**NEURAL NETWORK ASSIGNMENT**-9

{  
 "nbformat": *4*,  
 "nbformat\_minor": *0*,  
 "metadata": {  
 "colab": {  
 "provenance": []  
 },  
 "kernelspec": {  
 "name": "python3",  
 "display\_name": "Python 3"  
 },  
 "language\_info": {  
 "name": "python"  
 },  
 "accelerator": "GPU",  
 "gpuClass": "standard"  
 },  
 "cells": [  
 {  
 "cell\_type": "code",  
 "execution\_count": *2*,  
 "metadata": {  
 "colab": {  
 "base\_uri": "https://localhost:8080/"  
 },  
 "id": "bmtGYiRvFLiL",  
 "outputId": "a2bb85ed-11c2-4119-8169-3f79292ec45b"  
 },  
 "outputs": [  
 {  
 "output\_type": "stream",  
 "name": "stdout",  
 "text": [  
 "Epoch 1/5*\n*",  
 "235/235 [==============================] - 3s 7ms/step - loss: 0.6962 - val\_loss: 0.6960*\n*",  
 "Epoch 2/5*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6959 - val\_loss: 0.6958*\n*",  
 "Epoch 3/5*\n*",  
 "235/235 [==============================] - 1s 4ms/step - loss: 0.6957 - val\_loss: 0.6956*\n*",  
 "Epoch 4/5*\n*",  
 "235/235 [==============================] - 1s 4ms/step - loss: 0.6955 - val\_loss: 0.6954*\n*",  
 "Epoch 5/5*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6953 - val\_loss: 0.6952*\n*"  
 ]  
 },  
 {  
 "output\_type": "execute\_result",  
 "data": {  
 "text/plain": [  
 "<keras.callbacks.History at 0x7f356646d340>"  
 ]  
 },  
 "metadata": {},  
 "execution\_count": *2* }  
 ],  
 "source": [  
 "from keras.layers import Input, Dense*\n*",  
 "from keras.models import Model*\n*",  
 "*\n*",  
 "# this is the size of our encoded representations*\n*",  
 "encoding\_dim = 32 # 32 floats -> compression of factor 24.5, assuming the input is 784 floats*\n*",  
 "*\n*",  
 "# this is our input placeholder*\n*",  
 "input\_img = Input(shape=(784,))*\n*",  
 "# *\"*encoded*\"* is the encoded representation of the input*\n*",  
 "encoded = Dense(encoding\_dim, activation='relu')(input\_img)*\n*",  
 "# *\"*decoded*\"* is the lossy reconstruction of the input*\n*",  
 "decoded = Dense(784, activation='sigmoid')(encoded)*\n*",  
 "# this model maps an input to its reconstruction*\n*",  
 "autoencoder = Model(input\_img, decoded)*\n*",  
 "# this model maps an input to its encoded representation*\n*",  
 "autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy')*\n*",  
 "from keras.datasets import mnist, fashion\_mnist*\n*",  
 "import numpy as np*\n*",  
 "(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()*\n*",  
 "x\_train = x\_train.astype('float32') / 255.*\n*",  
 "x\_test = x\_test.astype('float32') / 255.*\n*",  
 "x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))*\n*",  
 "x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))*\n*",  
 "*\n*",  
 "autoencoder.fit(x\_train, x\_train,*\n*",  
 " epochs=5,*\n*",  
 " batch\_size=256,*\n*",  
 " shuffle=True,*\n*",  
 " validation\_data=(x\_test, x\_test))*\n*"  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "from keras.layers import Input, Dense*\n*",  
 "from keras.models import Model*\n*",  
 "*\n*",  
 "# Define input shape*\n*",  
 "input\_shape = (784,)*\n*",  
 "*\n*",  
 "# Define encoding dimensions*\n*",  
 "encoding\_dim1 = 64*\n*",  
 "encoding\_dim2 = 32*\n*",  
 "*\n*",  
 "# Define input layer*\n*",  
 "input\_img = Input(shape=input\_shape)*\n*",  
 "*\n*",  
 "encoded1 = Dense(encoding\_dim1, activation='relu')(input\_img)*\n*",  
 "encoded2 = Dense(encoding\_dim2, activation='relu')(encoded1)*\n*",  
 "decoded1 = Dense(encoding\_dim1, activation='relu')(encoded2)*\n*",  
