Illinois Institute of Technology

Advanced Artificial Intelligence Techniques for Slot-Die Coating Process

Mrinalini, A20348759, mnln17@hawk.iit.edu

August 13, 2021

Chicago, IL,USA

Data Science Practicum - CSP 572

Prof. Shlomo Argamon

Table of Contents

1.Team Composition

1.1.Sponsor

Argonne National Lab – Dr. Zhengchun Liu(Data Science and Learning) , Dr. Yuepeng Zhang(Applied Material Division)

1.2.Illinois Tech

Illinois Institute of Technology – Dr. Shlomo Argamon , Mrinalini LNU

2.Project Background

2.1. Sponsor's Line of Business

Argonne National Laboratory is a science and engineering research national laboratory operated by UChicago Argonne LLC for the United States Department of Energy. The facility is located in Lemont, Illinois, outside of Chicago, and is the largest national laboratory by size and scope in the Midwest.

2.2. Description of Sponsor's Problem

Our aim is to establish a smart manufacturing project which aims to use Machine Learning techniques to automatically optimize the slot die coating process. We have a physical model of the slot-die machine in the materials laboratory.

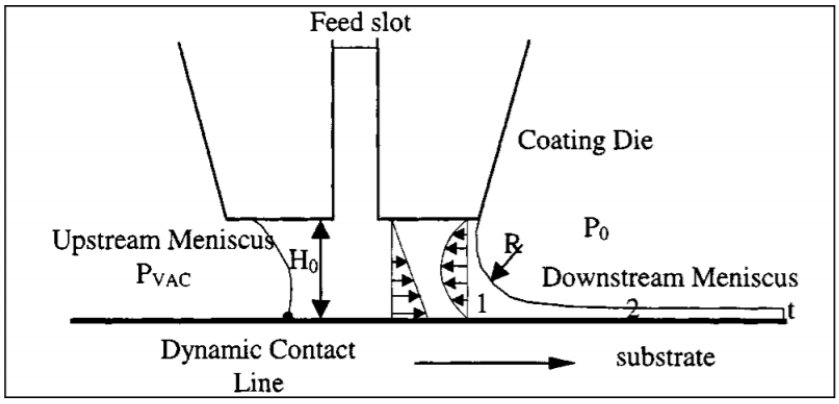
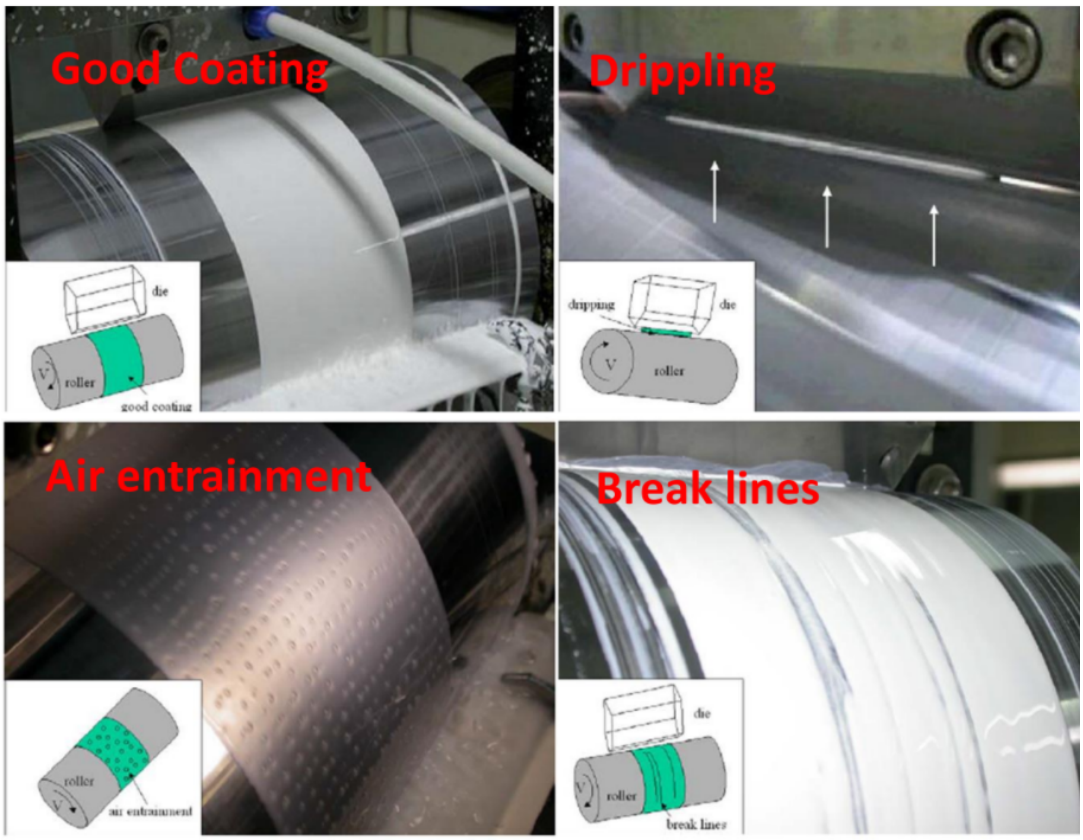


Figure 1. Coating bead of a slot coater.

At a given coating gap, H0, and constant speed, V, the flow rate associated with the Couette contribution is constant. As the film thickness t (and therefore the flow rate) decreases, the Poiseuille contribution (and therefore the adverse pressure gradient in the downstream meniscus region) has to increase. The minimum flow rate possible (minimum film thickness) is determined by the maximum pressure gradient possible at the downstream meniscus. The pressure gradient can be estimated by analyzing the pressure difference between points 1 and 2. It is approximately determined by the radius of curvature of the liquid-gas interface, R, and the surface tension of the liquid.

We still only partially understand the concepts that determine the operating limit.



W.-B. Chu et al. / Journal of Colliod and Interface Science 297 (2006) 215-225

The region of acceptable quality in the space of operating parameters of a coating process is called a coating window [4].. Their limits are set by coating defects. For the slot-die coating process the low-flow limit is important. It corresponds to the maximum web speed at a given film thickness, or the minimum film thickness at a given web speed, at which the coating bead remains stable. Other factors that affect the operating limits are coating speed, flow rate, vacuum pressure,coating gap,liquid viscosity and surface tension.

