Patches Are All You Need.

E4040.2024Fall.DLLM.report.lc3826.yt3019
Peter Chen, Yutao Mao
Columbia University

Abstract

Trockman and Kolter [1] *introduce ConvMixer, a novel architecture that operates directly on patches using only standard convolutional layers.*

Building upon their results, this paper 1) incorporate traditional training augmentation techniques into the ConvMixer network to evaluate potential performance improvements, 2) further investigate the impact of patch size parameters on model performance, and 3) examine whether weighted multiple kernel adjustment can enhance the final results and study the effect on different kernel size parameter.

1. Introduction

Trockman and Kolter [1] investigate the performance of vision models that rely on patch-based input representations. While Transformer-based architectures, such as the Vision Transformer (ViTs) [2], have recently outperformed traditional convolutional neural networks (CNNs) for vision tasks, the authors question whether the superior performance arises from the Transformer mechanism itself or from the patch-based input representation.

ConvMixer shares similarities with ViTs and MLP-Mixers by maintaining an isotropic structure: constant resolution and size throughout the network, separating spatial and channel mixing processes. But unlike ViTs, ConvMixer performs these operations solely with convolutional blocks, i.e. depthwise convolutions for spatial mixing and pointwise convolutions for channel mixing. Despite its architectural simplicity, ConvMixer demonstrates competitive performance, surpassing ViT, MLP-Mixer, and ResNet baselines under comparable conditions.

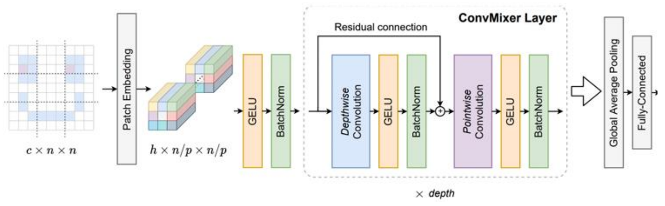

Fig. 1. Architect of ConvMixer training framework [1].

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

ConvMixer-Methodology. The ConvMixer architecture operates on patch-based input representations while utilizing standard convolutional layers to achieve competitive performance in vision tasks. The methodology consists of the following key components:

Patch Embedding. The input image $X \in \mathrm{R}^d$, where $d = c \times n \times n$, and c is the number of channels and $n \times n$ is the resolution, is divided into non-overlapping patches of size $p \times p$. Each patch is linearly embedded into a higher dimensional feature space using a convolutional operation with a kernel size $p \times p$, stride p, and h output channels, as

$$Z_0 = \mathrm{BN}\left(\mathrm{GELU}\left(\mathrm{Conv2D}(X, \mathrm{stride} = \mathrm{p}, \mathrm{kernel\ size} = \mathrm{p})\right)\right),$$

where $Z_0 \in$ R^{h $\times$ n/p $\times$ n/p} is the output tensor, Conv2D performs the convolution operation, GELU is the activation function, and BN refers to "Batch Normalization".

ConvMixer Embedding. The core of the ConvMixer architecture consists of a series of repeated blocks that mix spatial and channel information. Each ConvMixer block includes:

1. Depthwise Convolution:A depthwise convolution with a large kernel size k is applied to mix spatial information. The operation is followed by a GELU activation and Batch Normalization.

2. Residual Connection: The output of the depthwise convolution is added back to the input of the block via a residual connection to improve gradient flow and training stability.

3. Pointwise Convolution: A pointwise 1*1 convolution is used to mix channel information, followed by a GELU activation and Batch Normalization.

This combination of depthwise and pointwise convolutions enables efficient spatial and channel mixing, while the residual connection enhances the flow of information through the network. The ConvMixer layer is repeated d times, where d is the network depth.

Global Average Pooling and Classification. After the ConvMixer layers, global average pooling is applied to reduce the spatial dimensions, producing a feature vector of size *h*. This vector is passed through a fully connected layer to generate the final class predictions.

