-

Microscopium

Constant def

In[49]:=
```
convChannels = {32, 8, 1};
deconvChannels = {1, 8, 3};
linearSize = 1024;
latentSize = 256;
imageSize = {512, 512};
outputPath = "/Users/ChengLuhan/variational-autoencoder/output/";
useGPU = False;
inputImagePath = "/Users/ChengLuhan/micro/NewDataMicro";
```

Load datasets

In[9]:=
```
If[useGPU, Needs["CUDALink`"], Nothing];
Needs["NeuralNetworks`"];

loadData[imageSize_, numImage_, imagePath_] :=
  Module[
   {
    specPath, files, images, spec, foundimages,
    indexset, labels, dataset, foundImageIdx, foundLabelIdx
   },
   specPath = StringJoin[imagePath, "/metadata-joined.csv"];
   spec = Import[specPath, "CSV"];
   spec = AssociationThread[spec[[1]], spec[[2 ;; -1]] // Transpose];
   files = FileNames["*.jpeg", imagePath];
   foundimages = FileBaseName[#] ~~ ".jpeg" & /@ files;
   foundImageIdx = Position[spec["url"], #] & /@ foundimages // Flatten;
   foundLabelIdx = Position[spec["moa"], Except[""]] // Flatten;
   indexset = Intersection[foundImageIdx, foundLabelIdx][[1 ;; numImage]];
   labels = spec["moa"][[indexset]];
   images = Import[StringJoin[StringJoin[imagePath, "/"], #], "JPEG"] & /@
     spec["url"][[indexset]];
   dataset = Normal@AssociationThread[ImageResize[#, imageSize] & /@ images,
      spec["moa"][[indexset]]];
   dataset = Association["Image" → Keys[#], "MOA" → Values[#]] & /@ dataset //
     Dataset;
   dataset
  ];
```

In[12]:=
```
dataset = loadData[imageSize, 50, inputImagePath];
```

AutoEncoder

In[13]:=
```
inputShape = Join[{3}, imageSize];
activation = ElementwiseLayer["Sigmoid"];
```

In[15]:=
```
createConvs[channels_, blockType_, inputShape_] := Module[
    {
     convolutions = blockType[#, 4, "Stride" → 2, "PaddingSize" → 1] & /@ channels,
     batchnorms = Table[BatchNormalizationLayer[], Length[channels]],
     activations = Table[activation, Length[channels]],
    },
    NetChain[Flatten@Transpose[{convolutions, batchnorms, activations}],
     "Input" → inputShape]
   ];
encoder[channels_] := createConvs[channels, ConvolutionLayer, inputShape];
encoderOutputShape = {convChannels[[-1]],
    inputShape[[2]] * 0.5 ^ Length[convChannels] // IntegerPart,
    inputShape[[3]] * 0.5 ^ Length[convChannels] // IntegerPart};
encoderOutputSize = Times @@ encoderOutputShape;
deconder[channels_] :=
  createConvs[channels, DeconvolutionLayer, encoderOutputShape];
```

