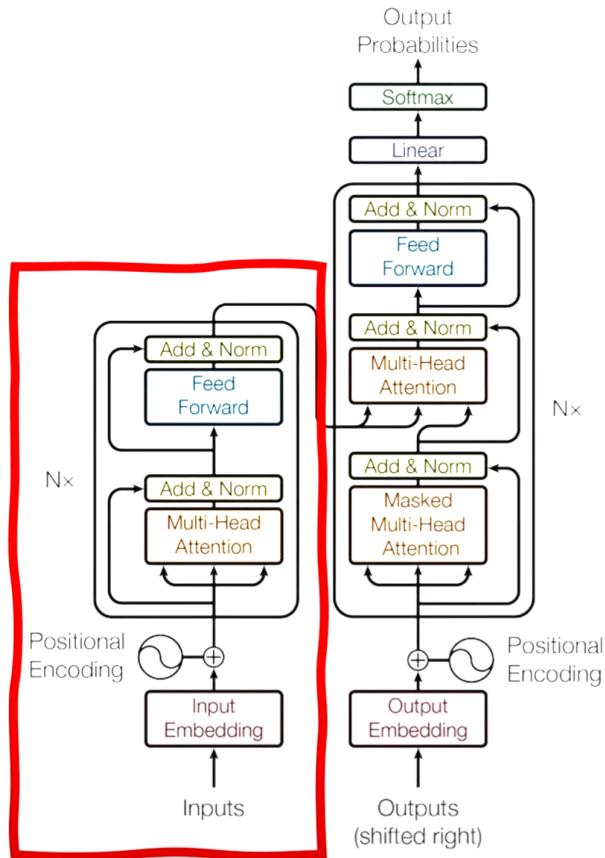


## VIT

- מודל הרחבת עבור Transformers רק עבר מידע ויזואלי (תמונות).  
המודל מורכב מ-Encoder ו Decoder כלהלן:



### בצורת Encoder עובד?

תחילה הקלט מומר לוקטורים נומריים או ל-Embeddings (התאמת חח"ע בין מידע לוקטורים) שנעשה ע"י Image Patches בה התמונה מתפרקת לסדרה על ידי הרכיב הראשון במודל. כל צבע בתמונה עובר דרך נוירונים באמצעות שכבות לנאריות והפלט הוא Embedding.

### איך ניתן לבצע את ה-Image Patching בצורה יעילה?

כל תמונה מורכבת מ-3 ערוצי צבע (אדום, ירוק וכחול), מרוחב ומוארך. בשביב לפרק את התמונה לרוב משתמש בחיבור בשם Einops שמאפשרת בסדר מערכים רב ממדים. ניתן לבצע את הסידור באמצעות הקוד הבא:

```
Rearrange('b c (h p1) (w p2) -> b (h w) (p1 p2 c)', p1=patch_size, p2=patch_size)
```

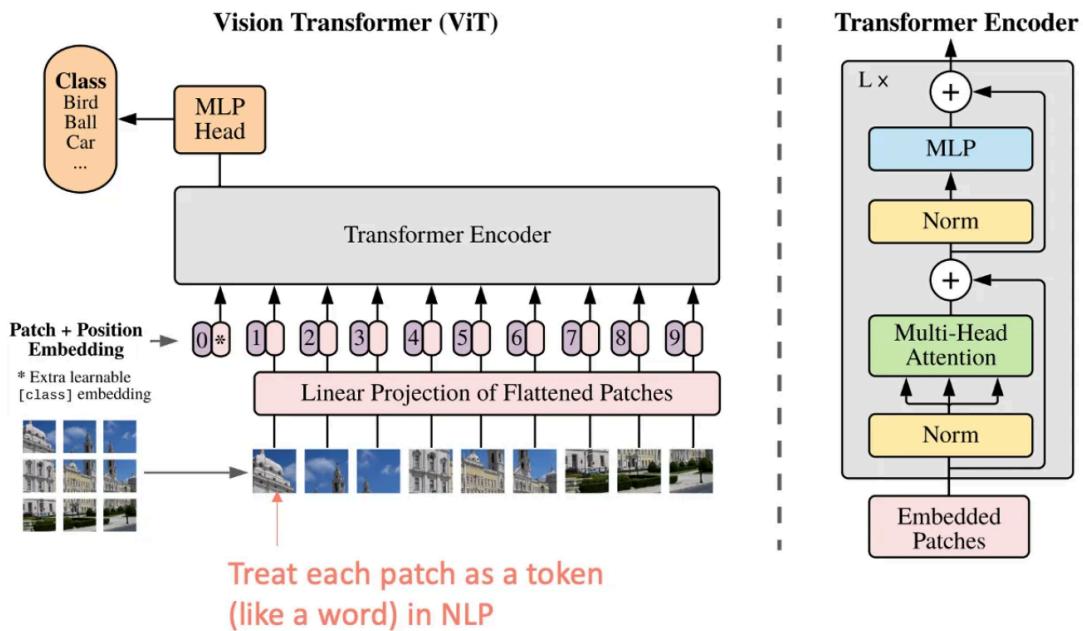
כאשר  $b$  זה גודל ה- $\text{batch}$ ,  $c$  זה 3 (כמויות ערוצי הצבע),  $h$  זה הגובה וה- $w$  זה הרוחב ו- $p1$  ו- $p2$  זה לפי כמות ה-patches שרצים לפרק את התמונה למשול  $16 \times 16$  או  $p1, p2 = 16$

