UNIVERSITÀ DEGLI STUDI
DI MILANO

Advanced Analysis of Misclassification

Francesco Pineschi

23/06/2024

Contents

1 Deep Learning e Neural Networks

Il *deep learning* è una branca dell'intelligenza artificiale che utilizza reti neurali profonde per analizzare dati complessi. Queste reti, composte da molti strati di *neuroni*, apprendono autonomamente a riconoscere pattern e a fare previsioni, rendendole particolarmente efficaci in compiti come riconoscimento immagini e linguaggio naturale.

1.1 Cos'è una rete neurale artificiale?

Una rete neurale è un insieme di strati sequenziali di neuroni, dove:

- Ogni neurone riceve input numerici, applica una funzione (spesso non lineare) e passa il risultato ai neuroni del layer successivo.
- La rete è formata da *pesi* (i parametri del modello), che sono numeri associati ai collegamenti tra i neuroni. Durante il training, i pesi vengono aggiornati per minimizzare l'errore tra il risultato prodotto dalla rete e i dati reali.

Una rete neurale è quindi un'architettura che apprende autonomamente a risolvere problemi complessi, come il riconoscimento delle immagini, l'analisi del linguaggio naturale o il controllo di robot.

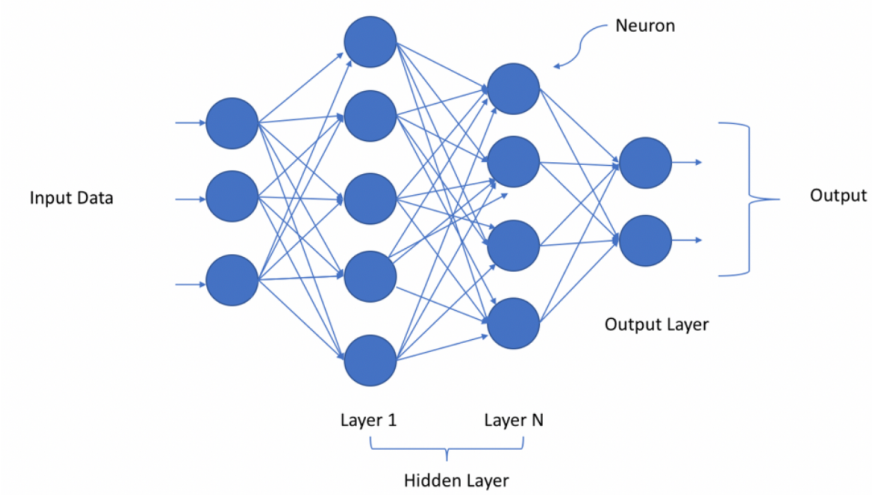


Figure 1: Struttura generica di una deep neural network.

1.2 Struttura dei layer in una rete neurale

- **Input Layer (Strato di Input):** È lo strato che riceve i dati grezzi in input (ad esempio, pixel di un'immagine o parametri numerici). Questo

strato non effettua calcoli, ma distribuisce i dati agli strati successivi.

- **Hidden Layers (Strati Nascosti):** Sono strati intermedi che elaborano gli input e ne estraggono caratteristiche rilevanti. Questi strati applicano trasformazioni complesse ai dati grazie all'uso di funzioni di attivazione non lineari (come *ReLU* o *Sigmoid*), permettendo alla rete di apprendere pattern complessi. Più hidden layer sono presenti, più "profonda" è la rete, da cui il termine *deep learning*.
- **Output Layer (Strato di Output):** È l'ultimo strato della rete che produce il risultato finale, come una classe predetta (esempio: "gatto" o "cane" in un classificatore di immagini). La funzione di attivazione usata nell'output dipende dal tipo di problema:
  - *Softmax* per la classificazione multi-classe.
  - *Sigmoid* per la classificazione binaria.
  - Nessuna attivazione per problemi di regressione.

1.3 Come funzionano i layer

Le *feature* (caratteristiche) sono rappresentazioni numeriche che descrivono aspetti rilevanti di un dato, come un'immagine o un testo. In una rete neurale, ogni layer (strato) elabora le feature ricevute dallo strato precedente, estraendo progressivamente rappresentazioni più astratte e complesse.

