AutoSumm: Automatic Model Creation for Text Summarization

Sharmila Reddy Nangi[1]*, Atharv Tyagi[2]*, Jay Mundra[2]*, Sagnik Mukherjee[2]*
Snehal Raj[2]*, Aparna Garimella[3], Niyati Chhaya[3]
[1]Stanford University, USA [3]Adobe Research, India
[2]Indian Institute of Technology Kanpur, India
srnangi@stanford.edu[1] {garimell,nchhaya}@adobe.com[3]
{atharv,jaym,sagnikm,snehal}@iitk.ac.in[2]

Abstract

Recent efforts to develop deep learning models for text generation tasks such as extractive and abstractive summarization have resulted in state-of-the-art performances on various datasets. However, obtaining the best model configuration for a given dataset requires an extensive knowledge of deep learning specifics like model architecture, tuning parameters etc., and is often extremely challenging for a non-expert. In this paper, we propose methods to automatically create deep learning models for the tasks of extractive and abstractive text summarization. Based on the recent advances in Automated Machine Learning and the success of large language models such as BERT and GPT-2 in encoding knowledge, we use a combination of Neural Architecture Search (NAS) and Knowledge Distillation (KD) techniques to perform model search and compression using the vast knowledge provided by these language models to develop smaller, customized models for any given dataset. We present extensive empirical results to illustrate the effectiveness of our model creation methods in terms of inference time and model size, while achieving near state-of-the-art performances in terms of accuracy across a range of datasets.

1 Introduction

Machine learning algorithms, particularly, deep learning techniques have led to the simplification of several computationally expensive tasks. However, training and optimizing these models for different tasks demand the experienced engineering resources and require expertise, making it difficult for non-experts. *Automated Machine Learning* is a strategy to automate this pipeline for model creation including automated generation of the model itself.

In the case of Natural Language Processing and Text analysis, the advent of large language models such as BERT(Devlin et al., 2019), GPT2(Radford et al., 2019), and more recently GPT3(Brown et al., 2020) have created resources that can be exploited for the creation of robust models for several downstream NLP tasks. However, the need for ML expertise creates a bottleneck. Further, these deep learning models have thousands of parameters and need fairly large datasets and computational resources for training.

We focus on providing algorithms for auto-generation of ML models for complex NLP tasks such as extraction and generation, making them accessible to non experts. Our proposed approaches feed off the knowledge available in large pretrained models to auto-generate new, smaller, customized models for a custom dataset. Specifically, the major contributions are as follows.
(1) We propose a method to create machine learning models that are efficient and customized to a given dataset for the tasks of extractive and abstractive summarization, using a combination of *neural architecture search* and task-specific *knowledge distillation* from large language models.
(2) Aditionally, we propose an alternate method for summarization model generation using Transformer distillation, which is superior in terms of performance and resource utilisation.
(3) We conduct extensive experiments and present results illustrating the effectiveness of the proposed methods for extractive and abstractive summarization on a range of datasets, and compare our models in terms of model creation efficiency, model size, inference time, and performance, with the state-of-the-art models.

To the best of our knowledge, this is the first effort towards automatically building customized and compressed models for text generation tasks, specifically summarization.

*Work done while authors were at Adobe Research.

2 Related Work

Neural Architectural Search is a trending area in AutoML, which automates the process of model creation by searching efficient model architectures, without human expertise. A typical NAS problem involves identifying the *search space*, employing a *search strategy* to find the best task-specific model architecture, and training the model from scratch. Most of the NAS experiments are done on images - (Real et al., 2017), (Real et al., 2018), (Suganuma et al., 2018) using neuro-evolutional and genetic algorithms, which are computationally very expensive and time consuming. Recently, gradient based methods like DARTS (Liu et al., 2019), SNAS (Xie et al., 2019) and (Dong and Yang, 2019) are proposed to speed-up the search strategy. But, the explorations of using NAS for language-related tasks are very limited. (Wang et al., 2020) propose TextNAS with a search space for better understanding of the text representations. They use a simple and efficient, ENAS (Pham et al., 2018), which is guided by reinforcement learning for model generation. However, these models mainly focus on text understanding and do not directly extend to the generation-related tasks like summarization.

Text Summarization: The neural attention model (Rush et al., 2015) marked the beginning of using deep neural architectures for text summarization. Seq2seq models variations with convolutional encoder (Chopra et al., 2016),(Narayan et al., 2018b), hierarchical attention-based RNN encoder (Nallapati et al., 2017), pointer-generator networks (See et al., 2017) were used for both extractive and abstractive tasks. With the recent advent of multi-head attention (Vaswani et al., 2017), transformer-based models like PEGASUS (Zhang et al., 2019), BERT-Summ (Liu and Lapata, 2019a) are proposed with pre-trained objectives tailored for summarization tasks. While these methods give good results, they demand extreme human expertise and computational overhead for designing and deployment.