 "decoded2 = Dense(input\_shape[0], activation='sigmoid')(decoded1)*\n*",  
 "autoencoder = Model(input\_img, decoded2)*\n*",  
 "autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy')*\n*",  
 "from keras.datasets import mnist, fashion\_mnist*\n*",  
 "import numpy as np*\n*",  
 "(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()*\n*",  
 "x\_train = x\_train.astype('float32') / 255.*\n*",  
 "x\_test = x\_test.astype('float32') / 255.*\n*",  
 "x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))*\n*",  
 "x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))*\n*",  
 "*\n*",  
 "# Train model*\n*",  
 "history = autoencoder.fit(x\_train, x\_train,*\n*",  
 " epochs=20,*\n*",  
 " batch\_size=256,*\n*",  
 " shuffle=True,*\n*",  
 " validation\_data=(x\_test, x\_test))*\n*",  
 "*\n*",  
 "# Predict on test data*\n*",  
 "decoded\_imgs = autoencoder.predict(x\_test)*\n*",  
 "*\n*",  
 "# Visualize reconstructed image and original image*\n*",  
 "import matplotlib.pyplot as plt*\n*",  
 "*\n*",  
 "# Choose an index of a test image to visualize*\n*",  
 "idx = 10*\n*",  
 "*\n*",  
 "# Reshape the test image*\n*",  
 "test\_img = x\_test[idx].reshape(28, 28)*\n*",  
 "*\n*",  
 "# Reshape the reconstructed image*\n*",  
 "reconstructed\_img = decoded\_imgs[idx].reshape(28, 28)*\n*",  
 "*\n*",  
 "# Plot the original and reconstructed images side by side*\n*",  
 "plt.figure(figsize=(10, 5))*\n*",  
 "plt.subplot(1, 2, 1)*\n*",  
 "plt.imshow(test\_img, cmap='gray')*\n*",  
 "plt.title('Original Image')*\n*",  
 "plt.subplot(1, 2, 2)*\n*",  
 "plt.imshow(reconstructed\_img, cmap='gray')*\n*",  
 "plt.title('Reconstructed Image')*\n*",  
 "plt.show()*\n*"  
 ],  
 "metadata": {  
 "colab": {  
 "base\_uri": "https://localhost:8080/",  
 "height": *1000* },  
 "id": "py3derg1HVE0",  
 "outputId": "5e4f6d63-abd8-4af3-be29-3114490de588"  
 },  
 "execution\_count": *3*,  
 "outputs": [  
 {  
 "output\_type": "stream",  
 "name": "stdout",  
 "text": [  
 "Epoch 1/20*\n*",  
 "235/235 [==============================] - 3s 6ms/step - loss: 0.6930 - val\_loss: 0.6929*\n*",  
 "Epoch 2/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6929 - val\_loss: 0.6928*\n*",  
 "Epoch 3/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6928 - val\_loss: 0.6927*\n*",  
 "Epoch 4/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6927 - val\_loss: 0.6926*\n*",  
 "Epoch 5/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6926 - val\_loss: 0.6925*\n*",  
 "Epoch 6/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6925 - val\_loss: 0.6924*\n*",  
 "Epoch 7/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6924 - val\_loss: 0.6923*\n*",  
 "Epoch 8/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6923 - val\_loss: 0.6922*\n*",  
 "Epoch 9/20*\n*",  
 "235/235 [==============================] - 2s 7ms/step - loss: 0.6922 - val\_loss: 0.6921*\n*",  
 "Epoch 10/20*\n*",  
 "235/235 [==============================] - 1s 6ms/step - loss: 0.6921 - val\_loss: 0.6920*\n*",  
 "Epoch 11/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6920 - val\_loss: 0.6919*\n*",  
 "Epoch 12/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6919 - val\_loss: 0.6918*\n*",  
 "Epoch 13/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6918 - val\_loss: 0.6917*\n*",  
 "Epoch 14/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6917 - val\_loss: 0.6916*\n*",  
