Auto Encoder

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Flatten, Reshape
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
UpSampling2D
from tensorflow.keras.datasets import mnist

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

# Normalize pixel values
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Reshape data to match the model input
#x_train = np.expand_dims(x_train, axis=-1)
#x_test = np.expand_dims(x_test, axis=-1)

# Add Gaussian noise
noise_factor=0.5
x_train_noisy=x_train+noise_factor*np.random.normal(loc=0.0,scale=1
.0,
size=x_train.shape)
x_test_noisy=x_test+noise_factor*np.random.normal(loc=0.0,scale=1.0
,
size=x_test.shape)

# Clip values to be between 0 and 1
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)

# Build the autoencoder
input_img = Input(shape=(28, 28, 1))

# Encoder
```

```
x = Conv2D(32, (3, 3), activation='relu',
padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)

# Decoder
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, x)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the model
autoencoder.fit(x_train_noisy, x_train, epochs=10, batch_size=64,
shuffle=True,
validation_data=(x_test_noisy, x_test),verbose=2)

# Denoise test images
x_test_denoised = autoencoder.predict(x_test_noisy)

# Display original, noisy, and denoised images
def display_images(noisy, denoised, n=10):
  #plt.figure(figsize=(20, 4))
  for i in range(n):
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(noisy[i].reshape(28, 28))

    plt.axis('off')
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(denoised[i].reshape(28, 28))
    plt.axis('off')
  plt.show()
```

BERT

```
import tensorflow as tf
from transformers import TFAutoModel, AutoTokenizer
from sklearn.model_selection import train_test_split
import numpy as np

# Step 1: Prepare Dataset
texts = [
    "I love this movie!",
    "This is terrible.",
    "I feel amazing today.",
    "Worst experience ever.",
    "Best product I bought!",
    "I hate this."
]
labels = [1, 0, 1, 0, 1, 0]  # 1=Positive, 0=Negative

# Split data
train_texts, val_texts, train_labels, val_labels =
train_test_split(
    texts, labels, test_size=0.2, random_state=42
)

# Step 2: Initialize Tokenizer
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Tokenize texts
train_encodings = tokenizer(train_texts, truncation=True,
padding=True, return_tensors="tf")
val_encodings = tokenizer(val_texts, truncation=True, padding=True,
return_tensors="tf")

# Step 3: Build Model
bert_base = TFAutoModel.from_pretrained(model_name)

# Input layers
```

```
input_ids = tf.keras.Input(shape=(None,), dtype=tf.int32,
name="input_ids")
attention_mask = tf.keras.Input(shape=(None,), dtype=tf.int32,
name="attention_mask")

# BERT outputs
outputs = bert_base(input_ids, attention_mask=attention_mask)

# Classification head
cls_output = outputs.last_hidden_state[:, 0, :]  # [CLS] token
x = tf.keras.layers.Dense(128, activation='relu')(cls_output)
x = tf.keras.layers.Dropout(0.2)(x)
final_output = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs=[input_ids, attention_mask],
outputs=final_output)

# Step 4: Compile and Train
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=3e-5),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    x={'input_ids': train_encodings['input_ids'],
       'attention_mask': train_encodings['attention_mask']},
    y=np.array(train_labels),
    validation_data=(
        {'input_ids': val_encodings['input_ids'],
         'attention_mask': val_encodings['attention_mask']},
        np.array(val_labels)
    ),
    batch_size=2,
    epochs=3
)

# Step 5: Prediction Function
```

```
def predict_sentiment(text):
    tokens = tokenizer(text, return_tensors="tf", truncation=True,
padding=True)
    prediction = model.predict({'input_ids': tokens['input_ids'],
                                'attention_mask':
tokens['attention_mask']})
    return "Positive" if prediction[0][0] > 0.5 else "Negative"

# Test predictions
print(predict_sentiment("I absolutely loved it!"))  # Should
output: Positive
print(predict_sentiment("This is the worst thing ever."))  # Should
output: Negative
```

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Sample corpus (replace this with your own text data)
corpus = [
    "this is a simple text generation example",
    "we are using lstm model",
    "lstm can remember long-term dependencies",
    "text generation is fun and useful"
]

# Tokenization
tokenizer = Tokenizer()
tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1

# Create input sequences
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

# Pad sequences
max_seq_len = max([len(x) for x in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_seq_len,
padding='pre')

# Create predictors and label
input_sequences = np.array(input_sequences)
X, y = input_sequences[:, :-1], input_sequences[:, -1]
y = tf.keras.utils.to_categorical(y, num_classes=total_words)
```

```
# Define the model
model = Sequential()
model.add(Embedding(total_words, 64, input_length=max_seq_len-1))
model.add(LSTM(100))
model.add(Dense(total_words, activation='softmax'))

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

# Train the model
model.fit(X, y, epochs=100, verbose=1)

# Function to generate text


def generate_text(seed_text, next_words=5):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list],
maxlen=max_seq_len-1, padding='pre')
        predicted = np.argmax(model.predict(token_list, verbose=0),
axis=-1)
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " " + output_word
    return seed_text

# Example
print(generate_text("text generation"))
```

```
!pip install datasets -q
from transformers import MarianMTModel, MarianTokenizer,
TrainingArguments, Trainer
from datasets import Dataset

# English-Hindi translation data as lists
data = {
    "source_text": [
        "Hello, how are you?",
        "What is your name?",
        "I love programming.",
        "Where is the nearest hospital?",
        "Please help me.",
        "Good morning!",
        "Thank you very much.",
        "Do you speak Hindi?",
        "This is a beautiful place.",
        "I am learning machine translation."
    ],
    "target_text": [
        "नमस्ते, आप कैसे हैं?",
        "तुम्हारा नाम क्या है?",
        "मुझे प्रोग्रामिंग पसंद है।",
        "सबसे नजदीकी अस्पताल कहाँ है?",
        "कृपया मेरी मदद कीजिए।",
        "सुप्रभात!",
        "बहुत बहुत धन्यवाद।",
        "क्या आप हिंदी बोलते हैं?",
        "यह एक सुंदर स्थान है।",
        "मैं मशीन अनुवाद सीख रहा हूँ।"
    ]
}

# Convert the data to a HuggingFace Dataset
dataset = Dataset.from_dict(data)

# Load the MarianMT model and tokenizer for English to Hindi
translation
```

```
model_name = "Helsinki-NLP/opus-mt-en-hi"
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)

# Preprocessing function to tokenize the dataset
def preprocess(example):
    inputs = tokenizer(example["source_text"], padding="max_length",
truncation=True, max_length=128)
    targets = tokenizer(example["target_text"], padding="max_length",
truncation=True, max_length=128)
    inputs["labels"] = targets["input_ids"]
    return inputs

# Tokenize the dataset
tokenized = dataset.map(preprocess, batched=True)
training_args = TrainingArguments(
    output_dir="./marian-en-hi-model",
    num_train_epochs=10,
    report_to=["none"],
)
# Define Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized
)

# Fine-tune the model
trainer.train()

import torch
device = "cuda"
model.to(device)

def translate(text):
    batch = tokenizer([text], return_tensors="pt", padding=True,
truncation=True)
```

```
    batch = {k: v.to(device) for k, v in batch.items()}  # Move input
tensors

    gen = model.generate(**batch)
    # Decode and return the translated text
    return tokenizer.decode(gen[0], skip_special_tokens=True)

# Test the translation function
translated_text = translate("Where are you going?")
print(translated_text)
```

OUTPUT :

आप कहाँ जा रहे हैं?