Knowledge Distillation & Model Compression: These techniques aim to take advantage of the immense knowledge from the pre-trained models. TinyBERT (Jiao et al., 2019) presents a distillation approach for text classification and natural language inference using BERT compression and distillation. Adabert (Chen et al., 2020a) present a differential NAS algorithm, leveraging a BERT model through knowledge distillation for classification and NLI tasks. (Chen et al., 2020b) transfers BERT knowledge to a encoder-decoder model for text generation. However, all these approaches are limited to the specific tasks, and are not directly extensible to a generation-based tasks.

3 Methodology

Figure 1 shows the overview of our model creation framework. The input is the dataset and task specifications (summary type, size) and the output is a custom trained summarization model, which can be further used to create text summaries. In this paper, we generate models for both extractive and abstractive summarization tasks, with the former being a binary classification task to extract summary sentences from the input, while the latter aims to generate summaries containing novel words and phrases that may not be present in the input text.

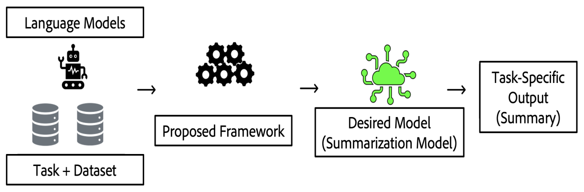

Language Models
Task + Dataset
Proposed Framework
Desired Model (Summarization Model)
Task-Specific Output (Summary)

Figure 1: Overview of the proposed approach.

Our proposed approaches distills knowledge from a language-model based *teacher* network to generate an encoder-decoder-based *child* model. We present two algorithms that aid in auto-creation of different types of resulting 'child' models - (1) a model with convolutional and recurrent units and (2) a mini-transformer based model. The first is achieved by our approach AUTOSUMM-CREATE and the second using AUTOSUMM-DISTILL, which are detailed as follows.

3.1 AutoSumm-Create

Figure 2 illustrates AUTOSUMM-CREATE method. Here, we combine knowledge distillation with *neural architecture search* to auto-create an encoder-decoder based summarization model. The stages in this method include:

1. Task-specific knowledge distillation: We leverage knowledge from a transformer-based BERT model (*teacher*) fine-tuned for extractive and abstractive summarization (Liu and Lapata, 2019b) on the given task-specific (summarization) dataset. The predictions from the teacher model are used for distillation, i.e., the sentences classification scores

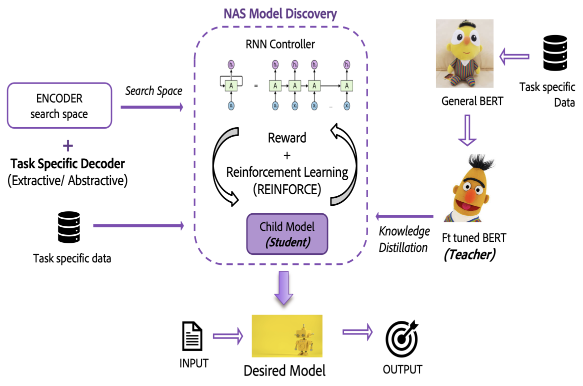


Figure 2: AUTOSUMM-CREATE.

for extractive and probability distributions over the vocabulary for abstractive are augmented to the ground truth. A Knowledge Distillation(L_{KD}) loss is included to perform informed search on the child models, ensuring that they mimic the performance of the teacher.

In extractive summarization, L_{KD} is the MSE loss between soft labels from augmented data and the scored predicted by the child model.

$$L_{KD} = \sum_{i=1}^{n} (y_i^{teacher} - y_i^{pred})^2 \tag{1}$$

In abstractive summarization, L_{KD} is calculated at each time step t using soft labels $P_{teacher}(y_t)$ from teacher model and the predicted labels $P_{pred}(y_t)$ from child model over vocab V as follows:

$$L_{KD} = \sum_t \sum_{w \in V} P_{teacher}(y_t = w | y_{1:t-1}, X). \\ log(P_{pred}(y_t = w | y_{1:t-1}, X)) \tag{2}$$

2. Neural Architectural Search: Augmented dataset, along with a small labelled custom dataset, is used to train the NAS module, which searches for the right combination of cells that result in the *child* model most suited for the summarization task. In our approach, we use NAS to search the encoder space while using a predefined (task-specific) decoder. The key components of this module are:

Search space. Following (Wang et al., 2020), we define macro search space, such that the model can be represented by a directed acyclic graph (DAG) with nodes representing a layer from the search space and edges representing the directionality of flow of information. The search space has 4 key cell types - CNN (kernel sizes 1,3,5,7), RNN (bidirectional GRU), Pooling layers (avg. pool and max. pool with stride 1 and uniform padding), and Multi-head self-attention (8 heads, no positional embeddings). We constrain the search space by (1) defining the number of skip connections allowed, (2) limiting the maximum number of layers in the child architecture, l (in our case $l \in$ 1,5,10,18,20), and (3) defining the cells allowed in the new architecture. These constraints define the exhaustive list of possibilities for the NAS algorithm.

Search algorithm: We implement ENAS (Pham et al., 2018), a reinforcement learning (RL) based algorithm used for several NAS implementations (Zoph and Le, 2017). It consists of an RNN controller network, that samples a model architecture from the search space and an RL reward to nudge this controller towards generating an optimal architecture.