2.2 Key Results of the Original Paper

ConvMixer-Experiment Results. Their work includes experiments on ImageNet-1k and CIFAR-10, where ConvMixer achieves strong top-1 accuracy with fewer parameters. It's notable that larger kernel sizes for the depthwise convolutions play a critical role in its success, effectively enabling the model to mix distant spatial information. The authors suggest that the patch-based representation, rather than novel Transformer operations, is a significant factor behind the strong performance of modern vision architectures.

2.3 Related Works Discussions

We then zoom into the following related works to further discuss Trockman and Kolter [1]'s work:

Vision Transformers. Patch-based models have become increasingly prevalent in modern computer vision, as they offer computationally efficient solutions for processing high-resolution images. ViT [2] pioneered this approach by splitting images into non-overlapping patches and treating them as sequential tokens, applying self-attention mechanisms to capture global context. However, the quadratic cost of self-attention remains a computational bottleneck, motivating the development of alternative methods.

MLP-Mixer. Tolstikhin [3] et al. introduced a patch-based isotropic architecture that replaces self-attention with multi-layer perceptrons with competitive results. Despite its simplicity, MLP-Mixer requires extensive training data and lacks inductive biases like locality, which are crucial for data-efficient learning. In contrast, ConvMixer [1] preserves the isotropic structure but uses depthwise and pointwise convolutions to mix spatial and channel information. By leveraging convolutional inductive biases, ConvMixer achieves strong performance with fewer parameters and more efficient training compared to ViTs and MLP-Mixers.

CycleMLP and ResMLP. CycleMLP [4] and ResMLP [5] incorporate modifications to enhance patch-based architectures. CycleMLP focuses on improving dense prediction tasks, while ResMLP explores data-efficient training strategies. However, these methods remain reliant on MLP-based operations, which are less computationally efficient than convolutional alternatives.

Despite these advantages, ConvMixer still faces multiple problems:

Computational Efficiency. The use of larger kernel sizes in depthwise convolutions, while beneficial for capturing distant spatial information, may lead to increased computational demands. This could pose challenges in resource-constrained environments.

Overfitting Risks. The reliance on the combination of depthwise and pointwise convolutions may introduce risks of overfitting, especially transfering to tasks to the other datasets without using appropriate regularization techniques [6].

Hyper-parameter Tuning. Authors admitted that due to the lack of computational resource, extensive hyper-parameter tuning is not conducted. This could mean that the reported performance might not fully represent the model's optimal capabilities.

3. Methodology

Contributions. We aim to: **1)** incorporate traditional training augmentation techniques into the ConvMixer network to evaluate potential performance improvements, **2)** further investigate the impact of patch size parameters on model performance, and **3)** examine whether weighted multiple kernel adjustment can enhance the final results and study the effect on different kernel size parameters.

3.1. Pipeline

The pipeline of this project can be divided into four parts:

Dataset Adjustment. ConvMixer is known for its computational efficiency. However, as our project focuses on parameter engineering, it requires extensive experimentation to identify patterns. Given our limited GPU resources, we evaluate ConvMixer on a smaller dataset: Tiny-ImageNet [7], which contains 200 classes with 1,000 images per class.

Traditional DL Adjustment. We indeed find the overfitting trend when ConvMixer is transferring to Tiny-ImageNet. Therefore, we consider to apply traditional deep DL adjustment techniques (e.g. dropout, L2-regularization, etc.) to adapt the ConvMixer.

Patches. The performance of ConvMixer heavily relies on the choice of patch size, as it determines the internal resolution and the model's ability to mix spatial information effectively. However, this is not fully studied and explained in ConvMixer. In this project, we conduct a systematic exploration of various patch size configurations. Specifically, we test smaller patch sizes (e.g., p = 4, 7) and larger patch sizes (e.g., p = 11, 14) to evaluate their impact on accuracy, computational cost, and convergence speed. We aim to identify an optimal trade-off between model performance and computational efficiency for Tiny-ImageNet.

Dynamic and Static Kernel Adjustment. While ConvMixer employs static depthwise convolutional kernels with fixed sizes, we hypothesize that weighted multiple kernel adjustment during training can improve spatial information mixing and model generalization. To test this, we introduce a dynamic kernel adjustment mechanism that adapts the kernel size. We also studied the effect of static kernel size to the model.

