Paschal UGWU (BSc Biochemistry)

+2348166207095 | ugwupaschal@gmail.com

August 2022 Biological Data Science Workshop Practice Report.

I was opportune to be selected for the Biological Data Science Workshop 2022. Drexel University, Rowan University, and the University of Chicago organized this workshop in August 2022. It was from it that I learned how to apply deep learning to biology. The workshop organizers received permission to use the datasets from the authors of the research works used. They also granted licenses to all attendees (me inclusive).

Having undergone this training, I decided to perform the tasks assigned to participants for my skill development. This training has exposed me to biopython and the application of Machine Learning to biological data. The report of my tasks is on the following page. It includes the codes I ran as well as the outputs I got. This report is explanatory so that any scientist can reciprocate with their datasets. All comments have "#" attached to them.

The following are ways I intend to apply my python and machine learning skills to marine biotechnology:

1. As a national resource for molecular biology information, I will use NCBI in Marine Biotechnology. My skills in using NCBI will be helpful for me to develop new information technologies to aid in understanding fundamental molecular and genetic processes that control health and diseases (especially malaria).
2. I will apply my Deep Learning skills in biotechnology, especially in the field of genotyping, toxicity prediction and evaluation, and the prediction of molecular properties of unknown substances.
3. I will also apply my Convoluted Neural Networks (CNN) skills in identifying cell types from phase contrast microscope images without molecular labeling. The success of this approach is because each cell shows a characteristic morphology. This image classification will help me solve several medical issues as a Marine Biotechnology.
4. If admitted to study Marine Biotechnology, I will apply my autoencoding skills in transcriptome profiling, single-cell RNA-Seq data analysis, and pathway studies of biological activities.

Overall, I will gladly tutor my fellow cohorts that are interested in learning the skills of biopython and Machine Learning with python. This tutorial will help us analyze datasets, complete complex tasks and do well in our thesis.

There are four sections in this report:

1. SECTION 1: The use of biopython for translation, transcription, NCBI search and NCBI BLAST.
2. SECTION 2: To demonstrate the application of deep learning in protein research.
3. SECTION 3: To demonstrate the use of Convoluted Neural Network (CNN).
4. SECTION 4: To demonstrate the use of autoencoders.

SECTION 1: The use of biopython for translation, transcription, NCBI search and NCBI BLAST.

```
# Let us install biopython
!pip install biopython
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting biopython
  Downloading biopython-1.80-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 19.2 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from biopython) (1.21.6)
Installing collected packages: biopython
Successfully installed biopython-1.80
```

```
# How to translate to a protein
from Bio.Seq import Seq
coding_dna =
Seq("ATATCCCGGTAAACCTACCGGGAATTAAGTCTATGACCTGAAGTCTAGCGTCAGTCGCCCGTGACTGCCACCTGC")
#To return reverse complement as a DNA sequence
print(coding_dna.reverse_complement())
#To transcribe a DNA sequence to an RNA sequence
print(coding_dna.transcribe())
#To convert a nucleotide sequence to a protein
print(coding_dna.translate())

# COX1_Search from NCBI after establishing a search query from NCBI (using the method
of refining search)
# Commands to get ready for search from Entrez
from Bio import Entrez
from Bio import SeqIO
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
Entrez.email= "ugwupaschal@gmail.com"
# Lets now search, setting the database(db) to nucleotide and copying our search
query from NCBI
all_records=[]
# In search terms, remember to add backslashes in front of any quote, to avoid
getting an error also use the retmax fuction to set maximum number to return, else it
would return 20 by default. In this case, we set our maximum value to 100000
handle1=Entrez.esearch(db="nucleotide", term="(COX1 or COXI) AND fungi [ORGN] AND
450:30000 [SLEN] NOT \"whole genome\" NOT \"complete genome\" NOT \"partial\" NOT
intron", retmax=100000)
# Lets check our store of records, for the list of searches returned by the NCBI
search query
# The IdList must be in a single quote and square bracket
records=Entrez.read(handle1)
print(len(records['IdList']))
```

```
85
```

```
# Lets know the actual record numbers. Each of these numbers corresconds to a particular genebank record
recordnums=records['IdList']
print(recordnums)
```

```
['2323990138', '2259093070', '2234646771', '815889166', '2163350252', '2192911159', '2192911141',
```

```
# Let us now get the actual record (we use the efetch function), rettype and retmode mean return type and return mode respectively
# Remember to close both handles that we used
handle=Entrez.efetch(db="nucleotide", id=recordnums, rettype='gb', retmode='xml')
my_genbank_records=Entrez.read(handle)
handle.close
handle1.close
```

```
<bound method HTTPResponse.close of <http.client.HTTPResponse object at 0x7f6efdef9c10>>
```

```
# Let us print our gene bank records and the organism field.
# We use [0] to get the first record of the field
print("My first fungi: ",my_genbank_records[0]['GBSeq_organism'],"and its accession number is: ",my_genbank_records[0]['GBSeq_primary-accession'])
```

```
My first fungi:  Tricharina praecox and its accession number is:  XM_051484780
```

```
# Let us list all our genebank records
list(my_genbank_records)
```

```
[{'GBSeq_locus': 'XM_051484780', 'GBSeq_length': '1488', 'GBSeq_strandedness': 'single', 'GBSeq_moltype': 'mRNA', 'GBSeq_topology': 'linear', 'GBSeq_division': 'PLN', 'GBSeq_update-date': '01-NOV-2022', 'GBSeq_create-date': '01-NOV-2022', 'GBSeq_definition': 'Tricharina praecox putative COX1 assembly protein Shy1 (BZA05DRAFT_409286), mRNA', 'GBSeq_primary-accession': 'XM_051484780', 'GBSeq_accession-version': 'XM_051484780.1', 'GBSeq_other-seqids': ['ref|XM_051484780.1|', 'gi|2323990138'], 'GBSeq_project': 'PRJNA895876', 'GBSeq_keywords': ['RefSeq'], 'GBSeq_source': 'Tricharina praecox', 'GBSeq_organism': 'Tricharina praecox', 'GBSeq_taxonomy': 'Eukaryota; Fungi; Dikarya; Ascomycota; Pezizomycotina; Pezizomycetes; Pezizales; Pyronemataceae; Tricharina', 'GBSeq_references': [{'GBReference_reference': '1', 'GBReference_position': '1..1488', 'GBReference_authors': ['Steindorff,A.S.', 'Seong,K.', 'Carver,A.', 'Calhoun,S.', 'Fischer,M.S.', 'Stillman,K.', 'Liu,H.', 'Drula,E.', 'Henrissat,B.', 'Simpson,H.J.', 'Schilling,J.S.', 'Lipzen,A.', 'He,G.', 'Yan,M.', 'Andreopoulos,B.', 'Pangilinan,J.', 'LaButti,K.', 'Ng,V.', 'Traxler,M.', 'Bruns,T.D.', 'Grigoriev,I.V.'], 'GBReference_title': 'Diversity of genomic adaptations to post-fire environment in Pezizales fungi points to a crosstalk between charcoal tolerance and sexual development', 'GBReference_journal': 'New Phytol (2022) In press', 'GBReference_xref': [{'GBXref_dbname': 'doi', 'GBXref_id': '10.1111/nph.18407'}], 'GBReference_pubmed': '35898177',
```

```
# We can not also check the various parameters that can be extracted from our genebank records using the following command:
list(my_genbank_records[0])
```

```
['GBSeq_locus',
 'GBSeq_length',
 'GBSeq_strandedness',
 'GBSeq_moltype',
 'GBSeq_topology',
 'GBSeq_division',
 'GBSeq_update-date',
 'GBSeq_create-date',
 'GBSeq_definition',
 'GBSeq_primary-accession',
 'GBSeq_accession-version',
 'GBSeq_other-seqids',
 'GBSeq_project',
 'GBSeq_keywords',
 'GBSeq_source',
 'GBSeq_organism',
 'GBSeq_taxonomy',
 'GBSeq_references',
 'GBSeq_comment',
 'GBSeq_feature-table',
 'GBSeq_sequence',
 'GBSeq_xrefs']
```

```
# Now let us print the locus, length, strandednes, topology, division , source
taxonomy, and the gene features of the first organism
print(my_genbank_records[0]['GBSeq_locus'])
print(my_genbank_records[0]['GBSeq_length'])
print(my_genbank_records[0]['GBSeq_strandedness'])
print(my_genbank_records[0]['GBSeq_topology'])
print(my_genbank_records[0]['GBSeq_division'])
print(my_genbank_records[0]['GBSeq_source'])
print(my_genbank_records[0]['GBSeq_taxonomy'])
print(my_genbank_records[0]['GBSeq_feature-table'])
```

```
XM_051484780
1488
single
linear
PLN
Tricharina praecox
Eukaryota; Fungi; Dikarya; Ascomycota; Pezizomycotina; Pezizomycetes; Pezizales; Pyronemataceae; Tricharina
[{'GBFeature_key': 'source', 'GBFeature_location': '1..1488', 'GBFeature_intervals': [{'GBInterval_from': '1', '
```

```
# To check the length of our fasta records
my_fasta_records=[]
print(len(my_genbank_records))
for i in range(len(my_genbank_records)):

my_fasta_records.append(SeqRecord(Seq(my_genbank_records[i]['GBSeq_sequence']),id=my_
```

```
genbank_records[i]['GBSeq_primary-
accession'],description=my_genbank_records[i]['GBSeq_definition']))
```

```
85
```

```
# To get our output files in both genebank and fasta formats
#1 In fasta format
one_file=open("my_seqs.fa","w")
SeqIO.write(my_fasta_records,one_file,"fasta");
one_file.close
```

```
<function TextIOWrapper.close()>
```

```
#2 In genebank format
one_file=open("my_seqs.gb","w")
handle=Entrez.efetch(db="nucleotide",id=recordnums,rettype="gbwithparts",retmode="tex
t")
my_genbank_records=SeqIO.parse(handle,"genbank")
SeqIO.write(my_genbank_records,one_file,"gb");
one_file.close
```

```
<function TextIOWrapper.close()>
```

```
#NOTE: In google colab, both the fasta file and genebank file would be saved
temporarily in the lacal files environment at the top left part of the computer
screen. They can then be downloaded and viewed.
#################### LET US NOW GO INTO NCBI BLAST ######################
# Install NCBI BLAST
!apt install ncbi-blast+
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libnvidia-common-510
Use 'apt autoremove' to remove it.
The following additional packages will be installed:
  ncbi-data
The following NEW packages will be installed:
  ncbi-blast+ ncbi-data
0 upgraded, 2 newly installed, 0 to remove and 27 not upgraded.
Need to get 14.6 MB of archives.
After this operation, 74.2 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 ncbi-data all 6.1.20170106+dfsg1-8 [3,518 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal/universe amd64 ncbi-blast+ amd64 2.9.0-2 [11.1 MB]
Fetched 14.6 MB in 3s (4,385 kB/s)
```

```
!curl \
ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/un
iprot_sprot.fasta.gz --output uniprot_sprot.fasta.gz
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 87.1M  100 87.1M    0     0   460k      0  0:03:13  0:03:13 --:--:--  207k
```

```
!gunzip -k uniprot_sprot.fasta.gz
# We now use makeblastdb tool to input our file in fasta format in a protein database for our blasting
!makeblastdb -in uniprot_sprot.fasta \
-dbtype prot \
-out uniprot_sprot_r2018_08
```

```
Building a new DB, current time: 02/09/2023 10:26:17
New DB name:   /content/uniprot_sprot_r2018_08
New DB title:  uniprot_sprot.fasta
Sequence type: Protein
Keep MBits: T
Maximum file size: 1000000000B
Adding sequences from FASTA; added 568744 sequences in 16.4451 seconds.
```

```
# Getting file from url to local location for blasting
!wget https://eagle.fish.washington.edu/cnidarian/Ab_4denovo_CLC6_a.fa
```

```
--2023-02-09 10:26:46--  https://eagle.fish.washington.edu/cnidarian/Ab_4denovo_CLC6_a.fa
Resolving eagle.fish.washington.edu (eagle.fish.washington.edu)... 128.95.149.81
Connecting to eagle.fish.washington.edu (eagle.fish.washington.edu)|128.95.149.81|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2030182 (1.9M)
Saving to: ‘Ab_4denovo_CLC6_a.fa’

Ab_4denovo_CLC6_a.f 100%[===================>]   1.94M   583KB/s    in 3.4s

2023-02-09 10:26:50 (583 KB/s) - ‘Ab_4denovo_CLC6_a.fa’ saved [2030182/2030182]
```

```
# How many sequences? Lets count ">" as we know each contig has 1
!grep -c ">" Ab_4denovo_CLC6_a.fa
!head -20 Ab_4denovo_CLC6_a.fa
```

```
5490
>solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_1
ACACCCCACCCCAACGCACCCTCACCCCCACCCCAACAATCCATGATTGAATACTTCATC
TATCCAAGACAAACTCCTCCTACAATCCATGATAGAATTCCTCCAAAAATAATTTCACAC
TGAAACTCCGGTATCCGAGTTATTTTGTTCCCAGTAAAATGGCATCAACAAAAGTAGGTC
TGGATTAACGAACCAATGTTGCTGCGTAATATCCCATTGACATATCTTGTCGATTCCTAC
CAGGATCCGGACTGACGAGATTTCACTGTACGTTTATGCAAGTCATTTCCATATATAAAA
TTGGATCTTATTTGCACAGTTAAATGTCTCTATGCTTATTTATAAATCAATGCCCGTAAG
CTCCTAATATTTCTCTTTTCGTCCGACGAGCAAACAGTGAGTTTACTGTGGCCTTCAGCA
AAAGTATTGATGTTGTAAATCTCAGTTGTGATTGAACAATTTGCCTCACTAGAAGTAGCC
TTC
>solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_2
ATTAGAGGCCTTGGGGTTGAAATATCTGACAGCAACAACCGACACAAATGCACCTGGGGT
CTTCCTAACTATCAAGGCAAAATCTGAAACTGGTAAATTTGGTATATATTCCCACTTTCT
CTCTCTGAATTAACCTCCAAACATACCTGACACAAGAAACGTCTAAAACGATCTGCCATG
TTGATGTGTGTGACTGCTTCATATATATTTAGATTAAGATACATATAGTAATATTCAAGA
ACGTTTGTC
>solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_3
TCCCGAGATAAGCACCTGCTCCTGTGGAAGCTGACACGTAATGAGACCAACTACGGCATC
CCATACAAGCAGCTCCATGGCCACAACCACTTTGTGTCTGATGTTGTGCTCTCCTCAGAT
GGCCAATTTGCTCTGTCTGCATCATGGGATGGATCCCTCAGACTCTGGGATCTTGTAACA
```

```
!blastx \
-query Ab_4denovo_CLC6_a.fa \
-db uniprot_sprot_r2018_08 \
-out Ab_4-uniprot_blastx.tab \
-evalue 1E-20 \
-num_threads 4 \
-max_target_seqs 1 \
-outfmt 6
```

```
#We us outfmt 6 in order to have our output in tabular format
```

```
Warning: [blastx] Examining 5 or more matches is recommended
Warning: [blastx] Number of threads was reduced to 2 to match the number of available CPUs
^C
```

```
!head -10 uniprot_sprot.fasta
```

```
>sp|Q6GZX4|001R_FRG3G Putative transcription factor 001R OS=Frog virus 3 (isolate Goorha) OX=654924 GN=FV3-001R PE=4 SV=1
MAFSAEDVLKEYDRRRRMEALLLSLYYPNDRKLLDYKEWSPPRVQVECPKAPVEWNNPPS
EKGLIVGHFSGIKYKGEKAQASEVDVNKMCCWVSKFKDAMRRYQGIQTCKIPGKVLSDLD
AKIKAYNLTVEGVEGFVRYSRVTKQHVAAFLKELRHSKQYENVNLIHYILTDKRVDIQHL
EKDLVKDFKALVESAHRMRQGHMINVKYILYQLLKKHGHGPDGPDILTVKTGSKGVLYDD
SFRKIYTDLGWKFTPL
>sp|Q6GZX3|002L_FRG3G Uncharacterized protein 002L OS=Frog virus 3 (isolate Goorha) OX=654924 GN=FV3-002L PE=4 SV=1
MSIIGATRLQNDKSDTYSAGPCYAGGCSAFTPRGTCGKDWDLGEQTCASGFCTSQPLCAR
IKKTQVCGLRYSSKGKDPLVSAEWDSRGAPYVRCTYDADLIDTQAQVDQFVSMFGESPSL
AERYCMRGVKNTAGELVSRVSSDADPAGGWCRKWYSAHRGPDQDAALGSFCIKNPGAADC
```

```
!wc -l Ab_4-uniprot_blastx.tab
!cat Ab_4-uniprot_blastx.tab
```

```
50 Ab_4-uniprot_blastx.tab
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_3	sp|O42248|GBLP_DANRE	82.456	171	30	0	1	513	35	205	2.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_5	sp|Q08013|SSRG_RAT	75.385	65	16	0	3	197	121	185	1.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_6	sp|P12234|MPCP_BOVIN	76.623	77	18	0	2	232	286	362	7.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_9	sp|Q41629|ADT1_WHEAT	82.258	62	11	0	3	188	170	231	6.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_13	sp|Q32NG4|GALD1_XENLA	54.444	90	40	1	1	270	140	228	1.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_23	sp|Q9GNE2|RL23_AEDAE	97.222	72	2	0	67	282	14	85	6.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_31	sp|B3EWZ9|HEPHL_ACRMI	56.589	129	53	1	2	379	26	154	1.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_31	sp|B3EWZ9|HEPHL_ACRMI	44.715	123	64	1	8	364	380	502	9.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_31	sp|B3EWZ9|HEPHL_ACRMI	44.531	128	65	3	11	376	732	859	1.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_32	sp|Q641Y2|NDUS2_RAT	88.034	117	14	0	2	352	334	450	6.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_37	sp|Q9D3D9|ATPD_MOUSE	56.098	123	54	0	2	370	46	168	4.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_39	sp|Q39613|CYPH_CATRO	75.000	120	23	1	55	393	1	120	3.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_40	sp|O09167|RL21_MOUSE	69.630	135	41	0	17	421	1	135	8.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_41	sp|P40429|RL13A_HUMAN	70.588	153	45	0	3	461	16	168	2.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_48	sp|Q9MZ15|VDAC2_PIG	74.684	79	20	0	1	237	150	228	7.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_52	sp|Q4PMB3|RS4_IXOSC	77.612	67	15	0	3	203	72	138	2.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_57	sp|P00829|ATPB_BOVIN	90.838	382	35	0	3	1148	86	467	0.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_59	sp|P62250|RS16_RAT	89.041	73	8	0	61	279	3	75	3.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_66	sp|Q4PMB3|RS4_IXOSC	79.104	67	14	0	24	224	3	69	1.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_68	sp|P25457|RL7B_SCHPO	65.500	200	68	1	1	597	47	246	7.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_70	sp|P56597|NDK5_HUMAN	67.059	85	27	1	1	252	127	211	9.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_73	sp|P27085|RS26_OCTVU	97.561	41	1	0	2	124	12	52	2.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_77	sp|Q9DFR6|RS13_GILMI	94.253	87	5	0	3	263	40	126	5.
solid0078_20110412_FRAG_BC_WHITE_WHITE_F3_QV_SE_trimmed_contig_85	sp|Q71V39|EF1A2_RABIT	90.780	141	13	0	1	423	56	196	6.
```

```
# Biopython_16S_rRNA
1. How many results do you get when you search for full 16s rRNA genes of Streptococcus manfredo?
2. Save in .fa

# Install biopython
!pip install biopython
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting biopython
  Downloading biopython-1.79-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl (2.3 MB)
     |████████████████████████████████| 2.3 MB 5.1 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from biopython) (1.21.6)
Installing collected packages: biopython
Successfully installed biopython-1.79
```

