Assignment Report

Deep learning 1402

Dr. Fatemizade

HW3

By Mohammad Amin Molaei Arpanahi

402011148

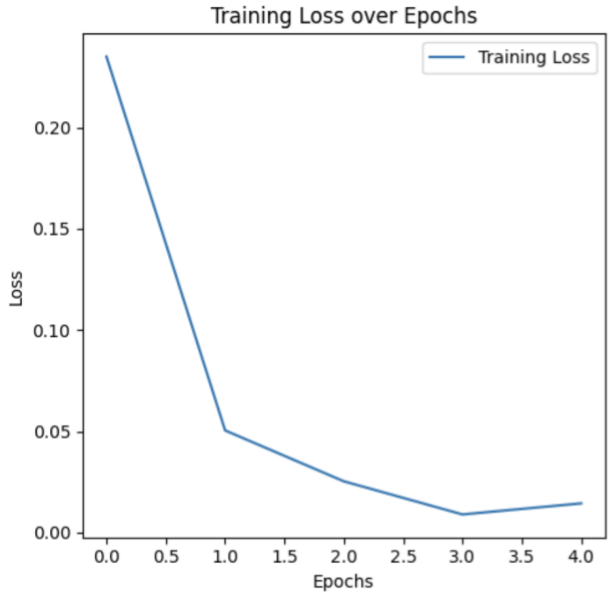
Q1

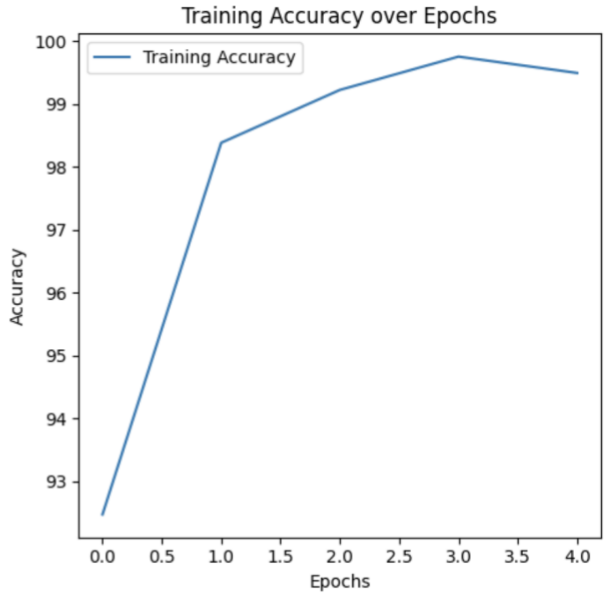
Part one

In the initial phase, I loaded the CIFAR dataset and filtered it to exclusively include images of cars and airplanes. Subsequently, I fine-tuned a neural network, opting for ResNet-50, which possesses an embedding space (size = (2048,))in its final fully connected layer. After attaching this layer, I added a sigmoid activation to compute the probabilities for the binary classes.

During training, I focused on learning only the weights of the last layer, while not setting the requires_grad attribute to True for all parameters. By manually adjusting the learning rate and batch size for the data loader, I achieved a training accuracy of approximately 100%. Moreover, the accuracy on the test dataset reached around 98%, indicating promising performance.

The corresponding visualizations for this phase are presented below.



