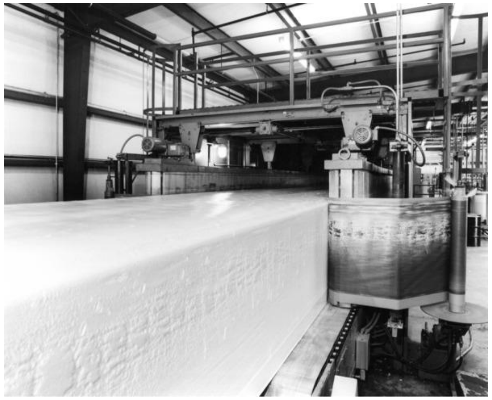
Computer Vision

EXPLORING THE MICROSTRUCTURE OF POLYURETHANE FOAMS: A DEEP LEARNING APPROACH

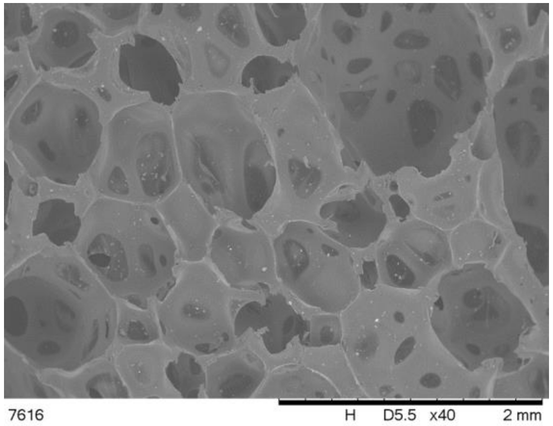
Dawid Walicki

Introduction

- Polyurethane foams are cellular materials made by reacting polyols and isocyanates, which are used in a variety of applications for their lightweight and insulating properties.
- Understanding the properties of polyurethane foams and their relation to the material's microstructure is critical in optimizing their performance in various applications.
- Scanning Electron Microscope (SEM) images are used to analyze the microstructure of the foam.
- The study aims to use Deep Learning to analyze SEM images to predict the mechanical properties of the foam, particularly the 40% tension.



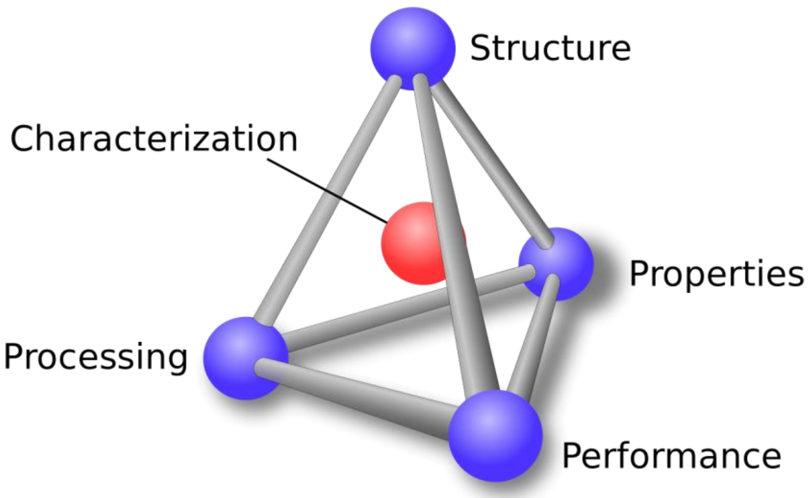
Flexible polyurethane foam manufacturing line



One of the samples (SEM images) used for DL model

Why Explore the Microstructure of Polyurethane Foams?

