```
# prompt: mount drive

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,)
```

```
!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) (2.5.0)
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>=3.5.6
```

Loading pre-trained vae model

```
latent_dim = 32  # Adjust as needed

def sampling(args):
    z_mean, z_log_var = args
    batch = tf.shape(z_mean)[0]
    dim = tf.shape(z_mean)[1]
    epsilon = tf.random.normal(shape=(batch, dim))
    return z_mean + tf.exp(0.5 * z_log_var) * epsilon

encoder_inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, 3, activation="relu", strides=2, padding="same")(encoder_inputs)
x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Flatten()(x)
x = layers.Dense(16, activation="relu")(x)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)
z = layers.Lambda(sampling, output_shape=(latent_dim,), name='z')([z_mean, z_log_var])

encoder = keras.Model(encoder_inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()

latent_inputs = keras.Input(shape=(latent_dim,))
x = layers.Dense(7 * 7 * 64, activation="relu")(latent_inputs)
x = layers.Reshape((7, 7, 64))(x)
x = layers.Conv2DTranspose(64, 3, activation="relu", strides=2, padding="same")(x)
x = layers.Conv2DTranspose(32, 3, activation="relu", strides=2, padding="same")(x)
decoder_outputs = layers.Conv2DTranspose(1, 3, activation="sigmoid", padding="same")(x)
decoder = keras.Model(latent_inputs, decoder_outputs, name="decoder")
decoder.summary()

class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.total_loss_tracker = keras.metrics.Mean(name="total_loss")
        self.reconstruction_loss_tracker = keras.metrics.Mean(
            name="reconstruction_loss"
        )
        self.kl_loss_tracker = keras.metrics.Mean(name="kl_loss")

    @property
    def metrics(self):
        return [
            self.total_loss_tracker,
            self.reconstruction_loss_tracker,
            self.kl_loss_tracker,
        ]

    def call(self, inputs):
        z_mean, z_log_var, z = self.encoder(inputs)
        #z = self.sampling(z_mean, z_log_var)#added
        reconstruction = self.decoder(z)
        return reconstruction

    def sampling(self, z_mean, z_log_var):
        epsilon = tf.random.normal(shape=tf.shape(z_mean))
        return z_mean + tf.exp(0.5 * z_log_var) * epsilon

    def train_step(self, data):
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            reconstruction_loss = tf.reduce_mean(
                tf.reduce_sum(
                    keras.losses.binary_crossentropy(data, reconstruction), axis=(1, 2)
                )
            )
            kl_loss = -0.5 * (1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var))
            kl_loss = tf.reduce_mean(tf.reduce_sum(kl_loss, axis=1))
            total_loss = reconstruction_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_weights)
        self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
        self.total_loss_tracker.update_state(total_loss)
        self.reconstruction_loss_tracker.update_state(reconstruction_loss)
        self.kl_loss_tracker.update_state(kl_loss)
        return {
            "loss": self.total_loss_tracker.result(),
            "reconstruction_loss": self.reconstruction_loss_tracker.result(),
            "kl_loss": self.kl_loss_tracker.result(),
```

```
        }
```

Model: "encoder"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 28, 28, 1)	0	-
conv2d (Conv2D)	(None, 14, 14, 32)	320	input_layer[0][0]
conv2d_1 (Conv2D)	(None, 7, 7, 64)	18,496	conv2d[0][0]
flatten (Flatten)	(None, 3136)	0	conv2d_1[0][0]
dense (Dense)	(None, 16)	50,192	flatten[0][0]
z_mean (Dense)	(None, 32)	544	dense[0][0]
z_log_var (Dense)	(None, 32)	544	dense[0][0]
z (Lambda)	(None, 32)	0	z_mean[0][0], z_log_var[0][0]

Total params: 70,096 (273.81 KB)
Trainable params: 70,096 (273.81 KB)
Non-trainable params: 0 (0.00 B)
Model: "decoder"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 32)	0
dense_1 (Dense)	(None, 3136)	103,488
reshape (Reshape)	(None, 7, 7, 64)	0
conv2d_transpose (Conv2DTranspose)	(None, 14, 14, 64)	36,928
conv2d_transpose_1 (Conv2DTranspose)	(None, 28, 28, 32)	18,464
conv2d_transpose_2 (Conv2DTranspose)	(None, 28, 28, 1)	289

```
# Loading:
encoder = encoder #Use the original encoder definition.
decoder = decoder #Use the original decoder definition.
vae = VAE(encoder, decoder)

# Load weights:
encoder.load_weights("/content/drive/MyDrive/GSOC/GSOC/encoder_weights.weights.h5")
decoder.load_weights("/content/drive/MyDrive/GSOC/GSOC/decoder_weights.weights.h5")

print("VAE weights loaded, model reconstructed.")
```

```
VAE weights loaded, model reconstructed.
```