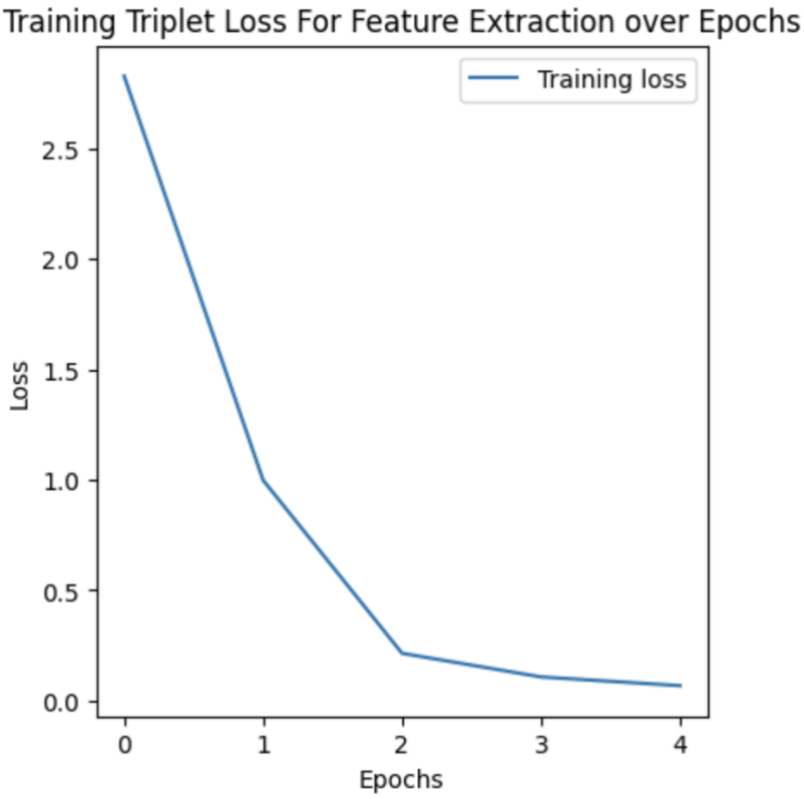


Part Two

In this phase, I constructed a dataset comprising anchor, positive, and negative samples to accurately compute the loss function in each epoch or batch. This structured dataset ensures the appropriate calculation of loss by pairing anchors with their corresponding positives and negatives, a requisite for proper training and optimization in each iteration.

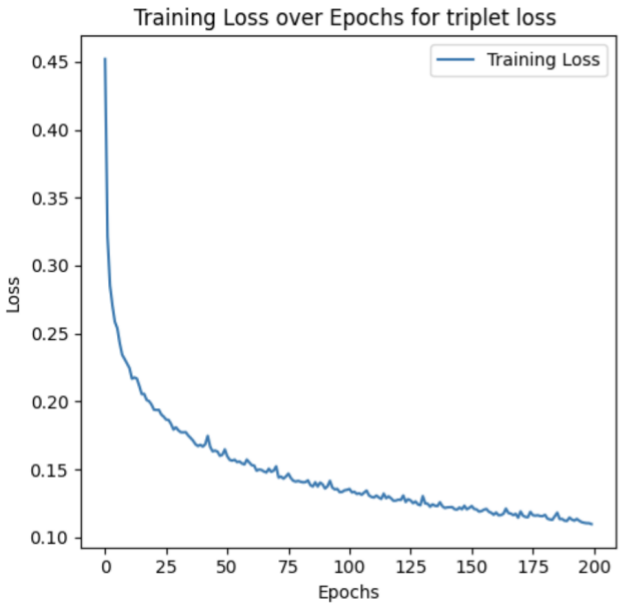
I commenced by training a pre-trained ResNet-50 model to optimize feature extraction. This process aimed to generate embeddings where the representations of different classes are located farther apart in the feature space. Subsequently, I tailored the features specifically for the dataset.

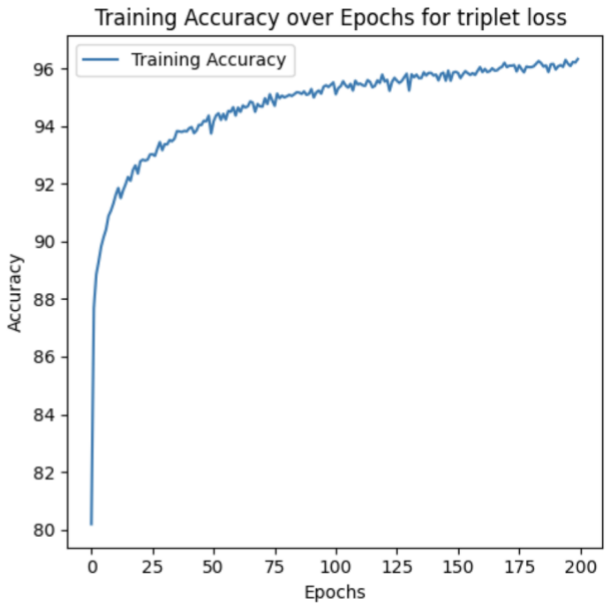
I implemented the triplet loss function, which is indicative of the minimum distance between similar data points and the maximum distance between dissimilar data points. The loss trends are visualized in the plot below.



Upon completing the feature extraction phase, I appended a fully connected layer at the end of the model. This augmentation aimed to facilitate classification tasks, particularly to discern between cars and airplanes. Subsequently, I proceeded to train the modified model using the extracted features to enhance the classification accuracy.

The corresponding visualizations for this phase are presented below.