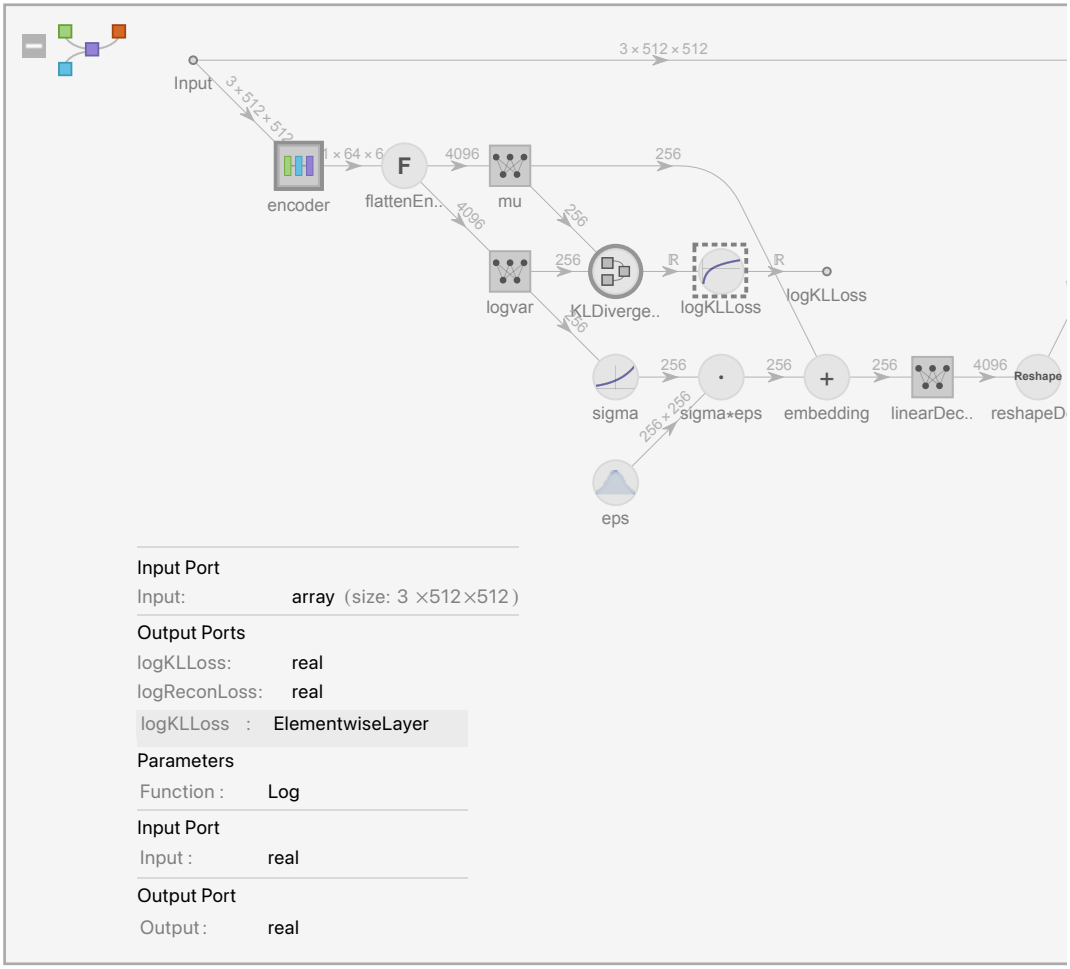
In[20]:=
```
netStructure = Association[{
     "encoder" → encoder[convChannels], "flattenEncoder" → FlattenLayer[],
     "mu" → LinearLayer[latentSize], "logvar" → LinearLayer[latentSize],
     "sigma" → ElementwiseLayer[Exp[0.5 * #] &],
     "eps" → RandomArrayLayer[NormalDistribution[] &],
     "sigma*eps" → DotLayer[], "embedding" → TotalLayer[],
     "linearDecoder" → LinearLayer[encoderOutputSize], "reshapeDecoder" →
      ReshapeLayer[encoderOutputShape], "decoder" → deconder[deconvChannels],
     "reconstructionLoss" → MeanSquaredLossLayer[],

     "KLDivergence" → NetGraph[{ElementwiseLayer[Power[#, 2] &],
        ElementwiseLayer[Exp], TotalLayer[], ElementwiseLayer[-# + 1 &],
        TotalLayer[], SummationLayer[], ElementwiseLayer[-0.5 * # &]},
       {NetPort["Input1"] → 1, NetPort["Input2"] → 2, {1, 2} → 3 → 4,
        {4, NetPort["Input2"]} → 5 → 6 → 7 -> NetPort["KLLoss"]}],
     "logReconLoss" → ElementwiseLayer[Log],
     "logKLLoss" → ElementwiseLayer[Log]
    }];
```

In[ ]:=

In[21]:=
```
net = NetGraph[
    netStructure,
    {
     "encoder" → "flattenEncoder" → {"mu", "logvar"},
     "logvar" → "sigma",
     {"sigma", "eps"} → "sigma*eps",
     {"mu", "sigma*eps"} →
      "embedding" → "linearDecoder" → "reshapeDecoder" → "decoder",
     {NetPort["Input"], "decoder"} → "reconstructionLoss",
     {"mu", "logvar"} → "KLDivergence",
     "reconstructionLoss" → "logReconLoss",
     "KLDivergence" → "logKLLoss",
     "logReconLoss" → NetPort["logReconLoss"],
     "logKLLoss" → NetPort["logKLLoss"]
    }
   ];
```

In[ ]:=
```
NetInitialize[net]
```

Out[ ]=
NetGraph



Input Port
Input: array (size: 3 ×512×512)

