```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import numpy as np

# Load the data from the .npz file
data = np.load('/content/rotated_mnist (1).npz')

# Access the data arrays
rotated_x_train = data['x_train']
rotated_y_train = data['y_train']
rotated_x_test = data['x_test']
rotated_y_test = data['y_test']

# Print the shape of the data arrays to show the data clust
print("Shape of rotated_x_train:", rotated_x_train.shape)
print("Shape of rotated_y_train:", rotated_y_train.shape)
print("Shape of rotated_x_test:", rotated_x_test.shape)
print("Shape of rotated_y_test:", rotated_y_test.shape)
```

```
Shape of rotated_x_train: (152400, 28, 28)
Shape of rotated_y_train: (152400,)
Shape of rotated_x_test: (26004, 28, 28)
Shape of rotated_y_test: (26004,)
```

```
Start coding or generate with AI.
```

```
!pip install tensorflow
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.0.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.13.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.71.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.13.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19,
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorfl
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0
```

## Loading pre-trained vae model

[ ] ↳ 2 cells hidden

## Unsupervised ORACLE MODEL

```
_, _, z_train = encoder.predict(rotated_x_train)
_, _, z_test = encoder.predict(rotated_x_test)

latent_dim = z_train.shape[1]

class SymmetryGenerator(keras.Model):
    def __init__(self, latent_dim):
        super(SymmetryGenerator, self).__init__()
        self.net = keras.Sequential([
            layers.Dense(64, activation='relu'),
            layers.Dense(64, activation='relu'),
            layers.Dense(latent_dim, kernel_regularizer=keras.regularizers.l2(0.001))
        ])

    def call(self, inputs):
        return self.net(inputs)

def invariance_loss(z, generator, classifier, epsilon=1):#value of epsilon experimented
    z_prime = z + epsilon * generator(z)
    return tf.reduce_mean(tf.square(classifier(z) - classifier(z_prime)))

def normalization_loss(z, generator):
    norms = tf.norm(generator(z), axis=1)
    mean_norm = tf.reduce_mean(norms)
    return (
        tf.reduce_mean(tf.square(norms - 1.0)) +
        tf.reduce_mean(tf.square(norms - mean_norm))
    )

def orthogonality_loss(generators, z):
    loss = 0
    for i in range(len(generators)):
        for j in range(i+1, len(generators)):
            dot_prods = tf.reduce_sum(generators[i](z) * generators[j](z), axis=1)
            loss += tf.reduce_mean(tf.square(dot_prods))
    return loss

def build_classifier(latent_dim):
    classifier = keras.Sequential([
        layers.Input(shape=(latent_dim,)),
        layers.Dense(128, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(1, activation='sigmoid') #Output layer is 1, as we are predicting the difference between z and z'
    ])
    classifier.compile(
        optimizer=keras.optimizers.Adam(learning_rate=1e-3),
        loss='mse' #Using mean squared error as we are training to predict the difference between z and z'
    )
    return classifier
```

```
4763/4763 ━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step
813/813 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
```

