
**UNIVERSITÀ DEGLI STUDI DI PADOVA**

**Dipartimento di Ingegneria Dell'Informazione**

---

Corso di Laurea in

**Ingegneria Informatica**

# LYMPHOMA: DATA AUGUMENTATION

*Studente:*
**Nicola Lorenzon**

*Anno accademico 2021/2022*

# Contents

# 1 Introduzione

## 1.1 Data Augumentation

L'obiettivo di questo elaborato è quello di presentare due modelli di *data augumentation* sviluppati ad hoc per un problema di classificazione di immagini mediche.
In campo medico spesso vi è la necessità di creare *pattern artificiali* mediante tecniche di Data Augumentation, vista la scarsa mole di dati, con i quali fare il training dei modelli di machine learning.
In questo caso, il dataset utilizzato è *DatasColor_29*.
DatasColor_29 contiente 374 immagini, ognuna rappresentante un linfoma appartenente ad una delle seguenti classi:

- CLL = Chronic Lymphocytic Leukemia
- FL = Follicular Lymphoma
- MCL = Mantle Cell Lymphoma

## 1.2 Modello

Il modello utilizzato per la classificazione dei linfomi è AlexNet.
AlexNet è una CNN (Convolunational Neural Network) formata da 8 layer. Il modello che ci mette a disposizione MATLAB, è pretrained pretrained in un dataset con un milione di pattern circa, in grado di classificare immagini appartenenti a più di mille classi.
Il modello accetta in input pattern di dimensione fissata 277x277 px

Per utilizzare questa rete pretrained a nostro vantaggio, si compie un'azione di *Transfer Learning*.
Il Transfer Learning è un metodo molto utilizzato per adattare un modello pretrained al proprio caso di studio.
L'idea è quella di togliere dal modello già addestrato gli ultimissimi layer (che compongono la classificazione vera e propria) e sostituirli con dei layer ad-hoc, cos' facendo lo sforzo computazionale per fare il training della rete sarà molto minore, si otterrà la convergenza più velocemente dal momento che i pesi non sono inizializzati *random*.

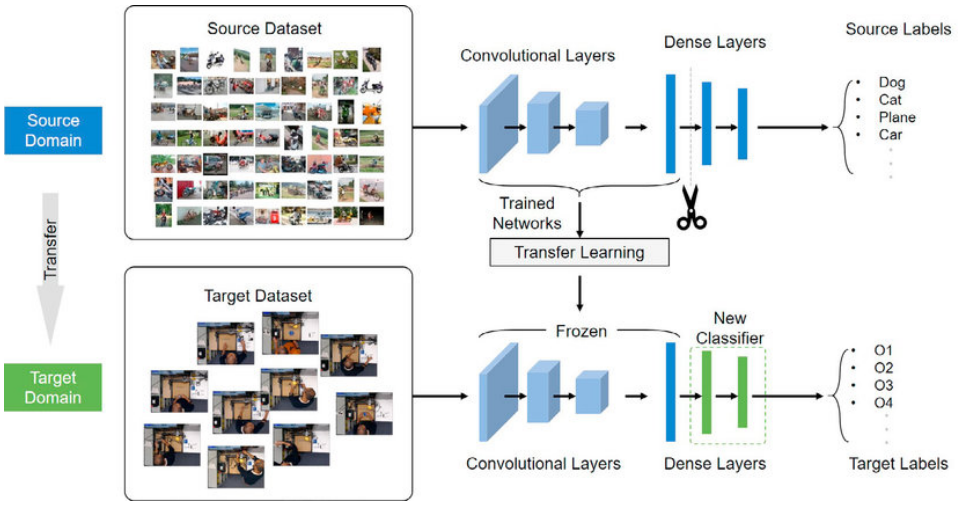


Figure 1.1: Transfer Learning

## 1.3 Train e Test

La parte di train e test è stata svolta mediante k-fold cross validation, con cinque fold. K-fold cross validation è utile in generale quando si hanno pochi pattern per evitare problemi di overfitting della rete, ovvero quando i pesi sono settati estremamente bene per il numero ristretto di pattern di training ma che, non essendo suffecentemente grande, non generalizza bene il problema, in questo modo nuovi pattern saranno probabilmente classificati male.
Saranno riportate, per ogni metodo di data augumentation utilizzato le dimensioni del training set e del test set.

