Section type :	Online
Mandatory or Optional :	Mandatory
Number of Questions :	21
Number of Questions to be attempted :	21
Section Marks :	100
Display Number Panel :	Yes
Section Negative Marks :	0
Group All Questions :	No
Enable Mark as Answered Mark for Review and Clear Response :	No
Section Maximum Duration :	0
Section Minimum Duration :	0
Section Time In :	Minutes
Maximum Instruction Time :	0
Sub-Section Number :	1
Sub-Section Id :	640653161311
Question Shuffling Allowed :	No

Question Number : 223 Question Id : 6406531046077 Question Type : MCQ

Correct Marks : 0

Question Label : Multiple Choice Question

THIS IS QUESTION PAPER FOR THE SUBJECT "DEGREE LEVEL : DEEP LEARNING PRACTICE (COMPUTER BASED EXAM)"

ARE YOU SURE YOU HAVE TO WRITE EXAM FOR THIS SUBJECT?
CROSS CHECK YOUR HALL TICKET TO CONFIRM THE SUBJECTS TO BE WRITTEN.

(IF IT IS NOT THE CORRECT SUBJECT, PLS CHECK THE SECTION AT THE TOP FOR THE SUBJECTS REGISTERED BY YOU)

Options :

6406533531407. ✔ YES

6406533531408. ✖ NO

Sub-Section Number :	2
Sub-Section Id :	640653161312
Question Shuffling Allowed :	Yes

Question Number : 224 Question Id : 6406531046078 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

Consider the following code snippets for loading and modifying VGGNet-16 and VGGNet-19 architectures for a classification task with 10 classes:

```
import torch
import torch.nn as nn
from torchvision.models import vgg16, vgg19

# Block A: VGGNet-16 with modified classification head
class VGG16Modified(nn.Module):
    def __init__(self, num_classes=10):
        super(VGG16Modified, self).__init__()
        self.vgg16 = vgg16(pretrained=True)
        self.vgg16.classifier[6] = nn.Linear(4096, num_classes)

    def forward(self, x):
        return self.vgg16(x)

model_a = VGG16Modified()

# Block B: VGGNet-19 with modified classification head
class VGG19Modified(nn.Module):
    def __init__(self, num_classes=10):
        super(VGG19Modified, self).__init__()
        self.vgg19 = vgg19(pretrained=True)
        self.vgg19.classifier[6] = nn.Linear(4096, num_classes)

    def forward(self, x):
        return self.vgg19(x)

model_b = VGG19Modified()

# Block C: Loading VGGNet-16 and freezing feature extraction layers
vgg16_model = vgg16(pretrained=True)
for param in vgg16_model.features.parameters():
    param.requires_grad = False
vgg16_model.classifier[6] = nn.Linear(4096, 10)

# Block D: VGGNet-19 with feature extraction layers unfrozen
vgg19_model = vgg19(pretrained=True)
vgg19_model.classifier[6] = nn.Linear(4096, 10)
```

Which of the following statements are true about the provided blocks?

(a) Block A uses VGGNet-16 and modifies the classification head to support 10 classes.

(b) Block B and Block C both use VGGNet-19 but differ in how feature extraction layers are handled.

(c) Block C freezes the feature extraction layers in VGGNet-16 for transfer learning.

(d) Block D unfreezes feature extraction layers in VGGNet-19, making it trainable end-to-end.

Select the correct options:

Options :

6406533531409. ✖ (a) and (c)

6406533531410. ✖ (b) and (d)

6406533531411. ✔ (a), (c), and (d)

6406533531412. ✖ All of these

Question Number : 225 Question Id : 6406531046079 Question Type : MCQ
Correct Marks : 5

Question Label : Multiple Choice Question

The given below picture is likely to describe which architecture:

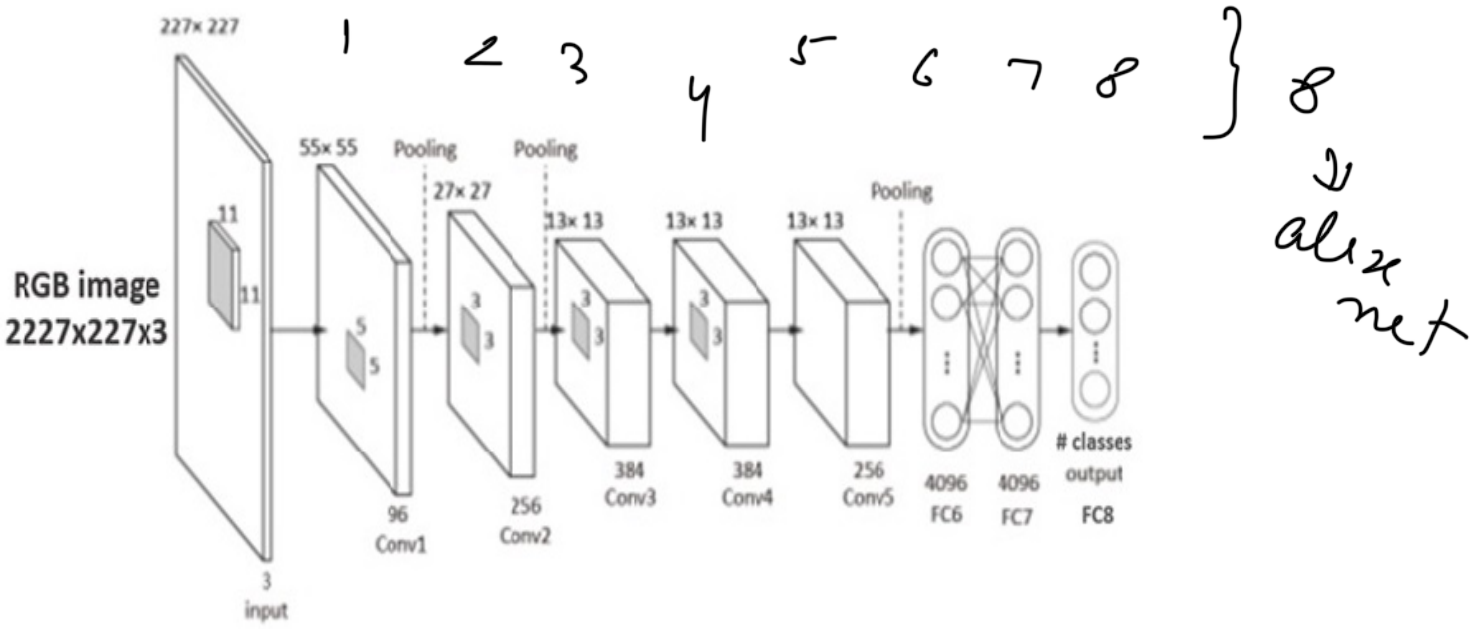


Figure 1: A simple CNN architecture for object detection

Options :

6406533531413. ✖ LeNet

6406533531414. ✖ ResNet

6406533531415. ✔ AlexNet

6406533531416. ✖ VGG16

6406533531417. ✖ None

Question Number : 226 Question Id : 6406531046080 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

Suppose I have an image of size 227×227. Which of the following code snippets correctly implements a Min Pooling operation with a window size of 2 × 2, stride of 2, and padding of 1 in PyTorch?

Options :

6406533531418. ✔

```
import torch
import torch.nn.functional as F

# Min Pooling
def min_pool2d(x, kernel_size, stride, padding):
    return -F.max_pool2d(-x, kernel_size=kernel_size, stride=stride, padding=padding)

input_tensor = torch.randn(1, 3, 227, 227)
output = min_pool2d(input_tensor, kernel_size=2, stride=2, padding=1)
```

6406533531419. ✖

```
import torch
import torch.nn.functional as F

# Min Pooling
def min_pool2d(x, kernel_size, stride, padding):
    return F.max_pool2d(x, kernel_size=kernel_size, stride=stride, padding=padding)

input_tensor = torch.randn(1, 3, 227, 227)
output = min_pool2d(input_tensor, kernel_size=2, stride=2, padding=1)
```

6406533531420. ✖

```
import torch
import torch.nn.functional as F

# Min Pooling
def min_pool2d(x, kernel_size, stride, padding):
    return F.avg_pool2d(x, kernel_size=kernel_size, stride=stride, padding=padding)

input_tensor = torch.randn(1, 3, 227, 227)
output = min_pool2d(input_tensor, kernel_size=2, stride=2, padding=1)
```

6406533531421. ✖

```
import torch
import torch.nn.functional as F

# Min Pooling
def min_pool2d(x, kernel_size, stride, padding):
    return F.max_unpool2d(x, kernel_size=kernel_size, stride=stride, padding=padding)

input_tensor = torch.randn(1, 3, 227, 227)
output = min_pool2d(input_tensor, kernel_size=2, stride=2, padding=1)
```

Question Number : 227 Question Id : 6406531046081 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

Consider a black-and-white image of dimension 227×227. Which of the following correctly demonstrates a function to flatten it along with the dimension of the newly formed vector?