להלן הקוד עבור ה-Patch Embedding

```
class PatchEmbedding(nn.Module):
    def __init__(self, in_channels = 3, patch_size = 8, emb_size = 128):
        self.patch_size = patch_size
        super().__init__()
        self.projection = nn.Sequential(
            # break-down the image in  $s_1 \times s_2$  patches and flat them
            Rearrange('b c (h p1) (w p2) -> b (h w) (p1 p2 c)', p1=patch_size, p2=patch_size),
            nn.Linear(patch_size * patch_size * in_channels, emb_size)
        )

    def forward(self, x: Tensor) -> Tensor:
        x = self.projection(x)
        return x
```

- הופך את כל המידע לייצוג גלובלי אחד בלבד בשביל הסיווג.  
 בוואו נזכר כי כל Patch הופך לוקטור כולל עוד וקטור למידה בהתחלה הנקרא **Position Embedding** המציג את כל התמונה ומתקבל מילדי מכל הקלטים.  
 מהו הטוקן המשמש כ-**Feature Extractor**



- נזכיר כי מודל **Transformers** הוא תלוי מיקום ולכן תורם למודל להבין איפה כל Patch נמצא בתמונה המקורי.  
 לא מדובר במספר סידורי אלא בוקטור (מעumi נוחות וחיסכון) שמורכב משילוב פונקציות  $\sin$  ו- $\cos$ .  
 נבחין כי ב-ViT ה-Positional Embedding נעשה לאחר ה-Input Embedding ולא במקביל.  
 כפי שניתן לראות להלן, זהו פרמטר וקטור שניינו למידה עם מימדים:

```
# Learnable params
num_patches = (img_size // patch_size) ** 2
self.pos_embedding = nn.Parameter(
    torch.randn(1, num_patches + 1, emb_dim))
self.cls_token = nn.Parameter(torch.rand(1, 1, emb_dim))
```

- בлок זה מתבצע  $N$  פעמים. נחلك אותו לרכיביו:  
 • **ה-Encoder:** מאפשר לשתף מידע בין הקלטים השונים.  
 • **ה-Multi-Head Attention:** יש לו 3 קלטים שהם query, key, value והמיושם הוא כדלהלן (יש כבר מימוש מספרית `(torch)`):

```
class Attention(nn.Module):
    def __init__(self, dim, n_heads, dropout):
        super().__init__()
        self.n_heads = n_heads
        self.att = torch.nn.MultiheadAttention(embed_dim=dim,
                                              num_heads=n_heads,
                                              dropout=dropout)
        self.q = torch.nn.Linear(dim, dim)
        self.k = torch.nn.Linear(dim, dim)
        self.v = torch.nn.Linear(dim, dim)

    def forward(self, x):
        q = self.q(x)
        k = self.k(x)
        v = self.v(x)
        attn_output, attn_output_weights = self.att(x, x, x)
        return attn_output
```

- **ה-Add & Norm:** שכבה הנורמליזציה. מנורמלת כל קלט לשכבה. הסיבה מדוע לא משתמשים ב-Batch Norm המקביל שכן Transformers משתמש בסדרות.attention השימוש השתרמשנו במיומש PreNorm בה הוא משתמש בפונקציה למשל

```
class PreNorm(nn.Module):
    def __init__(self, dim, fn):
        super().__init__()
        self.norm = nn.LayerNorm(dim)
        self.fn = fn
    def forward(self, x, **kwargs):
        return self.fn(self.norm(x), **kwargs)

norm = PreNorm(128, Attention(dim=128, n_heads=4, dropout=0.))
norm(torch.ones((1, 5, 128))).shape
```

