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:

In a Convolutional Neural Network, filters (kernels) are small learnable matrices that slide over the input image to detect important patterns such as edges, corners, or textures. Each filter focuses on identifying one specific type of feature.

The result of applying a filter across the input is called a feature map. A feature map represents the presence and location of that detected feature throughout the image, with higher values indicating stronger detection.

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 and Stride in CNNs (short & exam-ready)

Padding means adding extra pixels (usually zeros) around the border of the input image.
Its role is to control the spatial size of the output and to preserve edge information.

- ***No padding (valid)*** **→ output size decreases**
- ***With padding (same)*** **→ output size stays the same**

Stride is the number of pixels the filter moves at each step while sliding over the input.

- **Small stride (e.g., 1) → larger output feature map**
- **Large stride (e.g., 2) → smaller output feature map (downsampling)**

Effect on output dimensions

Output size=((N−F+2P)/S)+1
Where:

- **N = input size**
- **F = filter size**
- **P = padding**
- **S = stride**

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

Answer:

The receptive field in a CNN is the region of the input image that influences a particular neuron or feature map value. In other words, it defines how much of the original input a neuron can "see."

Importance in deep architectures:

- **As CNNs go deeper, the receptive field increases, allowing neurons to capture larger and more complex patterns.**
- **Small receptive fields detect local features (edges, textures), while larger ones detect global features (objects, shapes).**

- **A larger receptive field helps the network understand context and relationships within the image.**

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

Answer:

Filter size directly affects the number of parameters in a CNN.
Larger filters have more weights, so they increase the parameter count and computational cost.
For a convolution layer:

Parameters=(filter height×filter width×input channels)+bias

per filter.

Stride does not change the number of parameters because it does not affect filter size or weights.
However, stride reduces the spatial size of feature maps, which lowers computation and memory usage.

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

Answer:

LeNet

- **Depth: Shallow (about 5 layers)**
- **Filter sizes: Large filters (e.g., 5×5)**
- **Performance: Designed for simple tasks like handwritten digit recognition (MNIST); low computational cost but limited accuracy on complex images.**

AlexNet

- **Depth: Deeper than LeNet (8 layers)**
- **Filter sizes: Mix of large (11×11, 5×5) and smaller filters**

- **Performance: Significantly improved accuracy on large-scale image datasets (ImageNet); introduced ReLU and dropout, but computationally heavy.**

VGG

- **Depth: Very deep (16–19 layers)**
- **Filter sizes: Uniform small filters (3×3)**
- **Performance: High accuracy due to depth and simplicity of design; very large number of parameters and high memory requirements.**

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 required libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess 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, 10)
y_test = to_categorical(y_test, 10)

# Build CNN model
```

```
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Display model summary
model.summary()

# Train the model
history = model.fit(
    x_train, y_train,
    epochs=5,
    batch_size=64,
    validation_split=0.1
)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print("Test Accuracy:", test_accuracy)

#OUTPUT
Model: "sequential"
┌─────────────────────────────────┬────────────────────────┬───────────────
──────┐
│ Layer (type)                    │ Output Shape           │
Param # │
├─────────────────────────────────┼────────────────────────┼───────────────
──────┤
│ conv2d (Conv2D)                 │ (None, 26, 26, 32)     │
320 │
├─────────────────────────────────┼────────────────────────┼───────────────
──┤
│ max_pooling2d (MaxPooling2D)    │ (None, 13, 13, 32)     │
```

```
0 |
├─────────────────────────────────┼────────────────────────────┼──────────
──┤
| conv2d_1 (Conv2D)               | (None, 11, 11, 64)         |
18,496 |
├─────────────────────────────────┼────────────────────────────┼──────────
──┤
| max_pooling2d_1 (MaxPooling2D)  | (None, 5, 5, 64)           |
0 |
├─────────────────────────────────┼────────────────────────────┼──────────
──┤
| flatten (Flatten)               | (None, 1600)               |
0 |
├─────────────────────────────────┼────────────────────────────┼──────────
──┤
| dense (Dense)                   | (None, 128)                |
204,928 |
├─────────────────────────────────┼────────────────────────────┼──────────
──┤
| dense_1 (Dense)                 | (None, 10)                 |
1,290 |
└─────────────────────────────────┴────────────────────────────┴──────────
──┘
Total params: 225,034 (879.04 KB)

Trainable params: 225,034 (879.04 KB)

Non-trainable params: 0 (0.00 B)
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy:
0.9485 - loss: 0.1692 - val_accuracy: 0.9857 - val_loss: 0.0505
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy:
0.9840 - loss: 0.0509 - val_accuracy: 0.9887 - val_loss: 0.0378
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy:
0.9888 - loss: 0.0360 - val_accuracy: 0.9868 - val_loss: 0.0430
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy:
0.9916 - loss: 0.0262 - val_accuracy: 0.9892 - val_loss: 0.0403
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy:
0.9933 - loss: 0.0208 - val_accuracy: 0.9915 - val_loss: 0.0356
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9922
- loss: 0.0255
Test Accuracy: 0.9922000169754028
```

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:

```
# Import libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Preprocessing
# Normalize pixel values
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# One-hot encode labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(128, (3,3), activation='relu'),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Compile model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Model summary
model.summary()
```

```
# Train model
model.fit(
    x_train, y_train,
    epochs=10,
    batch_size=64,
    validation_split=0.1
)

# Evaluate model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print("Test Accuracy:", test_accuracy)
#OUTPUT
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ——————————————————————————— 47s 0us/step
c:\Users\vaibh\OneDrive\Documents\Python\assignments\.venv\Lib\site-packages\keras\src\datasets\cifar.py:18: VisibleDeprecationWarning: dtype(): align should be passed as Python or NumPy boolean but got `align=0`. Did you mean to pass a tuple to create a subarray type? (Deprecated NumPy 2.4)
  d = cPickle.load(f, encoding="bytes")
Model: "sequential_1"
┌─────────────────────────────────────┬─────────────────────────────┬─────────────────┐
│ Layer (type)                        │ Output Shape                │         Param # │
├─────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_2 (Conv2D)                   │ (None, 30, 30, 32)          │             896 │
├─────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_2 (MaxPooling2D)      │ (None, 15, 15, 32)          │               0 │
├─────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_3 (Conv2D)                   │ (None, 13, 13, 64)          │          18,496 │
├─────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_3 (MaxPooling2D)      │ (None, 6, 6, 64)            │               0 │
├─────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_4 (Conv2D)                   │ (None, 4, 4, 128)           │          73,856 │
├─────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_1 (Flatten)                 │ (None, 2048)                │               0 │
├─────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_2 (Dense)                     │ (None, 128)                 │         262,272 │
├─────────────────────────────────────┼─────────────────────────────┼─────────────────┤
```

```
┌─┼──────────────────────┐
│ dense_3 (Dense)        │ (None, 10)          │       1,290 │
└──────────────────────────────────────────────┴─────────────┘
└─┴──────────────────────┘
 Total params: 356,810 (1.36 MB)
Non-trainable params: 0 (0.00 B)

Epoch 1/10
704/704 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10s 14ms/step - accuracy:
0.4426 - loss: 1.5349 - val_accuracy: 0.5510 - val_loss: 1.2580
Epoch 2/10
704/704 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10s 14ms/step - accuracy:
0.5918 - loss: 1.1563 - val_accuracy: 0.6438 - val_loss: 1.0145
Epoch 3/10
704/704 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10s 14ms/step - accuracy:
0.6542 - loss: 0.9866 - val_accuracy: 0.6758 - val_loss: 0.9528
Epoch 4/10
704/704 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10s 14ms/step - accuracy:
0.6960 - loss: 0.8649 - val_accuracy: 0.6844 - val_loss: 0.9157
Epoch 5/10
704/704 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10s 15ms/step - accuracy:
0.7296 - loss: 0.7763 - val_accuracy: 0.6924 - val_loss: 0.8826
Epoch 6/10
704/704 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12s 17ms/step - accuracy:
0.7558 - loss: 0.6984 - val_accuracy: 0.7370 - val_loss: 0.7662
Epoch 7/10
704/704 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy:
0.7745 - loss: 0.6399 - val_accuracy: 0.7294 - val_loss: 0.8000
Epoch 8/10
704/704 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10s 14ms/step - accuracy:
0.7998 - loss: 0.5746 - val_accuracy: 0.7232 - val_loss: 0.8209
Epoch 9/10
704/704 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy:
0.8170 - loss: 0.5223 - val_accuracy: 0.7298 - val_loss: 0.8351
Epoch 10/10
704/704 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9s 13ms/step - accuracy:
0.8382 - loss: 0.4573 - val_accuracy: 0.7374 - val_loss: 0.8499
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7172
- loss: 0.9039
     Test Accuracy: 0.717199981212616
```

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

# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Data preprocessing
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# Load MNIST dataset
train_dataset = datasets.MNIST(root='./data', train=True,
                               transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False,
                              transform=transform, download=True)

train_loader = DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=64, shuffle=False)

# Define CNN model
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.pool = nn.MaxPool2d(2)
        self.fc1 = nn.Linear(64 * 5 * 5, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Initialize model, loss, optimizer
```

```
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
epochs = 5
for epoch in range(epochs):
    model.train()
    running_loss = 0.0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {running_loss/len(train_loader):.4f}")

# Evaluation
model.eval()
correct = 0
total = 0

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f"Test Accuracy: {accuracy:.2f}%")

#OUTPUT
Epoch [1/5], Loss: 0.3698
Epoch [2/5], Loss: 0.2264
Epoch [3/5], Loss: 0.1723
Epoch [4/5], Loss: 0.1326
Epoch [5/5], Loss: 0.1031
Test Accuracy: 92.19%
```