Pre-defined Model Specifications: As stated earlier, we auto-create the encoder layers in the model but predefine the task-specific decoder. For extractive summarization, the decoder is a scorer function with sigmoid activation, which takes in the text representations learnt from the encoder and scores each sentence on a scale of (0,1]. The sentences with the high scores are chosen as the final summary based on the summary size specified. For abstractive summarization, a recurrent neural network is used as the decoder. The input is the text representation from the encoder and the output is a generated summary (generated in auto-regressive manner, by decoding a word at every time step).

Loss: The architectures are trained with a cross-entropy loss at sentence level for extractive and vocab level for abstractive as follows:

$$Ext(L_{CE}) = \sum_{i=1}^{n} (p_{gt}(y_i).log(y_i^{child}) \tag{3}$$

$$Abs(L_{CE}) = \sum_t \sum_{w \in V} P_{gt}(y_t = w). \\ log(P_{pred}(y_t = w | y_{1:t-1}, X)) \tag{4}$$

Final Loss: The final end-end loss associated with this framework is computed as the weighted sum of the L_{KD} and L_{CE} in the NAS module:

$$L_{total} = \alpha.L_{CE} + (1 - \alpha).L_{KD} \tag{5}$$

RL Reward: A reward based on the performance of the child model, is sent back to the RNN controller. The policy gradients of the RNN controller are updated through REINFORCE(Williams, 1992) algorithm. Reward (R) is defined as $1 - Loss_{valid}$, normalized over the batchsize.

Re-training: The newly generated model, is trained using the user-provided training data optimizing for the total loss(L_{total}). This trained model can generate summaries for any given test sample.

3.2 AutoSumm-Distill

In this approach, the structure of the child is defined as a mini-transformer(4 layers). A knowledge distillation technique called *transformer distillation* (Jiao et al., 2019) is used to create a general-mini-transformer(4 layers) from a large transformer model (12 layers). Then, the knowledge is distilled from a task-specific fine-tuned BERT ('teacher') model to the general-mini-transformer. Figure 3 illustrates the workflow of this method. This method differs from AUTOSUMM-CREATE, in the child model architecture and the usage of two transformer teacher models. The key stages in this method are detailed below.

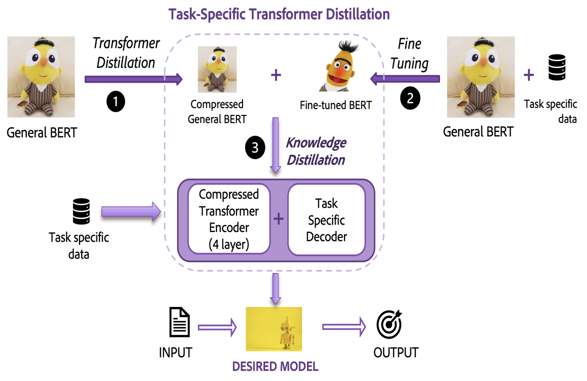


Figure 3: AUTOSUMM-DISTILL.

Knowledge distillation: There are two forms of knowledge distillation in this method (1 and 3 in Fig.3). We detail the knowledge distillation from a task-specific transformer teacher (we use BERT-Summ (Liu and Lapata, 2019b)) to the general-mini-transformer which forms the encoder layer for the final child model. The decoder is pre-defined based on the task, similar to AUTOSUMM-CREATE. A transformer model has various types of layers including multi-headed attention, embedding layers, and the hidden layers. The intuition behind knowledge distillation is to teach the layers in the child transformer to mimic the corresponding layers in the teacher transformer. This is implemented by introducing separate losses for each layer type.

Attention-based distillation builds on the intuition that the attention layers in BERT capture linguistic information such as syntax and coreference information. Specifically, the student aims to learn the matrices of the multi-headed attention from teacher. This loss is given by $L_{attn} = 1/h[\sum_{i=1}^{h} MSE(A_i^S, A_i^T),]$ where h is the number of attention heads A_i refers to the attention matrix corresponding to the i-th head of the teacher (T) or the student (S), l is the input text length and $MSE(.)$ refers to the mean squared error loss.

Hidden-state distillation distills knowledge from the output of transformer hidden layer, with $L_{hidn} = MSE(H^s W_h, H^T)$ where H^s and H^T refer to the hidden states of the student and teacher models. W_h is a learnable linear transformation.

Embedding-layer distillation: Formulated as $L_{embd} = MSE(E^s W_e, E^T)$ where E^s and E^T are embeddings in the student and teacher networks respectively. W_e plays a similar role as W_h. Using these distillation objectives along with the general distillation already done to compress the transformer model to general-mini-transformer, the final loss is the unified distillation loss of the corresponding layers between the teacher and the student model. As a reminder, this step helps auto-learn the task specific encoder for extractive and abstractive summarization.

Pre-defined Model Specifications: For extractive summarization, we define a single transformer layer on top of the newly created encoder with a classification layer as the decoder. For abstractive summarization, the decoder is 6-layer transformer.