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

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

```
# Import Entrez, Seq input/output, Sequence, and bio.alphabet modules
from Bio import Entrez
from Bio import SeqIO
from Bio import Seq

# Can put any email address below
Entrez.email = "ugwupaschal@gmail.com"
handle = Entrez.esearch(db="nucleotide", term="16S rRNA[gene] AND streptococcus[ORGN]
AND Manfredo AND genome")  # search sequences by a combination of keywords
records = Entrez.read(handle)  #store records from search
print(records['Count'])  #This prints how many results there are from your search
```

```
1
```

```
#This retrieves the Genbank record for the top result
handle = Entrez.efetch(db="nucleotide", id=records['IdList'][0], rettype="gb",
retmode="text")
record = SeqIO.read(handle, "genbank")
handle.close()
#Initialize variables
sixteen_s=[]
seqs=[]
locations=[]
print(record); print("-----------")
print(record.annotations["taxonomy"])
```

```
ID: AM295007.1
Name: AM295007
Description: Streptococcus pyogenes Manfredo complete genome
Database cross-references: BioProject:PRJNA270, BioSample:SAMEA1705956
Number of features: 7268
/molecule_type=DNA
/topology=circular
/data_file_division=BCT
/date=06-FEB-2015
/accessions=['AM295007']
/sequence_version=1
/keywords=['complete genome']
/source=Streptococcus pyogenes str. Manfredo
/organism=Streptococcus pyogenes str. Manfredo
/taxonomy=['Bacteria', 'Firmicutes', 'Bacilli', 'Lactobacillales', 'Streptococcaceae', 'Streptococcus']
/references=[Reference(title='Complete genome of acute rheumatic fever-associated serotype M5 Streptococ
Seq('TTGTTGATATTCTGTTTTTTCTTTTTTAGTTTTCCACATAAAAAATAGTTGAAA...AGC')
-----------
['Bacteria', 'Firmicutes', 'Bacilli', 'Lactobacillales', 'Streptococcaceae', 'Streptococcus']
```

```
#This goes through each feature of a genbank record (features are listed on the left of a Genbank record)
for feature in record.features:
  if feature.type=='gene' or feature.type == 'rRNA':  #If the feature is a Gene or rRNA then
    if 'gene' in feature.qualifiers:  #This looks to see if /gene= exists in the second column
      if feature.qualifiers['gene'][0]=='16S rRNA':  #If the first occurrence of gene is /gene="16S rRNA"
        if str(feature.location) not in locations:  #If the feature location is not already in the locations list
          print(feature.location)
          locations.append(str(feature.location))  #append the location to the locations list
          print(locations)
          sixteen_s.append(feature)  # append the feature itself to a list of 16S features
          seqs.append(feature.extract(record.seq))  #We can also extract just the sequences
print(len(sixteen_s))
```

```
[17042:18549](+)
['[17042:18549](+)']
[23043:24550](+)
['[17042:18549](+)', '[23043:24550](+)']
[81667:83174](+)
['[17042:18549](+)', '[23043:24550](+)', '[81667:83174](+)']
[259013:260520](+)
['[17042:18549](+)', '[23043:24550](+)', '[81667:83174](+)', '[259013:260520](+)']
[504767:506274](+)
['[17042:18549](+)', '[23043:24550](+)', '[81667:83174](+)', '[259013:260520](+)', '[504767:506274](+)']
[1573011:1574518](-)
['[17042:18549](+)', '[23043:24550](+)', '[81667:83174](+)', '[259013:260520](+)', '[504767:506274](+)', '[1573011:1574518](-)']
6
```

```
#output_handle=open("/content/drive/My Drive/Colab Notebooks/workshop/rRNAs.fa","w")
output_handle=open("rRNAs.fa","w")
#SeqIO.write(final, output_handle, "fasta")
for i in range(len(seqs)):
  output_handle.write(">%s %s %s\n%s\n" %
(record.id,record.description,sixteen_s[i].location,str(seqs[i])))  #This outputs the
record ID, description, location of the sequence and sequence itself to a file
output_handle.close()

# Biopython-Saving-ATPase-gene-IDs-from-Galdieria-sulphuraria-to-a-file
1. Save all the Protein IDs of ATPase genes from the first Galdieria sulphuraria
whole genome scaffold that you find.
2. Save them into a file called G_sulphuraria_atpase_ids.

# We can mount our work on google drive
from google.colab import drive
drive.mount('/content/drive')
```