We have created a digital twin for this machine to apply Machine Learning techniques and optimize output by hyper-parameter tuning. We need to integrate the data from the physical machine into the digital twin and get real-time classification outputs of types of defects vs input parameters. This would help us compare experimental and theoretical results so that we can maximize non-defective coating.

3. Problem Objective

The following objectives were delivered:

3.1.Objective 1:

Research methods to extract, load and transform data from the High-speed camera.
We used scikit, numpy, pandas, keras to extract image data into our desired format.

3.2.Objective 2:

Using Machine Learning and Computer Vision techniques such as Convolution Neural Networks to predict image outcome (i.e., defects classification).We used Transfer learning techniques (from VGG19, Resnet50, Inception v3 trained with ImageNet dataset) to build our defect classification model to predict our defect class.

3.3.Objective 3(in-progress):

In order to address the challenge of training generalizable neural network with limited labelled images, we also research Autoencoder - loss function, activation function, padding, stride, filters channels of the convolutional neural network as well as regularization term to the latent variable to extract relevant features from the image. We then feed latent variable to downstream Neural Networks to detect and classify defects..

Data Source

We want to use SC2+edgertronic and microscopic cameras to capture live images from the slot-die coating machine.

Data Description

Since we don’t have live data, we are using 220 sample microscopic(SEM) images. 50% of these images are labelled normal while the other half is defective. Each image is 6-7mb in size(2048X2048), so in total 1.5GB.

We hope to capture live data in 10,000+ samples, so our database would be >700GB.

This data was labelled manually which took considerable time. Therefore we planned to incorporate self-supervised learning techniques when we implement our algorithm on the live data.

4. Project Deliverables

4.1.Description of Deliverables

The deliverables contain the following :

- Project Scope Document
- Github Repository containing the code
- Transfer Learning and Autoencoders
- Project Report

All the above deliverables and results are shared with Zhengchun through mail as well as explained verbally in weekly meetings and presentations at Argonne National Laboratory.

4.2.Availability of Deliverable

The GitHub repository where all the code related to the project is located is as follows:- https://github.com/mrinalinigarg/Advanced-AI

5. Project Complications and Constraints

5.1.Project Constraints

The project team(I) was researching Deep Learning techniques and suitable cameras for the first three weeks while we waited for the Non-Disclosure Agreement to be signed and executed by Argonne National Lab.

For this project one major limitation was the programming environment. The RAM space available was simply not large enough, as such the amount of data that could be processed at one time was severely constrained. JLSE resources weren't made available to me, so I submitted a request with a detailed project description for the Chameleon cloud resource to IIT. I did get access during the second month of the project but it had no VM with good amount of space and computation power free.Moreover we are required to add image data one-by-one which isn’t efficient. Therefore, I ultimately had to use Google colab for GPU/TPU resources.

The 12 GB ram was not large enough for 220 images training, so I had to purchase Collab Pro to increase RAM to 35 GB.

Because of Covid, the project was remote. No physical access to actual experiment or lab. This made collaborating hard with the teammates at Argonne.

We did not have access to real data because we did not have hardware/cameras to capture images from the experiment. We used 220 sample microscopic images to train our models. We only captured live images in the last week of the project.

5.2.Project Complications

The current data sample is small, therefore we suggested using transfer learning from imagenet dataset. We implemented VGG19, Resnet50, googlenet(inception3). Imagenet data is for natural images, our images are special cases therefore it wasn't a good fit.

Next we tried to get our own model for transfer learning – we don't have a verified microscopic image data set to utilize transfer learning on .

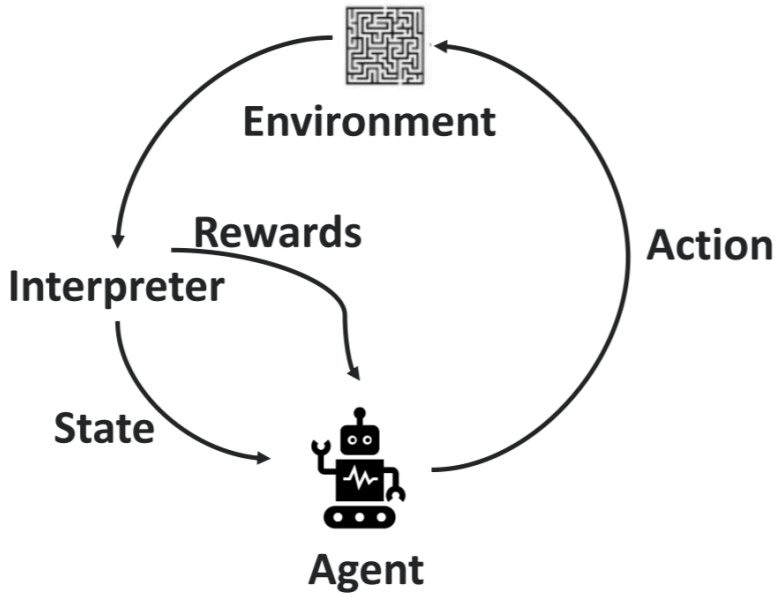
It is labor intensive to label large amounts of data, e.g., comparable to ImageNet. In order to mitigate the problem of data scarcity, we tried both transfer learning and self-supervised learning (i.e., Autoencoder). MSE cannot detect small errors, therefore we want to be able to use only the encoder part with regularization. This will generate latent variables, to extract useful features from images, then use it in the Neural Network for defects classification.Our goal is to keep the bottleneck of autoencoders as small as possible, so that only key representation of the images are kept in the latent variable. This part requires calculating our own loss function,which requires more research and time than currently available.

6.Project Execution

6.1.Reinforcement Learning – An Introduction

In a standard RL, an agent interacts with an environment in discrete timesteps. At each timestep, given an agent starting at a state value, the agent will make actions and be trained on a reward and punishment mechanism.

The agent is rewarded for “correct” actions and punished for the “wrong” ones. In doing so, the agent needs to figure out how to achieve maximum overall discounted rewards by exploring optimal behaviors to avoid wrong moves.



Reinforcement Learning Elements For Slot-Die Coating

States – Ink Properties

• Shear rate, viscosity, surface tension, particle size and shape aspect ratio, etc.
• States are predefined but affect coating quality (will not be tuned during learning)
• A latent representation of meniscus from a microscope.

