Lecture summarization using sequential models

Mayank Mishra (2016EE30506), Tarun Kumar Yadav (2016CS10359)

***Abstract*—Most online courses are very long and don't offer notes which makes it difficult for the students to take notes. We solve this problem by learning a sequential model trained on the video lectures. Learning to store information over extended time intervals is difficult using RNNs because of vanishing gradient problems. This makes it difficult to train vanilla RNNs. These problems are resolved by modern sequential models like LSTMs and GRUs. We train a Convolutional LSTMs on the given problem since the input to an LSTM cell is an image.**

I. Problem formulation

Basically we have the task of identifying whether the current frame should be saved (recorded) or not. We have labelled data for the given problem. This task is posed as a distribution matching problem. The goal is to learn a non-linear mapping from the space of video frames X to the space Y of binary labels. We assume that such a transformation exists that takes an image $\boldsymbol{x}_i \in X$ to its corresponding label $\boldsymbol{y}_i \in Y$. This problem can be cast into learning the parameters $\boldsymbol{\theta} \in \Theta$ that minimize the Kullback-Leibler divergence between the true distribution $q(\boldsymbol{y}|\boldsymbol{x})$ and the estimated distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})$.

$$D_{KL}\left[q(\boldsymbol{y}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})\right] = \mathbb{E}_q[\log q(\boldsymbol{y}|\boldsymbol{x})] - \mathbb{E}_q[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})] \tag{1}$$

where $\boldsymbol{x}$ denotes a frame.

Minimizing (1) with respect to $\boldsymbol{\theta}$ is equivalent to minimizing the second term on the right hand side since the first term is independent of $\boldsymbol{\theta}$. The final objective reduces to:

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} - \mathbb{E}_q[\log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})] \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathbb{E}_q[-\boldsymbol{y}\log h_{\boldsymbol{\theta}}(\boldsymbol{x}) - (1-\boldsymbol{y})\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}))]\end{aligned} \tag{2}$$

using $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) = h_{\boldsymbol{\theta}}(\boldsymbol{x})^{\boldsymbol{y}}(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}))^{1-\boldsymbol{y}}$, where $h(\boldsymbol{x})$ is given by $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{y}=1|\boldsymbol{x})$.

Minimizing the KL will definitely converge since KL between two probability distributions is lower bounded by zero, with the KL being zero when the density functions for both the distributions are equal at all points in their domain.

The result is the familiar logistic regression problem with the Cross Entropy loss function (2). Theoretically, this can be solved by any function parametrized by $\boldsymbol{\theta}$. We use neural networks for the realization of this function.

II. Convolutional LSTMs

We use the convolutional LSTM architecture that can be used for classification, regression etc. LSTMs are simple temporal models. Convolutional LSTMs only differ from LSTMs in the sense that their basic operations are convolution operations.

$$\begin{aligned}i_t &= \sigma(W_{xi} * x_t + W_{hi} * h_{t-1} + W_{ci} \circ c_{t-1} + b_i) \\ f_t &= \sigma(W_{xf} * x_t + W_{hf} * h_{t-1} + W_{cf} \circ c_{t-1} + b_f) \\ c_t &= f_t \circ C_{t-1} + i_t \circ tanh(W_{xc} * x_t + W_{hc} * h_{t-1} + b_c) \\ o_t &= \sigma(W_{xo} * x_t + W_{ho} * h_{t-1} + W_{co} \circ c_t + b_o) \\ h_t &= o_t \circ tanh(c_t)\end{aligned} \tag{3}$$

III. Experiments and results

A. Simple LSTM learned on embeddings of images

We tried to learn the embeddings of the given images using simple autoencoders. These embeddings were then used to train a simple LSTM. However, the LSTM failed to learn anything. We tried to increase the model capacity by increasing number of layers and making it a 3-layered stacked LSTM. However, the model overfitted and trivially learned to remember the training data but failed to generalize to unseen test data.