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

```
## Call appropriate modules to run BioPython and retreive NCBI data.
from Bio import Entrez
from Bio import SeqIO
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord

# Bypass NCBI email address input requirement.
Entrez.email = 'ugwupaschal@gmail.com'
#Get the G. suphuraria whole genome
handle = Entrez.esearch(db="nucleotide", term="Galdieria sulphuraria[ORGN] AND ATPase
AND scaf_4")
records = Entrez.read(handle)
handle.close()
print(records['Count'])
```

```
2
```

```
#Get first record
print(records['IdList'][0])
handle = Entrez.efetch(db="nucleotide", id=records['IdList'][0], rettype="gb",
retmode="text")
record = SeqIO.read(handle, "genbank")
handle.close()
```

```
452824956
```

```
# Look for Coding regions and look to see if ATPase is in the product
ATPase_list = []
for feature in record.features:
    if feature.type == 'CDS':
        if "ATPase" in feature.qualifiers["product"][0]:
            print(feature.qualifiers["product"])
            ATPase_list.append(feature.qualifiers["protein_id"][0])
            print(feature.qualifiers["protein_id"][0])
```

```
['archaeal ATPase']
EME31956.1
['archaeal ATPase isoform 2']
EME31957.1
['archaeal ATPase isoform 1']
EME31958.1
['ATPase with chaperone activity']
EME31974.1
['phospholipid-translocating P-type ATPase']
EME32026.1
['ATPase']
EME32062.1
['arsenite-translocating ATPase, ArsA family']
EME32117.1
```

```
#Output IDs to a file
output_handle = open("/content/drive/My Drive/Colab Notebooks/Galdieria sulphuraria_atpase_ids","w")
output_handle.write("\n".join(ATPase_list))
output_handle.close()
```

```
# Lets answer the following questions:
1. How many results do you get when you search for full 28 rRNA genes that are over 700 bp from a) nematode, b) a green algae species, and c) an ascomycete fungus? (Give a different number for a, b, and c). Please use python to answer this - Hint: use the [gene] tag and print (records['Count'])
2. Save all 28 rRNA genes that are over 700 bp from all green algae species to a file named "long_28rrna_greenalgae.fa"
# Install Biopython
# We are installing biopython version 1.68 because the latest version has removed Bio.Alphabet
!pip install biopython==1.68
from Bio import Entrez
from Bio import SeqIO
from Bio.Seq import Seq
```

```
from Bio.SeqRecord import SeqRecord
from Bio.Alphabet import generic_dna, generic_protein
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: biopython==1.68 in /usr/local/lib/python3.8/dist-packages (1.68)
```

```
Entrez.email = 'ugwupaschal@gmail.com'
# [SLEN] means Sequence Length (in this case between 700 to 2000)
handle = Entrez.esearch(db="nucleotide", term="28S rRNA[gene] AND nematoda[ORGN] AND
700:2000[SLEN]")
records = Entrez.read(handle)
handle.close()
print("Found ",records['Count']," records");

handle = Entrez.esearch(db="nucleotide", term="28S rRNA[gene] AND green algae[ORGN]
AND 700:2000[SLEN]")
green_algae_records = Entrez.read(handle)
handle.close()
print("Found ",green_algae_records['Count']," records");

handle = Entrez.esearch(db="nucleotide", term="28S rRNA[gene] AND ascomycota[ORGN]
AND 700:2000[SLEN]")
records = Entrez.read(handle)
handle.close()
print("Found ",records['Count']," records");
```

```
Found  357  records
Found  154  records
Found  1230  records
```

```
#print(len(green_algae_records['IdList']))
handle = Entrez.esearch(db="nucleotide", term="28S rRNA[gene] AND green algae[ORGN]
AND 700:2000[SLEN]",retmax=green_algae_records['Count'])
green_algae_records = Entrez.read(handle)
handle.close()
ID_list = ','.join(green_algae_records['IdList'])
handle = Entrez.efetch(db="nucleotide", id=ID_list, rettype="gb", retmode="xml")
genbank_records = Entrez.read(handle)
fasta_records = []
for i in range(len(genbank_records)):
fasta_records.append(SeqRecord(Seq(genbank_records[i]['GBSeq_sequence']),id=genbank_r
ecords[i]['GBSeq_primary-
accession'],description=genbank_records[i]['GBSeq_definition']))
long_file = open("long_28rrna_greenalgae.fasta","w")
SeqIO.write(fasta_records, long_file, "fasta")
long_file.close()
```

SECTION 2: To demonstrate the application of deep learning in protein research.

```
# To pip install the esm code, fetch the fasta file and the pre-computed embeddings.
!pip install git+https://github.com/facebookresearch/esm.git
!curl -LO
https://github.com/tcoard/Deep_Learning_For_Proteins/raw/master/coala40_reprs.tar.gz
!tar -xzf coala40_reprs.tar.gz
!curl -LO
https://github.com/tcoard/Deep_Learning_For_Proteins/raw/master/prot_bert_embeddings.
tar.gz
!tar -xzf prot_bert_embeddings.tar.gz
!curl -LO https://transfer.sh/tq5EsK/model.h5
!curl -LO
https://github.com/tcoard/Deep_Learning_For_Proteins/raw/master/reformatted_coala40.f
a
!pwd
!ls
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting git+https://github.com/facebookresearch/esm.git
  Cloning https://github.com/facebookresearch/esm.git to /tmp/pip-req-build-riudl5g3
  Running command git clone -q https://github.com/facebookresearch/esm.git /tmp/pip-req-build-riudl
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
Building wheels for collected packages: fair-esm
  Building wheel for fair-esm (PEP 517) ... done
  Created wheel for fair-esm: filename=fair_esm-0.5.0-py3-none-any.whl size=69640 sha256=85a407733
  Stored in directory: /tmp/pip-ephem-wheel-cache-5vbk7ktf/wheels/5c/8e/11/307eca5379b418a9989ef4d
Successfully built fair-esm
Installing collected packages: fair-esm
Successfully installed fair-esm-0.5.0
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 23.5M  100 23.5M    0     0  30.9M      0 --:--:-- --:--:-- --:--:-- 30.9M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 18.8M  100 18.8M    0     0  35.8M      0 --:--:-- --:--:-- --:--:-- 35.8M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  230k  100  230k    0     0   158k      0  0:00:01  0:00:01 --:--:--  158k
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
```

```
#To import needed statement
import random
import torch
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import esm
```

```
import scipy

from collections import Counter
from tqdm import tqdm
from tensorflow.keras import models, layers, regularizers, optimizers
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, SVR
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report

## Embedding Configuration
COALA Dataset: https://www.biorxiv.org/content/10.1101/2020.04.17.047316v1.abstract
ESM (source paper): https://www.pnas.org/doi/10.1073/pnas.2016239118
PROTBERT: https://www.biorxiv.org/content/10.1101/2020.07.12.199554v3
FASTA_PATH = "./reformatted_coala40.fa" #@param
EMBEDDING_METHOD = "PROTBERT" #@param ["ESM", "PROTBERT"]
if EMBEDDING_METHOD == "ESM":
    EMB_PATH = "./coala40_reprs/"
elif EMBEDDING_METHOD == "PROTBERT":
    EMB_PATH = "./prot_bert_embeddings"
EMB_LAYER = 34

# To show what the FASTA Looks like:
!cat reformatted_coala40.fa | head -10
```

