CS3612: FINAL PROJECT

Jiaqi Xue

520030910246

Computer Science and Technology (IEEE)

xuejiaqi@sjtu.edu.cn

ABSTRACT

This final project aims at helping students have a deep insight on machine learning, including an intuitive understanding of the feature and the model. This project includes a mandatory task and two optional tasks. Considering that I do not have sufficient computing resources, I choose to train LeNet and a network of my own design on simplified fashion-MNIST dataset. As for the optional task, I select the second optional task to try out some visualization methods to explain DNNs.

1 Mandatory Task: Fashion-MNIST clothing classification

1.1 Implement LeNet

LeNet, short for LeNet-5, is a convolutional neural network (CNN) architecture that was developed by Yann LeCun and his colleagues in the late 1990s[1]. It was one of the pioneering models in the field of deep learning and played a crucial role in advancing computer vision tasks, particularly handwritten digit recognition.

LeNet consists these elements:

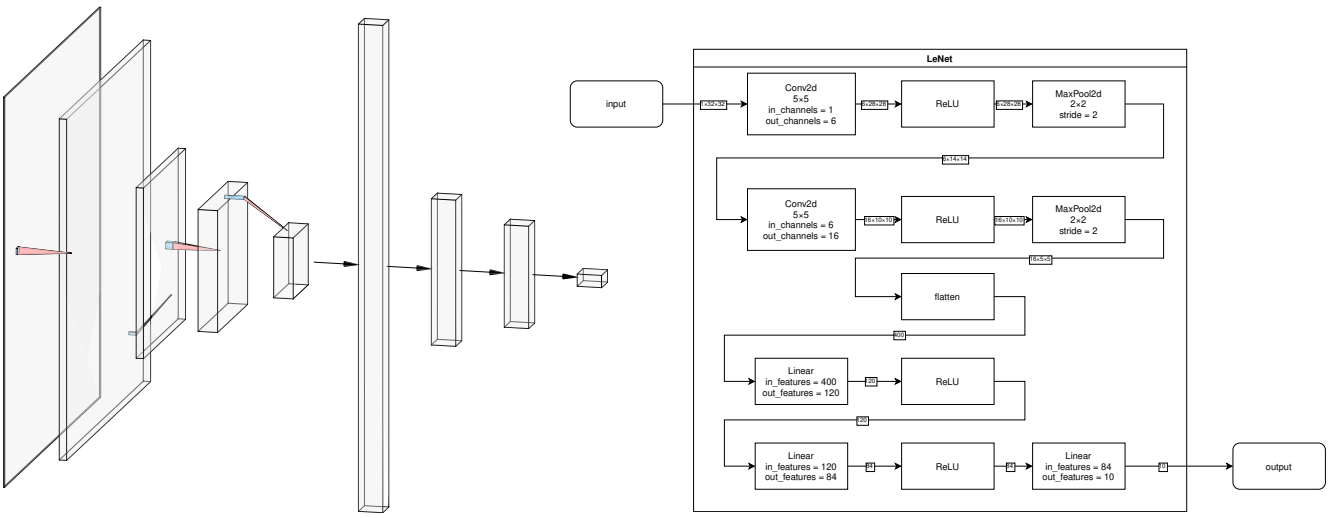
1. Input Layer: The network takes a grayscale image of size 32x32 pixels as input.
2. Convolutional Layers: LeNet starts with two convolutional layers, each followed by a subsampling layer. The convolutional layers use small filters with a receptive field of 5x5 and apply a convolution operation to extract features from the input image.
3. Subsampling Layers: These layers perform down-sampling or pooling operations to reduce the dimensionality of the feature maps and provide translation invariance. LeNet uses average pooling with a filter size of 2x2.

4. Fully Connected Layers: After the convolutional and subsampling layers, the feature maps are flattened into a vector and fed into two fully connected layers. The first fully connected layer consists of 120 neurons, while the second one has 84 neurons.

5. Output Layer: The final fully connected layer connects to the output layer, which consists of 10 neurons, representing the 10 possible classes.

6. Activation Function: Throughout the network, a non-linear activation function, typically the sigmoid or hyperbolic tangent (tanh), is applied to introduce non-linearity and enable the model to learn complex relationships in the data.

Having a basic understanding of LeNet, I choose to implement the LeNet via pytorch. Considering the size of the input image is 32×32, the structure of the Lenet is shown in Figure 1.