Actions - Coating Parameters

• Ink flow rate, web transfer speed, coating gap, etc.
• Actions will be explored during learning based on the return of reward

Reward mechanisms

• Reward agent: correct actions lead to defect-free coatings
• Punish agent: wrong actions result in defective coatings

Our goal is to find an optimal policy that maximizes rewards by delivering as much as high-quality coatings

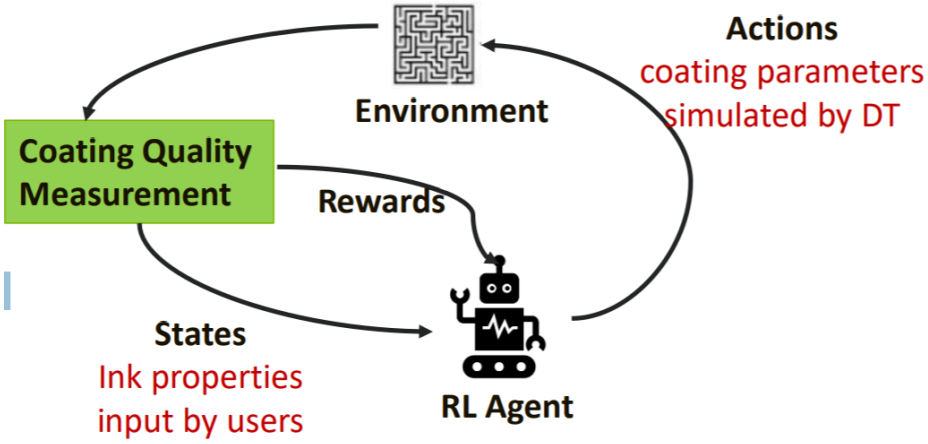
Previous version of Digital Twin: How Does Digital Twin Work?

Instead of using physical machines, we can use digital twins (DT) to simulate action and state values for RL agent training:

- Digital twin can simulate values for states and actions
- Provide initial starting points and save costs on real-time experiments
- No need to be 100% accurate because RL will fine-tune results on real machines

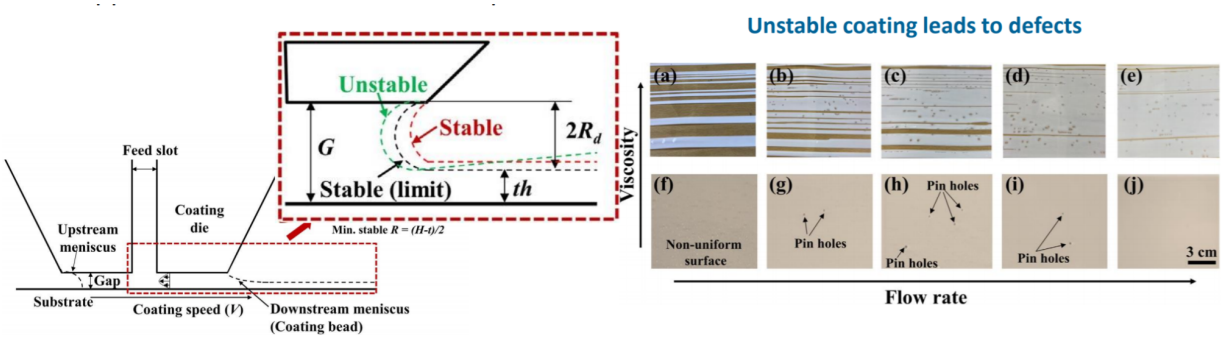
Approaches

• Coded physical models into python script to simulate defects and non-defects inputs
• Machine learning algorithms will be used to explore data patterns if physical models are not available



DT Development For A Newtonian Liquid Ink

- DT was built based on Minimum Permissible Wet Thickness (MPWT) model (Lee et al., 2019)
- Simulate stable and unstable coating conditions for different coating parameter ranges
- Ink materials: Yttria-stabilized-zirconia(YSZ) nanoparticle inks
- Application limits to Newtonian liquid



Lee, J., Kim, S., & Lee, C. (2019). Large area electrolyte coating through surface and interface engineering in roll-to-roll slot-die coating process. *Journal of Industrial and Engineering Chemistry, 76*, 443-449. doi:10.1016/j.jiec.2019.04.011

Minimum Permissible Wet Thickness Model

Pressure gradient in terms of curvature of downstream meniscus and surface tension of ink

$$\Delta P = Pi + Pa = \frac{\sigma}{Rd} \quad (1) \qquad \mathrm{Rd} = \frac{G - th_{m,w}}{2} \quad (2)$$

Pressure gradient according to the coated layer thickness, surface tension, and viscosity of the ink

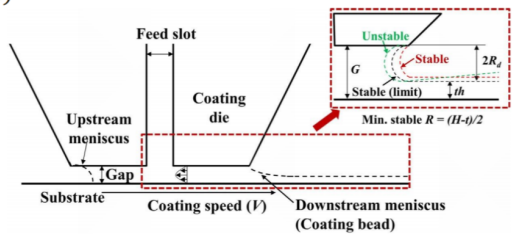
$$\Delta P = 1.34 * Ca^{\frac{2}{3}} * \frac{\sigma}{th_{m,w}} \quad (3)$$

Substitute equation 2,3 into 1:

$$\boldsymbol{th_{m,w}} = \frac{\boldsymbol{G}}{\frac{\boldsymbol{2}}{\boldsymbol{1.34 * Ca^{\frac{2}{3}}}} + \boldsymbol{1}} \quad (\boldsymbol{4})$$

Predict geometry of the coated layer using

$$\dot{m}_c = \frac{d}{dt}\int \rho f(x,t)dx = \rho(f_r - nd(th_{i,w})v) \quad (5)$$



Lee, J., Kim, S., & Lee, C. (2019). Large area electrolyte coating through surface and interface engineering in roll-to-roll slot-die coating process. *Journal of Industrial and Engineering Chemistry, 76*, 443-449. doi:10.1016/j.jiec.2019.04.011

6.2.New Development on Digital Twin: Overview Of Image Classification

• An artificial intelligence (AI) that trains computers to interpret and understand the visual world
• With digital images and deep learning models, machines can accurately identify and classify Objects

Image classifications use computer vision algorithms to classify images to replace human efforts in the coating quality checking stage of RL.