4. Implementation and Results

The following sections contain the detail of discussion and implementation for pipeline 2,3,4 mentioned above.

4.1 Traditional DL Technique

Since we switched our dataset to Tiny-ImageNet, it is unnecessary to use the same large-scale parameters originally designed for larger datasets. We therefore scale down the parameters proportionately to match the reduced size of Tiny-ImageNet (without using any DL techniques). However, we observe that ConvMixer suffers from severe overfitting on this smaller dataset:

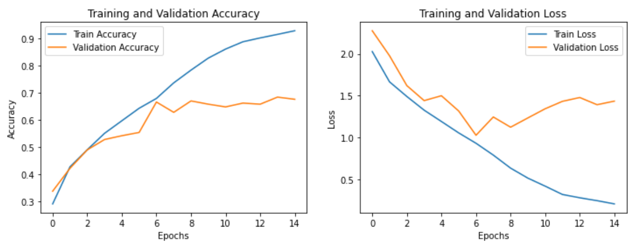

Fig. 2. ConvMixer training results with the following hyperparameters: dim=512, depth=8, kernel_size=9, patch_size=7.

Even though the training accuracy reaches 92.28%, the validation accuracy drops significantly (67.6%), comparing the accuracy reported in the original paper (81.37%). Additionally, the loss curve indicates a clear trend of overfitting.

Discussions of the Preliminary Result. Based on the previous analysis, we reduced the parameter size and dataset size, and transferred the implementation from PyTorch to TensorFlow. However, this process introduced several factors that could contribute to the observed decrease in accuracy:

1. *Parameter Reduction*
ConvMixer's original architecture was optimized for larger datasets like ImageNet-1k. Scaling down the parameters (e.g., reduced depth, dimension, and kernel size) to match Tiny-ImageNet may have reduced the model's capacity to effectively learn complex features, resulting in a performance drop. Smaller parameter budgets may not capture sufficient spatial and channel information, especially for datasets with fine-grained classes.

2. *Dataset Size*
Tiny-ImageNet, being a smaller dataset, contains fewer examples per class. This limited data increases the risk of overfitting, as the model has fewer examples to generalize well. Additionally, ConvMixer, which benefits from strong regularization techniques on larger datasets, may not adapt as effectively to smaller datasets without careful tuning.

3. *Framework Transfer from torch to tensorflow*
The original ConvMixer implementation used PyTorch with the timm library, which provides optimized training procedures, data augmentations, and regularization strategies. Transferring the implementation to TensorFlow introduces the following challenges:

Lack of Prebuilt Tools. TensorFlow lacks a direct equivalent to the \texttt{timm} library, which means that certain augmentation techniques, learning rate schedules, or optimizers may not have been implemented with the same level of optimization.

Numerical Differences. Differences in numerical precision, initialization methods, and computation kernels between PyTorch and TensorFlow could result in variations in training dynamics and convergence.

Technique-fine-tuned Result. We optimize the results by implementing various deep learning techniques and hyperparameter tuning. Specifically, we experimented with different combinations of dropout, L2-regularization, and early stopping. However, significant performance improvement was observed after switching the optimizer to AdamW. As shown in Figure 3, the validation accuracy increased from 67.6% to 76.4%. Typically, we can clearly see that the effect of overfitting is reduced with AdamW. We provide a discussion to this new result below. Pseudocode implementation is shown in the appendix.

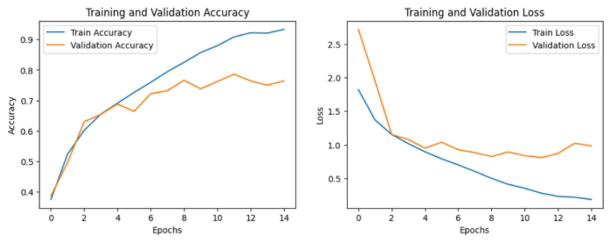

Fig. 3. ConvMixer training results with AdamW optimizer.

