CH5650 Term Paper

Name: Abhiram S
Roll no.: CH19B037

Problem Statement

Predict the **glass transition temperature** (T_g) of polymers, based only on the structure on their monomers, using a **convolutional neural network** (CNN). Link to paper.

Dataset

The dataset used consists of 104 polymers, with the **SMILES** codes of their monomers along with their glass transition temperatures. There is an extended dataset, containing 196 polymers. However, the smaller dataset was used for this term paper, as it contained polymers from a few classes only, with less variability in T_g. Thus, training and prediction was relatively easier and not as time consuming.

Approach

Data Preprocessing

The SMILES strings were converted into a format which can be fed into a CNN as follows:

1. First, a **dictionary** consisting of all possible characters in a SMILES code is defined as follows: [‘c’, ‘n’, ‘o’, ‘C’, ‘N’, ‘O’, ‘F’, ‘P’, ‘S’, ‘Cl’, ‘Br’, ‘I’, ‘0’, ‘1’, ‘2’,‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘.’, ‘-’, ‘ ¼ ’, ‘#’, ‘$’, ‘:’, ‘/’, ‘þ’, ‘)’, ‘(’, ‘@’, ‘{’, ‘}’,‘\’, ‘ ’, ‘[’, ‘]’]
2. Now, each SMILES code is converted into a binary matrix, i.e., consisting only of zeros and ones via **one-hot-encoding**: the i^{th} row of the matrix is filled with zeros except for the position of the dictionary which coincides with the same character on the i^{th} position in the monomer SMILES code. A one is placed in that case.
3. Thus, the dimensions of the binary matrix for each SMILES code is **($npos_{max}$, nd)**, where nd is the number of characters in the dictionary and $npos_{max}$ is the length of the longest SMILES code in the dataset.
4. In order to feed these matrices into a CNN, the number of channels must be specified. Since this is a binary matrix, the **number of channels is 1**. Thus, an extra dimension is added to each matrix and reshaped as **($npos_{max}$, nd, 1)**.
5. Thus, the final dataset of features is of shape **(mx, $npos_{max}$, nd, 1)**, where mx is the total number of polymers in the dataset (104). The target values (T_g) is in the form of a column vector of shape **(mx, 1)**.

For example, the image for poly(benzyl acrylate) [SMILES: C=CC(=O)OCc1ccccc1] is as follows:

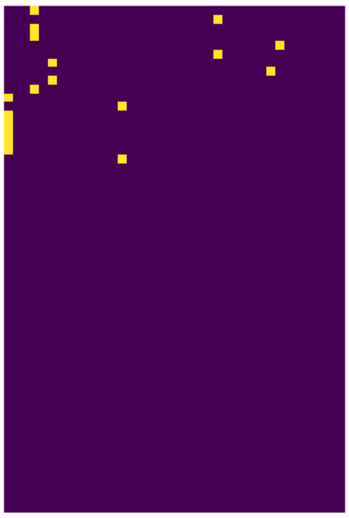

Fig. 1. Image for poly(benzyl acrylate)

The images for all the polymers in the dataset are given below:

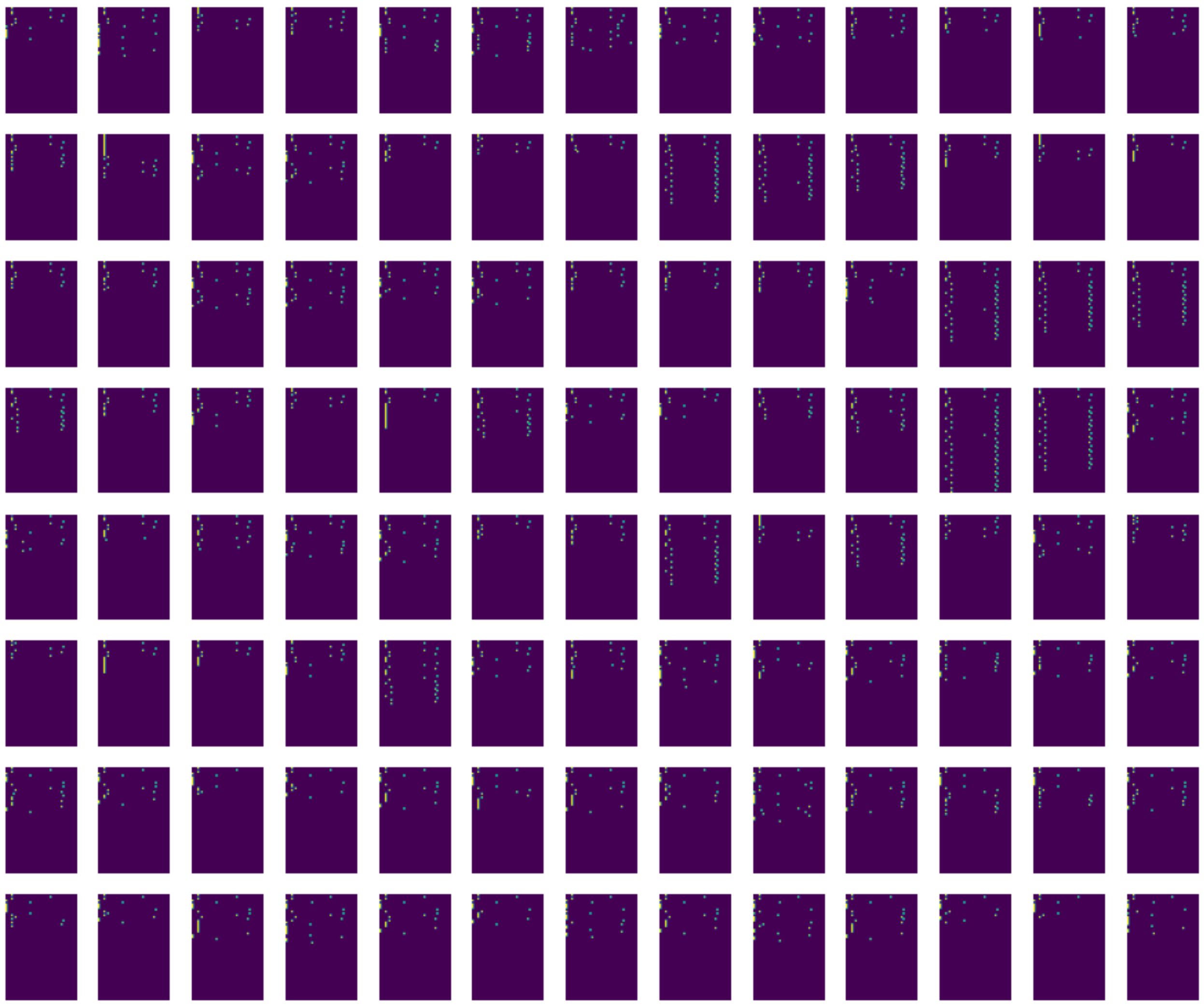
Fig. 2. Images of all monomer SMILES strings

CNN Model Architecture

The CNN model architecture used was same as given in the paper. **Tensorflow** with **Keras** API was used to build this model.

1. **Zero padding layer** for uniform padding of width 5. Input size = ($npos_{max}$, nd, 1)
2. **2D convolutional layer 1** with 256 filters of size (3, 3), followed by batch normalisation, ReLU activation and (3, 3) max pooling.
3. **2D convolutional layer 2** with 128 filters of size (3, 3), followed by batch normalisation, ReLU activation and (3, 3) max pooling.
4. **Flattening** the output of the second 2D convolutional layer.
5. **Densely connected layer** of 100 neurons, followed by batch normalisation, ReLU activation and dropout with rate = 0.2.
6. Finally, a **single neuron** with linear activation, which gives the output (T_g).

The model summary, with the layer wise and total number of trainable and non-trainable parameters are given below:

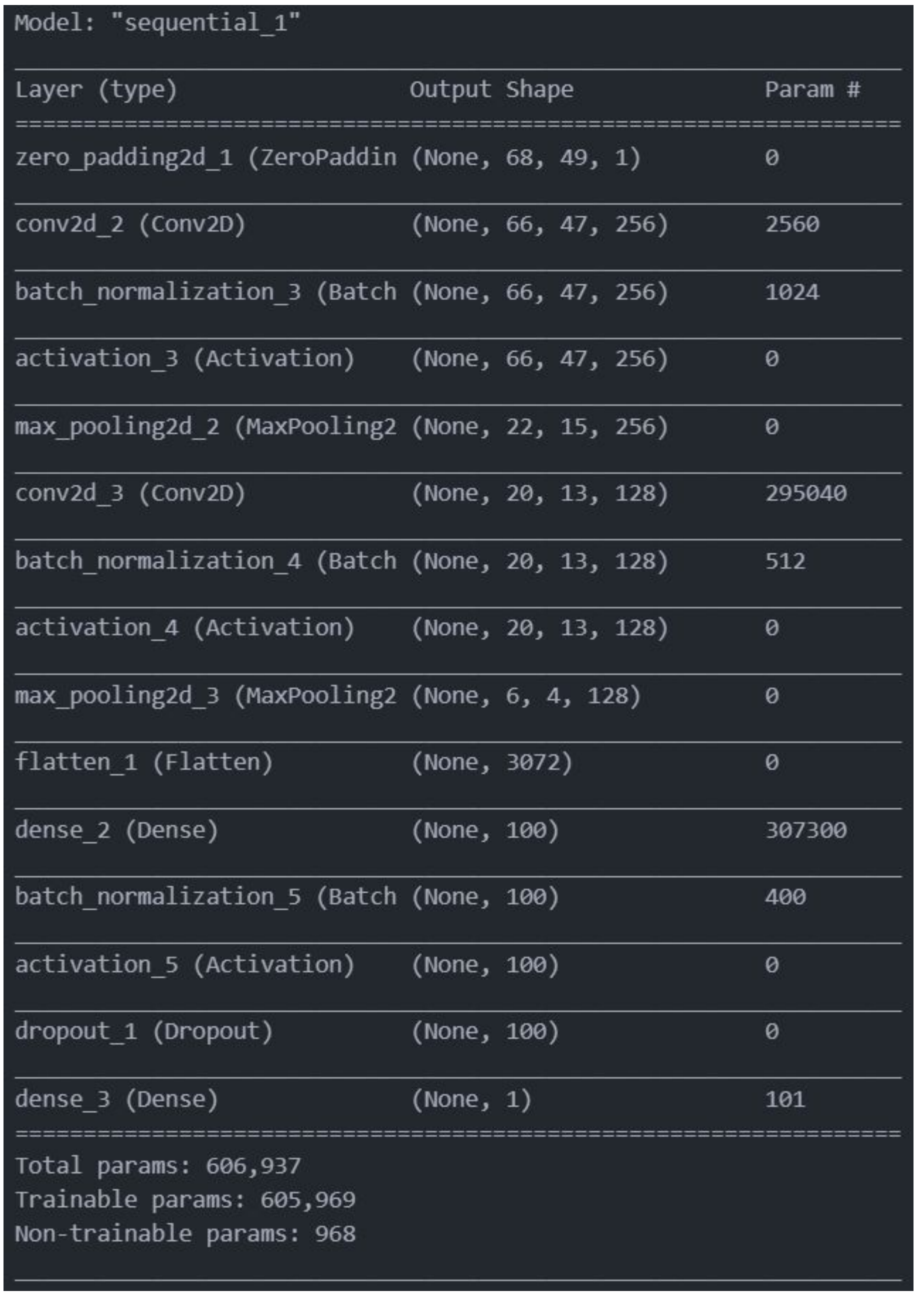
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_1 (ZeroPaddin (None, 68, 49, 1)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 66, 47, 256)       2560
_________________________________________________________________
batch_normalization_3 (Batch (None, 66, 47, 256)       1024
_________________________________________________________________
activation_3 (Activation)    (None, 66, 47, 256)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 22, 15, 256)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 20, 13, 128)       295040
_________________________________________________________________
batch_normalization_4 (Batch (None, 20, 13, 128)       512
_________________________________________________________________
activation_4 (Activation)    (None, 20, 13, 128)       0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               307300
_________________________________________________________________
batch_normalization_5 (Batch (None, 100)               400
_________________________________________________________________
activation_5 (Activation)    (None, 100)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 606,937
Trainable params: 605,969
Non-trainable params: 968
_________________________________________________________________
```

Fig. 3. Model summary

Model Training

