Deep Neural Network

Roberto Di Stefano

INDICE

progetto di Deep Neural Network, creato il 18/06/2024
Università degli Studi dell'Aquila, L'Aquila, Italia

I. Descrizione del progetto

In questo progetto sono state confrontate 6 diverse reti neurali nel portare a termine il task di classificazione di immagini. Le reti differiscono l'una dall'altra per quanto riguarda: l'architettura, lo schema di inizializzazione e dalla modalità di training. In particolare sono state utilizzate due diverse architetture:

- **NetA1:** composta dai seguenti layer:
  1) Un primo layer convoluzionale (input channels=1, output channels=4, kernel size=5, stride=2);
  2) Per secondo un layer completamente connesso (input features=576, output features=10)
  3) Per ultimo viene applicata una softmax per calcolare le probabilità delle dieci classi di classificazione.
- **NetA2:** composta dai seguenti layer:
  1) Un primo layer convoluzionale (input channels=1, output channels=4, kernel size=5, stride=2);
  2) Per secondo un ulteriore layer convoluzionale (input channels=4, output channels=12, kernel size=3, stride=2)
  3) Per terzo un layer completamente connesso (input features=300, output features=10)
  4) Per ultimo viene applicata una softmax per calcolare le probabilità delle dieci classi di classificazione.

Due diversi schemi di inizializzazione:

- **ByHand:** I kernel sono inizializzati solo con 0 e 1. E ogni kernel deve contenerne un diverso schema;
- **Default:** La inizializzazione standard di ciascun layer.

Due diverse modalità di training:

- **Conv1Frozen:** Vengono allenati tutti i layer ad eccezione del primo layer convoluzionale;
- **AllTrained:** Vengono allenati tutti i layer della rete neurale.

Complessivamente si hanno quindi 3 diversi schemi per ciascuna architettura:

- **HF:** Il primo layer della rete neurale viene inizializzato con la strategia ByHand, i restanti layer vengono inizializzati con la strategia di Default e viene utilizzata la modalità Conv1Frozen per allenare la rete;
- **HT:** il primo layer della rete neurale viene inizializzato con la strategia ByHand, i restanti layer vengono inizializzati con la strategia di Default e viene utilizzata la modalità AllTrained per allenare la rete;
- **DT:** vengono inizializzati tutti i layer con la stratedia di Default e la rete viene allenata con la modalità AllTrained.

N.B. Si cerca di inizializzare tutti i layer corrispondenti delle diverse reti neurali con lo stesso schema di inizializzazione, quando non è esplicitamente richiesto diversamente dallo schema utilizzato. L'analisi seguente in particolare ha tre principali obbiettivi:

- **inset comparison:** Confrontare le reti aventi la stessa architettura;
- **architecture comparison:** Confrontare le reti aventi lo stesso schema ma che differiscono per la propria architettura;
- **recovery comparison:** Confrontare tutte le reti aventi l'architettura NetA1 con la rete avente l'architettura NetA2 e lo schema HF.

II. Scelta degli iperparametri delle architetture

Il progetto in questione aveva una serie di vincoli per quanto riguarda alcuni dei parametri delle due architetture mentre lasciava libera scelta per quanto riguardava altri. In modo particolare per quanto riguarda la prima architettura vale la pena discutere la scelta dei seguenti parametri:

