**Assignment 4**

**Judging a Book by its Cover**

**BRAJ RAJ NAGAR (2022AIB2682)**

**SHASHANK G (2022AIB2684)**

**Non-Competitive Part**

1. **Convolutional Neural Network:**

Implementing the **CNN architecture**

On the architecture side, we’ll be using a simple model that employs three convolution layers with depths**32, 64, and 128,** respectively, followed by two fully connected layers for performing classification.

* Each convolutional layer involves a convolutional operation involving a **5×5 convolution filter** and is followed by a ReLU activation operation for introducing nonlinearity into the system and a **max-pooling operation with a 2×2 filter** to reduce the dimensionality of the feature map.
* After the end of the convolutional blocks, we flatten the multidimensional layer into a low dimensional structure for starting our classification blocks. After the first linear layer, the last output layer(also a linear layer) has 30 neurons for each of the 30 unique classes in our dataset.

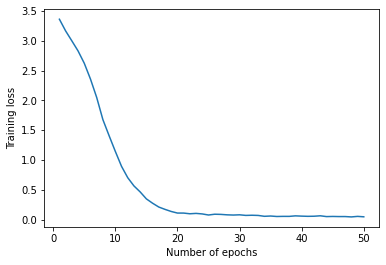
For building our model, we’ll make a **CNN class**inherited**from the *torch.nn.Module*** class for taking advantage of the Pytorch utilities. Apart from that, we’ll be using the ***torch.nn.Sequential***container to combine our layers one after the other.

* The ***Conv2D(), ReLU(),*** and ***MaxPool2D()***layers perform the convolution, activation, and pooling operations. We used padding of 1 to give sufficient learning space to the kernel as padding gives the image more coverage area, especially the pixels in the outer frame.
* After the convolutional blocks, the Linear() fully connected layers perform classification.

Defining the training parameters and beginning the training process

We begin the training process by selecting the device to train our model onto, i.e., CPU or a GPU. Then, we define our model hyperparameters which are as follows:

* We train our models over 50 epochs, and since we have a multiclass problem, we used the Cross-Entropy Loss as our objective function.
* We used the popular Adam optimizer with a learning rate of 0.001 to optimize the objective function.



**Test set accuracy = 10.543859649122806 %**

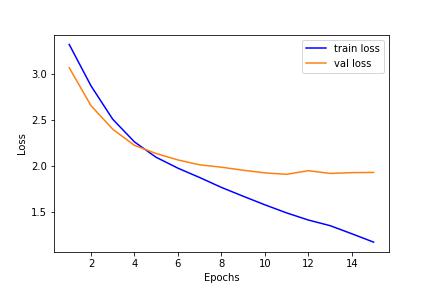
**2 . Recurrent Neural Network**

Implementing the RNN architecture

We have used a single stack of RNN layer with a hidden state size of 128 which is bidirectional. We have not used embedding layer but instead directly converted the titles into glove embeddings using torchtext library.

We then flatten the outputs of all timesteps of the RNN layer and pass it through a fully connected layer of size 128 units which is then followed by the classifying layer of 30 units.

We have used tanh as the activation function for the MLP layer. We have used a sequence length of 15 for all examples and batch size of 8192 and Adam optimizer with learning rate of 0.01. We train our model for 15 epochs.



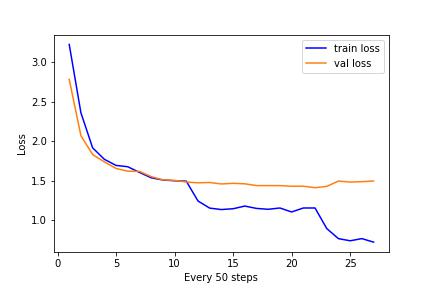
**Test set accuracy = 45.1579 %**

**Competitive part**

Bidirectional Encoder Representations from Transformers (BERT)

Bidirectional Encoder Representations from Transformers (BERT) is a natural language processing model. It is basically an encoder stack of the transformer architecture. Bert learns the language model by pre-training it on Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) which can then be fine-tuned by adding just one additional output layer to perform any specific downstream task.

We have trained it for 5 epochs with with early stopping of patience 5. We have used AdamW optimizer with learning rate of 5e-5. we have used batch size of 64 and checking and saving best model every 50 steps.



**Test set accuracy on leaderboard = 62.97%**