**COMPUTER VISION ASSIGNMENT\_9**

**1.What are the advantages of a CNN for image classification over a completely linked DNN?**

Convolutional Neural Networks (CNNs) have several advantages over fully connected Deep Neural Networks (DNNs) for image classification:

Spatial Invariance: CNNs use convolutional layers with filters that scan the image and identify features, which are then passed on to the next layer. This allows the network to learn spatial hierarchies of features, and be invariant to translations in the input.

Parameter Efficiency: The number of parameters in a CNN is much lower compared to a fully connected network, which helps to prevent overfitting, and also makes training faster.

Ability to capture Spatial Structures: CNNs are specifically designed to exploit the 2D structure of image data, by using filters to scan the image and identify features. This is beneficial for image classification tasks as the spatial structures within images often contain important information for recognizing objects.

Reduced Computational Complexity: The convolutional layers in a CNN share their parameters, which significantly reduces the number of parameters to be learned. This leads to reduced computational complexity, which makes CNNs more feasible to train and use for large-scale image classification tasks.

In summary, CNNs are better suited for image classification tasks compared to fully connected DNNs due to their ability to capture spatial hierarchies of features, their reduced parameter and computational complexity, and their ability to learn spatial structures within images.

**2. Consider a CNN with three convolutional layers, each of which has three kernels, a stride of two, and SAME padding. The bottom layer generates 100 function maps, the middle layer 200, and the top layer 400. RGB images with a size of 200 x 300 pixels are used as input. How many criteria does the CNN have in total? How much RAM would this network need when making a single instance prediction if we’re using 32-bit floats? What if you were to practice on a batch of 50 images?**

For a single convolutional layer, the number of parameters in the kernels (also known as filters) is given by:

parameters = (kernel\_width \* kernel\_height \* number\_of\_input\_channels + 1) \* number\_of\_kernels

In this case, each convolutional layer has three kernels, with a kernel size of (presumably) the same size as the input image, 200 x 300 x 3 = 180,000. Therefore, the number of parameters in each layer is:

parameters = (200 \* 300 \* 3 + 1) \* 3 = 540,900

So, the total number of parameters in the three convolutional layers is:

total\_parameters = 3 \* 540,900 = 1,622,700

To calculate the memory requirements for making a single instance prediction, we need to know the size of a float, which is 4 bytes. Therefore, the total memory requirement for the network weights (assuming all parameters are float32) is:

memory\_requirement = 4 \* 1,622,700 = 6,490,800 bytes = 6.2 MB

To calculate the memory requirements for making predictions on a batch of 50 images, we need to add the memory required for the activations and intermediate computations, which are proportional to the number of neurons in the network. Assuming an average of 1000 neurons per layer, the memory requirements for a batch of 50 images would be approximately:

memory\_requirement = 6.2 MB \* 50 + 1000 \* 50 \* 4 bytes = 320 MB

So, the network would need approximately 320 MB of RAM to make predictions on a batch of 50 images.

**3. What are five things you might do to fix the problem if your GPU runs out of memory while training a CNN?**

Here are five things you might do if your GPU runs out of memory while training a CNN:

Decrease the batch size: Decreasing the batch size reduces the amount of memory required for intermediate computations, making it easier for the GPU to fit all the data in memory.

Use smaller input image sizes: Decreasing the size of the input images reduces the memory requirements for activations, allowing the GPU to fit more data in memory.

Use lower precision data types: Reducing the precision of the data used in the network (e.g., float16 instead of float32) can reduce the memory requirements by half.

Prune the network: Removing redundant or unimportant neurons and connections in the network can significantly reduce the memory requirements, allowing the GPU to fit more data in memory.

Train the network on a GPU with more memory: If the above methods are not sufficient, the only solution may be to use a GPU with more memory. This allows the GPU to fit more data in memory, reducing the risk of running out of memory.

These are five ways to mitigate the problem of running out of memory on a GPU while training a CNN. It is important to balance the trade-off between memory requirements and accuracy, as reducing memory requirements may also reduce the accuracy of the network.

**4. Why would you use a max pooling layer instead with a convolutional layer of the same stride?**

Max pooling and convolutional layers with the same stride serve different purposes in a Convolutional Neural Network (CNN).

A convolutional layer uses filters to scan the input and identify features, which are then passed on to the next layer. The filters are trained to identify specific features in the input, and the size of the filters determines the spatial extent of the features being learned.

A max pooling layer, on the other hand, is used to reduce the spatial dimensions of the output from a convolutional layer. It works by taking the maximum value of each group of neurons (defined by the pooling window size and stride) and keeping only the maximum value, effectively reducing the spatial dimensions of the output.

Max pooling has several benefits:

Spatial down-sampling: By reducing the spatial dimensions of the output, max pooling helps to reduce the computational requirements of the network, allowing it to process larger images more efficiently.

Translation invariance: Max pooling helps to make the network invariant to small translations in the input, allowing the network to generalize better to new data.

Over-fitting reduction: By reducing the number of parameters in the network, max pooling helps to reduce the risk of overfitting, allowing the network to generalize better to new data.

Therefore, while a convolutional layer with a stride of two would also reduce the spatial dimensions of the output, it would not provide the other benefits of max pooling, such as translation invariance and over-fitting reduction. Additionally, max pooling provides a non-linear down-sampling of the input, while a convolutional layer with a stride of two is a linear down-sampling operation.

In summary, while a convolutional layer with a stride of two can reduce the spatial dimensions of the output, max pooling is a more appropriate layer for this purpose due to its additional benefits and its non-linear down-sampling operation.