- **kernel size (conv1):** Per quanto riguarda la dimensione del kernel del primo layer convoluzionale di entrambe le architetture si è scelto di utilizzare, come di convenzione, un valore dispari. I kernel dispari, grazie alla loro natura centrata, sono particolarmente efficaci nel rilevamento di pattern simmetrici oltretutto, nel caso in cui si sceglie di utilizzare padding, avere un kernel di dimensione dispari permette di semplificare i conti necessari per determinare la dimensione della rappresentazione in output. In modo particolare si è optato per una dimensione pari a 5 in modo da considerare un sufficiente numero di pixel nella determinazione di un determinato pattern.
- **stride (conv1):** Per quanto riguarda lo stride del primo layer convoluzionale si è scelto di utilizzare un valore pari a due in modo da semplificare in modo implicito la rappresentazione in output dell'immagine.
- **kernel size (conv2):** Anche in questo caso, come per il primo layer convoluzionale, si è utilizzata una dimensione dispari del kernel. Tuttavia rispetto al primo layer si è scelto un valore più piccolo in quanto in questo caso si sta determinando una pattern sui pattern determinati dal primo di conseguenza, generalmente, sono sufficienti un numero minore di valori per determinare il nuovo pattern.
- **output channels (conv2):** Per quanto riguarda il numero di canali di output si è deciso di triplicare il numero di canali in input, infatti è buona norma aumentare sempre ad ogni layer convoluzionale il numero di canali in maniera tale da determinare via via un numero maggiore di pattern.
- **stride (conv2):** Per motivi analoghi a quelli che ci hanno portato alla stessa scelta nel caso del primo layer convoluzionale, anche per il secondo, si è scelto un valore di stride pari a 2.

```python
class NetA1(nn.Module):
    def __init__(self, num_classes: int):
        super(NetA1, self).__init__()
        self.relu = nn.ReLU()
        self.conv1 =
            nn.Conv2d(in_channels=1,
            out_channels=4, kernel_size=5,
            stride=2)
        self.flatten =
            nn.Flatten(start_dim=-3)
        self.linear1 = nn.Linear(576,
            num_classes)
        self.softmax = nn.Softmax(dim=0)

    def freeze(self, layer: str):
        for param in getattr(self,
            layer).parameters():
                param.requires_grad = False

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.flatten(x)
        x = self.relu(self.linear1(x))
        x = self.softmax(x)
        return x
```

```python
class NetA1(nn.Module):
    def __init__(self, num_classes: int):
        super(NetA1, self).__init__()
        self.relu = nn.ReLU()
        self.conv1 =
            nn.Conv2d(in_channels=1,
            out_channels=4, kernel_size=5,
            stride=2)
        self.flatten =
            nn.Flatten(start_dim=-3)
        self.linear1 = nn.Linear(576,
            num_classes)
        self.softmax = nn.Softmax(dim=0)

    def freeze(self, layer: str):
        for param in getattr(self,
            layer).parameters():
                param.requires_grad = False

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.flatten(x)
        x = self.relu(self.linear1(x))
        x = self.softmax(x)
        return x
```

III. Inizializzazione

Come già accennato nel paragrafo (Par. I), per quanto riguarda l'inizializzazione, si hanno due diversi schemi: ByHand, Default. In particolare la strategia ByHand è utilizzata solamente nel primo layer convoluzione degli schemi HF e HT di entrambe le architetture, mentre si utilizza l'inizializzazione di Default per tutti gli altri layer. L'inizializzazione di default non necessità discussione in quanto è semplicemente quella standard utilizzata implicitamente dalle librerie nel momento in cui si inizializza ciacun layer. Per quanto riguarda invece l'inizializzazione ByHand, come su richiesta, si sono utilizzati solamente valori pari a 0 e 1. Ciascun parametro di inizializzazione è stato scelto in maniera tale da configurare pattern che avessero una qualche forma di simmetria, sempre però cercando di mantenere quanto più uguale il numero di parametri posti a 0 rispetto al numero di parametri posti a 1. Rendere quanto più uniforme la distribuzione dei parametri è un aspetto cruciale. Infatti, se al contrario si scegliesse di iniziali tutti i parametri allo stesso valore, la trasformazione applicata sarebbe la stessa su tutto l'input.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Fig. 1: In questa figura è rappresentata lo schema di inizializzazione ByHand utilizzato, ciascuna matrice contiene i parametri di inizializzazione di ciascun kernel