(a) LeNet Visualization (b) LeNet Diagram

Figure 1: Structure of LeNet

1.2 Design and Implement My Neural Network

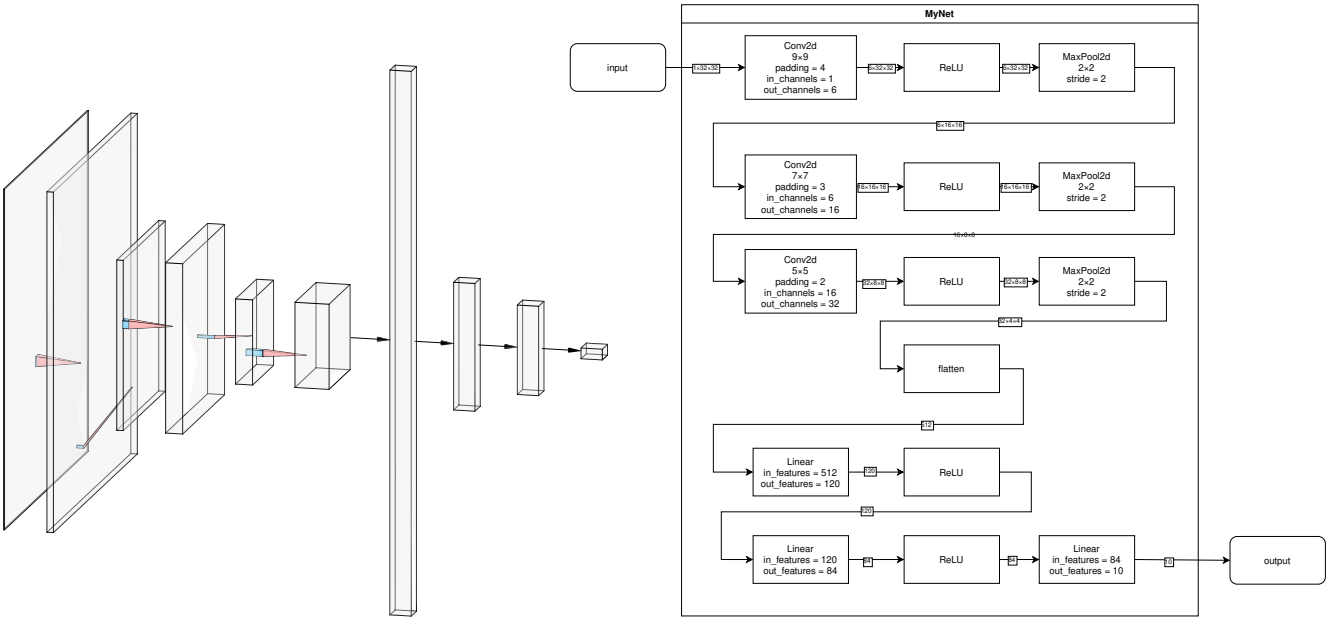
Based on the network structure of LeNet, I design and implement a convolutional neural network. Hereinafter referred to as "MyNet". The detailed structure of MyNet is shown in Figure 2.

The basic idea to design MyNet are:

1. Deeper architecture: Increase the depth of the network by adding more convolutional and fully connected layers. Deeper architectures have been shown to learn more complex representations, which can lead to improved performance.

2. Adjust Network size: Adjust the number of filters in the convolutional layers and the number of neurons in the fully connected layers. Increasing the number of filters/neurons may capture more complex features.

Compared with LeNet, MyNet consists one more convolutional layers and the size of fully connected layers is also larger. This may capture more complex features and lead to improved performance.



(a) MyNet Visualization (b) MyNet Diagram

Figure 2: Structure of MyNet

1.3 Training and Testing

After implementing the two networks, we can perform training and testing on simplified fashion-MNIST dataset.

I select the Stochastic Gradient Descent (SGD) as the optimizer. The goal of SGD is to minimize an objective function by iteratively adjusting the model parameters based on the gradients of the loss function with respect to those parameters. It helps in finding the best possible values for the model's parameters that minimize the difference between predicted and actual values.

Learning rate is an important hyperparameter in SGD. It determines the step size at each update and influences the convergence and stability of the optimization process. A high learning rate may result in overshooting the optimal solution, while a low learning rate may lead to slow convergence. Therefore, finding an appropriate learning rate is crucial for achieving good performance. The final learning rate I choose is 10^{-3}.

And the momentum factor is set to 0.9. Momentum helps accelerate the optimization process and enhance the convergence of the algorithm. SGD with momentum maintains a momentum vector to track the direction of previous

gradients. During each iteration, the gradient not only affects the current parameter update but also contributes to the momentum vector. The update equation for the momentum vector is as follows:

$$v = \beta \cdot v - \alpha \cdot \nabla J(\theta)$$

and the parameter update equation is:

$$\theta = \theta + v$$

Where

1. v is the momentum vector.
2. β is the momentum coefficient, indicating the contribution of previous gradients to the current update.
3. α is the learning rate.
4. $J(\theta)$ represents the gradient at the current iteration.

After trainign for 200 epochs, we can obtain the curves of training and testing loss of two nets. Shown in Figure 3.

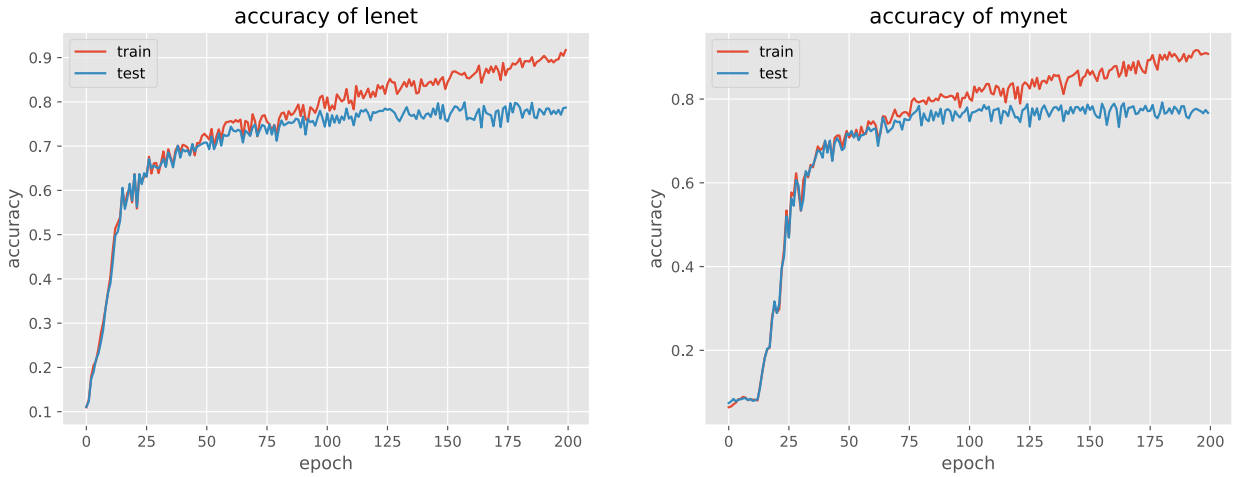
(a) Loss of LeNet

(b) Loss of MyNet

Figure 3: Loss curves

It can be seen that the loss of MyNet goes higher after it lowered, which means the model comes to overfitting while LeNet does not. This is reasonable because MyNet is deeper and larger than LeNet and can learn more complex representations and also more likely to goto overfitting.

