Comparative Analysis of Deep Learning Architectures for Scene Classification: From Scratch vs. Pre-trained Models

Atul Kumar
110143931
atulkum@uwindsor.ca

Nehme Haidura
110203614
haidura@uwindsor.ca

ABSTRACT

This study investigates the performance of various deep learning architectures on the Intel Image Classification Dataset, a multi-class dataset of natural and urban scene images. Our objective is to identify the most effective model for scene recognition by comparing six different approaches: a baseline StandardCNN, an AutoEncoder with a classification head, a CNN with integrated attention mechanisms, a Residual Network ResNet, Feature Pyramid Networks FPN, and several pre-trained models including CLIP and ResNet variants. We perform extensive hyperparameter tuning and evaluate each model using metrics such as accuracy, precision, recall, and F1-score. Our findings show that pretrained models outperform models trained from scratch, with CLIP-L14 achieving the highest validation accuracy of 96% in frozen configuration, followed by CLIP-B32 at 95%. Among models trained from scratch, attention mechanisms, particularly BAM achieved the best performance at 91.5%. These results underscore the importance of architectural choices and transfer learning in achieving state-of-the-art performance in image classification tasks.

1 INTRODUCTION

Image classification is a fundamental challenge in computer vision, with applications ranging from autonomous navigation to environmental monitoring. With the rise of deep learning, convolutional neural networks (CNNs) have become the standard for such tasks, continually evolving in complexity and effectiveness. This project explores the performance of several CNN-based models and modern architectural enhancements using the Intel Image Classification Dataset a well-labeled dataset containing 25,000 images across six distinct scene categories: buildings, forest, glacier, mountain, sea, and street. Our goal is to systematically compare traditional CNN architectures with more advanced alternatives, such as attention mechanisms, residual learning, and feature pyramids, as well as powerful pre-trained models adapted through transfer learning. We evaluate models both trained from scratch and adapted from state-of-the-art vision backbones. Key aspects of our methodology include hyperparameter tuning, structured experimentation across multiple configurations (e.g., dropout, pooling, batch normalization), and standardized evaluation metrics. By analyzing the strengths and weaknesses of each approach, we aim to uncover the most effective strategies for real-world scene classification tasks.

Conference'17, July 2017, Washington, DC, USA
© 2025 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

Wise-SrNet [11], an architecture that maintains spatial resolution without increasing computational cost, has been proposed that tackle the limitations of global average pooling and showed up to 26% accuracy improvement on high-resolution inputs.

Another work introduced Vision Eagle Attention [3], a spatial attention mechanism integrated into ResNet-18, which selectively emphasizes informative regions in images. This led to improved classification benchmarks on FashionMNIST and the Intel Dataset.

A third work [7] presented a comparative analysis of deep learning models on the Intel Image Classification dataset, highlighting the influence of network design, data augmentation, and transfer learning strategies, ultimately achieving near-perfect accuracy.

3 PROBLEM FORMULATION AND RESEARCH QUESTIONS

3.1 Dataset Description:

The dataset used in this project is the Intel Image Classification Dataset [1], provided by Kaggle. The data consists of real-world images collected from various natural and urban scenes, intended for training and evaluating image classification models. The dataset is composed of approximately 25,000 images, divided into six distinct classes: buildings, forest, glacier, mountain, sea, and street. Each class represents a unique scene category, making this dataset suitable for multi-class image classification tasks. The images are in RGB format, already uniformly sized to 150×150 pixels, and come pre-sorted into **training set** (14000 images), **test set** (3000 images), and **prediction set** (7000 images).

The "**prediction set**" provided in the competition is unlabeled and meant only for the submission to Kaggle's leaderboard. That's why we excluded this set from our experiments due to the lack of a ground truth label. To sum up, we utilize only the training and test sets, which are 14000 and 3000 in quantity, respectively. To fine-tune the hyperparameters, we reserve 20% of the training data as the validation set, i.e. around 2800 images. The test set is used for the final evaluation.

3.2 Problem Definition

This is the multi-class image classification problem. Given an input image, the task is to correctly predict which of the six classes mentioned, which this image belongs to.

The core challenges for this problem are mentioned as follows:

(1) **Inter-class Similarity**: One of the core challenges is to deal with the images that share overlapping visual textures, leading to wrong classification results. For example, we found some of the images of glaciers and mountains look very similar to each other.
(2) **Multi-class Visual Elements:** This was another core challenge that probably confused the model as well. We found some of the images containing elements from multiple classes, such as images containing both buildings and streets.