- understanding the relationship: microstructure and material properties
- the power of machine learning: predicting mechanical properties from SEM images
- efficiency and cost-effectiveness in R&D: minimizing experimental tests



Key Tools and Technologies

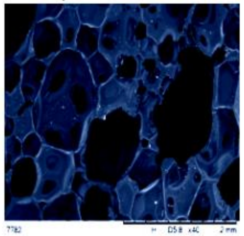
- Python: the backbone of the analysis
- PyTorch: models building and training
- efficiency and cost-effectiveness in R&D: minimizing experimental tests
- Google Colab: cloud-based GPUs

Decoding polyurethane foam: dataset description

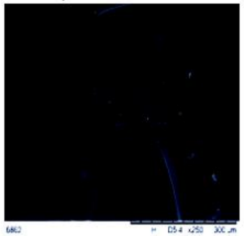
- Dataset comprised of SEM images and mechanical properties
- 19 different polyurethane foams formulations
- focus on 40x magnified images
- mechanical property: 40% tension

sample_index	sample_name	SAG	40%_tension	pHRR	T_2%	U600	start_time	expansion_time	gel_time
1	AS1A	2,58	2,58		234	4,1	12	257	570
2	AS2	2,5	2,32	165	193	3,5	17	320	500
3	AS3	2,66	2,66	109	228	7,4	10	410	510
4	AS4	2,88	2,88	74	225	10,4	9	410	740

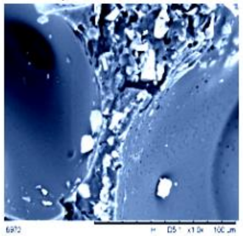
Property value: 3.07
shape: [224, 224, 3]



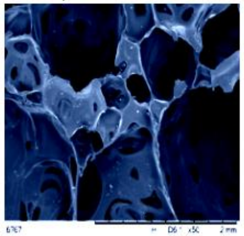
Property value: 3.16
shape: [224, 224, 3]



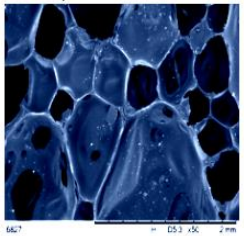
Property value: 3.68
shape: [224, 224, 3]



Property value: 2.19
shape: [224, 224, 3]



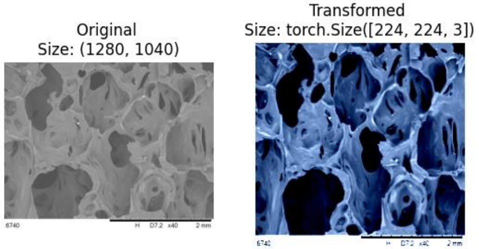
Property value: 2.99
shape: [224, 224, 3]



Decoding polyurethane foam: the experimentation

- dataset creation using python ‘os’ library
- data visualization for better understanding of dataset
- data transformation into tensors using PyTorch
- data batching for efficient computation
- training and testing loops with early stopping mechanism
- application of transfer learning with pretrained models

```
def create_dataset(PATH: str, data: pd.DataFrame, output_property: str):
    '''This function creates dataset - each sample with different magnifications is
      connected with chosen property'''
```




```
data_transform = transforms.Compose([
    transforms.Resize(size=(224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