Un ulteriore fondamentale aspetto dell'inizializzazione è che quando possibile si è utilizzata la stessa. Ciò significa che all'interno della stessa architettura l'inizializzazione di tutti i layer dello schema HF è esattamente la stessa di quelli dello schema HT mentre differisce dallo schema DT solo per il primo layer, infatti nel caso dello schema DT è stata utilizzata la inizializzazione di default. Allo stesso tempo l'inizializzazione del primo layer degli schemi HT e HF di entrambe le architetture è la stessa come anche quella dello schema DT. Per assicurarsi che effettivamente l'inizializzazione dei layer sopra indicati fosse la stessa si è calcolata la norma della differenza dei tensori di inizializzazione, ottenendo il seguente risultato:

```
Net_A1:
  |W_{conv_a1_hf} - W_{conv_a1_ht}| = tensor(0., grad_fn=<LinalgVectorNormBackward0>)
  |W_{linear_a1_hf} - W_{linear_a1_ht}| = tensor(0., grad_fn=<LinalgVectorNormBackward0>)
  |W_{linear_a1_hf} - W_{linear_a1_dt}| = tensor(0., grad_fn=<LinalgVectorNormBackward0>)

Net_A2:
  |W_{conv1_a2_hf} - W_{conv1_a2_ht}| = tensor(0., grad_fn=<LinalgVectorNormBackward0>)
  |W_{conv2_a2_hf} - W_{conv2_a2_ht}| = tensor(0., grad_fn=<LinalgVectorNormBackward0>)
  |W_{linear1_a2_hf} - W_{linear1_a2_ht}| = tensor(0., grad_fn=<LinalgVectorNormBackward0>)
  |W_{linear1_a2_hf} - W_{linear1_a2_dt}| = tensor(0., grad_fn=<LinalgVectorNormBackward0>)

Net_A1 Vs Net_A2:
  |W_{conv1_a1_hf} - W_{conv1_a2_hf}| = tensor(0.)
  |W_{conv1_a1_ht} - W_{conv2_a2_ht}| = tensor(0., grad_fn=<LinalgVectorNormBackward0>)
  |W_{conv1_a1_dt} - W_{conv2_a2_dt}| = tensor(0., grad_fn=<LinalgVectorNormBackward0>)
```

IV. Training

Come già accennato nel paragrafo (Par. I), per quanto riguarda le modalità di training, se ne sono adoperate due: Conv1Frozen e AllTrained. Nel primo caso si allenano tutti i layer della rete neurale ad eccezione del primo layer convoluzionale, nel secondo caso invece si allenano tutti i layer indiscriminatamente. Le scelte che meritano una discussione sono le seguenti:

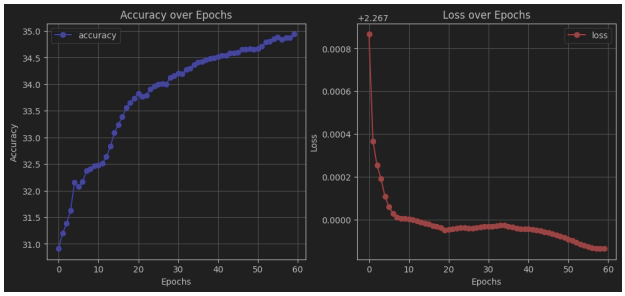
- **loss function:** È stata utilizzata la Cross Entropy Loss che è una loss function molto popolare per risolvere problemi di classificazione. Utilizzare questa loss function era una richiesta esplicita della consegna del progetto;
- **algoritmo di ottimizzazione:** Per quanto riguarda l'algoritmo di ottimizzazione si è scelto di utilizzare Adam (abbreviazione per Adaptive Moment Estimation), questo algoritmo ha il principale vantaggio di sfruttare le informazioni sui gradienti passati e di considerare un diverso valore di learning rate per ciascuna componente del gradiente. La conseguenza è quella di portare ad una accelerazione del processo di training e allo stesso tempo di evitare fenomeni di exploading/vanishing gradient;
- **batch size:** Rappresenta il numero di campioni di dati che il modello processa prima di aggiornare i suoi pesi. Un valore più grande produce una stima del gradiente più precisa ma uno più piccolo permette di introdurre del rumore che aiuta ad evitare fenomeni di overfitting. Per le raggioni espresse il valore scelto è stato 128 che è un numero di campioni ne troppo grande ne troppo piccolo;
- **numero di epochs:** Con il termine epoch ci si riferisce all'insieme di iterazioni necessarie per considerare l'intero dataset. Il numero di epoch è invece un valore proporzionale al tempo di training e corrisponde al numero di epoch eseguite. Questo valore è stato stabilito osservando l'andamento delle 6 reti e valutando dopo quante epoch si ha convergenza. Si osserva che per trovare un valore che fosse adeguatamente buono per tutte e 6 le reti non si è trovato un valore ottimale per nessuna delle 6;
- **learning rate:** Rappresenta in un certo senso la velocità con cui la rete impara. Più grande è più velocemente la rete impara, più piccolo è più precisa è la soluzione trovata. Si è cercato di trovare un valore che potesse andare bene a tutte le reti anche se non è l'ottimale per nessuna. Questo per rendere il confronto tra le diverse reti il più indipendente dalle variabili degli esperimenti non controllate.

```
batch_size = 128

train_dataloader= DataLoader(train_data, batch_size = batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

learning_rate = 3.35e-4
epochs = 60

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)

def test_loop(device, dataloader, model, loss_fn):
  size = len(dataloader.dataset)
  num_batches = len(dataloader)
  test_loss, correct = 0, 0

  with torch.no_grad():
    for X, y in dataloader:
      X, y = X.to(device), y.to(device)
      pred = model(X)
      test_loss += loss_fn(pred, y).item()
      correct += (pred.argmax(1) == y).type(torch.float).sum().item()

  test_loss /= num_batches
  correct /= size
  print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
  return 100*correct, test_loss
```