3.3 Research Questions:

(1) How do different hyperparameter configurations (such as dropout, optimizer, etc.) affect the generalization ability of models?
(2) How do advanced architectural components such as attention blocks, residual blocks, and multi-scale features (FPN) impact the model's performance, compared to models that are stacked up with only convolution and linear layers?
(3) How much performance advantage can pretrained models achieve compared to the models trained from scratch?
(4) Can attention-based models (pretrained or from-scratch) improve classification accuracy by focusing only on the most relevant features, despite the presence of inter-class similarity and multi-class visual elements within a single image?

4 METHODOLOGY

4.1 Reprocessing

The preprocessing pipeline was designed to ensure consistency across models while accommodating architectural requirements. For ResNet models, we used 256×256 input resolution to align it with the standard configuration of their convolutional layers. CLIP models we used built-in preprocessing function, which normalizes images and resizes them to 224×224. All other models we used the dataset's native 150×150 resolution. To evaluate the effect of data augmentation, we applied a dedicated pipeline to ResNet models during training. This included random horizontal and vertical flips, rotations, color jittering, affine transformations, and perspective distortions. These augmentations were applied randomly to increase training data diversity and model robustness.

4.2 Model Selection

To identify the best approach for our dataset, six different models were selected based on their effectiveness in image classification tasks. We trained the first five mentioned models from scratch, and the sixth category contains pre-trained models. The models are as follows:

(1) **StandardCNN:**
Or SimpleCNN is a basic neural network model. This model serves as a strong baseline due to its simplicity, interpretability, and solid performance on general image classification tasks.
(2) **AutoEncoder With Classification Head:**
While this model isn't usually used for image classification, it combines feature extraction through an autoencoder with a classification layer, making it useful for learning compact feature representations.
(3) **CNN with Attention:**
Integrates attention layers to focus on the most relevant parts of the image, improving classification accuracy. It help improve the model's interpretability and performance by dynamically weighting feature maps, which can be particularly effective in complex visual datasets.
(4) **Residual Learning with Skip Connections:**
is a technique used in DNNs where the input to a layer is added directly to its output. This allows the network to learn residuals (the difference between input and output), helping to train deeper models more effectively by minimizing problems like vanishing gradients.
(5) **Feature Pyramid Network:**
FPN's uses multi-scale feature maps to enhance detection and classification of objects at different sizes which helps improve recognition performance across.
(6) **Pre-trained Models:** These models come with knowledge already learned from large-scale datasets and state-of-the-art architectures. These models allow efficient transfer learning and rapid adaptation to new tasks with limited labeled data

In total, all mentioned deep learning models were tested on the dataset to assess their classification performance, enabling a comprehensive comparison of different architectural strategies and learning techniques.

4.3 Hyperparameter Tuning

We applied hyperparameter optimization to find the best set of hyperparameters for each DL model based on each model's structure and behavior.

For the StandardCNN, we used three Optimizers SGD, Adam, and AdamW, controlled the learning rate, and applied Dropout rate after convolutional layers to reduce overfitting. To stabilize training, we enabled and disabled Batch normalization. Also, we applied Pooling method for comparison purposes and layers configuration.

For AutoEncoder with Classification Head we tuned the learning rate, dropout rate, and tested only two optimizers Adam and AdamW. Batch normalization was enabled and pooling methods were applied.

Hyperparameter tuning for CNN with Attention was fixing the learning rate at 0.001, the we used AdamW as an optimizer, and experimented with different layers configurations. We applied different dropout rates enabled batch normalization, and tested various pooling strategies.

For Residual Learning with Skip Connections, we experimented with multiple ResNet variants. The learning rate was fixed at 0.1, and the SGD optimizer was used across all runs. Batch normalization was enabled, and dropout was not applied.

Feature Pyramid Networks (FPN) hyperparameter tuning was in we adopted the best-performing hyperparameters from each backbone as follows: In StandardCNN we used Adam optimizer, a learning rate of 0.001, average pooling, batch normalization enabled, and dropout set to 0.0. For AutoEncoder with Classification Head we also used Adam optimizer with a learning rate of 0.001, batch normalization enabled, and applied a dropout rate of 0.1. Moving to

CNN with Attention we used AdamW optimizer, a fixed learning rate of 0.001, batch normalization enabled, dropout set to 0.0, and max pooling also and employed the Bottleneck Attention Module (BAM) with a 7-layer configuration. Finally, ResNet18 we used SGD optimizer with a learning rate of 0.1, batch normalization enabled, and dropout set to 0.