We can also obtain the training and testing accuracy of two nets. Shown in Figure 4.



(a) Accuracy of LeNet

(b) Accuracy of MyNet

Figure 4: Accuracy curves

It can be seen that the accuracy of LeNet and MyNet is almost the same after training for 200 epochs. For both networks, the accuracy of the training set is higher than the accuracy of the test set.

1.4 PCA Feature Visualization

1.4.1 PCA Algorithm

Principal component analysis (PCA) is a method that projects each data point onto only a few principal components to obtain lower-dimensional data, while preserving as much of the variance of data as possible.

Algorithm 1 shows the detailed process of PCA.

Algorithm 1 Principal Component Analysis (PCA)

Input: Dataset X with n samples and m features
Output: Principal components PC and transformed data Y
Subtract the mean from each feature in X:
mean $\leftarrow$ mean(X)
$X \leftarrow X -$ mean
Compute the covariance matrix:
$C \leftarrow \frac{1}{n-1} X^T X$
Perform eigenvalue decomposition on C:
$V, D \leftarrow \text{eig}(C)$ *(V: eigenvectors, D: eigenvalues)*
Sort the eigenvalues and eigenvectors in descending order:
Sort D and V based on D in descending order
Select the k eigenvectors with the largest eigenvalues:
$PC \leftarrow V(:, 1:k)$ *(PC: principal components)*
Transform the data using the selected principal components:
$Y \leftarrow X \cdot PC$
return PC, Y

1.4.2 Mid-Level Features

The selected three mid-level features of LeNet is shown in Figure 5. Including a convolutional layer, a fully-connected layer, and the final layer of the neural network.

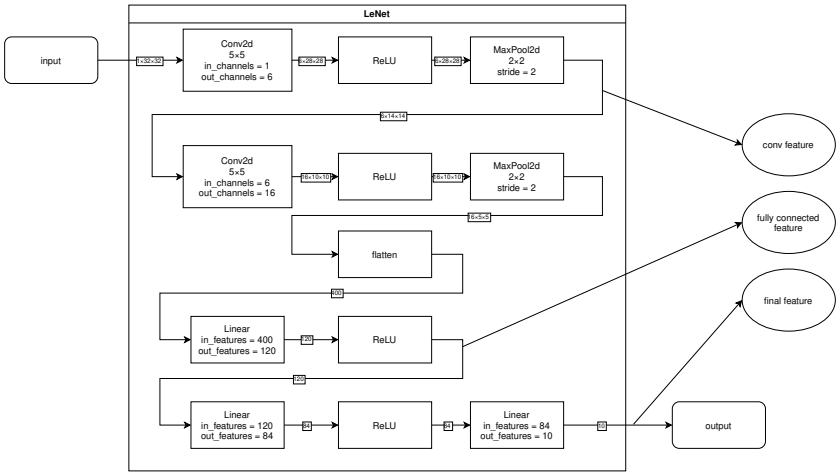


Figure 5: LeNet mid-level features

The selected three mid-level features of MyNet is shown in Figure 6. Including a convolutional layer, a fully-connected layer, and the final layer of the neural network.