Image classification
• A process of predicting of a specific class, in a supervised manner
• A subset of the classification problem, where an entire image is assigned with a label

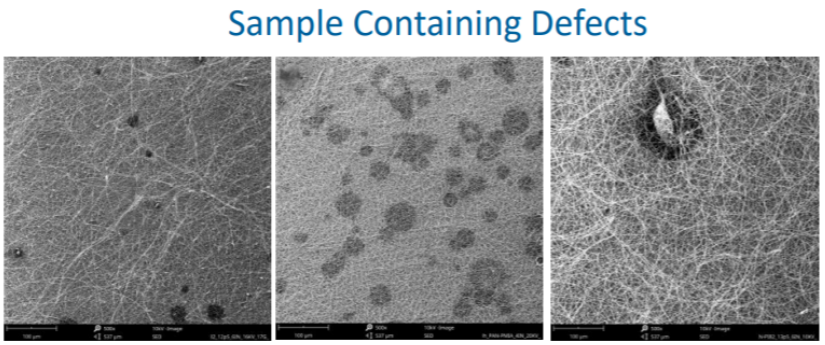
Task:
Identify defects from 222 SEM images. Electrospinning samples were used due to lack of coating data; Learning established during this study can and will be transferred to coating images

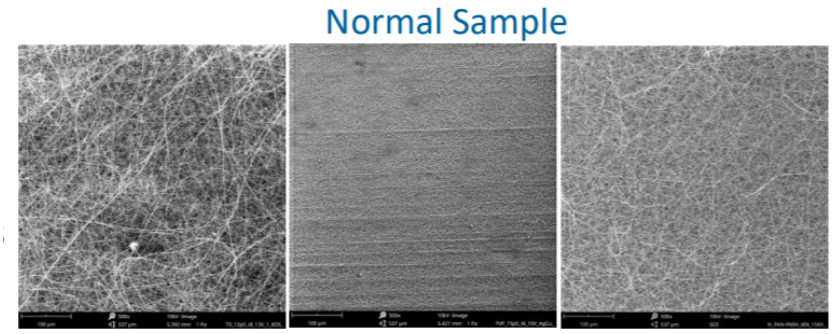
Challenges :

- Accuracy score only reached 65%.
- Hyper-parameter tuning, such as number of hidden layers, number of units.Improvement on accuracy scores was not ideal.
- Our image size is too small to train a robust neural network.
- We need more data.

6.3.Transfer Learning For Images Classification

- Transfer learning allows us to transfer knowledge from other related tasks into our problem.
- Our images include lines and bubbles, which are common shapes in other pre-trained tasks.
- Reduce data amounts and running time required for learning.

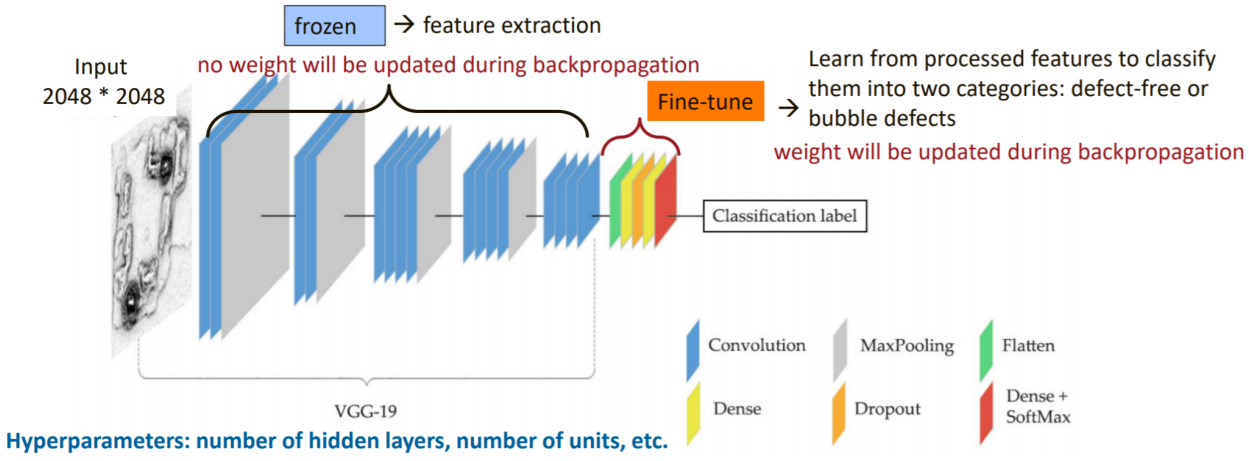




6.3.1.VGG19

- We transferred pre-trained knowledge in VGG19 to our project
- Two label classifications with 177 images for training and 45 images for validation
- Image size is 2048 * 2048, pixel density ranges from 0 to 255 (0 is black, 255 is white)

- VGG19 requires 3 channels because that's how the model is pre-trained, so have to convert our data to 2048X2048X3



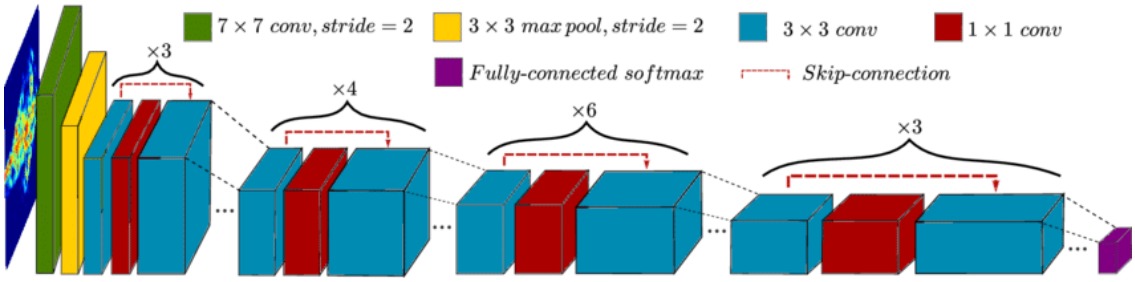
6.3.2.Resnet50

- The 2048X2048 image size was eating up all the RAM and I wasn't able to train the model. Google colab kept on crashing
- Reduced number of training epochs
- Reduced the number of images used for training
- Adding custom layers

```
#Adding custom Layers
x = model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation="relu")(x)#changed from 512, 256
x = Dense(512, activation="relu")(x)#changed from 512
x=Dropout(0.2)(x)
x = Dense(128, activation="relu")(x)
x = Dense(56, activation="relu")(x)
predictions = Dense(1, activation="sigmoid")(x)
# creating the final model
model_final = Model(model.input, predictions)
model_final.compile(loss='binary_crossentropy',
                    optimizer='adam',
                     metrics=['accuracy'])#change for F1score
                    # options = run_opts,
                    # run_metadata=runmeta)
```