Questo avviene prendendo i dati di output dal layer precedente (nel caso degli hidden layers) o direttamente dall'input layer, li moltiplica per i pesi (i parametri appresi durante il training), somma un valore di bias, e applica una funzione di attivazione.

Tuttavia, più strati possono aumentare la complessità computazionale e il rischio di *overfitting*, che può essere controllato con tecniche come il *dropout* o la regolarizzazione.
In sintesi, i layer nelle reti neurali di *deep learning* sono unità fondamentali che elaborano e trasformano i dati, consentendo alla rete di apprendere pattern complessi e fare previsioni accurate.

2 Training di una Rete Neurale

Il processo di **training** di una rete neurale consiste nell'aggiornare i pesi dei collegamenti tra i neuroni per minimizzare l'errore tra le predizioni della rete e i valori attesi. Il training si basa su due concetti fondamentali: la *propagazione in avanti* (*forward propagation*) e la *propagazione all'indietro* (*backpropagation*) con l'uso di tecniche di ottimizzazione come la *discesa del gradiente* (*gradient descent*).

2.1 Forward Propagation

Durante la *forward propagation*, l'input attraversa i vari strati della rete, passando per i neuroni che applicano delle funzioni di attivazione ai loro input ponderati.
L'output di un neurone j nel layer l è dato da:

$$a_j^{(l)} = f\left(\sum_i w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}\right)$$

dove:

- $a_i^{(l-1)}$ è l'attivazione del neurone i nel layer precedente $(l-1)$,
- $w_{ij}^{(l)}$ è il peso tra il neurone i del layer $l-1$ e il neurone j nel layer l,
- $b_j^{(l)}$ è il bias del neurone j nel layer l,
- $f(\cdot)$ è la funzione di attivazione (come ReLU, Sigmoid o Tanh).

2.2 Funzione di Costo

La **funzione di costo** quantifica l'errore tra le predizioni della rete e i valori desiderati (target). Per problemi di classificazione, la funzione di costo comune è l'*entropia incrociata* (*cross-entropy*):

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{C} y_{i,c}\log(h_\theta(x_i)_c)$$

dove:

- m è il numero di esempi di addestramento,
- C è il numero di classi,
- $y_{i,c}$ è il valore target per l'esempio i e la classe c,
- θ rappresenta i parametri (pesi) del modello.
- $h_\theta(x_i)_c$ è la probabilità predetta dalla rete per l'esempio i appartenente alla classe c.

2.3 Backpropagation e Discesa del Gradiente

Per minimizzare la funzione di costo, la rete utilizza la **backpropagation**, una tecnica per calcolare i gradienti della funzione di costo rispetto ai pesi della rete.
Il gradiente della funzione di costo $J(\theta)$ rispetto a un peso $w_{ij}^{(l)}$ è dato da:

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)}$$

dove:

- $\delta_j^{(l)}$ è l'errore del neurone j nel layer l,
- $a_i^{(l-1)}$ è l'attivazione del neurone i nel layer precedente.

L'errore $\delta_j^{(l)}$ è calcolato propagando l'errore del layer successivo:

$$\delta_j^{(l)} = \left(\sum_k w_{jk}^{(l+1)} \delta_k^{(l+1)} \right) f'(z_j^{(l)})$$

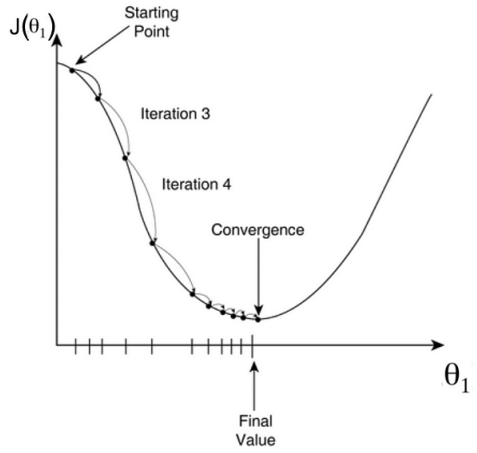
dove $f'(z_j^{(l)})$ è la derivata della funzione di attivazione rispetto all'input ponderato $z_j^{(l)}$.