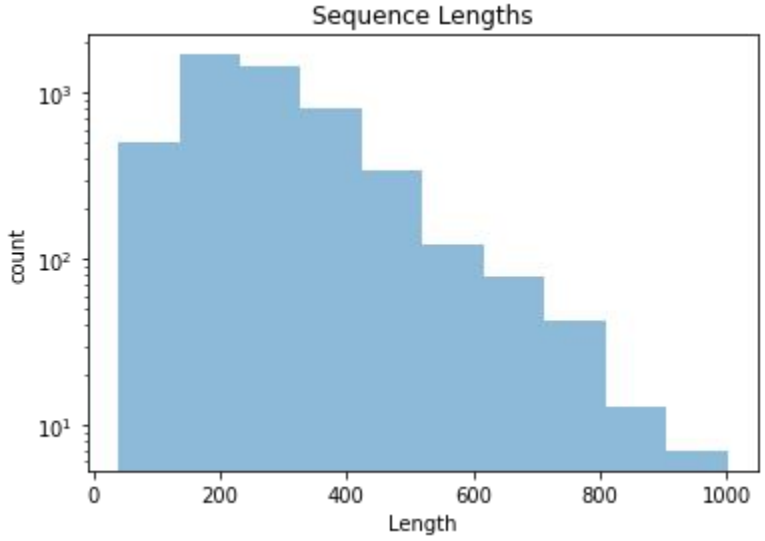
```
>EBP77072.1|AFAM001-1|BETA-LACTAM
MCGRRFRLSAAWLRWRAWSLPCAAERLHRWSRPENRSSQSGTLWLERQRDARKAGNMPLNLRPRRTLACLAAFAVSACVGTVVPASVPPQTRSPTVSPRPVPAPAPAQREDRKPLTLPASLDPGFARPPPELNESINALWKAFPG
>WP_122865681.1|AFAM006-1|BETA-LACTAM
MGEMVFMKQKTIMRWGVLVLSVLVFSIAAWEISENRSSEKELKAEATPQINSVNERLEVLENKYHAKLGVYAANGNEKVKYNAQERFAYTSTYKAIISGLLLKNRTEEELQKKIFFSKEDLVDYSPITEKFVDKGMTLKAIIHAA
>WP_120169520.1|AFAM019-1|BETA-LACTAM
MRAPTIPPRSIRTRLTAAASAIILSVSTACSASVDSTTANPTPQPSNPSQITDTVSPSSPPASDASASSSDELSKDLVAALRNLETRYDARIGVFAKEGLDGKSFAWRADERFAYASTYKALAAGALLDKFGTDILDTTVQIPEG
>WP_109415184.1|AFAM040-1|BETA-LACTAM
MTRLFFTLLSLLATAQIFAQTDLKQRVEQIISTKKADIGVSIQSSGTGETMDIRGNQHYPMISIFKFHIALAVLNKVDNKQLSLKQKIFVRKDELLEDTWSPFREKYPKGDIQITLEEALRWTVSHSDNNLGDLLIRLAGGVRAI
>AJI55001.1|Bla3-9|BETA-LACTAM
MNKIIILILTIFISSIGFSTDAANIEISNRLHNLEKLYGGKIGVYTINRNNNSNFSHNQSFYFPICSTYKFLVVGAILKQSMTDNNLLDKEVKISANQIIGYSPVTKKHINQTMTVSELSKAAIQSDNTATNLLIEKLGGLNNLN
```

```
AA_LIST = ('X', 'A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'P',
           'Q', 'R', 'S', 'T', 'V', 'W', 'Y')

# tokenizes a list of sequences into numerical forms (numbers from 0 to 20)
def tokenize_sequences(sequences, SeqCol='ISM', seqlen=600):
    # limits the length of sequences to a fixed value
    def fix_size(x):
        if len(x) < seqlen:
            return x + 'X'*(seqlen-len(x))
```

```
        elif len(x) > seqlen:
            return x[:seqlen]
        else:
            return x

    # fix length of each sequence, and place them into a numpy array
    data = np.vstack([list(fix_size(sequence)) for sequence in sequences])

    # tokenize (i.e. assign a numerical value) each amino acid
    aa_tokenizer = {AA_LIST[k]:k for k in range(len(AA_LIST))}

    # apply this to the entire data
    return np.vectorize(aa_tokenizer.get)(data)

# ************************************************
labels = [] # the drug resistances
sequences = [] # the raw sequences
raw_embeddings = [] # the raw (pretrained) embeddings
# ************************************************

for header, _seq in esm.data.read_fasta(FASTA_PATH):
    sequences.append(_seq)

    scaled_effect = header.split('|')[-1]
    labels.append(scaled_effect)

    # We will load premade embeddings for each protein by their header
    embs = torch.load(f'{EMB_PATH}/{header[1:]}.pt')

    if EMBEDDING_METHOD == "ESM":
        raw_embeddings.append(embs['mean_representations'][EMB_LAYER])
    elif EMBEDDING_METHOD == "PROTBERT":
        raw_embeddings.append(embs)

# Reform the list of embeddings into a numpy array of all embeddings
# (was a list of tensors previously)
embeddings = torch.stack(raw_embeddings, dim=0).numpy()
sequence_OH = tokenize_sequences(sequences)

# Example of Tokenized Sequences
print(sequences[0])
print(sequence_OH[0])
```

```
MCGRRFRLSAAWLRWRAWSLPCAAERLHRWSRPENRSSQSGTLWLERQRDARKAGNMPLNLRPRRTLACLAAFAVSACVGTVVPASVPPQTRS
[11  2  6 15 15  5 15 10 16  1  1 19 10 15 19 15  1 19 16 10 13  2  1  1
  4 15 10  7 15 19 16 15 13  4 12 15 16 16 14 16  6 17 10 19 10  4 15 14
 15  3  1 15  9  1  6 12 11 13 10 12 10 15 13 15 15 17 10  1  2 10  1  1
  5  1 18 16  1  2 18  6 17 18 18 13  1 16 18 13 13 14 17 15 16 13 17 18
 16 13 15 13 18 13  1 13  1 13  1 14 15  4  3 15  9 13 10 17 10 13  1 16
 10  3 13  6  5  1 15 13 13 13  4 10 12  4 16  8 12  1 10 19  9  1  5 13
  6  9 17  6  8  1 18  1 15  8  3  6  3 19 14 10  1 19 15  4 12  3 10  5
 13 14 14 16 18 16  9 10 19 18  1 10  1 18 10  3  1 18  3 15  6  3 18  9
 10  3  3 14 18 15  8  6 15  4  3 10 17 10  5 16 14 13  8  1  1 15 18 10
  1  4  6 16  8 12 13 17 18  1  4 10 10  4 14  1 18 12 15 16  3 12 17  1
 12  3 16 10 14 15 15  8  6  6 13 14  9 18 15  3  5  8  1 15  9  9 10  6
 10  8 15  5  6 13  6  4 15 10 10 14  1 14  8  1  6 10 17 19  3 14 16 11
 16 10 12 15  6  5  4  1  1 15 16  9 10 13 15  4 18 15 14  1  1 11  3 15
 20 10  1  3 13  8  3  6 11 16 13  6  1  1  1 15  1 10 17 15 10  1 15  6
  4 10 10 16  1  4 16 17  9 20  1 10  3 10 10  4 15 17 15 17  6 13 12 15
 10 15  1  6 10 13  1  3 19 15 19  6  7  9 17  6 17  6 14 12 10 13 13  8
 17  1  6 20 12  3  8  6  8  1 17  1 13  3  6 17 15 20  1  8 18 18 11 10
  1  3 17 17  1 16 18 13  1 15 11 14 10 11 14  1 18  1  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

```python
#@title Sequence Length Graph
import matplotlib.pyplot as plt

# plt.xlim([0, 1800])
sequenceLengths = [len(_seq) for _seq in sequences]
plt.hist(sequenceLengths, alpha=0.5)
plt.title('Sequence Lengths')
plt.xlabel('Length')
plt.ylabel('count')
plt.yscale("log")

plt.show()
# sequenceLengths
```



```
#@title Training Parameters
train_size = 0.8 #@param {type:"slider", min:0, max:1, step:0.05}
X_embed_train, X_embed_test, y_train, y_test = train_test_split(embeddings, labels,
train_size=train_size, random_state=3351)
X_OH_train, X_OH_test, y_train, y_test = train_test_split(sequence_OH, labels,
train_size=train_size, random_state=3351)
labelEncoder = OneHotEncoder()
y_train_OH = labelEncoder.fit_transform(
    np.array(y_train).reshape(-1, 1)
).toarray()

drug_index = {
    resistance: i
    for i, resistance in enumerate(list(labelEncoder.categories_[0]))
}

y_train_numerical = np.array([drug_index[drug] for drug in y_train])
y_test_numerical = np.array([drug_index[drug] for drug in y_test])
#@title One-Hot Encoding (for training)
OH = labelEncoder.fit_transform(labelEncoder.categories_[0].reshape(-1, 1)).todense()
for row, resistance in zip(OH, labelEncoder.categories_[0]):
  print(f"{resistance:>27}: {row}")
```

```
            AMINOGLYCOSIDE: [[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
              BETA-LACTAM: [[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
 FOLATE-SYNTHESIS-INHABITOR: [[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
            GLYCOPEPTIDE: [[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]]
               MACROLIDE: [[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
               MULTIDRUG: [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]]
                PHENICOL: [[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
               QUINOLONE: [[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
            TETRACYCLINE: [[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]]
            TRIMETHOPRIM: [[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

```
#@title Numerical Encoding (for direct comparison)
for key, value in drug_index.items():
  print(f"{key:>27}: {value}")
```

```
            AMINOGLYCOSIDE: 0
              BETA-LACTAM: 1
 FOLATE-SYNTHESIS-INHABITOR: 2
            GLYCOPEPTIDE: 3
               MACROLIDE: 4
               MULTIDRUG: 5
                PHENICOL: 6
               QUINOLONE: 7
            TETRACYCLINE: 8
            TRIMETHOPRIM: 9
```

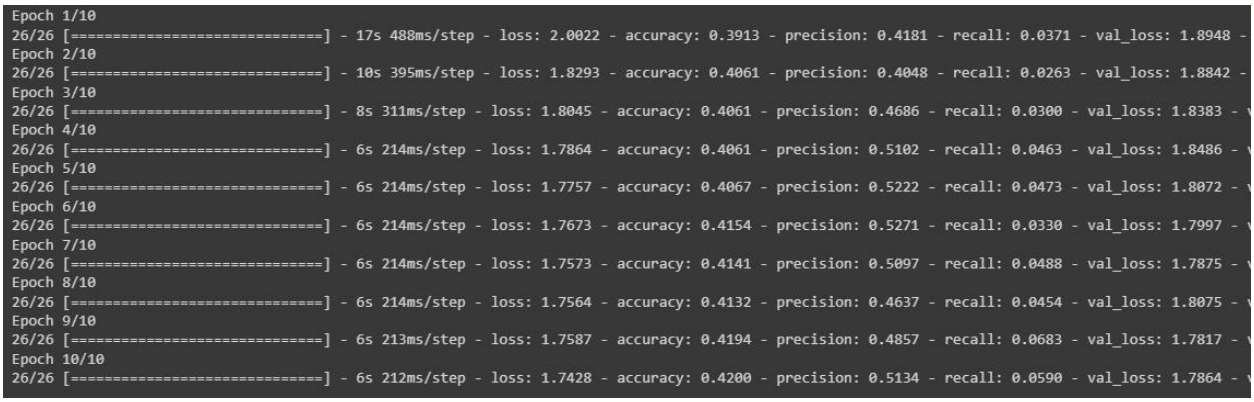
```
#@title Class Distribution
labels, counts = np.unique(y_train, return_counts = True)
dict(zip(labels, counts))
```

```
{'AMINOGLYCOSIDE': 408,
 'BETA-LACTAM': 1635,
 'FOLATE-SYNTHESIS-INHABITOR': 512,
 'GLYCOPEPTIDE': 642,
 'MACROLIDE': 55,
 'MULTIDRUG': 62,
 'PHENICOL': 169,
 'QUINOLONE': 87,
 'TETRACYCLINE': 423,
 'TRIMETHOPRIM': 55}
```

```
epochs = 10 #@param {type:"slider", min:10, max:200, step:5}
train_from_scratch = True #@param {type:"boolean"}
dense_layer_nodes = 64 #@param {type:"slider", min:8, max:128, step:1}
activation_function = "relu" #@param ["relu", "tanh", "sigmoid"]
if train_from_scratch:
    model = models.Sequential()

    # Embedding Layer
    emb_layer = layers.Embedding(input_dim=21, output_dim=10)
    emb_layer._name = "embedding_layer"
    model.add(emb_layer)
    model.add(layers.Dropout(0.25))

    # Convlutional Neural Networks
    model.add(
        layers.Conv1D(
            filters=32,
            kernel_size=15,
            activation=activation_function,
            kernel_regularizer=regularizers.L2(5e-4),
            padding="same",
            bias_regularizer=regularizers.L2(0.0001),
        )
    )
    model.add(layers.Dropout(0.2))
    model.add(layers.MaxPooling1D())

    model.add(
        layers.Conv1D(
            filters=16,
            kernel_size=5,
```

```
        activation=activation_function,
        kernel_regularizer=regularizers.L2(8e-4),
        padding="same",
        bias_regularizer=regularizers.L2(0.0001),
    )
)
model.add(layers.Dropout(0.2))
model.add(layers.MaxPooling1D())

model.add(
    layers.Conv1D(
        filters=8,
        kernel_size=3,
        activation=activation_function,
        kernel_regularizer=regularizers.L2(4e-4),
        padding="same",
        bias_regularizer=regularizers.L2(0.0001),
    )
)
model.add(layers.Dropout(0.2))
model.add(layers.GlobalMaxPooling1D())

# Dense Layers
model.add(
    layers.Dense(
        units=dense_layer_nodes,
        activation=activation_function,
        kernel_regularizer=regularizers.L2(0.001),
        bias_regularizer=regularizers.L2(0.0001),
    )
)
final_dense = layers.Dense(
    units=dense_layer_nodes,
    activation=activation_function,
    kernel_regularizer=regularizers.L2(0.001),
    bias_regularizer=regularizers.L2(0.0001),
)
final_dense._name = "final_dense_layer"
model.add(final_dense)

# Output Layer
model.add(layers.Dense(units=10, activation="softmax"))

o = optimizers.Adam(learning_rate=5e-3)
```

