M^2SN: Adaptive and Dynamic Multi-modal Shortcut Network Architecture for Latency-Aware Applications

Yifei Pu
Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
pkq2006@sjtu.edu.cn

Chi Wang
Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
wangchi@sjtu.edu.cn

Xiaofeng Hou*
Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
hou-xf@cs.sjtu.edu.cn

Cheng Xu
Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
jerryxu@sjtu.edu.cn

Jiacheng Liu
Computer Science and Engineering
Chinese University of Hong Kong
Hong Kong, China
jcliu@cse.cuhk.edu.hk

Jing Wang
Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
jing618@sjtu.edu.cn

Minyi Guo
Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
guo-my@cs.sjtu.edu.cn

Chao Li*
Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
lichao@cs.sjtu.edu.cn

***Abstract*—Multi-modal neural networks have demonstrated exceptional performance by merging information across modalities, surpassing the state-of-the-art uni-modal DNNs. However, this accuracy improvement comes at the cost of increased computation, leading to higher inference latency. This defect significantly limits the practical value of multi-modal DNNs, especially for latency-aware applications. Therefore, we propose an adaptive and efficient multi-modal shortcut architecture called M^2SN to reduce the execution latency with accuracy guarantees. It skips ineffective network layers to reduce computational costs as well as alleviate the overfitting problem adaptive to specific models and scenarios. The key contributions of M^2SN are twofold: 1) We design and insert shortcuts into each uni-modal network to perform adaptive computing. 2) We design a navigator to dynamically choose the optimal shortcuts. Unlike previous approaches, M^2SN features high generality as it does not rely on any prior knowledge. The experimental results show that M^2SN can reduce 28.3% average latency while obtaining the same or higher accuracy compared with SOTA baselines.**

***Index Terms*—Multi-Modal Network, Dynamic Network, Latency-Aware Optimization**

I. Introduction

Recently, there is a growing trend to replace traditional uni-modal networks with multi-modal neural networks [1], [2], [3] for their performance superiority. Multi-modal DNNs generally consist of multiple encoder DNNs to extract representations from different modalities and a fusion and head DNN to fuse the modality features. By federating information from different modalities, multi-modal neural networks can outperform the SOTA uni-modal neural networks with up to 30% accuracy in different multimedia applications [4].

Despite the performance advantage, multi-modal networks also possess certain shortcomings. The first challenge is the high computation cost. Multi-modal networks inevitably introduce repetitive encoder and extra fusion operations, which result in unacceptable latency in latency-aware applications [5]. Taking MM-IMDb dataset [1] as an example, using multiple modalities gains less than 1% additional accuracy but brings more than $2\times$ additional inference latency [6]. Secondly, multi-modal networks may even achieve lower accuracy than uni-modal networks in some cases. Fusing modalities on the deepest features of networks may not produce the correct results due to over-fitting problems. For example, The accuracy using the fused feature produced by the language, visual, and audio modalities is lower than merely using the language modality on the MUStARD dataset [1], [6].

To solve these challenges, a dynamic and flexible execution strategy is required. If some ineffective steps in the multi-model networks can be simplified or skipped, the computational overhead can be greatly reduced. In addition, one can gain better accuracy from an adaptive decision of how to skip each layer and modality. Although previous works can exit the computation [7], they still lack a mechanism that enables skipping at the granularity of network layers.

*Corresponding author

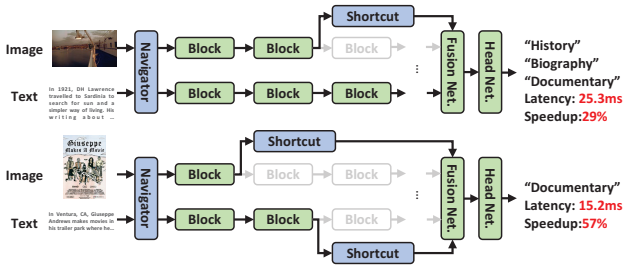

Figure 1: M^2SN can adaptively predict correct labels for different data samples in MM-IMDb with significant speedup.