In Pre-trained Models All models were fine-tuned using the Adam optimizer, with a fixed learning rate of 0.001 and a weight decay of 0.0001

4.4 Training and Evaluation

To train and evaluate our models, the dataset was split into an 80:20 ratio. Specifically, 80% of the data was used for training and validation while an independent test set was held out and only used for final model evaluation. All models were trained on the same dataset splits and had identical preprocessing to ensure fair and consistent comparisons.

Hyperparameter tuning was performed with different hyperparameters, as mentioned in the previous section. Each model was trained for up to 50 epochs with early stopping enabled, stopping training if validation accuracy did not improve for 5 consecutive epochs. Model selection was based on the highest validation accuracy observed during training.

Final test set performance was evaluated using accuracy, precision, recall, and F1-score, providing a comprehensive view of each model's effectiveness across all classes. For clarity, the best validation accuracy for each run is marked with a large cyan dot in all relevant plots.

Note: The highest validation accuracies for each model are marked with a large cyan-colored dot in all plots for easy identification.

5 RESULTS AND DISCUSSIONS

5.1 StandardCNN:

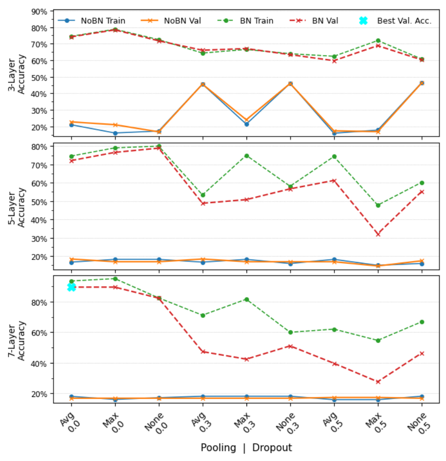

Figure 1: StandardCNN (optimizer=SGD)

In training CNNs with SGD, the optimizer relies heavily on stable and well-scaled gradients to update the parameters effectively. However, without Batch Normalization (BN), the activations are not normalized across layers, leading to significant shifts that result in unstable flow of gradients, increasing the likelihood of vanishing and exploding gradients. This is why we see a stagnant accuracy in Figure 1 with almost all the hyperparameter configurations across all three architectures, when no batch normalization is applied. Since, as we go deeper, there is often a problem of vanishing or exploding gradients (which gets worsened when the activations are not normalized). That's why, even without BN, we still see some spikes in the 3-layer model achieving close to 50% accuracy.

Since BN normalizes the activations across mini-batches, which reduces internal covariate shift and makes optimization smoother and deep networks tend to be much more expressive. The results after applying BN show a dramatic improvement in accuracy, as shown in the same Figure 1, with a model going from a low of 14% without BN to over 89% with BN. Another important aspect to note is the relationship between the network depth and model performance. When we increased the depth of the network from 3-layer to 5-layer, and then further to 7-layer, the performance kept increasing, likely because the model no longer faced the issues of vanishing or exploding gradients. Although the 3-layer and 5-layer models showed little difference, the 7-layer model improved by 10% compared to the 3-layer and 5-layer counterparts, illustrating its greater capacity to capture more meaningful features.

In terms of pooling, with a 3-layer network, the network has less capacity to extract complex features, so max pooling works the best, as it only selects the dominant activations, ensuring better generalization. However, as we increase the depth of the network, the models also get better at extracting the meaningful features. That's why, with a 5-layer model, we see a better performance when no pooling is applied, as the model retains more detailed information at each layer. Although max pooling still outperforms the average because stronger activations still help the model more than smoothing the features too much, with a little increase in depth. Lastly, with the 7-layer model, the average pooling outperforms when there is no dropout. It's probably because models with additional layers make better use of stable and generalized feature representations. Although when we introduce dropout, models with no pooling performed the best, as keeping all features helps when a lot of neurons are dropped.

Lastly, to see the difference at a large scale, we intentionally applied the dropout after every layer. With a 3-layer, the impact is minimal across all dropouts. However, as we increased the depth to 5 and 7 layers and introduced the dropout, it hurt the overall performance (as depicted, since we introduced too much dropout). Interestingly, average pooling with dropout 0.5 outperformed 0.3 with a 5-layer model. In contrast, as we went deeper with 7-layer and introduced more and more dropout, the performance got worse and worse, illustrating that deactivating too many neurons during training can hurt the performance, no matter how deep the network is.

In training CNNs with Adam optimizer, the learning dynamics fundamentally differ from SGD due to Adam's adaptive learning rate mechanism. As shown in Figure 2, without Batch Normalization (BN), Adam achieves remarkably higher baseline accuracies compared to SGD across all network depths. This indicates that Adam can handle gradient scaling issues that would otherwise lead to vanishing or exploding gradients.