B. Convolutional LSTM on raw images

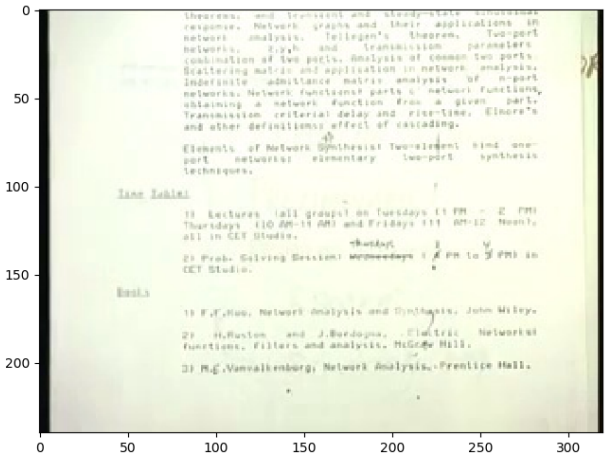
We tried to learn a 3-layered convolutional LSTM on raw images. All images were resized to (240, 320, 3) where 3 is the number of channels in the image (RGB channels). The model performance is as shown in Table I.

C. Convolutional LSTM with CNN and attention with preprocessed images

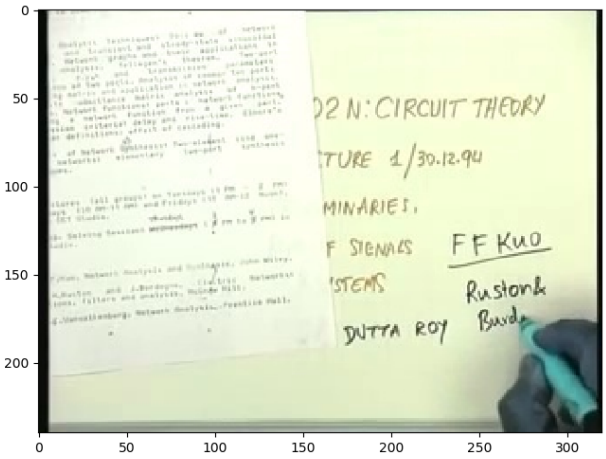
The raw images were pre-processed making them (240, 320, 6) where 6 is the number of channels in the images. We computed diff (refer Fig. 2) and optical flow using 2 consecutive images. The raw images were also passed through a Canny edge detector and the corresponding edge maps were obtained (refer Fig. 1). The images thus became 6-channeled (RGB, diff, optical flow, edgemap). These 6-channeled images were used to train a 3-layered convolutional LSTM with a simple CNN (with shared parameters across temporal dimensions) at the output of each cell in the third layer. This CNN was followed by an attention layer. The model did not perform substantially better than the simple convolutional LSTM model as shown in Table II.

D. Convolutional LSTM on Fourier transform

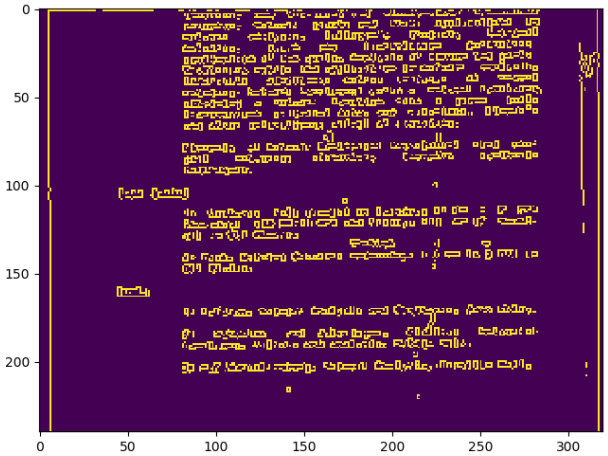
We also tried to train a simple convolutional LSTM on the Fourier transform of the images (240, 320, 2) where 2 is the number of channels (magnitude and phase spectrum). However, the model failed to learn anything even on the training data as the loss did not converge.