 "Epoch 15/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6916 - val\_loss: 0.6915*\n*",  
 "Epoch 16/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6915 - val\_loss: 0.6914*\n*",  
 "Epoch 17/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6914 - val\_loss: 0.6913*\n*",  
 "Epoch 18/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6913 - val\_loss: 0.6912*\n*",  
 "Epoch 19/20*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6912 - val\_loss: 0.6911*\n*",  
 "Epoch 20/20*\n*",  
 "235/235 [==============================] - 2s 7ms/step - loss: 0.6911 - val\_loss: 0.6910*\n*",  
 "313/313 [==============================] - 1s 2ms/step*\n*"  
 ]  
 },  
 {  
 "output\_type": "display\_data",  
 "data": {  
 "text/plain": [  
 "<Figure size 1000x500 with 2 Axes>"  
 ],  
 "image/png": "\n"  
 },  
 "metadata": {}  
 }  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "from keras.layers import Input, Dense\n",  
 "from keras.models import Model\n",  
 "\n",  
 "# this is the size of our encoded representations\n",  
 "encoding\_dim = 32 # 32 floats -> compression of factor 24.5, assuming the input is 784 floats\n",  
 "\n",  
 "# this is our input placeholder\n",  
 "input\_img = Input(shape=(784,))\n",  
 "# \"encoded*\"* is the encoded representation of the input*\n*",  
 "encoded = Dense(encoding\_dim, activation='relu')(input\_img)*\n*",  
 "# *\"*decoded*\"* is the lossy reconstruction of the input*\n*",  
 "decoded = Dense(784, activation='sigmoid')(encoded)*\n*",  
 "# this model maps an input to its reconstruction*\n*",  
 "autoencoder = Model(input\_img, decoded)*\n*",  
 "# this model maps an input to its encoded representation*\n*",  
 "autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy', metrics=['accuracy'])*\n*",  
 "*\n*",  
 "from keras.datasets import fashion\_mnist*\n*",  
 "import numpy as np*\n*",  
 "(x\_train, \_), (x\_test, \_) = fashion\_mnist.load\_data()*\n*",  
 "x\_train = x\_train.astype('float32') / 255.*\n*",  
 "x\_test = x\_test.astype('float32') / 255.*\n*",  
 "x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))*\n*",  
 "x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))*\n*",  
 "*\n*",  
 "#introducing noise*\n*",  
 "noise\_factor = 0.5*\n*",  
 "x\_train\_noisy = x\_train + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_train.shape)*\n*",  
 "x\_test\_noisy = x\_test + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_test.shape)*\n*",  
 "*\n*",  
 "history=autoencoder.fit(x\_train\_noisy, x\_train,*\n*",  
 " epochs=10,*\n*",  
 " batch\_size=256,*\n*",  
 " shuffle=True,*\n*",  
 " validation\_data=(x\_test\_noisy, x\_test\_noisy))*\n*"  
 ],  
 "metadata": {  
 "colab": {  
 "base\_uri": "https://localhost:8080/"  
 },  
 "id": "N200d5zfINuV",  
 "outputId": "07531316-10b6-4c7f-a172-ef8e4eca15b9"  
 },  
 "execution\_count": *10*,  
 "outputs": [  
 {  
 "output\_type": "stream",  
 "name": "stdout",  
 "text": [  
 "Epoch 1/10*\n*",  
 "235/235 [==============================] - 3s 9ms/step - loss: 0.6990 - accuracy: 0.0012 - val\_loss: 0.6989 - val\_accuracy: 6.0000e-04*\n*",  
 "Epoch 2/10*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6987 - accuracy: 0.0013 - val\_loss: 0.6987 - val\_accuracy: 6.0000e-04*\n*",  
 "Epoch 3/10*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6985 - accuracy: 0.0013 - val\_loss: 0.6984 - val\_accuracy: 6.0000e-04*\n*",  
 "Epoch 4/10*\n*",  
 "235/235 [==============================] - 2s 7ms/step - loss: 0.6982 - accuracy: 0.0013 - val\_loss: 0.6982 - val\_accuracy: 6.0000e-04*\n*",  