When Batch Normalization is applied with Adam, we can notice a significant improvement. The 3-layer model with BN achieves

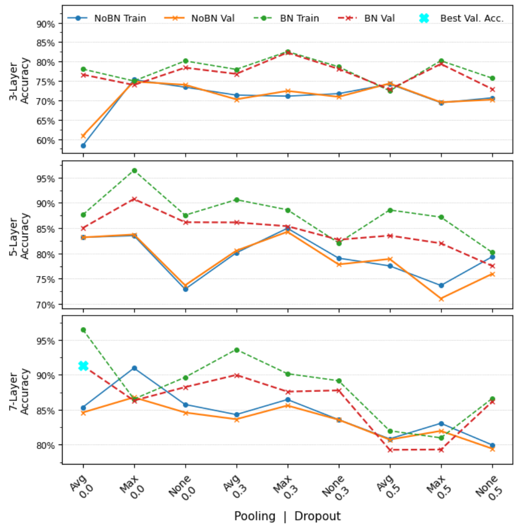

Figure 2: StandardCNN (optimizer=Adam)

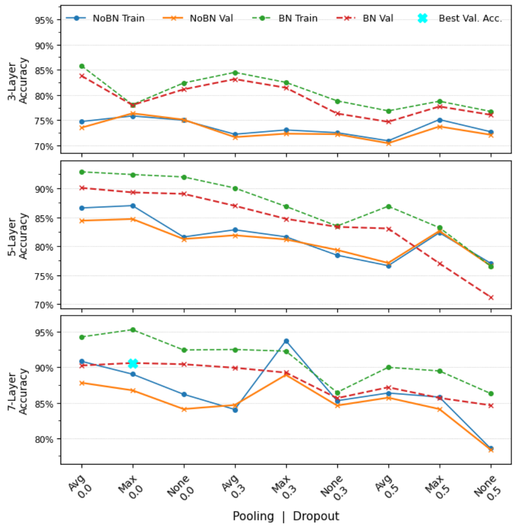

Figure 3: StandardCNN (optimizer=AdamW)

82% accuracy with Adam, showing a 4% improvement over SGD's maximum of approximately 78%. This demonstrates that even in shallow networks, Adam provides optimization benefits beyond what SGD can achieve. Moreover, BN still provides value by reducing internal covariate shift and enabling more stable feature distributions, particularly beneficial in deeper architectures. For instance, while SGD with BN achieved a maximum validation accuracy of around 89% in the 7-layer model, Adam with BN pushes this further to over 91% validation accuracy, a 2% improvement that demonstrates Adam's superior optimization capabilities.

Regarding pooling strategies with Adam, the patterns differ notably from SGD. In the 3-layer network, all pooling methods perform comparably well, indicating that Adam's optimization is less sensitive to architectural choices in shallow networks. As depth increases to 5 and 7 layers, the interaction between pooling and dropout becomes more complex. In 5-layer model, dropout 0.5 causes a slight drop, while in 7-layer model we can see a higher drop. This suggests that preserving all spatial information helps when significant regularization is applied. Without dropout, max pooling outperforms average pooling in 5-layer model, but when a deeper model is applied as in 7-layer model, it's quite the opposite, average pooling outperforms max pooling. Possibly because Adam's stable optimization allows the model to benefit from the smoothing effect of average pooling without losing important gradient signals in deeper models. In training CNNs with AdamW optimizer, we introduce a crucial modification to the standard Adam algorithm through decoupled weight decay. Unlike Adam, which applies weight decay through the gradient, AdamW decouples weight decay from the gradient-based update, applying it directly to the weights.

As shown in Figure 3, without Batch Normalization, AdamW demonstrates remarkably stable performance across all network depths, maintaining 72-76% validation accuracy in the 3-layer model. The performance is slightly better than Adam and outperforms SGD's. This suggests that AdamW's special way of applying weight decay acts like a built-in stabilizer, helping the network train properly even when the activations aren't normalized.

When Batch Normalization is applied, AdamW reveals unique characteristics. In the 3-layer model, AdamW with BN achieves 74-83% validation accuracy, slightly higher than Adam by 1% and SGD by 5%. We can notice that in 5-layer model, the accuracy didn't improve from Adam, which is 90% and It is worth noting that in 7-layer model, AdamW achieves validation accuracies reaching 90%, just 1% below Adam's peak performance, but with notably more stable training curves and better generalization metrics.