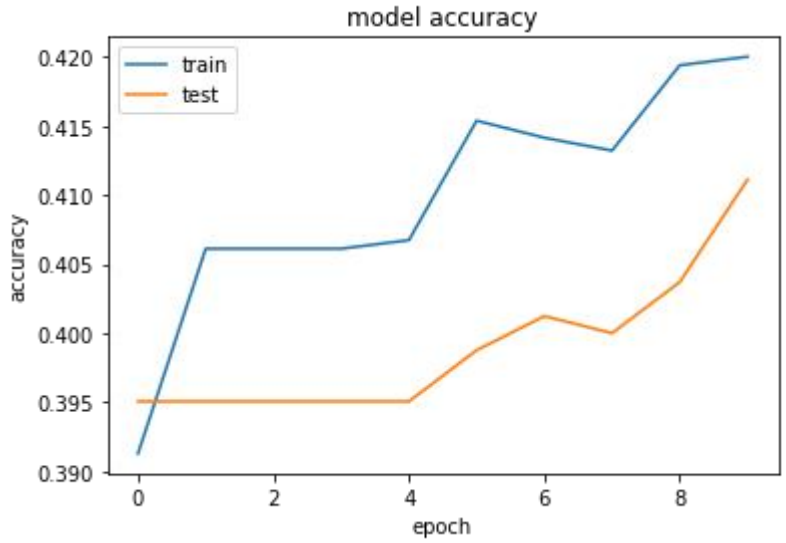
```
    model.compile(
        optimizer=o,
        loss="categorical_crossentropy",
        metrics=["accuracy", "Precision", "Recall"],
    )
else:
    model = models.load_model("model.h5")

history = model.fit(
    X_OH_train,
    y_train_OH,
    validation_split=0.2,
    epochs=epochs,
    batch_size=128,
    use_multiprocessing=True,
    verbose=1,
)
```

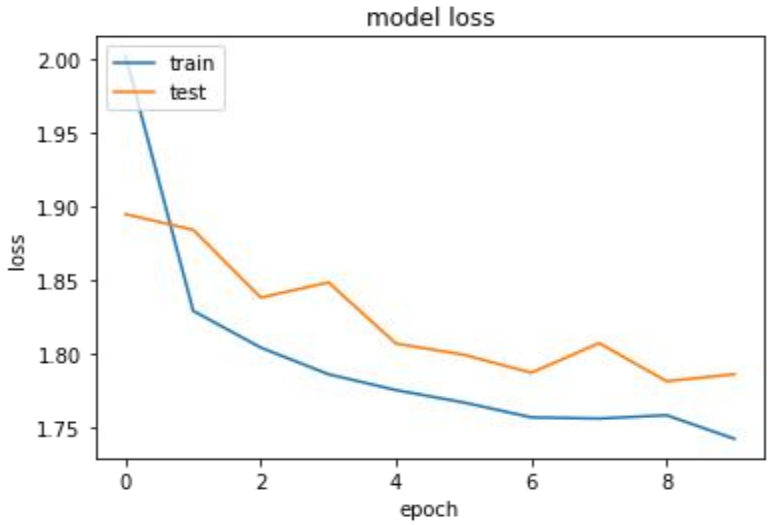



```
# @title What Does the Embedding Layer Do?
for index, amino_acid in enumerate(AA_LIST):
  print(f"{amino_acid}: {model.get_layer('embedding_layer').get_weights()[0][index]}")
```

```
X: [ 0.01021278 -0.02546532 -0.0216385   0.06660565 -0.05578854 -0.04410203
 -0.04211442  0.00646948 -0.01851141  0.00088043]
A: [ 0.14522146  0.00181326  0.10822492 -0.10666203  0.08743387 -0.02172974
  0.05794251 -0.0254023  -0.08171476  0.05479505]
C: [-0.36086252  0.20693369 -0.25131398  0.2164445  -0.37535545  0.33965483
  0.03525663 -0.03150395 -0.13327666 -0.16019696]
D: [-0.03984692 -0.01268936 -0.21050522  0.02130016 -0.03630684  0.27266145
  0.10673857 -0.02725819  0.06679615  0.0655782 ]
E: [-0.18074143  0.15520887 -0.17098442  0.10761734 -0.1490757   0.13723537
 -0.10935806  0.05548935  0.0527284  -0.05783186]
F: [-0.10356066  0.07514222 -0.1787959   0.15850353 -0.15912908  0.16740094
 -0.09414938 -0.02453906 -0.06940825 -0.01943   ]
G: [ 0.08554253 -0.17866321 -0.08674045 -0.13546067  0.07273582 -0.0020092
  0.06573879 -0.11538886  0.12294222 -0.03603984]
H: [ 0.08642146 -0.12130201  0.11327723 -0.02342233  0.18004511 -0.1768061
  0.15216424 -0.16394591 -0.1884866   0.08196507]
I: [-0.00492929  0.00096251  0.02262378  0.00190651 -0.06294964 -0.01075464
 -0.11391471  0.09258754 -0.06196152 -0.00515222]
K: [-0.1008336   0.07341755 -0.14325671  0.0759105  -0.14793369  0.13594884
 -0.14296098  0.04528189 -0.06807911 -0.08215145]
L: [ 0.00023981  0.08106371 -0.03166438  0.09940179  0.02495156 -0.0040456
 -0.01504157  0.17201988  0.09389512  0.02864979]
```

```
#@title Training Accuracy and Loss
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

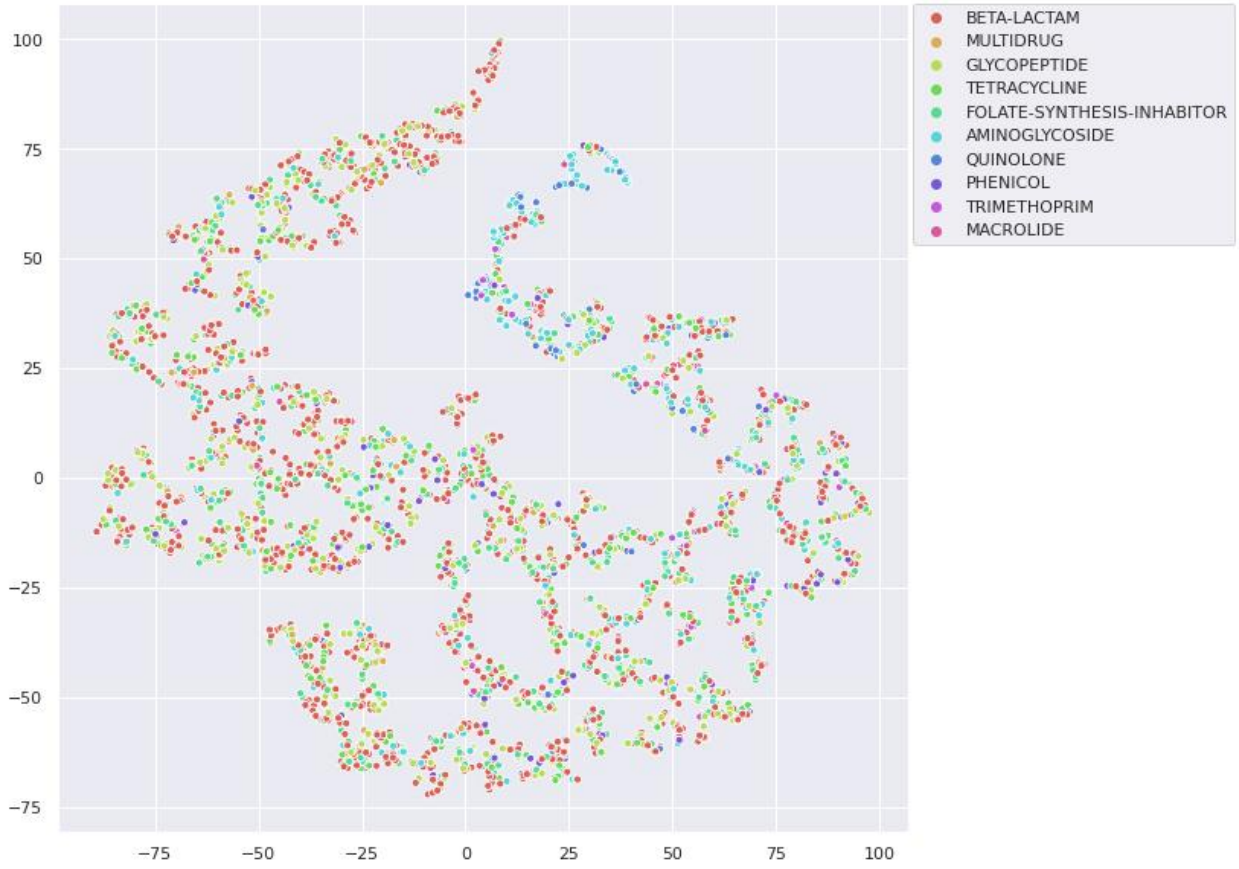





```
#@title Classification Reports
from sklearn.metrics import classification_report
values = np.argmax(model.predict(X_OH_test), axis = 1)
actual = np.array(y_test_numerical)

print(classification_report(y_true = actual, y_pred = values, target_names =
drug_index.keys()))
```

	precision	recall	f1-score	support
AMINOGLYCOSIDE	0.44	0.26	0.32	94
BETA-LACTAM	0.42	0.97	0.59	413
FOLATE-SYNTHESIS-INHABITOR	0.00	0.00	0.00	159
GLYCOPEPTIDE	0.00	0.00	0.00	147
MACROLIDE	0.00	0.00	0.00	22
MULTIDRUG	0.00	0.00	0.00	12
PHENICOL	0.00	0.00	0.00	52
QUINOLONE	0.00	0.00	0.00	17
TETRACYCLINE	0.00	0.00	0.00	83
TRIMETHOPRIM	0.00	0.00	0.00	13
accuracy			0.42	1012
macro avg	0.09	0.12	0.09	1012
weighted avg	0.21	0.42	0.27	1012