 "Epoch 5/10*\n*",  
 "235/235 [==============================] - 2s 7ms/step - loss: 0.6980 - accuracy: 0.0014 - val\_loss: 0.6980 - val\_accuracy: 6.0000e-04*\n*",  
 "Epoch 6/10*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6978 - accuracy: 0.0014 - val\_loss: 0.6978 - val\_accuracy: 6.0000e-04*\n*",  
 "Epoch 7/10*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6976 - accuracy: 0.0014 - val\_loss: 0.6976 - val\_accuracy: 6.0000e-04*\n*",  
 "Epoch 8/10*\n*",  
 "235/235 [==============================] - 1s 6ms/step - loss: 0.6974 - accuracy: 0.0014 - val\_loss: 0.6974 - val\_accuracy: 6.0000e-04*\n*",  
 "Epoch 9/10*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6972 - accuracy: 0.0014 - val\_loss: 0.6972 - val\_accuracy: 6.0000e-04*\n*",  
 "Epoch 10/10*\n*",  
 "235/235 [==============================] - 1s 5ms/step - loss: 0.6970 - accuracy: 0.0014 - val\_loss: 0.6970 - val\_accuracy: 5.0000e-04*\n*"  
 ]  
 }  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "import matplotlib.pyplot as plt*\n*",  
 "*\n*",  
 "# Get the reconstructed images*\n*",  
 "reconstructed\_imgs = autoencoder.predict(x\_test\_noisy)*\n*",  
 "*\n*",  
 "# Select one image to display*\n*",  
 "img\_to\_display = 0*\n*",  
 "*\n*",  
 "# Display the original, noisy, and reconstructed images side by side*\n*",  
 "plt.subplot(1, 3, 1)*\n*",  
 "plt.imshow(x\_test[img\_to\_display].reshape(28, 28))*\n*",  
 "plt.title('Original')*\n*",  
 "*\n*",  
 "plt.subplot(1, 3, 2)*\n*",  
 "plt.imshow(x\_test\_noisy[img\_to\_display].reshape(28, 28))*\n*",  
 "plt.title('Noisy')*\n*",  
 "*\n*",  
 "plt.subplot(1, 3, 3)*\n*",  
 "plt.imshow(reconstructed\_imgs[img\_to\_display].reshape(28, 28))*\n*",  
 "plt.title('Reconstructed')*\n*",  
 "*\n*",  
 "plt.show()*\n*"  
 ],  
 "metadata": {  
 "colab": {  
 "base\_uri": "https://localhost:8080/",  
 "height": *246* },  
 "id": "8Xct\_3R4IsuS",  
 "outputId": "4e7bed0f-88bc-4615-f7a8-217aeb14c356"  
 },  
 "execution\_count": *11*,  
 "outputs": [  
 {  
 "output\_type": "stream",  
 "name": "stdout",  
 "text": [  
 "313/313 [==============================] - 1s 2ms/step*\n*"  
 ]  
 },  
 {  
 "output\_type": "display\_data",  
 "data": {  
 "text/plain": [  
 "<Figure size 640x480 with 3 Axes>"  
 ],  
 "image/png": "\n"  
 },  
 "metadata": {}  
 }  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "# Plot the loss and accuracy over epochs\n",  
 "plt.subplot(*2*, 1, 1)\n",  
 "plt.plot(history.history['loss'], label='Training Loss')\n",  
 "plt.plot(history.history['val\_loss'], label='Validation Loss')\n",  
 "plt.legend()\n",  
 "\n",  
 "plt.subplot(2, 1, 2)\n",  
 "plt.plot(history.history['accuracy'], label='Training Accuracy')\n",  
 "plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')\n",  
 "plt.legend()\n",  
 "\n",  
 "plt.show()\n"  
 ],  
 "metadata": {  
 "colab": {  
 "base\_uri": "https://localhost:8080/",  
 "height": *430* },  
 "id": "fG-971EPJS0Y",  
 "outputId": "ffa37052-ee0c-4c6d-cec7-b17df9c80798"  
 },  
 "execution\_count": *12*,  
 "outputs": [  
 {  
 "output\_type": "display\_data",  
 "data": {  
 "text/plain": [  
 "<Figure size 640x480 with 2 Axes>"  
 ],  
 "image/png": "\n"  
 },  
 "metadata": {}  
 }  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "source": [],  
 "metadata": {  
 "id": "uKnOospHJW*4*L"  
 },  
 "execution\_count": null,  
 "outputs": []  
 }  
 ]  
}