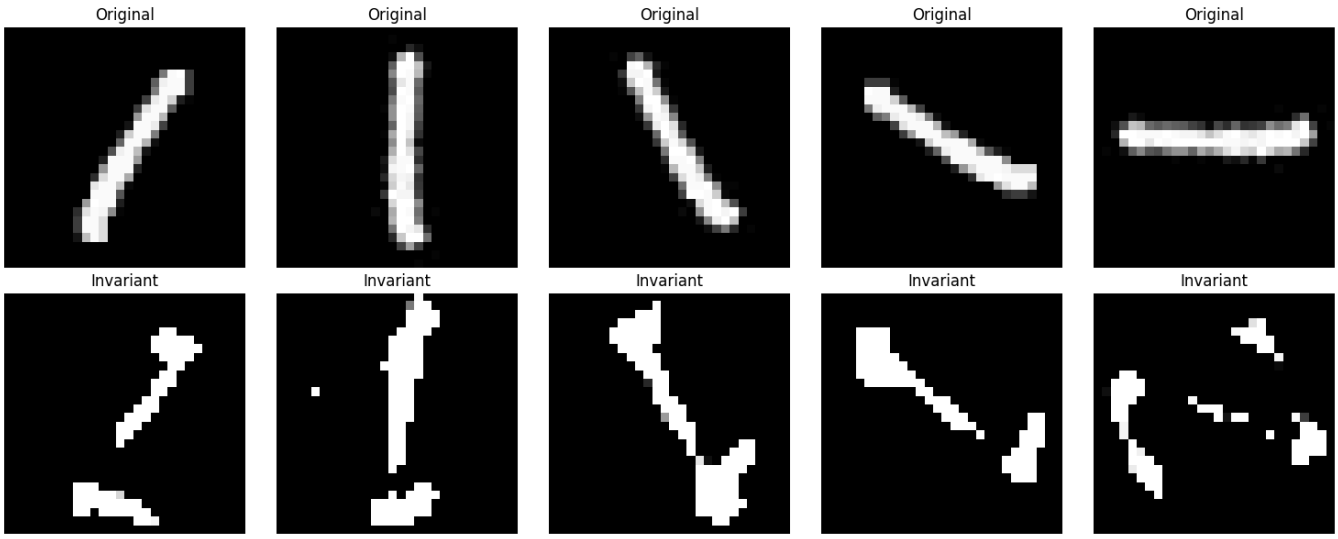
```
def train_symmetry_generator(z, num_generators=2):
    generators = [SymmetryGenerator(latent_dim) for _ in range(num_generators)]
    classifier = build_classifier(latent_dim)

    # Initialize optimizer after creating generators
    optimizer = keras.optimizers.Adam(learning_rate=1e-5) # Moved optimizer initialization here

    for epoch in range(100):
        with tf.GradientTape() as tape:
            inv_loss = sum(invariance_loss(z, gen, classifier) for gen in generators)
            norm_loss = sum(normalization_loss(z, gen) for gen in generators)
            ortho_loss = orthogonality_loss(generators, z) if num_generators > 1 else 0

            total_loss = inv_loss + norm_loss + ortho_loss

        # Get gradients for all generator variables
        all_gen_vars = [var for gen in generators for var in gen.trainable_variables]
        grads = tape.gradient(total_loss, all_gen_vars) # Calculate gradients for all generator variables

        # Apply gradients to all generator variables
```

```
        optimizer.apply_gradients(zip(grads, all_gen_vars)) # Apply gradients to all generator variables

        #Train classifier to predict the difference between z and z'
        z_prime = z + 1e-3 * generators[0](z)
        classifier.train_on_batch(z, z-z_prime)
        #print(f"Epoch: {epoch}, Total Loss: {total_loss}")

        #print(generators[0](z[:1]))
    return generators, classifier


generators, classifier = train_symmetry_generator(z_train, num_generators=2)
```

## Rotation Invariant Model

```
import matplotlib.pyplot as plt
def build_rotation_invariant_network(encoder, generators, latent_dim):

    input_image = keras.Input(shape=(28, 28, 1))

    # Encode the input image to latent space
    z_mean, z_log_var, z = encoder(input_image)

    # Apply the learned symmetry transformations
    transformed_z = z
    for generator in generators: #Apply each generator
        transformed_z = transformed_z + 0.1 * generator(transformed_z) #Apply the preserved symmetry.

    # Decode the transformed latent representation
    transformed_image = decoder(transformed_z)

    # Build the rotation-invariant model
    rotation_invariant_model = keras.Model(inputs=input_image, outputs=transformed_image)

    return rotation_invariant_model




rotation_invariant_network = build_rotation_invariant_network(encoder, generators, latent_dim)



#Example Usage and Visualization:
def visualize_rotation_invariance(original_images, invariant_images, num_samples=5):
    plt.figure(figsize=(15, 6))
    for i in range(num_samples):
        plt.subplot(2, num_samples, i+1)
        plt.imshow(original_images[i].squeeze(), cmap='gray')
        plt.title("Original")
        plt.axis('off')

        plt.subplot(2, num_samples, num_samples+i+1)
        plt.imshow(invariant_images[i].squeeze(), cmap='gray')
        plt.title("Invariant")
        plt.axis('off')

    plt.tight_layout()
    plt.show()

#Generate invariant images
invariant_images = rotation_invariant_network.predict(rotated_x_train[:5])
visualize_rotation_invariance(rotated_x_train[:5], invariant_images)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
```


Original
Original
Original
Original
Original
Invariant
Invariant
Invariant
Invariant
Invariant