Training and Re-training:

General distillation & Fine-tuning: The above model is trained in a phased manner. The first distillation or training is done from a large transformer (BERT) to the general – mini- transformer. Parallelly, a large BERT model is fine-tuned for the specified tasks. Both these steps need not be repeated for every new dataset from the user and every run of the model. The fine-tuned model and the general-mini-transformers may be created once per task and once per a very large benchmark dataset.

Task-specific Distillation: This process of teaching the student model from a fine-tuned teacher model is repeated each time a new user dataset is given to the system. Once trained, this is coupled with the specific decoder.

Re-training: Once the final child model i.e. mini-transformer encoder and corresponding decoder are created, this complete model is trained on the input user dataset. The final model is the output for the user along with test summaries for any given text input.

4 Experiments

We evaluate our proposed framework by performing experiments that test the performance of the newly created models against benchmark summarization datasets on both extractive and abstractive tasks.

4.1 Datasets

Table 1 summmarizes the train/val/test split of all the datasets. **The CNN / Daily Mail dataset** (Hermann et al., 2015) contains online news articles (781 tokens on average) paired with multi-sentence summaries (3.75 sentences or 56 tokens on average).

The New York Times (NYT) Annotated Corpus contains the full text and metadata of NYT articles from 1987 to 2007. Following (Durrett et al., 2016), we extracted and filtered out the articles from 2000-2007, with abstractive summaries having more than 50 words. The articles were split based on the date of publication, where the articles from January 1, 2007 were chosen as test set.

X-Sum (Narayan et al., 2018a) dataset is collected from online BBC articles, with short one sentence summaries. **The Gigaword** (Rush et al., 2015) dataset contains 4M examples from news articles for sentence summarization / headline generation task. The summaries are very short with 9 tokens per summary. **The Contract dataset** (Manor and Li, 2019) is a dataset compiled from two websites dedicated to explaining unilateral contracts in plain English: TL;DRLegal[1] and TOS;DR[2]. It is a small dataset with 500 samples.

Dataset	#train	#valid	#test
CNN/DM	287,113	13,368	11,490
NYT	106,826	6,000	6,258
XSum	204,045	11,332	11,334
Gigaword	300,000	10,000	2,000
Contract	400	23	23

Table 1: Dataset details

[1]https://tldrlegal.com/
[2]https://tosdr.org/

4.2 Models

The generated models through AUTOSUMM-CREATE for extractive and abstractive are CHILD-EXT and CHILD-ABS respectively. -KD denote the *child* model variations trained through Knowledge distillation (KD). The fine-tuned models through AUTOSUMM-DISTILL are FT-TINYBERT-EXT and FT-TINYBERT-ABS. We compare the performance of our models against BERT-Summ (Liu and Lapata, 2019a), as it had a general framework for extractive and abstractive and was shown to give state-of-the-art performances. These baseline models are FT-BERT-EXT and FT-BERT-ABS.

4.3 Implementation Details

For all our experiments, we use the existing splits if available, otherwise we split the data according to the statistics in Table 1 and keep them constant across all the experiments.

In our AUTOSUMM-CREATE experiments, we perform a 20-layer neural architectural search for encoder. The decoders are task-specific and predefined as explained in our previous section. We use GloVe word embeddings while providing the input to the generated model. We set the batch size as 128, max input length as 64, hidden unit dimension for each layer as 32, dropout ratio as 0.5 and L2 regularization. We utilize Adam optimizer and learning rate decay with cosine annealing. The parameter of KD proportion α is varied in NAS module.[3]. We also perform experiments with varying layer size, discussed in the later sections.

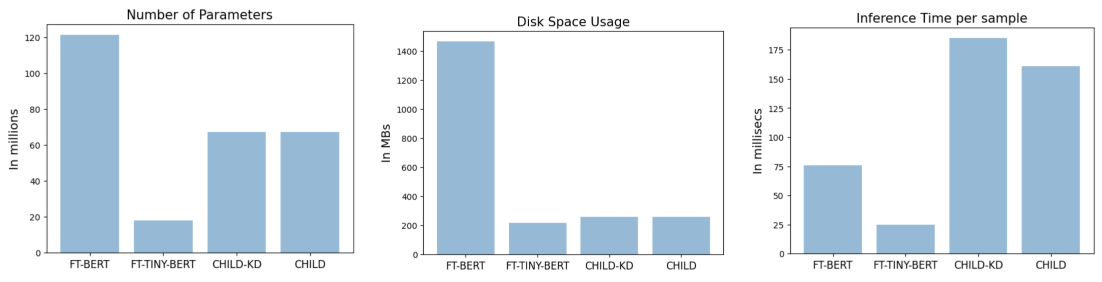
4.4 Evaluation metrics

Summarization quality is evaluated using F1 measure of ROUGE score (Lin, 2004) calculated between generated and ground-truth summary.[4]We report unigram and bigram overlap (ROUGE-1 and ROUGE-2) to assess informativeness and the longest common sub-sequence (ROUGE-L) to access fluency. Additional metrics like number of parameters, disk-space and the inference time taken are considered to compare the computational efficiency between models.