Why AdamW Improves Performance. AdamW combines the benefits of the Adam optimizer with decoupled weight decay regularization. Traditional Adam tends to overfit on smaller datasets like Tiny-ImageNet because its weight decay is coupled with the learning rate schedule, resulting in suboptimal regularization. In contrast, AdamW applies weight decay independently of the gradient updates, ensuring more effective regularization. This helps reduce overfitting, particularly when training models like ConvMixer with limited data.

Is This Good Enough? No. Even though our results are closer to the original accuracy reported in the paper, it is important to note that we are working on a smaller dataset with fewer categories. This inherently reduces the difficulty of the task compared to the original large-scale dataset, making direct comparisons less meaningful. Achieving comparable performance on Tiny-ImageNet does not necessarily indicate that the model is performing optimally or generalizing well.

4.2 Extended Study on Patch Size

General Discussions on Patch Sizes. The patch size in ConvMixer significantly impacts model performance, efficiency, and generalization. Smaller patches retain more spatial detail but increase computational cost and risk overfitting, especially on smaller datasets like Tiny-ImageNet. Larger patches reduce computational burden but may lose fine-grained features, leading to underfitting.

Patch size also determines the receptive field within depthwise convolutions, influencing how spatial information is mixed. Smaller patches require deeper networks or larger kernels to mix information effectively, while larger patches naturally offer a broader context but limit the model's granularity.

Finally, ConvMixer's original patch sizes were optimized for large datasets. For smaller datasets, re-evaluating patch size helps balance computational efficiency and accuracy while mitigating overfitting.

We trained four trails with different batch sizes while leaving on the rest parameters (kernel size, etc.) same. The experiment was running on the original ConvMixer with Adam optimizer:

TABLE I
EXPERIMENT RESULTS ON DIFFERENT PATCH SIZE NUMBER

	Trail 1	**Trail 2**	**Trail 3**	**Trail 4**
Patch size	14	11	7	4
#Para (10^6)	2.77	2.66	2.55	2.5
Time (s)	626.79	938.77	3205.98	7658.9
Acc	57.4%	63.4%	67.6%	67.2%

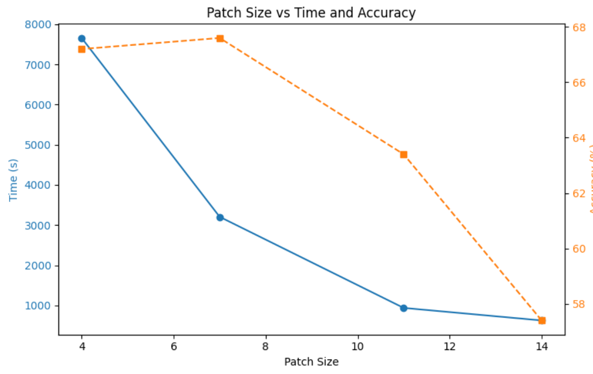

Fig. 4. Patch size experiment with original ConvMixer setting.

Discussions. This trend highlights a trade-off: smaller patch sizes allow the model to retain more spatial detail, leading to better accuracy. However, this comes at the cost of increased computational overhead, as indicated by the sharp rise in training time. Additionally, the parameter count decreases slightly with smaller patch sizes, which suggests that the improved accuracy is not merely due to an increase in model complexity but rather to the richer spatial features captured by smaller patches.

The results indicate that a patch size of 7 achieves the best balance between accuracy and computational cost. Specically, smaller patches (e.g., 4) cannot no longer provide marginal accuracy gains, along with their significant increase in training time may limit their practical use.

4.3 Influence from Kernel Selection and Size towards Model Training

The interplay between patch size and kernel size in ConvMixer directly influences the model's ability to mix spatial and channel information effectively. While patch size determines the resolution of the input representation, kernel size controls the receptive field within each ConvMixer layer.