Output Ports
logKLLoss: real
logReconLoss: real

logKLLoss : ElementwiseLayer

Parameters
Function : Log

Input Port
Input : real

Output Port
Output : real

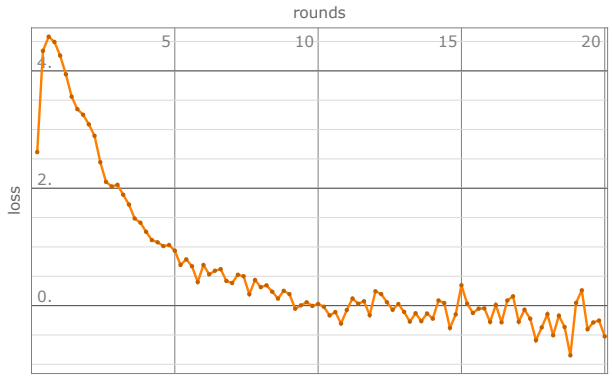
In[22]:=
```
results = NetTrain[
  net,
  Association[
   "Input" → Transpose[ImageData /@ Normal@dataset[All, "Image"], 2 ↔ 4]],
  All,
  MaxTrainingRounds → 20,
  TargetDevice → If[useGPU, "GPU", "CPU"],
  LossFunction → {"logReconLoss", "logKLLoss"}
 ]
```

Out[22]= NetTrainResultsObject

NetTrain Results

summary	batches: 100, rounds: 20, time: 5.4min, examples /s: 3.7
data	training examples: 50, processed examples: 1200, skipped examples: 0
method	ADAM optimizer, batch size 12, CPU
round	loss: -2.41×10^{-1}, log recon loss: −1.71, log klloss: 1.47

‹ loss ›



In[26]:=
```
trained = results["TrainedNet"];
features = NetTake[trained, "mu"] @
    Transpose[ImageData /@ Normal@dataset[All, "Image"], 2 ↔ 4];
labels = dataset[All, "MOA"];
```

FileNameJoin : Options expected (instead of trained) beyond position 1 in FileNameJoin [~/variational −autoencounder /trained /, trained]. An option must be a rule or a list of rules.

Out[30]= FileNameJoin[~/variational-autoencounder/trained/, trained]

In[57]:=
```
Switch[
  FileType[outputPath],
  Directory, Nothing,
  None, CreateDirectory[outputPath]
 ];
```

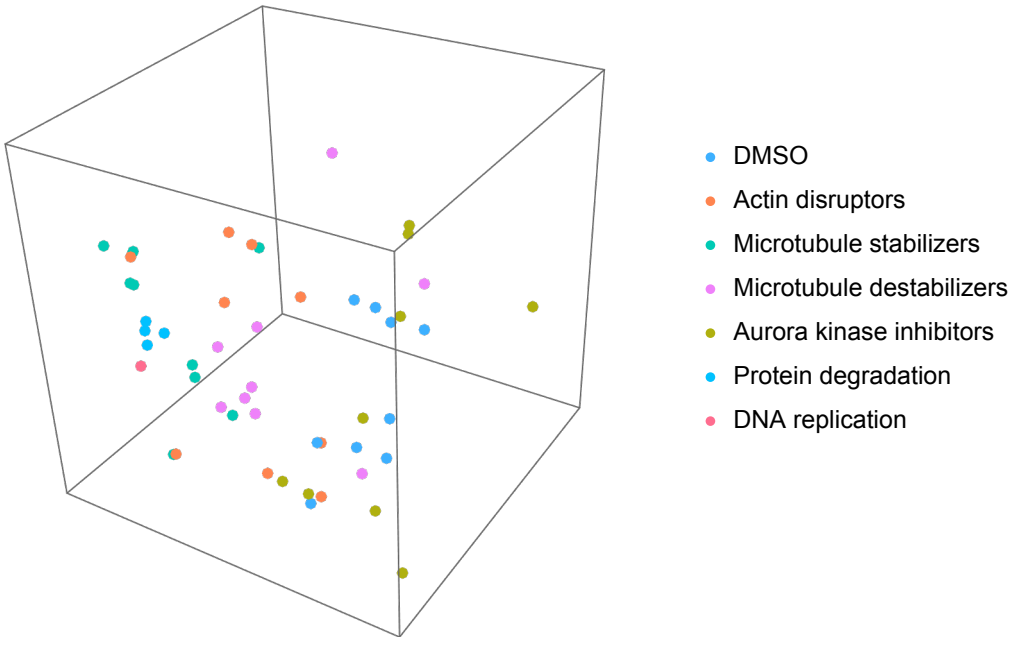
In[59]:=
```
Export[FileNameJoin[{outputPath, "network", "trained_network.wlnet"}], trained];
```

