Image Classification Using CNN-LSTM Hybrid Model With Skip Connections

CS 554 - Introduction to Machine Learning and Artificial Neural Networks - Group E

Doğa Yılmaz
Graduate School of Engineering and Science
Özyeğin University
Istanbul, Turkey
doga.yilmaz.11481@ozu.edu.tr

Sena Odabaşı
Graduate School of Engineering and Science
Özyeğin University
Istanbul, Turkey
sena.odabasi@ozu.edu.tr

Onur Kirman
Graduate School of Engineering and Science
Özyeğin University
Istanbul, Turkey
onur.kirman@ozu.edu.tr

Berk Buzcu
Graduate School of Engineering and Science
Özyeğin University
Istanbul, Turkey
berk.buzcu@ozu.edu.tr

***Abstract*—Recently, with the advance of computer power and machine learning, there has been an increase in the study of the image classification problem. There have been approaches for image classifications that use CNNs and LSTMs. But they lag important aspects of images when it comes to human-like classification. The ones that propose human-like accurate models use massive and extreme deep models that require excessive computational resources. Motivated by these aforementioned problems, we proposed a new model that utilizes each of the best models with skip connections. The proposed model uses the feedback from LSTM to increase the overall performance of CNN. We benchmarked with single CNN and LSTM models, AutoEncoder and CNN + LSTM models over the same dataset of ours. The results indicate that overall our approach is better compared to the other models.**

***Index Terms*—CNN, Image Classification, LSTM, Neural Networks,**

I. Introduction

Identifying objects have always been a fascinating topic that computer scientists wanted to achieve. Since the increase in computational power and advances in machine learning, researchers started to come up with unique solutions to such problems. Yet, Image Classification is one of the main problems in Computer Vision. It takes images and categorizes them into classes. Convolutional Neural Networks(CNN) is a deep neural network that consists of layers like Multi-Layer Perceptrons. They can be used to classify images and object recognition.

In this project, we proposed a solution method to a single-label image classification problem that combines Convolutional Neural Networks(CNN) and Long Short-Term Memory(LSTM). We have prepared the data for the processing part, data-loader classes, training and testing scripts, a CNN model similar to LeNet [1] for creating baseline results.

LSTM is a kind of recurrent neural network. They are firstly introduced by Hochreiter & Schmidhuber [2]. Since the model is good with classification, we decided to use them in our project and combined LSTM with a CNN model.

There are four main models: CNN, CNN + LSTM, CNN + LSTM + Skip Connections and Auto-Encoder. The first model, CNN, uses Rectified Linear Unit (ReLU) activation function and Softmax activation function. In the second model, a CNN model and LSTM are merged. The third model is a developed version of the second model. Lastly, an autoencoder which consists of two parts: an encoder and a decoder is selected as a model to compare the solutions. Between the encoder and the decoder parts, a softmax function exists.

In order to test our proposed algorithm, we use CIFAR-10 [3], Caltech 101 [4], and Tiny ImageNet [5] datasets. Each dataset is divided into training and testing parts randomly

The rest of the paper is structured as follows: Section II Datasets & Data Loaders, Section III a review of literature, Section IV introduces the proposed methodology, Section V has the Implementation Details, Section VI contains the results of the experiments using the proposed model, and Section VII is the Conclusion.

II. Datasets & Data Loaders

We have collected the datasets to analyze and prepare them for processing. We used PyTorch library [6] for data handling. Below more information about our data preparation is provided.

- CIFAR-10 [3]: CIFAR-10 dataset is pre-loaded in TorchVision library. The testing and training classes are already separated with 50000 training, 10000 testing images. We also reserved 10% of the training data for validation which took 5000 images out of the training dataset. In the end, we end up having train, test, and validation sizes as 45000, 1000, and 5000 respectively.