Dynamic Weighted Kernel. Using multiple kernels simultaneously allows a convolutional layer to view the input at different scales at once. Smaller kernels capture fine details, while larger ones provide broader context. By combining their outputs, the model can automatically learn how much attention to give each kernel size. This leads to richer feature representations, possibly improving the training accuracy. Instead of relying on a single, fixed receptive field, the layer dynamically mixes information across multiple scales, allowing the network to discover more nuanced patterns and ultimately enhancing performance and robustness. This ideas were used in GoogLeNet [9] and SKNet [10] we will then implement it to ConvMixer to see its performance.

Further Static Kernel Studies. Statically changing the kernel size provides a simpler and computationally stable approach. By explicitly selecting a fixed kernel size, the model's receptive field can be tuned to specific datasets. Specifically, MixConv [8] has demonstrated that a kernel size of 9*9 achieves an optimal balance between local and global feature extraction. The study highlights that smaller kernels (e.g., 3*3) are insufficient for capturing long-range dependencies, while overly large kernels increase computational overhead without substantial gains.

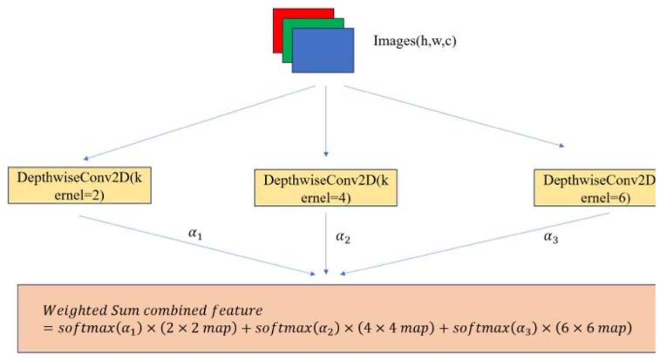
Images(h,w,c)
DepthwiseConv2D(kernel=2)
DepthwiseConv2D(kernel=4)
DepthwiseConv2D(kernel=6)
α_1
α_2
α_3
Weighted Sum combined feature
$= softmax(\alpha_1) \times (2 \times 2\ map) + softmax(\alpha_2) \times (4 \times 4\ map) + softmax(\alpha_3) \times (6 \times 6\ map)$

Fig. 5. Illustration of dynamic weighted kernel framework

Setup - Dynamic Weighted Kernel. To incorporate multiple kernel sizes simultaneously, we use parallel depthwise convolutions with kernel weights W_k for $k = \{1, 2, \ldots, n\}$, applied to the input X, producing outputs $O_k = X * W_k$.

Trainable mixing weights α_k are converted to probability via softmax:

$$\widetilde{\alpha_k} = \frac{e^{\alpha_k}}{\sum_{j=1}^{n} e^{\alpha_j}},$$

The final output O is computed as a weighted sum of these outputs:

$$O = \sum_{k=1}^{n} \widetilde{\alpha_k}\, O_k = \sum_{k=1}^{n} \widetilde{\alpha_k}(X * W_k)$$

This enables dynamic adaptation to different scales, enhancing feature extraction and representation.

We tested combination of kernel sizes {2, 4, 6, 8, 10}, and observed an improvement in validation accuracy from 67.6% to approximately 70%, indicating the effectiveness of the method.

However, the dynamic weighted kernel approach increased training time, and due to inefficient computation resources, we cannot fully test different combinations of kernal size. This opens a new direction for future research, focusing on optimizing the multiple-kernel-size range and selection to balance accuracy and computational efficiency. Pseudocode implementation are shown in the appendix.

Setup - Static Kernel Size. Similar to patches experiment, we tried 4 trails with different kernel sizes by controlling all other parameters same. Results are shown in Table II.