Regarding pooling strategies, AdamW shows distinct preferences compared to both SGD and Adam. In shallow networks, max pooling outperforms other strategies. As we move to deeper architectures, an interesting pattern is noticeable, without dropout, AdamW strongly favors Max pooling in the 7-layer model, reaching 90% validation accuracy, unlike Adam and SGD. However, AdamW shows greater resilience to the choice of pooling strategy, noticed by fewer swings compared to the other two methods.

A particularly noteworthy pattern is the performance trajectory across depths. While SGD showed dramatic improvements with depth when BN was applied and Adam showed consistent improvements, AdamW shows a more subtle pattern. Suggesting that AdamW's regularization may limit the benefits of very deep architectures on this particular task.

After comparing the three optimizers we can say that *Adam* is the best overall optimizer for this StandardCNN architecture due to its excellent balance of performance, stability, and robustness across all tested conditions. It provides the most reliable training experience whether or not batch normalization is available.

5.2 AutoEncoder With Classification Head:

In evaluating the AutoEncoder with Classification Head approach [14], we explore a two-stage training process that combines unsupervised feature learning with supervised classification. This method first trains an autoencoder to learn compressed representations of the input data, these learned features are then frozen, and only a classification head is trained on top. Given the computational

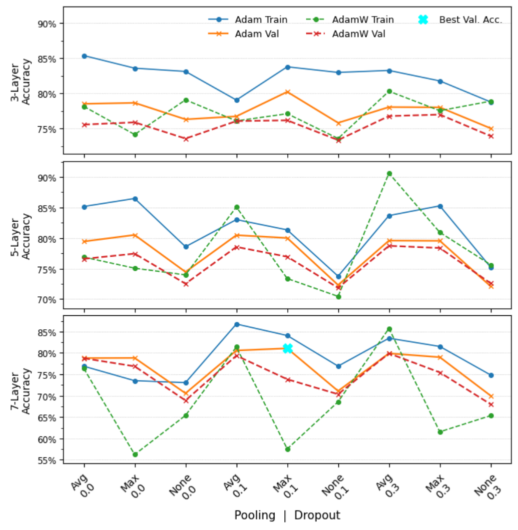

Figure 4: AutoEncoder With Classification Head (optimizers =Adam & AdamW)

expense of this sequential training setup and the consistently superior performance of Adam-based optimizers combined with Batch Normalization in our prior experiments, we excluded SGD and no-BN configurations from this analysis. Instead, we focused on comparing the performance of Adam and AdamW optimizers, both with Batch Normalization applied.

As shown in Figure 4, Adam with the autoencoder approach shows notably different characteristics compared to standard training. In the 3-layer configuration, the best validation accuracy Adam achieved was 80% with max pooling and dropout 0.1, which is lower than the 82% achieved with standard CNN training. This performance gap suggests that the fixed feature representations learned by the autoencoder, while capturing general data structures, may not be optimally tuned for the specific classification task. In the 5-layer architecture, we observe that the highest validation accuracy Adam achieved was 80%, the same as in the 3-layer model. However, Adam exhibits extreme volatility across different pooling and dropout configurations, with validation accuracy swinging from a low of 72% with no pooling and dropout 0.1 to peaks of 80% with max pooling and dropout 0. This instability suggests that the intermediate representations learned by deeper autoencoders may be more sensitive to architectural choices, possibly because the unsupervised pretraining creates features that interact unpredictably with different pooling strategies. The 7-layer results are even more concerning. Adam optimizer was able to increase the performance slightly to 81% with max pooling and dropout 0.1, which is marginally higher than the previous layers' best performance but still significantly lower than what Adam achieved with standardCNN (91% in the 7-layer architecture). Moreover, we still observe dramatic swings across different pooling and dropout setups, from a low of 69% to 81%. This indicates that additional depth provides no benefit in the autoencoder setting and that deep autoencoder features may be overly specialized for reconstruction tasks, becoming problematic when combined with aggressive regularization.

Moving to AdamW with the autoencoder approach, in the 3-layer configuration, AdamW achieves its best validation accuracy of 76% with average pooling and dropout 0.3, which is 4% lower than Adam's best of 80% and 7% lower than AdamW's performance with standardCNN of 83%. This underperformance suggests that AdamW's decoupled weight decay may over-regularize when applied to a limited parameter space.

In the 5-layer architecture, AdamW demonstrates slightly more stable performance, though it still exhibits significant variation. The decoupled weight decay appears to provide some stabilization, particularly evident in configurations with heavy dropout where AdamW achieved 78% with average pooling and dropout 0.3, which is 2% higher than the 3-layer model. This modest improvement with depth contrasts sharply with standard training, where deeper networks showed substantial gains. However, both optimizers struggle to match StandardCNN performance where AdamW achieved 90%, highlighting a fundamental limitation of the two-stage training approach.