V. Inset Comparison

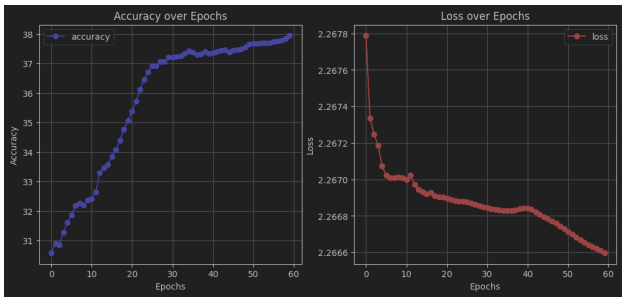
A. NetA1

HF:



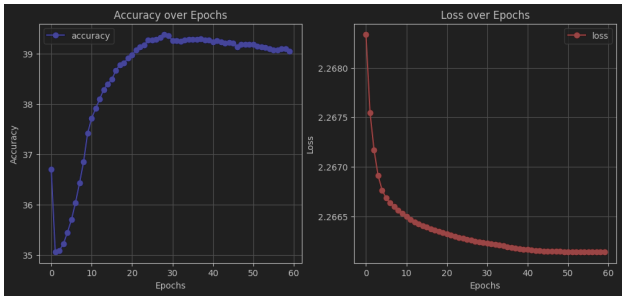

```
time lapse of each batch is: 3.66
training time is: 03:40.71
```

HT:




```
time lapse of each batch is: 5.13
training time is: 05:07.57
```

DT:




```
time lapse of each batch is: 5.16
training time is: 05:09.49
```

Volendo confrontare le tre reti neurali aventi l'architettura NetA1, la prima cosa che si può osservare è che tra le tre, la rete che raggiunge la massima accuratezza, è la DT. Infatti, i picchi di accuratezza sono rispettivamente di:

- **massima accuratezza HF:** 34.9%;
- **massima accuratezza HT:** 38%;
- **massima accuratezza DT:** 39.4%.

Questo risultato è coerente con il fatto che lo schema DT è anche quello con l'inizializzazione di default e con il fatto che tutti i layer vengano allenati.

un altro parametro di confronto è il tempo di training per batch: I tempi sono rispettivamente:

- **time/batch HF:** 3.66s;
- **time/batch HT:** 5.13s;
- **time/batch DT:** 5.16s;

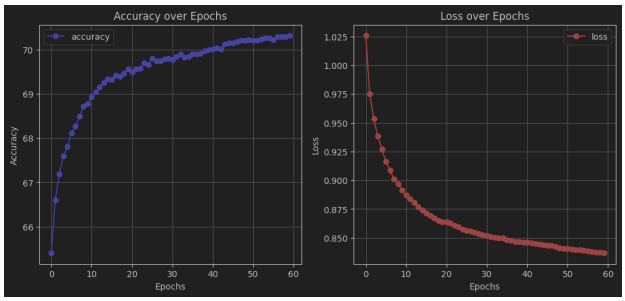
Lo schema che ottiene i tempi più bassi è chiaramete lo schema HF questo è dovuto al fatto che in questo schema non si allena il primo layer. Questo è il principale vantaggio dello schema HF.