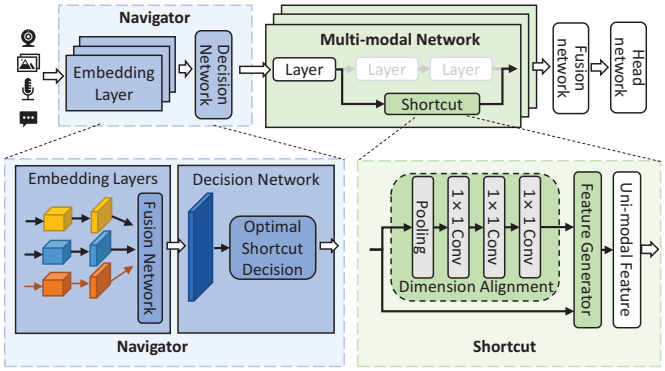

Figure 2: An overview design of M^2SN

In this paper, we propose an adaptive and dynamic multi-modal shortcut network architecture (M^2SN) to achieve low overhead with high accuracy. We show an example design and results in Figure 1. The key design consideration behind M^2SN is that features from early layers in different modalities are adequate to provide correct representations. By intelligently choosing the modality features and fast skipping from early layers, we can achieve satisfying performance with minimal computation. There are two main modules in our design, namely the shortcut and the navigator. We build shortcut modules and insert them into each modality relatively to produce the deepest features and skip the following steps. We design a navigator to adaptively and dynamically choose the optimal shortcuts. Unlike some previous adaptive multi-modal network architectures, M^2SN is a general method for multi-modal networks using the late fusion method, which does not rely on prior knowledge.

In this work, we make the following contributions:

- We propose M^2SN, an adaptive, general multi-modal shortcut architecture to reduce the execution latency of multi-modal networks while keeping or even improving the accuracy.
- We design specific shortcut modules for different modality networks to directly propose the final extracted figure after an arbitrary layer.
- We design a lightweight navigator to adaptively choose the optimal shortcut for different samples.
- We evaluate M^2SN on several real-world multi-modal neural networks and datasets. M^2SN can reduce 28.3% average latency with the same or higher accuracy.

II. Related Work

Multi-modal Neural Network. Multi-modal neural networks [2], [8] merge information from different modalities like image, text, audio, video, etc to prduce the final result. They have been demonstrated to perform much better than uni-modal networks in many application fields [1], [6]. Late fusion methods [9] are widely used in multi-modal networks, which extract features from the modalities separately and then fuse them to solve the tasks. *M^2SN can be applied to various multi-modal neural networks based on late fusion methods to reduce the inference latency while keeping the high accuracy.*

Adaptive Multi-modal Network. Many adaptive methods have been recently proposed to improve the computational efficiency of multi-modal networks. SkipNet [10] proposes an adaptive architecture of a uni-modal network that can skip some of the layers while keeping the accuracy, which can easily apply to multi-modal networks. AdaMML [11] is designed to adaptively drop some input modalities for efficient multi-modal learning in video recognition tasks. MMExit [7] implements a 2D early exit architecture, which sorts the modalities based on their importance and maps the 2D exit space into a linear space. *Differently, M^2SN is a general adaptive multi-modal network architecture that does not limit to the task, and can reduce the inference latency without the dependence of priori knowledge.*

III. M^2SN Architecture Design

A. Overview of M^2SN Architecture

To reduce the inference latency of multi-modal neural networks, we propose M^2SN, an adaptive multi-modal neural network architecture to ensure satisfying performance with minimal inference latency. M^2SN is based on late fusion multi-modal neural network architecture [9] and can be applied to arbitrary late fusion multi-modal networks. A late fusion multi-modal neural network uses several uni-modal networks to extract respective features from each modality separately, then fuses the features to solve the multi-modal tasks.