The 7-layer results reveal AdamW's limitations with very deep autoencoders. Performance becomes highly unstable with validation accuracies ranging from 67% to 79%. The dramatic drops observed with certain pooling configurations suggest that the deep autoencoder features may be overly specialized to reconstruction rather than classification. This specialization becomes problematic when combined with aggressive pooling that discards spatial information the autoencoder deemed important for reconstruction.

Comparing Adam and AdamW directly, we observe that Adam can achieve higher accuracies in optimal configurations, reaching up to 81% in the 7-layer model, which was the best validation accuracy across all the layers and both optimizers.

The overall pattern suggests that the autoencoder approach fundamentally conflicts with modern optimization techniques. The frozen feature space prevents both Adam and AdamW from leveraging their adaptive capabilities fully, resulting in a scenario where neither optimizer can overcome the inherent limitations of features optimized for reconstruction rather than classification. The consistent performance gap compared to standardCNN, combined with doubled computational costs and increased instability, makes a compelling case against using autoencoder-based feature extraction for this classification task.

5.3 CNN with Attention:

In exploring attention mechanisms for CNNs, we investigate five different attention modules: CBAM (Convolutional Block Attention Module), BAM (Bottleneck Attention Module), Squeeze and Excitation (SE), Double Attention, and Triplet Attention. These mechanisms aim to enhance the network's ability to focus on relevant features by introducing learnable weights that emphasize important channels and/or spatial locations. All experiments use AdamW optimizer with Batch Normalization, building upon our best-performing configuration from previous experiments.

As shown in Figure 5, BAM demonstrates the strongest performance among all attention mechanisms, achieving a peak validation accuracy of 91.5% with max pooling and dropout 0.0, the highest accuracy across all our experiments. This represents a 0.5% improvement over standard CNN with AdamW (91%). This could be because BAM decomposes the process of inferring a 3D attention map into two parallel streams (spatial and channel branches) that

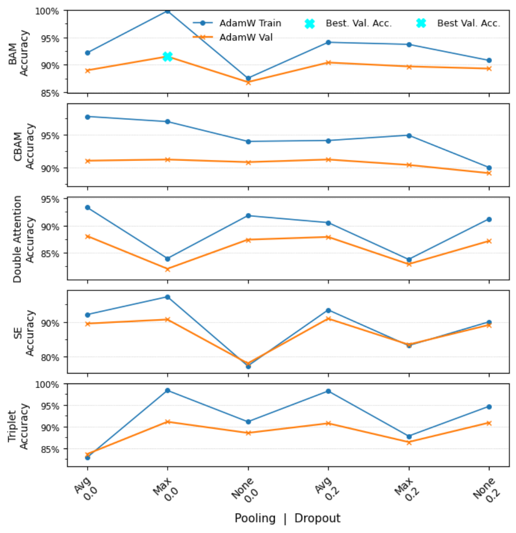

Figure 5: CNN with Attention

explicitly learn "what" and "where" to focus on, significantly reducing computational overhead [6]. This design proves highly effective, maintaining remarkably stable performance with validation accuracies.

CBAM, which applies channel-wise and spatial attention sequentially, shows remarkably stable performance across all configurations as shown in Figure 5, with validation accuracies tightly clustered between 89–91%. Unlike BAM's peak performance, CBAM maintains consistent results regardless of pooling or dropout choices. This stability suggests that BAM's bottleneck design with parallel channel and spatial attention branches creates a robust feature refinement mechanism that is less sensitive to architectural variations. In CBAM, the channel-wise attention gate focuses on the most important channels based on their spatial relationships, while the spatial attention gate identifies important regions based on channel relationships [13]. CBAM achieves its best performance of 91% with max pooling and dropout 0.0, nearly matching BAM's peak. The sequential nature of CBAM's dual attention provides effective feature refinement, though with slightly more sensitivity to architectural choices compared to BAM's parallel design.

Double Attention as shown in Figure 5 exhibits high variability with validation accuracies between 82% and 88%. Its two-step mechanism first gathers features from the entire space through second-order attention pooling, then adaptively distributes features to each location via another attention mechanism [16]. This complex design shows extreme sensitivity to architectural choices, with a dramatic drop to 82% under certain configurations (max pooling with dropout 0.0). When properly configured, it can achieve 88% accuracy, but the inconsistency makes it less practical than more stable alternatives.