1. The loss function used for training the model is the **mean absolute percentage error**, defined as:

$$\text{Loss} = \frac{100}{\text{mx}} \sum_{i=1}^{\text{mx}} \left| \frac{A_i - F_i}{A_i} \right|$$

   Where A_i is the actual and F_i is the predicted T_g.
2. An **Adam optimiser** with the default values (learning rate = 0.001, β_1 = 0.9 and β_2 = 0.999) was used to minimise the loss function.
3. A **90/10 train test split** was done before model training. Model training was done for **500 epochs**, with **mini batch size of 15** and a **validation split of 0.1**.

4. After training, predictions were made on both the train and test sets and the **loss and R^2** were calculated for both sets.
5. Model training was performed on an **NVIDIA GPU** with the help of **CUDA** and **cuDNN**.

Results

The loss and R^2 for training and sets sets are summarised in the following table:

	Loss	R^2
Training Set	2.99%	0.964
Test Set	5.47%	0.814

Table 1. Results summary

The training and validation loss as a function of epochs is as follows:

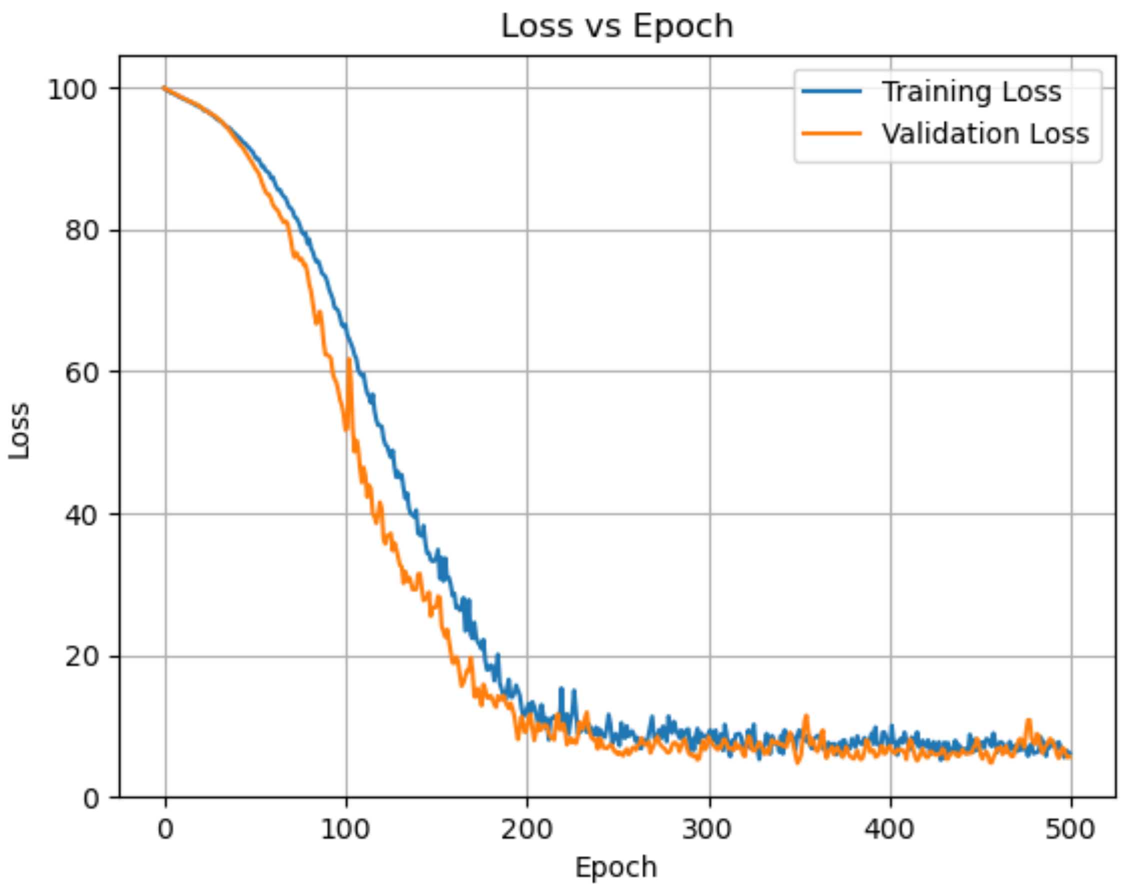

Fig. 4. Training and validation loss history

The parity plot for both the training and test sets is as follows:

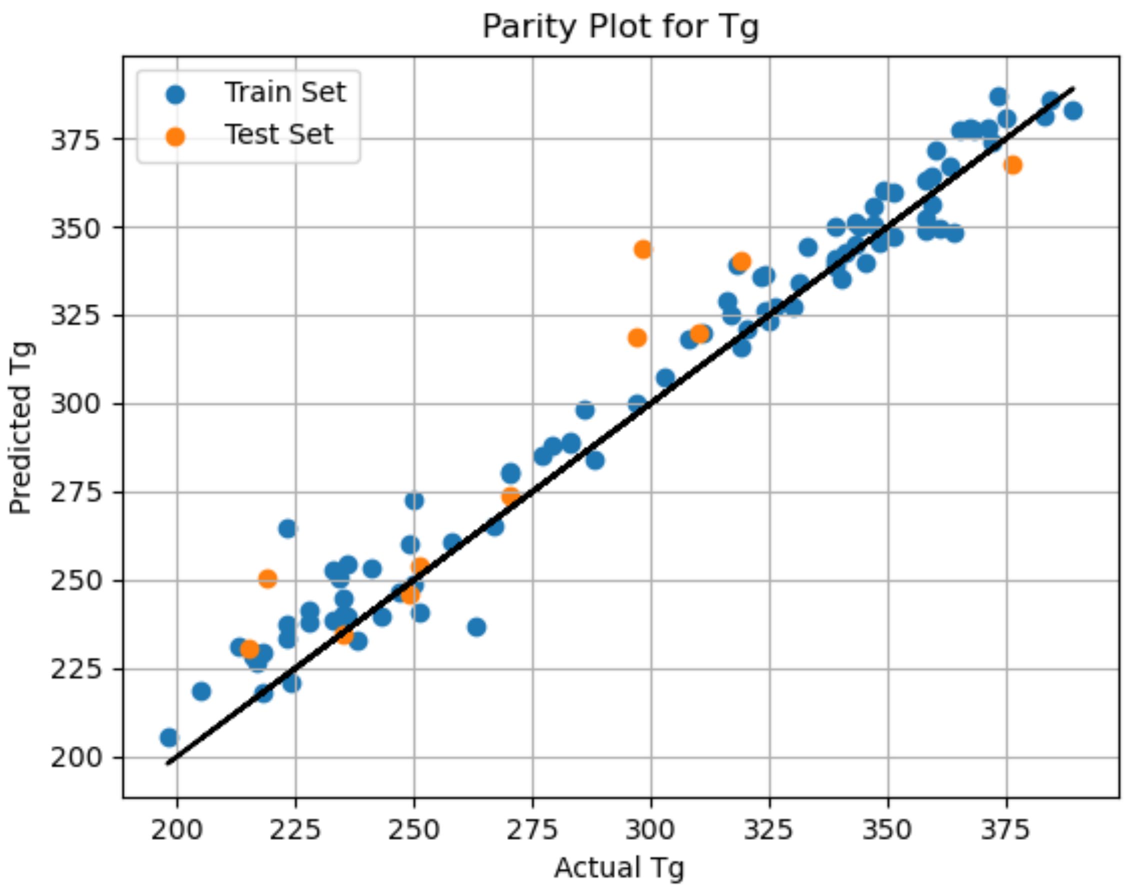

Fig. 5. Parity plot for training and test sets

Discussion

1. The **error** is around **3%** for the training set and **5.5%** for the test set, which indicates a decent fit. The relatively low R^2 values are due to the small size of the training and test sets.
2. Running the model repeatedly can give different outputs. This is due to **inherent stochasticity** in the CNN, for e.g. during the initialisation of weights.
3. **Shuffling** the dataset before splitting into train and test sets can also affect the output. For instance, if the "more difficult" examples end up in the test set, the model would perform worse than usual.
4. The performance is also affected by the **order of the dictionary**, since it is essentially arbitrary. Different permutations of the dictionary will yield different images, and hence different CNN outputs.
5. The effect of all these can be countered by training for different shuffling of data and dictionary orders and **aggregating** the results at the end.
6. **Hyperparameter optimisation** can also be done to improve performance. Some of the hyperparameters that can be tuned are:
   i. Number of filters in 2D convolutional layer
   ii. Filter size (although this is generally (3, 3))
   iii. Max pooling window size
   iv. Zero padding width
   v. Dropout probability
   vi. Learning rate, β_1 and β_2 (for Adam optimiser)
   vii. Mini batch size
   viii. Dictionary permutation
6. Hyperparameter tuning was performed in the paper, but wasn't done here due to complexity and lack of time for training.