Options :

6406533531422. ✖ `img.reshape(-1) -> (227, 227, 1)`

6406533531423. ✔ `img.flatten() -> (51529,)`

6406533531424. ✖ `img.reshape((227*227,)) -> (227*227*3, 1)`

6406533531425. ✖ `img.flatten((227, 227)) -> (51529,)`

227
×227
1589
454x
454xx
51529

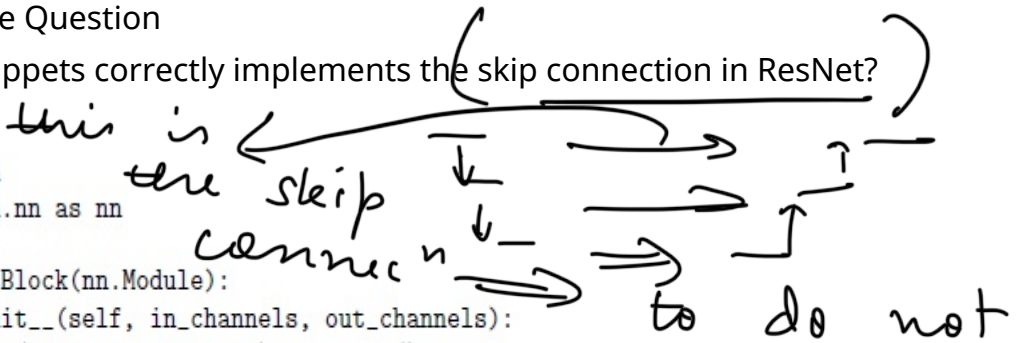
Question Number : 228 Question Id : 6406531046082 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

Which of the following code snippets correctly implements the skip connection in ResNet?

Options :



```
import torch
import torch.nn as nn

class ResNetBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(ResNetBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)

    def forward(self, x):
        residual = x
        x = self.conv1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x += residual
        return self.relu(x)
```

6406533531426. ✔

```
import torch
import torch.nn as nn

class ResNetBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(ResNetBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x += x
        return self.relu(x)
```

6406533531427. ✖

6406533531428. ✖

```
import torch
import torch.nn as nn

class ResNetBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(ResNetBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)

    def forward(self, x):
        residual = self.conv1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x += residual
        return self.relu(x)
```

```
import torch
import torch.nn as nn

class ResNetBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(ResNetBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.conv2(x)
        return self.relu(x)
```

6406533531429. ✖

Question Number : 229 Question Id : 6406531046083 Question Type : MCQ
Correct Marks : 5
Question Label : Multiple Choice Question

Suppose you are implementing a YOLO function with the following classes - Dog, cat, car, house.

Consider this image.

dog = 1
cat = 0
car = 6
house = 3
final = 10

What is the maximum number of bounding box that can be found in this image?

Options :

6406533531430. ✖ 11

6406533531431. ✔ 10

6406533531432. ✖ 12

6406533531433. ✖ 14

Question Number : 230 Question Id : 6406531046084 Question Type : MCQ
Correct Marks : 5

Question Label : Multiple Choice Question

The following code snippets represent different blocks in the Fast R-CNN pipeline. Identify the correct arrangement of these blocks in the Fast R-CNN architecture:

A. **Block A**

```
import torch
import torch.nn as nn

class RegionProposal(nn.Module):
    def __init__(self, in_channels):
        super(RegionProposal, self).__init__()
        self.conv = nn.Conv2d(in_channels, 256, kernel_size=3, stride=1, padding=1)
        self.cls_layer = nn.Conv2d(256, 18, kernel_size=1)
        self.reg_layer = nn.Conv2d(256, 36, kernel_size=1)

    def forward(self, x):
        features = torch.relu(self.conv(x))
        cls_logits = self.cls_layer(features)
        reg_deltas = self.reg_layer(features)
        return cls_logits, reg_deltas
```