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:

```
# Import libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Directory paths (example structure)
# dataset/
# ├── train/
# │   ├── class1/
# │   └── class2/
# └── validation/
#     ├── class1/
#     └── class2/

train_dir = "dataset/train"
val_dir = "dataset/validation"

# Image preprocessing and augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(rescale=1./255)

# Load images from directories
```

```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)

validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)

# Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(128, (3,3), activation='relu'),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(train_generator.num_classes, activation='softmax')
])

# Compile model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Model summary
model.summary()

# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)
```

#OUTPUT

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                      │ (None, 126, 126, 32)        │             896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 63, 63, 32)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 61, 61, 64)          │          18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 30, 30, 64)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_2 (Conv2D)                    │ (None, 28, 28, 128)         │          73,856 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten (Flatten)                    │ (None, 100352)              │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense (Dense)                        │ (None, 128)                 │      12,845,184 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 2)                   │             258 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
Total params: 12,938,690 (49.36 MB)

Trainable params: 12,938,690 (49.36 MB)

Non-trainable params: 0 (0.00 B)

Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.5000 -
```

```
loss: 0.6989 - val_accuracy: 0.5000 - val_loss: 5.2251
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step - accuracy: 0.5000 -
loss: 5.2025 - val_accuracy: 0.5000 - val_loss: 0.7361
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 173ms/step - accuracy: 0.5000 -
loss: 0.7190 - val_accuracy: 0.5000 - val_loss: 0.7908
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step - accuracy: 0.5000 -
loss: 0.7758 - val_accuracy: 0.5000 - val_loss: 0.6804
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step - accuracy: 0.5000 -
loss: 0.6877 - val_accuracy: 0.5000 - val_loss: 0.6999
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 139ms/step - accuracy: 0.5000 -
loss: 0.7029 - val_accuracy: 0.5000 - val_loss: 0.6727
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 175ms/step - accuracy: 0.5000 -
loss: 0.6768 - val_accuracy: 0.5000 - val_loss: 0.6508
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step - accuracy: 0.5000 -
loss: 0.6683 - val_accuracy: 0.5000 - val_loss: 0.6342
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 174ms/step - accuracy: 0.5000 -
loss: 0.6562 - val_accuracy: 0.5000 - val_loss: 0.6399
Epoch 10/10
     1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step - accuracy:
     0.5000 - loss: 0.6231 - val_accuracy: 0.6667 - val_loss: 0.5932
```

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:

1 Data Preparation

Dataset structure (standard medical imaging layout):

```
chest_xray/
├── train/
│   ├── NORMAL/
```

```
|    └── PNEUMONIA/
├── val/
|    ├── NORMAL/
|    └── PNEUMONIA/
└── test/
     ├── NORMAL/
     └── PNEUMONIA/
```

Preprocessing strategy

- **Resize images to `224×224`**
- **Normalize pixel values**
- **Use data augmentation to reduce overfitting (critical in medical data)**

---

2 Model Training (Keras CNN)

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models

# Paths
train_dir = "chest_xray/train"
val_dir = "chest_xray/val"

# Data generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=15,
    zoom_range=0.1,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_gen = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode="binary"
)
```

```
val_gen = val_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode="binary"
)

# CNN Model
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu',
input_shape=(224,224,3)),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

# Compile
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Train
history = model.fit(
    train_gen,
    epochs=10,
    validation_data=val_gen
)

# Save model
model.save("pneumonia_cnn.h5")
```

Sample Training Output

```
Epoch 1/10
accuracy: 0.79 - loss: 0.46 - val_accuracy: 0.83
Epoch 10/10
```

```
accuracy: 0.93 - loss: 0.18 - val_accuracy: 0.90
```

---

3 Model Inference Logic

```
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model

model = load_model("pneumonia_cnn.h5")

def predict_xray(img_path):
    img = image.load_img(img_path, target_size=(224,224))
    img = image.img_to_array(img) / 255.0
    img = np.expand_dims(img, axis=0)

    prediction = model.predict(img)[0][0]
    return "Pneumonia" if prediction > 0.5 else "Normal"
```

---

4 Deployment with Streamlit (Web App)

```
import streamlit as st
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np

st.title("🫁 Chest X-ray Pneumonia Detector")

model = load_model("pneumonia_cnn.h5")

uploaded_file = st.file_uploader("Upload Chest X-ray Image",
type=["jpg","png","jpeg"])

if uploaded_file:
    img = image.load_img(uploaded_file, target_size=(224,224))
    st.image(img, caption="Uploaded X-ray", use_column_width=True)

    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)[0][0]

    if prediction > 0.5:
```

```
        st.error("🩺 Prediction: Pneumonia")
    else:
        st.success("✅ Prediction: Normal")
```

5 End-to-End Summary (Startup View)

- Data prep: medical image normalization + augmentation
- Model: CNN with binary classification
- Training: binary cross-entropy + Adam
- Deployment: Streamlit UI for real-time inference
- Use case: fast clinical decision support (not diagnosis)