TABLE II
EXPERIMENT RESULTS WITH DIFFERENT KERNEL SIZES K

	Trail 1	**Trail 2**	**Trail 3**	**Trail 4**
K	2	6	10	14
#Para (10^6)	2.23	2.36	2.63	3.02
Time (s)	1073.87	1221.63	1545.17	3781.46
Acc	52.8	65.0	68.8	64.8

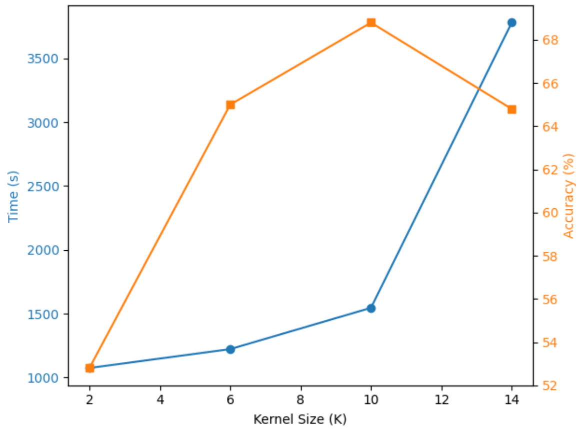

Fig. 6. Kernel size experiment with original ConvMixer setting.

Discussions. As the kernel size increases, the accuracy improves from 52.8% at $K = 2$ to a peak of 68.8\% at $K = 10$. However, further increasing K to 14 results in a slight drop in accuracy to 64.8%. This trend suggests that there exists an optimal kernel size that balances local and global feature mixing.

The observed behavior aligns with the findings of MixConv [8], which concluded that an intermediate kernel size, such as $K = 9$, provides the best trade-off between capturing fine-grained spatial details and maintaining computational efficiency. In our case, $K = 10$ achieves the optimal performance, while larger kernels introduce diminishing returns and increased computational cost, as evidenced by the significant rise in training time (from 1545.17s to 3781.46s).

5. Future Work

In this study, we systematically evaluated how each of the three techniques individually impacts or enhances the performance of the ConvMixer architecture. This involved isolating each method and analyzing its contribution to improving accuracy, reducing overfitting, and enhancing computational efficiency. However, due to time constraints and the complexity of integrating multiple adjustments simultaneously, we were unable to thoroughly investigate the interplay and combined effects of these techniques.

A deeper exploration into how these methods interact—whether they complement, reinforce, or possibly conflict with each other—remains an open avenue for future work. Understanding the synergistic potential between these approaches could lead to more robust and efficient configurations, unlocking greater performance improvements for ConvMixer. Future studies will aim to address this gap, potentially leveraging larger datasets and extended computational resources to perform a more comprehensive analysis.

6. Conclusion

In this work, we explored ConvMixer's performance and extended its capabilities through systematic analysis of patch sizes, kernel sizes, and dynamic kernel adjustments. Our experiments on the Tiny-ImageNet dataset revealed key findings:

1. Smaller patch sizes improve accuracy by retaining more spatial detail but incur higher computational costs, with a patch size of 7 offering the best balance.
2. Kernel size plays a critical role in mixing spatial information, where an intermediate kernel size ($K = 10$) achieves optimal performance. Larger kernels provide diminishing returns while increasing training time.
3. Dynamic weighted kernels allow multi-scale spatial feature extraction, improving accuracy but at the expense of higher computational overhead.

Overall, our results align with existing studies. Future work will focus on optimizing dynamic kernel mechanisms to enhance ConvMixer's generalization and computational performance on larger datasets.

7. Acknowledgement

We appreciate the help from Prof. Zoran Kostic and TA William Ho for answering our question.

8. References

[1] A. Trockman and J. Z. Kolter, "Patches are all you need?," arXiv preprint arXiv:2201.09792, Jan. 2022.

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," arXiv preprint arXiv:2010.11929, Oct. 2020.

[3] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy, "MLP-Mixer: An all-MLP architecture for vision," arXiv preprint arXiv:2105.01601, May 2021.

[4] S. Chen, E. Xie, C. Ge, D. Liang, and P. Luo, "CycleMLP: A MLP-like architecture for dense prediction," arXiv preprint arXiv:2107.10224, Jul. 2021.

[5] H. Touvron, P. Bojanowski, M. Caron, M. Cord, A. El-Nouby, E. Grave, A. Joulin, G. Synnaeve, J. Verbeek, and H. Jégou, "ResMLP: Feedforward networks for image classification with data-efficient training," arXiv preprint arXiv:2105.03404, May 2021.