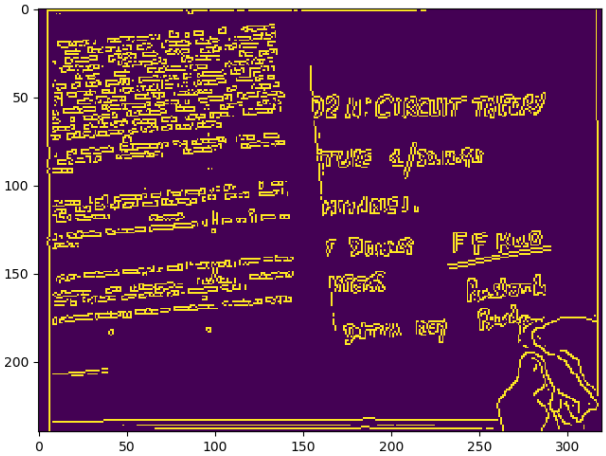
(a) Frame with label = 0



(b) Frame with label = 1

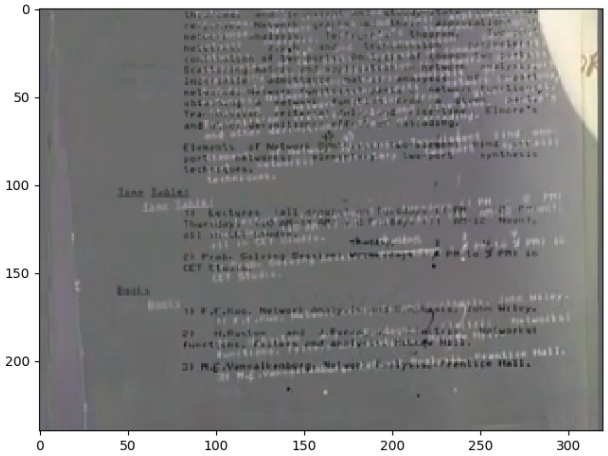


(c) Frame with label = 0

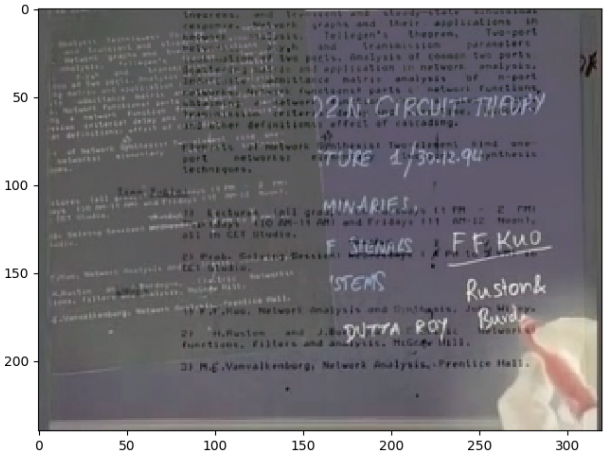


(d) Frame with label = 1

Fig. 1: Frames with their edgemaps detected using Canny edge detector



(a) when both frames have label = 0



(b) when one frame has label = 0 and one frame has label = 1

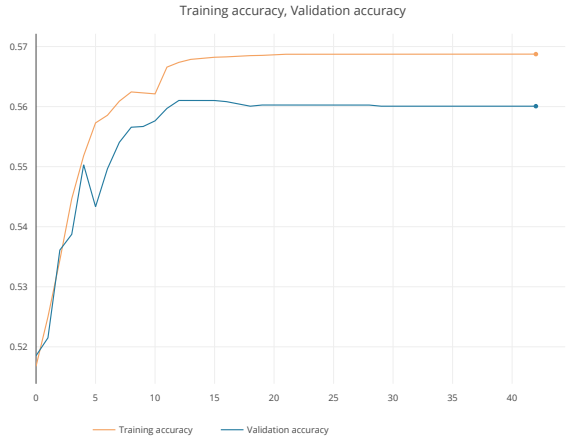
Fig. 2: Diff of 2 consecutive images

TABLE I: Metrics on simple convolutional LSTM

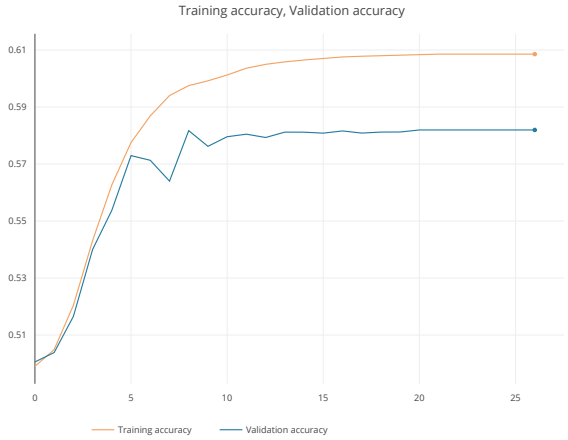
Metric	Train set	Validation set