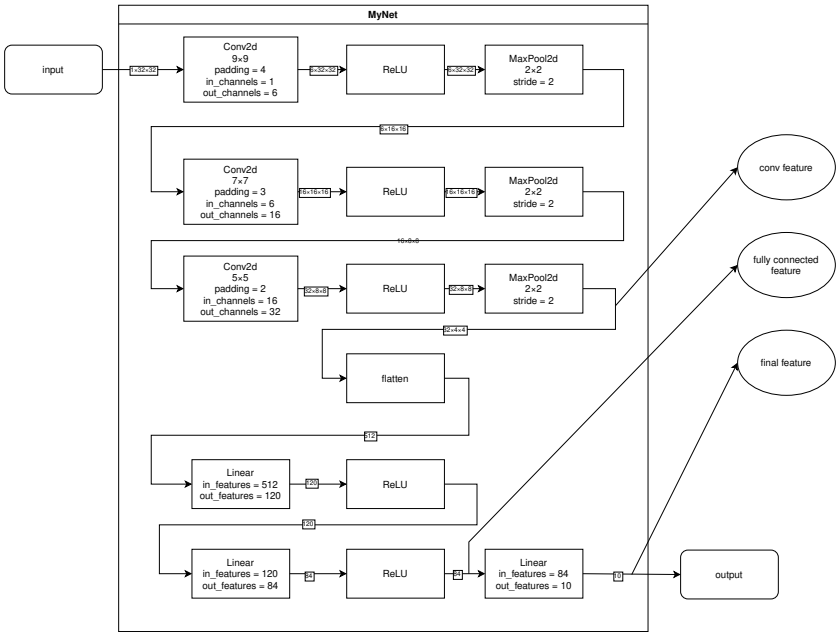


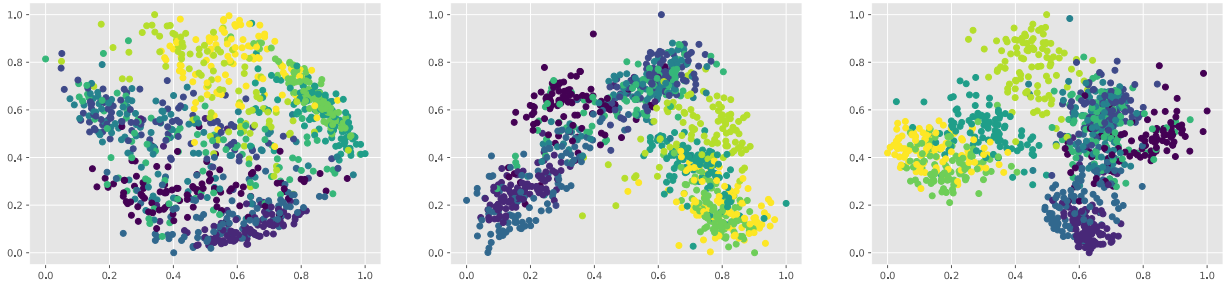
Figure 6: MyNet mid-level features

1.4.3 PCA Result

PCA allows you to reduce the number of variables or features in your data. It achieves this by creating new variables, known as principal components, which are linear combinations of the original variables. The result indicates how many principal components are required to capture most of the variation in the data.

PCA can be used to visualize high-dimensional data in a lower-dimensional space, typically in a two or three dimensional plot. The result of PCA can provide insights into the grouping or clustering of data points, the presence of outliers, and the overall structure of the data.

Apply PCA algorithm on the three mid-level features of LeNet. The result is shown in Figure 7.

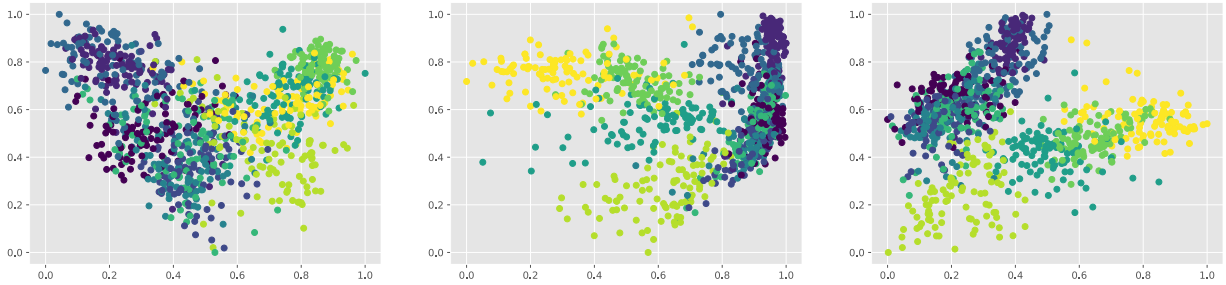


(a) convolutional layer (b) fully-connected layer (c) final layer

Figure 7: PCA Result of LeNet mid-level features

It can be seen that data points of the same color positions close to each other which shows the PCA algorithm and the LeNet is valid. More importantly, as the depth of the network increases, the different categories are more separated.

Apply PCA algorithm on the three mid-level features of MyNet. The result is shown in Figure 8.



(a) convolutional layer (b) fully-connected layer (c) final layer

Figure 8: PCA Result of MyNet mid-level features

The PCA results of NyNet were not as separated compared to the PCA results of LeNet. This may explain the poor performance of MyNet compared with LeNet. However, it still can be seen that with the network goes deeper, the different categories are more separated witch shows that the MyNet is also valid.

Figure 9 provides a more intuitive visualization by using the original image as data points. It shows that cloths of same class positions close to each other.

(a) convolutional layer of LeNet

(b) convolutional layer of MyNet

(c) fully-connected layer of LeNet

(d) fully-connected layer of MyNet

(e) final layer of LeNet

(f) final layer of MyNet

Figure 9: PCA Result with original images

1.5 t-SNE Feature Visualization

1.5.1 t-SNE Algorithm

t-Distributed Stochastic Neighbor Embedding(t-SNE) is a popular dimensionality reduction technique used in machine learning and data visualization. The primary goal of t-SNE is to transform high-dimensional data into a two or three dimensional representation while preserving the underlying relationships between data points.

The t-SNE algorithm aims to minimize the divergence between the two probability distributions: the one defined in the high-dimensional space and the one defined in the lower-dimensional space. It accomplishes this by optimizing a cost function using gradient descent. The optimization process iteratively adjusts the positions of data points in the lower-dimensional space to find an arrangement that captures the relationships present in the original data.

Algorithm 2 shows the detailed process of t-SNE.

