max Pooling:** keep the highest grade achieved — best performance highlight.

Get more info $\rightarrow$ **Keras: GlobalAveragePooling2D**

Summary: Global Pooling

Concept	Key Point
Global pooling definition	Reduces entire feature map to one value per channel
Global Average Pooling (GAP)	Computes average of all spatial values
Global Max Pooling (GMP)	Takes maximum value across spatial map
Output size	Depends only on channels (e.g., C feature maps $\rightarrow$ vector of length C)
Benefits	Fewer parameters, prevents overfitting, used in lightweight models
Caveats	Loses spatial information, unsuitable for detection/segmentation

Current Section

The Fully Connected Layer in CNNs

Imagine analyzing MRI images: convolutional and pooling layers extract edges, shapes, and textures, but the system still needs to make the final decision (e.g., tumor present or not). **Question:** How do we connect extracted spatial features into a final classification or regression output?

Concept: Fully Connected Layer

A **Fully Connected (FC) layer** connects every neuron from the previous layer to every neuron in the next layer.
Mathematically, for input $x \in \mathbb{R}^n$:

$$y = f(Wx + b)$$

where:

- $W \in \mathbb{R}^{m \times n}$ is the weight matrix,
- $b \in \mathbb{R}^m$ is the bias vector,
- $f(\cdot)$ is an activation function (ReLU, softmax, etc.).

Key Role:

- Acts as a "decision-making" layer, combining features into class scores.
- Translates high-level feature maps into task-specific predictions.

The Fully Connected Layer in CNNs

TensorFlow: Fully Connected Layer Example

```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu',
                           input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Flatten(),                    # Flatten feature maps
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()
# Output: Dense(128) + Dense(10) layers after feature extraction
```

Real-World Examples

- **VGG16, AlexNet, LeNet:** end with fully connected layers for classification.
- **Medical imaging:** final FC layers decide between “healthy” vs. “disease.”
- **Speech recognition:** dense layers map extracted features to phoneme classes.

The Fully Connected Layer in CNNs

Common Mistakes

- Flattening too early in the network, losing spatial context prematurely.
- Using excessively large FC layers $\rightarrow$ exploding parameter count.

Caveats

- FC layers often introduce millions of parameters (risk of overfitting).
- Require regularization (dropout, weight decay) to generalize well.
- Modern CNNs often replace large FC layers with Global Average Pooling to reduce complexity.

Get more info $\rightarrow$ TensorFlow: Dense Layer

Summary: Fully Connected Layer in CNNs

Concept	Key Point
Fully connected definition	Each neuron connects to all outputs of the previous layer
Formula	$y = f(Wx + b)$, where W are weights and f is activation
Role in CNNs	Combines extracted features into final predictions
Benefits	Powerful representation, can capture complex feature interactions
Drawbacks	Millions of parameters, prone to overfitting, less efficient
Modern alternatives	Global Average Pooling to reduce parameters while preserving performance

Current Section

Parameter Counting in CNNs

When building CNNs, it is essential to know how many trainable parameters each layer has. We compare two small networks: one ending with a fully connected layer, the other using Global Average Pooling.

—

General Formulas:

1. **Conv2D layer:**

$$\text{Params} = N_f \times (k_h \times k_w \times C_{\text{in}} + 1)$$

- N_f = number of filters (output channels) - k_h, k_w = kernel height and width - C_{in} = number of input channels - The $+1$ accounts for the bias term of each filter

2. **Dense (Fully Connected) layer:**

$$\text{Params} = (n_{\text{in}} \times n_{\text{out}}) + n_{\text{out}}$$

- n_{in} = number of input features - n_{out} = number of output neurons - The $+n_{\text{out}}$ is for biases (one per neuron)

3. **Pooling layers (MaxPooling, AveragePooling, Global Average Pooling):**

$$\text{Params} = 0$$

Parameter Counting in CNNs

Pooling only applies fixed mathematical operations and does not learn weights.

—

Example Networks:

Model A: With Fully Connected Layer

```
model_a = tf.keras.Sequential([
    tf.keras.layers.Conv2D(8, (3,3), activation='relu',
                           input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D((2,2), stride=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
model_a.summary()
```

Step-by-step parameter count:

- Conv2D(8 filters, 3×3 kernel, 1 input channel):

$$8 \times (3 \times 3 \times 1 + 1) = 8 \times (9 + 1) = 80$$

- After pooling: feature map size = $13 \times 13 \times 8 = 1352$

Parameter Counting in CNNs

- Dense(32 neurons):

$$(1352 \times 32) + 32 = 43{,}296$$

- Dense(10 neurons):

$$(32 \times 10) + 10 = 330$$

Total parameters (Model A):

$$80 + 43{,}296 + 330 = 43{,}706$$

—

Model B: With Global Average Pooling

```
model_b = tf.keras.Sequential([
    tf.keras.layers.Conv2D(8, (3,3), activation='relu',
                           input_shape=(28,28,1)),
    tf.keras.layers.MaxPooling2D((2,2), stride=2),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(10, activation='softmax')
])
model_b.summary()
```

Step-by-step parameter count:

Parameter Counting in CNNs

- Conv2D(8 filters, 3×3 kernel, 1 input channel):

$$8 \times (3 \times 3 \times 1 + 1) = 8 \times 10 = 80$$

- GAP layer:

$$\text{Params} = 0$$

  Output size = 8 (one per channel)
- Dense(10 neurons):

$$(8 \times 10) + 10 = 90$$

Total parameters (Model B):

$$80 + 0 + 90 = 170$$

—

Comparison:

- Model A (with fully connected hidden layer): 43,706 parameters
- Model B (with Global Average Pooling): 170 parameters

This demonstrates why modern CNNs often prefer **Global Average Pooling** — it reduces the parameter count dramatically, lowering memory cost and risk of overfitting.