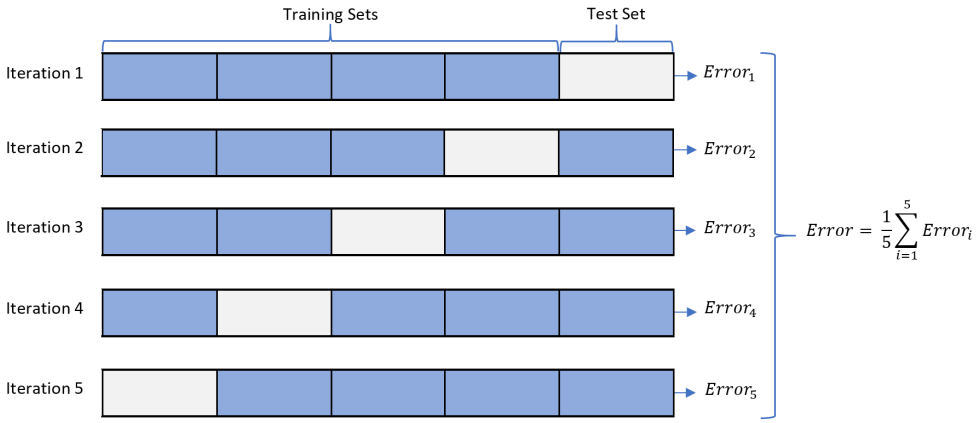


Figure 1.2: K-Fold

# 2 Codice e Risultati

Questo capitolo si divide in tre parti, corrispondenti ai tre metodi utilizzati per fare data augumentation sul dataset in questione.

## 2.1 Prima Funzione

Questa sezione illustrerà i risultati ottenuti dalla rete neurale addestrata sul dataset base, al quale sono applicate delle semplici trasformate fornite dal *Deep Learning Toolbox* quali:

Riflessione rispetto all'asse X e Y
Ridimensionamento in base all'asse X e Y
Rotazione compresa fra -10 e +10 gradi
Traslazioni comprese fra 0 e 5 pixel sull'asse X e Y

Di seguito, è riportato il codice utilizzato:

```
%creazione pattern aggiuntivi mediante tecnica standard
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXScale',[1 2], ...
    'RandYReflection',true, ...
    'RandYScale',[1 2],...
    'RandRotation',[-10 10],...
    'RandXTranslation',[0 5],...
    'RandYTranslation', [0 5]);
trainingImages = augmentedImageSource(imageSize,trainingImages,categorical(y'),'DataAugmentation',imageAugmenter);
```

Figure 2.1: Codice Prima Funzione

Di seguito sono riportati i risultati ottenuti in sia in training con [2.2] che in test con la tabella [2.1]