Algorithm 2 t-SNE (t-Distributed Stochastic Neighbor Embedding)

Input: Dataset X with n samples and m features, perplexity $Perp$, learning rate η, number of iterations N_{iter}
Output: Low-dimensional representations Y
Compute pairwise distances between samples in X:
$D \leftarrow$ pairwise_distances(X)
Initialize the low-dimensional representations randomly:
$Y \leftarrow$ random_initialization(n, d) *(d: desired dimension)*
for $iter = 1$ **to** N_{iter} **do**
    Compute pairwise similarities between samples in the high-dimensional space:
    $P \leftarrow$ compute_similarity$(D, Perp)$ *(using perplexity-based method)*
    Compute pairwise similarities between samples in the low-dimensional space:
    $Q \leftarrow$ compute_similarity(Y) *(using Student's t-distribution)*
    Compute the gradient of the KL divergence between P and Q:
    $\frac{\partial C}{\partial Y} \leftarrow 4 \sum_{i,j} (P_{ij} - Q_{ij})(Y_i - Y_j)(1 + \|Y_i - Y_j\|^2)^{-1}$
    Update the low-dimensional representations:
    $Y \leftarrow Y + \eta \frac{\partial C}{\partial Y}$
    Apply early exaggeration (optional, in the initial iterations):
    if $iter < 100$ **then**
        $P \leftarrow P \times 4$ *(increase the pairwise similarities)*
    end if
    Normalize the low-dimensional representations:
    $Y \leftarrow$ normalize(Y) *(to have zero mean)*
end for
return Y

Lower the high dimension data points using PCA is a method to accelerate the process of t-SNE when the dimension is very high.

1.5.2 Mid-Level Features

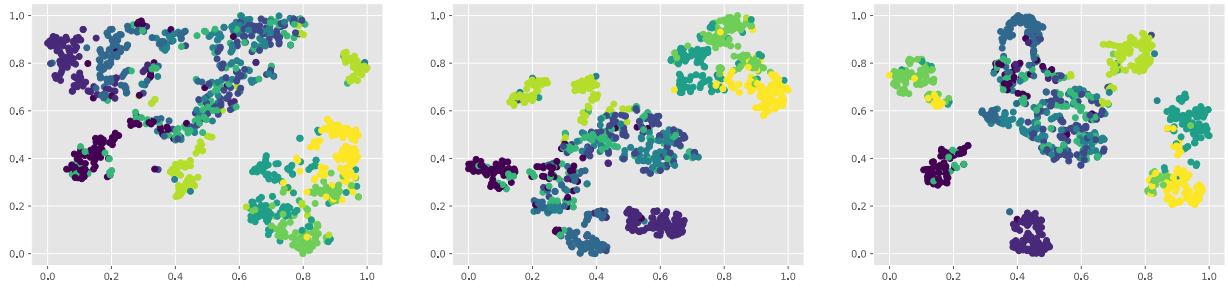
The selected three mid-level features of LeNet is shown in Figure 5. Including a convolutional layer, a fully-connected layer, and the final layer of the neural network.

The selected three mid-level features of MyNet is shown in Figure 6. Including a convolutional layer, a fully-connected layer, and the final layer of the neural network.

1.5.3 t-SNE Result

Data points that belong to the same cluster or have similar properties tend to cluster together in the t-SNE plot. Clusters can appear as distinct groups or tightly packed regions. Different classes or categories of data points may be visually separated in the t-SNE plot.

Apply t-SNE algorithm on the three mid-level features of LeNet. The result is shown in Figure 10.

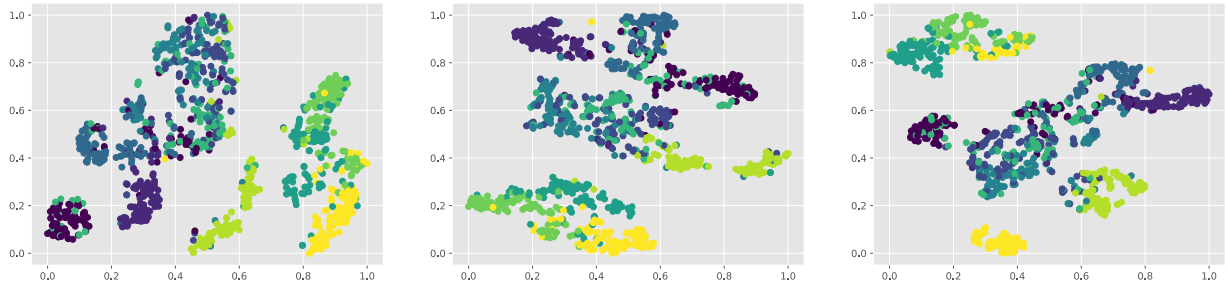


(a) convolutional layer (b) fully-connected layer (c) final layer

Figure 10: t-SNE Result of LeNet mid-level features

It can be seen that data points of the same color goes clustered which shows the t-SNE algorithm and the LeNet is valid. More importantly, as the depth of the network increases, the clusters gets more obvious.

Apply t-SNE algorithm on the three mid-level features of MyNet. The result is shown in Figure 11.



(a) convolutional layer (b) fully-connected layer (c) final layer

Figure 11: t-SNE Result of MyNet mid-level features

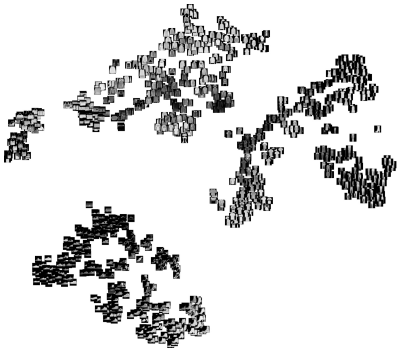
The t-SNE results of NyNet were not as clustered compared to the PCA results of LeNet. This may explain the poor performance of MyNet compared with LeNet.

Figure 12 provides a more intuitive visualization by using the original image as data points. It shows that cloths of same class get clustered.