5 Results and discussion

Extractive Summarization: Table 2 shows results comparing the performance of our generated

[3]Results are included in supplementary material
[4]https://github.com/google-research/google-research/tree/master/rouge



(a) Number of parameters (↓) (b) Disk Space(↓) (c) Inference Time(↓)

Figure 4: Efficiency comparison for the extractive summarization models on the CNN/DM dataset

MODEL	CNN/DM			NYT		
	R-1	R-2	R-L	R-1	R-2	R-L
FT-BERT-EXT	43.58	20.69	28.08	45.68	26.20	33.11
FT-TINYBERT-EXT	42.28	19.53	26.76	43.94	24.61	32.06
CHILD-EXT	39.10	14.68	20.78	45.68	26.38	35.04
CHILD-EXT-KD	41.08	18.73	26.72	45.89	26.60	35.20

Table 2: A comparison of the generated models for Extractive summarization on CNN/DM and NYT ; FT-BERT-EXT is used as a baseline to compare against

Model	R-1	R-2	R-3
FT-BERT-ABS	21.03	6.04	19.34
FT-TINYBERT-ABS	32.36	13.80	29.93
CHILD-ABS	25.04	8.14	23.13
SOTA (Song et al., 2020)	39.08	20.47	36.69

Table 3: Abstractive Summarization on Gigaword

MODELS	R1	R2	RL
Baseline Lead-K	24.38	7.52	17.63
CHILD-ABS	**40.04**	**23.63**	**35.21**

Table 4: Abstractive Summarization on Contracts

models and the baseline for extractive summarization across different datasets. The ROUGE scores show that the summaries by the auto-generated models from our proposed framework are close to the state-of-the-art BERT baseline. Whereas, our models gained significantly in terms of computational efficiency. Figure 4 illustrates the same - when the models trained on CNN/DM dataset are compared along three aspects - Number of parameters (in millions), disk space for storing the model (in MB) and the inference time(in milliseconds) needed to generate the summary of an input sample. These graphs depict that the generated models with comparable performance to FT-BERT significantly reduce the disk space usage and the number of parameters. We note that the generated models from AUTOSUMM-CREATE lose some performance in terms of inference time, which is because the model architecture consists of RNNs and does not have the advantage of parallel computation present in BERT models. However, our FT-TINYBERT-EXT model overcomes this and significantly reduces the inference time.

Abstractive Summarization: Table 3 compares the performance of our abstractive-summarization models on Gigaword dataset, curated for extreme summarization. It is to be noted that our proposed summarization model with Transformer distillation FT-TINYBERT-ABS beats the FT-BERT-ABS with a huge margin, across all R-1,R-2,R-L.

The other dataset for extreme summarization is the Contract dataset. Table 4 shows the performance of our generated CHILD-ABS model on contracts dataset. We compare our results against the reported best performing Lead-K scores by Cohen et al.,(2018). Note that the limited size of the dataset was a bottleneck to train FT-TINYBERT-ABS model.

Model Architectures: The AUTOSUMM-CREATE approach generates a new encoder architecture from scratch for a desired task and dataset. It is an interesting study to dive deeper into the distribution of the cells in the generated models. Table 5 shows the distribution of cell type in the generated models with a 10 layer encoder architecture, on extractive (on CNN/DM) and abstractive (on Gigaword) tasks. It can be observed that the pooling and the attention layers are sparse in the extractive models, but are major contributors in the abstractive architecture. Most recent models use the multi-head attention from transformer to get good results in the language generation task. A similar pattern is observed in the models generated through our AutoSumm framework.

Percentage of cells	Extractive		Abstractive	
	Child Model (w/o KD)	Child Model (with KD)	Child Model (w/o KD)	Child Model (with KD)
CNN(1,3,5,7)	0,0,0,2	0,0,0,6	1,1,0,0	1,0,0,2
RNN	8	4	1	0
Avg-Pool, Max-pool	0,0	0,0	2,0	1,0
Attention	0	0	5	6

Table 5: Division of CNN and RNN cells for Extractive (CNN/DM) and Abstractive(Gigaword)

5.1 Ablation Study

Variation in Layer Size: We analyze the performance of our framework across varying sizes of the target model i.e. varying the number of layers to be generated by the RNN-controller. We experiment with CHILD-EXT models. Figure 5 illustrates the results of this experiment for extractive summarization on the CNN/DM dataset. We observe that the CNN and RNN layers are the major constituents in these architectures. We can see that RNN cells are more preferred when the architecture is restricted to fewer layers (like 2, 5, 6), but as we increase the layers, Convolutional layers with larger stride(7) are preferred. Table 6 refers to the performance of these models. While the model with 15 layers gives the best performance, the performance does not drop too much with varying number of layers. Hence, a smaller model, with fewer layers and in turn lesser number of parameters can be efficiently generated for extractive summarization through our approach.