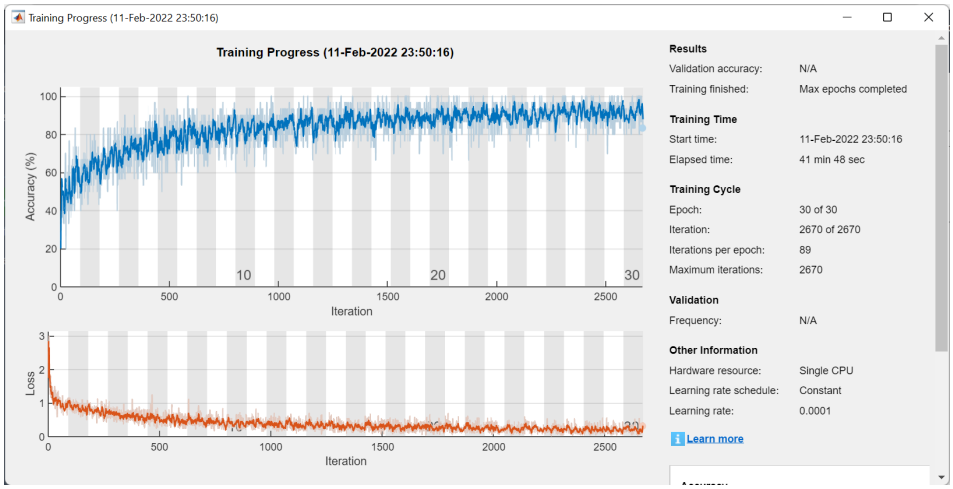


Figure 2.2: Risultati Prima Funzione

| **Fold** | **Accuracy** |
|---|---|
| Fold 1 | 0.7867 |
| Fold 2 | 0.6667 |
| Fold 3 | 0.7467 |
| Fold 4 | 0.8000 |
| Fold 5 | 0.7067 |

Table 2.1: Risultati Prima Funzione

Si ha quindi una media di accuracy del 74.13%

## 2.2 Seconda Funzione

La seconda funzione presentata prende spunto dal paper [5].
L'idea è quella di applicare delle trasformate ad hoc alle immagini, generandone 8 per ognuna.
Di seguito il codice matlab utilizzato per la realizzazione delle immagini.

```
function [trainImages,trainLabels] = foo2(trainImages,trainLabels)
k = length(trainImages(1,1,1,:));
for i = 1:length(trainImages(1,1,1,:))
    for j = 1:8
        clear augImg
        augImg(:,:,:,j) = trainImages(:,:,:,i);
        % Specchia l'immagine con 50% prob su assi X e Y
        if randi(2) == 1
            augImg(:,:,:,j) = flipdim(augImg(:,:,:,j),2);
        end
        if randi(2) == 1
            augImg(:,:,:,j) = flipdim(augImg(:,:,:,j),1);
        end

        % Rotazone di k*90 con k = (0, 1, 2, 3)
        augImg(:,:,:,j) = imrotate(augImg(:,:,:,j),90*randi([0,3]));

        % Ritaglia l'immagine con una probabilità del 50%
        if randi(2) == 1
            rect = randomWindow2d([227 227],"Scale",[0.4 1],"DimensionRatio",[1 1; 1 1]);
            helpImg = imcrop(augImg(:,:,:,j),rect);
            augImg(:,:,:,j) = imresize(helpImg,[227 227]);
        end
        % Aggiungo le imm al training set
        trainImages(:,:,:,k+j) = augImg(:,:,:,j);
        trainLabels(k+j) = trainLabels(i);
    end
    k = k + j;
end
end
```

Figure 2.3: Codice seconda funzione

Di seguito sono riportati i risultati ottenuti in sia in training con [2.4] che in test con la tabella [2.2]