- **ה-Feed Forward:** שכבה לינארית שמטרתה לקחת את ה-Attention ומעבירה אותו לשכבה הבאה או מוציאה אותו כפלט. בקוד אנו משתמשים ב-2 שכבות לינאריות ובפונקציית ה-GELU בשם Activision ובנוסף אנו משתמשים ב-Dropout כדי להימנע מ-Overfitting

```
class FeedForward(nn.Sequential):
    def __init__(self, dim, hidden_dim, dropout = 0.):
        super().__init__(
            nn.Linear(dim, hidden_dim),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dim, dim),
            nn.Dropout(dropout)
        )
ff = FeedForward(dim=128, hidden_dim=256)
ff(torch.ones((1, 5, 128))).shape
```

- **ה-Residual:** מטרתו לסייע בזרימה של הרשת למפלס כדי למנוע את בעיית Vanishing Gradients

```
class ResidualAdd(nn.Module):
    def __init__(self, fn):
        super().__init__()
        self.fn = fn

    def forward(self, x, **kwargs):
        res = x
        x = self.fn(x, **kwargs)
        x += res
        return x

residual_att = ResidualAdd(Attention(dim=128, n_heads=4, dropout=0.))
residual_att(torch.ones((1, 5, 128))).shape
```

## כל המודל:

בקוד הבא חסרים כמה Norms ו-Dropouts והוא מורכב מ-4 שכבות בלבד ומועד לתמונה קטנה יותר.

```
class ViT(nn.Module):
    def __init__(self, ch=3, img_size=144, patch_size=4, emb_dim=32,
                 n_layers=6, out_dim=37, dropout=0.1, heads=2):
        super(ViT, self).__init__()

        # Attributes
        self.channels = ch
        self.height = img_size
        self.width = img_size
        self.patch_size = patch_size
        self.n_layers = n_layers

        # Patching
        self.patch_embedding = PatchEmbedding(in_channels=ch, patch_size=patch_size, emb_size=emb_dim)
        # Learnable params
        num_patches = (img_size // patch_size) ** 2
        self.pos_embedding = nn.Parameter(
            torch.randn(1, num_patches + 1, emb_dim))
        self.cls_token = nn.Parameter(torch.rand(1, 1, emb_dim))

        # Transformer Encoder
        self.layers = nn.ModuleList([])
        for _ in range(n_layers):
            transformer_block = nn.Sequential(
                ResidualAdd(PreNorm(emb_dim, Attention(emb_dim, n_heads = heads, dropout = dropout))),
                ResidualAdd(PreNorm(emb_dim, FeedForward(emb_dim, emb_dim, dropout = dropout))))
            self.layers.append(transformer_block)

        # Classification head
        self.head = nn.Sequential(nn.LayerNorm(emb_dim), nn.Linear(emb_dim, out_dim))

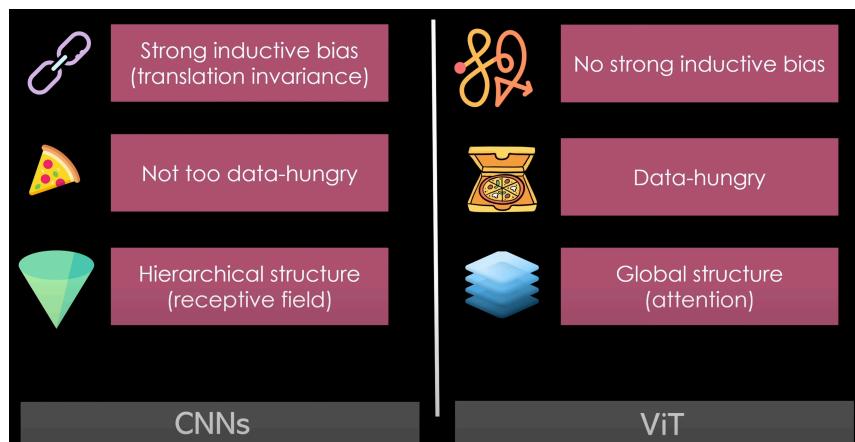
    def forward(self, img):
        # Get patch embedding vectors
        x = self.patch_embedding(img)
        b, n, _ = x.shape

        # Add cls token to inputs
        cls_tokens = repeat(self.cls_token, '1 1 d -> b 1 d', b = b)
        x = torch.cat([cls_tokens, x], dim=1)
        x += self.pos_embedding[:, :(n + 1)]

        # Transformer layers
        for i in range(self.n_layers):
            x = self.layers[i](x)

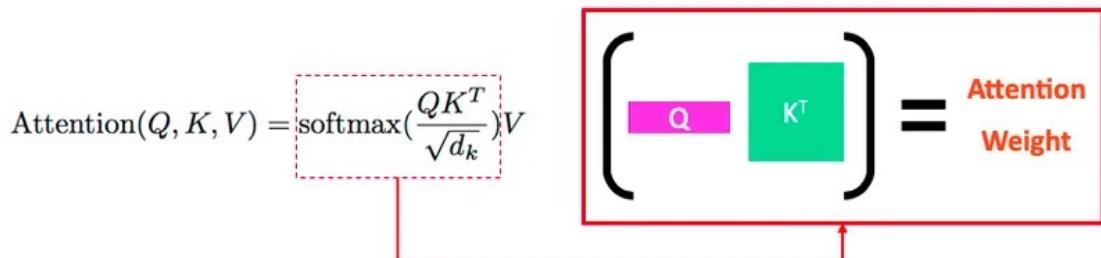
        # Output based on classification token
        return self.head(x[:, 0, :])
```