2.4 Aggiornamento dei Pesi

I pesi vengono aggiornati usando la **discesa del gradiente**. La regola di aggiornamento è:

$$w_{ij}^{(l)} := w_{ij}^{(l)} - \eta \frac{\partial J}{\partial w_{ij}^{(l)}}$$

dove η è il *learning rate*, che controlla la velocità con cui i pesi vengono aggiornati.
Questo processo si ripete iterativamente per ogni batch di dati di addestramento finché il modello convergerà a una soluzione ottimale, minimizzando l'errore della rete.



Cost Function – "One Half Mean Squared Error":

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Objective:

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

Derivatives:

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

Figure 2: Rappresentazione del gradient descent

3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks represent a class of deep learning models specifically used for processing visual data. They have demonstrated remarkable performance in various computer vision tasks, ranging from image classification to object detection and segmentation.

An example of CNN application in the real world includes image classification tasks, where CNNs can accurately categorize images into predefined classes. For instance, in medical imaging, CNNs have been employed to diagnose diseases based on X-ray or MRI scans, showcasing their potential in critical healthcare applications.

Key components of CNNs include convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply learnable filters to input data, capturing local patterns and features. Pooling layers reduce spatial dimensions, preserving important features while reducing computational complexity. Fully connected layers integrate extracted features for classification or regression tasks. By leveraging these components, CNNs can effectively learn representations of image's input data, giving accurate predictions across various domains.

3.1 Convolutional Layers

Convolutional layers are fundamental components of Convolutional Neural Networks. They play a crucial role in feature extraction by applying convolution operations to input images using filters.

- **Filters**: Filters, also known as kernels, are small matrices applied to input images during convolution. Each filter extracts specific features from the input by performing element-wise multiplication and summation operations.
- **Kernel Size**: The kernel size refers to the spatial dimensions of the filter. It determines the receptive field of the filter and influences the types of features extracted. Common kernel sizes include 3x3 and 5x5.
- **Strides**: Strides determine the step size of the filter as it traverses the input image during convolution. A stride of 1 means the filter moves one pixel at a time, while larger strides result in downsampling of the output feature map.
- **Padding**: Padding is the process of adding additional border pixels to the input image before convolution. It helps preserve spatial information and prevent loss of information at the image boundaries. Common padding types include 'valid', which applies no padding, and 'same', which pads the input to ensure the output feature map has the same spatial dimensions as the input.

- **Activation Function**: The activation function introduces non-linearity into the convolutional layer's output. Common activation functions include ReLU (Rectified Linear Unit), which introduces sparsity and addresses the vanishing gradient problem.

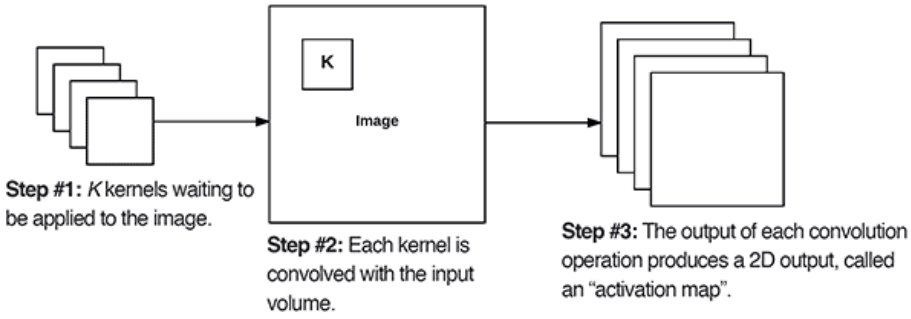


Figure 3: Illustration of a Convolutional Neural Network architecture.

3.2 Pooling Layers

Pooling layers are essential components in Convolutional Neural Networks used for down-sampling and feature reduction. They help reduce the spatial dimensions of the input feature maps, making the network more computationally efficient and reducing overfitting.