Build and train the classifier

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Hyperparameters from document
h_norm = 1.0  # Normalization loss weight
h_ortho = 1.0  # Orthogonality loss weight


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

# Define classifier/oracle model
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')
    ])
    classifier.compile(
        optimizer=keras.optimizers.Adam(learning_rate=1e-3),
        loss='binary_crossentropy',
```

```
            metrics=['accuracy']
        )
        return classifier

    # Build and train classifier
    classifier = build_classifier(latent_dim)
    history = classifier.fit(
        z_train, rotated_y_train,
        validation_data=(z_test, rotated_y_test),
        epochs=20,
        batch_size=128
    )

    # Symmetry generator network
    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)
            ])

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

    # Enhanced loss functions as in  paper's formulations
    def invariance_loss(z, generator, classifier, epsilon=1e-3):
        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)) +  # Eq. 10 first term
            tf.reduce_mean(tf.square(norms - mean_norm))  # Eq. 10 second term
        )

    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 h_ortho * loss


    def train_symmetry_generator(z, classifier, num_generators=1):
        generators = [SymmetryGenerator(latent_dim) for _ in range(num_generators)]
        optimizer = keras.optimizers.Adam(learning_rate=3e-4)

        for epoch in range(50):
            with tf.GradientTape() as tape:
                # Calculate each loss component separately
                inv_loss = sum(invariance_loss(z, gen, classifier) for gen in generators)
                norm_loss = h_norm * 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

            grads = tape.gradient(total_loss, [g.trainable_variables for g in generators])
            for gen, grad in zip(generators, grads):
                optimizer.apply_gradients(zip(grad, gen.trainable_variables))

        return generators

    # Train with orthogonality constraints
    generators = train_symmetry_generator(z_train, classifier, num_generators=1)
```

```
4763/4763 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step
813/813 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
Epoch 1/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.5251 - loss: -183796896.0000 - val_accuracy: 0.5316 - val_loss: -277598643
Epoch 2/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5313 - loss: -11918983168.0000 - val_accuracy: 0.5316 - val_loss: -3407849
Epoch 3/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5343 - loss: -84893753344.0000 - val_accuracy: 0.5316 - val_loss: -1330701
Epoch 4/20
```

```
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5316 - loss: -283134164992.0000 - val_accuracy: 0.5316 - val_loss: -339975
Epoch 5/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5324 - loss: -669665918976.0000 - val_accuracy: 0.5316 - val_loss: -695308
Epoch 6/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5320 - loss: -1314637742080.0000 - val_accuracy: 0.5316 - val_loss: -12424
Epoch 7/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5306 - loss: -2288690135040.0000 - val_accuracy: 0.5316 - val_loss: -20291
Epoch 8/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5298 - loss: -3681084768256.0000 - val_accuracy: 0.5316 - val_loss: -31050
Epoch 9/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5332 - loss: -5503632015360.0000 - val_accuracy: 0.5316 - val_loss: -45249
Epoch 10/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5316 - loss: -7954837274624.0000 - val_accuracy: 0.5316 - val_loss: -63464
Epoch 11/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5301 - loss: -11121402576896.0000 - val_accuracy: 0.5316 - val_loss: -8629
Epoch 12/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5286 - loss: -15023093055488.0000 - val_accuracy: 0.5316 - val_loss: -1144
Epoch 13/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5297 - loss: -19804398288896.0000 - val_accuracy: 0.5316 - val_loss: -1486
Epoch 14/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5309 - loss: -25511384317952.0000 - val_accuracy: 0.5316 - val_loss: -1897
Epoch 15/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5303 - loss: -32416555073536.0000 - val_accuracy: 0.5316 - val_loss: -2382
Epoch 16/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5312 - loss: -40520575352832.0000 - val_accuracy: 0.5316 - val_loss: -2951
Epoch 17/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5316 - loss: -50074843348992.0000 - val_accuracy: 0.5316 - val_loss: -3611
Epoch 18/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5319 - loss: -60907124162560.0000 - val_accuracy: 0.5316 - val_loss: -4373
Epoch 19/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5320 - loss: -73517131366400.0000 - val_accuracy: 0.5316 - val_loss: -5244
Epoch 20/20
1191/1191 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.5311 - loss: -88266216833024.0000 - val_accuracy: 0.5316 - val_loss: -6234
```