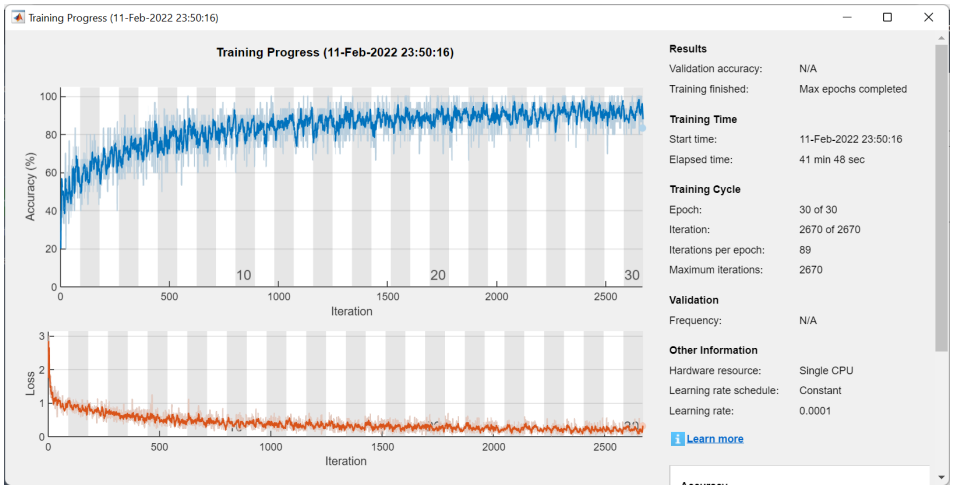


Figure 2.4: Risultati Seconda Funzione

| Fold | Accuracy |
|---|---|
| Fold 1 | 0.8667 |
| Fold 2 | 0.8667 |
| Fold 3 | 0.8000 |
| Fold 4 | 0.8667 |
| Fold 5 | 0.8000 |

Table 2.2: Risultati Seconda Funzione

Si ha quindi una media di accuracy del 84.00%

## 2.3 Terza Funzione

La terza funzione prende spunto dal paper [5] e consiste in un'ensemble di tre funzioni per image augumentation.
Per ogni immagine ne vengono fatte tre, aggiungendo con probabilità 50%:
Un rumore gaussiano basso, $\mu = 0, \sigma^2 = 0.1$
Una rotazione di $k * 90°$ con $k \in [1, 2, 3]$
Una specchiatura secondo l'asse X e/o Y
Di seguito il codice matlab utilizzato per la realizzazione delle immagini.

```
function [trainImages,trainLabels] = foo3(trainImages,trainLabels)
k = length(trainImages(1,1,1,:));
for i = 1:length(trainImages(1,1,1,:))
    for j = 1:3
        clear augImg
        augImg(:,:,:,j) = trainImages(:,:,:,i);
        % Specchia l'immagine con 50% prob su assi X e Y
        if randi(2) == 1
            augImg(:,:,:,j) = flipdim(augImg(:,:,:,j),2);
        end
        if randi(2) == 1
            augImg(:,:,:,j) = flipdim(augImg(:,:,:,j),1);
        end
        % Rotazone di k*90 con k = (0, 1, 2, 3)
        augImg(:,:,:,j) = imrotate(augImg(:,:,:,j),90*randi([0,3]));
        % Gaussian noise
        if randi(2) == 1
            augImg = imnoise(augImg,'gaussian', 0, 0.1);
        end
        % Aggiungo le imm al training set
        trainImages(:,:,:,k+j) = augImg(:,:,:,j);
        trainLabels(k+j) = trainLabels(i);
    end
    k = k + j;
end
end
```

Figure 2.5: Codice Terza funzione

Di seguito sono riportati i risultati ottenuti in sia in training con [2.6] che in test con la tabella [2.3]

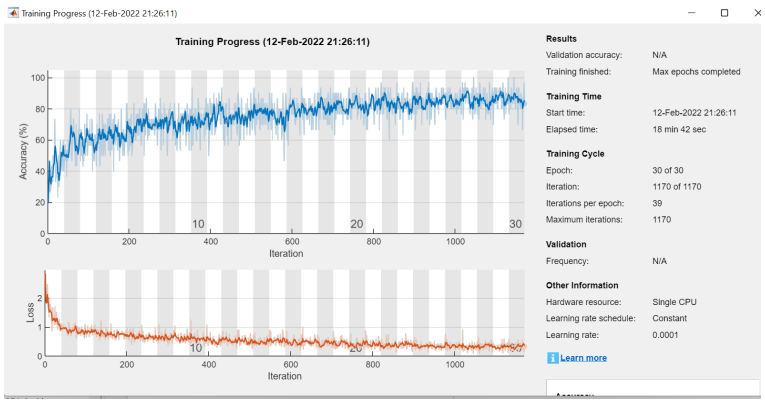


Figure 2.6: Risultati Terza Funzione

| **Fold** | **Accuracy** |
|---|---|
| Fold 1 | 0.8133 |
| Fold 2 | 0.8400 |
| Fold 3 | 0.8267 |
| Fold 4 | 0.8800 |
| Fold 5 | 0.8533 |