Osservando specialmente l'andamento del valore ottenuto dalla loss function si osserva che lo schema DT è anche più stabile rispetto agli altri due schemi. Gli andamenti degli altri due schemi sono invece più instabili e hanno delle somiglianze e questo può essere giustificato dalla stessa inizializzazione ByHand che si riconosce anche sui parametri allenati della rete HT.

Osservando ancora l'andamento dell'accuratezza delle tre reti si osserva che le prime due mantengono, anche sulle ultime epoch, una crescita non trascurabile questo potrebbe essere sintomo di un numero di epoch troppo basso. Per quanto riguarda lo schema DT invece si osserva che la rete converge anzi, si ha un sensibile decremento dell'accuracy nelle ultime epoch, sintomo probabilmente di un learning rate troppo alto e di conseguenza di fenomeni di overfitting. Si osserva che il valore di training rate e numero di batch è stato stabilito per essere discretamente buono per tutte e sei le reti e di conseguenza non sarà ottimale per nessuana delle 6.

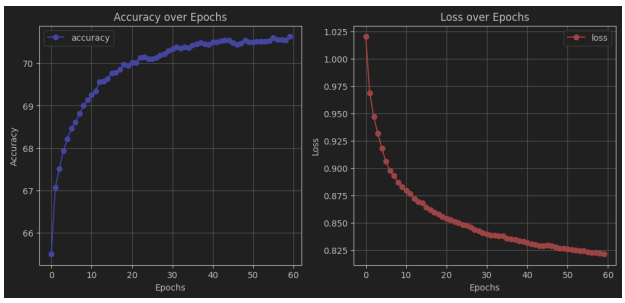
B. NetA2

HF:



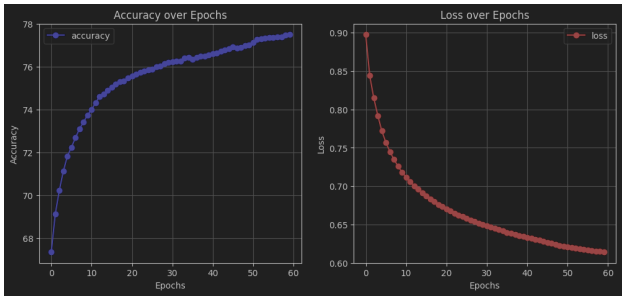

```
time lapse of each batch is: 4.51
training time is: 04:30.51
```

HT:




```
time lapse of each batch is: 5.95
training time is: 05:57.19
```

DT:




```
time lapse of each batch is: 6.07
training time is: 06:03.84
```

Volendo confrontare le tre reti neurali aventi l'architettura NetA2, la prima cosa che si può osservare è che tra le tre, la rete che raggiunge la massima accuratezza, è la DT. Infatti, i picchi di accuratezza sono rispettivamente di:

- **massima accuratezza HF:** 70.3%;
- **massima accuratezza HT:** 70.6%;
- **massima accuratezza DT:** 79.6%.

Questo risultato è coerente con il fatto che lo schema DT è anche quello con l'inizializzazione di default e con il fatto che tutti i layer vengano allenati.

un altro parametro di confronto è il tempo di training per batch: I tempi sono rispettivamente:

- **time/batch HF:** 4.51s;
- **time/batch HT:** 5.95s;
- **time/batch DT:** 6.07s;

Lo schema che ottiene i tempi più bassi è chiaramente lo schema HF questo è dovuto al fatto che in questo schema non si allena il primo layer. Questo è il principale vantaggio dello schema HF. Per quanto riguarda invece i restanti 2 schemi (HT, DT) i tempi sono praticamente identici questo è in accordo con il fatto che in entrambi i casi si usi la stessa modalità di training.