```
train_dataloader = DataLoader(dataset=train_data,
                              batch_size=64,
                              num_workers=1,
                              shuffle=True)
```

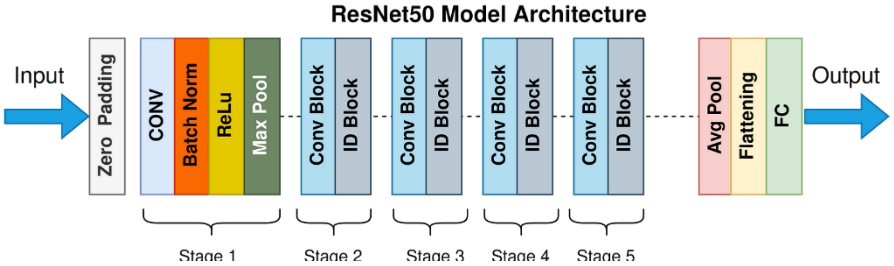
```
def train_and_test(model: torch.nn.Module,
                   train_dataloader: torch.utils.data.DataLoader,
                   test_dataloader: torch.utils.data.DataLoader,
                   loss_fn: torch.nn.Module,
                   optimizer: str,
                   device: torch.device,
                   save_model_name: str,
                   learning_rate: int = 0.1,
                   epochs: int = 5,
                   patience: int = 3):
```

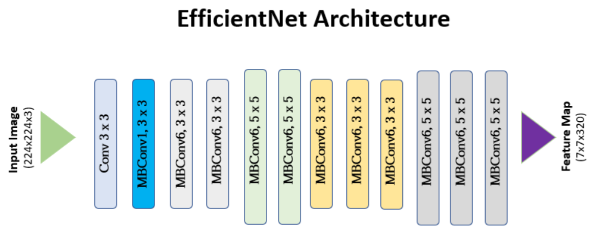
```
resnet50 = models.resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)
num_ftrs = resnet50.fc.in_features
resnet50.fc = nn.Linear(num_ftrs, 1) #adjusting model to my output, regression problem
```

Harnessing pretrained models and custom architecture

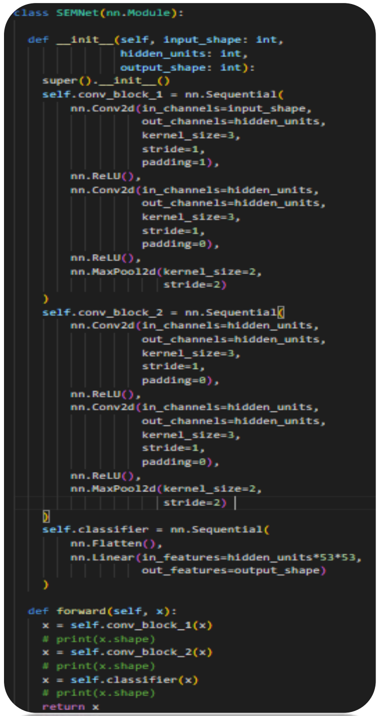
TRANSFER LEARNING:

RESNET50, RESNET34, AND EFFICIENTNET





CUSTOM MODEL



```python
class SEMNet(nn.Module):

  def __init__(self, input_shape: int,
               hidden_units: int,
               output_shape: int):
    super().__init__()
    self.conv_block_1 = nn.Sequential(
        nn.Conv2d(in_channels=input_shape,
                  out_channels=hidden_units,
                  kernel_size=3,
                  stride=1,
                  padding=1),
        nn.ReLU(),
        nn.Conv2d(in_channels=hidden_units,
                  out_channels=hidden_units,
                  kernel_size=3,
                  stride=1,
                  padding=0),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2,
                     stride=2)
    )
    self.conv_block_2 = nn.Sequential(
        nn.Conv2d(in_channels=hidden_units,
                  out_channels=hidden_units,
                  kernel_size=3,
                  stride=1,
                  padding=0),
        nn.ReLU(),
        nn.Conv2d(in_channels=hidden_units,
                  out_channels=hidden_units,
                  kernel_size=3,
                  stride=1,
                  padding=0),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2,
                     stride=2)
    )
    self.classifier = nn.Sequential(
        nn.Flatten(),
        nn.Linear(in_features=hidden_units*53*53,
                  out_features=output_shape)
    )

  def forward(self, x):
    x = self.conv_block_1(x)
    # print(x.shape)
    x = self.conv_block_2(x)
    # print(x.shape)
    x = self.classifier(x)
    # print(x.shape)
    return x
```

Experimentation and analysis

- property predictions: exploring multiple properties (searching for structure – properties relationships)
- data transformations and augmentation: adding richness to the dataset
- hyperparameter tuning:
  - batch sizes: 1 to 64
  - learning rates: 0.00001 to 0.1
  - loss functions: Mean Squared Error (MSE), L1Loss, SmoothL1Loss
  - optimisers: Stochastic Gradient Descent (SGD), Adam, RMSprop
  - convolutional network hidden layer sizes in SEMNet

```
learning_rate: int = 0.1,
```

```
learning_rate = 0.00001
```

```
data_transform_augmentation = transforms.Compose([
    transforms.Resize(size=(224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(20),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

```
batch_size=8,
```

```
loss_fn = nn.MSELoss()
```

```
batch_size=64,
```

```
loss_fn = nn.SmoothL1Loss()
```

```
model = SEMNet(input_shape=3, hidden_units=48, output_shape=1)
```