(a) convolutional layer of LeNet

(b) convolutional layer of MyNet

(c) fully-connected layer of LeNet



(d) fully-connected layer of MyNet

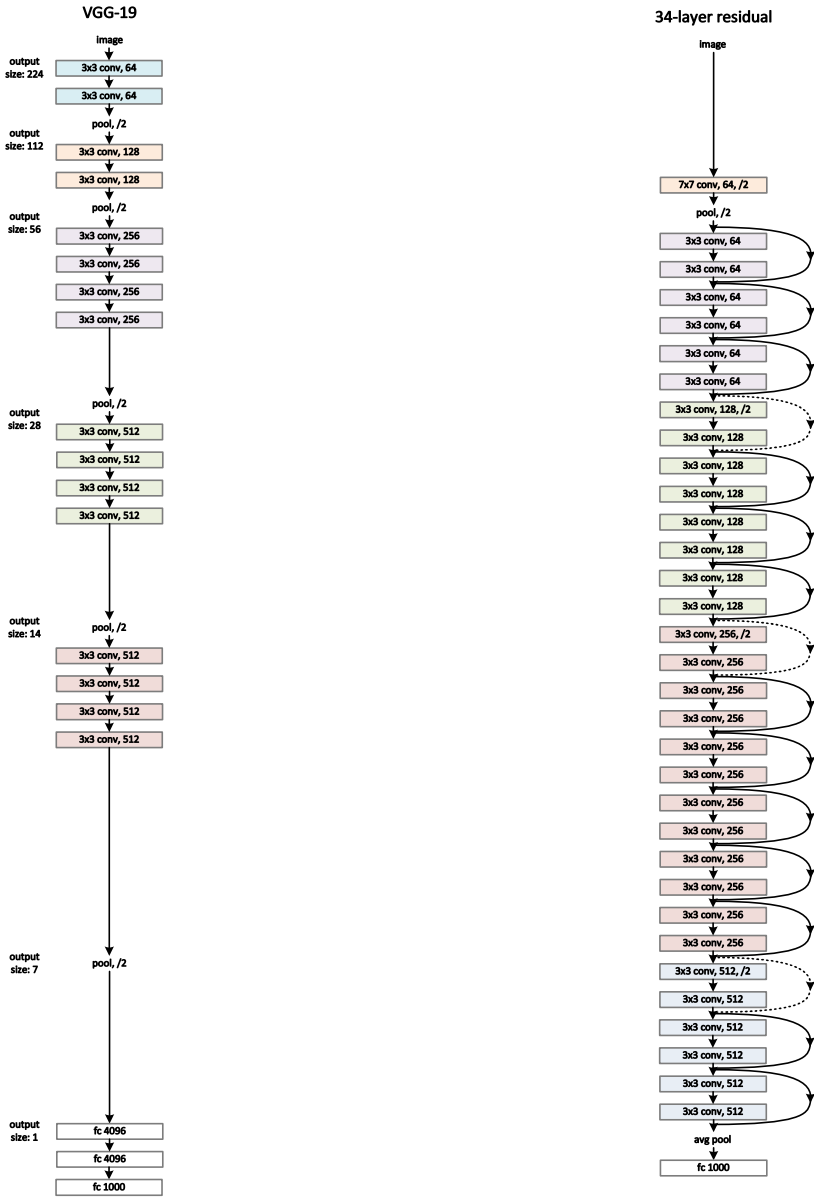
(e) final layer of LeNet

(f) final layer of MyNet

Figure 12: t-SNE Result with original images

2 Optional task: Visualization methods to explain DNNs

2.1 Pre-Trained DNN Models



(a) vgg19 (b) resnet34

Figure 13: Architectures of Pre-Trained Networks

The provided pre-trained DNN is resnet34[2] which is trained on Imagenet1k dataset[3]. The architecture of resnet34 is shown in Figure 13 (b).

I select another DNN named vgg19 which is also trained in Imagenet1k dataset. The architecture of vgg19 is shown in Figure 13 (a).

2.2 Grad-CAM

2.2.1 Grad-CAM Algorithm

Gradient-weighted Class Activation Mapping(Grad-CAM) is a visualization technique used to understand the decisions made by convolutional neural networks (CNNs) in image classification tasks. [4] It provides insights into the regions of an input image that are most influential in influencing the network's final prediction. The overview of Grad-CAM is:

1. Feed forward an input image through the network, and the class score is computed for the target class of interest.
2. Compute the gradients of the target class score with respect to the feature maps in the last convolutional layer. These gradients represent the importance of the feature maps for the target class.
3. Calculate the importance weights by averaging the gradients across the spatial dimensions of each feature map, resulting in a weight map.
4. Linearly combined the weight map is with the feature maps by taking the element-wise product.
5. Finally, sum the resulting weighted feature maps along the channel dimension and pass through a ReLU activation to obtain the Grad-CAM heatmap. The heatmap represents the regions in the input image that are most relevant to the network's prediction for the target class.

The detailed process of Grad-CAM is shown in Algorithm 3.

Algorithm 3 Grad-CAM (Gradient-weighted Class Activation Mapping)

Input: Pre-trained model M, input image I, target class index c
Output: Grad-CAM heatmap H
Forward pass through the model to obtain the output logits:
$O \leftarrow M(I)$
Compute the gradient of the target class logit with respect to the final convolutional layer feature maps:
$\frac{\partial O_c}{\partial A} \leftarrow \text{backprop}(O_c, A)$ *(A: final convolutional layer feature maps)*
Perform global average pooling (GAP) on the gradient:
$w \leftarrow \text{GAP}(\frac{\partial O_c}{\partial A})$
Initialize the Grad-CAM heatmap as zeros:
$H \leftarrow \text{zeros}(\text{spatial\_size}(A))$
Weight the feature maps with the gradient importance:
for each feature map a_i in A **do**
    $H \leftarrow H + w_i \times a_i$
end for
Apply ReLU activation to the Grad-CAM heatmap:
$H \leftarrow \text{ReLU}(H)$
Normalize the Grad-CAM heatmap:
$H \leftarrow \frac{H - \min(H)}{\max(H) - \min(H)}$
return H