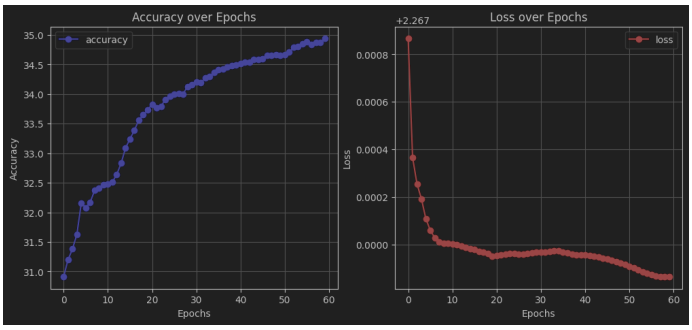
L' andamento dei valori ottenuti dalla loss function e quello dell'accuratezza risultano piuttosto stabili per tutte e tre le reti. Si può osservare anche un andamento piuttosto simile sintomo del fatto che la presenza del secondo layer convoluzionale, inizializzato allo stesso modo e allenato in tutte e tre le reti, rende meno influente il primo layer convoluzionale, per quanto, la diversa inizializzazione di quest'ultimo è la causa della differenza prestazionale delle tre reti.

Un ulteriore aspetto interessante è che il picco di accuratezza ottenuto dallo schema HF è molto simile a quello dello schema HT questo è giustificabile dal fatto che, avendo lo stesso schema di inizializzazione, la rete converge in un minimo locale e che probabilmente il learning rate è troppo basso per permettere una fuga da questo minimo. A seguito di alcuni test che non verranno riportati in questo documento si osserva che per le tre reti dell'architettura NetA2 è conveniente scegliere un learning rate più grande di quello utilizzato.

VI. Architecture Comparison

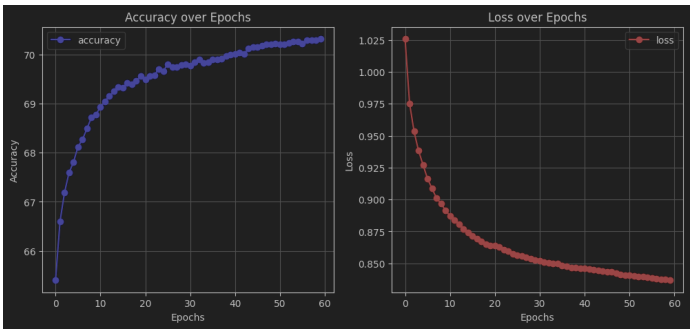
A. HF

NetA1HF




```
time lapse of each batch is: 3.66
training time is: 03:40.71
```

NetA2HF




```
time lapse of each batch is: 4.51
training time is: 04:30.51
```

I picchi di accuratezza delle due reti sono rispettivamente:

- **massima accuratezza NetA1HF:** 34.9%;
- **massima accuratezza NetA2HF:** 70.3%.

L'accuratezza della NetA2HF è di conseguenza quasi il doppio della NetA1HF, vediamo tuttavia che se nelle ultime epoch si ha convergenza sull'andamento della accuratezza di NetA2HF, nel caso della NetA1HF si ha ancora un incremento non trascurabile che potrebbe lasciare pensare che con un maggior numero di epoch si poteva raggiungere un accuratezza maggiore.

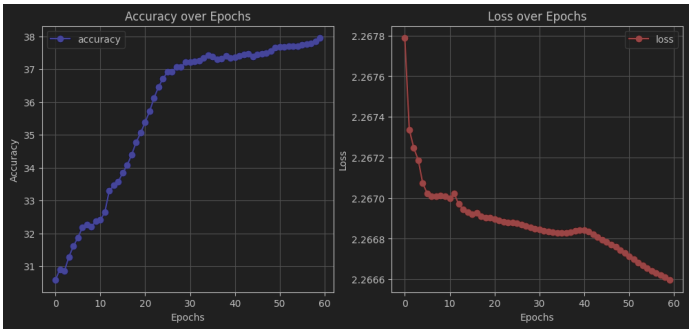
Per quanto riguarda invece il tempo per epoch si ha il seguente risultato:

- **time/batch NetA1HF:** 3.66s;
- **time/batch NetA2HF:** 4.51s.

Pur utilizzando lo stesso metodo di training la NetA1HF ottiene tempi più bassi questo si giustifica dal fatto che la rete NetA2HF allena anche il secondo layer convoluzionale.