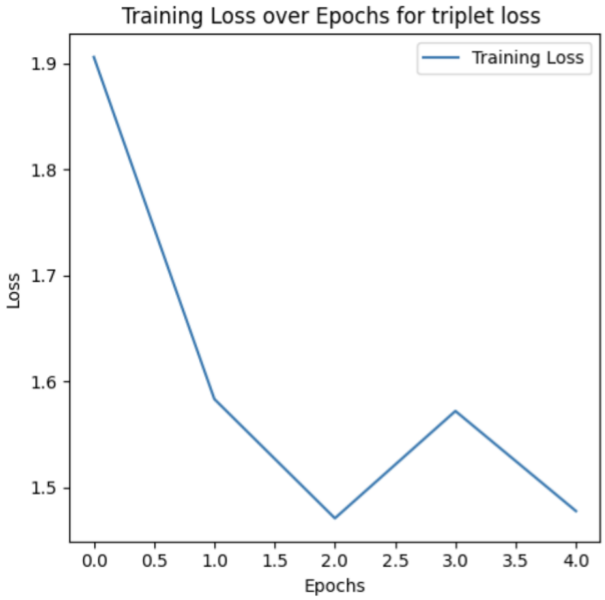


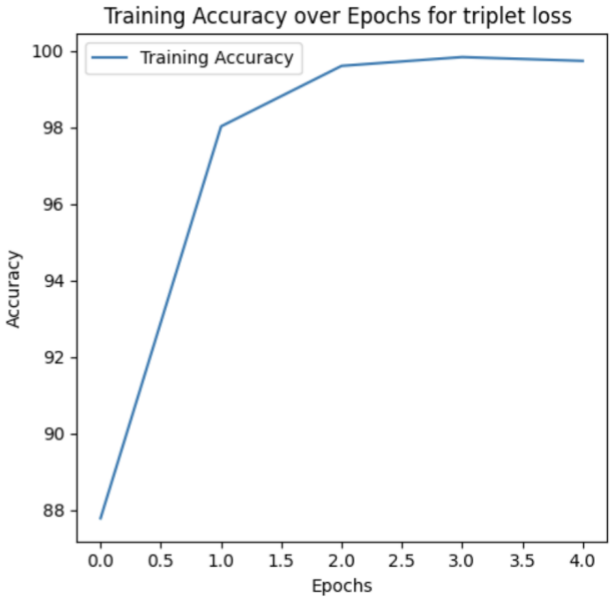


Part Three

In the final segment of this assignment, I incorporated the calculation of triplet loss alongside binary cross-entropy loss during each epoch and batch. This combined approach enabled simultaneous training of the model to minimize both losses. For every batch iteration, I directly generated anchor, positive, and negative data from the dataset used in the initial phase. This approach was chosen as it necessitated access to the labels for BCE loss.

Upon completing the training phase, the model achieved the following results. It's important to highlight that no model selection techniques were employed, and the learning rate and batch size were determined through multiple trials and experimentation.


Training Loss over Epochs for triplet loss
Training Loss
Loss
Epochs
1.9
1.8
1.7
1.6
1.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Training Accuracy over Epochs for triplet loss
Training Accuracy
Accuracy
Epochs
100
98
96
94
92
90
88
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Q2

Theory Part

a) Conventional CNNs perform correlation by applying the kernel's weights to a fixed filter across all channels of the input. However, in deformable convolutional neural networks, the grid used for sampling isn't consistent each time. Instead, it adapts with offsets for every block, enhancing flexibility in grid sampling. This adaptive approach aims to bolster the network's robustness against deformed inputs, such as those that are scaled or rotated.

b) When we train the grid sample offsets, we're essentially teaching the filter how to better align with the input image. For instance, when the image is scaled, these grid sample offsets extend from the center, enabling a more comprehensive incorporation of information into the model. Consequently, using deformable convolutions in tasks such as semantic segmentation can significantly enhance the accuracy of localization. This adaptability allows the model to efficiently capture intricate details and improve the precision of identifying object boundaries or regions within the image. Indeed, an important insight from this concept is that the conventional fixed grid might not efficiently capture all the relevant information within images. By employing learned grids, the network can adapt to extract maximum correlations between the weights of kernels and the input data. This adaptability allows the model to dynamically adjust its receptive field, enabling it to focus on the most relevant features in different regions of the input image. Consequently, this adaptive approach potentially enhances the

network's ability to extract crucial details and relationships, improving its overall performance in understanding complex visual patterns.