As shown in Figure 2, we design a module to transfer the early feature into the deepest feature, called **Shortcut**. A shortcut module connects a layer and the tail of the uni-modal network, the input of a shortcut is the feature obtained after an early layer, and the output is the deepest feature of the uni-modal network, which can be directly sent to the fusion network. While using the shortcut module, we ignore the rest of the layers in the uni-modal network and produce the deepest feature using the feature extracted by the early layer.

While shortcut can significantly reduce the inference latency, the next critical problem is how to keep the correctness of prediction while using shortcut. To solve this problem, we design **Navigator**, a lightweight module that dynamically decides the optimal shortcut for the corresponding input data.

B. Shortcut Design

In the design of M^2SN, shortcut is a lightweight network module that maps the feature obtained by the early layers into the deepest feature of a uni-modal network. Shortcut module behaves like a shortcut in the real world, skipping the original network layers with high latency and going straight to the end of the uni-modal network. Let x be the input of the uni-modal network, F_i be i-th layer in the network, the original uni-modal network can be denoted as

$$y = F^m\left(F^{m-1}\left(\cdots F^1\left(x\right)\right)\right) \tag{1}$$

where m is the number of layers in the uni-modal network.

We use x_i to represent the output feature of the i-th layer in the network, then the shortcut connecting the i-th layer and the end of the uni-modal network can be denoted as

$$y = S^i\left(x_i\right) = F^m\left(F^{m-1}\left(\cdots F^i\left(x_i\right)\right)\right) \tag{2}$$

which means we use a shortcut module to replace the rest layers after the i-th layer.

There are two critical problems in the shortcut design. The first problem is that shortcut must be lightweight, the time cost of a shortcut module should be much lower than the original skipped network layers. The second problem is correctness, the feature obtained by a shortcut module must achieve the same effect as the feature extracted by the original uni-modal network.

As shown in Figure 2, to solve these problems, we design a shortcut module with two paths. In some networks, such as convolutional networks, the dimension of the input feature does not match the output feature. For these networks, our shortcut first uses a dimension alignment module that utilizes one large-stride pooling layer and three 1×1 convolutional layers to align the dimension of the input and output features. For other uni-modal neural networks, such as transformer, the input feature has the same dimension as the output feature, so the shortcut module in these networks can skip the dimension alignment module.

After the alignment module is a feature generator, which transfers the aligned input feature into the deepest feature of the original uni-modal network. To keep the lightweight of shortcut module, we use a linear layer to undertake this task. We show the feasibility of this design in Section V below.

C. Navigator Design

In the design of M^2SN, navigator is a lightweight network module that leverages the input data samples to choose the optimal shortcuts. The main task of a navigator module is choosing the shortcuts that can produce correct results with the least latency, and the time cost of navigator should be much lower than the total latency of the multi-modal network.

As shown in Figure 2, in the design of navigator module, we utilize multiple lightweight embedding layers to obtain the corresponding embedding of each modality, fuse the embeddings using a fusion network, and design a decision network to produce the optimal shortcut decision. Specifically, in the implementation of navigator, we use a short ResNet [12] with a few blocks for image/video modality and a few attention layers for language modality to construct the embedding layers. After fusing the embeddings by a concatenate layer, we use a linear layer with softmax activation function to produce the optimal shortcut decision.

D. Inference Algorithm

In the inference process of M^2SN, for an input data sample, we first send it to navigator module to obtain the corresponding embeddings of each modality and produce the optimal shortcut decision. While the uni-modal neural networks receive the input data sample and the shortcut decision, the encoder layers of each uni-modal network start to execute until meet the chosen shortcut, then the shortcut module transfers the feature obtained by the first several encoder layers into the deepest feature of the uni-modal network, and send it to the multi-modal fusion network. After all the uni-modal networks finish the process, the multi-modal fusion network fuses the features and produce the prediction result by a head network.