Accuracy	56.87	56.01
F1 score	59.84	58.73

TABLE II: Metrics on convolutional LSTM with CNN at output with attention at the output of CNN

Metric	Train set	Validation set
Accuracy	60.87	58.24
F1 score	62.94	61.26



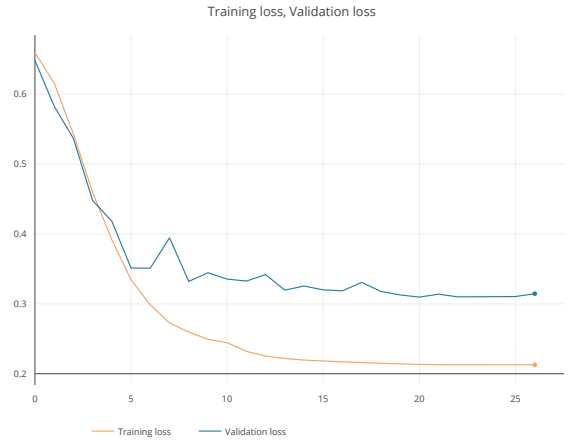
(a) Simple convolutional LSTM



(b) Convolutional LSTM with CNN and attention

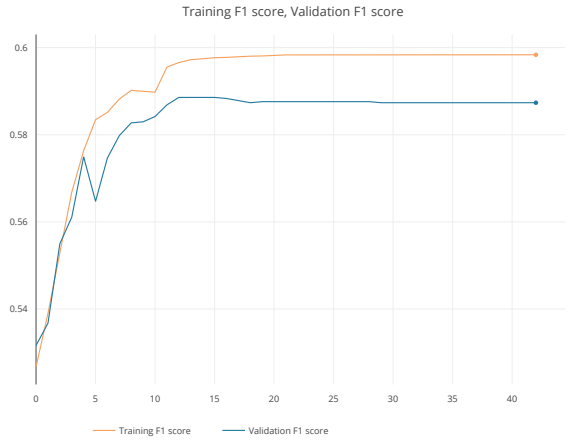
Fig. 3: Training and validation accuracy at each epoch

(a) Simple convolutional LSTM



(b) Convolutional LSTM with CNN and attention

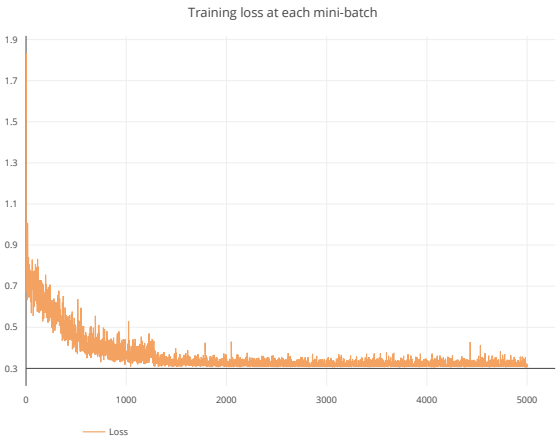
Fig. 4: Training and validation loss at each epoch