c) Convolutional neural networks traditionally employ the same fixed grid for all kernels, which poses challenges when the input undergoes spatial deformations. Particularly, images processed with square filters struggle to capture crucial correlations between the input and the kernel weights when subjected to spatial deformations. For instance, after rotation, the alignment between the input and the kernels is disrupted, leading to difficulties for the model when encountering inputs with varying degrees of rotation. To address this issue, techniques like deformable CNNs offer a solution by introducing more flexibility to accommodate spatial differences. Deformable CNNs adapt their grids, allowing the model to better align with and capture essential information from inputs undergoing spatial transformations. This increased flexibility enables the network to effectively handle spatial variations, including rotations, thereby enhancing its robustness and performance in scenarios involving spatially diverse inputs.

d) To compute the offsets in a deformable convolutional layer, it's essential to establish a convolutional layer that takes the input feature maps and outputs a number of sampling points as channels. By doing so, the offsets can be learned through interactions between the input feature map and the desired output target. This approach enables the model to learn the necessary spatial transformations or shifts needed to adjust the sampling grid dynamically. Consequently, the deformable convolutional layer can adapt its sampling points based on the learned offsets, allowing it to capture more informative features and effectively handle spatial variations in the input data.

Practical Part

It seems like you're comparing two CNN architectures—one using conventional convolutional layers (Conv2D) and the other using deformable convolutional layers. Both architectures comprise two convolutional blocks, each composed of Conv2D/DeformableConv2D, BatchNormalization, ReLU activation, and MaxPooling layers. Additionally, there's a fully connected layer at the end that matches the output neurons with the output channels of the preceding convolutional layers.

Your observations indicate that both models demonstrate improvements in loss and accuracy across epochs. However, the model employing deformable convolutional layers achieves higher performance metrics compared to the model using conventional convolutional layers within the same epoch.

If you have specific results or data from these models recorded in a notebook and would like to discuss the outcomes or delve deeper into the analysis or specific findings, please provide additional details or information from the recorded results. This way, I can assist you further based on the observed data and insights obtained from these models.

CNN

Accuracy on train: 88.99

Accuracy on test: 72

Average loss: 1.6098

Running Time: 9 minutes and 16 seconds

Deformable CNN

Accuracy on train: 90.03

Accuracy on test: 72

Average loss: 1.5965

Running Time: 10 minutes and 4 seconds

Q3

Data preparation

To prepare my dataset for training, validation, and testing, I developed functions to perform displacement, rotation, and scaling on the images. The displacement function involves randomly shifting the image by a certain percentage and returning the shifted image. For rotation, I applied a random degree rotation within the range of 15 to 30 degrees and then selected the largest image without any dark areas, saving the index of the largest square for cropping the original image. This step helped eliminate scaling biases in the rotation class.

Regarding the scaling function, I resized the image by selecting a random coefficient and then cropped it back to its original shape. This resulted in the image being scaled up by a factor of 1 divided by the scaling rate. After applying these functions to the original images, I created a data loader with a batch size of 32 due to limitations in GPU RAM.

I also passed some input to my data sets so you can change that and train the model in very shorter time.

I also adjusted the label classification to enhance the understanding of the images. The images corresponding to each label are displayed below.

Figure 1 – Unchanged image (label = 0)

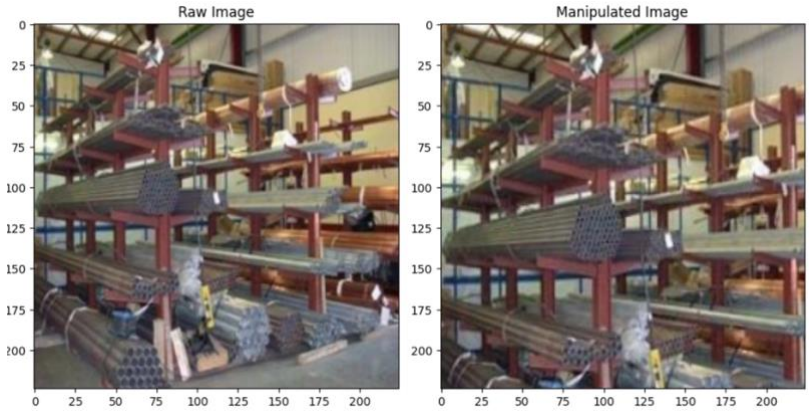


Figure 2 – Displacement (label = 1)

