



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

B.E / SEM VIII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML57
SUBJECT	ADVANCE ARTIFICIAL INTELLIGENCE LAB
COURSE CODE	CSL801
PRACTICAL NO.	
DOP	
DOS	



Output:

1) HMM

```
▼ HMM

[12] !pip install hmmlearn
Collecting hmmlearn
  Downloading hmmlearn-0.3.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (161.1/161.1 kB 2.9 MB/s eta 0:00:00)
Requirement already satisfied: numpy>=1.10 in /usr/local/lib/python3.10/dist-packages (from hmmlearn)
Requirement already satisfied: scikit-learn!=0.22.0,>=0.16 in /usr/local/lib/python3.10/dist-packages (from hmmlearn)
Requirement already satisfied: scipy>=0.19 in /usr/local/lib/python3.10/dist-packages (from hmmlearn)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from hmmlearn)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from hmmlearn)
Installing collected packages: hmmlearn
Successfully installed hmmlearn-0.3.2

[13] import numpy as np
     from hmmlearn import hmm
     import matplotlib.pyplot as plt

     # Define the states (weather conditions)
     states = ["Sunny", "Rainy", "Cloudy"]

     # Define the observation symbols (also weather conditions in this case)
     symbols = ["Sunny", "Rainy", "Cloudy"]

     # Transition probabilities matrix (rows sum to 1)
     transition_probabilities = np.array([
         [0.7, 0.1, 0.2], # Sunny -> Sunny, Rainy, Cloudy
         [0.3, 0.5, 0.2], # Rainy -> Sunny, Rainy, Cloudy
         [0.2, 0.3, 0.5] # Cloudy -> Sunny, Rainy, Cloudy
     ])

     # Emission probabilities matrix (rows sum to 1)
     emission_probabilities = np.array([
         [0.8, 0.1, 0.1], # Sunny -> Sunny, Rainy, Cloudy
         [0.1, 0.8, 0.1], # Rainy -> Sunny, Rainy, Cloudy
         [0.1, 0.1, 0.8] # Cloudy -> Sunny, Rainy, Cloudy
     ])

     # Initialize the HMM model
     model = hmm.CategoricalHMM(n_components=len(states), n_iter=100)

     # Set model parameters
     model.startprob_ = np.array([0.6, 0.3, 0.1]) # Initial state probabilities
     model.transmat_ = transition_probabilities # State transition probabilities
     model.emissionprob_ = emission_probabilities # Emission probabilities

[24] # Generate or input observed sequences (weather conditions)
     observed_sequence = np.array([[2], [0], [2]]) # Example observed sequence: Sunny, Rainy, Cloudy

     # Predict outcomes based on observed sequences
     predicted_states = model.predict(observed_sequence)

[25] # Map predicted state indices to state names
     predicted_weather = [states[state_index] for state_index in predicted_states]
     observed_weather = [symbols[obs[0]] for obs in observed_sequence]

     # Plotting
     plt.figure(figsize=(10, 6))

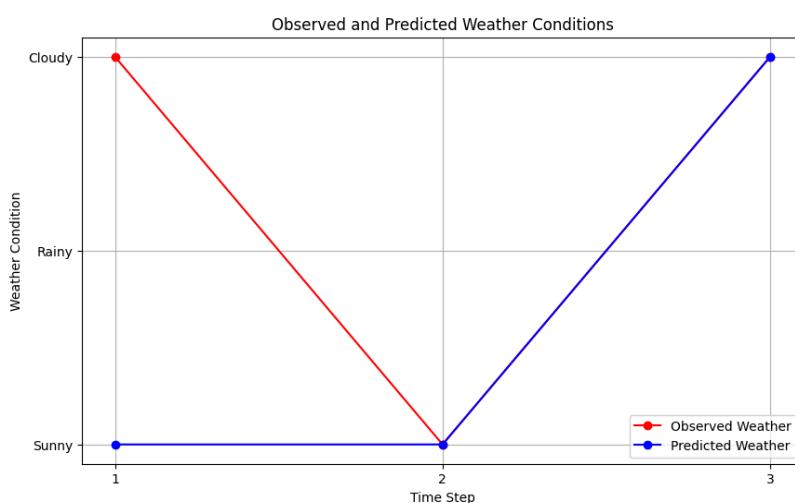
     # Plot observed weather
     plt.plot(range(len(observed_sequence)), observed_sequence, 'ro-', label='Observed Weather')

     # Plot predicted weather
     plt.plot(range(len(predicted_states)), predicted_states, 'bo-', label='Predicted Weather')
```



```
plt.xticks(range(len(observed_sequence)), range(1, len(observed_sequence) + 1))
plt.xlabel('Time Step')
plt.ylabel('Weather Condition')
plt.title('Observed and Predicted Weather Conditions')
plt.legend()
plt.grid(True)
plt.yticks(range(len(states)), states)

plt.show()
```



```
[26] # Predict probabilities of each state for each observation
state_probabilities = model.predict_proba(observed_sequence)

# Print observed and predicted weather with probabilities
print("Observed Weather:")
for obs, prob in zip(observed_weather, observed_sequence):
    print(f"{obs}: {symbols[prob[0]]}")

print("\nPredicted Weather with Probabilities:")
for time_step, (state_probs, pred) in enumerate(zip(state_probabilities, predicted_weather)):
    print(f"Time Step {time_step + 1}:")
    for state, prob in zip(states, state_probs):
        print(f"{state}: {prob:.4f}")
    print(f"Predicted Weather: {pred}")
    print()

Observed Weather:
Cloudy: Cloudy
Sunny: Sunny
Cloudy: Cloudy

Predicted Weather with Probabilities:
Time Step 1:
Sunny: 0.5285
Rainy: 0.1424
Cloudy: 0.3291
Predicted Weather: Sunny

Time Step 2:
Sunny: 0.7771
Rainy: 0.0652
Cloudy: 0.1577
Predicted Weather: Sunny

Time Step 3:
Sunny: 0.2418
Rainy: 0.0565
Cloudy: 0.7017
Predicted Weather: Cloudy
```



2) GMM

```
▼ GMM

[1] import numpy as np
    from sklearn import datasets
    from sklearn.mixture import GaussianMixture
    from sklearn.model_selection import train_test_split, GridSearchCV
    from sklearn.preprocessing import StandardScaler
    from sklearn.metrics import accuracy_score
    import matplotlib.pyplot as plt
    from matplotlib.patches import Ellipse

[2] # Load the Iris dataset
    iris = datasets.load_iris()
    X = iris.data
    y = iris.target

[3] # Feature scaling
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

[4] # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

[5] # Initialize the Gaussian Mixture Model with 3 components (since there are 3 classes in the Iris
    gmm = GaussianMixture(random_state=42)

[6] # Perform grid search for hyperparameter tuning
    param_grid = {
        'n_components': [2, 3, 4],
        'covariance_type': ['full', 'tied', 'diag', 'spherical']
    }

    grid_search = GridSearchCV(gmm, param_grid=param_grid, cv=5)
    grid_search.fit(X_train)

    best_gmm = grid_search.best_estimator_
    print("Best GMM parameters:", best_gmm.get_params())

    Best GMM parameters: {'covariance_type': 'full', 'init_params': 'kmeans', 'max_iter': 100, 'means_init': 'k-means++', 'n_components': 3, 'reg_covar': 1e-05, 'tol': 0.001}

[7] # Fit the best model to the training data using the Expectation-Maximization algorithm
    best_gmm.fit(X_train)

[8] # Predict the labels for the test set
    y_pred = best_gmm.predict(X_test)

[9] # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy}")

    Accuracy: 0.6333333333333333
```

```

2s [10] # Visualization of GMM clusters with ellipses
def plot_gmm(gmm, X):
    n_features = X.shape[1]

    if n_features < 2:
        print("Visualization requires at least 2 features.")
        return

    plt.scatter(X[:, 0], X[:, 1], c=gmm.predict(X), cmap='viridis', s=40, edgecolors='k', marker='o')

    for i in range(gmm.n_components):
        mean = gmm.means_[i, :2]
        cov_matrix = gmm.covariances_[i][:2, :2]
        eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
        angle = np.degrees(np.arctan2(eigenvectors[1, 0], eigenvectors[0, 0]))

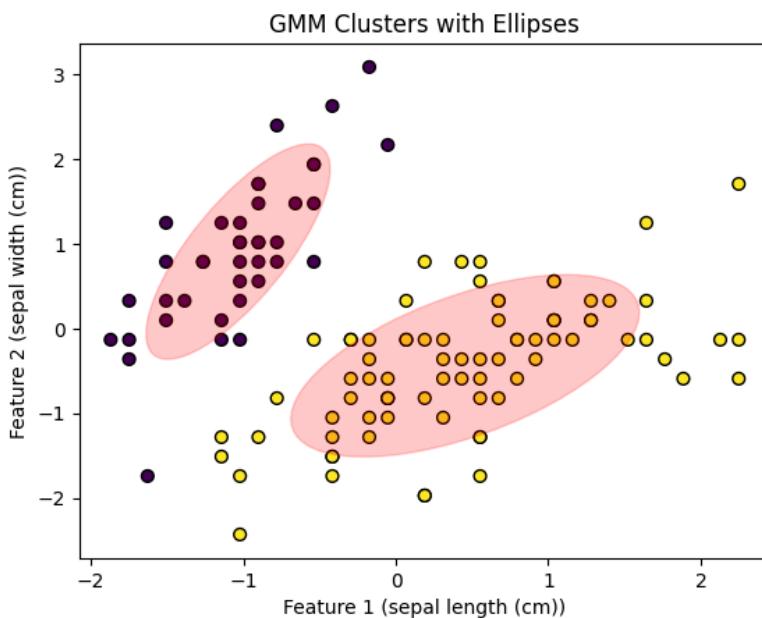
        # Plot the ellipse representing the covariance matrix
        ellipse = Ellipse(mean, 2 * np.sqrt(2 * eigenvalues[0]), 2 * np.sqrt(2 * eigenvalues[1]),
                           angle, color='red', alpha=0.2)
        plt.gca().add_patch(ellipse)

    plt.title('GMM Clusters with Ellipses')
    plt.xlabel(f'Feature 1 ({iris.feature_names[0]})')
    plt.ylabel(f'Feature 2 ({iris.feature_names[1]})')
    plt.show()

# Visualization for all features
plot_gmm(best_gmm, X_train)

```

<ipython-input-10-ac183fc31b9c>:18: MatplotlibDeprecationWarning: Passing the angle parameter of __init__



```

2s [11] # Outlier detection
log_likelihood = best_gmm.score_samples(X_train)
threshold = np.percentile(log_likelihood, 5)
outliers = X_train[log_likelihood < threshold]
print(f"Number of outliers: {len(outliers)}")

```

Number of outliers: 6



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

B.E / SEM VIII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML57
SUBJECT	ADVANCE ARTIFICIAL INTELLIGENCE LAB
COURSE CODE	CSL801
PRACTICAL NO.	
DOP	
DOS	



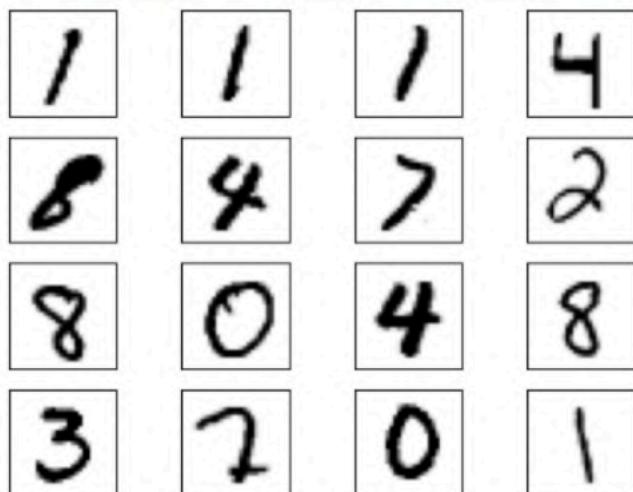
Code & Output:

Loading Dataset

```
train_set = torchvision.datasets.MNIST(  
    root=".", train=True, download=True, transform=transform  
)
```

Dataset Look

```
real_samples, mnist_labels = next(iter(train_loader))  
for i in range(16):  
    ax = plt.subplot(4, 4, i + 1)  
    plt.imshow(real_samples[i].reshape(28, 28), cmap="gray_r")  
    plt.xticks([])  
    plt.yticks([])
```



Discriminator & Generator Model

```
class Discriminator(nn.Module):  
    def __init__(self):  
        super().__init__()  
        self.model = nn.Sequential(  
            nn.Linear(784, 1024),  
            nn.ReLU(),  
            nn.Dropout(0.3),  
            nn.Linear(1024, 512),  
            nn.ReLU(),  
            nn.Dropout(0.3),  
            nn.Linear(512, 256),  
            nn.ReLU(),  
            nn.Dropout(0.3),  
            nn.Linear(256, 1),  
            nn.Sigmoid(),  
        )  
  
    def forward(self, x):  
        x = x.view(x.size(0), 784)  
        output = self.model(x)  
        return output
```



```
discriminator = Discriminator().to(device=device)
```

```
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(100, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, 784),
            nn.Tanh(),
        )

    def forward(self, x):
        output = self.model(x)
        output = output.view(x.size(0), 1, 28, 28)
        return output

generator = Generator().to(device=device)
```

Some Training and Optimization parameters:

```
lr = 0.0001
num_epochs = 50
loss_function = nn.BCELoss()

optimizer_discriminator = torch.optim.Adam(discriminator.parameters(), lr=lr)
optimizer_generator = torch.optim.Adam(generator.parameters(), lr=lr)
```

Training Loop

```
for epoch in range(num_epochs):
    for n, (real_samples, mnist_labels) in enumerate(train_loader):
        # Data for training the discriminator
        real_samples = real_samples.to(device=device)
        real_samples_labels = torch.ones((batch_size, 1)).to(
            device=device
        )
        latent_space_samples = torch.randn((batch_size, 100)).to(
            device=device
        )
        generated_samples = generator(latent_space_samples)
        generated_samples_labels = torch.zeros((batch_size, 1)).to(
            device=device
        )
        all_samples = torch.cat((real_samples, generated_samples))
        all_samples_labels = torch.cat(
            (real_samples_labels, generated_samples_labels)
        )

        # Training the discriminator
        discriminator.zero_grad()
        output_discriminator = discriminator(all_samples)
        loss_discriminator = loss_function(
            output_discriminator, all_samples_labels
        )
        loss_discriminator.backward()
        optimizer_discriminator.step()

        # Data for training the generator
        latent_space_samples = torch.randn((batch_size, 100)).to(
            device=device
        )
```



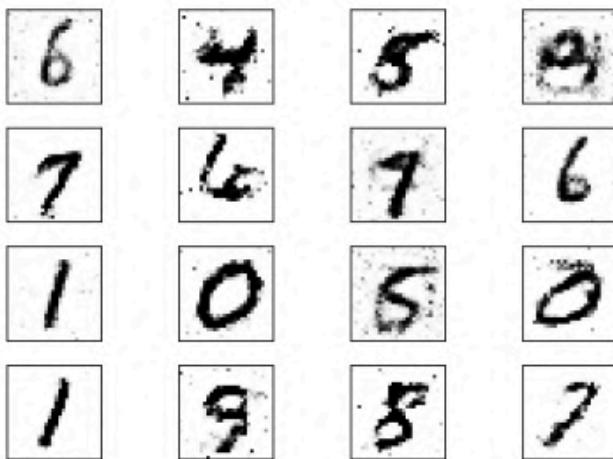
```
# Training the generator
generator.zero_grad()
generated_samples = generator(latent_space_samples)
output_discriminator_generated = discriminator(generated_samples)
loss_generator = loss_function(
    output_discriminator_generated, real_samples_labels
)
loss_generator.backward()
optimizer_generator.step()

# Show loss
if n == batch_size - 1:
    print(f"Epoch: {epoch} Loss D.: {loss_discriminator}")
    print(f"Epoch: {epoch} Loss G.: {loss_generator}")
```

Generated Output by GAN:

```
latent_space_samples = torch.randn(batch_size, 100).to(device)
generated_samples = generator(latent_space_samples)
```

```
generated_samples = generated_samples.cpu().detach()
for i in range(16):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(generated_samples[i].reshape(28, 28), cmap="gray_r")
    plt.xticks([])
    plt.yticks([])
```





**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

B.E / SEM VIII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML57
SUBJECT	ADVANCE ARTIFICIAL INTELLIGENCE LAB
COURSE CODE	CSL801
PRACTICAL NO.	
DOP	
DOS	



Code & Output:

Loading Dataset

```
class FashionMNIST(Dataset):
    def __init__(self, transform=None):
        self.transform = transform
        fashion_df = pd.read_csv('../input/fashion-mnist_train.csv')
        self.labels = fashion_df.label.values
        self.images = fashion_df.iloc[:, 1:].values.astype('uint8').reshape(-1, 28, 28)

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        label = self.labels[idx]
        img = Image.fromarray(self.images[idx])

        if self.transform:
            img = self.transform(img)

        return img, label

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
])
dataset = FashionMNIST(transform=transform)
data_loader = torch.utils.data.DataLoader(dataset, batch_size=64, shuffle=True)
```