Pooling operations typically fall into three main categories:

- **Max Pooling**: Max pooling selects the maximum value from each subregion of the input feature map. It identifies the most significant features while discarding less important ones. Max pooling is the most common type of pooling operation used in CNNs due to its simplicity and effectiveness.
- **Average Pooling**: Average pooling computes the average value from each subregion of the input feature map. It provides a smoothed representation of the features and is less prone to overfitting compared to max pooling.
- **Global Pooling**: Global pooling computes a single value for each feature map by applying a pooling operation across the entire map. It reduces the spatial dimensions to a single value per feature map, often used as the input to the final classification layer of the network.

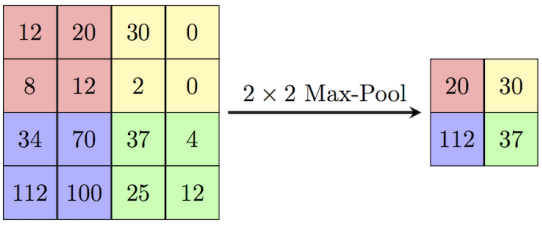


Figure 4: Illustration of Max Pooling (2x2) performed on a matrix (4x4) of integers.

3.3 Fully Connected Layers

Fully connected layers, also known as dense layers, integrate the features extracted by convolutional and pooling layers for final classification or regression tasks. Dense layers connect every neuron in one layer to every neuron in the next layer, enabling global information propagation through the network.
In Keras, fully connected layers are implemented using the `Dense` layer. These layers require input data to be in the form of a one-dimensional vector. However, the output of convolutional and pooling layers is typically a three-dimensional tensor. Therefore, before passing the output to the dense layers, the tensor is flattened into a one-dimensional vector using the `Flatten` layer.
The `Flatten` layer serves the purpose of reshaping the output of the preceding convolutional and pooling layers into a format suitable for input to the dense layers. It collapses the spatial dimensions of the feature maps into a single vector while preserving the relationships between the features.

4 VGG16: Architettura e Utilizzo

VGG16 è un'architettura di rete neurale convoluzionale (CNN) sviluppata da *Visual Geometry Group* (VGG), il dipartimento di scienze ingegneristiche dell'Università di Oxford. La rete è stata presentata e nella copetizione *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* nel 2014. VGG16 è un modello valido e utilizzato nei problemi di Object Recognition per il rapporto tra efficienza e poca profondità della rete (solo 13 convolutional layer).

4.1 Struttura di VGG16

VGG16 è chiamata così perché utilizza 16 livelli (o layer) di profondità, tra cui 13 strati convoluzionali e 3 strati completamente connessi (fully connected). La struttura segue un design semplice e modulare, con filtri convoluzionali fissi di dimensione 3×3, applicati a diverse risoluzioni di input. Ogni strato convoluzionale è seguito da un'operazione di max-pooling, che riduce la dimensione spaziale delle mappe di attivazione, pur mantenendo le caratteristiche rilevanti per la classificazione.

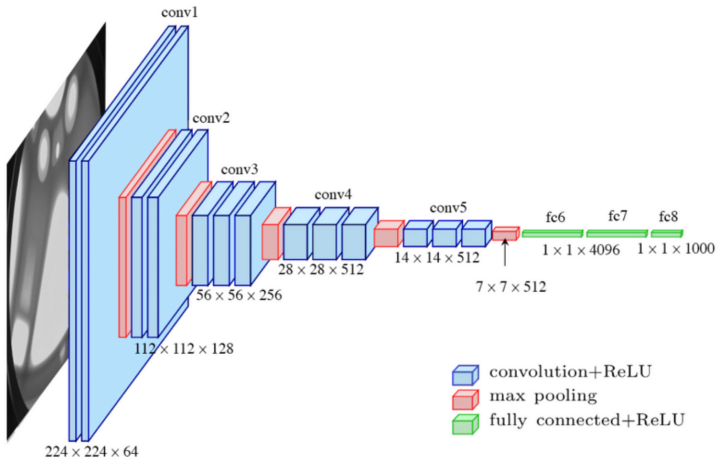


Figure 5: Struttura di VGG16

1. **Primo Stack:**
   - Vengono applicati 64 filtri convoluzionali bidimensionali di dimensione 3x3, con stride 1 e padding 1.
   - L'output, con dimensioni 224x224x64, è successivamente processato da un secondo filtro convoluzionale identico.
   - Le dimensioni dell'output rimangono inalterate, ma vengono ridotte attraverso un *max pooling layer* bidimensionale, che produce un volume finale di 112x112x64.