[6] Anonymous Reviewer, "Review of Patches Are All You Need?," OpenReview, https://openreview.net/forum?id=TVHS5Y4dNvM&noteId=ocgid221QSc.

[7] Z. Hu, "Tiny ImageNet Dataset," Hugging Face, https://huggingface.co/datasets/zh-plus/tiny-imagenet.

[8] Y. Li, N. Wang, J. Shi, X. Hou, and J. Liu, "Adaptive kernel selection in CNNs," arXiv preprint arXiv:1907.09595, Jul. 2019.

[9] C. Szegedy et al., "Going deeper with convolutions," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1-9.

[10] https://github.com/ecbme4040/e4040-2024Fall-Project-DLLM-lc3826-ym3019

9. Appendix

The Appendix should contain an abundance of detail which may not be suitable for the main text)

9.1 Individual Student Contributions in Fractions

	Lc3826	Ym3019
Last Name	CHEN	MAO
Fraction of (useful) total contribution	1/2	1/2
What I did 1	Experiment design	ConvMix Kernal experiment run
What I did 2	Paper writing (all sections apart from intro)	ConxVix converting torch to tensorflow framework
What I did 3	ConvMixer Patches Experiment run	Paper intro part writing

9.2 Pseudocode – AdamW Implementation

Code:https://github.com/ecbme4040/e4040-2024Fall-Project-DLLM-lc3826-ym3019/blob/main/Conv_AdamW.ipynb

```
DEF Residual(tf.keras.layers.Layer):
   DEF __init__(self, fn):
      CALL super(Residual, self).__init__()
      SET self.fn TO fn

   DEF call(self, x):
      RETURN self.fn(x) + x

DEF ConvMixer(dim, depth, kernel_size=9, patch_size=7, n_classes=1000):
   SET layers_list TO []

   APPEND layers.Conv2D(filters=dim, kernel_size=patch_size, strides=patch_size, padding="valid") TO layers_list
   APPEND layers.Activation("gelu") TO layers_list
   APPEND layers.BatchNormalization() TO layers_list

   FOR _ IN RANGE(depth):
      APPEND Sequential([
         Residual(Sequential([
            layers.DepthwiseConv2D(kernel_size=kernel_size, padding="same"),
            layers.Activation("gelu"),
            layers.BatchNormalization()
         ])),
         layers.Conv2D(filters=dim, kernel_size=1, padding="valid"),
         layers.Activation("gelu"),
         layers.BatchNormalization()
      ]) TO layers_list

   APPEND layers.GlobalAveragePooling2D() TO layers_list
   APPEND layers.Flatten() TO layers_list
   APPEND layers.Dense(units=n_classes, activation="softmax") TO layers_list
   RETURN Sequential(layers_list)
```

9.3 Pseudocode – Dynamic Weighted Kernel Implementation

```
DEF DynamicDepthwiseConv2D(tf.keras.layers.Layer):
   DEF __init__(self, kernel_sizes, **kwargs):
      CALL super(DynamicDepthwiseConv2D, self).__init__(**kwargs)
      SET self.kernel_sizes TO kernel_sizes
      SET self.depthwise_convs TO []
      SET self.alpha TO None

   DEF build(self, input_shape):
      FOR kernel_size IN self.kernel_sizes:
         APPEND layers.DepthwiseConv2D(kernel_size=kernel_size, padding='same') TO self.depthwise_convs
      SET self.alpha TO self.add_weight(
         shape=(LEN(self.kernel_sizes),),
         initializer='ones',
         trainable=True,
         name='alpha'
      )

   DEF call(self, input_tensor):
      SET outputs TO []
      FOR conv IN self.depthwise_convs:
         APPEND conv(input_tensor) TO outputs

      SET alphas TO tf.nn.softmax(self.alpha)
      SET combined TO tf.zeros_like(outputs[0])
      FOR i IN RANGE(LEN(outputs)):
         SET combined TO combined + alphas[i] * outputs[i]

      RETURN combined
```