IV. M^2SN Training Method

The training process of M^2SN consists of two stages. The first stage is to train the parameters of the multi-modal network part, which contains the shortcut modules, the uni-modal networks, and the fusion network. The second stage is to train the navigator, which is used to decide the choice of shortcuts.

A. Shortcut Module Training

For an existing multi-modal neural network, we first choose some layers in each uni-modal network to be the shortcut points. A layer in a uni-modal network is chosen to be an shortcut point means we add a shortcut module that transforms the feature extracted by the layer to the final feature extracted by the whole uni-modal network directly. We define the layers between two shortcut points as a block.

As we can see in Section III, the output of a shortcut is a feature with the same size as the original feature extracted by the uni-modal network. It is hard to design a loss function that measures the difference between two features, so we design a task head to directly predict the result of the whole multi-modal network using the uni-modal feature. While training the parameters of shortcuts, we just need to predict the result by the shortcut and the corresponding task head, then calculate the task loss and update the parameters with gradient descent.

Two training methods are then introduced to train the shortcut module. The integral training method performs better, while the peripheral training method can save most of the training time using the frozen pre-trained model.

1) Integral Training Method: In the integral training method, we use the parameters of the pre-trained multi-modal model to help train our shortcuts, but we do not freeze the pre-trained model during the training process. As shown in Algorithm 1 (Line 3-9), we 1) predict the label by the first block with pre-trained parameters, the first shortcut module, and the first

Algorithm 1: M^2SN Training Algorithm

Input: Pre-trained model $\mathcal{M}_0$, preference λ
Output: Trained model $\mathcal{M}$, trained navigator $\mathcal{N}$

1 **for** $i = 1, \cdots, n$ **do**
2 **for** $j = 1, \cdots, m_i$ **do**
3 **if** *Integral Training* **then**
4 **for** $k = 1, \cdots, e$ **do**
5 *//Train j-th shortcut in i-th modality using k-th epoch.*
6 Predict label by j-th shortcut .
7 Calculate the task loss.
8 Update j-th shortcut and block parameters.
9 Freeze j-th block parameters.
10 **if** *Peripheral Training* **then**
11 **for** $k = 1, \cdots, e$ **do**
12 Predict label by j-th shortcut .
13 Calculate the task loss.
14 Update j-th shortcut parameters.
15 **for** $i = 1, \cdots, e$ **do**
16 Decide shortcut weights by navigator.
17 Predict label using weighted features.
18 Calculate loss according to Eq. (4).
19 Update $\mathcal{N}$ parameters.
20 Anneal temperature parameter τ.

task head; 2) calculate the task loss; 3) update the parameters of the first block and shortcut. When the parameters converge after we use several epochs of data samples to train, we just freeze the parameters of the first block, and then start to train the next block and the next shortcut. We repeat the training processes one by one until all the shortcut modules and uni-modal networks are trained.

2) Peripheral Training Method: However, integral training leads to extremely long training time in real practice. In integral training, we must re-train the parameters of the whole multi-modal network, even if we already have the pre-trained model. So we propose another training method to reduce the training time, called peripheral training. As shown in Algorithm 1 (Line 10-14), in this method, we use the pre-trained parameters of the multi-modal network to train our shortcut modules. While training the parameters of shortcut modules, the parameters of the pre-trained multi-modal model are frozen, only the parameters of shortcut modules are updated in the backward propagation process.

B. Navigator Training

The training process of the navigator starts after training all the shortcut modules. The direct predicted result of the navigator is the chosen probabilities of shortcuts, and we choose the shortcut with the highest probability to produce the one-hot decision in the inference part, which can be denoted as

$$e_i = \text{one-hot}\left(\arg\max_j(y_{i,j})\right) \tag{3}$$

However, we cannot use the one-hot decisions in the training process, since it is not directly differentiable. As shown in Algorithm 1 (Line 15-20), we predict the chosen probabilities of the shortcuts and adopt the probabilities as computational weights. Then for each uni-modal network, we produce the features by each shortcut using the trained shortcut network and fuse the features using the weights of corresponding shortcuts to be the extracted feature of this uni-modal network. We use the weighted fused features to predict the task result, calculate the loss introduced below, and update the parameters.