- Caltech 101 [4]: Similar to the CIFAR-10 dataset, Torch library has a Caltech101 dataset. There are 8677 images and 101 classes in it. Similar to others, the dataset is already separated into training and testing. But the original version has some flaws such as not equally distributed classes. So, we have used a method that takes the most common labels with stratification to overcome this inequality between classes. In the end, we had 10 classes with train, test, and validation sizes of 2432, 676, and 271 respectively.
- Tiny ImageNet [5]: Tiny Image Dataset is not pre-loaded like previous ones. We have downloaded it from its source and written the necessary data-loader classes to make our data compatible with PyTorch. Also, we had issues of scarcity and complexity of data problems, meaning that each class has a huge amount of image set where there exist 200 classes. This required us to reduce the size first to 10, and in the end to even 5. In short, due to its high complexity, our models do not converge when they are given a high number of classes. So, we decreased the label size significantly to observe the models' performances. In the end, we had 5 classes with train, test, and validation sizes of 1800, 500, and 200 respectively.

III. Related Work

Long-Short Term Memory (LSTM) and Convolutional Neural Networks (CNN) have both been used for image recognition purposes as well as Natural Language Processing (NLP) purposes for a long time. One instance is Yao and Guan's [7] improved structure with LSTM for NLP. The research showed that LSTM based models show an increased F1 score with fewer resources than word2vec approaches. Similarly, LSTM was also used for vision-related problems. But LSTM can't process images alone given the dimensionality issues. Hence, Bappy et al. [8] propose a hybrid architecture with LSTM and encoder-decoder layers for the image forgery detection subproblem.

AutoEncoder is the simple process of stripping the data to form one smaller representation of it, and then regenerating the image from this representation. It is one of the most fundamental unsupervised learning algorithms, with the intention of eliminating a teacher and learning only from the data [9]. AutoEncoders have been tried in the classification problems as well. Due to the model's nature, we can first encode the data and use the encoded data to make a classification. One example of this is Geng et al. [10], where they classify high resolution synthetic aperture radar images classification. Due to the model's nature, it handles the high resolution images better than the other proposed architectures.

In order to compare the solutions, a CNN paper from literature is selected as benchmark for each dataset. Doon [11] made CIFAR-10 classification using Deep CNN and achieved 90% accuracy with training data and 87.57% with testing data. Feng [12] proposed KamiNet which is a CNN model for Tiny Image Dataset. In KamiNet, the depths are increased, the data is augmented with Crop+Flip scheme and they make data filtering. Their training accuracy is 97.5% and testing accuracy is 49.5%. Yang [13] implemented 18 layer resnet convolutional neural network and for the Caltech 101 it achieved 100% accuracy with training and 63.38% accuracy with testing data.

IV. Methodology

For the training methods, we have utilized a few loss functions, meaning functions that we utilize to metric our model's success during training, and combinations of them where necessary for models that needed it.

$$CrossEntropyLoss = -\sum_{i=1}^{outputsize} y_i * lo\acute{y}_i$$

We mainly used the Cross Entropy Loss function in our models, but for the AutoEncoders, we had to alter the loss function a bit to better fit the AutoEncoder classification idea, for the purposes of that, we also used the MSE Loss function.

$$MSELoss = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

We used the below formula to mix both these loss functions.

$$AutoEncoderLoss = 0.5 * MSELoss(encodedoutput) + \\ 0.5 * CrossEntropyLoss(modeloutput)$$

We take the encoded output and run it through a fully connected layer with 10 outputs, then we multiply the outcome with 0.5 and add it to the model's outcome, multiplied by 0.5. This way, we take into consideration the model's classification part as well as including the decoded part.