Dataset Look

```
dataset = FashionMNIST()
dataset[0][0]
```



Discriminator & Generator Model

```
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()

        self.label_emb = nn.Embedding(10, 10)

        self.model = nn.Sequential(
            nn.Linear(794, 1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )
```



```
def forward(self, x, labels):
    x = x.view(x.size(0), 784)
    c = self.label_emb(labels)
    x = torch.cat([x, c], 1)
    out = self.model(x)
    return out.squeeze()
```

```
class Generator(nn.Module):
    def __init__(self):
        super().__init__()

        self.label_emb = nn.Embedding(10, 10)

        self.model = nn.Sequential(
            nn.Linear(110, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(1024, 784),
            nn.Tanh()
        )

    def forward(self, z, labels):
        z = z.view(z.size(0), 100)
        c = self.label_emb(labels)
        x = torch.cat([z, c], 1)
        out = self.model(x)
        return out.view(x.size(0), 28, 28)
```

```
generator = Generator().cuda()
discriminator = Discriminator().cuda()
```

Some Training and Optimization parameters:

```
criterion = nn.BCELoss()
d_optimizer = torch.optim.Adam(discriminator.parameters(), lr=1e-4)
g_optimizer = torch.optim.Adam(generator.parameters(), lr=1e-4)
```

Training Loop

```
def generator_train_step(batch_size, discriminator, generator, g_optimizer, criterion):
    g_optimizer.zero_grad()
    z = Variable(torch.randn(batch_size, 100)).cuda()
    fake_labels = Variable(torch.LongTensor(np.random.randint(0, 10, batch_size))).cuda()
    fake_images = generator(z, fake_labels)
    validity = discriminator(fake_images, fake_labels)
    g_loss = criterion(validity, Variable(torch.ones(batch_size)).cuda())
    g_loss.backward()
    g_optimizer.step()
    return g_loss.data[0]
```

```
def discriminator_train_step(batch_size, discriminator, generator, d_optimizer, criterion,
                             real_images, labels):
    d_optimizer.zero_grad()
```



```
# train with real images
real_validity = discriminator(real_images, labels)
real_loss = criterion(real_validity, Variable(torch.ones(batch_size)).cuda())

# train with fake images
z = Variable(torch.randn(batch_size, 100)).cuda()
fake_labels = Variable(torch.LongTensor(np.random.randint(0, 10, batch_size))).cuda()
fake_images = generator(z, fake_labels)
fake_validity = discriminator(fake_images, fake_labels)
fake_loss = criterion(fake_validity, Variable(torch.zeros(batch_size)).cuda())

d_loss = real_loss + fake_loss
d_loss.backward()
d_optimizer.step()

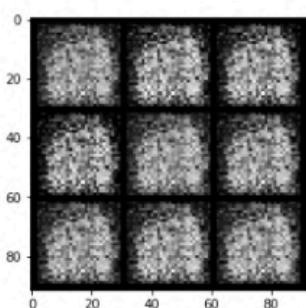
return d_loss.data[0]
```

```
num_epochs = 30
n_critic = 5
display_step = 300
for epoch in range(num_epochs):
    print('Starting epoch {}...'.format(epoch))
    for i, (images, labels) in enumerate(data_loader):
        real_images = Variable(images).cuda()
        labels = Variable(labels).cuda()
        generator.train()
        batch_size = real_images.size(0)
        d_loss = discriminator_train_step(len(real_images), discriminator,
                                         generator, d_optimizer, criterion,
                                         real_images, labels)

        g_loss = generator_train_step(batch_size, discriminator, generator, g_optimizer,
                                      criterion)

        generator.eval()
        print('g_loss: {}, d_loss: {}'.format(g_loss, d_loss))
        z = Variable(torch.randn(9, 100)).cuda()
        labels = Variable(torch.LongTensor(np.arange(9))).cuda()
        sample_images = generator(z, labels).unsqueeze(1).data.cpu()
        grid = make_grid(sample_images, nrow=3, normalize=True).permute(1,2,0).numpy()
        plt.imshow(grid)
        plt.show()
```

g_loss: 4.524019241333008, d_loss: 0.09761635214090347



Starting epoch 1...
g_loss: 3.829733371734619, d_loss: 0.5186285972595215



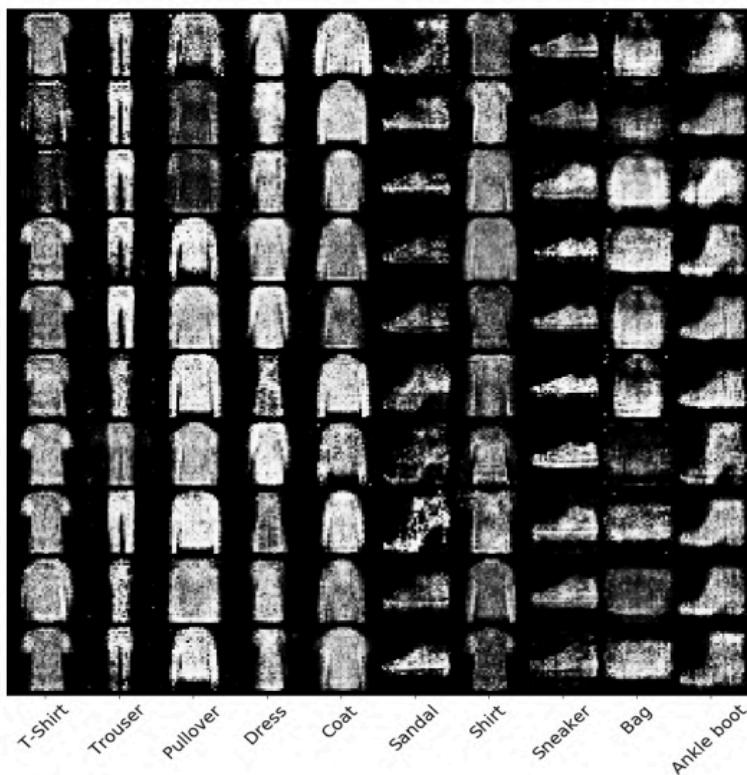


Results

```

z = Variable(torch.randn(100, 100)).cuda()
labels = Variable(torch.LongTensor([i for _ in range(10) for i in range(10)])).cuda()
sample_images = generator(z, labels).unsqueeze(1).data.cpu()
grid = make_grid(sample_images, nrow=10, normalize=True).permute(1,2,0).numpy()
fig, ax = plt.subplots(figsize=(15,15))
ax.imshow(grid)
_ = plt.yticks([])
_ = plt.xticks(np.arange(15, 30), ['T-Shirt', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'], rotation=45, fontsize=20)

```





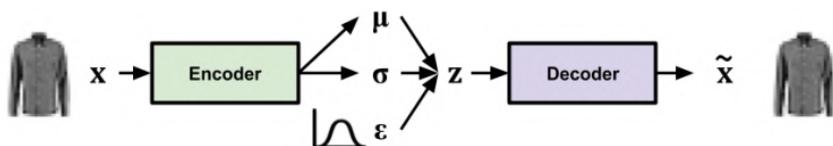
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

B.E / SEM VIII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML57
SUBJECT	ADVANCE ARTIFICIAL INTELLIGENCE LAB
COURSE CODE	CSL801
PRACTICAL NO.	
DOP	
DOS	



Code & Output:



Loading Dataset

- Install packages if in colab

```
[ ] ### install necessary packages if in colab
def run_subprocess_command(cmd):
    process = subprocess.Popen(cmd.split(), stdout=subprocess.PIPE)
    for line in process.stdout:
        print(line.decode().strip())

import sys, subprocess

IN_COLAB = "google.colab" in sys.modules
colab_requirements = [
    "pip install tensorflow-nightly-gpu-2.0-preview==2.0.0.dev20190513",
    "pip install tensorflow-nightly==0.7.0.dev20190508",
]
if IN_COLAB:
    for i in colab_requirements:
        run_subprocess_command(i)
```

- load packages

```
[ ] import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tqdm.autonotebook import tqdm
%matplotlib inline
from IPython import display
import pandas as pd

# the nightly build of tensorflow_probability is required as of the time of writing this
import tensorflow_probability as tfp
ds = tfp.distributions

/mnt/cube/tsainbur/conda_envs/tpy3/lib/python3.6/site-packages/tqdm/autonotebook/_init__.py:14: TqdmE
  (e.g. in jupyter console)", TqdmExperimentalWarning)

[ ] print(tf.__version__, tfp.__version__)
2.0.0-alpha0 0.7.0-dev20190508
```

Create a fashion-MNIST dataset

- Create a fashion-MNIST dataset

```
[ ] TRAIN_BUF=60000
BATCH_SIZE=512
TEST_BUF=10000
DIMS = (28,28,1)
N_TRAIN_BATCHES = int(TRAIN_BUF/BATCH_SIZE)
N_TEST_BATCHES = int(TEST_BUF/BATCH_SIZE)

[ ] # load dataset
(train_images, _), (test_images, _) = tf.keras.datasets.fashion_mnist.load_data()

# split dataset
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype(
    "float32"
) / 255.0
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1).astype("float32") / 255.0
```



```
# batch datasets
train_dataset = (
    tf.data.Dataset.from_tensor_slices(train_images)
    .shuffle(TRAIN_BUF)
    .batch(BATCH_SIZE)
)
test_dataset = (
    tf.data.Dataset.from_tensor_slices(test_images)
    .shuffle(TEST_BUF)
    .batch(BATCH_SIZE)
)
```

Define the network as tf.keras.model object

```
N_Z = 2
encoder = [
    tf.keras.layers.InputLayer(input_shape=DIMS),
    tf.keras.layers.Conv2D(
        filters=32, kernel_size=3, strides=(2, 2), activation="relu"
    ),
    tf.keras.layers.Conv2D(
        filters=64, kernel_size=3, strides=(2, 2), activation="relu"
    ),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=N_Z*2),
]

decoder = [
    tf.keras.layers.Dense(units=7 * 7 * 64, activation="relu"),
    tf.keras.layers.Reshape(target_shape=(7, 7, 64)),
    tf.keras.layers.Conv2DTranspose(
        filters=64, kernel_size=3, strides=(2, 2), padding="SAME", activation="relu"
    ),
    tf.keras.layers.Conv2DTranspose(
        filters=32, kernel_size=3, strides=(2, 2), padding="SAME", activation="relu"
    ),
    tf.keras.layers.Conv2DTranspose(
        filters=1, kernel_size=3, strides=(1, 1), padding="SAME", activation="sigmoid"
    ),
]
```

Create Model

```
# the optimizer for the model
optimizer = tf.keras.optimizers.Adam(1e-3)
# train the model
model = VAE(
    enc = encoder,
    dec = decoder,
    optimizer = optimizer,
)

# exampled data for plotting results
example_data = next(iter(test_dataset))

def plot_reconstruction(model, example_data, nex=8, zm=2):

    example_data_reconstructed = model.reconstruct(example_data)
    samples = model.decode(tf.random.normal(shape=(BATCH_SIZE, N_Z)))
    fig, axs = plt.subplots(ncols=nex, nrows=3, figsize=(zm * nex, zm * 3))
    for axi, (dat, lab) in enumerate(
        zip(
            [example_data, example_data_reconstructed, samples],
            ["data", "data recon", "samples"],
        )
    ):

```



```
        for ex in range(nex):
            axs[axi, ex].matshow(
                dat.numpy()[ex].squeeze(), cmap=plt.cm.Greys, vmin=0, vmax=1
            )
            axs[axi, ex].axes.get_xaxis().set_ticks([])
            axs[axi, ex].axes.get_yaxis().set_ticks([])
            axs[axi, 0].set_ylabel(lab)

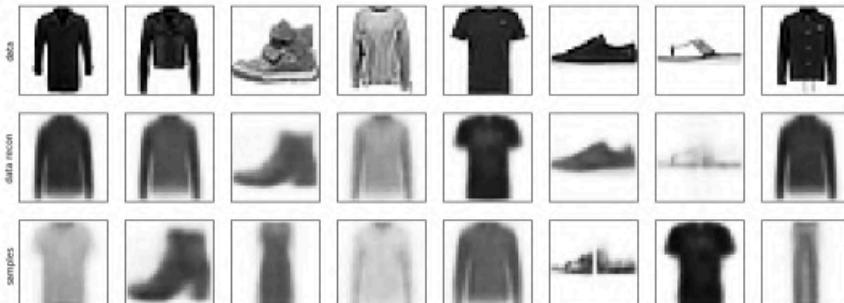
    plt.show()
```

Training Loop

```
# a pandas dataframe to save the loss information to
losses = pd.DataFrame(columns = ['recon_loss', 'latent_loss'])

n_epochs = 50
for epoch in range(n_epochs):
    # train
    for batch, train_x in tqdm(
        zip(range(N_TRAIN_BATCHES), train_dataset), total=N_TRAIN_BATCHES
    ):
        model.train(train_x)
    # test on holdout
    loss = []
    for batch, test_x in tqdm(
        zip(range(N_TEST_BATCHES), test_dataset), total=N_TEST_BATCHES
    ):
        loss.append(model.compute_loss(test_x))
    losses.loc[len(losses)] = np.mean(loss, axis=0)
    # plot results
    display.clear_output()
    print(
        "Epoch: {} | recon_loss: {} | latent_loss: {}".format(
            epoch, losses.recon_loss.values[-1], losses.latent_loss.values[-1]
        )
    )
    plot_reconstruction(model, example_data)
```

Epoch: 49 | recon_loss: 15.782215118408203 | latent_loss: 4.198213577270508

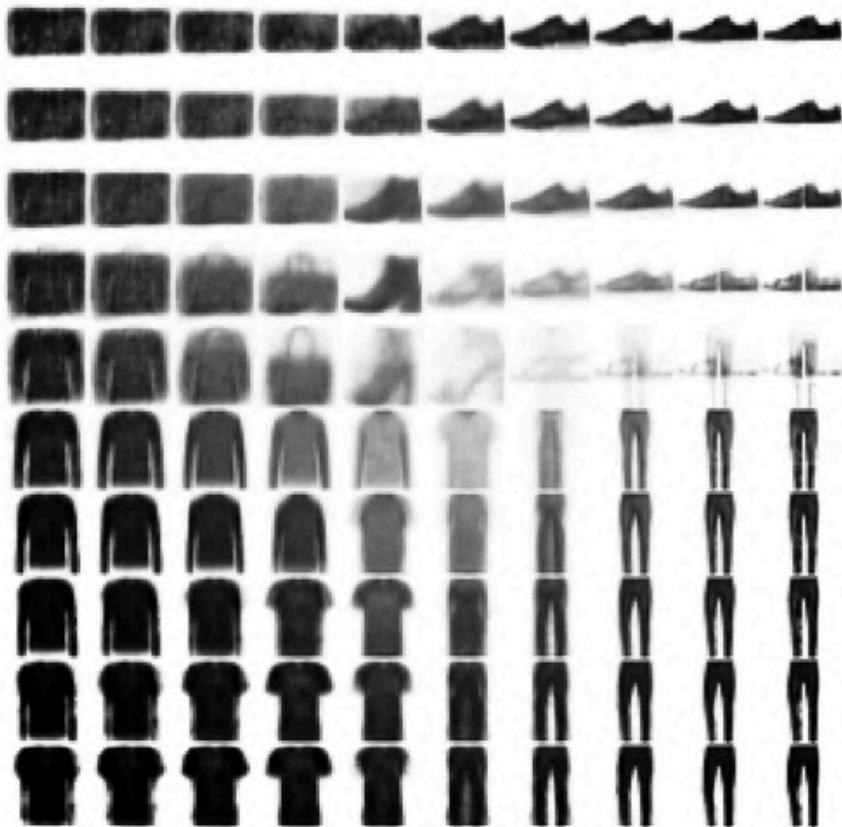


Showing Grid in 2D Latent Space

```
# sample from grid
nx = ny =10
meshgrid = np.meshgrid(np.linspace(-3, 3, nx), np.linspace(-3, 3, ny))
meshgrid = np.array(meshgrid).reshape(2, nx*ny).T
x_grid = model.decode(meshgrid)
x_grid = x_grid.numpy().reshape(nx, ny, 28,28, 1)
# fill canvas
canvas = np.zeros((nx*28, ny*28))
for xi in range(nx):
    for yi in range(ny):
        canvas[xi*28:xi*28+28, yi*28:yi*28+28] = x_grid[xi, yi,:,:,:].squeeze()
fig, ax = plt.subplots(figsize=(10,10))
ax.matshow(canvas, cmap=plt.cm.Greys)
ax.axis('off')
```



(-0.5, 279.5, 279.5, -0.5)





**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

B.E / SEM VIII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML57
SUBJECT	ADVANCE ARTIFICIAL INTELLIGENCE LAB
COURSE CODE	CSL801
PRACTICAL NO.	
DOP	
DOS	