**5. When would a local response normalization layer be useful?**

A local response normalization layer is typically used in Convolutional Neural Networks (CNNs) to normalize the activations within a local neighborhood of neurons. This layer helps to stabilize the training of the network by preventing large activations from dominating the learning process.

Local response normalization is particularly useful in networks that process image data, where the input can vary greatly in scale and contrast. The normalization helps to ensure that features with similar importance are treated equally, regardless of their relative scale or contrast.

This layer is typically used after a convolutional layer, and works by normalizing the activations within a local neighborhood of neurons. The normalization is performed by dividing the activation of each neuron by a scale factor, which is a function of the activations of other neurons in the same local neighborhood.

In summary, a local response normalization layer can be useful in CNNs to stabilize the training process by preventing large activations from dominating the learning process and to ensure that features with similar importance are treated equally, regardless of their relative scale or contrast.

**6. In comparison to LeNet-5, what are the main innovations in AlexNet? What about GoogLeNet and ResNet’s core innovations?**

LeNet-5 was an early pioneer in the field of Convolutional Neural Networks (CNNs), introducing key concepts such as convolutional layers, pooling layers, and backpropagation training.

AlexNet, introduced in 2012, was one of the first networks to demonstrate the practicality of deep learning for large-scale image classification tasks. The main innovations in AlexNet over LeNet-5 are:

Deep architecture: AlexNet has 8 layers, including 5 convolutional layers and 3 fully connected layers, compared to LeNet-5's 7 layers.

Large scale: AlexNet was trained on the ImageNet dataset, which contained over 1 million images, compared to LeNet-5's MNIST and CIFAR datasets, which contained a few thousand images each.

ReLU activation function: AlexNet introduced the Rectified Linear Unit (ReLU) activation function, which significantly improved the training speed and performance of the network compared to the sigmoid activation function used in LeNet-5.

GoogLeNet, introduced in 2014, was one of the first networks to demonstrate the practicality of deep learning for large-scale image classification tasks. The main innovations in GoogLeNet over AlexNet are:

Inception modules: GoogLeNet introduced the Inception module, a building block that combined multiple convolutional and pooling layers into a single module, allowing the network to learn more complex features at multiple scales.

Global average pooling: GoogLeNet introduced global average pooling, a technique that replaced the traditional fully connected layers at the end of the network with a global average pooling layer, reducing the number of parameters in the network and improving its ability to generalize to new data.

ResNet, introduced in 2015, is a deep residual network that aimed to address the problem of vanishing gradients in deep networks. The main innovations in ResNet over GoogLeNet are:

Residual connections: ResNet introduced residual connections, which allow the network to learn a residual mapping from the input to the output, rather than trying to learn a mapping from scratch. This makes it easier for the network to learn identity mappings, and prevents the gradients from vanishing or exploding during training.

Extreme depth: ResNet demonstrated that it is possible to train extremely deep networks (over 150 layers) using residual connections, overcoming the previous limitations of vanishing gradients in deep networks.

In summary, the main innovations of AlexNet over LeNet-5 were the deeper architecture, large-scale training, and ReLU activation function. The main innovations of GoogLeNet over AlexNet were the inception modules and global average pooling. The main innovations of ResNet over GoogLeNet were the residual connections and the demonstration of training extremely deep networks.

**7. On MNIST, build your own CNN and strive to achieve the best possible accuracy.**

Building a CNN for the MNIST dataset requires several steps, including defining the network architecture, loading the data, training the model, and evaluating its performance. Here's an example of building a simple CNN using the Keras library in Python:

# Import necessary libraries

import numpy as np

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras import backend as K

# Load the MNIST data and preprocess it

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1)

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1)

input\_shape = (28, 28, 1)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

# Convert the labels to one-hot encoding

num\_classes = 10

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

# Define the model architecture

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3),

activation='relu',

input\_shape=input\_shape))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

# Compile the model

model.compile(loss=keras.losses.categorical\_crossentropy,

optimizer=keras.optimizers.Adadelta(),

metrics=['accuracy'])

# Train the model

batch\_size = 128

epochs = 12

model.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

verbose=1,

validation\_data=(x\_test, y\_test))

# Evaluate the model on the test data

score = model.evaluate(x\_test, y\_test, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

This model consists of two convolutional layers followed by a max pooling layer and two fully connected layers. The model uses the ReLU activation function for the hidden layers and the softmax activation function for the output layer. The model is trained using the Adadelta optimizer and the categorical cross-entropy loss function.

By tuning the hyperparameters such as the number of convolutional filters

**8. Using Inception v3 to classify broad images. a. Images of different animals can be downloaded. Load them in Python using the matplotlib.image.mpimg.imread() or scipy.misc.imread() functions, for example. Resize and/or crop them to 299 x 299 pixels, and make sure they only have three channels (RGB) and no transparency. The photos used to train the Inception model were preprocessed to have values ranging from -1.0 to 1.0, so make sure yours do as well.**

**9. Large-scale image recognition using transfer learning.**

**a. Make a training set of at least 100 images for each class. You might, for example, identify your own photos based on their position (beach, mountain, area, etc.) or use an existing dataset, such as the flowers dataset or MIT&#39;s places dataset (requires registration, and it is huge).**

**b. Create a preprocessing phase that resizes and crops the image to 299 x 299 pixels while also adding some randomness for data augmentation.**

**c. Using the previously trained Inception v3 model, freeze all layers up to the bottleneck layer (the last layer before output layer) and replace output layer with appropriate number of outputs for your new classification task (e.g., the flowers dataset has five mutually exclusive classes so the output layer must have five neurons and use softmax activation function).**

**d. Separate the data into two sets: a training and a test set. The training set is used to train the model, and the test set is used to evaluate it.**