1) Training Loss with Latency Awareness: To keep the prediction accuracy while reducing the inference latency, we design a loss function to help train the navigator:

$$\mathcal{L} = \mathcal{L}_{task} + \lambda \sum_{i=1}^{n} Latency_{i,e_i} \tag{4}$$

where $Latency_{i,e_i}$ include the inference latency of the layers before the e_i-th shortcut point in the i-th modality and the latency of the j-th shortcut module.

In this loss function, we use the original task loss to evaluate the prediction accuracy of the decision made by the navigator and use the inference latencies of all the uni-modal networks to control the latency. Parameter λ is used to control the importance of latency. Lower λ may lead to higher accuracy with lower latency.

2) Optimization using Gumbel-Softmax: Gumbel-Softmax [13] is used to make the one-hot decisions differentiable. Although we use the fused features instead of a one-hot chosen feature to avoid the problem of gradient descent, Gumble-Softmax is also useful to control the training process. We transform the probability vectors with the following form:

$$\hat{e}_{i,j} = \frac{\exp((\log(y_{i,j}) + G_{i,j})/\tau)}{\Sigma_j \exp((\log(y_{i,j}) + G_{i,j})/\tau)} \tag{5}$$

where $G_{i,j}$ are samples independently drawn from the distribution $Gumbel(0, 1)$, and τ denotes the softmax temperature.

When we use a large τ to be the temperature, the distribution of $\hat{e}$ is similar to a uniform distribution, and a small τ leads to a categorical distribution. During the training process, we anneal τ from a large value, which means we adopt approximately uniform probabilities to each shortcut at first, and $\hat{e}$ becomes similar to a one-hot vector with the decrease of τ, which means we obtain a certain chosen of the best shortcut.

Research Area	Size	Dataset	Modalities	# Samples	Task
Affective Computing	S	MUStARD	{l, v, a}	690	sarcasm
	M	CMU-MOSI	{l, v, a}	2,199	sentiment
	L	CMU-MOSEI	{l, v, a}	22,777	sentiment, emotions
Multimedia	S	Kinetics-S	{v, a, o}	2,624	human action
	M	MM-IMDb	{l, i}	25,959	movie genre
	L	Kinetics-L	{v, a, o}	306,245	human action

Table I: Description of the used M^2C tasks and datasets.

V. Evaluation

A. Experimental Setup

Datasets. We use 6 representative multi-modal datasets provided by the *Multibench* [1] and the *MMBench* [6] benchmark. As shown in Table I, the sizes of data samples range from 690 to 306,245. These 6 datasets cover the most common modalities, including *image, language, audio, video* and *optical*

Type	Scheme	Description
Uni-modal Networks	*image* [12]	ResNet
	audio [14]	Librosa
	language [15], [16]	BERT/GloVe
SOTA M2C Model	*LF (BL)* [6]	Concatenation (Baseline)
	MMExit [7]	*MMExit* schema

Table II: SOTA M^2C models for performance evaluation.

flow, and the corresponding multi-modal classification tasks include *sarcasm, sentiment, emotions, human action* and *movie genre*.

Implementation. We use the uni-modal networks mentioned in Table II to extract the features of different modalities. We implement M^2SN by adding 3 shortcuts per uni-modal neural network using late fusion networks. For ablation studies, we implement *Fixed Shortcut (FS)* method that chooses the fixed shortcuts instead of the decision shortcuts by the navigator to compare with our method, and implement *Non-Latency (N-L)* method that sets the parameter λ in Eq. (4) to 0 in order to ignore the effect of inference latency. We train and run models on a server with one GeForce RTX 2080Ti GPU.