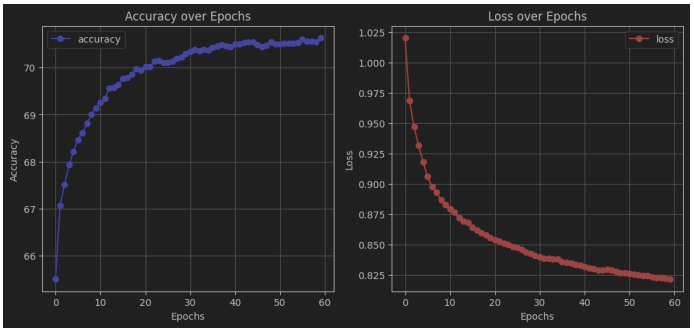
B. HT

NetA1HT




```
time lapse of each batch is: 5.13
training time is: 05:07.57
```

NetA2HT




```
time lapse of each batch is: 5.95
training time is: 05:57.19
```

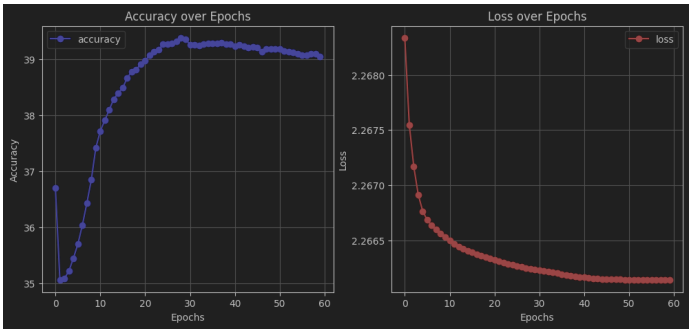
I picchi di accuratezza delle due reti sono rispettivamente:

- **massima accuratezza NetA1HT:** 38.0%;
- **massima accuratezza NetA2HT:** 70.6%.

Anche con lo schema HT si ottengono comunque i risultati migliori con l'architettura NetA2. Quello che però si osserva è che il vantaggio ottenuto dal passaggio dallo schema HF a quello HT è maggiore nel caso dell'architettura NetA1. Questo si giustifica con il fatto che nella seconda architettura il primo layer ha meno influenza rispetto a quanto ne ha nella prima architettura nella quale è l'unico layer convoluzionale.

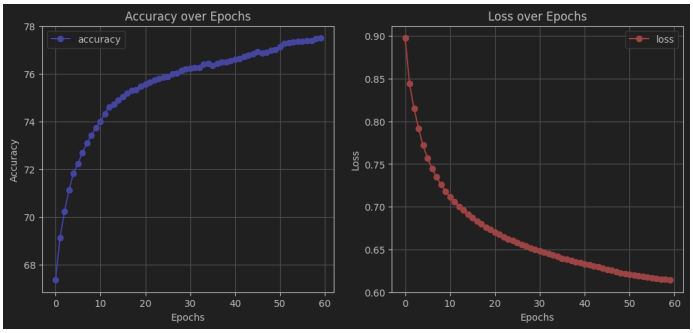
C. DT

NetA1DT




```
time lapse of each batch is: 5.16
training time is: 05:09.49
```

NetA2DT




```
time lapse of each batch is: 6.07
training time is: 06:03.84
```

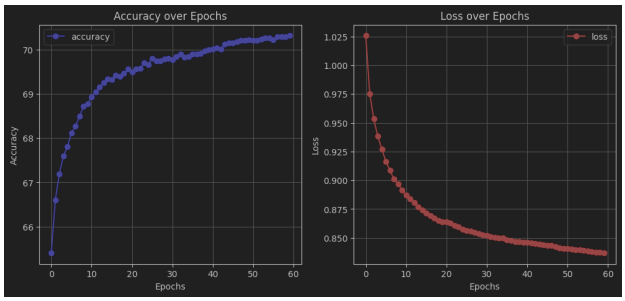
I picchi di accuratezza delle due reti sono rispettivamente:

- **massima accuratezza NetA1DT:** 39.4%;
- **massima accuratezza NetA2DT:** 79.6%.