## הבדלים בין CNN ל-ViT ומתי להשתמש באיזה מודל:



## VIT VS CNN (paper)

מודלי Transformer משתמשים במה שנקרא self-attention.  
הנוסחה ל-Attention במודל Transformer משתמשת ב-3 געלמים:  
Q(Query), K(Key), V(Value)  
המטרה היא לחשב את הקשר בין Query token (attention weight) ולחכפל את Value token (Single Head).



עם זאת עבור VIT אנו משתמשים ב-multi-head יש מטריצות הטלה Multi-Head Attention בו לכל head יש מטריצות הטלה (Q, K, V) ובעזרתם מחשבים את הקשר.

מה זה Self-Attention אם Q,K,V שבהם אנו משתמשים מחושבים כולם מאותו קלט, נקרא לה Self-Attention. עם זאת, החלק העליון ב-decoder שב-Transformer שcn הוא מחשב את ה-Attention באמצעות Q מה-encoder ו-K,V מה-decoder.

### כיצד VIT עובד?

התמונה מחולקת ל-N חתיכות בגודל 16X16 כאשר כל אחת היא תלת מימדית (גובה X רוחב X ערוצים) ולכן ניתן לטפל בהם באמצעות מודל transformer שימושי במיוחד דו מימדי. לכן ה-VIT משטח אותם להטלה לינארית כדי להפוך אותם למידע דו מידע וכך ניתן להתייחס לכל חתיכה בתור טוקן שיופיע קלאטי למודל ה-Transformer. בנוסף VIT מבצע תחיליה pre-training ורך לאחר מכן fine-tuning.

**הבדיה עם VIT:**  
אימון VIT דורש המון כמויות דedata. לעומת זאת Transformer מודלים מדויקים עם פחות DATA. אך נשים מדויקים יותר מאשר כמות הדטה קטנה ואנו עוקב CNN בכמה מקרים.

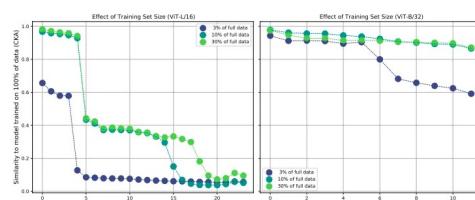
### הבדלים בין CNN ל-VIT:

ישנם 5 הבדלים משמעותיים בין המודלים ונರחיב על כל אחד מהם:

**1. VIT יש יותר דימויון בין הייצוג שמוסג ב-Deep Layers לבין ה-Shallow Layers.**  
ההבדל המרכזי ביניהם הוא שדה הראייה הגדל בשכבה הראשונית כאשר CNN יש שדה ראייה קבוע בגודל 3 או 7 בזמן ש-VIT משתמש ב-Self Attention את שדה הראייה גם בשכבות הנזוכות. לאחר הבדיקה ביצוגים בכל שבה בין המודלים, ניתן לראות דמיון ב-VIT וחוסר דמיון ב-CNN.

יתכן כי הסיבה לכך היא שב-VIT מקבלים את הייצוג הגלובלי מההתחלה לעומת CNN שמקבלים בשכבה עמוקה יותר (למשל CNN לוקח 70 שכבות עד שמקבלים ייצוג לעומת VIT של רק 40). בנוסף, הדמיון בין השכבה העמוקה בין 2 המודלים הוא נמוך מזה הייצוג האבסטרקט של תמונה שונה.

