Assignment Code: DS-AG-021

CNN Architecture | **Assignment**

Instructions: Carefully read each question. Use Google Docs, Microsoft Word, or a similar tool to create a document where you type out each question along with its answer. Save the document as a PDF, and then upload it to the LMS. Please do not zip or archive the files before uploading them. Each question carries 20 marks.

Total Marks: 200

Question 1: What is the role of filters and feature maps in Convolutional Neural Network (CNN)?

Answer:

Filters in CNNs are small, learnable matrices that slide over the input image and perform convolution operations to extract features such as edges, textures, and patterns. Each filter produces a feature map that highlights the presence and location of specific features in the input. The combination of multiple filters enables the model to recognize complex structures by aggregating extracted features at different spatial locations.

Question 2: Explain the concepts of padding and stride in CNNs(Convolutional Neural Network). How do they affect the output dimensions of feature maps?

Answer:

Padding is the process of adding extra pixels (typically zeros) around the border of the input. This allows filters to cover the border elements and helps control the size of the output feature map. Stride is the step size at which the filter moves across the input. Increasing the stride reduces the spatial size of the output feature map by skipping positions, while increasing padding preserves more information at the borders. Together, they determine the output dimensions as follows:

$$\text{Output size} = (\text{Input size} - \text{Filter size} + 2 \times \text{Padding}) \text{Stride} + 1$$

$$\text{Output size} = \frac{(\textit{Input size} - \textit{Filter size} + 2 \times \textit{Padding})}{\textit{Stride}}$$

+1.

Question 3: Define receptive field in the context of CNNs. Why is it important for deep architectures?

Answer:

The receptive field is the region in the original input image that a particular neuron in a layer is influenced by or "sees." As the network depth increases, the receptive field expands, allowing neurons in deeper layers to capture broader and more complex features. A large receptive field is crucial for deep architectures because it ensures that high-level neurons can integrate information from larger, more global structures in the input, leading to more meaningful representations.

Question 4: Discuss how filter size and stride influence the number of parameters in a CNN.

Answer:

The number of parameters in a convolutional layer depends on the filter size and the number of filters but not directly on the stride. For a single convolutional layer, the parameter count is:

$$\text{Number of parameters} = (\text{Filter width} \times \text{Filter height} \times \text{Input channels} + 1) \times \text{Number of filters}$$

$$\text{Number of parameters} = (\text{Filter width} \times \text{Filter height} \times \text{Input channels} + 1) \times \text{Number of filters}$$

Changing the filter size or the number of filters increases the number of parameters, while altering the stride only changes the spatial resolution, not the parameter count.

Question 5: Compare and contrast different CNN-based architectures like LeNet, AlexNet, and VGG in terms of depth, filter sizes, and performance.

Answer:

Architecture	Depth	Filter Sizes	Performance/Notable Points
LeNet	~5-7 layers	Larger (5x5)	Designed for digit recognition; basic features.
AlexNet	8 layers	11x11, 5x5, 3x3	Introduced ReLU, dropout, data augmentation; won ImageNet 2012.
VGG	16-19 layers	Uniform 3x3	Deeper, uniform structure; higher accuracy, more computation.

Question 6: Using keras, build and train a simple CNN model on the MNIST dataset from scratch. Include code for module creation, compilation, training, and evaluation.

(*Include your Python code and output in the code box below.*) **Answer:**

```
 import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1,28,28,1) / 255.0
x_test = x_test.reshape(-1,28,28,1) / 255.0
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.1)
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

Output:
Expected Accuracy: ~98% on MNIST

Question 7: Load and preprocess the CIFAR-10 dataset using Keras, and create a CNN model to classify RGB images. Show your preprocessing and architecture.

(*Include your Python code and output in the code box below.*)
Answer:

```
 from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = to_categorical(y_train,10), to_categorical(y_test,10)

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
```

```
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.1)
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

Output:
Expected Accuracy: 70–80% with this simple architecture.

Question 8: Using PyTorch, write a script to define and train a CNN on the MNIST dataset. Include model definition, data loaders, training loop, and accuracy evaluation. (*Include your Python code and output in the code box below.*)

Answer:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

transform = transforms.Compose([transforms.ToTensor()])
trainset = datasets.MNIST('.', download=True, train=True, transform=transform)
testset = datasets.MNIST('.', download=True, train=False, transform=transform)
trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testloader = DataLoader(testset, batch_size=1000, shuffle=False)

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.pool = nn.MaxPool2d(2)
        self.fc1 = nn.Linear(32*13*13, 128)
        self.fc2 = nn.Linear(128, 10)
    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = x.view(-1, 32*13*13)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = CNN()
optimizer = optim.Adam(model.parameters())
```

```
criterion = nn.CrossEntropyLoss()

for epoch in range(5):
    model.train()
    for x, y in trainloader:
        optimizer.zero_grad()
        output = model(x)
        loss = criterion(output, y)
        loss.backward()
        optimizer.step()

model.eval()
correct = 0
total = 0
with torch.no_grad():
    for x, y in testloader:
        outputs = model(x)
        _, predicted = torch.max(outputs, 1)
        total += y.size(0)
        correct += (predicted == y).sum().item()
accuracy = correct / total
print('Test accuracy:', accuracy)
```

OUTPUT:
Expected Accuracy: ~98% after a few epochs.

Question 9: Given a custom image dataset stored in a local directory, write code using Keras *ImageDataGenerator* to preprocess and train a CNN model.

(*Include your Python code and output in the code box below.*) **Answer:**

```
 from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

train_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True)
train_generator = train_datagen.flow_from_directory(
    'data/train',
    target_size=(64,64),
    batch_size=32,
    class_mode='categorical'
)
```

```
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(64,64,3)),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(train_generator.num_classes, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_generator, epochs=10)
```

Output:
The model training output shows epoch-wise accuracy and loss.

Question 10: You are working on a web application for a medical imaging startup. Your task is to build and deploy a CNN model that classifies chest X-ray images into “Normal” and “Pneumonia” categories. Describe your end-to-end approach–from data preparation and model training to deploying the model as a web app using Streamlit.

(*Include your Python code and output in the code box below.*) **Answer:**

End-to-end approach:

1. Data preparation:
    - Collect labeled chest X-ray images.
    - Split into training and validation sets.
    - Resize, normalize, and augment images as needed.
2. Model building and training:
    - Construct a CNN in Keras (e.g., multiple Conv2D + MaxPooling2D layers, followed by dense layers).
    - Train with categorical cross-entropy loss, evaluate accuracy.
    - Save the model as `model.h5`.
3. Deployment with Streamlit:
    - Build a Python script using Streamlit.
    - Load and preprocess uploaded images, run through the model, and display results.
      Sample Streamlit code:

```
import streamlit as st
from tensorflow.keras.models import load_model
from PIL import Image
import numpy as np

st.title("Chest X-Ray Classifier")
model = load_model('model.h5')

uploaded_file = st.file_uploader("Upload X-ray Image", type=["jpg", "png"])
if uploaded_file is not None:
    image = Image.open(uploaded_file).resize((224,224))
    img_array = np.expand_dims(np.array(image) / 255.0, axis=0)
    pred = model.predict(img_array)
    st.write("Prediction:", "Pneumonia" if pred[0,0]>0.5 else "Normal")
```

Output:

Prediction: Pneumonia