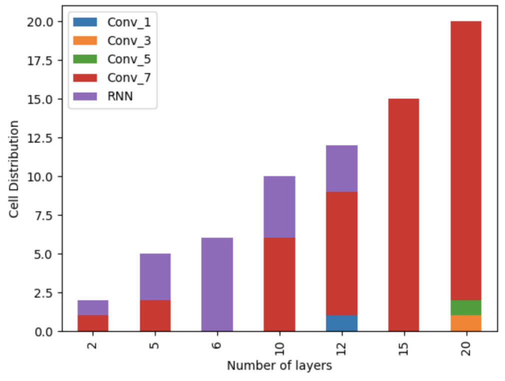


Figure 5: Cell distribution across varying layer size

Table 7 and Figure 6 presents the results when the same experiment is conducted on XSUM and Contract datasets with the layer sizes 5, 10, 12 and 15. While the trend in RNN preference for fewer layers and CNN preference for more layers still continues, it is noted that the larger architectures generated use the pooling and attention layers.

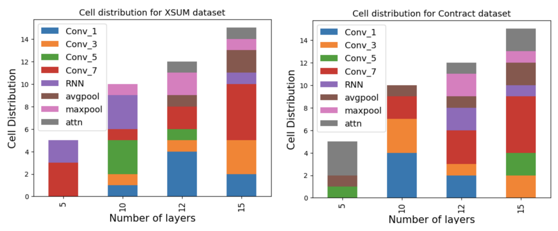
Cross-Dataset Experiments: Table 8 shows experiments with CHILD-EXT-KD-X models trained on one dataset (X) and tested on another. The visualisations of these results as in Figure 7, also show that the architectures trained on one dataset can be used to generate summaries on a different dataset without significant loss in performance, establishing the generalizability of the proposed approach making it usable by non-experts for real-world applications.

Layer Size	R-1	R-2	R-L
2	40.8	18.5	26.55
5	41.03	18.67	26.71
10	41.08	18.73	26.7
12	40.98	18.65	26.68
15	**41.1**	**18.78**	**26.84**
20	41.1	18.7	26.73

Table 6: CHILD-EXT-KD model performance with layer size variation on CNN/DM

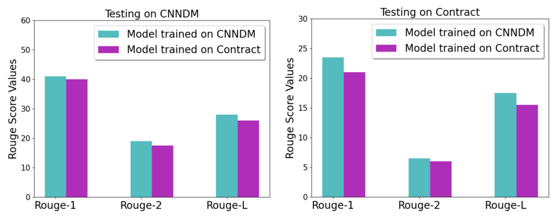
Layer Size	XSUM			CONTRACT		
	R-1	R-2	R-L	R-1	R-2	R-3
5	20.7	3.69	13.52	21.15	6.14	15.96
10	20.84	3.64	13.68	21.2	6.26	15.99
12	20.73	3.59	13.63	21.23	6.12	15.9

Table 7: Varying layer size on CHILD-EXT-KD for XSUM and Constract



(a) XSUM (b) Contract

Figure 6: Layer distribution for XSUM and Contract



(a) Test on CNNDM (b) Test on Contract

Figure 7: Cross-Data experiments

Training data variation: To reiterate, in our AUTOSUMM-CREATE framework, we generate an architecture with cells from a given search space, and *re-train* the generated architecture with the

CHILD-EXT-KD	CNN/DM (R1:R2:RL)	XSUM (R1:R2:RL)	Contract (R1:R2:RL)
CHILD-EXT-KD-CNNDM	**41.06, 18.91, 27.04**	16.78, 1.83, 12.35	24.07, 6.42, 17.71
CHILD-EXT-KD-XSUM	18.05, 2.7, 13.52	**20.84, 3.64, 13.68**	24.07, 6.42, 17.71
CHILD-EXT-KD-Contract	40.32, 17.84, 25.57	18.52, 2.43, 11.93	21.22, 6.07, 15.95

Table 8: Cross-Dataset experiments on CHILD-EXT-KD models

Model	1 (R1 : R2 : RL)	3 (R1 : R2 : RL)	5 (R1 : R2 : RL)
CHILD-EXT-KD	33.05, 14.86, 24.1	**41.08, 18.73, 26.72**	37.31, 17.43, 24.05
CHILD-EXT	30.20, 12.04, 21.38	**40.00, 18.18, 25.91**	38.02, 18.03, 24.86
FT-BERT-ABS	33.84, 15.72, 24.88	**43.58, 20.69, 28.08**	39.85, 19.26, 24.93
FT-TINYBERT-EXT	32.65, 14.95, 24.07	**42.29, 19.53, 26.76**	38.48, 18.04, 23.87

Table 9: Performance comparison with different decoding sizes

	Contract
Input	Third party vendors including Google use cookies to serve ads based on a user s prior visits to our website or other websites . google s use of advertising cookies enables it and its partners to serve ads to you based on you visit to our sites and or other sites on the internet .
Reference	This service allows tracking via third party cookies for purposes including targeted advertising .
CHILD-ABS	This service allows tracking via third party cookies
	Gigaword
Input	Tens of thousands of demonstrators marched through brussels sunday , calling for EU action to defend jobs , amid widespread anger at a decision by French carmaker Renault to close a factory here .
Reference	Tens of thousands march through brussels to defend jobs
FT-BERT-ABS	Tens of thousands march in brussels against renault 's closures plant closures closure
CHILD-ABS	Thousands march in brussels as eu protest over