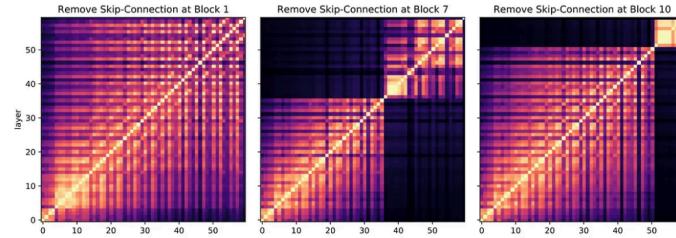
**2. בניגוד ל-CNN ה-VIT מושג את הייצוג הגלובלי מה-Shallow Layers (הייצוג הлокאלי המושג מה-Shallow Layers הוא גם חשוב).**  
מהבדיקות שנעשו, התגלה כי לא ניתן לקבל ייצוג לוקאלי מה-Shallow Layers עם כמות DATA קטנה. מכאן ש-VIT ייחד עם DATA גודל מביא לדיקוג גבוה אף מה הקשר בין כמות הדטה לייצוג המתקבל?



מכאן אנו מקבלים שהיצוג בשכבה העמוקה, שידוע כתרום לדיקוק, ניתן לאמן רק עם כמה נתונים גודלה.

### 3. דילוג בין חיבורים ב-ViT יותר משפיע מב-CNN ומשפיע על הביצועים ודמיון בין ייצוגים.

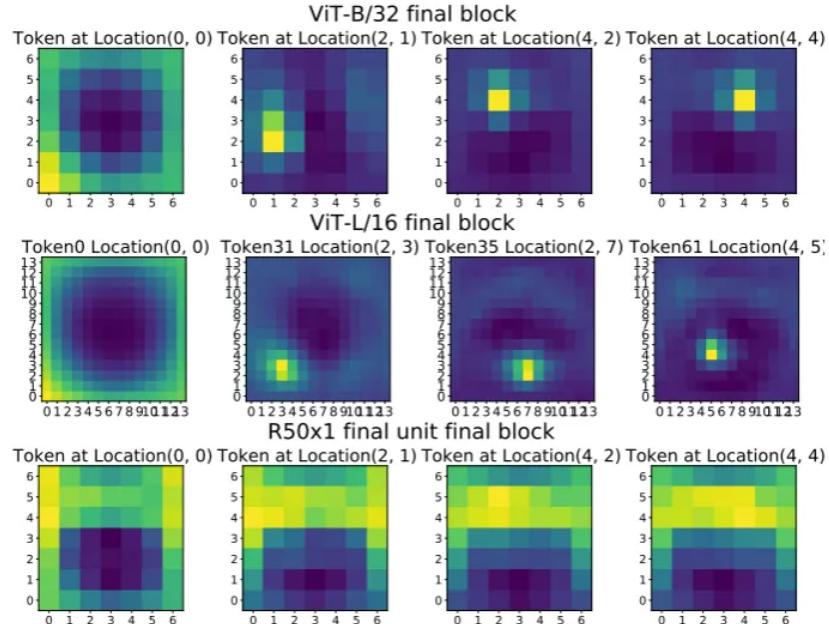
במחקר זה ה证实 ניסוי בו חישבנו את דמיון הייצוג המתקבל כתלות בביטול דילוג השכבה ה-*i*.



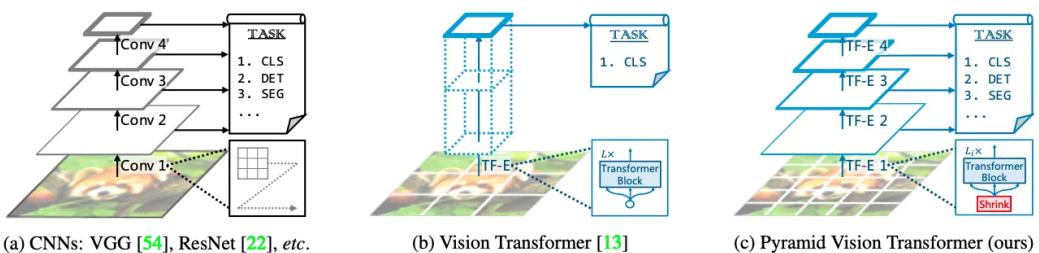
מכאן ניתן להגע למסקנה שדילוג בין חיבורים משפיע ישירות על הייצוג וכאש מרבלים אותו, הדמיון בין השכבות משתנה דרסטית (בפרט בשכבה האמצעית שם יורדת הדיוק ב-4%).