Baselines and the State-Of-The-Art We implement *Late Fusion (LF)* method as our baseline. In each of the modalities, we use only the uni-modal model ($uni_1 - uni_3$) mentioned in Table II and connect it to the head network to obtain the prediction. We also implement the state-of-the-art method *MMExit* [7] and compare it with the performance.

B. Experimental Results

1) Comparison with Baselines and SOTA: **Latency Reduction.** The purpose of the shortcuts module and navigator module we designed is to reduce the latency of multi-modal networks. In order to verify whether they play their role and whether the cost is worth it, we conducted delay comparisons on six datasets over four sets of models. Since the inference time between different model groups is not at the same level, for better demonstration, we use the total inference time of *LF* as baseline, and divide the inference time of M^2SN by the baseline to achieve a unified quantitative comparison. The green part in Figure 3 is the latency reduction percentage. It is obvious that the module design has achieved our ideal effect. The latency cost of these modules accounts for between 12% and 16% of the baseline inference time, and the improvement ratio it brings ranges from 16% to 42%. In view of different datasets characteristics also differ. Our results here can prove to some extent that the design of these modules throws light on the latency reduction problem.

Performance Evaluation. In order to verify the performance of our method, as shown in Table III we compare the accuracy and inference time of different uni-modal and multi-modal models in different datasets. M^2SN shows significant improvement in both accuracy and inference latency compared with baseline (*LF*). For the special datasets that obtain higher accuracy in uni-modal models than the LF method, M^2SN can also achieve higher accuracy than the best uni-modal models. When comparing with the state-of-the-art method *MMExit*, M^2SN achieves fewer inference latencies in all 6 datasets and

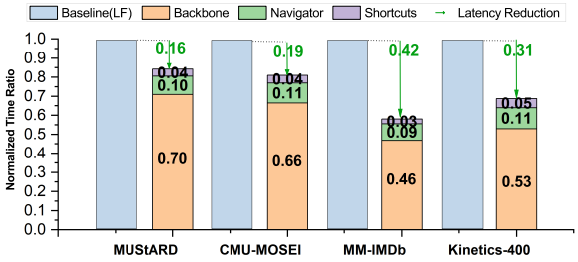

Figure 3: Relative inference latency of M^2SN and the baseline *LF*, and the latency proportion of modules in M^2SN.

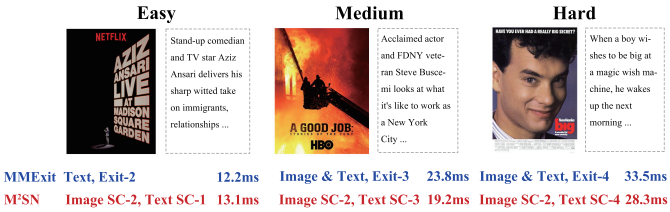

Figure 4: Latency comparison between M^2SN and *MMExit* with various difficulties, SC-N refers to Shortcut position.

obtains higher accuracy in 4 of the 6 datasets with the default parameters. In the rest 2 datasets, the accuracy loss is small, and it is easy to achieve higher accuracy than *MMExit* by ignoring the awareness of latency, M^2SN(N-L) shows higher accuracy with close latency compared to *MMExit*. The performance of *Fixed Shortcut (FS)* shows the necessity of navigator, the fixed choice of shortcuts leads to a much worse result both in accuracy and latency than the dynamically chosen shortcuts by navigator, even if we adopt the best shortcuts.

Hard Sample Resolver. It is easy to think that the inference latency of M^2SN and *MMExit* will increase a lot in some hard data samples because the models need more original layers to produce the exact features. As shown in Figure 4, we build 3 sample sets of the MM-IMDb dataset with the difficulties of the data samples, from **"Easy"** to **"Hard"**. We give an example from each of the three sets and show that as the difficulty increases M^2SN can get better latency reduction. In **"Hard"** set, the average latency of *MMExit* is 33.2ms, which is close to the latency of the baseline *LF* (35.3ms), while M^2SN can still keep a low average latency 28.5ms.