```
#Assign a model
from keras.models import Model
embedding_model = Model(inputs = model.input,
             outputs = model.get_layer("final_dense_layer").output)
x_emb = embedding_model.predict(X_OH_train)
#@title Graph Embeddings
iterations = 1000 #@param {type:"slider", min:250, max:2000, step:50}
tsne = TSNE(init = "pca", perplexity = 15, early_exaggeration=20, n_iter = iterations)
X_embed_train_tsne = tsne.fit_transform(x_emb)
sns.set(rc={"figure.figsize": (10, 10)})
palette = sns.color_palette("hls", len(set(y_train)))
sc = sns.scatterplot(X_embed_train_tsne[:,0], X_embed_train_tsne[:,1],
hue=y_train ,marker='.', legend="full", s=100, palette=palette)
plt.legend(bbox_to_anchor=(1.005, 1.0), loc=2, borderaxespad=0.0)
```



What is PCA?

Principal Component Analysis (PCA) is a dimensionality-reduction method that transforms a large set of variables into a smaller one while still containing most of the information in the large set.

It does this through a process of computing the principal components and using them to perform a change of basis on the data.

Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

```
pca = PCA(n_components=20)
X_embed_train_pca = pca.fit_transform(X_embed_train)
X_embed_test_pca = pca.transform(X_embed_test) # for a later step
print(X_embed_train.shape)
print(X_embed_train_pca.shape)
```

```
(4048, 1280)
(4048, 20)
```

Visualize Embeddings

Here, we plot the first two principal components on the x- and y- axes. Each point is then colored by its scaled effect (what we want to predict).

Visually, we can see a separation based on color/effect, suggesting that our representations are useful for this task, without any task-specific training!

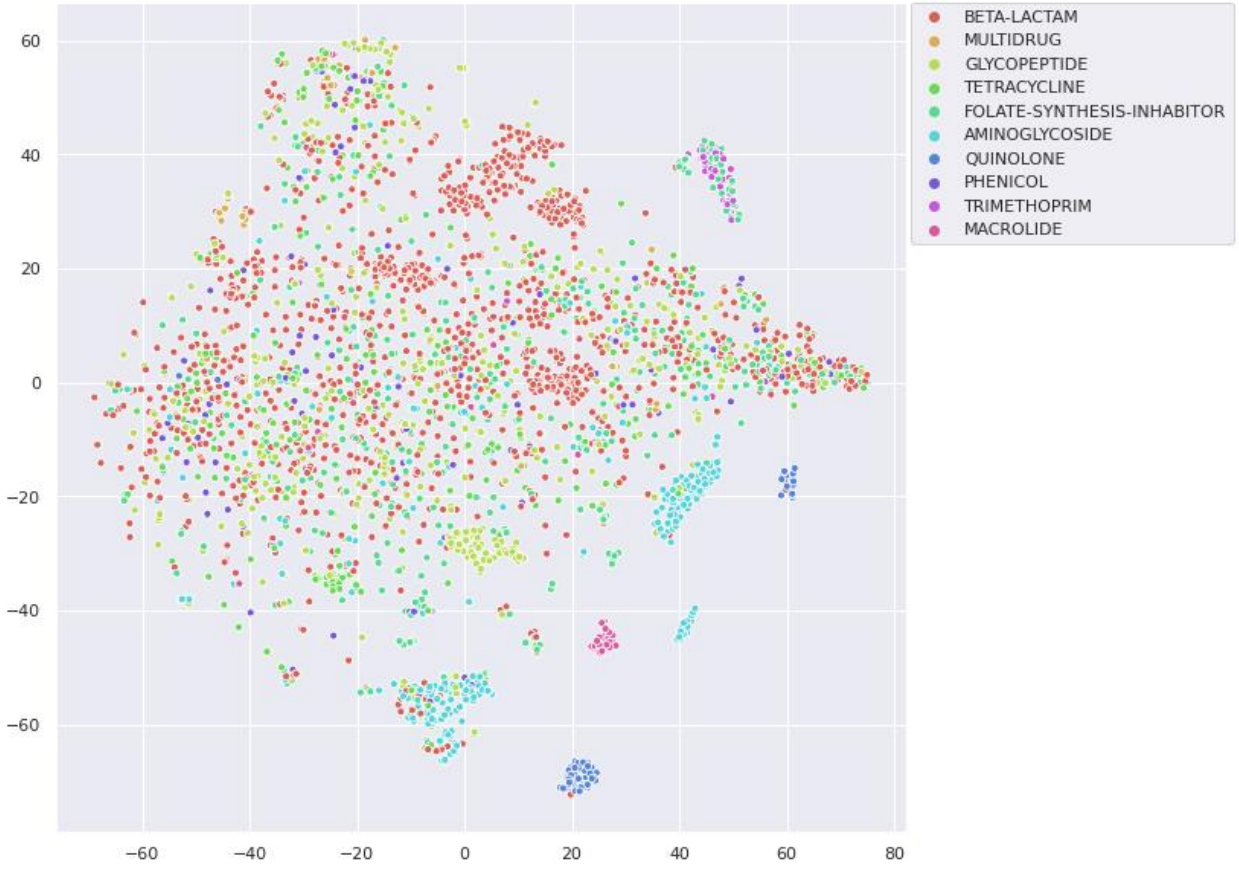
```
#@title PCA Plot
%timeit
sns.set(rc={"figure.figsize": (10, 10)})
palette = sns.color_palette("hls", len(set(y_train)))
sc = sns.scatterplot(X_embed_train_pca[:,0], X_embed_train_pca[:,1],
hue=y_train ,marker='.', legend="full", s=100, palette=palette)
plt.legend(bbox_to_anchor=(1.005, 1.0), loc=2, borderaxespad=0.0)
```

TSNE

t-distributed stochastic neighbor embedding (t-SNE) is a dimensionality-reduction method that transforms a large set of variables into a smaller one by modeling each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability. You can modify the number of iterations that the t-SNE runs with the slider; consider keeping the value below 500 if you are on CPU.

```
Interactive and in depth explanation of t-SNE.
https://observablehq.com/@robstelling/t-sne_en
```

```
Documentation: https://scikit-
learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

#@title TSNE Plot
iterations_ = 750 #@param {type:"slider", min:250, max:2000, step:50}
tsne = TSNE(init = "pca", n_iter = iterations_)
X_embed_train_tsne = tsne.fit_transform(X_embed_train)

sns.set(rc={"figure.figsize": (10, 10)})
palette = sns.color_palette("hls", len(set(y_train)))
sc = sns.scatterplot(X_embed_train_tsne[:,0], X_embed_train_tsne[:,1],
hue=y_train ,marker='.', legend="full", s=100, palette=palette)
plt.legend(bbox_to_anchor=(1.005, 1.0), loc=2, borderaxespad=0.0)
```



Clustering on Embeddings

We will use two different regression models:

1.[K-nearest-neighbors](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html)

2. [Random Forest Regressor](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html?highlight=randomforestregressor#sklearn.ensemble.RandomForestRegressor)

We will be fitting these classifiers on the PCA-transformed embeddings, as they perform just as well as the raw embeddings while also being significantly faster. Don't believe it? Try swapping X-embed_train_pca for X_embed_train (and likewise for test).

```
knn = KNeighborsClassifier(
    n_neighbors = 10,
    weights = "uniform",
    algorithm = "kd_tree",
    leaf_size = 15,
    p = 1,
)

rfc = RandomForestClassifier(
    n_estimators = 20,
    criterion = "gini",
    min_samples_split = 10,
    min_samples_leaf = 4,
    max_features = "sqrt"
)

#@title Classification Reports: KNN
knn.fit(X_embed_train_pca, y_train_numerical)
pred_y = knn.predict(X_embed_test_pca)
print(classification_report(y_true = actual, y_pred = pred_y, target_names =
drug_index.keys()))
```

	precision	recall	f1-score	support
AMINOGLYCOSIDE	0.77	0.77	0.77	94
BETA-LACTAM	0.56	0.81	0.66	413
FOLATE-SYNTHESIS-INHABITOR	0.42	0.30	0.35	159
GLYCOPEPTIDE	0.42	0.34	0.37	147
MACROLIDE	1.00	0.59	0.74	22
MULTIDRUG	0.40	0.33	0.36	12
PHENICOL	0.25	0.02	0.04	52
QUINOLONE	1.00	1.00	1.00	17
TETRACYCLINE	0.52	0.19	0.28	83
TRIMETHOPRIM	0.67	0.62	0.64	13
accuracy			0.55	1012
macro avg	0.60	0.50	0.52	1012
weighted avg	0.53	0.55	0.52	1012

```
#@title Classification Reports: RFC
rfc.fit(X_embed_train_pca, y_train_numerical)
```

```
pred_y = rfc.predict(X_embed_test_pca)
print(classification_report(y_true = actual, y_pred = pred_y, target_names =
drug_index.keys()))
```

	precision	recall	f1-score	support
AMINOGLYCOSIDE	0.90	0.78	0.83	94
BETA-LACTAM	0.54	0.89	0.67	413
FOLATE-SYNTHESIS-INHABITOR	0.63	0.28	0.38	159
GLYCOPEPTIDE	0.41	0.27	0.32	147
MACROLIDE	1.00	0.55	0.71	22
MULTIDRUG	0.50	0.08	0.14	12
PHENICOL	1.00	0.02	0.04	52
QUINOLONE	1.00	1.00	1.00	17
TETRACYCLINE	0.35	0.14	0.21	83
TRIMETHOPRIM	0.71	0.92	0.80	13
accuracy			0.57	1012
macro avg	0.70	0.49	0.51	1012
weighted avg	0.60	0.57	0.52	1012

Initialize / Run GridSearch

Finding the right parameters for any classifiers is a challenge. To aid in finding them for KNN and K-Means from above, we ran [grid search](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html). It works by generating multiple permutations of possible parameters, and testing them with cross-validation.