Table 2.3: Risultati Terza Funzione

Si ha quindi una media di accuracy del 84.27%

## 2.4 Quarta Funzione

In questo caso è stata aggiunto, oltre a specchiare e ruotare, del rumore 'Salt  Pepper'. Di seguito il codice matlab utilizzato per la realizzazione delle immagini.

```
function [trainImages,trainLabels] = foo4(trainImages,trainLabels)

k = length(trainImages(1,1,1,:));

for i = 1:length(trainImages(1,1,1,:))

    for j = 1:3
        clear augImg
        augImg(:,:,:,j) = trainImages(:,:,:,i);

        % Specchia
        if randi(2) == 1
            augImg(:,:,:,j) = flipdim(augImg(:,:,:,j),2);
        end
        if randi(2) == 1
            augImg(:,:,:,j) = flipdim(augImg(:,:,:,j),1);
        end

        % Ruota
        augImg(:,:,:,j) = imrotate(augImg(:,:,:,j),90*randi([0,4]));

        % Salt & Pepper noise
        augImg(:,:,:,j) = imnoise(augImg(:,:,:,j), 'salt & pepper', 0.005);

        % Saturation, Contrast, Brightness
        if randi(2) == 1
            augImg(:,:,:,j) = jitterColorHSV(augImg(:,:,:,j),'Contrast',0.8,'Saturation',0.2,'Brightness',0.3);
        end

        % Aggiungo le imm al training set
        trainImages(:,:,:,k+j) = augImg(:,:,:,j);
        trainLabels(k+j) = trainLabels(i);
    end
    k = k + j;
end

end
```

Figure 2.7: Codice quarta funzione

Di seguito sono riportati i risultati ottenuti in sia in training con [2.8] che in test con la tabella [2.4]


Figure 2.8: Risultati Quarta Funzione

| **Fold** | **Accuracy** |
|---|---|
| Fold 1 | 0.8400 |
| Fold 2 | 0.7600 |
| Fold 3 | 0.8000 |
| Fold 4 | 0.8000 |
| Fold 5 | 0.8000 |

Table 2.4: Risultati Quarta Funzione

Si ha quindi una media di accuracy del 80.00%

## 2.5 Quinta Funzione

In questo caso si ha testato un approccio con la trasformata discreto coseno DCT.
In questo primo approccio, la funzione è stata applicata in modo standard, andando poi ad azzerare alcuni valori (in modo random) e facendo la trasformata inversa.
Sono state applicate successivamente delle rotazioni e specchiature con una certa probabilità.
Si è preso spunto da [1], [2] e [3].
Di seguito il codice matlab utilizzato per la realizzazione delle immagini.

```
function [trainImages,trainLabels] = foo5(trainImages,trainLabels)
k = length(trainImages(1,1,1,:));
% for image warping:
for i = 1:length(trainImages(1,1,1,:))
    for j = 1:2
        clear augImg
        augImg(:,:,:,j) = trainImages(:,:,:,i);
        for i=1:3
            % Applico DCT all'imagine
            DCT= dct2(augImg(:,:,i,j));
            d=DCT;
            % Metto a 0 qualche pixel random
            DCT(randi([0 1], size(DCT,1),size(DCT,2))==0)=0;
            % Non modifico il pixel (1,1)
            DCT(1,1)=d(1,1);
            % Applico la inversa DCT
            image(:,:,i)= idct2(DCT);
        end
        image=uint8(image);

        % con una prob dell'80% viene fatta una rotazione
        if rand < 0.8
            image = imrotate(image, 90*randi([0,3]));
        end
        % con una prob del 50% viene fatto un flip, prima sull'asse Y poi
        % sull'asse X
        if rand < 0.5
            image = flipdim(image, 2);
        end
        if rand < 0.5
            image = flipdim(image, 1);
        end

        % Aggiungo le imm al training set
        trainImages(:,:,:,k+j) = image;
        trainLabels(k+j) = trainLabels(i);
    end
    k = k + j;
end
end
```