B. **Block B**

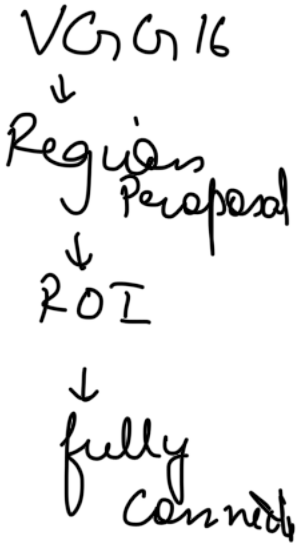
```
import torchvision.models as models

class FeatureExtractor(nn.Module):
    def __init__(self):
        super(FeatureExtractor, self).__init__()
        vgg = models.vgg16(pretrained=True)
        self.features = vgg.features  # Use pre-trained VGG16 convolutional layers

    def forward(self, x):
        return self.features(x)
```

C. **Block C**

```
from torchvision.ops import roi_pool

class ROIPooling(nn.Module):
    def __init__(self, output_size=(7, 7)):
        super(ROIPooling, self).__init__()
        self.output_size = output_size

    def forward(self, feature_map, proposals):
        return roi_pool(feature_map, proposals, output_size=self.output_size)
```

D. **Block D**

```
class FullyConnectedHead(nn.Module):
    def __init__(self, in_features, num_classes):
        super(FullyConnectedHead, self).__init__()
        self.fc1 = nn.Linear(in_features, 4096)
        self.fc2 = nn.Linear(4096, 4096)
        self.cls_score = nn.Linear(4096, num_classes)
        self.bbox_pred = nn.Linear(4096, num_classes * 4)  # 4 coordinates per class

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        cls_logits = self.cls_score(x)
        bbox_deltas = self.bbox_pred(x)
        return cls_logits, bbox_deltas
```



Which of the following is the correct arrangement of these blocks in the Fast R-CNN architecture?

Options :

6406533531434. ✔ B, A, C, D

6406533531435. ✖ A, B, C, D

6406533531436. ✖ B, C, D, A

6406533531437. ✖ C, A, B, D

Question Number : 231 Question Id : 6406531046085 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

Which of the following code snippets correctly models the loss function in YOLO?

Options :

5 loss

6406533531438. ✔

```
coord_loss = lambda_coord * mse_loss(predictions[..., :2], targets[..., :2])
conf_loss = mse_loss(predictions[..., 4], targets[..., 4])
noobj_loss = lambda_noobj * mse_loss(predictions[..., 4], torch.zeros_like(targets[..., 4]))
class_loss = mse_loss(predictions[..., 5:], targets[..., 5:])
total_loss = coord_loss + conf_loss + noobj_loss + class_loss
```

6406533531439. ✖

```
coord_loss = lambda_coord * mse_loss(predictions[..., :4], targets[..., :4])
conf_loss = lambda_noobj * mse_loss(predictions[..., 4], targets[..., 4])
class_loss = mse_loss(predictions[..., 5:], targets[..., 5:])
total_loss = coord_loss + conf_loss + class_loss
```

6406533531440. ✖

```
coord_loss = lambda_coord * mse_loss(predictions[..., :2], targets[..., :2])
conf_loss = mse_loss(predictions[..., 4], targets[..., 4])
class_loss = mse_loss(predictions[..., 5:], targets[..., 5:])
total_loss = coord_loss + conf_loss + class_loss
```

6406533531441. ✖

```
coord_loss = lambda_coord * mse_loss(predictions[..., :2], targets[..., :2])
noobj_loss = mse_loss(predictions[..., 4], torch.zeros_like(targets[..., 4]))
total_loss = coord_loss + noobj_loss
```

$$R = \frac{TP}{TP+FN} = \frac{30}{50} = 0.6$$

Question Number : 232 Question Id : 6406531046086 Question Type : MCQ

Correct Marks : 5

$$P = \frac{TP}{TP+FP} = \frac{30}{40} = \frac{3}{4} = 0.75$$

Question Label : Multiple Choice Question

Consider the following predictions and ground truths for a binary classification problem:
- True Positives (TP): 30 - False Positives (FP): 10 - False Negatives (FN): 20

Which of the following correctly calculates the Precision and F1-Score for this classification task?