SE attention shows the unstable performance among different pooling or dropout setups, with validation accuracies ranging from 78% to 90% as noticed in Figure 5. The SE block's squeeze layer reduces spatial dimensions by averaging each channel, followed by an excitation layer that applies learnable gating to select informative channels [5]. The best configuration achieves only 90%, which is close to the other mechanisms, but the instability suggests that spatial information is crucial for this dataset, and pure channel attention provides limited benefit.

Triplet Attention performance ranges from 83% to 91% validation accuracy as shown in Figure 5. By building inter-dimensional dependencies through rotation operations and capturing cross-dimension interactions with a three-branch structure, Triplet Attention encodes both inter-channel and spatial information with minimal computational overhead [2]. It performs particularly well with max pooling configurations, achieving 91% accuracy, though it shows some sensitivity to dropout strategies, with performance dropping when we increase the dropout to 0.2.

The results reveal important insights about attention mechanism design. BAM's parallel processing of spatial and channel attention achieves the highest accuracy of 91.5%, because of its computationally efficient parallel design. More complex mechanisms like Double Attention don't necessarily perform better, with its two-stage gathering and distribution process creating instability. Regarding architectural interactions, most attention mechanisms prefer max pooling when no dropout is applied, except for the Double attention mechanism, where a dramatic drop happened. The marginal improvements over standardCNN (0.5% for BAM) suggest that for this task, modern optimizers and normalization techniques already extract most learnable patterns.

5.4 Residual Learning with Skip Connections (ResNet):

In exploring Residual Learning with Skip Connections, we implement ResNet architectures ranging from ResNet-18 to ResNet-152 to evaluate their performance on our image classification task. ResNet's fundamental innovation lies in its skip connections, enabling the training of deeper networks without degradation [1]. For this experiment, we test two training approaches: with and without data augmentation. The data augmentation pipeline includes several transformations such as random horizontal/vertical flips, rotation (15 degrees), color jittering, affine transformations, and perspective changes, designed to increase the diversity of training samples. Following the original ResNet paper's recommendations, we use SGD optimizer with learning rate 0.1, batch normalization, and no dropout, as ResNet's architecture already incorporates sufficient regularization through its design.

(1) **With Data Augmentation:**

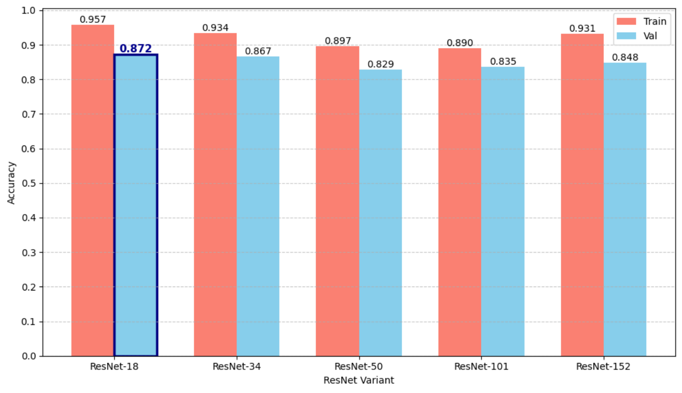

Figure 6: ResNet Performance with Data Augmentation

With data augmentation, ResNet demonstrates interesting behavior across different depths. As shown in Figure 6, ResNet-18 achieves the highest validation accuracy at 87%, followed by ResNet-34 at 86%. However, as we increase depth, performance unexpectedly decreases: ResNet-50 drops to 82%, ResNet-101 slightly recovers to 83%, and ResNet-152 reaches 84%. This inverse relationship between depth and performance suggests that data augmentation helps generalization, but deeper models struggle to utilize their increased capacity effectively, and maybe for this particular dataset, the additional capacity of deeper networks may lead to overfitting despite data augmentation.