The implementation of Grad-CAM is much easier via hooks in pytorch. For example, the hook on the last convolutional layer can be utilized in the code below:

```
class GradCAM:
    def __init__(self, model: torch.nn.Module, hook_layer: torch.nn.Module):
        self.model = model
        self.gradient: torch.Tensor
        self.activation: torch.Tensor
        hook_layer.register_forward_hook(
            lambda model, input, output: (
                output.register_hook(
                    lambda x: (setattr(self, 'gradient', x), None)[1]),
                setattr(self, 'activation', output.detach()), None
            )[2]
        )
```

Grad-CAM operates by utilizing the gradients of the target class score with respect to the feature maps in the last convolutional layer. And the argument `hook_layer` is the last convolutional layer of `model`.

2.2.2 Grad-CAM Result

When a CNN makes a prediction, Grad-CAM helps identify which parts of the image influenced that prediction the most. It produces a heatmap that highlights the regions of the image that the network focused on during the decision-making process. The intensity of the heatmap indicates the importance of each region, with higher intensity regions being more influential in the prediction.

The test image in Figure 14 (a) contains a cat and a dog. Accroding to Imagenet1k dataset, index 242 represents boxer dog and index 282 represents tiger cat. Applying Grad-CAM algorithm on both resnet34 and vgg19 with target labels being cat and dog, the result is shown in Figure 14.

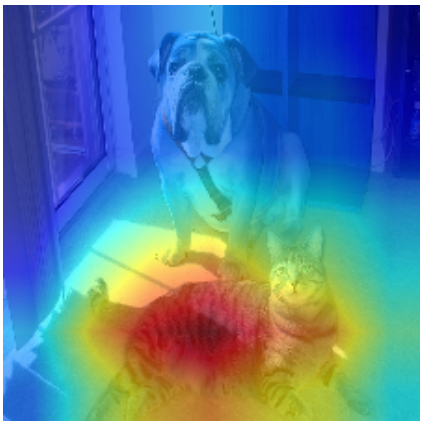
We can conclude that for both networks, the target label and the heated region are correctly matched. Which indicates that the both neural network and the implementation of Grad-CAM algorithm is valid.

Accroding to Figure 14 (d), the basis of determining that the image is a dog is mainly from the head region of the dog. And accroding to Figure 14 (e), the basis vgg19 used to determin that the image is a cat is from the head region and the body region. While resnet34 recognizes the whole cat region as the determination basis.

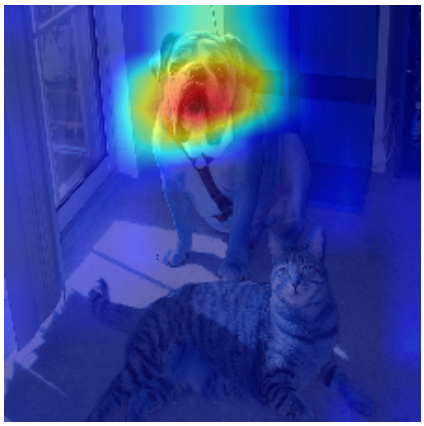
It can be seen that the heated region of resnet34 is larger than the region of vgg19. This might because resnet34 is more deeper than vgg19 and contains more convolutional layers. Thus the receptive field of resnet34 is larger than vgg19 and the heated region in heatmap is larger.

(a) original image

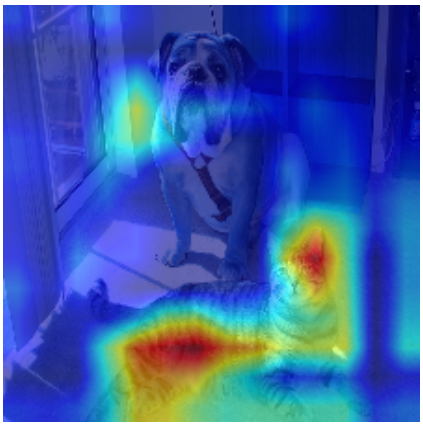
(b) resnet34 with target dog



(c) resnet34 with target cat



(d) vgg19 with target dog



(e) vgg19 with target cat

Figure 14: Grad-CAM result

By visualizing the important regions, Grad-CAM provides insights into how the network "sees" and interprets the input image. It helps human understand which image features are relevant for the network's decision, enabling them to validate the model's behavior, diagnose potential biases, and gain insights into its internal workings.

2.3 Integrated Gradients

2.3.1 Integrated Gradients Algorithm

Integrated Gradients is a technique used in the field of interpretability and explainability of machine learning models. [5] It aims at providing insights into the decision-making process of complex models, such as deep neural networks, by attributing importance or relevance scores to input features or pixels.

The process of computing Integrated Gradients including:

1. Choose a baseline input as a reference point. Usually a black image.
2. Use linear interpolation to obtain the path between the baseline and the input.
3. Compute the gradients of the model's output with respect to the input features at each intermediate point.
4. Integrated the computed gradients along the path from the baseline to the input. This integration step aggregates the gradient information and provides an attribution score for each feature.
5. Scale the integrated gradients by the difference between the input and baseline and average them over multiple random paths to obtain the final attribution scores.

The detailed process of Integrated Gradients is shown in Algorithm 4.