2. **Secondo Stack:**

    - Il numero di filtri convoluzionali raddoppia: vengono applicati 128 filtri bidimensionali di dimensione 3x3, ottenendo un volume iniziale di 112x112x128.
    - Successivamente, viene applicato un secondo filtro convoluzionale di pari dimensioni.
    - Infine, un *max pooling layer* riduce il volume finale a 56x56x128.

3. **Terzo Stack:**

    - Sull'input proveniente dallo stack precedente vengono applicati 256 filtri convoluzionali bidimensionali di dimensione 3x3.
    - Questa operazione è ripetuta per tre volte.
    - Un *max pooling layer* viene infine utilizzato per ridurre il volume a 28x28x256.

4. **Quarto Stack:**

    - Vengono applicati 512 filtri convoluzionali bidimensionali di dimensione 3x3 per tre volte consecutive.
    - Successivamente, un *max pooling layer* riduce il volume finale a 14x14x512.

5. **Quinto Stack:**

    - Struttura simile al quarto stack: tre layer convoluzionali e un *max pooling layer* riducono l'output a 7x7x512.

6. **Sesto Stack:**

    - Nella parte finale, il primo *fully connected layer* appiattisce il volume 7x7x512 in un vettore di 25,088 neuroni.
    - Questo è collegato a un *dense layer* di 4096 neuroni, seguito da un *dropout layer* con altrettanti neuroni.

7. **Settimo Stack:**

    - È presente un secondo *dense layer* di 4096 neuroni, seguito da un ulteriore *dropout layer*.

8. **Ottavo Stack:**

    - L'architettura si completa con un *output layer* di tipo *dense*, composto da 1000 neuroni per la classificazione finale.

4.2 Utilizzo di VGG16

VGG16 è ampiamente utilizzata per compiti di classificazione delle immagini, riconoscimento oggetti e segmentazione. La sua architettura profonda e semplice la rende una delle reti neurali più facili da comprendere e implementare, ma allo stesso tempo molto efficace per diverse applicazioni nel campo del deep learning. Nonostante l'aumento della complessità delle reti più moderne, VGG16 rimane un modello di riferimento, spesso utilizzato come base per il trasferimento di conoscenze (transfer learning) in vari problemi di visione artificiale.

5 Grad-CAM: Gradient-Weighted Class Activation Mapping

Grad-CAM (Gradient-weighted Class Activation Mapping) è una tecnica di visualizzazione utilizzata per interpretare e spiegare le decisioni di una rete neurale convoluzionale (CNN). Grad-CAM si basa sul calcolo dei gradienti della classe di output rispetto alle feature map di un layer convoluzionale, al fine di generare mappe di attivazione che evidenziano le regioni rilevanti dell'immagine per una determinata predizione.

Consideriamo un esempio di applicazione di Grad-CAM su un modello addestrato a riconoscere immagini di animali.

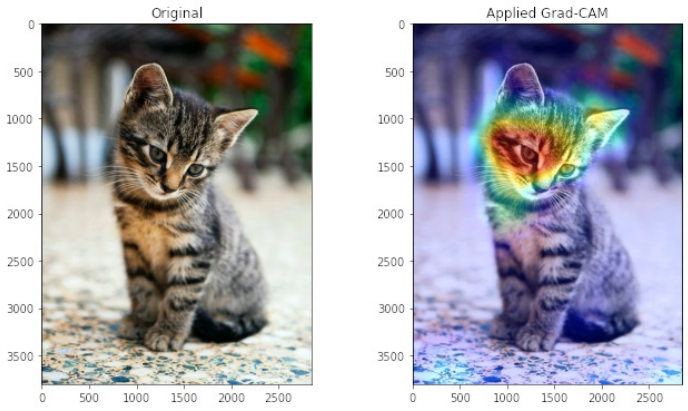


Figure 6: A sinistra: immagine di input (gatto). A destra: mappa di attivazione Grad-CAM sovrapposta.