In[44]:=
```
visualize3D[features_, labels_] := Module[
  {},
  coords = DimensionReduce[features, 3];
  uniqueLabels = Normal@Keys@Counts@labels;
  ListPointPlot3D[
   Table[Extract[coords, Position[labels, i]], {i, uniqueLabels}],
   PlotLegends → PointLegend[96, uniqueLabels],
   PlotStyle → Map[ColorData[96], Range[Length@uniqueLabels]],
   BoxRatios → 1,
   Axes → None,
   Boxed → True,
   AspectRatio → 1
  ]
 ]
```

In[45]:=
```
embeddingProject = visualize3D[features, labels];
imageProject = visualize3D[dataset[All, "Image"], labels];
```

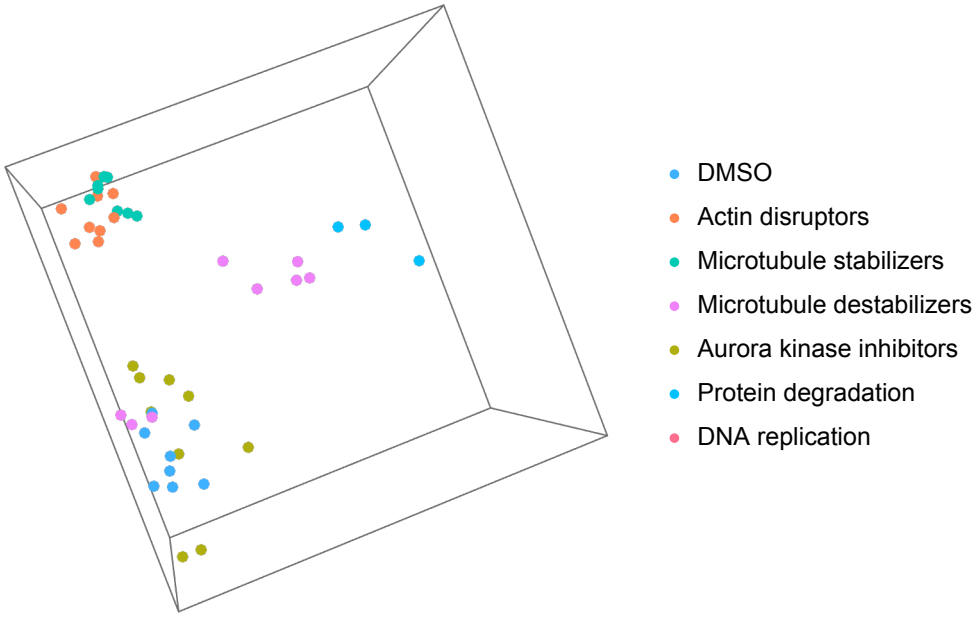
In[47]:=
```
embeddingProject
```

Out[47]=



In[48]:=
```
imageProject
```

Out[48]=



In[66]:=
```
Export[StringJoin[{outputPath, "3Dproject", "embedding_project3d.dxf"}],
  embeddingProject];
Export[StringJoin[{outputPath, "3Dproject", "image_project3d.dxf"}],
  imageProject];
```

Export : The Export element Graphics3D contains a malformed data structure and could not be exported to DXF format.

Export : The Export element Graphics3D contains a malformed data structure and could not be exported to DXF format.

Import : File /home /luhanc /vf38 _scratch /luhanc /micro /output /trained _network.wlnet not found during Import.

Cluster analysis

In[•]:=
```
clusters =
  ClusterClassify[dataset[All, "Image"], Length@Counts[dataset[All, "MOA"]]];
```

In[•]:=
```
clusters[dataset[1, "Image"], "Probabilities"]
```

Out[•]=
$$\langle | 1 \to 1.,\ 2 \to 1.10977 \times 10^{-154},\ 3 \to 6.8565 \times 10^{-134},\ 4 \to 1.71803 \times 10^{-92},$$
$$5 \to 1.55247 \times 10^{-61},\ 6 \to 1.75808 \times 10^{-48},\ 7 \to 2.33761 \times 10^{-55},\ 8 \to 3.46232 \times 10^{-45},$$
$$9 \to 1.31704 \times 10^{-34},\ 10 \to 5.38385 \times 10^{-46},\ 11 \to 1.59894 \times 10^{-30},\ 12 \to 2.80968 \times 10^{-39} | \rangle$$