Figure 2.9: Codice Quinta Funzione

Di seguito sono riportati i risultati ottenuti in sia in training con [2.10] che in test con la tabella [2.5]

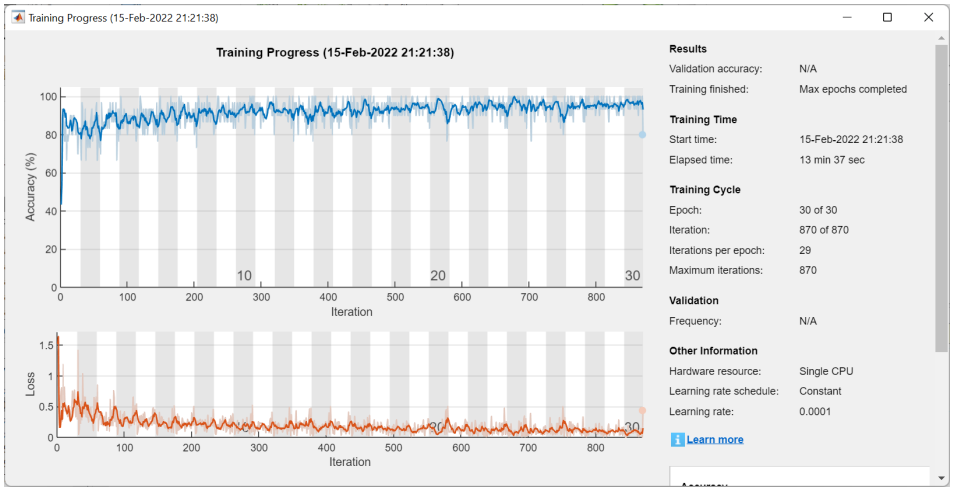


Figure 2.10: Risultati Quinta Funzione

| **Fold** | **Accuracy** |
|---|---|
| Fold 1 | 0.7867 |
| Fold 2 | 0.8133 |
| Fold 3 | 0.7663 |
| Fold 4 | 0.9067 |
| Fold 5 | 0.7733 |

Table 2.5: Risultati Quinta Funzione

Si ha quindi una media di accuracy del 80.93%

## 2.6 Sesta Funzione

Dopo aver riscontrato [4] che:
$DCT[DCT(im)] \simeq im$
Si generano 4 immagini per ognuna.
Si è preso spunto da [1], [2] e [3].
Si ha provato ad usare questo approccio con il codice che segue:

```
function [trainImages,trainLabels] = foo6(trainImages,trainLabels)
k = length(trainImages(1,1,1,:));
for i = 1:length(trainImages(1,1,1,:))
    for j = 1:4
        clear augImg
        augImg(:,:,:,j) = trainImages(:,:,:,i);
        for i=1:3

            % Applica DCT all'immagine
            DCT= dct2(augImg(:,:,i,j));
            d = DCT;
            % Metti a 0 qualche pixel
            DCT(randi([0 1], size(DCT,1),size(DCT,2))==0)=0;
            % Lascia non modificato il primo pixel (1,1)
            DCT(1,1)=d(1,1);

            % Riapplica DCT
            im(:,:,i) = dct2(DCT)

        end
        image=uint8(im);
        trainImages(:,:,:,k+j) = im;%im(:,:,:);
        trainLabels(k+j) = trainLabels(i);
    end
    k = k+j;
end

end
```

Figure 2.11: Codice sesta funzione

Di seguito sono riportati i risultati ottenuti in sia in training con [2.12] che in test con la tabella [2.6]