For our fully connected layers, we have used the RELU activation layers and logarithmic softmax at the outputs.

$$Relu(x) = \begin{cases} 0 & x \leq 0 \\ x & \text{otherwise} \end{cases}$$

The softmax activation layer is a smooth approximation to the argmax value, which means it helps us smooth out and normalize the output data in order to classify our prediction.

$$Softmax = \frac{\exp(x_i)}{\sum_j \exp(x_j))}$$

We have mainly utilized the LogSoftmax function as it can be seen below. The LogSoftmax function's logarithmic nature helps us penalise the features more than the Softmax activation function, hence it offers a sense of numerical stability. This behavior is observed to make the models converge faster, and is usually appended as a layer at the end of neural networks.

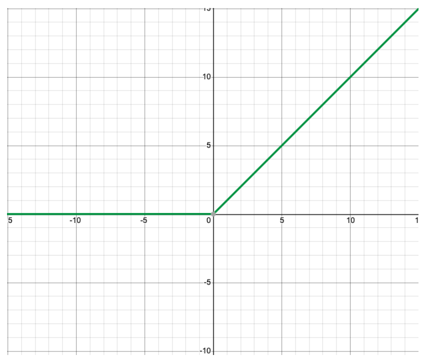


Fig. 1. RELU Activation Function

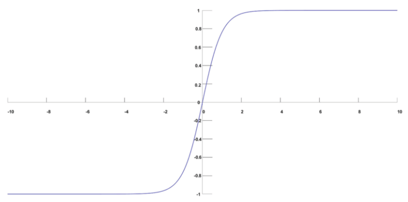


Fig. 2. Softmax Activation Function

$$LogSoftmax = \log(\frac{\exp(x_i)}{\sum_j \exp(x_j)})$$

MaxPooling helps us downsample our data after convolution steps. It helps us eliminate overfitting by outshining the features in sub regions within the data. Figure 3 is a 2D representation of this process, as you can see, from kernels of size two, we pick the largest element and place it on the corresponding slot, esentially reducing a matrix of size 4x4 to 2x2 with the vital features extracted.

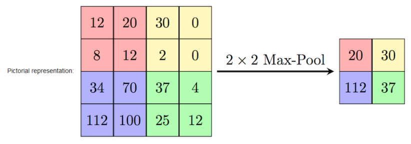


Fig. 3. 2D MaxPooling

While testing our algorithms, we have only considered the exact match ratio (accuracy) of our methods. We ran validation sets every 5 epochs.

V. Implementation Details

A. Preprocessing

1) Data Transforms: Transformations of datasets are needed to be able to use them in the PyTorch [6] environment. By its nature, PyTorch uses tensors as its computational object. For that purpose, we added a conversion of images into torch format. Also, while converting them into a tensor object, we normalize the images by centering their pixel value on mean and standard deviation to between 0-1. Moreover, for datasets, like Caltech101 [4], we needed an additional transformation of input sizes. Because it has images that are not uniformly shaped and images that can have a single channel instead of RGB. This requires additional care of transformation. For that purpose, we added a resizing operation that can convert any given input to a 32x32 shape and an image with a channel size of 3 to make sure that each image in the dataset has an RGB color format.

2) Data Augmentation: Our goal is to become resistant to errors while increasing our performance. So for that purpose, we used a technique called data augmentation that can help us create resilient and powerful models. It can increase the number of data we have using the existing data and can decrease the chance of overfitting in our training. It can be composed of different image transformations such as random crops, rotations, flips, and color modifications. We used only the image flips on both vertical and horizontal axis. This ensures that we train a model that is not prone to such rotational and flipped distortions in testing while prohibiting the overfitting to a certain extent.

3) Parser: In machine learning, we use a lot of parameters to train our models and get the best results on the test dataset. While doing so, things can pretty complicated and time-consuming. To overcome that issue, we used a best practice of hyper-parameter parser that can set our parameters from console command on the fly. We used argparse which is a command-line parser library. We created 6 different arguments for our parser which are model, dataset, batch size, epoch, learning rate, and validation frequency selections.

B. Models

Whit in the scope of this project, we have designed and implemented 4 different classifier models by using different building blocks and techniques. In this section, we will explain the models and their implementation thoroughly.