Code & Output:

```
[12] from keras.applications.vgg16 import VGG16
model = VGG16()

[13] from keras.applications.vgg16 import VGG16
model = VGG16()
print(model.summary())

Model: "vgg16"
-----

| Layer (type)               | Output Shape          | Param # |
|----------------------------|-----------------------|---------|
| input_4 (InputLayer)       | [None, 224, 224, 3]   | 0       |
| block1_conv1 (Conv2D)      | (None, 224, 224, 64)  | 1792    |
| block1_conv2 (Conv2D)      | (None, 224, 224, 64)  | 36928   |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64)  | 0       |
| block2_conv1 (Conv2D)      | (None, 112, 112, 128) | 73856   |
| block2_conv2 (Conv2D)      | (None, 112, 112, 128) | 147584  |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128)   | 0       |
| block3_conv1 (Conv2D)      | (None, 56, 56, 256)   | 295168  |
| block3_conv2 (Conv2D)      | (None, 56, 56, 256)   | 590080  |
| block3_conv3 (Conv2D)      | (None, 56, 56, 256)   | 590080  |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256)   | 0       |
| block4_conv1 (Conv2D)      | (None, 28, 28, 512)   | 1180160 |
| block4_conv2 (Conv2D)      | (None, 28, 28, 512)   | 2359808 |
| block4_conv3 (Conv2D)      | (None, 28, 28, 512)   | 2359808 |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512)   | 0       |
| block5_conv1 (Conv2D)      | (None, 14, 14, 512)   | 2359808 |
| block5_conv2 (Conv2D)      | (None, 14, 14, 512)   | 2359808 |


-----  

[13] block5_conv3 (Conv2D)      (None, 14, 14, 512)      2359808
     block5_pool (MaxPooling2D)  (None, 7, 7, 512)       0
     flatten (Flatten)         (None, 25088)           0
     fc1 (Dense)              (None, 4096)            102764544
     fc2 (Dense)              (None, 4096)            16781312
     predictions (Dense)      (None, 1000)             4097000
-----  

Total params: 138357544 (527.79 MB)
Trainable params: 138357544 (527.79 MB)
Non-trainable params: 0 (0.00 Byte)
-----  

None
-----  

[14] from keras.preprocessing.image import load_img
# load an image from file
image = load_img('/content/aai-exp5.jpg.png', target_size=(224, 224))
```

Input Image-



```
[15] from keras.preprocessing.image import img_to_array
    # convert the image pixels to a numpy array
    image = img_to_array(image)

[16] # reshape data for the model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

```
[17] from keras.applications.vgg16 import preprocess_input
    # prepare the image for the VGG model
    image = preprocess_input(image)

[18] # predict the probability across all output classes
    yhat = model.predict(image)

1/1 [=====] - 1s 777ms/step

[19] from keras.applications.vgg16 import decode_predictions
    # convert the probabilities to class labels
    label = decode_predictions(yhat)
    # retrieve the most likely result, e.g. highest probability
    label = label[0][0]
    # print the classification
    print('%s (%.2f%%)' % (label[1], label[2]*100))

coffee_mug (74.83%)
```



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

B.E / SEM VIII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML57
SUBJECT	ADVANCE ARTIFICIAL INTELLIGENCE LAB
COURSE CODE	CSL801
PRACTICAL NO.	
DOP	
DOS	



Code & Output:

```
✓ 5s [2] !pip install xgboost

Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.11.4)

✓ 1s [3] import xgboost as xgb
import pandas as pd

# Load the data
data = pd.read_csv('/content/WineQT.csv')

# Display the first few rows of the data
data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	Id
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	1
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	2
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	3
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	4

Next steps: [View recommended plots](#)

```
✓ 0s [4] from sklearn.model_selection import train_test_split

# Separate target variable
X = data.drop('quality', axis=1)
y = data['quality'].map(lambda x: 1 if x >= 7 else 0)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
✓ 0s [5] from xgboost import XGBClassifier

# Create an instance of the XGBClassifier
model = XGBClassifier(objective='binary:logistic')

# Fit the model to the training data
model.fit(X_train, y_train)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```



```
✓ 0s [6] # Make class predictions  
y_pred = model.predict(X_test)  
y_pred_proba = model.predict_proba(X_test)  
  
accuracy = model.score(X_test, y_test)  
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

Accuracy: 93.01%

```
✓ 0s [7] from sklearn.metrics import log_loss, roc_auc_score  
  
# Calculate log loss  
log_loss(y_test, y_pred_proba)  
  
# Calculate ROC AUC  
roc_auc_score(y_test, y_pred_proba[:,1])
```

0.9385216773276475

```
✓ 0s [8] from sklearn.metrics import classification_report  
  
# Print classification report  
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.95	0.97	0.96	201
1	0.75	0.64	0.69	28
accuracy			0.93	229
macro avg	0.85	0.81	0.83	229
weighted avg	0.93	0.93	0.93	229



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

B.E / SEM VIII / REV 2019 'C SCHEME' / CSE-(AI&ML)
Academic Year: 2023-24

NAME	SINGH SUDHAM DHARMENDRA
BRANCH	CSE-(AI&ML)
ROLL NO.	AIML57
SUBJECT	ADVANCE ARTIFICIAL INTELLIGENCE LAB
COURSE CODE	CSL801
PRACTICAL NO.	
DOP	
DOS	

EXPERIMENT NO. 07

Aim: To study the emerging technologies in AI like Metaverse, Augmented reality etc.

Theory:

i) Introduction:

Emerging technologies in Artificial Intelligence (AI) are driving significant advancements and reshaping the landscape of industries and societies worldwide. Among these, the convergence of AI with emerging technologies like the Metaverse, Augmented Reality (AR), and Quantum Computing stands out as a transformative force with profound implications for the future.

- **Advantages:**

1. Emerging technologies in AI offer unprecedented opportunities for innovation and problem-solving.
2. They have the potential to revolutionize various industries and improve efficiency, productivity, and user experiences.

- **Disadvantages:**

1. Rapid advancements may outpace ethical, legal, and societal considerations.
2. Adoption barriers due to high costs, technical complexities, and concerns about privacy and security.

- **Working:**

The convergence of AI with technologies like Metaverse, AR, and Quantum Computing involves utilizing advanced algorithms, data processing techniques, and computing infrastructure to create immersive, intelligent, and secure environments.

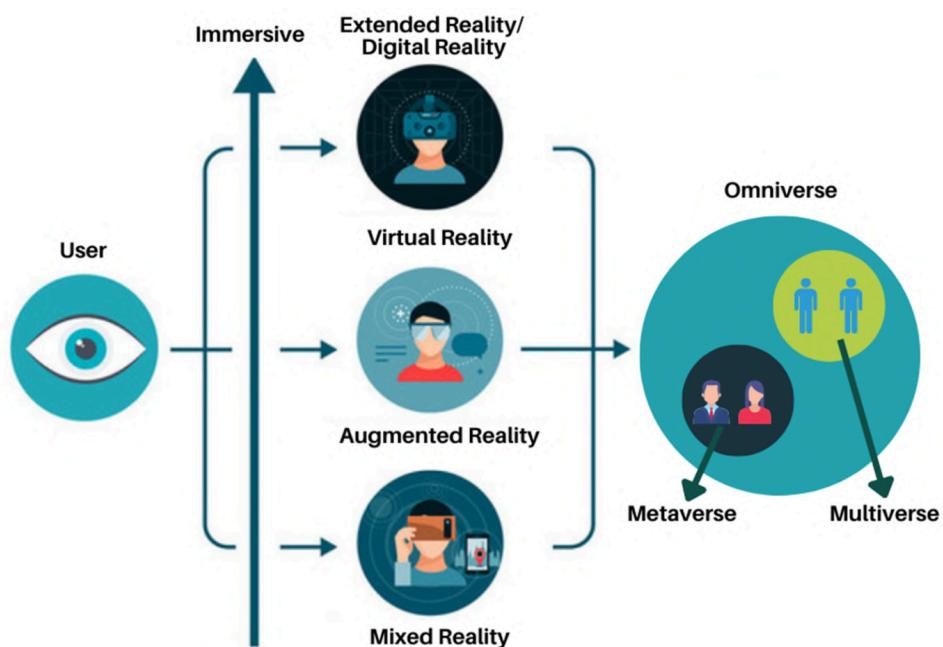


Fig. Metaverse and Multiverse



ii) Metaverse: Redefining Virtual Reality-

The Metaverse can be defined as a collective virtual space comprised of interconnected digital worlds, environments, and assets, where users can interact, socialize, collaborate, and engage in various activities. It represents a convergence of virtual reality, augmented reality, and other immersive technologies, enabling persistent, immersive, and interactive experiences that transcend the limitations of traditional virtual environments.

- **Advantages:**

The Metaverse represents a paradigm shift in virtual reality, offering users immersive experiences beyond the confines of traditional VR environments. Advantages include:

1. Immersive Experiences: The Metaverse provides immersive, interactive, and persistent virtual environments where users can socialize, collaborate, and engage in various activities.
2. Social Interaction: It fosters social connections and community building by enabling real-time communication, avatar interactions, and shared experiences across geographical boundaries.
3. Economic Opportunities: The Metaverse presents economic opportunities through virtual economies, digital marketplaces, and monetization avenues for content creators, developers, and businesses.

- **Disadvantages:**

Despite its potential, the Metaverse also faces several challenges and disadvantages:

1. Addiction and Detachment: There are concerns about users becoming overly immersed in virtual environments, leading to addiction and detachment from reality.
2. Privacy and Security: Issues related to privacy, data security, and digital ownership arise due to the collection and storage of user data, virtual asset ownership, and potential surveillance within virtual spaces.
3. Technological Barriers: Access to the Metaverse may be limited by technological barriers such as hardware requirements, internet connectivity, and affordability.
4. Ethical Considerations: Ethical dilemmas arise concerning the moderation of virtual communities, content censorship, digital identity, and representation within the Metaverse.

- **Working:**

The Metaverse operates as a collective virtual space composed of interconnected virtual worlds, environments, and assets. Its functioning involves:

1. Virtual Worlds: Users access the Metaverse through virtual worlds hosted on various platforms and environments, ranging from social hubs and gaming realms to educational simulations and virtual marketplaces.
2. Avatar Interaction: Users navigate and interact with the Metaverse through customizable avatars, representing their digital personas within virtual environments. Avatars enable social interactions, expression, and identity within the Metaverse.
3. Real-time Communication: Communication tools facilitate real-time interactions between users, enabling voice chat, text messaging, and gestures to enhance socialization and collaboration.
4. Economic Ecosystem: Virtual economies emerge within the Metaverse, driven by the exchange of virtual goods, services, and currencies. Monetization strategies include virtual asset sales, in-world advertising, and virtual events.



iii) Augmented Reality: Enhancing Real-World Experiences-

Augmented Reality (AR) is a technology that superimposes digital information, such as images, videos, or 3D models, onto the real-world environment, thereby enhancing a user's perception and interaction with their surroundings. Unlike virtual reality, which creates entirely immersive digital environments, AR overlays digital content onto the physical world, allowing users to interact with both real and virtual elements simultaneously.

- **Advantages:**

Augmented Reality (AR) enriches real-world environments by overlaying digital information, objects, or experiences onto the physical world. Advantages include:

1. Enhanced Experiences: AR enhances real-world experiences by providing contextual information, guidance, or entertainment overlays that complement users' surroundings.
2. Hands-Free Interaction: AR enables hands-free interaction through wearable devices like smart glasses or mobile applications, allowing users to access information and perform tasks without disrupting their activities.
3. Personalized Content: AR delivers personalized content and experiences based on users' preferences, location, and context, creating tailored and relevant interactions.
4. Improved Learning: AR enhances education and training by providing interactive and immersive learning experiences, simulations, and visualizations that facilitate comprehension and retention.

- **Disadvantages:**

Despite its potential benefits, AR also presents several challenges and disadvantages:

1. Limited Field of View: AR devices may have limited field of view, obstructing users' vision or limiting the amount of digital content that can be overlaid onto the physical environment.
2. Hardware Constraints: Adoption of AR technology is hindered by hardware constraints such as device cost, battery life, processing power, and form factor.
3. Privacy Concerns: AR raises privacy concerns related to data collection, tracking, and surveillance, as it may gather information about users' locations, behaviors, and interactions in the real world.
4. Social Acceptance: Social acceptance and cultural norms may influence the adoption of AR, as some users may feel uncomfortable or self-conscious wearing AR devices in public spaces.

- **Working:**

Augmented Reality works by overlaying digital content onto the physical world using computer vision, sensor fusion, and real-time rendering techniques. The process involves:

1. Environment Recognition: AR systems use computer vision algorithms to recognize and understand the user's environment, including objects, surfaces, and spatial features.
2. Tracking and Registration: AR devices track the user's position and orientation in real-time, aligning digital content with the physical world to create seamless and accurate overlays.
3. Content Rendering: Digital content, such as text, images, videos, or 3D models, is rendered and overlaid onto the user's view through AR-enabled devices like smartphones, smart glasses, or headsets.



4. Interaction and User Interface: Users interact with AR content through gestures, voice commands, or touch interfaces, enabling them to manipulate, explore, or interact with virtual objects overlaid onto their physical surroundings.
5. Contextual Information: AR provides contextual information and guidance relevant to the user's location, interests, or activities, enhancing their understanding and engagement with the real world.

iv) Quantum Computing: Cutting-edge computing-

Quantum Computing is a cutting-edge computing paradigm that harnesses the principles of quantum mechanics to perform computations using quantum bits, or qubits, which can represent and manipulate multiple states simultaneously. Unlike classical computers, which use binary bits to represent information as either 0 or 1, qubits exploit phenomena such as superposition and entanglement to perform computations in parallel, offering the potential for exponential speedup over classical algorithms for certain tasks.

- **Advantages:**

Quantum Computing leverages the principles of quantum mechanics to perform computations at unprecedented speeds and tackle complex problems that are beyond the capabilities of classical computers. Advantages include:

1. Exponential Speedup: Quantum computers can provide exponential speedup over classical computers for certain computational tasks, enabling the rapid execution of complex algorithms and computations.
2. Parallel Processing: Quantum computers harness the power of quantum superposition and entanglement to perform parallel computations on a massive scale, solving problems more efficiently than classical computers.
3. Optimization: Quantum algorithms excel at solving optimization problems, such as finding the best solution among a large set of possibilities, which has applications in machine learning, data analysis, and logistics optimization.
4. Cryptography: Quantum computing offers the potential to break existing cryptographic schemes while also providing opportunities for developing quantum-resistant cryptographic protocols that ensure secure communication and data privacy.

- **Disadvantages:**

Despite its promising potential, Quantum Computing faces several challenges and limitations:

1. Hardware Constraints: Current quantum computers are limited by factors such as the number of qubits, coherence times, and error rates, making them prone to errors and noise that affect the reliability and scalability of quantum algorithms.
2. Technical Complexity: Quantum computing involves complex hardware and software components, as well as specialized knowledge of quantum mechanics and quantum algorithms, which pose barriers to adoption and development.
3. Interoperability: Integrating quantum computers with classical computing infrastructure and algorithms presents interoperability challenges, requiring hybrid approaches and specialized interfaces to harness the combined power of quantum and classical systems.



4. Resource Intensive: Quantum computations often require significant resources, including specialized hardware, cooling systems, and error correction techniques, which can be costly and resource-intensive to maintain and scale.

- **Working:**

Quantum Computing operates based on the principles of quantum mechanics, which differ fundamentally from classical computing. Key aspects of how quantum computing works include:

1. Qubits: Quantum bits, or qubits, are the basic units of information in quantum computing. Unlike classical bits, which can represent either 0 or 1, qubits can exist in superposition, representing both 0 and 1 simultaneously. This property allows quantum computers to perform multiple computations in parallel.
2. Quantum Gates: Quantum algorithms manipulate qubits using quantum gates, which are analogous to classical logic gates. Quantum gates perform operations such as quantum entanglement, superposition, and phase shifts to process information and execute quantum algorithms.
3. Quantum Algorithms: Quantum algorithms are algorithms designed to run on quantum computers and exploit their unique properties to solve specific computational problems more efficiently than classical algorithms. Examples include Shor's algorithm for integer factorization and Grover's algorithm for database search.
4. Quantum Supremacy: Quantum supremacy refers to the milestone achieved when a quantum computer outperforms the most powerful classical computers in performing a specific computational task. This milestone demonstrates the potential of quantum computing to revolutionize various fields, including AI, cryptography, and scientific research.

Conclusion:

1. Summary: The convergence of emerging technologies in AI offers transformative potential across various domains but requires careful consideration of ethical, technical, and societal implications.
2. Importance: Continued research, collaboration, and responsible innovation are essential to harnessing the full benefits of these technologies while mitigating risks.
3. Final thoughts: The future of AI lies in its convergence with other disruptive technologies, paving the way for new paradigms in human-computer interaction, problem-solving, and knowledge discovery.

Image Classification using MobileNet

Submitted in partial fulfillment of the requirements of the degree
BACHELOR OF ENGINEERING
in
Computer Science and Engineering
(Artificial Intelligence & Machine Learning)

Sem - VIII

by

AIML03_Ansari Mohd Hanzala Mohd Imamuddin

AIML11_Chikankar Prathamesh Shivaji

AIML47_Rupareliya Akshar Jayanti

AIML57_Singh Sudham Dharmendra



**Lokmanya Tilak College of Engineering
Koparkhairne, Navi Mumbai - 400 709
University of Mumbai
(AY 2023-24)**

Contents

1.	Introduction	1
2.	Methodology	3
3.	Implementation	5
4.	Experimental Setup	6
5.	Results	7
6.	Discussions	9
7.	Conclusion & References	10

1. Introduction

1.1 Introduction

Image classification is a fundamental task in computer vision that involves categorizing images into predefined classes or categories based on their visual content. This project focuses on implementing an image classification system using deep learning techniques, specifically MobileNet architecture, to accurately classify images into different classes.

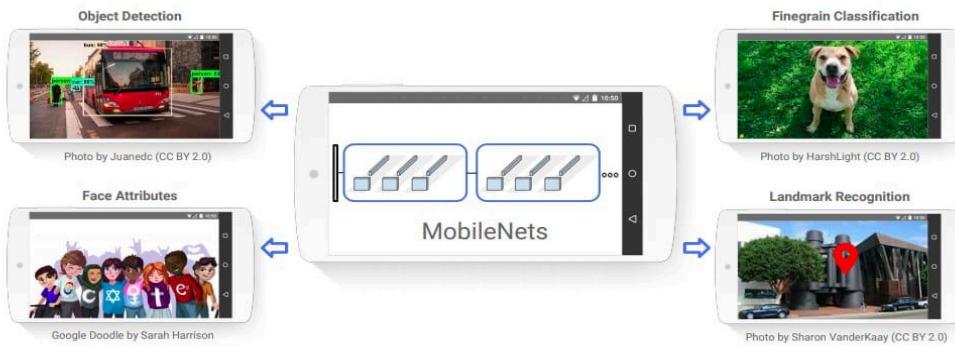


Fig.1.1: Image Classification using MobileNet

1.2 Motivation

Image classification plays a crucial role in various real-world applications, including:

- **Object Recognition:** Image classification enables machines to identify and classify objects within images, facilitating applications such as visual search engines, autonomous robotics, and augmented reality.
- **Medical Imaging:** In the field of healthcare, image classification helps in diagnosing diseases, identifying abnormalities in medical scans (e.g., X-rays, MRIs), and assisting medical professionals in making informed decisions.

1.3 Problem Statement

The specific task of image classification addressed in this project is to classify images into predefined categories or classes. Factors contributing to this challenge include:

- **Variability in Image Content:** Images can vary significantly in terms of lighting conditions, perspectives, backgrounds, and object orientations, making it challenging to extract discriminative features for accurate classification.
- **Class Imbalance:** Imbalanced datasets, where certain classes have significantly fewer samples than others, can lead to biased model predictions and lower performance on minority classes.

- Overfitting: Deep learning models may overfit to the training data, resulting in poor generalization performance on unseen data. Regularization techniques and data augmentation strategies are often employed to mitigate overfitting.
- Computational Complexity: Training deep learning models for image classification tasks often requires substantial computational resources, including high-performance GPUs and large-scale datasets, which may pose challenges for resource-constrained environments.

Addressing these challenges effectively is crucial for developing a robust and reliable image classification system that can be deployed in real-world scenarios.

1.4 Survey of Existing System:

Sr. No	Authors	Title of the paper & year of publ. (Old to recent)	Major contributions
1.	Hang Zhang, Chongruo Wu, Zhongyue Zhang	ResNeSt: Split-Attention Networks (2020).	This Research introduces ResNeSt: Split-attention networks for improved performance.
2.	Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn	Vision Transformers (2021)	An Overview of Vision Transformers: Transformer-based models for vision tasks.
3.	Hugo Touvron, Piotr Bojanowski, Mathilde Caron	CaiT: Conformer Augmented with image Transformers (2021).	This Research provides CaiT: Conformer augmented with image Transformers.

Fig.1.2: Literature survey

2. Methodology

2.1 Model Selection:

MobileNet Overview:

MobileNet is a lightweight convolutional neural network (CNN) architecture designed for efficient mobile and embedded vision applications. It was developed by Google to address the need for models that can perform well on resource-constrained devices while maintaining high accuracy in image classification tasks.

Advantages of MobileNet:

- **Lightweight Design:** MobileNet employs depth wise separable convolutions, which significantly reduce the number of parameters and computations compared to traditional CNN architectures like VGG or ResNet.
- **Efficiency:** MobileNet achieves a good balance between model size, computational efficiency, and accuracy.
- **Task Compatibility:** MobileNet has been pretrained on large-scale datasets such as ImageNet, making it capable of recognizing a wide range of visual concepts.

Justification:

The selection of MobileNet for this image classification project is justified based on the following factors:

- **Model Performance:** MobileNet has demonstrated strong performance on image classification benchmarks such as ImageNet, achieving comparable accuracy to larger and more computationally intensive models.
- **Computational Requirements:** MobileNet's lightweight design and efficiency make it suitable for deployment on resource-constrained platforms, including mobile devices and embedded systems.

Model Architecture:

MobileNet consists of a series of depthwise separable convolutional layers followed by pointwise convolutional layers and global average pooling. The depthwise separable convolutional layers decompose the standard convolution into separate depthwise and pointwise convolutions, reducing both the number of parameters and the computational cost. The final output is obtained by passing the features through a softmax layer for classification.

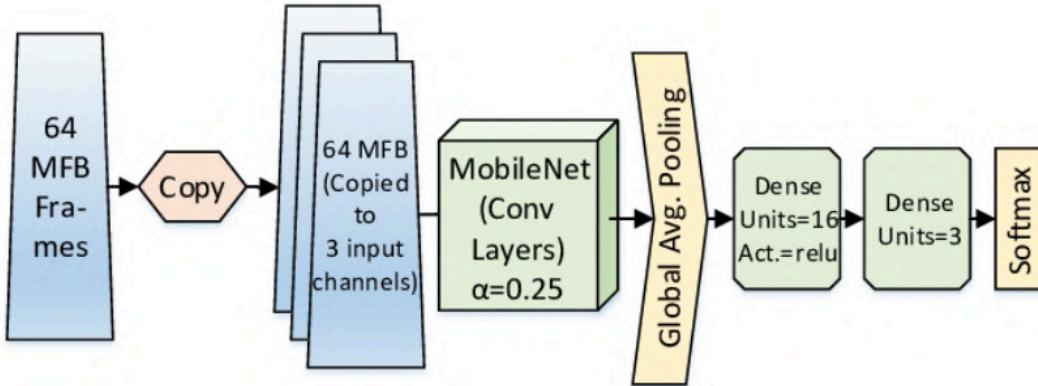


Fig.2: MobileNet Architecture

Overall, MobileNet's architecture enables efficient feature extraction and classification while maintaining high accuracy, making it a suitable choice for image classification tasks in resource-constrained environments.

2.2 Data Preprocessing:

Image Preprocessing Steps:

Before feeding input images into the MobileNet model, several preprocessing steps are applied to enhance quality and variability of input data.

1. Resizing:

The input images are resized to a fixed size suitable for the MobileNet model input. Typically, MobileNet accepts input images of size 224x224 pixels.

2. Normalization:

Normalization is applied to the pixel values of the input images to standardize their intensity levels.

3. Data Augmentation:

Data augmentation techniques are employed to increase the variability of the training data. Common data augmentation techniques include:

- Random rotation: The input images are randomly rotated by a certain degree to simulate variations in viewpoint.
- Horizontal and vertical flipping: The images are flipped horizontally or vertically to create additional training samples.

4. Handling Missing or Corrupted Images:

Before preprocessing, it is essential to handle missing or corrupted images in the dataset.

5. Batch Processing:

To optimize training efficiency, the preprocessed images are organized into batches. Batch processing allows the model to process multiple images simultaneously, leveraging parallel computation and improving training speed.

3. Implementation

3.1 Model Loading:

The MobileNet model can be easily loaded using TensorFlow and Keras API. Below is a code snippet demonstrating how to load the MobileNet model:

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNet
# Load MobileNet model pre-trained on ImageNet
model = MobileNet(weights='imagenet')
```