- Validation Accuracy looks really bad!

```
Epoch 1/15
24/24 - 110s - loss: 1.0912 - accuracy: 0.5139 - val_loss: 0.9238 - val_accuracy: 0.5789
Epoch 2/15
24/24 - 107s - loss: 1.2563 - accuracy: 0.4306 - val_loss: 0.6139 - val_accuracy: 0.5789
Epoch 3/15
24/24 - 108s - loss: 0.7711 - accuracy: 0.5278 - val_loss: 0.6344 - val_accuracy: 0.6842
Epoch 4/15
24/24 - 108s - loss: 0.6605 - accuracy: 0.5694 - val_loss: 0.6258 - val_accuracy: 0.6842
Epoch 5/15
24/24 - 110s - loss: 0.6802 - accuracy: 0.5833 - val_loss: 0.6197 - val_accuracy: 0.5263
Epoch 6/15
24/24 - 109s - loss: 0.6510 - accuracy: 0.6111 - val_loss: 0.6209 - val_accuracy: 0.4737
Epoch 7/15
24/24 - 110s - loss: 0.6471 - accuracy: 0.6250 - val_loss: 0.6324 - val_accuracy: 0.4737
```

- 

6.3.3.GoogleNet(Inception3)

- The 2048X2048 image size was eating up all the RAM and I wasn't able to train the model. Google colab kept on crashing
- I reduced the size to 512X512
- Reduced the number of images used for training
- Unfreezing last 5 layers of the pre-trained model

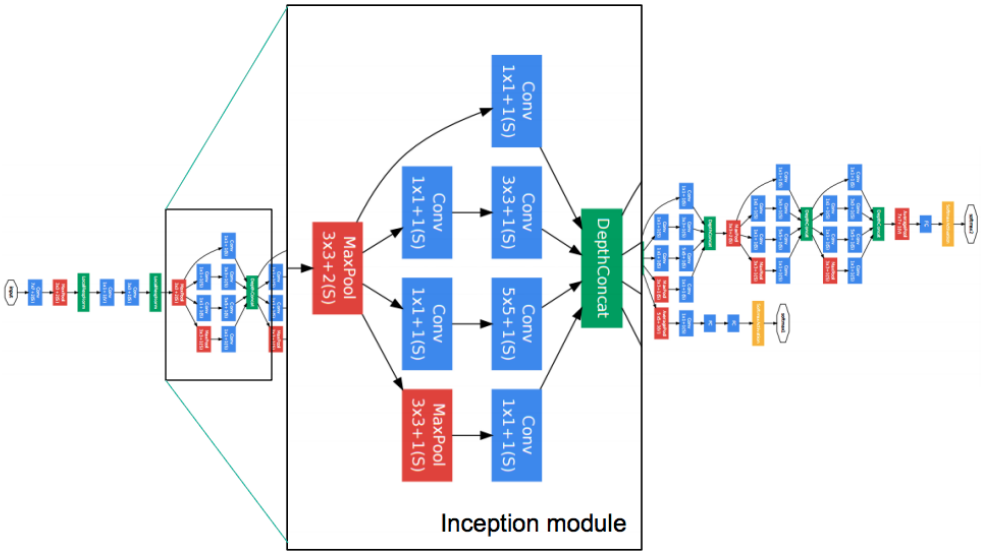
```
for layer in pre_trained_model.layers[0:305]:#total 311 layers for Inception3, so just keeping last 5 layer flexible
#for layer in pre_trained_model.layers:
    layer.trainable = False
count=1
for layer in pre_trained_model.layers:
    count+=1
    print(count,layer, layer.trainable)
```

- Adding my custom layers to the model.

```
from tensorflow.keras import layers
from tensorflow.keras import Model
# Flatten the output layer to 1 dimension
x = layers.Flatten()(last_output)
# Add a fully connected layer with 512 hidden units and ReLU activation
x = layers.Dense(512, activation='relu')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a final sigmoid layer for classification
x = layers.Dense(1, activation='sigmoid')(x)

# Configure and compile the model
model = Model(pre_trained_model.input, x)
model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.0001),
              metrics=['acc'])
```

- With each epoch validation accuracy increased: >85%

```
Epoch 1/15
59/59 - 162s - loss: 22.7036 - acc: 0.6667 - val_loss: 8.6069 - val_acc: 0.6667
Epoch 2/15
59/59 - 151s - loss: 8.8830 - acc: 0.7910 - val_loss: 56.2354 - val_acc: 0.4889
Epoch 3/15
59/59 - 153s - loss: 7.2286 - acc: 0.8531 - val_loss: 5.1820 - val_acc: 0.8444
Epoch 4/15
59/59 - 150s - loss: 1.6562 - acc: 0.8983 - val_loss: 4.6210 - val_acc: 0.8444
Epoch 5/15
59/59 - 161s - loss: 0.9886 - acc: 0.9548 - val_loss: 6.5808 - val_acc: 0.8667
Epoch 6/15
59/59 - 154s - loss: 1.1490 - acc: 0.9492 - val_loss: 4.8671 - val_acc: 0.8444
Epoch 7/15
59/59 - 153s - loss: 0.6284 - acc: 0.9492 - val_loss: 6.9270 - val_acc: 0.8667
Epoch 8/15
59/59 - 151s - loss: 0.2882 - acc: 0.9718 - val_loss: 5.8352 - val_acc: 0.8667
Epoch 9/15
59/59 - 158s - loss: 0.2034 - acc: 0.9831 - val_loss: 5.9733 - val_acc: 0.8667
```



Conclusions with Transfer Learning:

Resnet50 uses Imagenet data, which is for natural images, our images are special cases therefore it wasn't a good fit.
We saw promising results with the Googlenet model.