1) Model 1: CNN Model: Today in the literature CNN models are widely used in image classification tasks. For a baseline score, we have selected CNNs since their performance on this task is proven by other work [14]. Our model has 6 convolutional layers for feature extraction and 3 fully connected layers for classification. We used Rectified Linear Unit (ReLU) as activation for hidden layers. Also, we have used the Softmax function as the activation function for the output layer. Visualization of our CNN model can be found in Fig. 4.

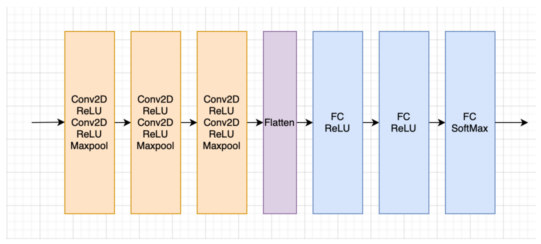

Conv2D ReLU Conv2D ReLU Maxpool
Conv2D ReLU Conv2D ReLU Maxpool
Conv2D ReLU Conv2D ReLU Maxpool
Flatten
FC ReLU
FC ReLU
FC SoftMax

Fig. 4. Architecture of our first model

2) Model 2: CNN + LSTM Model: Although it is not commonly used in the literature, some approaches attempt to merge CNN and LSTM for image classification such as the model proposed by Yin Q. et al. [15] and Shi X. et al. [16]. Our second model is inspired form already available work by [15]. Our second model has 6 convolutional layers for feature extraction. Following that we used a flattening layer to prepare our data for feeding into the LSTM layer. By using the LSTM layer we aim is to capture the recurring features of the provided data. After that, we feed the output of the LSTM to the fully connected layers to get the classification output.

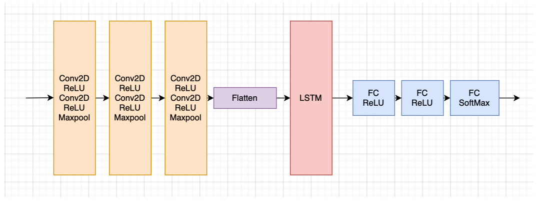


Fig. 5. Architecture of our second model

3) Model 3: CNN + LSTM + Skip Connection Model: Our aim in the third model is to mix the first model and the second model. The architecture is again similar to previous models starts with 6 convolutional layers which are responsible from feature extraction. Following that our model has 2 branches which are LSTM branch and Fully connected branch. After the operations in each branch we merge them using '+' operator which does element wise summation. Following that there are 3 fully connected layers in our model. The visualization of our model is shown in figure 6 below. To the best of our knowledge adding skip connection to a CNN LSTM model is a novel idea. By doing so we aim to combine characteristics of both model 1 and model 2 into the same model. In the results and conclusion we will discuss about the possible benefits and disadvantages of our model.

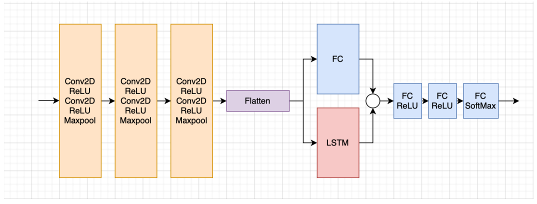


Fig. 6. Architecture of our third model

4) Model 4: Auto-Encoder Model: Finally, we have also decided to include an Auto-Encoder classifier model experimentally since we were curious on how it would perform. The model acts as a normal auto-encoder would, but before the decoding process, we take the encoded input and run it through a fully connected layer to acquire a classification. The main concern is that we need to consider this fully connected output in our loss function so it's performance is also included in the training. We have created a method in order to improve to consider the classification task of the model. Figure 7 illustrates our Auto-Encoder model.