```
knn_grid = {
    'n_neighbors': [5, 10],
    'weights': ['uniform', 'distance'],
    'algorithm': ['ball_tree', 'kd_tree', 'brute'],
    'leaf_size' : [15, 30],
    'p' : [1, 2],
}

# only added with the checkbox active above
rfc_grid = {
    'n_estimators' : [20],
    'criterion' : ['gini', 'entropy'],
    'max_features': ['sqrt', 'log2'],
    'min_samples_split' : [5, 10],
    'min_samples_leaf': [1, 4]
}
cls_list = [KNeighborsClassifier, RandomForestClassifier]
param_grid_list = [knn_grid, rfc_grid]
```

```
### Run Grid Search
result_list = []
grid_list = []
for cls_name, param_grid in zip(cls_list, param_grid_list):
    print(cls_name)
    grid = GridSearchCV(
        estimator = cls_name(),
        param_grid = param_grid,
        scoring = 'r2',
        verbose = 1,
        n_jobs = -1 # use all available cores
    )
    grid.fit(X_embed_train_pca, y_train_numerical)
    result_list.append(pd.DataFrame.from_dict(grid.cv_results_))
    grid_list.append(grid)
```

```
<class 'sklearn.neighbors._classification.KNeighborsClassifier'>
Fitting 5 folds for each of 48 candidates, totalling 240 fits
<class 'sklearn.ensemble._forest.RandomForestClassifier'>
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

Browse the Sweep Results

The following tables show the top 5 parameter settings, based on `mean_test_score`.
Given our setup, this should really be thought of as `validation_score`.
K Nearest Neighbors:
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

```
result_list[0].sort_values('rank_test_score')[:5]
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algorithm	param_leaf_size	param_n_neighbors	param_p	param_weights	params	spli
20	0.012605	0.001255	0.209969	0.007528	kd_tree	15	10	1	uniform	{'algorithm': 'kd_tree', 'leaf_size': 15, 'n_n...	
44	0.001510	0.000061	0.129676	0.003616	brute	30	10	1	uniform	{'algorithm': 'brute', 'leaf_size': 30, 'n_nei...	
4	0.011073	0.005683	0.165145	0.004388	ball_tree	15	10	1	uniform	{'algorithm': 'ball_tree', 'leaf_size': 15, 'n...	
28	0.012280	0.002846	0.177265	0.009809	kd_tree	30	10	1	uniform	{'algorithm': 'kd_tree', 'leaf_size': 30, 'n_n...	
										{'algorithm': 'ball_tree',	

```
### Random Forest
# Here is how to see the results of the random forest if it was run before
# (requires you to have had RFC on)
result_list[1].sort_values('rank_test_score')[:5]
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_features	param_min_samples_leaf	param_min_samples_split	param
12	1.667045	0.037218	0.009679	0.000137	entropy	log2	1	5	
3	0.471131	0.017896	0.010631	0.002603	gini	sqrt	4	10	
6	0.460945	0.013936	0.009571	0.000404	gini	log2	4	5	
10	1.567228	0.015148	0.009728	0.000500	entropy	sqrt	4	5	
7	0.451664	0.008666	0.009340	0.000299	gini	log2	4	10	

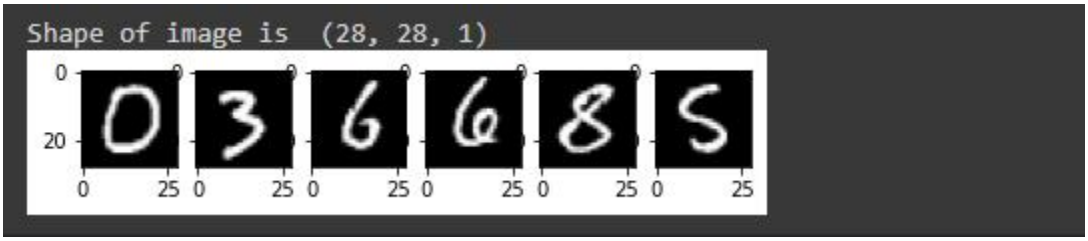
SECTION 3: To demonstrate the use of Convoluted Neural Network (CNN).

```
#NOTE: Convoluted Neural Network (CNN) is used to decode patterns…

# Let us load the necessary libraries for python implementation of convoluted neural networks
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras import layers
import matplotlib.pyplot as plt
from keras.models import Model

# Lets download the datasets
num_classes = 10 #Numbers from 0-9
input_shape = (28, 28, 1) #28*28 images with single greyscale layer

#Load data and seperate into test and training
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

#Prepare Data by dividing into 255 to make the calculations easier
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")
```

```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
60000 train samples
10000 test samples
```

```
#Display example images, 10, 30, 50. Can change these to see other ones.
sampleTestImages = [10, 30, 50, 123, 1234, 1235]
fig, figPlacement = plt.subplots(1, len(sampleTestImages))
examples = list(zip(sampleTestImages, figPlacement))
print('Shape of image is ', x_test[sampleTestImages[0]].shape)
for example in examples:
  example[1].imshow(tf.squeeze(x_test[example[0]], 2), cmap='gray')
```

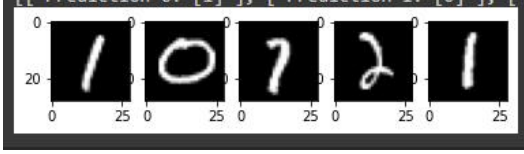



```
# To define and build our model
# Note: we have 2 convolutional layers with 3 by 3 filers
# relu stands for rectified linear activation unit
# softmax function is used to make final decision of classification
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

#Show model settings
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dropout (Dropout)           (None, 1600)              0

 dense (Dense)               (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
```

```
# To test the images
def myTestFunction(sampleTestImages):
  fig, figPlacement = plt.subplots(1, len(sampleTestImages))
  examples = list(zip(sampleTestImages, figPlacement))
  predictions = []
  for example in examples:
    example[1].imshow(tf.squeeze(x_test[example[0]], 2), cmap='gray')
    predictions.append(tf.math.argmax(model.predict(tf.expand_dims(x_test[example[0]],
0)), 1))
  print([['Prediction ' + str(i) + ': ' + str(np.array(predictions[i]))] for i in
range(len(sampleTestImages))])
# Let us now predict our image (N.B. In best case we expect an accuracy of 10%
because their are 10 classes)
myTestFunction(sampleTestImages = [5, 69, 111, 244, 768])
# Notice from our result that the model could not predict any of the numbers
accurately, since we performed no training
```

```
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
[['Prediction 0: [1]'], ['Prediction 1: [0]'], ['Prediction 2: [7]'], ['Prediction 3: [2]'], ['Prediction 4: [1]']]
```

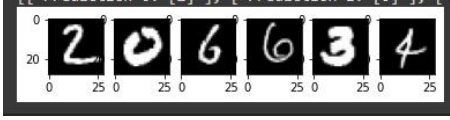


```
#Choose training parameters and train the model (LET'S TRAIN)
batch_size = 32
epochs = 2

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

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.1)
```

```
Epoch 1/2
1688/1688 [==============================] - 63s 37ms/step - loss: 0.0647 - accuracy: 0.9800 - val_loss: 0.0445 - val_accuracy: 0.9873
Epoch 2/2
1688/1688 [==============================] - 67s 39ms/step - loss: 0.0537 - accuracy: 0.9833 - val_loss: 0.0355 - val_accuracy: 0.9898
<keras.callbacks.History at 0x7f051e1d6ac0>
```

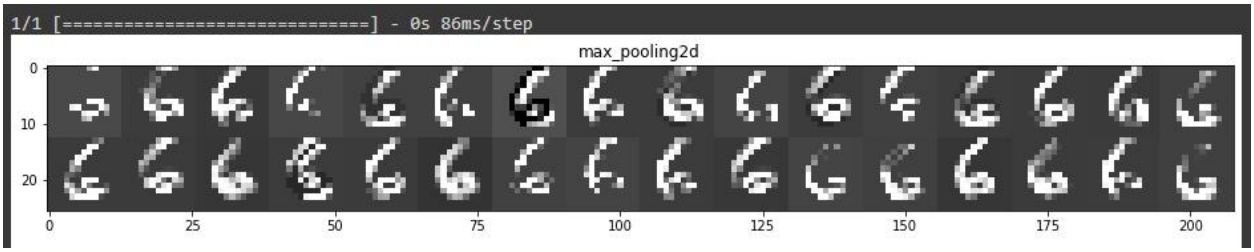
```
# Let us check our accuracy again
myTestFunction(sampleTestImages = [1, 25, 50, 100, 200, 300])
# Notice the accuracy now that we have performed training of the datasets
```

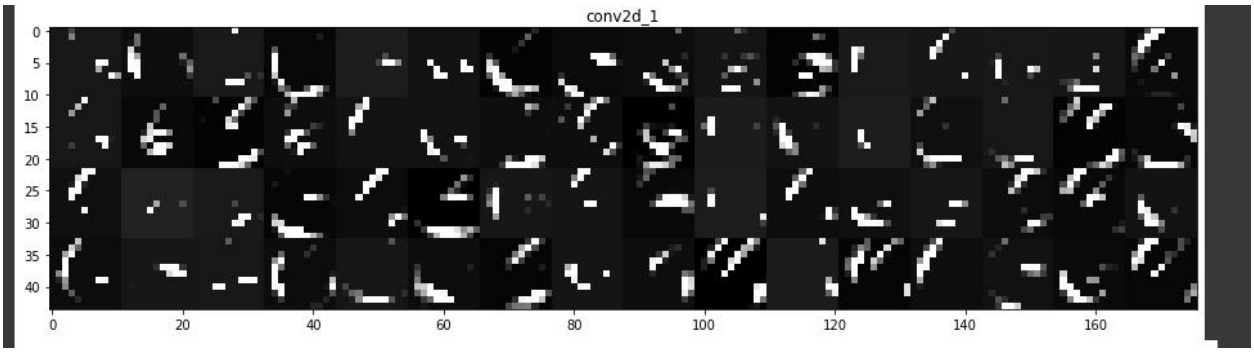
```
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
[['Prediction 0: [2]'], ['Prediction 1: [0]'], ['Prediction 2: [6]'], ['Prediction 3: [6]'], ['Prediction 4: [3]'], ['Prediction 5: [4]']]
```

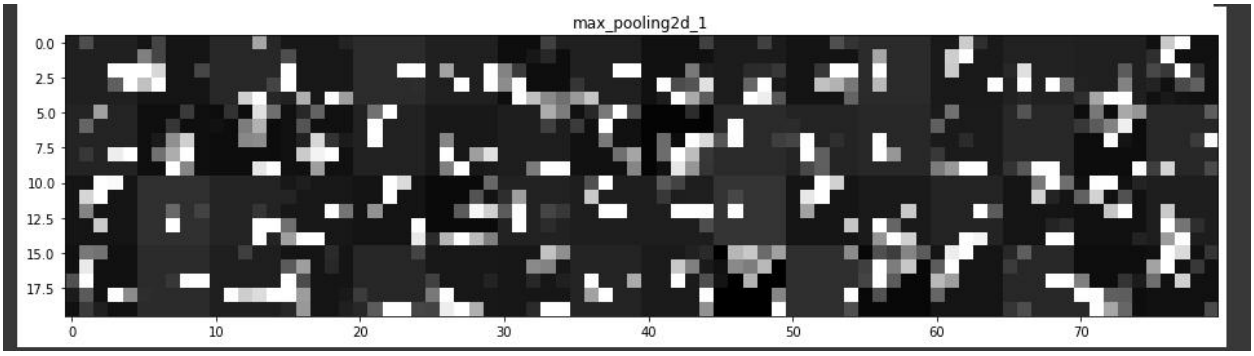


```
#What the model is looking at... Note: After first max pooling layer the machine
first sees some unclear patterns. In the second convolutional layer it becomes
clearer with some visible curves. In the final (and third) convolutional layer the
machine is able to recognize the hand written digits
exampleNumber = 600
#We can always change the exampleNumber. The algoithm is the same
layer_outputs = [layer.output for layer in model.layers[1:7]]
activation_model = Model(inputs=model.input,outputs=layer_outputs)

activations = activation_model.predict(tf.expand_dims(x_test[exampleNumber], 0))

layer_names = []
for layer in model.layers[1:4]:
    layer_names.append(layer.name)
```

```
images_per_row = 16
for layer_name, layer_activation in zip(layer_names, activations):
    n_features = layer_activation.shape[-1]
    size = layer_activation.shape[1]
    n_cols = n_features // images_per_row
    display_grid = np.zeros((size * n_cols, images_per_row * size))
    for col in range(n_cols):
        for row in range(images_per_row):
            channel_image = layer_activation[0,
                                             :, :,
                                             col * images_per_row + row]
            channel_image -= channel_image.mean()
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')
            display_grid[col * size : (col + 1) * size,
                         row * size : (row + 1) * size] = channel_image
    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='gray')
```







SECTION 4: To demonstrate the use of autoencoders.