Table 10: Examples of Abstractive summaries on Contract and Gigaword

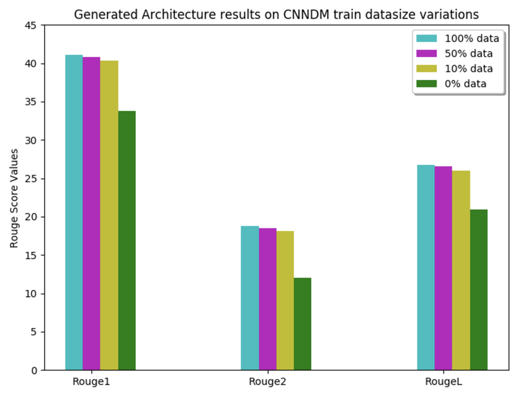


Figure 8: Performance vs Training data variation

training size and Knowledge Distillation. In order to measure the amount of data required for this re-training, we performed an experiment by varying the size of the training data used for this purpose. Figure 8 illustrates the result of this experiment, where we vary the amount of training data(from 0% to 100% of the total available data), used to re-train an architecture searched for extractive summarization on CNN/DM dataset. Here, 0% data refers to randomly initialized model that has not been re-trained. Note that the Rouge scores(R-1, R-2, R-L) reported for all these model variations are calculated on the test set. While it is intuitive that, more training data results in improved performance, we note that even with 10% data the decent performance value is achieved. We believe such an observation can help support the hypothesis that the using the proposed framework, we can build usable models with limited training data.

Decoding Size variation: Table 9 denotes the results of varying ROUGE scores with the decoding summary sizes (1,3,5) on CNN/DM extractive summarization task. While, a summary size of 3 sentences yields best result (some of it due to the nature of the training data), we observe that the proposed framework allows generating shorter or longer summaries without significant loss in performance, again establishing the generalizability.

Table 10 shows qualitative examples of the output summaries for a couple of newly generated models. Through the above experiments, we establish that the auto-generated models generated using the proposed NAS and transformer-distillation based frameworks report near state-of-the-art performance for both extractive and abstractive summarization. We establish the generalizability of the models through various experiments, while also showing the efficacy when learning with limited data.

6 Conclusions

We present a framework for auto- generation of ML models for extract and generate tasks by leveraging knowledge distillation, NAS, and transformer-distillation techniques. The proposed approach successfully creates new model architectures that are more efficient in terms of inference time and space while achieving near state-of-the-art performance in terms of accuracies across datasets for extractive and abstractive summarization. We believe our work can help create the foundation towards democratizing the use of deep-learning for NLP applications for non-experts in practice.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. 2020a. Adabert: Task-adaptive bert compression with differentiable neural architecture search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2463–2469. International Joint Conferences on Artificial Intelligence Organization. Main track.

Yen-Chun Chen, Zhe Gan, Yu Cheng, J. Liu, and Jing jing Liu. 2020b. Distilling knowledge learned in bert for text generation. In *ACL*.

Sumit Chopra, Michael Auli, and Alexander M. Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, San Diego, California. Association for Computational Linguistics.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Xuanyi Dong and Yi Yang. 2019. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1761–1770.

Greg Durrett, Taylor Berg-Kirkpatrick, and Dan Klein. 2016. Learning-based single-document summarization with compression and anaphoricity constraints. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1998–2008, Berlin, Germany. Association for Computational Linguistics.

K. Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, W. Kay, Mustafa Suleyman, and P. Blunsom. 2015. Teaching machines to read and comprehend. In *NIPS*.

Xiaoqi Jiao, Y. Yin, L. Shang, Xin Jiang, X. Chen, Linlin Li, F. Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *ArXiv*, abs/1909.10351.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Hanxiao Liu, K. Simonyan, and Yiming Yang. 2019. Darts: Differentiable architecture search. *ArXiv*, abs/1806.09055.

Yang Liu and Mirella Lapata. 2019a. Text summarization with pretrained encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3730–3740, Hong Kong, China. Association for Computational Linguistics.

Yang Liu and Mirella Lapata. 2019b. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*.

Laura Manor and Junyi Jessy Li. 2019. Plain English summarization of contracts. In *Proceedings of the Natural Legal Language Processing Workshop 2019*, pages 1–11, Minneapolis, Minnesota. Association for Computational Linguistics.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 3075–3081. AAAI Press.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018a. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018b. Ranking sentences for extractive summarization with reinforcement learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1747–1759, New Orleans, Louisiana. Association for Computational Linguistics.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104, Stockholmsmässan, Stockholm Sweden. PMLR.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2018. Regularized evolution for image classifier architecture search.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. volume 70 of *Proceedings of Machine Learning Research*, pages 2902–2911, International Convention Centre, Sydney, Australia. PMLR.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal. Association for Computational Linguistics.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Kaiqiang Song, Bingqing Wang, Zhe Feng, Ren Liu, and Fei Liu. 2020. Controlling the amount of verbatim copying in abstractive summarization. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8902–8909. AAAI Press.

Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. 2018. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5369–5373. International Joint Conferences on Artificial Intelligence Organization.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Yujing Wang, Yaming Yang, Yi-Ren Chen, Jing Bai, Ce Zhang, Guinan Su, Xiaoyu Kou, Y. Tong, M. Yang, and L. Zhou. 2020. Textnas: A neural architecture search space tailored for text representation. In *AAAI*.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256.

S. Xie, H. Zheng, Chunxiao Liu, and L. Lin. 2019. Snas: Stochastic neural architecture search. *ArXiv*, abs/1812.09926.

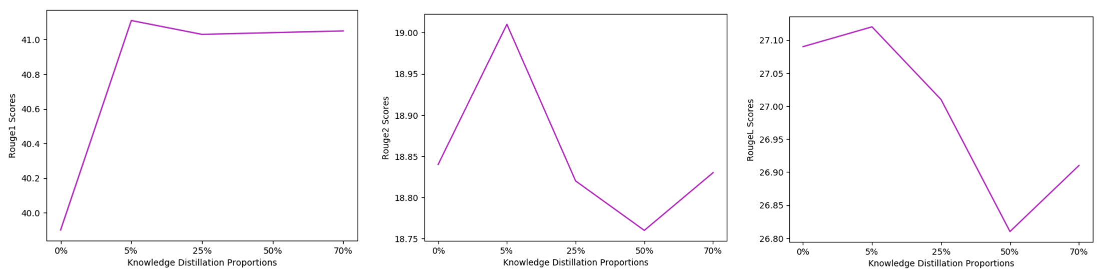
Jingqing Zhang, Y. Zhao, Mohammad Saleh, and Peter J. Liu. 2019. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. *ArXiv*, abs/1912.08777.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning.

A Additional Results

KD Proportion variation: Figure 9 illustrates the variation in performance as we vary the KD proportion of loss during training of models through AUTOSUMM-CREATE. It is to be noted that the models with KD perform better than the models without KD. However, it cab be observed that the performance reaches a saturation after certain point and increasing KD-proportion might not better the performance.

Model Architecture: Figure 10 illustrates an example of a model created with NAS in AUTOSUMM-CREATE. It is visualised through tensorboard.



(a) Rouge-1 (b) Rouge-2 (c) Rouge-L

Figure 9: Variation of performance with the increase in KD proportion on CNN DM dataset

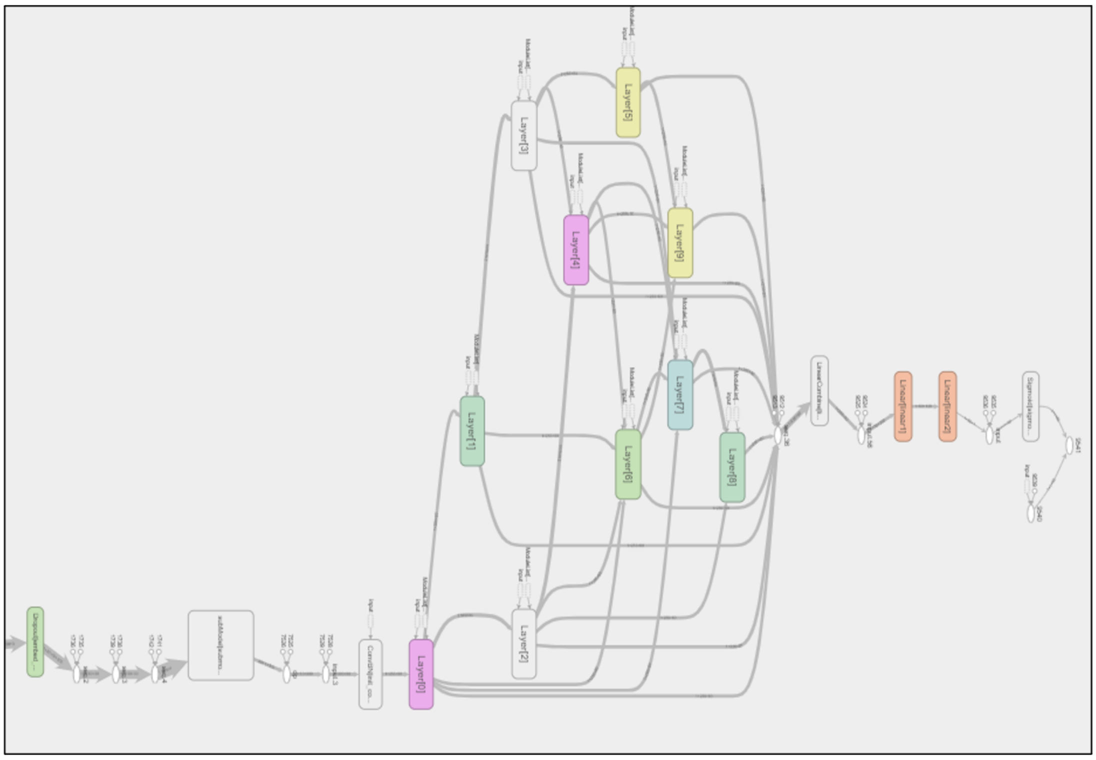


Figure 10: Model created with NAS module in AUTOSUMM-CREATE, as visualised through tensorboard