2) Ablation Studies: **Impact of Training Method.** During the training process of the shortcut module, we mentioned two training methods in Section IV-A. To compare the training cost and the effect of improvement between the integral training and the peripheral training method, we conducted corresponding experiments in MM-IMDb dataset. It is intuitive that more effort will bring higher returns. As shown in Table IV, the training time of the integral training method is 3-5 times, and training resource requirements are also at the same magnitude, resulting in improved accuracy.

Impact of Latency Awareness. Obviously, the choice of λ in our navigator design is very important, related to whether it can

Model	MUStARD			CMU-MOSI			CMU-MOSEI			Kinetics-S			MM-IMDb			Kinetics-L		
	Acc	F1	Time	Acc	F1	Time	Acc	F1	Time	Acc	F1	Time	Acc	F1	Time	Acc	F1	Time
uni_1	0.605	0.611	27.5	0.741	0.688	358.5	0.613	0.565	3652.3	0.573	0.554	269.3	0.556	0.434	10.8	0.716	0.673	269.3
uni_2	0.481	0.472	25.3	0.487	0.435	340.1	0.429	0.381	3481.5	0.386	0.352	902.3	0.262	0.104	20.5	0.187	0.155	902.3
uni_3	0.548	0.552	22.5	0.439	0.402	341.0	0.407	0.352	3551.3	-	-	-	-	-	-	-	-	-
LF	0.539	0.528	43.3	0.753	0.721	570.1	0.615	0.573	5902.2	0.583	0.568	1271.6	0.563	0.452	35.3	0.705	0.659	1271.6
MMExit	0.623	0.622	40.7	0.758	0.702	523.3	0.623	0.588	5172.8	0.579	0.571	978.2	0.572	0.459	25.6	0.702	0.662	978.2
M^2SN	**0.621**	**0.622**	**36.5**	**0.762**	**0.718**	**462.3**	**0.622**	**0.587**	**4782.3**	**0.607**	**0.582**	**873.7**	**0.561**	**0.457**	**20.5**	**0.724**	**0.685**	**873.7**
M^2SN**(N-L)**	0.625	0.631	39.6	0.763	0.723	528.5	0.627	0.591	5207.2	0.612	0.597	994.5	0.575	0.468	25.3	0.735	0.698	994.5
FS	0.529	0.515	42.3	0.731	0.714	530.4	0.608	0.562	5472.3	0.565	0.558	1021.3	0.552	0.442	26.4	0.683	0.652	1107.3

Table III: Accuracy, F1 score, and inference latency of the baseline, SOTA, M^2SN and its ablation studies model in 6 datasets.

Train	Method	Accuracy	F1 Score	Infer time (ms)	Train time (s)	Peak memory (MB)	Train parameters (M)
Peripheral	*LF*	0.561	0.457	35.3	1638	1531	0.5
	M^2SN	0.573	0.461	20.5	1826	2251	11.1
Integral	*LF*	0.575	0.463	35.3	6532	7251	38.2
	M^2SN	0.582	0.472	19.8	8110	8766	49.3

Table IV: Shortcut module training method comparison.

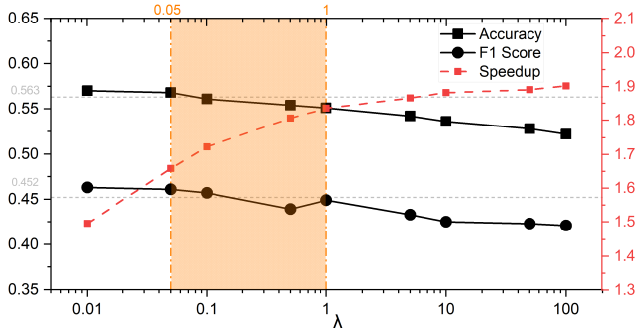

Figure 5: Training results with λ in Eq. (4) scales the latency. The gray horizontal line is the performance of baseline (*LF*), and the vertical orange area is a recommended range for λ to get a better trade-off between accuracy loss and latency gain.