Options :

$$F_1 = \frac{2 \times P \times R}{P+R}$$

6406533531442. ✔

```
precision = TP / (TP + FP)  # 30 / (30 + 10) = 0.75
recall = TP / (TP + FN)     # 30 / (30 + 20) = 0.60
f1_score = 2 * (precision * recall) / (precision + recall)
```

$$= \frac{2 \times 0.6 \times 0.75}{1.35} = 0.66$$

6406533531443. ✖

```
precision = TP / (TP + FP)  # 30 / (30 + 10) = 0.60
recall = TP / (TP + FN)     # 30 / (30 + 20) = 0.75
f1_score = 2 * (precision * recall) / (precision + recall)
```

6406533531444. ✖

```
precision = TP / (TP + FP)  # 30 / (30 + 10) = 0.75
recall = TP / (TP + FN)     # 30 / (30 + 20) = 0.60
f1_score = precision + recall
```

6406533531445. ✖

```
precision = TP / (TP + FP)  # 30 / (30 + 10) = 0.60
recall = TP / (TP + FN)     # 30 / (30 + 20) = 0.75
f1_score = (precision * recall) / (precision + recall)
```

Question Number : 233 Question Id : 6406531046087 Question Type : MCQ
Correct Marks : 5
Question Label : Multiple Choice Question
How many convolutional layers are there in the original YOLO architecture?
Options :
6406533531446. ✔ 24
6406533531447. ✖ 19
6406533531448. ✖ 53
6406533531449. ✖ 75

Question Number : 234 Question Id : 6406531046088 Question Type : MCQ
Correct Marks : 5
Question Label : Multiple Choice Question
Consider the following code snippet that calculates the Intersection over Union (IoU) for two bounding boxes:

```
import torch

def calculate_iou(box1, box2):
    # Calculate intersection
    x1 = torch.max(box1[0], box2[0])
    y1 = torch.max(box1[1], box2[1])
    x2 = torch.min(box1[2], box2[2])
    y2 = torch.min(box1[3], box2[3])
    intersection = torch.clamp(x2 - x1, min=0) * torch.clamp(y2 - y1, min=0)

    # Calculate union
    area1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
    area2 = (box2[2] - box2[0]) * (box2[3] - box2[1])
    union = area1 + area2 - intersection

    # Compute IoU
    iou = intersection / union if union > 0 else 0
    return iou

# Bounding boxes: [x1, y1, x2, y2]
box1 = torch.tensor([0, 0, 2, 2])
box2 = torch.tensor([1, 1, 3, 3])

print(calculate_iou(box1, box2))
```

What will the function output for the given bounding boxes?

Options :
6406533531450. ✔ 0.142
6406533531451. ✖ 0.250

6406533531452. ✖ 0.333

6406533531453. ✖ 0.500

Question Number : 235 Question Id : 6406531046089 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

Consider the following code snippets used in a multiscale deep network for single-image depth estimation. Both blocks play different roles in the network.

Which of the following best describes the roles of the two blocks?

A. Block A:

```
import torch
import torch.nn as nn

class DepthNetwork(nn.Module):
    def __init__(self):
        super(DepthNetwork, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
            nn.ReLU(),
            nn.Conv2d(64, 128, kernel_size=5, stride=2, padding=2),
            nn.ReLU()
        )
        self.fc = nn.Sequential(
            nn.Linear(128 * 28 * 28, 1024),
            nn.ReLU(),
            nn.Linear(1024, 128 * 56 * 56)
        )

    def forward(self, x):
        x = self.encoder(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        x = x.view(x.size(0), 128, 56, 56)
        return x
```

B. Block B:

```
import torch
import torch.nn as nn

class DepthRefinementNetwork(nn.Module):
    def __init__(self):
        super(DepthRefinementNetwork, self).__init__()
        self.refinement = nn.Sequential(
            nn.Conv2d(131, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 1, kernel_size=3, padding=1)
        )

    def forward(self, coarse_depth, rgb):
        x = torch.cat((coarse_depth, rgb), dim=1)  # Concatenate coarse depth and RGB
        x = self.refinement(x)
        return x
```

Options :

6406533531454. ✔ Block A generates high-level global features and outputs an initial coarse

depth map, while Block B refines the depth map using local details from the RGB image and the coarse depth map.