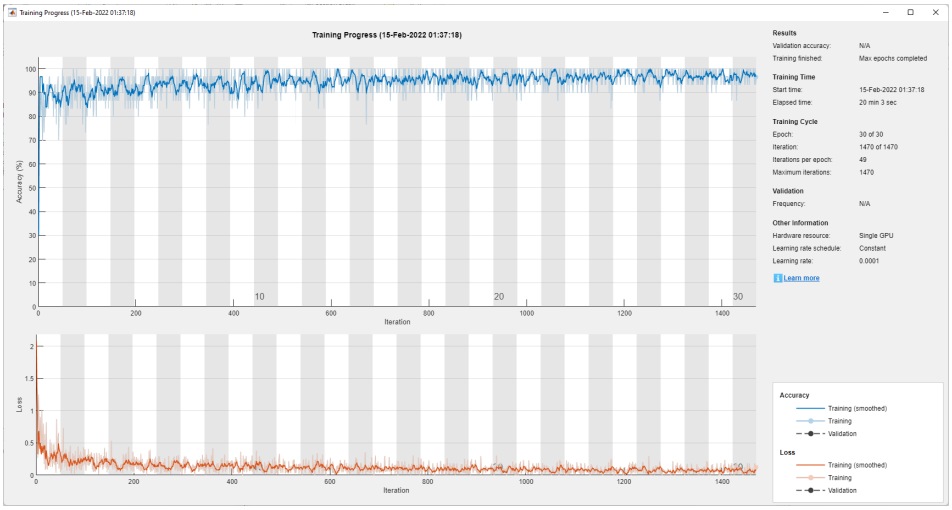


Figure 2.12: Risultati Sesta Funzione

| **Fold** | **Accuracy** |
|---|---|
| Fold 1 | 0.7867 |
| Fold 2 | 0.7600 |
| Fold 3 | 0.7333 |
| Fold 4 | 0.7600 |
| Fold 5 | 0.7800 |

Table 2.6: Risultati Sesta Funzione

Si ha quindi una media di accuracy del 76.40%
Dai risultati sembra che il training set non generalizzi bene il problema e, sebbene la rete si addestri facilmente, non riesce poi a classificare in modo corretto.

## 2.7 Conclusioni

Di seguito una tabella [2.7] che riassume quali sono stati i risultati per ogni funzione con il relativo numero di pattern nel training set:

| **Fold** | **Num Pattern Training Set** | **Accuracy** |
|---|---|---|
| Foo1 | 270 | 0.7413 |
| Foo2 | 2670 | 0.8400 |
| Foo3 | 1170 | 0.8427 |
| Foo4 | 1170 | 0.8000 |
| Foo5 | 870 | 0.8093 |
| Foo6 | 1470 | 0.7640 |

Table 2.7: Overview Risultati

# Bibliography


[1] L. Nanni et al. *General Purpose (GenP) Bioimage Ensemble of Handcrafted and Learned Features with Data Augmentation*. 2021. arXiv: 1904.08084 [cs.CV].

[2] Loris Nanni et al. "Comparison of Different Image Data Augmentation Approaches". In: *Journal of Imaging* 7 (Nov. 2021), p. 254. DOI: 10.3390/jimaging7120254.

[3] Jia Shijie et al. "Research on data augmentation for image classification based on convolution neural networks". In: *2017 Chinese Automation Congress (CAC)*. 2017, pp. 4165–4170. DOI: 10.1109/CAC.2017.8243510.

[4] "The magnitude of taking discrete cosine transform of an image two times is similar to the original". In: (2017). URL: https://dsp.stackexchange.com/questions/34821/the-magnitude-of-taking-discrete-cosine-transform-of-an-image-two-times-is-simil.

[5] Georg Wimmer, Andreas Uhl, and Andreas Vecsei. "Evaluation of domain specific data augmentation techniques for the classification of celiac disease using endoscopic imagery". In: *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*. 2017, pp. 1–6. DOI: 10.1109/MMSP.2017.8122221.