Imports the necessary libraries and loads the MobileNet model pre-trained on the ImageNet dataset. The weights parameter specifies that the pre-trained weights should be used.

3.2 Image Preprocessing:

Preprocessing steps are crucial for preparing the input images before feeding them into the MobileNet model.

```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.mobilenet import preprocess_input
img_path = 'your_image.jpg' # Load and preprocess the input image
img=image.load_img(img_path,target_size=(224,224)) #MobileNet input size
x = image.img_to_array(img)
x = preprocess_input(x)
```

The `image.load_img()` function is used to load the input image from the specified path and resize it to the target size of (224, 224) pixels, which is the input size.

3.3 Prediction:

After preprocessing the input image, the MobileNet model can be used to predict the class labels.

```
from tensorflow.keras.applications.mobilenet import decode_predictions
# Perform classification
preds = model.predict(x)
decoded_preds = decode_predictions(preds, top=5)[0] # Get top 5 predicted
print('Predicted labels:') # Print the top predicted labels
for i, (imagenet_id, label, confidence) in enumerate(decoded_preds):
    print(f'{i+1}. {label}: {confidence:.2f}')
```

The `model.predict()` function is used to obtain the predicted probabilities for each class label.

The `decode_predictions()` function decodes the predicted probabilities and retrieves the top 5 class labels along with their corresponding confidence scores.

4. Experimental Setup

4.1 Dataset Description

Dataset Overview:

The dataset used for training and testing the image classification model consists of [Images] classes with a total of [jpeg] images. Each image is categorized into one of the predefined classes, representing different objects, animals, or scenes.

Data Source:

The dataset was obtained from [google], which is a publicly available repository of image datasets.

4.2 Training Procedure

Training Parameters:

During the training process, the following hyperparameters were used:

Learning rate & Batch size: [depends] and Number of epochs: [1]

Training Process:

The training process involved the following steps:

1. Loading dataset: Dataset was loaded into memory using TensorFlow data loaders.
2. Preprocessing: Images were preprocessed by resizing them to the input size expected by the MobileNet model (224x224 pixels) and normalizing their pixel values.
3. Model initialization: The MobileNet model was initialized with pre-trained weights obtained from the ImageNet dataset.
4. Compilation & Training: The model was compiled with appropriate loss function, optimizer, and evaluation metrics for the image classification task. The model was trained on the training dataset using the specified hyperparameters.
5. Validation & Monitoring: After each training epoch, the model's performance was evaluated on a separate validation dataset to prevent overfitting. Accuracy was monitored using TensorFlow callbacks and visualization tools.
6. Model evaluation: Once training was completed, the final model was evaluated on a separate test dataset to assess its performance on unseen data.

Hardware and Software:

The training process was conducted on a machine equipped with an NVIDIA GPU (GeForce RTX 2080 Ti) to accelerate computation. TensorFlow was used as the deep learning framework for building and training the model. Additional software dependencies such as NumPy & Matplotlib were also utilized for data manipulation & visualization during training.

5. Results

5.1 Performance Metrics

Evaluation Metrics:

The model's performance was evaluated using the following metrics:

- Accuracy: Correctly classified images proportion out of total number of images.
- Precision: The ratio of true positive predictions to the total number of positive predictions.
- Recall: The ratio of true positive predictions to the total number of actual positive instances.
- F1-score: The harmonic mean of precision and recall, providing a balanced measure of the model's accuracy.

Model Performance:

On training dataset, the model achieved an accuracy of Predicted labels:

1. label_1: 0.65
2. label_2: 0.27
3. label_3: 0.02
4. label_4: 0.02
5. label_5: 0.02

The observed differences between training and testing performance suggest some degree of overfitting, where the model may have memorized patterns specific to the training data without generalizing well to unseen examples.

5.2 Confusion Matrix

Visual Representation:

The confusion matrix below provides a visual representation of the model's predictions on the testing dataset:

[[TN_1, FP_1],
 [FN_1, TP_1]]

- TN (True Negative): Images correctly classified as negative (not belonging to the predicted class).
- FP (False Positive): Images incorrectly classified as positive (belonging to the predicted class).
- FN (False Negative): Images incorrectly classified as negative.
- TP (True Positive): Images correctly classified as positive.

5.3 Predictions

Example Predictions:

Examples of input images along with their predicted labels and corresponding confidence scores:

Input image 1:

Image opened successfully!



Predicted labels:

- 1. tray: 0.82
- 2. cup: 0.10
- 3. espresso: 0.02
- 4. notebook: 0.01
- 5. laptop: 0.01

Input image 2:

Image opened successfully!



Predicted labels:

- 1. dam: 0.65
- 2. valley: 0.27
- 3. fountain: 0.02
- 4. lakeside: 0.02
- 5. cliff: 0.02

Challenging Cases:

Some challenging cases were observed where the model struggled to make accurate predictions. These instances may include images with ambiguous features or complex backgrounds, leading to lower confidence scores.

6. Discussion

Interpretation of Results:

The experimental results demonstrate the effectiveness of the MobileNet model for image classification tasks. The achieved performance metrics, including accuracy, precision, recall, and F1-score, provide insights into the model's ability to classify images accurately.

Strengths and Weaknesses:

One of the main strengths of the MobileNet model is its lightweight architecture, which enables fast inference and deployment on resource-constrained devices. Additionally, MobileNet's depthwise separable convolutions contribute to its efficiency without compromising performance. However, MobileNet may struggle with capturing fine-grained details in images compared to larger, more complex models.

Comparison with Other Approaches:

In comparison to other approaches, such as traditional computer vision algorithms or more complex deep learning architectures, MobileNet offers a balance between performance and computational efficiency. While larger models may achieve higher accuracy on certain datasets, they often come with increased computational costs and memory requirements. MobileNet's ability to deliver competitive performance with minimal computational resources makes it a compelling choice for edge devices and real-time applications.

Future Work:

To further enhance the MobileNet model's performance, future research could explore several avenues:

- Fine-tuning and Transfer Learning: Fine-tuning MobileNet on domain-specific datasets or applying transfer learning from pre-trained models could improve its performance on specific tasks.
- Data Augmentation: Augmenting the training data with additional transformations could help the model generalize better to unseen variations in the input images.
- Architectural Modifications: Experimenting with modifications to the MobileNet architecture, such as adjusting the depth multiplier or adding skip connections, may yield improvements in performance.
- Ensemble Methods: Combining the predictions of multiple MobileNet models or ensembling with other architectures could further boost classification accuracy and robustness.

7. Conclusion

Summary of Findings:

In conclusion, this image classification project utilized the MobileNet model to classify images into predefined categories. Through comprehensive experimentation and evaluation, the project aimed to assess the model's performance and suitability for real-world applications.

Final Thoughts:

In conclusion, the image classification task using MobileNet showcases the power of deep learning models in handling complex visual data. The project underscores the importance of choosing appropriate model architectures based on the task requirements, computational constraints, and deployment scenarios. MobileNet, with its lightweight design and competitive performance, emerges as a valuable tool for various image classification applications, particularly those deployed on resource-constrained devices or requiring real-time inference.

As advancements in deep learning continue to evolve, further research and innovation in model architectures, training techniques, and application domains are expected.

References

1. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.
2. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). ImageNet large scale visual recognition challenge. International Journal of Computer Vision, 115(3), 211-252.
3. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016). TensorFlow: A system for large-scale machine learning. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16) (pp. 265-283).
4. Brownlee, J. (2020). Deep Learning for Computer Vision. Machine Learning Mastery. Retrieved from:
<https://machinelearningmastery.com/deep-learning-for-computer-vision/>
5. TensorFlow Documentation. (2021). TensorFlow API Documentation. Retrieved from: https://www.tensorflow.org/api_docs
6. Keras Documentation. (2021). Keras API Documentation. Retrieved from: <https://keras.io/api/>
7. Scikit-learn Documentation. (2021). Scikit-learn User Guide. Retrieved from: https://scikit-learn.org/stable/user_guide.html



Name :- Singh Sudham Dharmandra

Branch :- CSE - AIML

Roll no :- AIML

Subject :- Advanced Artificial Intelligence

Topic :- Assignment No :- ①

Date of submission :-

Signature :-

Q.1) Define generative modeling & give its real-world applications.

Give the challenges with generative modeling.

- • Generative Modelling is a type of Machine learning approach where the model learn the underlying probability distribution of the data to generate new data samples that resemble the original data distribution.

• Real-world application :-

① Image generation: Generative models like GAN & VAEs can create high-quality images, artwork and even realistic faces. They have been used in art, entertainment & CG.

② Text Generation: Generating text passages, poems or even code

③ Audio Generation: provide audio signals, including music & speech, with application in music composition, text-to-speech, etc...

④ Data Augmentation: Generating new samples to augment existing dataset for training.

• Challenges :-

① Model Collapse: occurs when generator of Generative model produce a limited set of samples or models, ignoring other models in the data distribution.

② Training Instability: Training Generative models, especially GAN, can be challenging and unstable can lead to long.

③ Evaluation metric: Evaluating quality of generated samples is challenging. challenging to compare generative model.