6406533531455. ✖ Block A performs refinement of the depth map using concatenated coarse depth and RGB features, while Block B generates the coarse depth map from the input RGB image.

6406533531456. ✖ Both Block A and Block B are coarse networks, with Block B performing an additional refinement step.

6406533531457. ✖ Block A generates a low-resolution coarse depth map but does not include global features, while Block B produces fine-grained depth without refinement.

Question Number : 236 Question Id : 6406531046090 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

Which of the following code snippets correctly implements the **downsampling** operation in a U-Net architecture?

Options :

6406533531458. ✔

```
import torch
import torch.nn as nn

class DownSampleBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DownSampleBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
        self.relu = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.pool(x)
        return x
```

6406533531459. ✖

```
import torch
import torch.nn as nn

class DownSampleBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DownSampleBlock, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        x = self.conv(x)
        x = self.relu(x)
        return x
```

6406533531460.

```
import torch
import torch.nn as nn

class DownSampleBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DownSampleBlock, self).__init__()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        x = self.pool(x)
        return x
```

6406533531461.

```
import torch
import torch.nn as nn

class DownSampleBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DownSampleBlock, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv(x)
        return x
```

Question Number : 237 Question Id : 6406531046091 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

Consider the following predictions and ground truth values:
- Predictions: [3.0,−0.5, 2.0, 7.0] - Ground Truth: [2.5, 0.0, 2.0, 8.0]
The Mean Squared Error (MSE) is calculated using one of the following code snippets. Identify the correct implementation and its result.

Options :

6406533531462.

```
import torch

predictions = torch.tensor([3.0, -0.5, 2.0, 7.0])
targets = torch.tensor([2.5, 0.0, 2.0, 8.0])

mse = ((predictions - targets).abs()).mean()
print(mse.item())
# Output: 0.500
```

6406533531463.

```
import torch

predictions = torch.tensor([3.0, -0.5, 2.0, 7.0])
targets = torch.tensor([2.5, 0.0, 2.0, 8.0])

mse = ((predictions - targets) ** 2).mean()
print(mse.item())
# Output: 0.625
```

6406533531464. ✖

```
import torch

predictions = torch.tensor([3.0, -0.5, 2.0, 7.0])
targets = torch.tensor([2.5, 0.0, 2.0, 8.0])

mse = ((predictions - targets) ** 2).sum() / predictions.size(0)
print(mse.item())
# Output: 0.625
```

6406533531465. ✖

```
import torch

predictions = torch.tensor([3.0, -0.5, 2.0, 7.0])
targets = torch.tensor([2.5, 0.0, 2.0, 8.0])

mse = ((predictions - targets) ** 2).sum()
print(mse.item())
# Output: 2.500
```

Question Number : 238 Question Id : 6406531046092 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

In a stereo vision system, left-right disparity is used to compute depth by comparing corresponding points in the left and right images. Which of the following statements about left right disparity is correct?

Options :

6406533531466. ✖ Disparity is the absolute difference in pixel intensities between the left and right images.

6406533531467. ✖ Disparity increases as the depth of an object increases from the cameras.

6406533531468. ✔ Disparity is the horizontal shift between corresponding points in the left and right images.

6406533531469. ✖ Disparity is the vertical difference between points in the left and right images.

Question Number : 239 Question Id : 6406531046094 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

What are the primary roles of the **generator** and **discriminator** in a Super-Resolution GAN (SRGAN) architecture?

Options :

6406533531475. ✔ The generator produces high-resolution images from low-resolution inputs, while the discriminator distinguishes real high-resolution images from generated high-resolution images.

6406533531476. ✖ The generator distinguishes real high-resolution images from generated images, while the discriminator produces high-resolution images from low-resolution inputs.

6406533531477. ✖ Both the generator and discriminator produce high-resolution images from low-resolution inputs.

6406533531478. ✖ Both the generator and discriminator distinguish real high-resolution images from generated high-resolution images.

Question Number : 240 Question Id : 6406531046095 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

Which of the following code snippets correctly adds gaussian noise to an image?