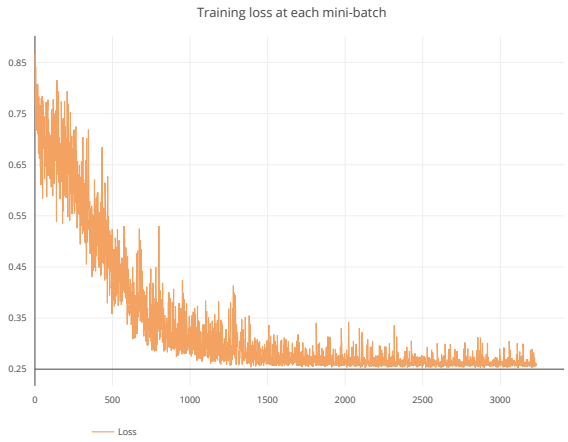
(a) Simple convolutional LSTM

(b) Convolutional LSTM with CNN and attention

Fig. 5: Training and validation F1 score at each epoch

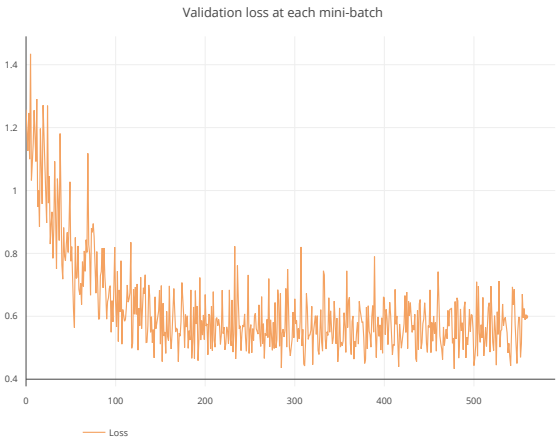


(a) Simple convolutional LSTM

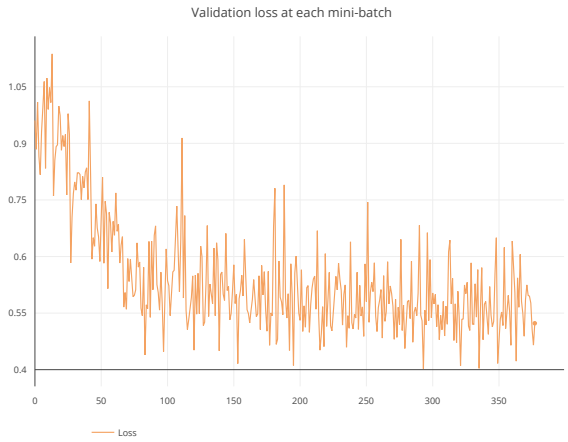


(b) Convolutional LSTM with CNN and attention

Fig. 6: Training loss at each iteration



(a) Simple convolutional LSTM



(b) Convolutional LSTM with CNN and attention

Fig. 7: Validation loss at each iteration

IV. Conclusion

In this work we train sequential models on NPTEL lectures and use it for lecture summarization. We try different model architectures. We also try pre-processing the data to impart som prior knowledge (prior knowledge is always helpful). We hypothesize that doing this kind of pre-processing should lead to better training as there is significant difference in the edgemaps of images with label 0 and 1 (refer Fig. 1) and as the labels change the diff shows significant variation (refer Fig. 2).

Acknowledgement

We would like to thank Dr. Prathosh A. P. for providing the dataset.