Next we tried to get our own model for transfer learning – we don't have a verified microscopic image data set to train weights on. Therefore has to drop this idea.

Since we expect to have large amounts of unlabelled data – we try to use autoencoders to generate our models.

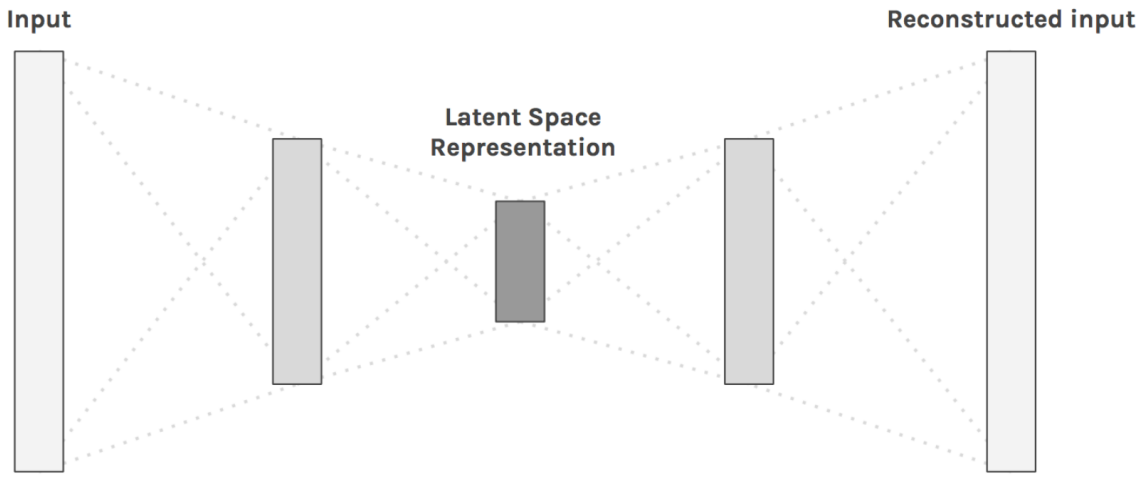
6.4.What are autoencoders?

Autoencoders are artificial neural networks[5], trained in an unsupervised manner, that aim to first learn encoded representations of our data and then generate the input data (as closely as possible) from the learned encoded representations. Thus, the output of an autoencoder is its prediction for the input.

The primary applications of an autoencoder is for anomaly detection or image denoising. We know that an autoencoder's task is to be able to reconstruct data that lives on the manifold i.e. given a data manifold, we would want our autoencoder to be able to reconstruct only the input that exists in that manifold[6]. Thus we constrain the model to reconstruct things that have been observed during training, and so any variation present in new inputs will be removed because the model would be insensitive to those kinds of perturbations.

Another application of an autoencoder is as an image compressor. If we have an intermediate dimensionality d lower than the input dimensionality n, then the encoder can be used as a compressor and the hidden representations (coded representations) would address all (or most) of the information in the specific input but take less space.

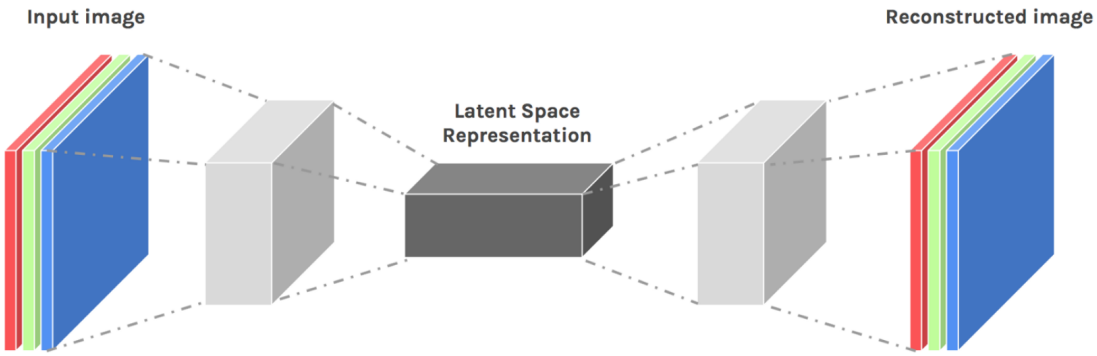
Autoencoders (AE) are a family of neural networks for which the input is the same as the output*. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation.



Convolutional Autoencoders[10]

A really popular use for autoencoders is to apply them to images. The trick is to replace fully connected layers by convolutional layers. These, along with pooling layers, convert the input from wide and thin (let's say 100 x 100 px with 3 channels—RGB) to narrow and thick. This helps the network extract visual features from the images, and therefore obtain a much more accurate latent space representation. The reconstruction process uses upsampling and convolutions.

The resulting network is called a Convolutional Autoencoder (CAE)[9].



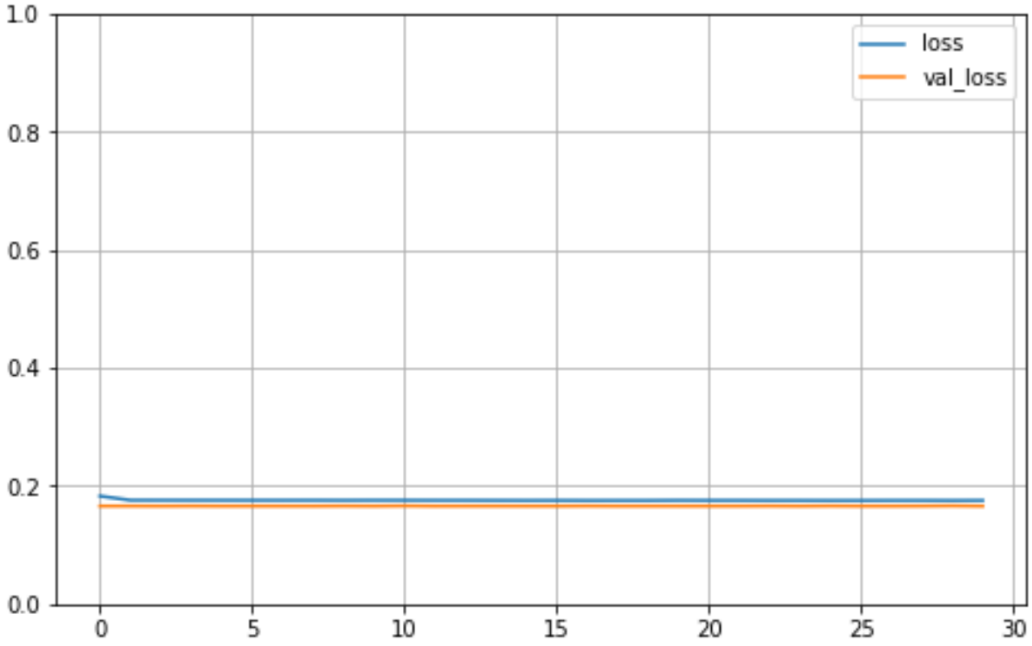
6.4.1.Using Dense Layers:

- We had to change the input dimensions to 512X512X1. Earlier we had changed the input to 3 channels because our pre-trained network in VGG19, Resnet50 and Googlenet has 3 channels. So there was input mismatch
- Make sure that padding was kept “same” so that the layer dimensions matched.
- Had to take care of the activation function and the loss function to match our desired out of regression on 2 classes.
- When we try to train an image using dense layers, we do not get good reconstructions:

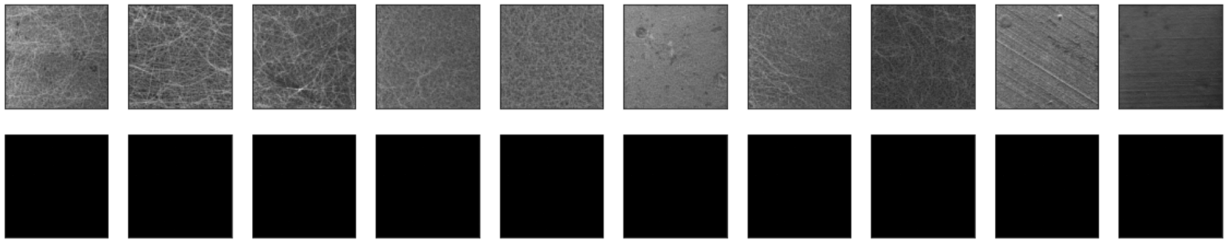
```
encoder_input = keras.Input(shape=(512, 512, 1), name='img')
x = keras.layers.Flatten()(encoder_input)
x = keras.layers.Dense(1048, activation="relu")(x)
x = keras.layers.Dense(512, activation="relu")(x)
encoder_output  = keras.layers.Dense(64, activation="relu")(x)
encoder = keras.Model(encoder_input, encoder_output, name='encoder')

decoder_input = keras.layers.Dense(64, activation="relu")(encoder_output)

x = keras.layers.Dense(512, activation="relu")(decoder_input)
x = keras.layers.Dense(1048, activation="relu")(x)
x = keras.layers.Dense(786432/3, activation="relu")(x)

decoder_output = keras.layers.Reshape((512, 512, 1))(x)

opt = tf.keras.optimizers.Adam(lr=0.001, decay=1e-6)

autoencoder = keras.Model(encoder_input, decoder_output, name='autoencoder')
```

```
img (InputLayer)             [(None, 512, 512, 1)]     0
_________________________________________________________________
flatten (Flatten)            (None, 262144)            0
_________________________________________________________________
dense (Dense)                (None, 1048)              274727960
_________________________________________________________________
dense_1 (Dense)              (None, 512)               537088
_________________________________________________________________
dense_2 (Dense)              (None, 64)                32832
_________________________________________________________________
dense_3 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_4 (Dense)              (None, 512)               33280
_________________________________________________________________
dense_5 (Dense)              (None, 1048)              537624
_________________________________________________________________
dense_6 (Dense)              (None, 262144)            274989056
_________________________________________________________________
reshape (Reshape)            (None, 512, 512, 1)       0
=================================================================
Total params: 550,862,000
Trainable params: 550,862,000
Non-trainable params: 0
_________________________________________________________________
```

- Training and Validation loss



- Original Vs Reconstructed Images

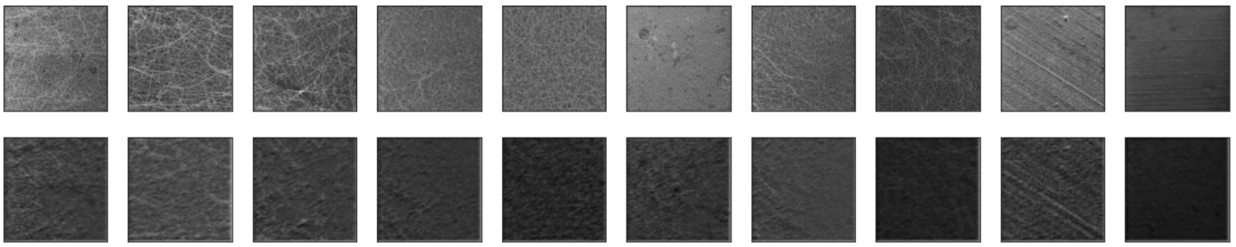
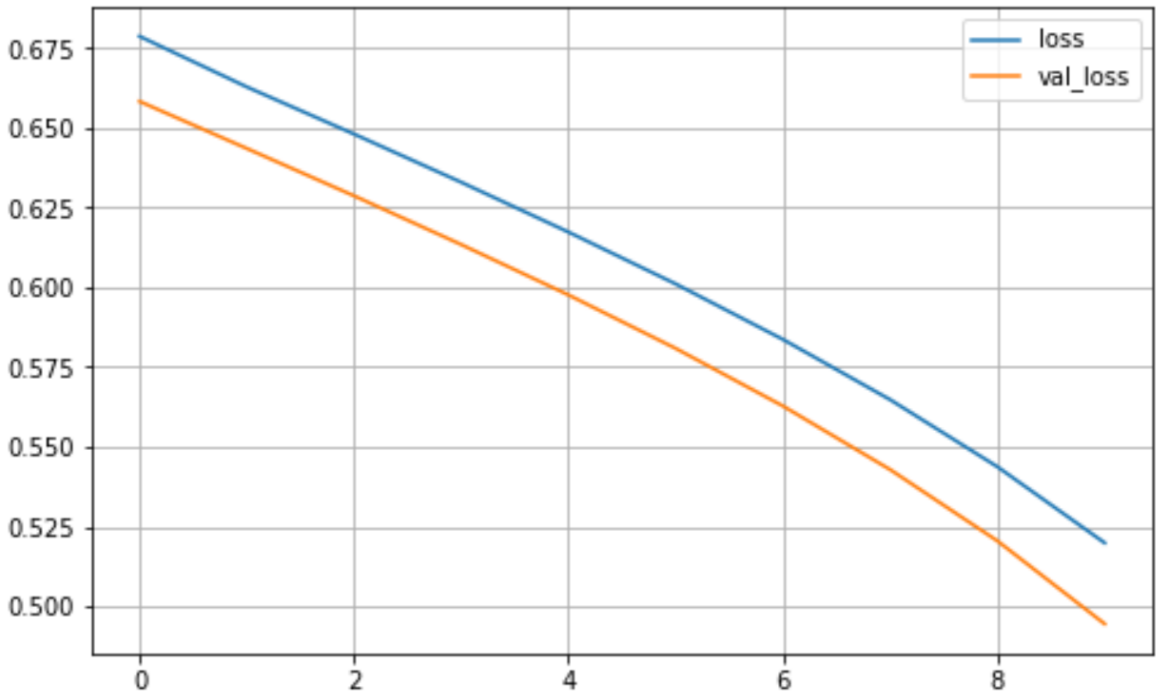