choose a good shortcut. After experimental tests on the order of 0 to 100, we have currently obtained empirical results. As marked by the orange area in Figure 5, the acceleration effect is more obvious when the ratio of the cost part to task loss ranges from 0.05 to 1. and the accuracy loss is incomprehensible, but the improvement effect after 1 is weakening, and the accuracy loss begins to spike.

VI. Conclusion

Multi-modal networks, while significantly improving accuracy, also lead to an explosion in inference latency, which will make them difficult to deploy into many latency-aware real-world applications. To address this problem, we propose a novel multi-modal shortcut architecture called M^2SN. M^2SN adaptively adjusts its parameters according to latency constraints to achieve the highest accuracy within a limited latency. M^2SN can better apply multi-modal networks to next-generation latency-aware scenarios such as automated driving, mobile robotics, etc.

VII. Acknowledgements

We sincerely thank all the anonymous reviewers for their valuable comments that helped us to improve the paper. This work is supported by the Shanghai S&T Committee International Cooperation Project (No. 23510713200). Xiaofeng Hou is also sponsored by Shanghai Pujiang Program (No. 23PJ1405100). Corresponding authors are Xiaofeng Hou and Chao Li.

References

[1] P. P. Liang, Y. Lyu, X. Fan, Z. Wu, Y. Cheng, J. Wu, L. Y. Chen, P. Wu, M. A. Lee, and Y. Zhu, "Multibench: Multiscale benchmarks for multimodal representation learning," in *NeurIPS*, 2021.

[2] C. Zhang, Z. Yang, X. He, and L. Deng, "Multimodal intelligence: Representation learning, information fusion, and applications," in *STSP*, 2020.

[3] J.-M. Pérez-Rúa, V. Vielzeuf, S. Pateux, M. Baccouche, and F. Jurie, "Mfas: Multimodal fusion architecture search," in *CVPR*, 2019.

[4] J. Arevalo, T. Solorio, M. Montes-y Gómez, and F. A. González, "Gated multimodal units for information fusion," in *ICLR*, 2017.

[5] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "Transfuser: Imitation with transformer-based sensor fusion for autonomous driving," *PAMI*, 2023.

[6] C. Xu, X. Hou, J. Liu, C. Li, T. Huang, X. Zhu, M. Niu, L. Sun, P. Tang, T. Xu, K.-T. Cheng, and M. Guo, "Mmbench: Benchmarking end-to-end multi-modal dnns and understanding their hardware-software implications," in *IISWC*, 2023.

[7] X. Hou, J. Liu, X. Tang, C. Li, K.-T. Cheng, L. Li, and M. Guo, "Mmexit: Enabling fast and efficient multi-modal dnn inference with adaptive network exits," in *Euro-Par*, 2023.

[8] Z. Sun, P. Sarma, W. Sethares, and Y. Liang, "Learning relationships between text, audio, and video via deep canonical correlation for multimodal language analysis," in *AAAI*, 2020.

[9] Y. Wu, E. Y. Chang, K. C.-C. Chang, and J. R. Smith, "Optimal multimodal fusion for multimedia data analysis," in *ACMMM*, 2004.

[10] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in *ECCV*, 2018.

[11] R. Panda, C.-F. Chen, Q. Fan, X. Sun, K. Saenko, A. Oliva, and R. S. Feris, "Adamml: Adaptive multi-modal learning for efficient video recognition," in *ICCV*, 2021.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[13] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *ICLR*, 2017.

[14] B. McFee, C. Raffel, D. Liang, D. P. W. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *SciPy*, 2015.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *ACL*, 2019.

[16] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*, 2014.