Algorithm 4 Integrated Gradients

Input: Model M, input example x, baseline example x', number of steps N
Output: Attributions A
Compute the input difference:
$\Delta x \leftarrow x - x'$
Initialize the integrated gradients attributions:
$A \leftarrow 0$
for $i = 1$ **to** N **do**
    Compute the interpolated input:
    $x_i \leftarrow x' + \frac{i}{N} \cdot \Delta x$
    Compute the gradient of the model output with respect to the interpolated input:
    $g_i \leftarrow \text{gradient}(M(x_i))$
    Accumulate the attributions:
    $A \leftarrow A + \frac{\Delta x}{N} \odot g_i$ *(element-wise multiplication)*
end for
return A

As for implementation, the compute of gradient can be done by pytorch via the code below:

```
x.requires_grad_()
output = self.model(x)[0, target]
grad = torch.autograd.grad(output, x)[0]
```

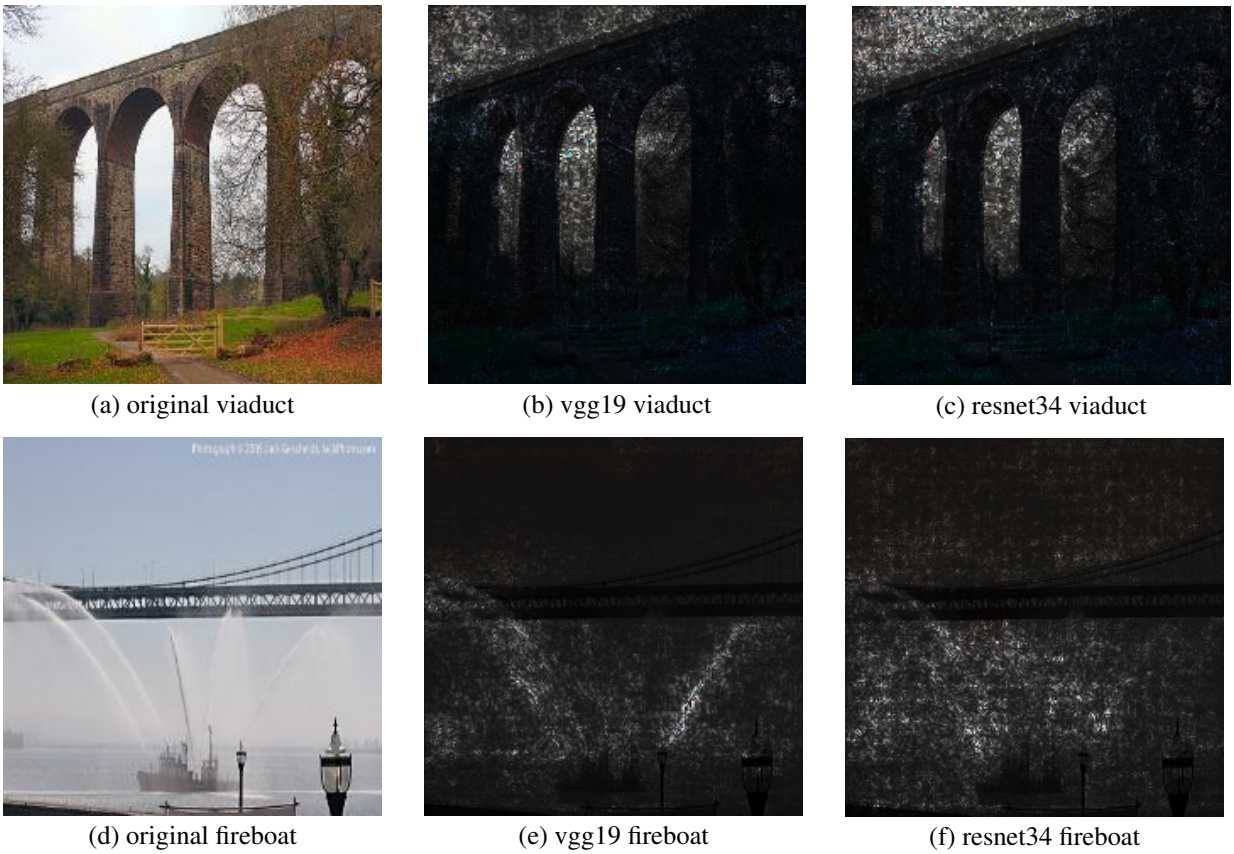
`requires_grad_` method is used to enable the gradient tracking for the tensor.

2.3.2 Integrated Gradients Result

The result of Integrated Gradients indicates the importance or contribution of each feature or input to the overall prediction made by a machine learning model. Integrated Gradients is an interpretability technique that aims to explain the predictions of deep learning models by attributing relevance scores to the input features.

The image in Figure 15 (a) is a viaduct and the image in Figure 15 (d) is a fireboat.

Performing Integrated Gradients algorithm on vgg19 net and resnet34 using input image Figure 15 (a) and Figure 15 (d) gets the output in Figure 15.



(a) original viaduct (b) vgg19 viaduct (c) resnet34 viaduct

(d) original fireboat (e) vgg19 fireboat (f) resnet34 fireboat

Figure 15: Integrated Gradients result

It can be seen that the lighted pixels correctly matches the important regions of the input image. This shows that the pre-trained model and the implementation of Integrated Gradients are valid.

Compared with the output of vgg19 which is clear, the output of resnet34 is blurred with lots of scattered light pixels. This may be related to the structure of the residual network. The skip connections in resnet enable the information to flow through the network more easily during both the forward and backward passes, so it may also make the light pixels of Integrated Gradients superimposed and become more complicated.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1.1

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2.1

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 2.1

[4] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, oct 2019. 2.2.1

[5] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks, 2017. 2.3.1