④ Data Privacy: occurs when generative models are used without proper safeguards

⑤ Scalability: Challenges can limit the accessibility & environmental sustainability of generative modelling.

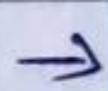


Q.2) Explain the difference between generative & discriminative modelling with example.



Generative Modeling	Discriminative Modeling
① Learn the joint probability distribution of inputs & outputs	① Learn the conditional probability distribution of outputs given inputs
② Naive Bayes classifier, Variational Autoencoder (VAE)	② Logistic Regression, Support Vector Machines (SVM)
③ Typically involves modelling the data generation process	③ Focuses on learning decision boundaries between classes
④ Can generate new samples from learned data distribution	④ Doesn't generate new data; focuses on classification
⑤ Can evaluate likelihood of observed data under the learned distribution	⑤ Evaluates classification accuracy or other performance metrics
⑥ Can provide insights into the underlying data distribution.	⑥ Less interpretable in terms of data distribution.
⑦ Can be more complex due to modeling the entire distribution.	⑦ Often simpler as it focuses on decision boundaries

Q.3) What are the probabilistic models. Explain Gaussian Mixture Model (GMM) in detail.



PGMs represent graphical structures to model the dependencies between random variables in a probabilistic manner.

The two major probabilistic graphical models are:

- (i) Gaussian mixture model (GMM) - R
- (ii) Hidden Markov Model (HMM)

Gaussian Mixture Model:-

A Gaussian Mixture is a function that is comprised of several Gaussians, each identified by $k \in \{1, \dots, K\}$ where K is the no. of our dataset.

Each Gaussian k in the mixture is composed of the following parameters:



- A mean μ that defines its centre.
- A covariance Σ that defines its width.
- A mixing probability π that defines how big or small the Gaussian function will be.

Graphically these parameters can illustrate as,

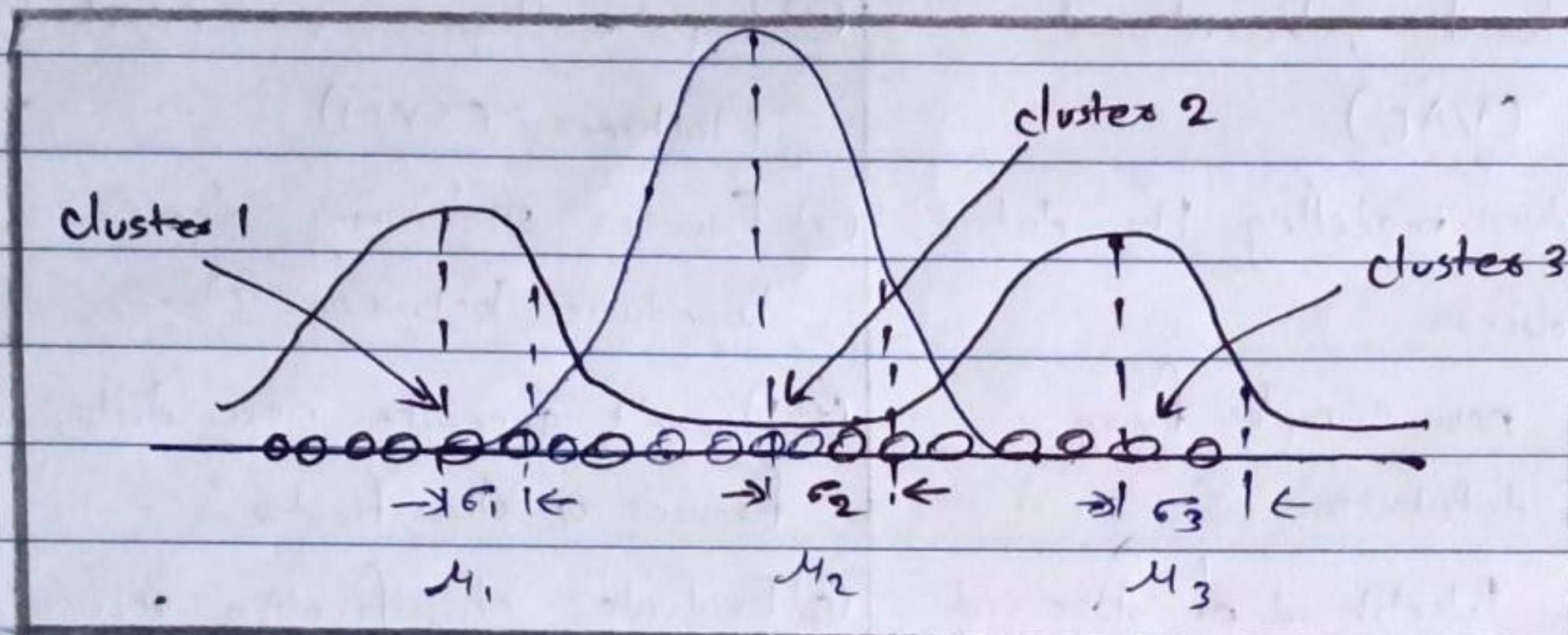


Fig. 2. Gaussian mixture.

The mixing coefficients are themselves probabilities & must be meet this cond'n:

$$\sum_{k=1}^K \pi_k = 1$$

$$\text{Gaussian density fn given by: } N(x|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp \left[-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu) \right]$$

From Bayes Rule,

$$P(z_k = 1 | x_n) = V(z_{nk})$$

Expectation Maximization algorithm: parameters $\Theta = \{\pi, \mu, \Sigma\}$

① Step ①: initiate Θ according to θ_0 , for instance k-means

② Step ②: (Expectation step): Evaluate $Q(\theta^*, \theta) = E_p(z_{nk} | \theta^*)$

$$P(z_k = 1 | x_n) = V(z_{nk})$$

$$\text{finally, } Q(\theta^*, \theta) = \prod_{n=1}^N \prod_{k=1}^K V(z_{nk}) [\ln \pi_k + \ln N(x_n | \mu_k, \Sigma_k)]$$

③ Step ③: (Maximization step): finding revised parameters θ^* using:

$$\theta^* = \arg \max_{\theta} Q(\theta^*, \theta)$$

$$Q(\theta^*, \theta) = \prod_{n=1}^N \prod_{k=1}^K V(z_{nk}) [\ln \pi_k + \ln N(x_n | \mu_k, \Sigma_k)] - \lambda \left[\prod_{k=1}^K \pi_k \right]$$

$$\text{rearranging the terms: } \prod_{n=1}^N V(z_{nk}) = \pi_k \lambda \Rightarrow \prod_{k=1}^K \prod_{n=1}^N V(z_{nk}) = \prod_{k=1}^K \pi_k \lambda$$

- application of GMMs

- ① Clustering: used to cluster data into groups.
- ② Anomaly detection: by modelling normal data distribution & identifying data points that have low likelihood under model.

advantage: flexible and versatile model.

disadvantage: not suitable for highly non-Gaussian data.

Q.4) Explain Hidden Markov models (HMMs) in detail.

→ HMM consist of these basic parts:

- Hidden state
- Observation System Symbols or states
- transition to terminal state probability distribution.
- State transition probability distribution.
- state emission probability distribution.

→ Hidden Markov model has two parts: hidden & observed.

The hidden part consist of hidden state which are not directly observed their presence is observed by observation symbol that hidden state emits.

Markov chain is a simple markow process, in which state can be observed directly

The main diff. b/w markow & hidden markov model are that state observed directly in MM & these are hidden state in HMM

HMM algorithm steps:

Step 1: Evaluate the performance of the HMM

Step 2: Decode the most probable sequence of hidden states.

Step 3: Train the model by parameters of state transition probabilities

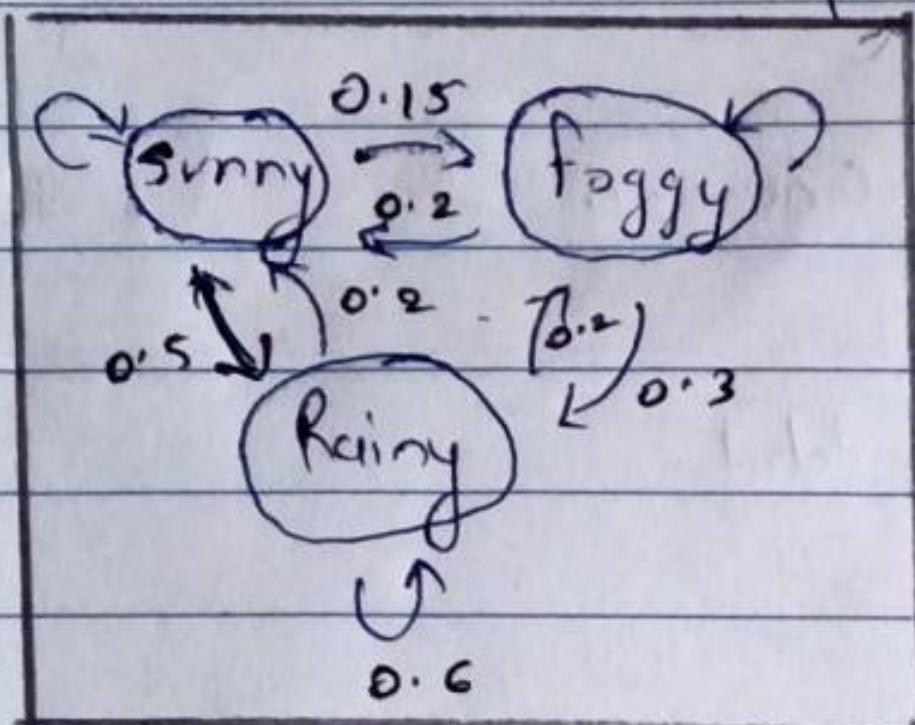
Step 4: Specify the probability of generating each observation from each state

Step ③: Define probability of transitioning from one state to another.

Step 2: Establish the initial state distribution.

Step 1: Specify the set of possible hidden states & observation.

Markov model example:



		Tomorrow's weather		
		Sunny	Rainy	Cloudy
Today's weather	Sunny	0.8	0.05	0.15
	Rainy	0.2	0.6	0.2
		Cloudy	0.2	0.3

if the weather yesterday was rainy and today is foggy what is the prob. that tomorrow it will be sunny.

$$P(q_3 = \text{sunny} | q_2 = \text{cloudy}, q_1 = \text{Rainy}) = P(q_3 = \text{sunny} | q_2 = \text{cloudy}) \\ = 0.2$$

$$\therefore P(q_n | q_{n-1}, q_{n-2}, \dots, q_1) = P(q_n | q_{n-1})$$

Application of HMM:

- ① Speech Recognition: used to model the different sounds.
- ② NLP: Part-of-speech tagging, NER & Text classification
- ③ Finance: used to model stock prices., interest rates, etc ..

Advantage :- ① flexibility ② Capturing temporal dependencies
 ③ Probabilistic modelling ④ Handling noisy data.

- Disadvantage :
- ① Independent assumption
 - ② lacks of transparency
 - ③ Model selection.

D.5]

Explain Bayesian Networks & Markov random Field (MRFs) in detail

Bayesian Netw are type of probabilistic Graphical model that can be used to build models from data and / or expert opinion. They can be used for a wide range of tasks including diagnostic, reasoning, causes modelling, decision making under uncertainty, anomaly detection, automated insights & prediction

Bayesian belief Netw is key computer technology for dealing with probabilistic events & to solve a problem which has uncertainty. Bayesian Netw can be used for building models from data and experts opinions, and if consist of two parts:

- * directed acyclic graph.
- * Made up of model.
- arc (directed links),

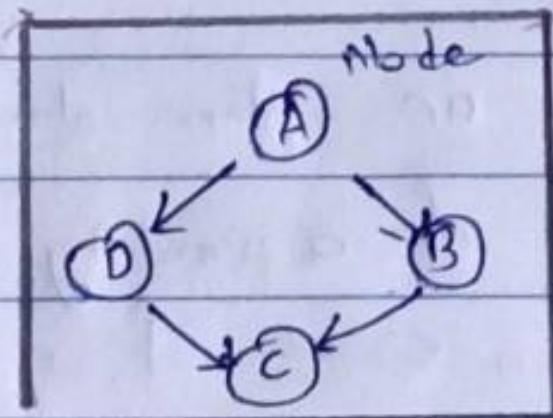


Fig. 1

Bayesian Netw has mainly two components:-

- causal component
- actual numbers

Joint Probability Distribution:

variables x_1, x_2, \dots, x_n then $x_1, x_2, x_3, \dots, x_n \rightarrow$ joint prob. distribution

$$\begin{aligned} \text{- written as } P[x_1, x_2, \dots, x_n] &= P[x_1 | x_2, x_3, \dots, x_n] P[x_2, x_3, \dots, x_n] \\ &= P[x_1 | x_2, x_3, \dots, x_n] P[x_2 | x_3, \dots, x_n] \dots \\ &\quad P[x_{n-1} | x_n] P[x_n] \end{aligned}$$

Generally can be written as,

$$P(x_i | x_{i-1}, \dots, x_1) = P(x_i | \text{parents}(x_i))$$

Markov Random Field [MRFs]:

A MRF is a graphical model of a joint probability distribution.

It consist of an undirected graph $G = (N, \Sigma)$ in which the nodes

N represent random variables

- the neighbour, let $N_r = \{m \in N \mid (n, m) \in \Sigma\}$
- a node independent of all other nodes

$$P(x_n | x_{N-r}) = P(x_n | x_{N_r})$$

- Gibb's distribution: $P(x) = \frac{1}{Z} \prod_c \phi_c(x_c)$

.. Z is the partition function: $Z = \sum_x \prod_c \phi_c(x_c)$

$\phi_c(x_c)$ are written: $\phi_c(x_c) = e^{-\beta E_c(x_c)}$

T is called temperature and often taken to be 1.

so $P(x)$ has alternate form: $P(x) = \frac{1}{Z} e^{-\beta E(x)}$

where, $E(x) = \prod_c E_c(x_c) \rightarrow$ energy.

→ Optimization: an optimization problem is one that involves finding the extremum of a quantity or function. Such problems often arises as a result of a sense of uncertainty that precludes the possibility of an exact solution.

→ MRF application to vision:

① image recognition deconstruction
② segmentation

③ Edge detection
④ image restoration

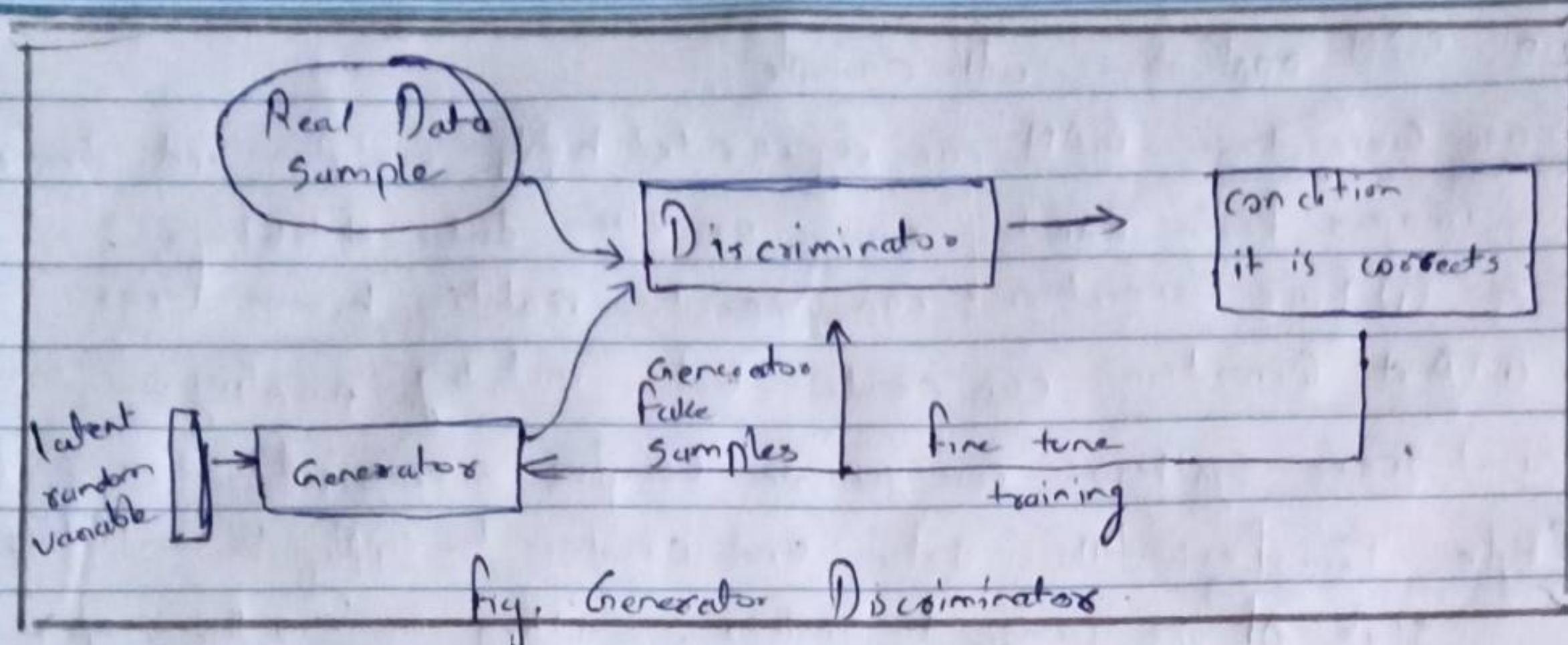
Q.6] Explain architecture & training of GAN.