6.4.2.Using Convolution Layers:

- Using CNN in autoencoders for images

```
def create_layers():
  layers = []
  size = 32

  #encoder layers
  for i in range(0, 3):
    x = Conv2D(size, (3, 3), activation='relu', padding='same')
    layers += [x]
    x = MaxPooling2D((2, 2), padding='same')
    layers += [x]
    size = size // 2


  #deocder layers
  for i in range(0, 3):
    size = size * 2
    if i == 2:
      x = Conv2D(size, (3, 3), activation='relu', padding='same')
    else:
      x = Conv2D(size, (3, 3), activation='relu', padding='same')
    layers += [x]
    x = UpSampling2D((2, 2))
    layers += [x]
```

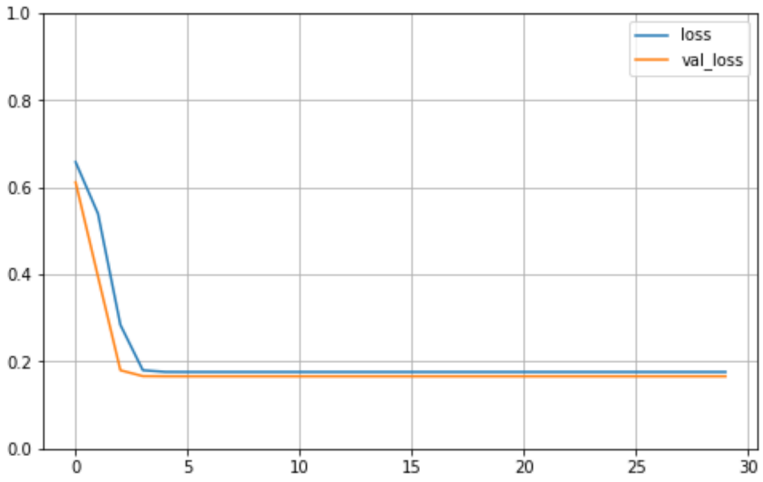
```
  x = Conv2D(1, (3, 3), activation='sigmoid', padding='same')
  layers += [x]

  return layers

def autoencoder():
  input_img = Input(shape=(512, 512, 1))

  layers = create_layers()

  #create the auto encoder network
  x = input_img
  for layer in layers:
    x = layer(x)

  autoencoder = Model(input_img, x)
  #autoencoder.compile(optimizer = 'adam', loss = 'binary_crossentropy')
  autoencoder.compile(optimizer = 'sgd', loss = 'mse')
```

- When using sigmoid, we need to rescale our input data to [0, 1] using min-max normalization
- We get pretty good reconstruction of images
- 
- This is still 32X32X4 in the bottle next. Our next goal is to make it as small as possible.
- Training loss vs validation loss, looks pretty stable!
- 

- When we use optimizer=adam

```
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(4, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional
x = layers.Conv2D(4, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu',padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

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

- Training loss vs validation loss



Conclusions:

Autoencoders.We see promising results(high accuracy) with Googlenet and Autoencoders(reconstructed Images). MSE cannot detect small errors, we can try using adversarial networks. They can enhance features. Our goal is not to create perfect image reconstructions with autoencoders, but to preserve maximum information in the latent space. Our goal is to keep the bottleneck of autoencoders as small as possible, so that the minimum amount of data can represent our images . Autoencoders [1] are useful for learning to encode observable data into a latent space of smaller dimensionality and thus perform dimensionality reduction (manifold learning). However, the latent variable space often lacks structure [3] and it is impossible, by construction, to sample from the data distribution. We want to be able to use only the encoder part with regularization[2]. We want to use only Linear activation functions , not Relu. This is true only for the output layer when MSE loss is used, i.e., treat reconstruction as a regression problem.This will generate latent variables, to extract useful features from images, then use it in the Neural Network for defects classification. This part requires calculating our own loss function,which requires more research and time than currently available.

References

[1] Y. Bengio et al. Learning deep architectures for AI. Foundations and trends R in Machine Learning, 2(1):1–127, 2009
[2]https://arxiv.org/abs/2004.05485
[3] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research, 11(Dec):3371–3408, 2010
[4](https://aiche.onlinelibrary.wiley.com/doi/10.1002/aic.690461003)
[5]https://cs.stanford.edu/~quocle/tutorial2.pdf
[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998.
[7] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders. arXiv preprint arXiv:1511.05644, 2015.
[8]X. Chen, D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel. Variational lossy autoencoder. arXiv preprint arXiv:1611.02731, 2016.
[9]O. Fabius and J. R. van Amersfoort. Variational recurrent auto-encoders. ArXiv e-prints, Dec. 2014
[10]I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.