Il modello predice correttamente "gatto" con una probabilità del 90%. La mappa di attivazione Grad-CAM evidenzia principalmente la testa del gatto, indicando che questa regione ha contribuito maggiormente alla predizione.

5.1 Come funziona Grad-CAM

Il metodo Grad-CAM sfrutta i gradienti delle feature map rispetto alla predizione di una classe target per capire quali regioni dell'immagine influenzano maggiormente la decisione finale della rete neurale. Il funzionamento può essere suddiviso nei seguenti passi:

1. **Passare l'immagine input attraverso la rete CNN:** L'immagine input viene fornita al modello (Forward pass), e il modello calcola la predizione. L'output finale è generalmente una classe o una probabilità associata a diverse classi. Ad esempio, se il modello è una rete addestrata per riconoscere oggetti nelle immagini, potrebbe predire "gatto" con probabilità 0.9, oppure "cane" con probabilità 0.1.

2. **Calcolo dei gradienti:** Si calcolano i gradienti della predizione della classe target rispetto alle feature map di un layer convoluzionale intermedio. Questi gradienti rappresentano quanto varia l'output di quella classe se viene leggermente modificata una particolare feature map. Più alto è il gradiente, maggiore è l'importanza di quella feature map per la predizione.

3. **Ponderare le feature map:** Le feature map vengono ponderate in base all'importanza calcolata, cioè in base ai gradienti mediati su ciascuna mappa. Questa operazione genera i coefficienti α_k^c, che rappresentano il peso della k-esima feature map A^k rispetto alla classe target c.

4. **Creare la mappa di attivazione:** Le feature map ponderate vengono sommate per ottenere una mappa di attivazione combinata. Questa mappa rappresenta visivamente le regioni dell'immagine che hanno avuto la maggiore influenza sulla predizione della classe target. Solo le regioni positive vengono mantenute utilizzando la funzione ReLU, che elimina i valori negativi, preservando le attivazioni che hanno contribuito positivamente.

5. **Visualizzazione della mappa Grad-CAM:** La mappa di attivazione viene sovrapposta all'immagine originale per produrre una visualizzazione interpretabile. Le regioni evidenziate in rosso corrispondono alle parti dell'immagine che hanno maggiormente influenzato la predizione del modello.

5.2 Formula di Grad-CAM

La formula matematica di Grad-CAM è:

$$L^c_{\text{Grad-CAM}} = \text{ReLU}\left(\sum_k \alpha_k^c A^k\right)$$

Dove:

- $L^c_{\text{Grad-CAM}}$ è la mappa di attivazione ponderata per la classe c.

- A^k è la k-esima feature map del layer convoluzionale selezionato.
- α_k^c è il coefficiente che misura l'importanza della feature map A^k per la classe c.

Il coefficiente α_k^c viene calcolato come:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$$

Dove:

- $\frac{\partial y^c}{\partial A_{ij}^k}$ è il gradiente della predizione della classe c rispetto all'attivazione A^k alla posizione (i, j).
- Z è il numero totale di elementi nella feature map A^k, ossia il prodotto tra altezza e larghezza della mappa: $Z = H \times W$.

Questa formula indica che il contributo di ciascuna feature map A^k è ponderato sulla base dei gradienti spazialmente mediati rispetto alla classe c. Infine, viene applicata la funzione ReLU per rimuovere i valori negativi, lasciando solo le regioni con influenza positiva.

5.3 Differenza tra i gradienti calcolati in Grad-CAM e durante il training

I gradienti calcolati in Grad-CAM e quelli utilizzati durante il training di una rete neurale convoluzionale servono a scopi differenti e non devono essere confusi.