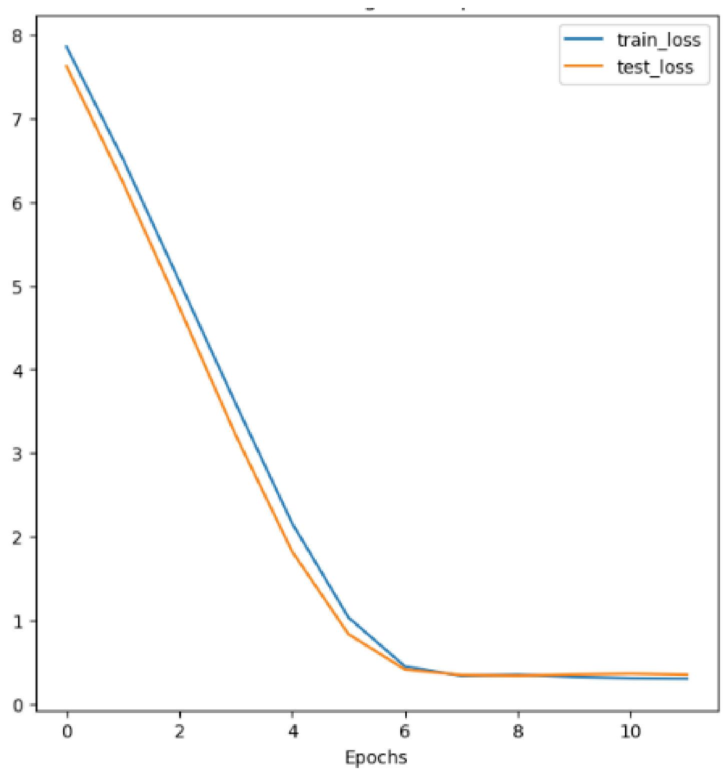
```
optimizer = torch.optim.SGD(params=model.parameters(),
                            lr=learning_rate)
```

```
optimizer = torch.optim.Adam(params=model.parameters(),
                             lr=learning_rate)
```

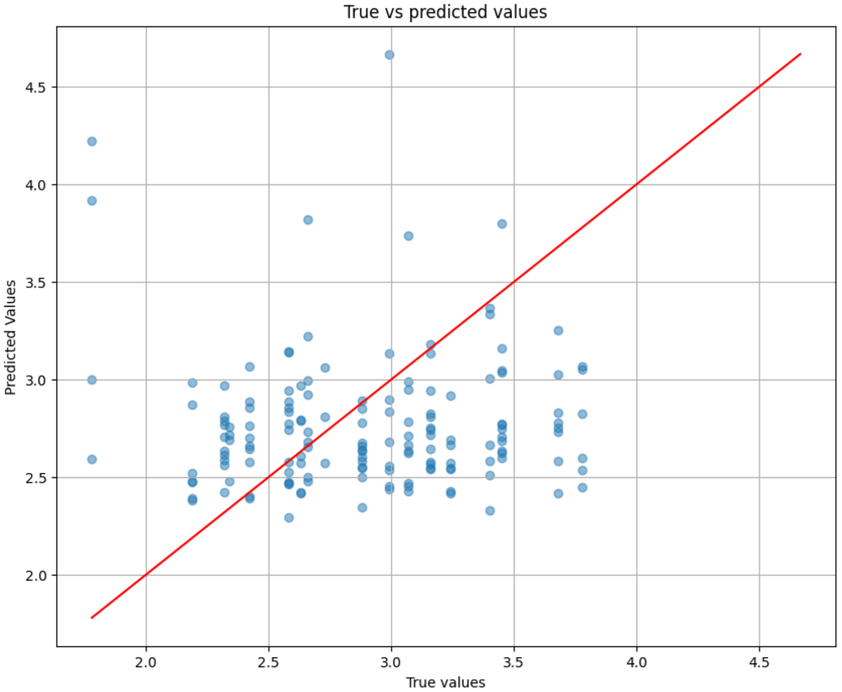
```
optimizer = torch.optim.RMSprop(params=model.parameters(),
                                lr=learning_rate)
```