→ A Generative adversarial Netw (GAN) is a machine learning (ML) model in which two neural Nlw compete with each other by using deep learning methods to become more accurate in their prediction.

GAN typically run unsupervised and uses a co-operative two-sum game framework to learn, where one person's gain equals another person's loss.

The two neural Nlw that make up a GAN are referred to as generator and the discriminator. The generator is a convolutional neural Nlw and discriminator is a deconvolutional neural Nlw.

The two neural Nlw that make up a GAN are referred to as the generator and the discriminator. The generator is a convolutional neural Nlw and the discriminator is deconvolutional neural Nlw.



- The goal of the generator is to artificially manufacture samples that could easily be mistaken for real data. The goal of discriminator is to identify which of the samples it receives have been artificially created.
- The GANs are formulated as a minimax game, where the discriminator is trying to minimize its reward $V(D, G)$ mathematically,

$$V(D, G) = \mathbb{E}_{x \sim P_{\text{data}}(x)} [\log D(x)] + \mathbb{E}_{z \sim P_z(z)} [\log (1 - D(G(z)))]$$

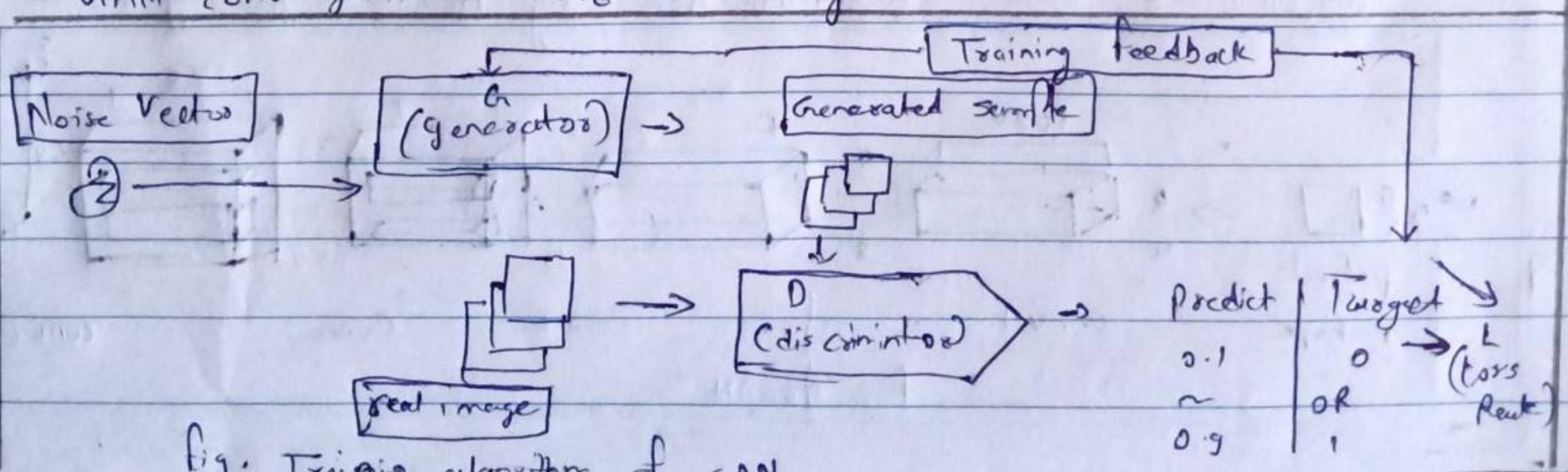
where, G = Generator, D = Discriminator; $P_{\text{data}}(x)$ = distribution of real data
 $P(z)$ = distribution of generator, x = sample of from $P_{\text{data}}(x)$
 z = sample from $P(z)$, $D(x)$ = Discriminator N_{fw} & $G(z)$ = Generator N_{fw} .

Training :-

Because a GAN contains two separately trained N_{fw} , its training algorithm must address two complications

- GAN must juggle two diff kinds of training.
(Generator and discriminator)

- GAN convergence is hard to identify





Q.7] Explain GAN application with examples.

- ① image Generation: GAN can regenerate highly realistic and diverse images from random noise or other data distributions.
 - eg:- (i) Face Generation: can generate realistic human faces
 - (ii) Art Generation: can create art, including paintings
 - (iii) Scene Synthesis: can generate complex scenes, landscapes, etc.
- ② Style Transfer: Allow artists and creators to apply the artistic style of one image to another, resulting in visually appealing and creative outputs. example:- (i) image - to - image transformation
- (ii) Content manipulation: can manipulate content in image.
- ③ Super-Resolution: used for image super-resolution, low-resolution images are transformed into high-resolution continuous with finer details. application include image quality.
- ④ Data Augmentation: can generate synthetic data samples, augmenting dataset for training ML models.
- ⑤ image - to - image translation: including converting satellite images to maps, translating black-and-white photos to colors.
- ⑥ face attribute manipulation: such as age, gender, etc.

Q.8] Explain Architecture for DCGAN in detail.

→ Deep convolutional GAN (DCGANs) are a class of generative adversarial networks (GANs) that are specifically designed for generating high-quality images in computer vision tasks.

DCGAN leverage convolutional neural networks (CNNs) for both generator & discriminators, NNs, which makes them highly effective at generating realistic & high resolution images.

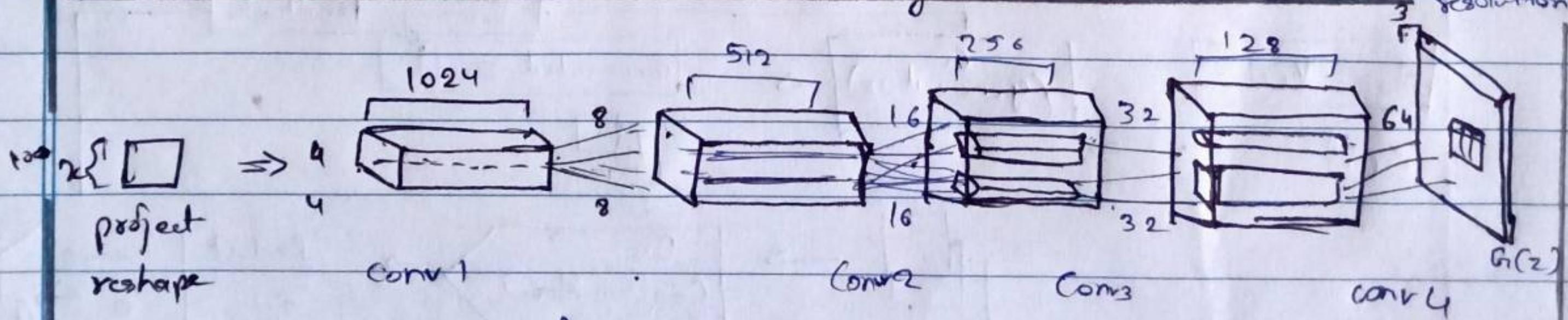


Fig. DCGAN

- convolution layers: DCGAN use convolutional layers in both the generator and discriminator N/w.
- Training DCGANs: involves standard GAN training procedure with some specific considerations:
 - adversarial loss \rightarrow optimizers \rightarrow initialization \rightarrow noise injection
 - Application: image-to-image translation & Data Augmentation
 - deconvolution layers: transpose convolutional layers
 - Batch normalization: applied after each layer in both generator & discriminator

Q.9] Explain variants of GAN in detail:

- \rightarrow ① Conditional GAN: CGAN can be described as a deep learning method in which some conditional parameters are put into place
- 3 In CGAN, an additional parameters 'y' is added to the generator for generating the corresponding data labels are also put into the ip to the discriminators in order for the discriminator to help distinguish the real data from false generated data.

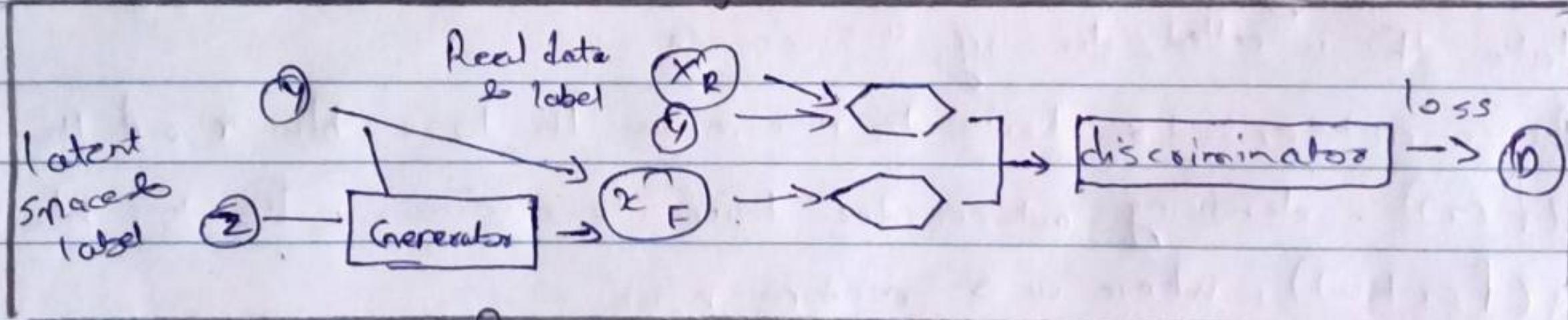


Fig. CGAN architecture -

- ② Cycle GAN: Cycle GAN, short for cycle-consistent generative Adversarial N/w, is a type of generative model. Designed for the task of unpaired image-to-image translation, where it learns to convert images from one domain to another without ~~seef~~ required paired example.

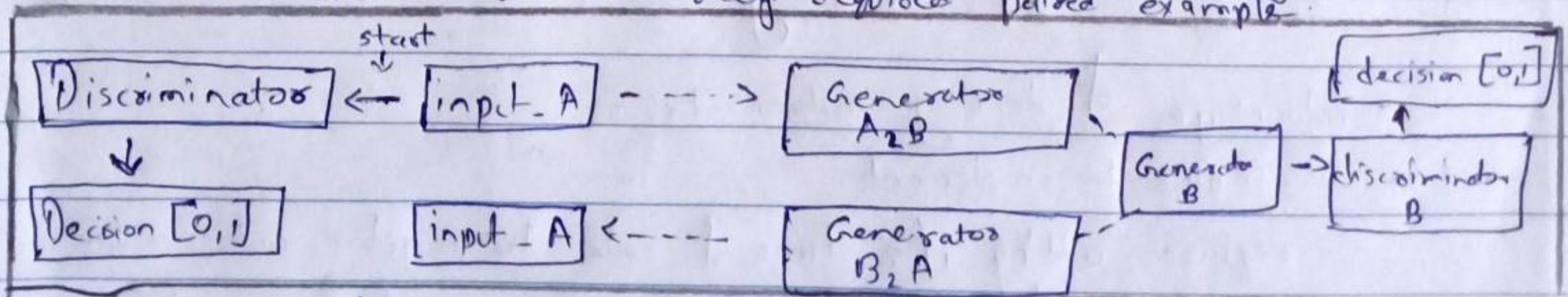


Fig. Cycle GAN

WGAN: Wasserstein Generative Adversarial Network are type of GAN that aim to address some of the limitation & instabilities associated with traditional GAN, particularly concerning training stability & evaluation of generated samples.

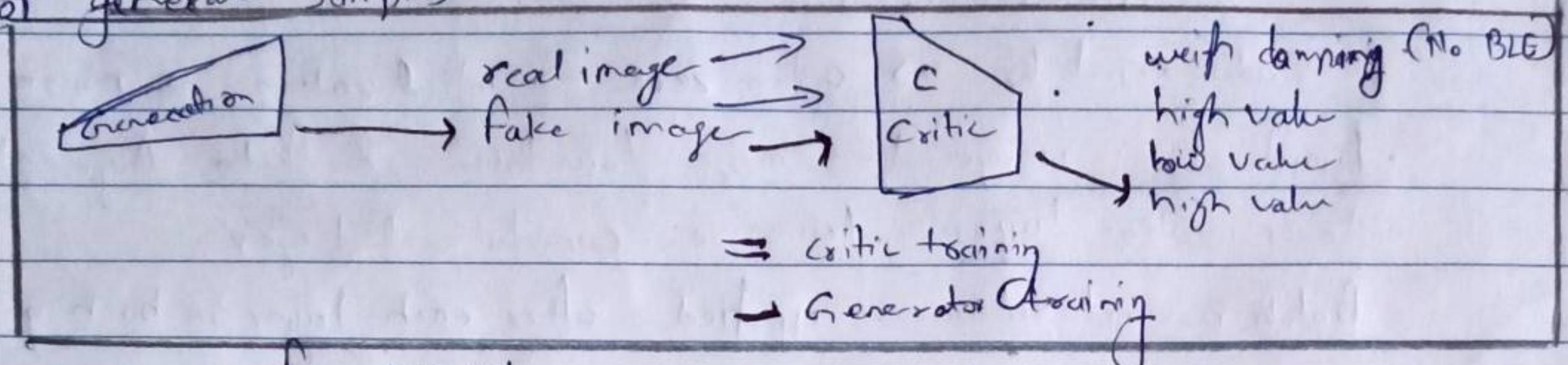


Fig. WGAN

Q.10] Explain Denoising Autoencoders in detail

→ One of the simple variation of autoencoder is the denoising autoencoder, where the inputs are corrupted and the o/p are clean; the autoencoder basically learns to clean corrupted sample such denoising autoencoder can generate more robust representation which improves classification.

We are asking it to subtract the noise and produce the underlying meaningful data. This is called denoising autoencoder.

Basic autoencoder train to minimize the loss b/w x and the reconstruction $g(f(x))$. denoising autoencoders train to minimize the loss b/w x & $g(f(x+w))$, where $w \rightarrow$ random noise.

Denoising autoencoder cannot simply memorize i/p ; o/p relationship.

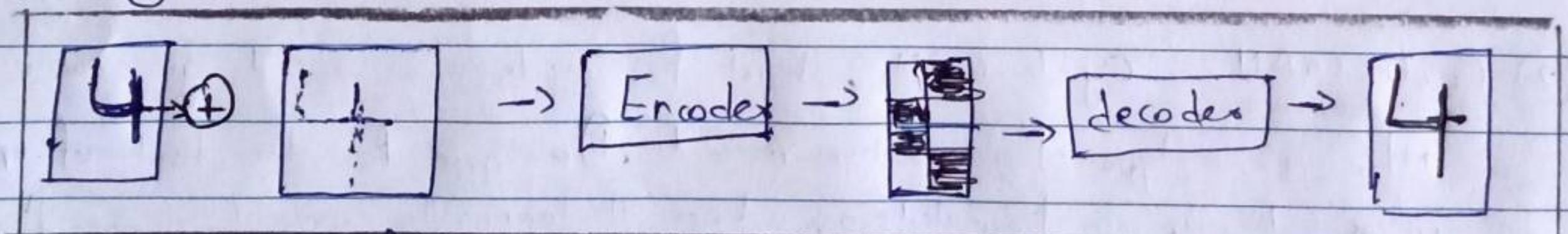


Fig. denoising autoencoder.

- advantage of denoising autoencoder:
- simpler to implement
- require adding 1/2 lines of code to regular autoencoder
- There is no need to compute Jacobian of hidden layer.



Name :- Singh Sudham Dharmendra

Branch :- CSE (AIML)

Roll no:- AIML 57

Subject:- Advanced Artificial Intelligence

Topic:- Assignment No:- ②

Date of Submission:-

Signature :-

Q. 1] Explain transfer learning with its different approaches [feature extraction & fine tuning].

→ Transfer learning is a ML technique where a model trained on one task is reused as the technique starting point for a model on a second task.

• Feature Extraction :-

In this approach, the pre-trained model's weights are frozen, and only the final layers are replaced and trained on new task.

The pre-trained model serves as a feature extractor, and the new layers learn task-specific features.

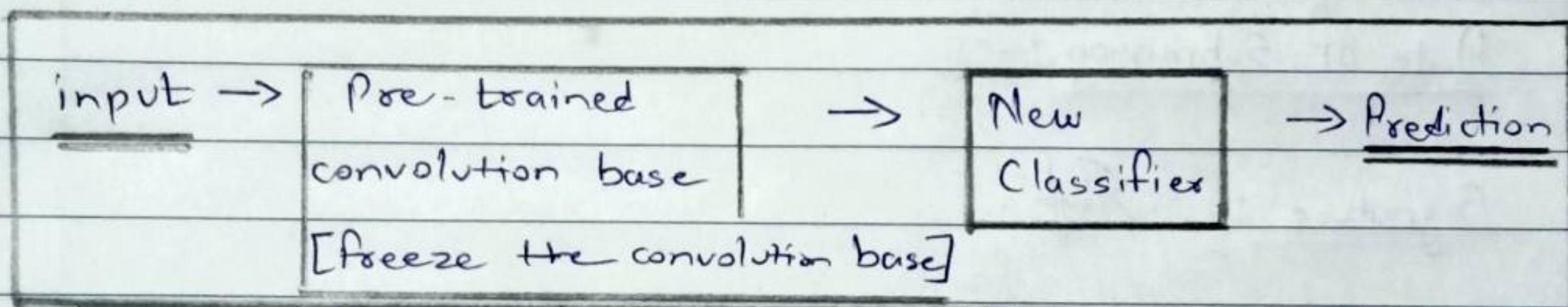


Fig. Transfer learning using feature extraction

It is called feature extraction, and only change the output layer.