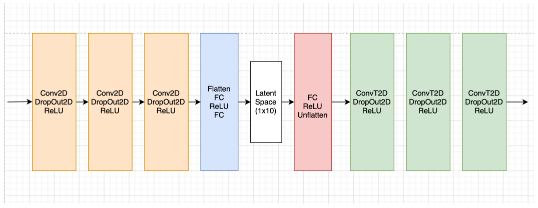


Fig. 7. Architecture of our fourth model

VI. Results

In this section, we have made experiments in order to see the feasibility of proposed models. We have calculated and compared the results according to the accuracy. Accuracy is calculated by dividing number of correct classification to number of total images.

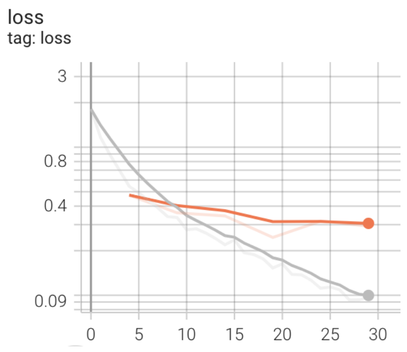


Fig. 8. CNN model training (gray) and validation (orange) set losses

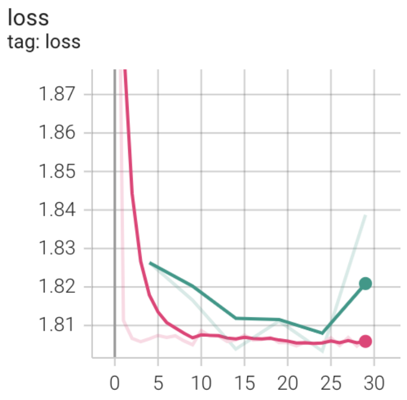


Fig. 9. CNN + LSTM model training (Magenta) and validation (Green) set losses

The accuracy results of the model can be found in the table I. We tested the model on three different dataset. As mentioned in Section II, the datasets are CIFAR-10 [3], Caltech 101 [4], and Tiny ImageNet. For CIFAR-10 and Caltech 101 datasets, 10 categories are selected and divided into training and testing parts. But for Tiny ImageNet dataset, the results were not good when we used 10 categories like the others. So, we decided to select only 5 categories from Tiny ImageNet.

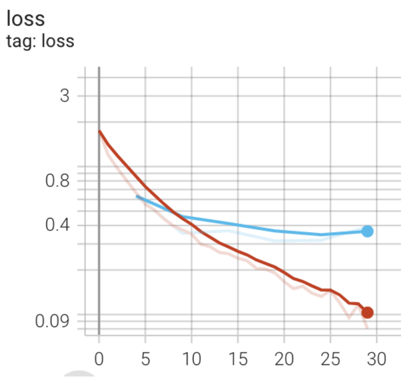


Fig. 10. CNN + LSTM model with skip connection training (red) and validation (blue) set losses

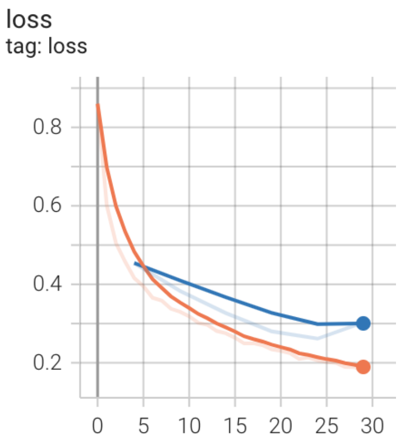


Fig. 11. Auto-Encoder model training (orange) and validation (blue) set losses

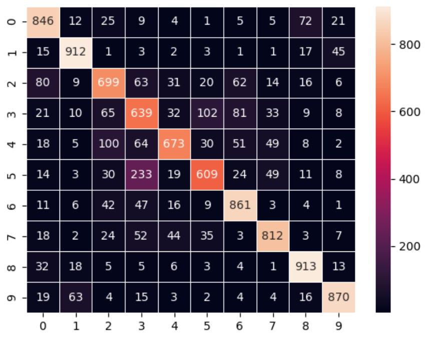
TABLE I
RESULTS TABLE

	CIFAR10	Caltech101	Tiny ImageNet
Number of Images	10000	676	1000
Number of Categories	10	10	5
Batch Size	16	16	16
Epoch	30	30	30
LR	0.0001	0.0001	0.0001
CNN	80%	91%	64%
CNN LSTM	80%	78%	59%
CNN LSTM SKIP	78%	91%	65%
Autoencoder	66%	90%	63%