```
# prompt: save the generator in drive

from google.colab import drive
import os

# Assuming 'generators' is the list of trained generators from the previous code.
# Save each generator's weights to your Google Drive.
drive.mount('/content/drive')

save_dir = '/content/drive/MyDrive/GSOC/saved_generators'  # Specify your desired save directory
os.makedirs(save_dir, exist_ok=True)

for i, generator in enumerate(generators):
    generator.save_weights(os.path.join(save_dir, f'generator_{i}.h5'))

print(f"Generators saved to {save_dir}")
```

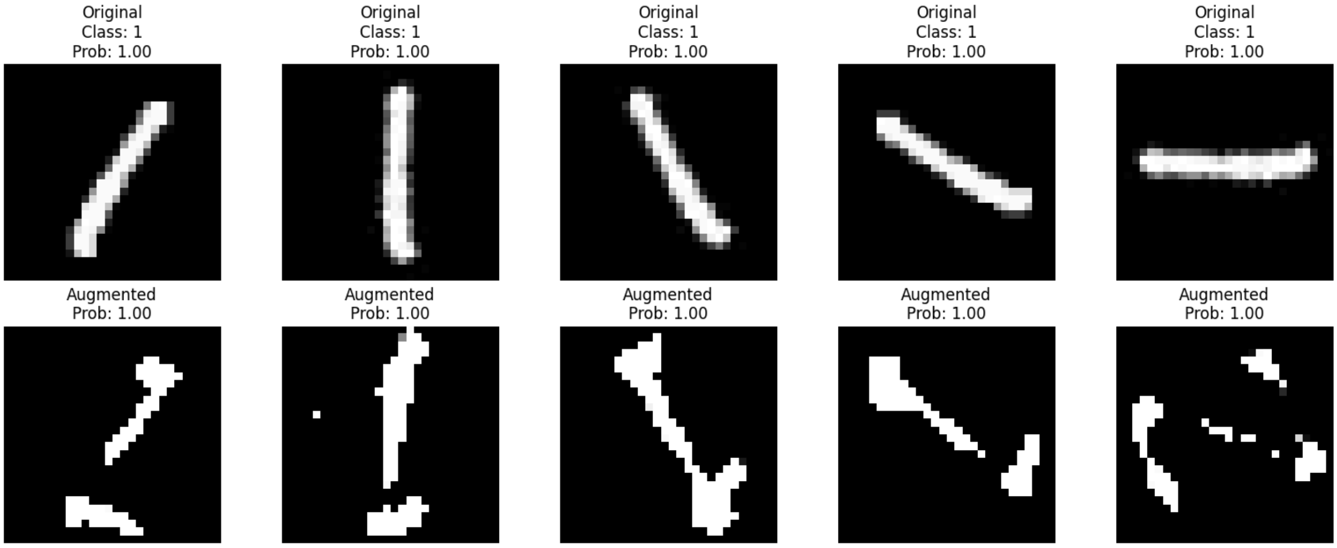
```
# Apply learned symmetry to generate new samples
def generate_symmetric_samples(z, generator, epsilon=0.1):
    return z + epsilon * generator(z)


augmented_z = generate_symmetric_samples(z_train, generators[0])
augmented_images = decoder.predict(augmented_z)
```

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

```
import matplotlib.pyplot as plt
import numpy as np

def visualize_symmetry_effects(original_images, augmented_images, classifier, num_samples=5):
    """Visualize original vs symmetry-transformed images with classification probabilities"""

    # Get predictions for both sets
    orig_probs = classifier.predict(encoder.predict(original_images[:num_samples])[2])
    aug_probs = classifier.predict(augmented_z[:num_samples])

    plt.figure(figsize=(15, 6))

    for i in range(num_samples):
        # Original image
        plt.subplot(2, num_samples, i+1)
        plt.imshow(original_images[i].squeeze(), cmap='gray')
        plt.title(f"Original\nClass: {int(rotated_y_train[i])}\nProb: {orig_probs[i][0]:.2f}")
        plt.axis('off')
```

```
        # Transformed image
        plt.subplot(2, num_samples, num_samples+i+1)
        plt.imshow(augmented_images[i].squeeze(), cmap='gray')
        plt.title(f"Augmented\nProb: {aug_probs[i][0]:.2f}")
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Generate and visualize augmented images
visualize_symmetry_effects(rotated_x_train[:5], augmented_images[:5], classifier)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 298ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
```




```
# Apply symmetry transformation to latent space
epsilon = 0.1  # Small step size for infinitesimal transformation
z_transformed = z_train + epsilon * generators[0](z_train)