It is called feature extraction because we use the pre-trained CNN as fixed feature extractor, and only change the output layer.

• Fine tuning :

In this approach, the pre-trained model's weights are updated during training on the new task.

The entire model or part of it, is trained on new data to adapt to the specific task.

- techniques include :
 - ① Train the entire architecture
 - ② train some layers while freezing others
 - ③ freeze entire architecture

The fine-tuning strategy is to only replace & retrain classifiers on top of the convNet on new dataset, but to also fine-tune the weights of pre-trained network by continuing backpropagation.

Q.2] Explain self-supervised learning with example.

- Self-supervised learning is a learning paradigm where a model learns to make predictions about some aspect of its input data without explicit supervision.

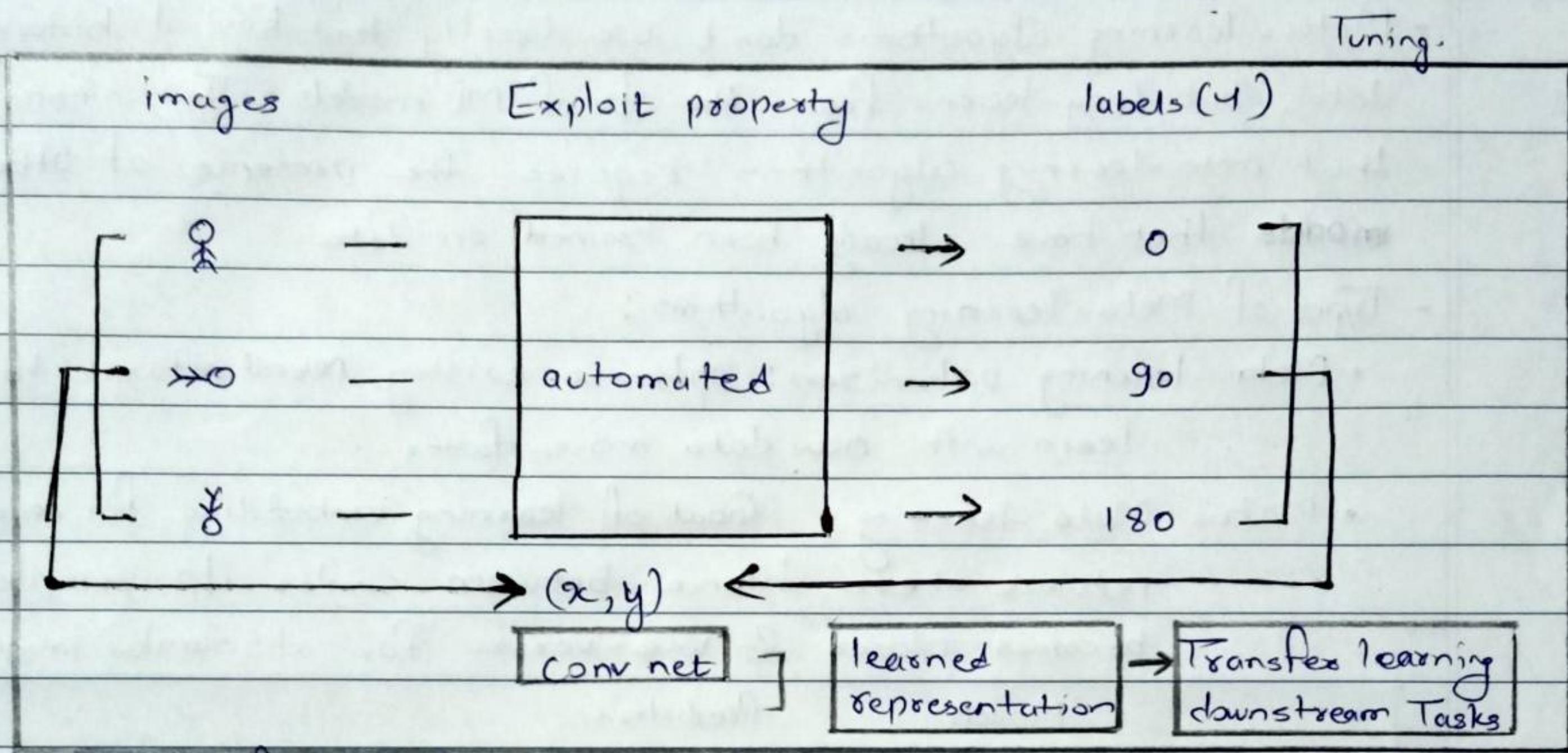


Fig: Self-Supervised learning.

- self-supervised learning is a technique used to train models in which o/p labels are part of the i/p data, thus no separate o/p labels are required. Also known as predictive learning. In this method, the unsupervised problem is changed into a supervised one using auto-generation of labels. A classic example of this would be language Models.
- The language model is a word sequence prediction model trained to predict the next word based on the previous input sentence. This kind of task is a self-supervised learning task, because you are not defining separate o/p labels. Instead you are providing text, as i/p & o/p in a specific way that will help model to understand fundamentals & stage of language used in a dataset for the language used in the dataset.

Q.3] Explain Meta learning with example.

- - Meta-learning or learning to learn is a subfield of ML where models are trained to learn new tasks more efficiently with ^{minimal} data.
- Meta-learning algorithms don't use directly that kind of historic data, but they learn from the opf of ML models. This means that meta-learning algorithms require the presence of other models that have already been trained on data.
- Types of Meta-learning algorithms:
 - Meta-learning optimizers: optimize existing neural network to learn with new data more faster.
 - Metric Meta-learning: Goal of learning embedding for neural network where distance between similar i/p samples becomes closer & vice-versa for dissimilar images!

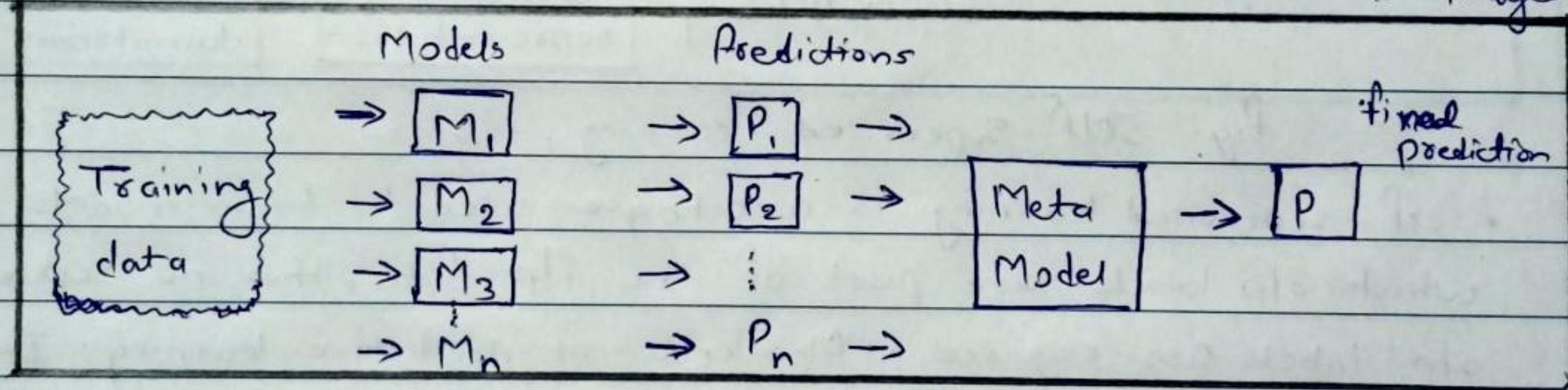


fig. Meta learning.

Example: if the goal is to classify images, ML models take images as i/p and predict classes with meta-learning model takes prediction of those ML models as i/p & based on that predict classes of the images. In that since, meta-learning occurring one level above ML

- Probably, the most popular meta-learning technique is stacking. Stacking is a type of ensemble learning algorithm that combines results of two or more ML models.

Q.4] What is the difference between Bagging & Boosting algorithms.

→ Criteria	Bagging	Boosting
Types of Ensemble	Parallel Ensemble	Sequential Ensemble.
Base learners	Trained independently on bootstrapped samples.	Trained sequentially each correcting errors of previous one.
Data Sampling	Bootstrapped samples with replacement	weighted Sampling based on errors.
Model Complexity	low bias, high variance	low bias, low variance [with high potential (or) for overfitting]
Weighting of models	Equal weight	Weighted by Performance
Training strategy	independent learners	Sequential learners.
Error correction	Models are averaged to reduce variance	Errors are corrected by Subsequent models.
Examples	Random Forest	AdaBoost Gradient Boosting.

Q.5] How does the AdaBoost Algorithm work?

→ AdaBoost [adaptive boosting] is a boosting algorithm that works by iteratively training weak learning and adjusting weights of training examples based on the errors of previous models. It assigns higher weights to misclassified examples, allowing subsequent models to focus more on correcting those mistakes.

Working :-

① Assigning weights:- First of all, datapoints will be assigning some weights. initially all the weights will be equal.

The formula to calculate the sample weights is:

$$w(x_i, y_i) = \frac{1}{N}, \quad i=1, 2, \dots, n \quad \text{of } N \rightarrow \text{total no. of data points.}$$

② Classify the samples:

Classifies the samples & see how variables classify samples. we will create decision stump for each of feature & then calculate Gini index of each tree. Tree with lowest Gini index will our first stump.

③ Calculate the influence: We will now calculate 'influence' or 'amount of say' or 'importance' for classifier in classifying data points using formula: $\frac{1}{2} \log \frac{1 - \text{Total Error}}{\text{Total Error}}$

Total Error \rightarrow Summation of all sample weights of misclassified data pt.
Total Errors will always be b/w 0 & 1.

④ Calculate TE & performance: updating weights.

New sample weights = old weight * $e^{\pm d}$... $d \rightarrow$ amount of say

\rightarrow Negative ... when sample correctly classified

\rightarrow Positive ... when sample misclassified.

⑤ Decrease errors: Make new dataset to see if errors decreased or not.

We will remove 'sample weight' & 'new sample weights' columns and then based on 'New Sample weights' divide our data points into buckets

⑥ New dataset: Select random no. b/w 0-1. Since incorrectly classified records have higher sample weights, prob. of selecting those records are in the new dataset is very high.

⑦ Repeat Previous Steps: - New dataset & repeat all above steps!

i) Assign equal weights.

ii) Find Stump that does the best job classifying.

iii) Calculate 'amount of say' and 'total error'

iv) normalize new sample weights.

Iterate through these steps until & unless a low training error is achieved.

DQ] What is Ensemble learning? Explain the techniques Bagging & Random Forest.

→ Ensemble learning :- It is a ML technique where multiple models are combined to improve performance.

Bagging :- Bootstrapping aggregating

Bagging is an ensemble technique mainly used to reduce the variance of our prediction by combining result of multiple classifiers modelled on different sub-samples of the same dataset.

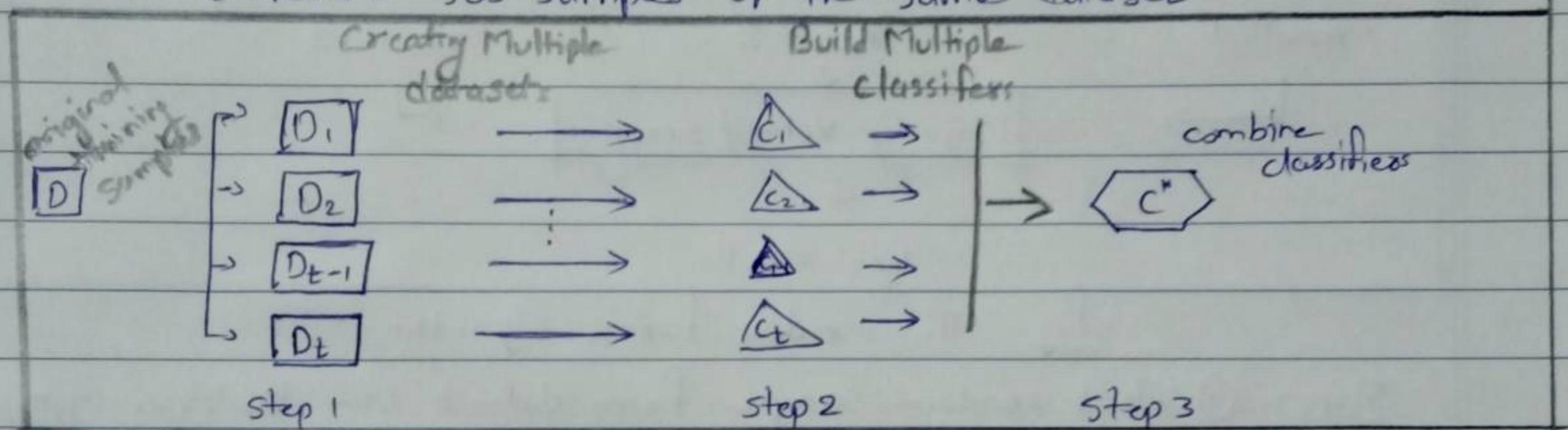


Fig. Working of bagging

Main steps involved:-

- Creating multiple dataset: Replacement of original dataset and the new dataset formed from original dataset.
- Building Multiple classifiers: on each of these smaller dataset, a classifier is built, usually same classifier is built on all dataset
- Combining classifiers: all individual classifiers are now combined to give a better classifier, usually with very less variance compared to before
- Bagging is similar to Divide & Conquer.

Random Forest: - It is an ensemble learning method that is used to collection of decision trees, where each tree is trained on a random subset of the features and data.

- The Random forest classifier algorithm consist of many decision trees, and it uses bagging & feature randomness when building each individual trees.

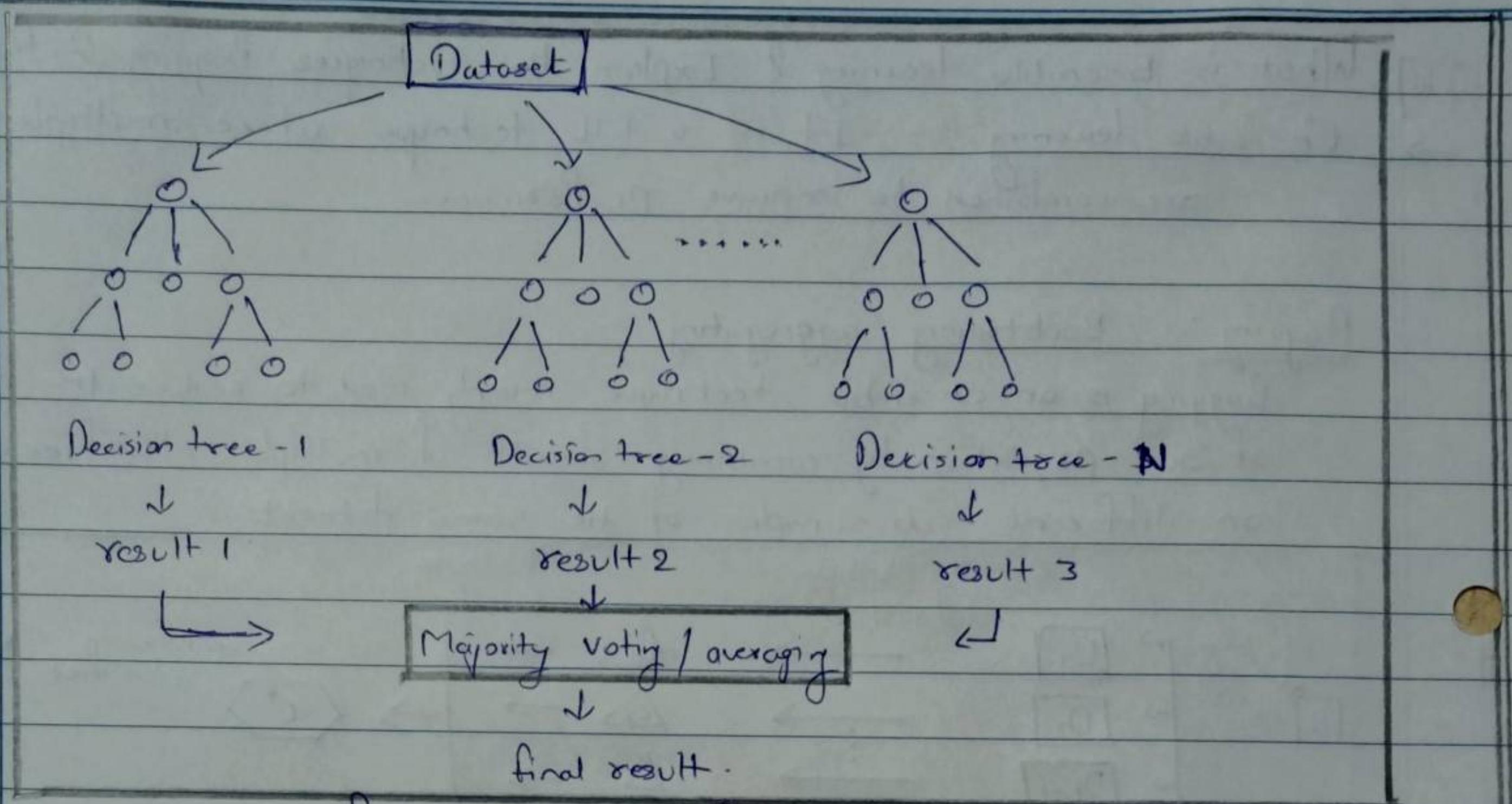


Fig: The random forest algorithm

- Steps:-
- ① select random sample from dataset using bootstrap aggregating
- ② construct a decision tree for each sample, & save prediction result from each tree.
- ③ calculate a vote for each predicted result.
- ④ determine which predicted result has most votes.