Options :

6406533531479. ✔

```
import torch

def add_noise(image, noise_std=0.1):
    noise = torch.randn_like(image) * noise_std
    noisy_image = image + noise
    return noisy_image
```

6406533531480. ✖

```
import torch

def add_noise(image, noise_std=0.1):
    noise = torch.randint_like(image, low=-noise_std, high=noise_std)
    noisy_image = image + noise
    return noisy_image
```

6406533531481. ✖

```
import torch

def add_noise(image, noise_std=0.1):
    noise = torch.rand_like(image) * noise_std
    noisy_image = image - noise
    return noisy_image
```

6406533531482. ✖

```
import torch

def add_noise(image, noise_std=0.1):
    noise = torch.ones_like(image) * noise_std
    noisy_image = image * noise
    return noisy_image
```

Question Number : 241 Question Id : 6406531046096 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

What are the benefits of using depthwise and pointwise convolutions (as in depthwise separable convolutions) compared to standard convolutions?

Options :

6406533531483. ✔ They reduce the number of parameters and computational cost, making the model more efficient.

6406533531484. ✖ They improve the accuracy of the model by learning richer spatial features.

6406533531485. ✖ They reduce overfitting by performing regularization during convolution operations.

6406533531486. ✖ They allow convolutions to operate across both spatial dimensions and depth channels simultaneously.

Question Number : 242 Question Id : 6406531046097 Question Type : MCQ

Correct Marks : 5

Question Label : Multiple Choice Question

Which of the following code snippets correctly implements the **generator** for a super-resolution GAN (SRGAN) network?

Options :

6406533531487. ✖

```
import torch
import torch.nn as nn

class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.upsample = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=9, stride=1, padding=4),
            nn.ReLU(),
            nn.Conv2d(64, 256, kernel_size=3, stride=1, padding=1),
            nn.PixelShuffle(upscale_factor=2),
            nn.Conv2d(64, 3, kernel_size=3, stride=1, padding=1)
        )

    def forward(self, x):
        return self.upsample(x)
```

6406533531488. ✖

```python
import torch
import torch.nn as nn

class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 3, kernel_size=3, stride=1, padding=1)
        )

    def forward(self, x):
        return self.model(x)
```

6406533531489. ✖

```python
import torch
import torch.nn as nn

class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=9, stride=1, padding=4),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 3, kernel_size=3, stride=1, padding=1)
        )

    def forward(self, x):
        return self.model(x)
```

6406533531490. ✔

```python
import torch
import torch.nn as nn

class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.upsample = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=9, stride=1, padding=4),
            nn.ReLU(),
            nn.Conv2d(64, 256, kernel_size=3, stride=1, padding=1),
            nn.PixelShuffle(upscale_factor=2),
            nn.Conv2d(64, 3, kernel_size=3, stride=1, padding=1)
        )
        self.refine = nn.Sequential(
            nn.Conv2d(3, 3, kernel_size=3, stride=1, padding=1),
            nn.Tanh()
        )

    def forward(self, x):
        x = self.upsample(x)
        return self.refine(x)
```

Sub-Section Number :	3
Sub-Section Id :	640653161313
Question Shuffling Allowed :	Yes

Question Number : 243 Question Id : 6406531046093 Question Type : MSQ
Correct Marks : 5 Max. Selectable Options : 0
Question Label : Multiple Select Question
Which of the following are common use cases of super-resolution using SRGAN?
Options :
6406533531470. ✔ Enhancing medical imaging scans for improved diagnosis.
6406533531471. ✔ Increasing the resolution of satellite images for geographical analysis.
6406533531472. ✔ Generating high-quality images for video streaming and upscaling.
6406533531473. ✖ Translating text from one language to another in images.
6406533531474. ✔ Improving the resolution of historical or archival images.

SPG

Section Id :	64065375596
Section Number :	11
Section type :	Online
Mandatory or Optional :	Mandatory
Number of Questions :	19
Number of Questions to be attempted :	19
Section Marks :	35
Display Number Panel :	Yes
Section Negative Marks :	0
Group All Questions :	No
Enable Mark as Answered Mark for Review and Clear Response :	No
Section Maximum Duration :	0
Section Minimum Duration :	0
Section Time In :	Minutes
Maximum Instruction Time :	0
Sub-Section Number :	1
Sub-Section Id :	640653161314
Question Shuffling Allowed :	No

Question Number : 244 Question Id : 6406531046098 Question Type : MCQ
Correct Marks : 0