Model Evaluation

LEARNING CURVES



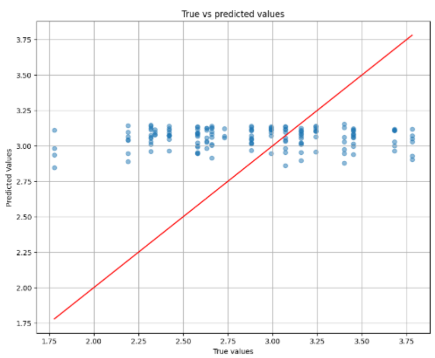
PREDICTED VS TRUE VALUES VISUALISATION

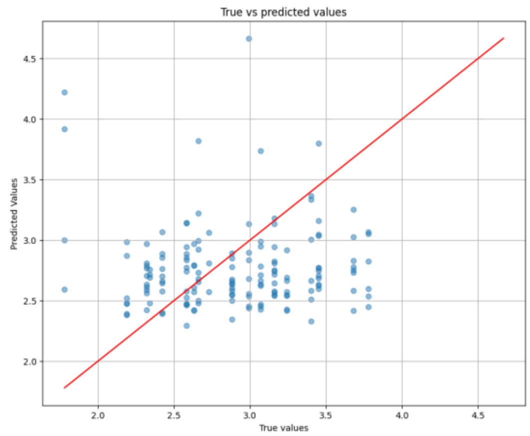


Results and findings

- promising potential for Computer Vision in analyzing microstructure
- custom model: the best performance in predicting the 40% tension property
- hyperparameter optimization: a challenging but necessary task
- optimal learning rate: 0.0001, optimizer: Adam, loss function: MSE

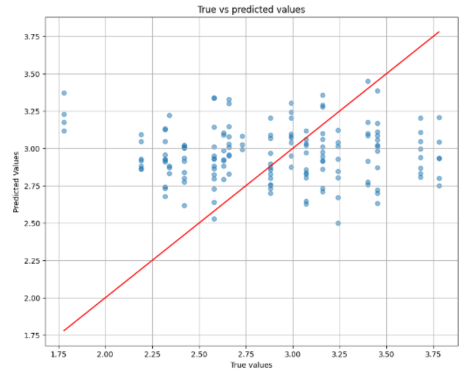
Resnet50 (basic)





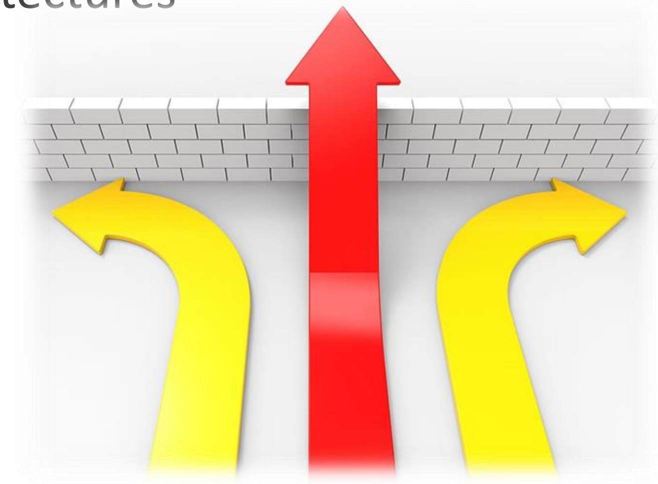
Custom model

Resnet50 (with data augmentation)



Limitations and future scope

- Models are not without their limitations: limited dataset, lack of property diversity, inherent complexity of phenomena studied
- These challenges underline the importance of further investigation and refinement
- Future work: more diverse dataset, advanced model tuning, exploration of different network architectures



Thank you for your attention

Dawid Walicki

https://github.com/DWalicki95/CV_microstructure_images