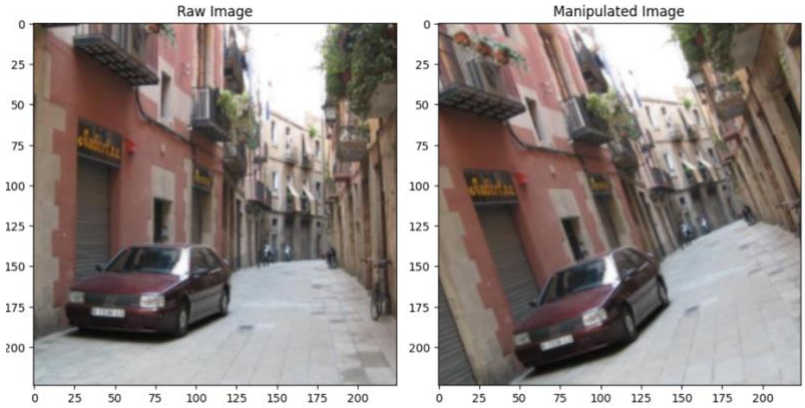


Figure 3 – Rotation (label = 2)

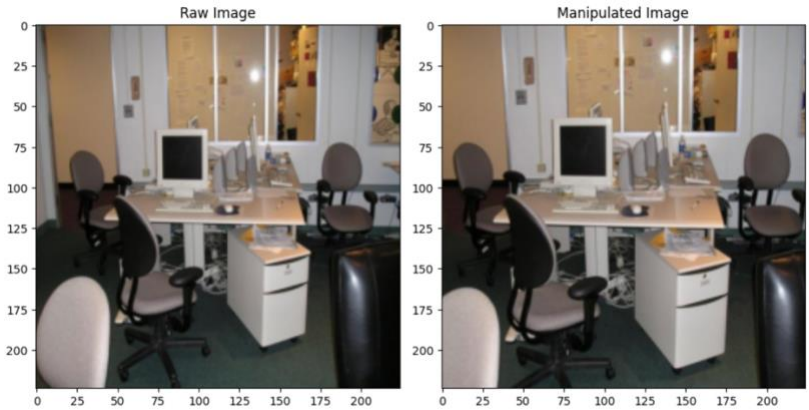


Figure 4 – Scaling (label = 3)

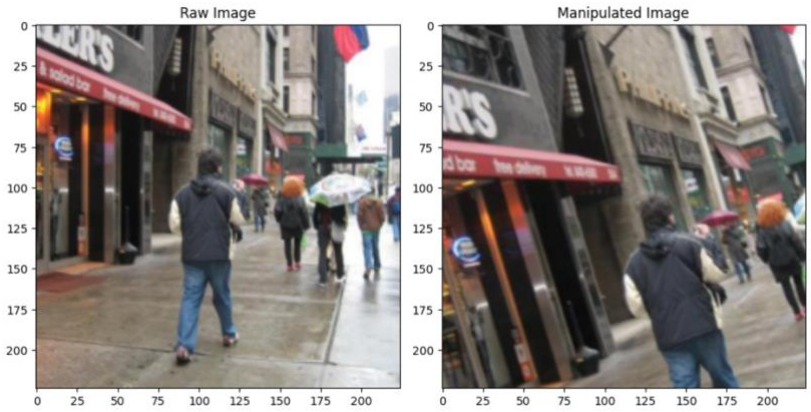


Figure 5 – Displacement and Rotation (label=4)

Figure 6 – Displacement and Scaling (label=5)

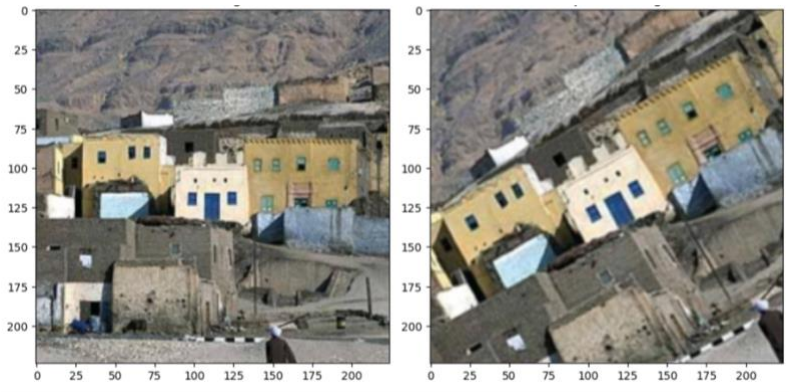


Figure 7 – Rotation and Scaling (label=6)