1. **Gradiente durante il training:** Durante l'addestramento di una rete neurale, i gradienti sono utilizzati per aggiornare i pesi del modello tramite il processo di backpropagation. Questi gradienti indicano quanto ogni peso della rete deve essere modificato per ridurre l'errore totale, ovvero la differenza tra le predizioni del modello e le etichette corrette. L'aggiornamento dei pesi segue la direzione opposta ai gradienti per minimizzare la funzione di costo.
2. **Gradiente in Grad-CAM:** In Grad-CAM, i gradienti non sono usati per aggiornare i pesi del modello, ma per identificare l'importanza delle *feature maps* rispetto alla predizione di una classe specifica. I gradienti vengono calcolati rispetto alle attivazioni di un determinato livello convoluzionale e l'output di una classe. Servono a localizzare visivamente le aree dell'immagine che maggiormente influenzano la predizione del modello per quella classe.

6 Grad-CAM IntraLayer

Grad-CAM IntraLayer estende il metodo Grad-CAM trattando i layer intermedi della rete convoluzionale come se fossero layer finali. Questo approccio consente di ottenere mappe di attivazione non solo per l'ultimo strato convoluzionale, ma anche per i layer intermedi, fornendo una visione più dettagliata delle feature estratte dalla rete a diverse profondità.

6.1 Come funziona Grad-CAM IntraLayer

Il principio di Grad-CAM IntraLayer è calcolare i gradienti della classe predetta rispetto alle feature map dei layer intermedi, pesare le attivazioni di questi layer in base all'importanza attribuita dai gradienti, e visualizzare le aree più rilevanti dell'immagine in termini di contributo alla predizione.
In Grad-CAM tradizionale, i gradienti sono calcolati solo per l'ultimo strato convoluzionale. In Grad-CAM IntraLayer, si applica lo stesso processo ai layer intermedi, trattandoli come se fossero layer finali. La mappa di attivazione generata per ciascun layer intermedio rappresenta le regioni dell'immagine che contribuiscono maggiormente all'output predetto in quel layer.

6.2 Applicazione di Grad-CAM IntraLayer

Utilizzando Grad-CAM IntraLayer, ogni layer convoluzionale viene trattato come un potenziale punto di uscita per l'analisi delle attivazioni. Le mappe di attivazione generate per ciascun layer intermedio forniscono informazioni su quali regioni dell'immagine vengono considerate importanti a diversi livelli di astrazione.

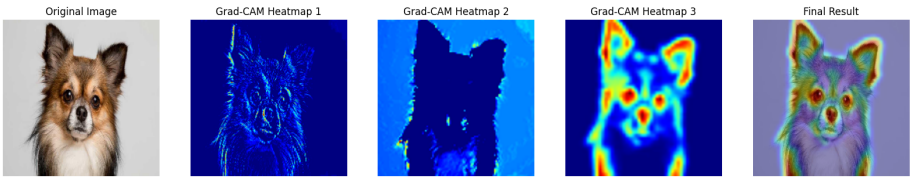


Figure 7: Mappe di attivazione Grad-CAM IntraLayer per tre layer convoluzionali.

6.3 Vantaggi di Grad-CAM IntraLayer

- **Maggiore interpretabilità**: Grad-CAM IntraLayer offre una visione dettagliata delle attivazioni a diversi livelli della rete, consentendo una comprensione più granulare del comportamento del modello.
- **Diagnosi approfondita del processo di classificazione**: Questo metodo consente di visualizzare non solo dove il modello ha fatto la predizione finale, ma anche in quali layer intermedi ha iniziato a prendere decisioni. È

particolarmente utile per individuare il punto in cui il modello potrebbe iniziare a commettere errori. Per esempio, se le attivazioni di layer più profondi iniziano a concentrarsi su aree irrilevanti dell'immagine, possiamo identificare dove il modello "sbaglia" nel processo di classificazione.

- **Analisi delle feature intermedie**: Permette di visualizzare l'importanza di feature estratte nei livelli intermedi, che possono catturare rappresentazioni intermedie rilevanti come bordi, texture o forme.

References

[1] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, 2016. ISBN: 978-0262035613

[2] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, *Deep Learning*, Nature, vol. 521, no. 7553, pp. 436–444, 2015

[3] Selvaraju, R.R. et al. *Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization*, International Journal of Computer Vision, 2019

[4] K. Simonyan, A. Zisserman *Very Deep Convolutional Networks for Large-Scale Image Recognition* International Conference on Learning Representations, 2015