# Decode the transformed latent representations
decoded_images = decoder.predict(z_transformed)
```

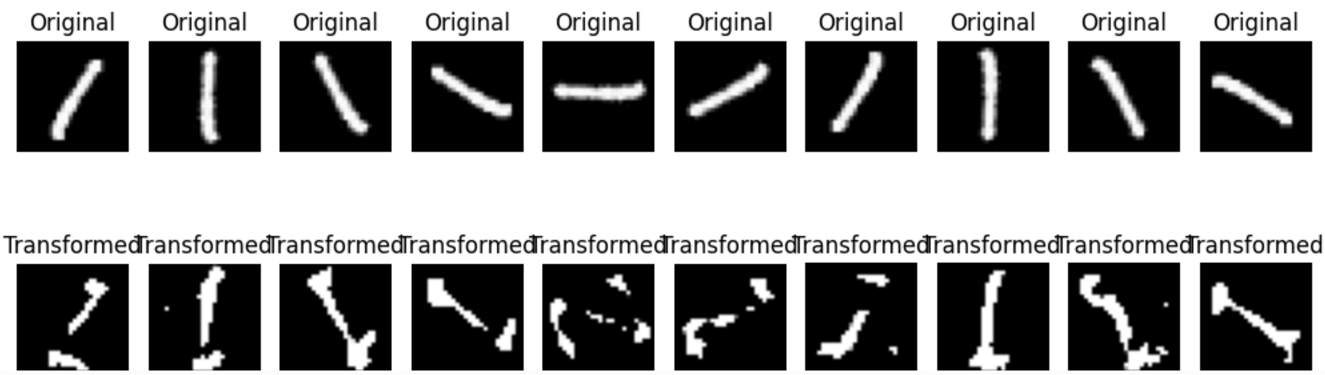
```
4763/4763 ━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step
```

```
def plot_latent_flow(z_samples, generator, decoder, epsilon=0.1, steps=5):
    """Visualize continuous symmetry transformations in latent space"""

    flow_images = []
    current_z = z_samples.copy()

    for _ in range(steps):
        current_z = generate_symmetric_samples(current_z, generator, epsilon)
        flow_images.append(decoder.predict(current_z))

    plt.figure(figsize=(2*steps, 2))
    for i in range(steps):
        plt.subplot(1, steps, i+1)
        plt.imshow(flow_images[i][0].squeeze(), cmap='gray')
        plt.title(f"Step {i+1}")
        plt.axis('off')
    plt.show()

# Visualize flow for a sample image
sample_z = z_train[:1]  # Use first training sample
```

```
plot_latent_flow(sample_z, generators[0], decoder, epsilon=0.2, steps=5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 322ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
```


```
import matplotlib.pyplot as plt

# Number of images to visualize
num_images = 10

plt.figure(figsize=(10, 4))
for i in range(num_images):
    # Original image
    plt.subplot(2, num_images, i + 1)
    plt.imshow(rotated_x_train[i].squeeze(), cmap='gray')
    plt.axis('off')
    plt.title("Original")

    # Transformed image
    plt.subplot(2, num_images, num_images + i + 1)
    plt.imshow(decoded_images[i].squeeze(), cmap='gray')
    plt.axis('off')
    plt.title("Transformed")

plt.tight_layout()
plt.show()
```




```
def analyze_transformations(encoder, decoder, classifier, generator, images):

    # Get latent representations
    z = encoder.predict(images)[2]

    # Generate transformed versions
    z_prime = z + 0.1 * generator(z)
    x_prime = decoder.predict(z_prime)

    # Classification consistency
    orig_preds = classifier.predict(z)
    trans_preds = classifier.predict(z_prime)
    class_consistency = np.mean(np.abs(orig_preds - trans_preds))

    # Latent space displacement
    latent_change = np.mean(np.linalg.norm(z_prime - z, axis=1))

    # Image reconstruction difference
    # Reshape x_prime to match the shape of images before calculating MSE
    x_prime = x_prime.squeeze() # remove the channel dimension from x_prime
    image_mse = np.mean((images - x_prime)**2)

    print(f"Classification Consistency: {class_consistency:.4f}")
    print(f"Average Latent Displacement: {latent_change:.2f}")
```

```
    print(f"Image Space MSE: {image_mse:.4f}")

# Analyze on validation set
analyze_transformations(encoder, decoder, classifier, generators[0], rotated_x_test[:100])
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
Classification Consistency: 0.0003
Average Latent Displacement: 9.53
Image Space MSE: 4526.2056
```

BinaryCrossEntropy in Oracle or Classification model(just for comparison)

[ ] ↳ 6 cells hidden