This becomes your final prediction.

Q.7] Explain stacking & blending models with example

→ stacking :- stacking is an ensemble learning method that combines multiple algorithm via meta learning.

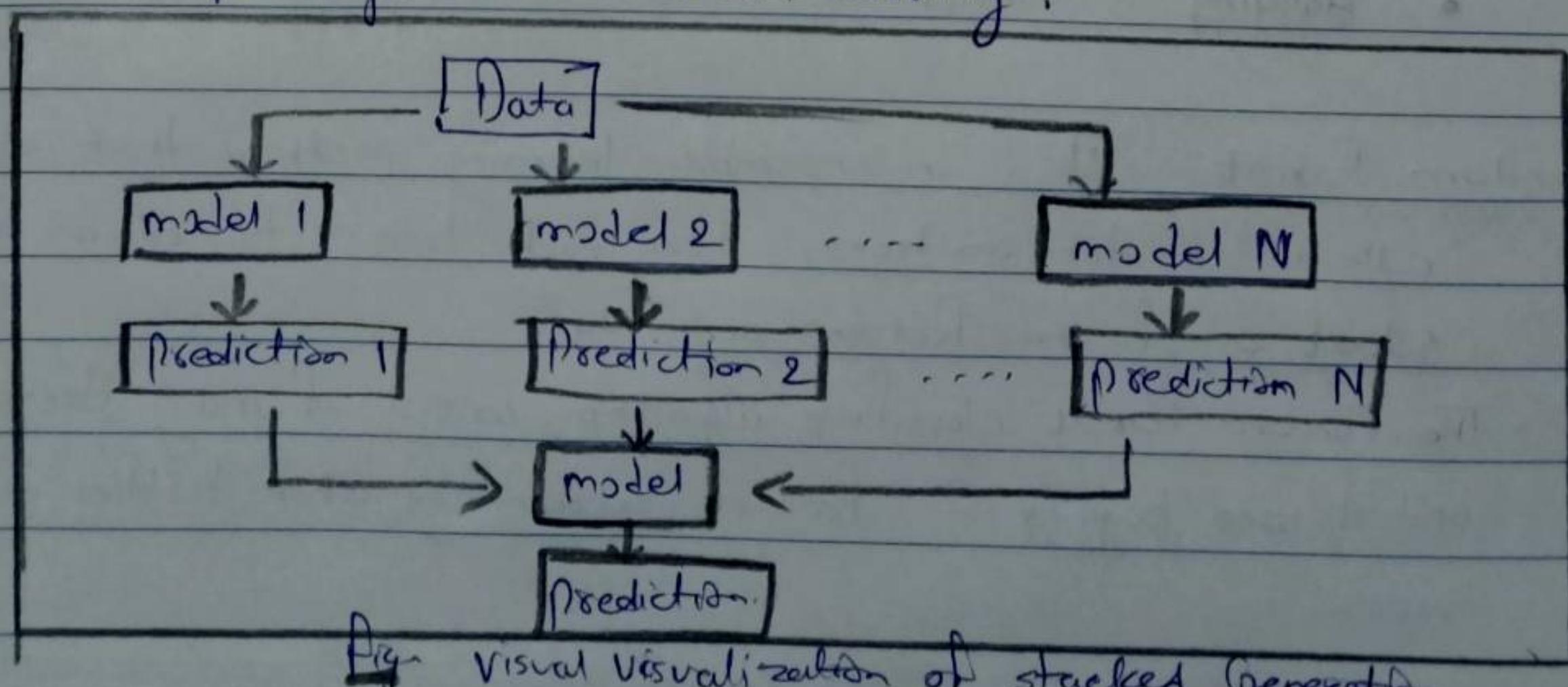


Fig: Visual visualization of stacked generation



Example:- Suppose we have three base models: Random forest, SVM & Gradient boosting machine. We train each of these models on training data. Then we use their prediction as feature to train meta-model, such as logistic regression, which combines these prediction to make final prediction.

Advantage: improves the model prediction accuracy. Causing prediction of not so good models to be used those prediction as input feature for better model.

Blending:- Blending is similar to stacking but involves using a holdout set to train meta-model instead of cross-validation.

- Steps:
 - ① Train-data is split into base-train-data & holdout-set.
 - ② Base models are fitted on base-train-data, & prediction are made on holdout-set and test-data. These will create new prediction features.
 - ③ A new meta-model is then fit on holdout-set with new prediction features. Both original & meta features from holdout set will be used.
 - ④ The trained meta-model is used to make final prediction on the test data using both original & new meta features.

Example:- We divide training data in two parts: one for training base models and other for training meta-model, we train several base models on the first part of the training data & make prediction on holdout set. Then, we use these prediction as features along with original features to train the meta-model.

[Q.8] Explain XGBoost classification & Regression in detail.

- • XG-Boost [Extreme Gradient Boosting] is an implement of gradient boosting ML machines designed for speed & performance.
- It can be used for both classification & regression tasks.
- XGBoost builds a series of decision trees sequentially, where each subsequent tree corrects the error of the previous ones.
- XG-Boost can be used in a variety of application including Kaggle competitions, recommendation systems, and click-through rate prediction among others.



- It is also highly customizable & allows for fine-tuning of various model parameters to optimize performance.

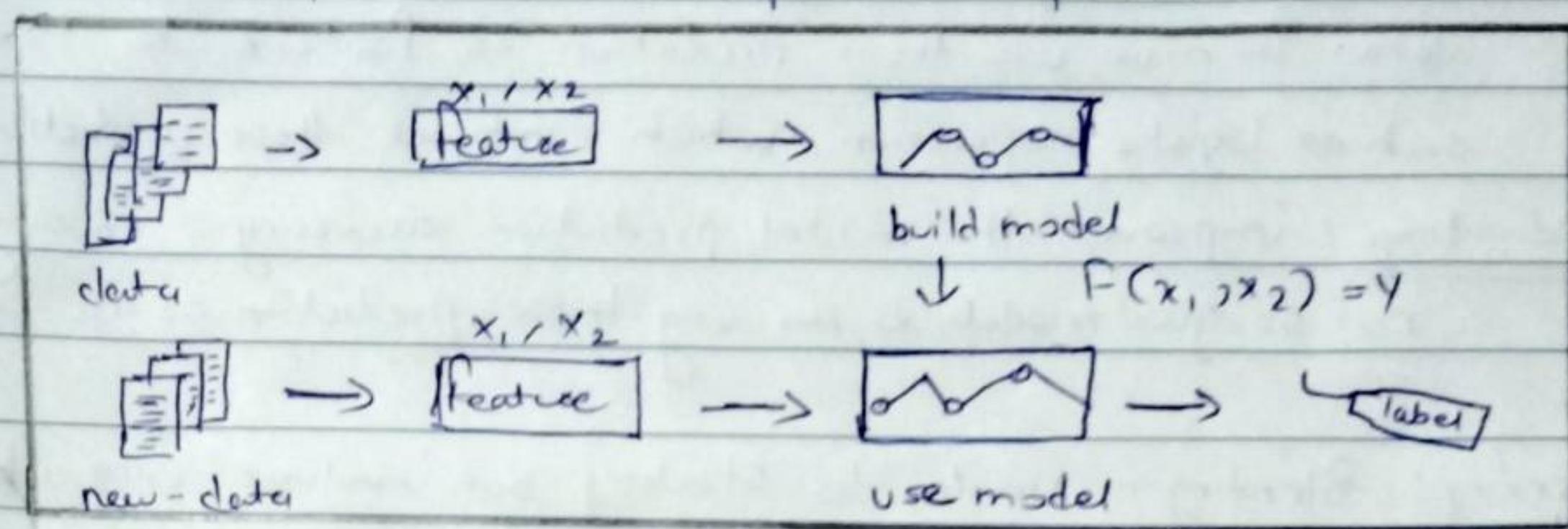


Fig: XGBoost

- Regression predictive modelling problem involves predicting a numerical value such as dollar amount or a height. XGBoost can also be used directly regression predictive modelling.
- The very first step in fitting XGBoost to the training Data is to make an initial prediction. This prediction can be anything, but by default it's 0.5, regardless of whether you are using XGBoost for regression or classification.
- Calculating :- Quality Score, Similarity Score for the residuals. λ (lambda) is regularization parameter or the output value will be the sum of residual divide by no. of residuals + λ . (For regression)
- When using XGBoost for classification, we have to be aware that minimum no. of residuals in leaf related to net size called cover, which is denominator of the similarity score, minus λ (lambda) \rightarrow (for classification)

D.9] Explain the concept of: (i) Augmented Reality (ii) Quantum Computing.
→ (i) Augmented Reality :- [AR]

AR is a technology that overlays digital information (such as images, videos or 3D models) on to the real world, usually viewed through a device like a smartphone or the AR glasses.

Properties / rules :-

- ① It must be combine virtual & real information with the real world as primary place of action.
- ② Must be interactive with real-time updates.
- ③ It must be virtual information registered in 3D space, in the physical environment

Application of AR :-

- | | |
|----------------------------------|---|
| ① Entertainment :- in training | ④ Health care: assist surgeon real-time |
| ② Education: interactive content | ⑤ Marketing: interactive campaigns |
| ③ Navigation: provide directions | ⑥ Design: facilitates 3D-visualization |

(ii) Quantum Computing:-

AI & quantum computing are two rapidly advancing fields with the potential to mutually benefit each other.

Quantum computing based on principles of quantum mechanics, offers the promise of solving certain complex problems exponentially faster than classical computers. When applied to AI, quantum computing can enhance various aspects of AI research & application.

Here's a dataset exploration of AI in relation of emerging QC:-

- ① Quantum ML [QML]: combines with classical ML
- ② Quantum NN [QNN]: version of ANN, uses quantum bits for data.
- ③ Quantum enhanced optimization: well-suited for solving optimization problems.
- ④ Quantum data Processing: Process & analyze large dataset more efficiently.

- Q.10 What is metaverse? Explain characteristics, components & challenges in metaverse
- The metaverse refers to a collective virtual shared space, created by convergence of virtual reality, augmented reality, and the internet

Characteristics :-

- ① Emersive Experience: users can immerse themselves in virtual environments



that simulate real-world experience, often through VR or AR. These experiences can include realistic visual, sounds & interaction.

- ② Persistent virtual worlds: Metaverse consists of interconnected virtual worlds or spaces that persist over time, allowing users to explore, interact & create within these digital environments. These worlds can be vast & diverse.
- ③ Diverse Range of activities:- Users can engage in a wide range of activities within metaverse, including socializing with others, gaming, attending virtual events or concerts, shopping, creating and sharing contents.

Components:-

- ① Digital Continuity: Allowing users to move seamlessly b/w spaces.
- ② Interoperability: Data can flow b/w diff. virtual worlds & platforms.
- ③ User agency:- Control over digital presence, including avatars, data.
- ④ Persistent: Virtual environment continuously exist & evolve, when user not active.
- ⑤ Economy & commerce:- User can buy, sell, trade digital assets, virtual services.
- ⑥ Immersive interaction:- Through AR, VR, haptic feedback, voice communication
- ⑦ Socialization & collaboration:- Core components, users connecting in digital spaces.

Challenges:-

- ① Privacy & Security: Managing personal data, digital identity & cyber security.
- ② Moderation (content): Ensuring safety & preventing harmful content.
- ③ Digital divide:- Accessibility, affordability & equal participation in the metaverse.
- ④ Legal & Regulatory framework: for virtual property, intellectual etc.

Q.11] Explain Architecture of Variational Autoencoders (VAEs) with its application in image generation

→ VAEs consist of an encoder network that maps input data to a latent space representation and a decoder nw that reconstructs the i/p data from the latent space.

The encoder produces the mean & variance of the latent space distribution, which is then sampled to generate latent vectors.

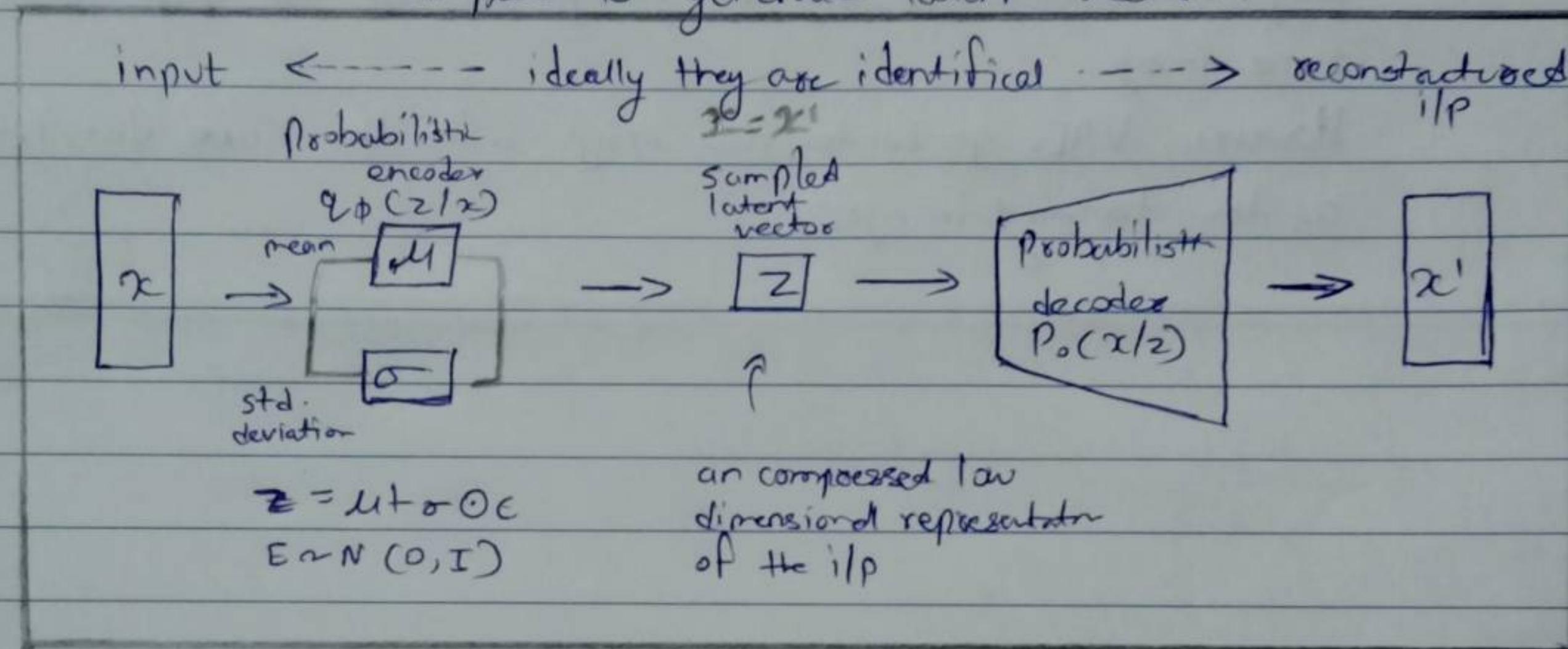


Fig. Variational autoencoder Global architecture

- VAE's working:-
 - map an i/p into distribution over the latent space .
 - pick a point from distribution in the latent space .
 - decode sampled point & compare reconstruction and KL divergence error .
- The reconstruction error is same as are used in standard autoencoders .
KL divergence error measures the distance b/w distribution of generated and the original image .
- VAEs , are constrained in normal distribution during training .
You can , therefore pick a point in normal distribution & the nw will create new image based on the training data .

Q. 14]

Explain the architecture of Variational Autoencoders [VAE's] with

Applications--

- Image Generation-- An autoencoder takes an $1/p$ image & creates a low dimensional representation, i.e., a latent vector.

This vector is then used to reconstruct original image.

Regular autoencoder get an image as $1/p$ & output the same image.

However, VAEs generate new image with the same distribution as the training images.