### 4. ה-ViT שומר יותר מידע מרוחבי מאשר CNN.

הבה נבחן בהבדלים בין ViT ל-CNN:



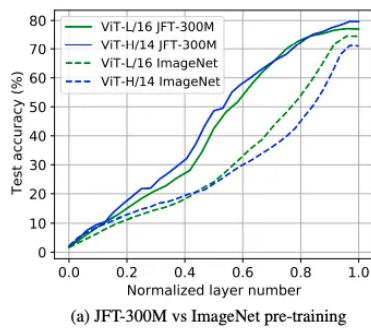
ה-ViT הוא השורה האמצעית והעליונה וה-CNN הוא השורה התחתונה. צפוי, ב-ViT בשכבה האחרונה הדמיון של המיקום המתאים גבוה וモוביל למידע מרוחבי גבוה ביחס ל-CNN בו דזוקה הדמיון בי מיקומים לא מתאימים הוא גבוה כלומר הוא לא שומר טוב על מיקום. הסיבה לשינוי בмагמה כנראה טמון בשינוי המבנה של הרשת:



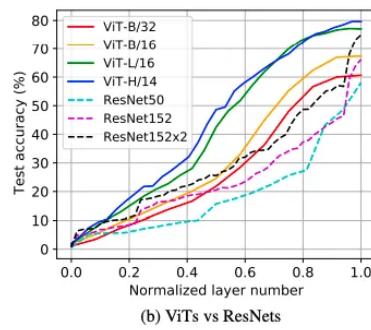
בזמן ש-CNN לוקחת ייצוג רצולוציות נמוכות 5 שלבים כאשר בכל שלב הרזולוציה מופחתת בחצי, ה-ViT ממיר לטוקן בגודל 16x16 ובכך מפחית את הרזולוציה במקום זהה ובכך ViT שומר טוב על מיקום.

## 5. ה-ViT יכול ללמד ייצוגי ביןינים באיכות גבוהה עם כמותות גבוהות של נתונים.

הבה נסתכל באיכות הייצוג של השכבה האמצעית באמצעות הניסוי הבא:



(a) JFT-300M vs ImageNet pre-training



(b) ViTs vs ResNets

בניסוי זה מנסים לבדוק האם ניתן להשתמש בשכבה האמצעית כדי לסוג באמצעות מודל לינארי. ככל שמקבלים דיקט טוב יותר עם מודלים פשוטים כאלה, השכבה תקבל ייצוג טוב יותר. תחילה ניתן לראות בניסוי כי DATASET טרנום טוב יותר לדיקט (ImageNet) עם 1.3 מיליון תמונות לעומת 300 מילון תמונות. בנוסף, ניתן להבחין כי מודל גדול יותר כמו CNN מפיק ייצוגים טובים יותר.

עם זאת בגרף הימני יש עלייה חדה של מודל CNN בשכבה האחורונה בהשוואה ל-ViT אך למה? במחקר שהתנהל ב-2019 הוכנו לשכבה האמצעית במודל CNN את הפונקציה הבאה:

$$l_{sn}(x, y, T) = -\frac{1}{b} \sum_{i \in 1..b} \log \left( \frac{\sum_{\substack{j \in 1..b \\ j \neq i \\ y_i = y_j}} e^{-\frac{\|x_i - x_j\|^2}{T}}}{\sum_{\substack{k \in 1..b \\ k \neq i}} e^{-\frac{\|x_i - x_k\|^2}{T}}} \right)$$

הפונקציה שנקראת Soft Nearest Neighbor Loss מעידה על מצב הקשר של הפיצ'רים לפי הקטגוריות. ערך גבוה ע"י הפונקציה מעיד שהפיצ'רים לפי מחלוקת קשורים וערך נמוך מעיד על כך שהם נפרדים. להלן דוגמה:



כעת נתבונן בערך של הפונקציה בכל בלוק במודל CNN:



ידעו שזויה רשת סיווג תמונות בעלת ביצועים גבוהים אך אפשר לראות שהיא לא מפרידה את הפיצ'רים של כל מחלוקת חוץ משכבה האחורונה וזה עוללה להיות הסיבה לקפיצה שראינו בגרף.