#Autoencoders

Autoencoders consist of

1) Encoder: that extracts features

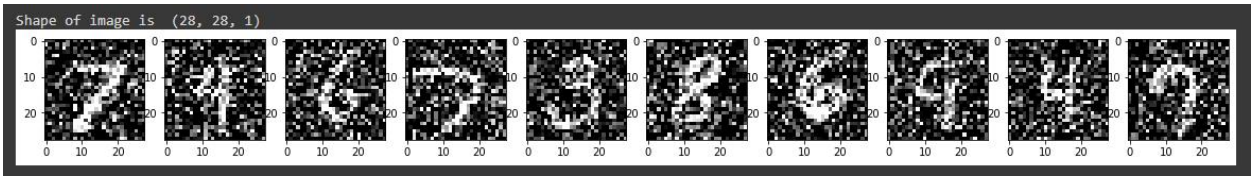
2) Decoder: that generates images from knowledge

This is for the reasons of generating more data for "fake detection and training", "denoising", "dimensional reduction", "image compression", and "recommendation systems".

```
# Import required packages
import tensorflow as tf
import keras
from keras import layers
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
#Download datasets
(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))

#Create noisy examples
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)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

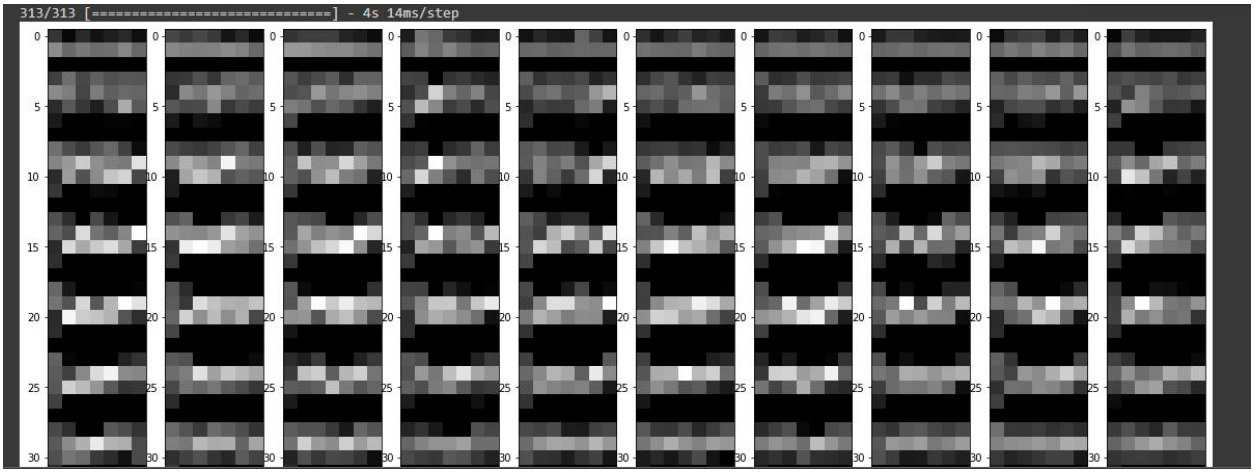
```
#Display noisy images
sampleTestImages = [34, 376, 50, 70, 90, 110, 130, 150, 170, 220]
fig, figPlacement = plt.subplots(1, len(sampleTestImages))
fig.set_size_inches(20, 10)
examples = list(zip(sampleTestImages, figPlacement))
print('Shape of image is ', x_test[sampleTestImages[0]].shape)
for example in examples:
  example[1].imshow(tf.squeeze(x_test_noisy[example[0]], 2), cmap='gray')
```



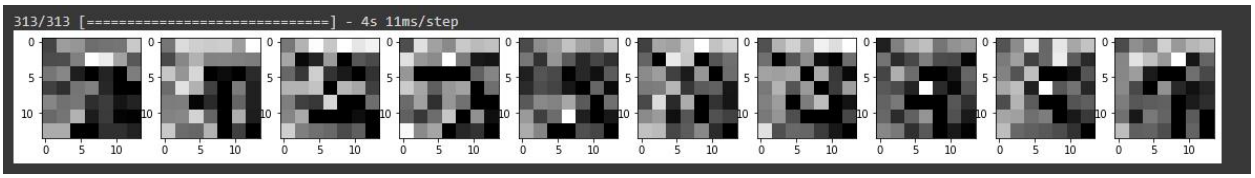

```
#Create model
input_img = keras.Input(shape=(28, 28, 1))

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

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
up1 = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(up1)
up2 = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(up2)

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

#Train model
#NOTE: An epoch is when all the training data is used at once and is defined as the
total number of iterations of all the training data in one cycle for training the
machine learning model. Another way to define an epoch is the number of passes a
training dataset takes around an algorithm.
autoencoder.fit(x_train_noisy, x_train,
                epochs=2,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test_noisy, x_test))
```

```
Epoch 1/2
469/469 [==============================] - 212s 448ms/step - loss: 0.1656 - val_loss: 0.1160
Epoch 2/2
469/469 [==============================] - 182s 387ms/step - loss: 0.1132 - val_loss: 0.1089
<keras.callbacks.History at 0x7f9802b5ffd0>
```

```
#Display encoding examples
encoder = keras.Model(input_img, encoded)
encoded_imgs = encoder.predict(x_test_noisy)

sampleTestImages = [34, 376, 50, 70, 90, 110, 130, 150, 170, 220]
fig, figPlacement = plt.subplots(1, len(sampleTestImages))
fig.set_size_inches(20, 10)
examples = list(zip(sampleTestImages, figPlacement))
for example in examples:
  example[1].imshow(encoded_imgs[example[0], :, :, 0:5].reshape((7, 35)).T,
cmap='gray')
```

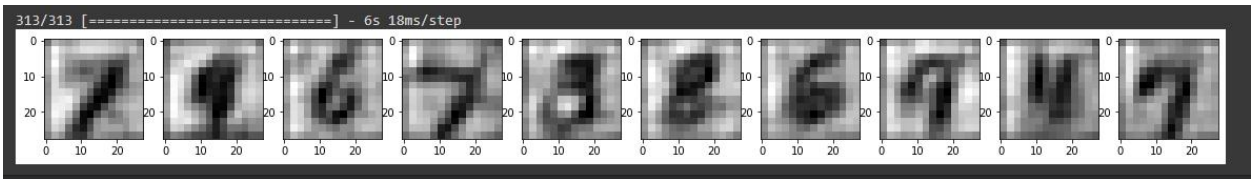



```
#Display first level examples of decompression process
up1Model = keras.Model(input_img, up1)
up1_imgs = up1Model.predict(x_test_noisy)

sampleTestImages = [34, 376, 50, 70, 90, 110, 130, 150, 170, 220]
fig, figPlacement = plt.subplots(1, len(sampleTestImages))
fig.set_size_inches(20, 10)
examples = list(zip(sampleTestImages, figPlacement))
for example in examples:
  example[1].imshow(up1_imgs[example[0], :, :, 5].reshape((14, 14)), cmap='gray')
```

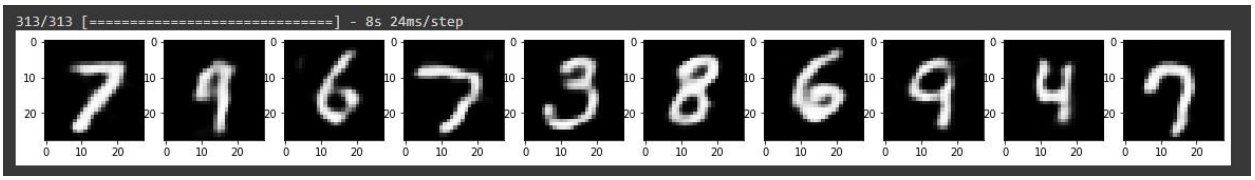



```
#Display second level examples of decompression process
```

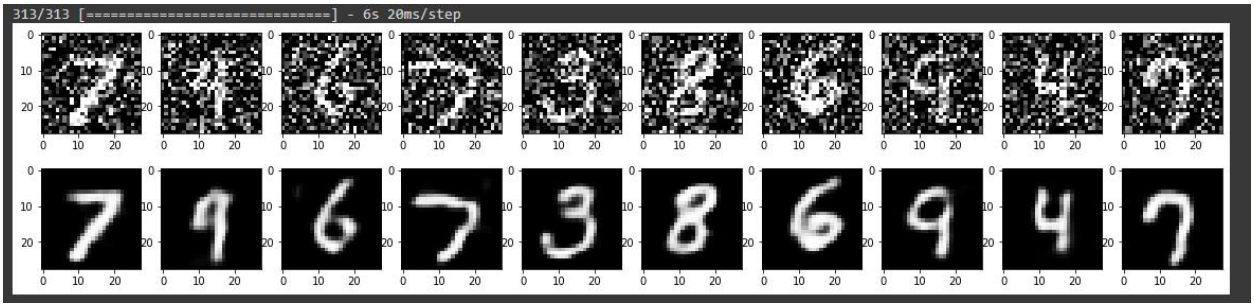
```
up2Model = keras.Model(input_img, up2)
up2_imgs = up2Model.predict(x_test_noisy)

sampleTestImages = [34, 376, 50, 70, 90, 110, 130, 150, 170, 220]
fig, figPlacement = plt.subplots(1, len(sampleTestImages))
fig.set_size_inches(20, 10)
examples = list(zip(sampleTestImages, figPlacement))
for example in examples:
  example[1].imshow(up2_imgs[example[0], :, :, 20].reshape((28, 28)), cmap='gray')
```




```
#Display decoded examples
decoded_imgs = autoencoder.predict(x_test_noisy)

sampleTestImages = [34, 376, 50, 70, 90, 110, 130, 150, 170, 220]
fig, figPlacement = plt.subplots(1, len(sampleTestImages))
fig.set_size_inches(20, 10)
examples = list(zip(sampleTestImages, figPlacement))
for example in examples:
  example[1].imshow(decoded_imgs[example[0]].reshape((28, 28)), cmap='gray')
```




```
#Compare noisy images to denoised versions
decoded_imgs = autoencoder.predict(x_test_noisy)

sampleTestImages = [34, 376, 50, 70, 90, 110, 130, 150, 170, 220]
fig, figPlacement = plt.subplots(1, len(sampleTestImages))
fig.set_size_inches(20, 10)
examples = list(zip(sampleTestImages, figPlacement))
for example in examples:
  example[1].imshow(tf.squeeze(x_test_noisy[example[0]], 2), cmap='gray')

fig, figPlacement = plt.subplots(1, len(sampleTestImages))
fig.set_size_inches(20, 10)
examples = list(zip(sampleTestImages, figPlacement))
for example in examples:
  example[1].imshow(decoded_imgs[example[0]].reshape((28, 28)), cmap='gray')
```


313/313 [==============================] - 6s 20ms/step