Train and Validation

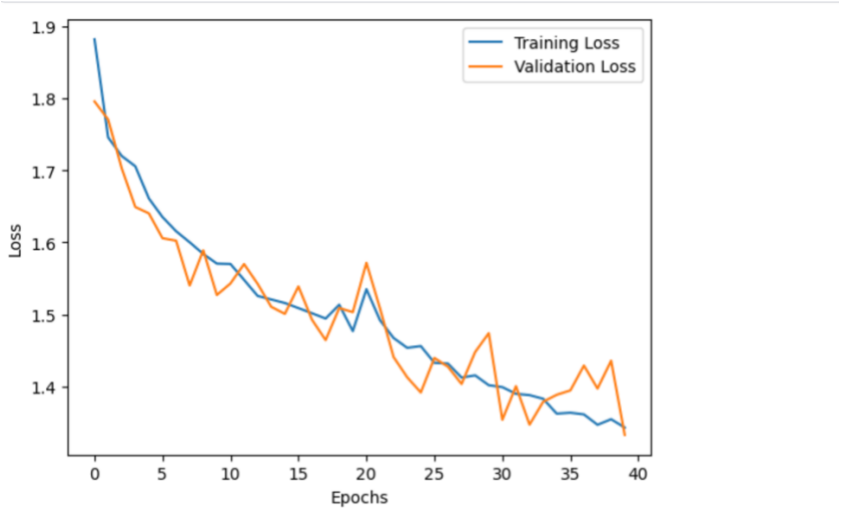
After setting up the data loaders, I populated the classes to construct the "changer" model, designed to detect image manipulations. Once these initial steps were completed, I commenced training the model to update its parameters and classify paired images. Simultaneously, I evaluated the model's performance by assessing the error rate and accuracy on the validation set to determine its effectiveness on unseen data. Finally, I saved the trained model.
I utilized the following hyperparameters for my model:

- Optimizer: Adam
- Learning rate: 0.0001
- Number of epochs: 20
- Batch size: 32

The obtained results after training are as follows:

- Accuracy on the training set: 94.6%
- Accuracy on the validation set: 90%
- Average training loss: 1.28
- Average validation loss: 1.30

Below is the plot depicting the training and validation loss versus the epoch number.



Testing

In this phase, I tested my model on 4000 input data samples containing pairs of original images and their corresponding changed images. The model achieved an accuracy of **85 percent** on the test set.
I employed the scikit-learn module to compute the confusion matrix, which provided insights into the accuracy for each class and misinterpreted classes. The confusion matrix is presented below.

Confusion Matrix:
[[495 0 0 0 0 0 0]
[0 447 0 0 3 29 1]
[3 2 394 24 11 17 24]
[0 5 0 468 0 6 0]
[1 1 4 0 360 105 4]
[0 19 0 2 4 427 0]
[1 2 69 40 32 105 247]]

The model has better performance for some classes.
The accuracy for one change is somehow better than the input including two changes.
My model has a little lower accuracy for rotation + scaling classes.

- I added some lines of code due to free the torch GPU RAM.
- I Answered the questions in notebook as below.

سوالات تئوری:

سوال تئوری1:
مقادیر زیر را برای لایه های کانولوشن چه قدر در نظر گرفته اید؟
channel out size (conv(1*1))
padding(conv)
دلیل استفاده از کانولوشن 1در1 چیست؟

با توجه به این که دو مسیر با یکدیگر جمع می شوند باید تعداد کانال ها در مسیر سمت راست با تعداد کانال در
مسیر سمت چپ برابر باشد به عبارتی چون خروجی بعد از جمع شوند تغییر کانال نمیدهد تعداد کانال های
خروجی دقیقا باید برابر تعداد کانال خروجی بلاک ، استیج که در شکل اول قابل مشاهده است برداشت شود .

پدینگ در این کانولوشن یک در یک طبیعتا باید صفر باشد چرا که در صورت انتخاب حاشیه اطلاعاتی از خود ورودی به دست نمی آید.

کانولوشن یک در یک برای یادگیری اطلاعات کانال و تغییر تعداد کانال به صورت قابل یادگیری استفاده می شود.

بررسی کنید عکس ها با یک ،2 یا 3 تغییر کدام بهتر تشخیص داده میشوند؟

با توجه به کانفیوجن ماتریس بالا مدل به طور کلی به ترتیب برای کلاس های بدون تغییر و کلاس های با یک تغییر و دو تغییر بهترین عملکرد را داراست .