After a several experiments the optimum parameters are selected and the final result table is made by using the optimum parameters. The batch size is 16. We have done experiments with learning rate 0.01, 0.001 and 0.0001. The best performer was 0.0001. We have decided to use epoch length as 30.

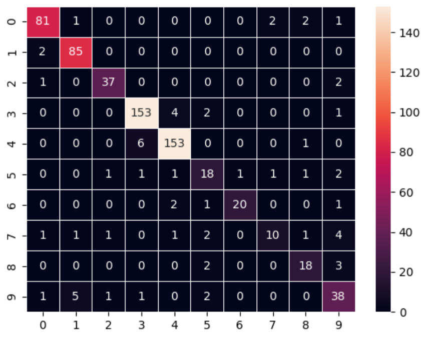
For CIFAR-10 dataset, first model and second model have achieved 80% accuracy. Skip connections model was not as high as the first two models and it performed 78% accuracy. For the last model, the accuracy was 66%. In Fig 12, the confusion matrix for model 3 can be seen.

Second dataset was Caltech 101. Model 1 CNN and Model 3 Skip Connections provided same accuracy level which is 91% for Caltech 101. Model 2, combining LSTM with CNN did not perform as well as them. An 18 layer Resnet CNN model had achieved 63.38% accuracy [13]. Our proposed model is better when it is compared with the benchmark model. In Fig 13, the confusion matrix for model 3 can be seen.



	0	1	2	3	4	5	6	7	8	9
0	846	12	25	9	4	1	5	5	72	21
1	15	912	1	3	2	3	1	1	17	45
2	80	9	699	63	31	20	62	14	16	6
3	21	10	65	639	32	102	81	33	9	8
4	18	5	100	64	673	30	51	49	8	2
5	14	3	30	233	19	609	24	49	11	8
6	11	6	42	47	16	9	861	3	4	1
7	18	2	24	52	44	35	3	812	3	7
8	32	18	5	5	6	3	4	1	913	13
9	19	63	4	15	3	2	4	4	16	870

Fig. 12. Confusion matrix of CIFAR10 for Model 3: CNN + LSTM + Skip Connection



	0	1	2	3	4	5	6	7	8	9
0	81	1	0	0	0	0	0	2	2	1
1	2	85	0	0	0	0	0	0	0	0
2	1	0	37	0	0	0	0	0	0	2
3	0	0	0	153	4	2	0	0	0	1
4	0	0	0	6	153	0	0	0	1	0
5	0	0	1	1	1	18	1	1	1	2
6	0	0	0	0	2	1	20	0	0	1
7	1	1	1	0	1	2	0	10	1	4
8	0	0	0	0	0	2	0	0	18	3
9	1	5	1	1	0	2	0	0	0	38

Fig. 13. Confusion matrix of Caltech 101 for Model 3: CNN + LSTM + Skip Connection

Tiny ImageNet was the most challenging dataset of all. Very similar to Caltech 101, CNN + LSTM was the worst model. When Skip Connection is added to the model 2, the improvement was clearly obtained. Also, model 3 performed better than KamiNet [12] which only performed 49.5%. They were working with category size of 200, but we used 5 categories.

In Fig 14, the confusion matrix for model 3 can be seen.

The models performed differently for each dataset. CNN or CNN + LSTM models are best models for CIFAR10. For Caltech 101, CNN or CNN + LSTM + Skip Connection models are the best ones. And for Tiny ImageNet the best performing model is the proposed model.