La massima accuratezza ottenuta della rete NetA2DT è nettamente migliore, anche in questo caso quasi il doppio. L'aspetto più interessante è però relativo all'andamento infatti si osserva che la rete NetA2 ha un andamento ancora in crescita anche sulla sessantesima epoch, mentre la rete NetA1DT presenta fenomeni di overfitting nelle ultime batch. Questo risultato evidenzia la necessità di diversi learning rate per le due reti.

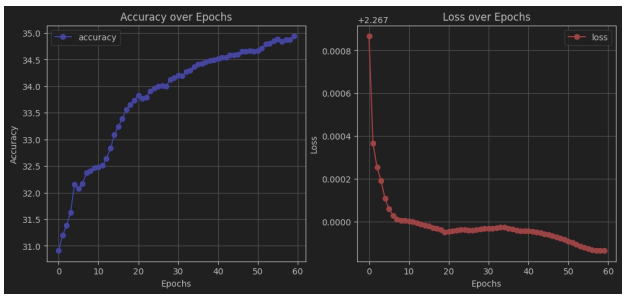
VII. Recovery Comparison

NetA2HF:



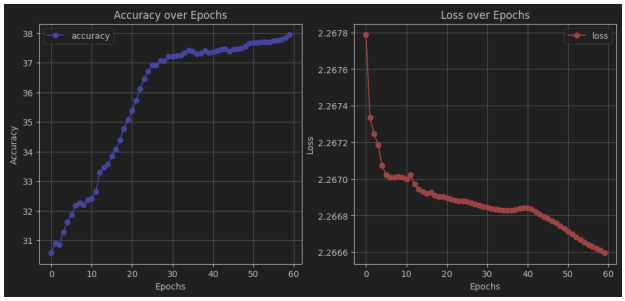

```
time lapse of each batch is: 4.51
training time is: 04:30.51
```

NetA1HF:



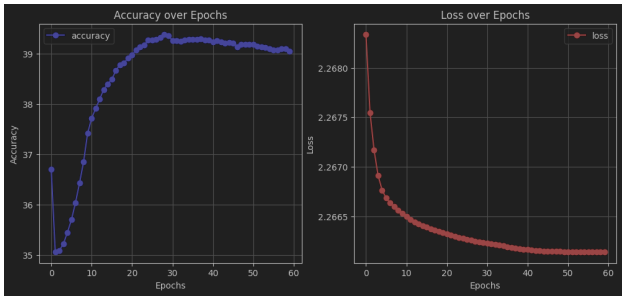

```
time lapse of each batch is: 3.66
training time is: 03:40.71
```

NetA1HT:




```
time lapse of each batch is: 5.13
training time is: 05:07.57
```

NetA1DT:




```
time lapse of each batch is: 5.16
training time is: 05:09.49
```

Confrontando le reti aventi l'architettura NetA1 con la rete NetA2HF si riscontano le seguenti accuratezze:

- **massima accuratezza NetA1HF:** 34.9%;
- **massima accuratezza NetA1HT:** 38%;
- **massima accuratezza NetA1DT:** 39.4%;
- **massima accuratezza NetA2HF:** 70.3%.

E i seguenti tempi per epoch:

- **time/batch NetA1HF:** 3.66s;
- **time/batch NetA1HT:** 5.13s;
- **time/batch NetA1DT:** 5.16s;
- **time/batch NetA2HF:** 4.51s;

La rete NetA2HF ottiene un accuratezza di molto maggiore rispetto alle altre con un tempo di epoch maggiore rispetto alla rete NetA1HF ma comunque minore rispetto alle reti NetA1HT e NetA1DT.

Questo confronto dimostra che i vantaggi ottenuti inserendo un secondo layer convoluzionale non riescono ad essere compensati dalla modalità di training, inoltre ciò dimostra che utilizzare un architettura più complessa, non allenando uno dei layer, può essere una buona tecnica per mantenere i tempi di training bassi e l'accuratezza alta. Chiaramente ciò dipende anche dal layer che si va aggiungere, in questo caso il parametro più importate che permette la grande differenza è il numero di kernel che, rispetto al primo layer, vengono triplicati. Triplicare il numero di canali permette di ottenere un gran numero di pattern e quindi una rappresentazione più informativa.