VII. CONCLUSION

In this paper, we tried to come up with a new neural network architecture for image classification problem. We introduced a model that utilizes LSTM and CNN using skip connections.

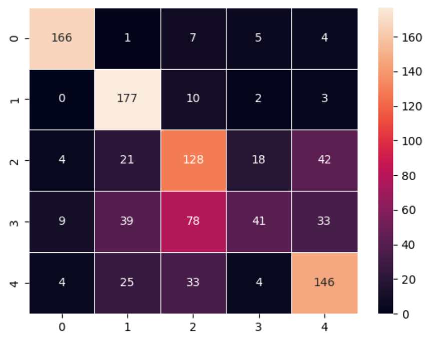


Fig. 14. Confusion matrix of Tiny ImageNet for Model 3: CNN + LSTM + Skip Connection

Also, we rigorously benchmarked our model to models such as CNN, CNN + LSTM and AutoEncoder over three different image datasets. These image datasets are CIFAR-10, Caltech 101 and Tiny ImageNet which models perform differently. Almost all of the benchmark tests, our proposed approach got better results. For Caltech 101 and Tiny ImageNet datasets, our proposed model CNN + LSTM + Skip Connection was the best performer model. On the other hand, for CIFAR10 dataset, CNN or CNN + LSTM may be preferred.

From the loss graphs we also observed that the skip connections has helped CNN + LSTM model converge faster, so one of the main contributions of the skip connection in this instance could be considered the improvement on the computational efficiency of the model.

In the future work, it would be better to explore deeper networks that mimics the LSTM + CNN with skip connection. This may increase model's learning capability and precision. Moreover, the parameters of the model can be optimized by Grid Search or a similar method.

Acknowledgment

We would like to thank Prof. Dr. Ethem Alpaydın (Özyeğin University) for providing the opportunity to research and analyze of machine learning and computer vision fields in CS 554 - Introduction to Machine Learning and Artificial Neural Networks graduate course.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. *Advances in neural information processing systems*, pages 473–479, 1997.

[3] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. (0), 2009.

[4] Li Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.

[5] Jiayu Wu, Qixiang Zhang, and Guoxi Xu. Tiny imagenet challenge. *Technical Report*, 2017.

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[7] Lirong Yao and Yazhuo Guan. An improved lstm structure for natural language processing. In *2018 IEEE International Conference of Safety Produce Informatization (IICSPI)*, pages 565–569, 2018.

[8] Jawadul H. Bappy, Cody Simons, Lakshmanan Nataraj, B. S. Manjunath, and Amit K. Roy-Chowdhury. Hybrid lstm and encoder–decoder architecture for detection of image forgeries. *IEEE Transactions on Image Processing*, 28(7):3286–3300, 2019.

[9] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 37–49, Bellevue, Washington, USA, 02 Jul 2012. PMLR.

[10] Jie Geng, Jianchao Fan, Hongyu Wang, Xiaorui Ma, Baoming Li, and Fuliang Chen. High-resolution sar image classification via deep convolutional autoencoders. *IEEE Geoscience and Remote Sensing Letters*, 12(11):2351–2355, 2015.

[11] Raveen Doon, Tarun Kumar Rawat, and Shweta Gautam. Cifar-10 classification using deep convolutional neural network. In *2018 IEEE Punecon*, pages 1–5. IEEE, 2018.

[12] Shaoming Feng and Liang Shi. Kaminet—a convolutional neural network for tiny imagenet challenge. *CS 231N*, 2015.

[13] Yifei Yang and Yuan Jiang. Image classification for caltech101. 2020.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[15] Qiwei Yin, Ruixun Zhang, and XiuLi Shao. Cnn and rnn mixed model for image classification. In *MATEC Web of Conferences*, volume 277, page 02001. EDP Sciences, 2019.

[16] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.