(2) **With No Data Augmentation:**

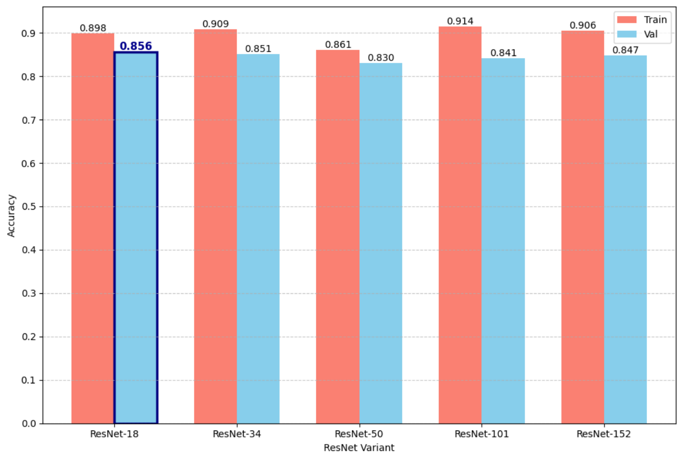

Figure 7: ResNet Performance with No Data Augmentation

Without data augmentation, ResNet shows similar patterns but with generally lower performance. As shown in Figure 7, ResNet-18 and ResNet-34 achieve 85% validation accuracy, while ResNet-50 maintains 82%, and it's interesting that it matches the score for with the data augmentation method. ResNet-101 and ResNet-152 achieve 84% and 84%, indicating the same, as the previous method that increased depth doesn't translate to better performance on this dataset. The consistency of ResNet-50's performance across both settings 82% is noteworthy, suggesting this architecture may have hit a complexity limit for this dataset regardless of data augmentation.

After comparing both approaches, we can notice that data augmentation provides modest but consistent improvements, particularly for shallower networks. ResNet-18 benefits most with a 2% improvement, while ResNet-34 gains 1%. Deeper networks show minimal or no gains. For instance, ResNet-50 showed no improvements at all, suggesting that as ResNet models become deeper, data augmentation provides smaller and smaller improvements. The best overall performance comes from ResNet-18 with data augmentation at 87%, which falls behind our best previous results with attention mechanisms (91.5% with BAM). This indicates that for this dataset, moderate complexity architectures with modern optimization techniques outperform the deeper, more complex ResNet architectures.

5.5 Feature Pyramid Network

To evaluate the effectiveness of Feature Pyramid Network (FPN), we extended the discussed models by adding the FPN modules, inspired by [12], in their respective classification heads. Although FPNs are usually used for detection and segmentation tasks, where we are more concerned about multi-scale feature representations. However, we aimed to leverage these features and observe the difference in performance

As we have performed extensive hyperparameter tuning for models such as StandardCNN, AutoEncoder With Classification Head, AttentionCNN, and ResNet, we utilized the best hyperparameter configurations found for these baseline models and tested their FPN-enhanced counterparts. This allows us to see the potential effects of FPNs and fairly compare them with their regular counterparts.

To get the features at multiple scales, we followed the classical design for each architecture. It consists of a bottom-up pathway where, it first extract the feature maps from different depths, then pass through 1x1 conv to reduce dimension, then iteratively upsamples and finally passes through 3x3 conv to smooth the feature map.

Fundamentally, we followed the same steps for each of the models and leveraged these fused feature maps to evaluate the effectiveness. However, as shown in 8, the results were quite contrary to our expectations. For all the FPN-enhanced versions, the performance was significantly lower than their regular counterparts, except for ResNet.

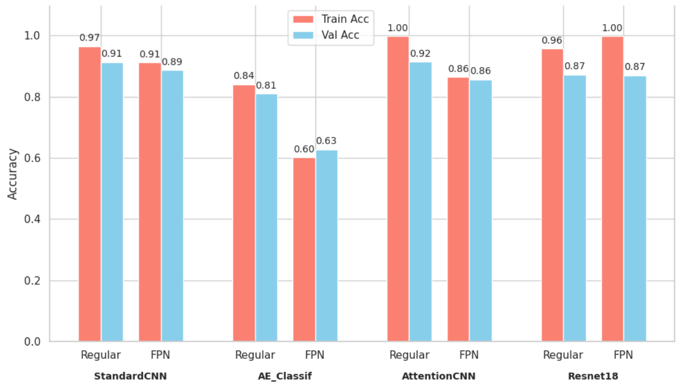

Figure 8: Feature Pyramid Network (FPN) Performance (Regular vs FPN architecture)

In the case of StandardCNN, regular architecture outperformed the FPN-enhanced variant by 2.5%. This is likely because the sequential feature extraction in the regular CNN allows the deeper layers to focus on more abstract and class-specific information. When we introduced FPN, it brought features from both early and later layers, which may again reintroduce less relevant details that the model has already learned to discard. It could interfere with the model's ability to focus on the most important features, and hence reduce the clarity of the final representation.

The most significant drop in performance was observed when we applied FPN to the autoencoder with a classification head. To keep the encoder-decoder pipeline the same as regular architecture, we only introduce FPN to the encoder part, where the decoder part still uses the same latent features as the original setup. We provide these FPN features to only the classification head. However, this design has unintentionally hurt the classification performance. By introducing FPN to autoencoders, we probably again introduce a mixture of fine-grained and high-level information that has no clean separability between the features of different classes and degrade the overall accuracy.

For the AttentionCNN, we anticipated a potential drop in performance with FPN. We have utilized the Bottleneck Attention Module (BAM), as it has outperformed others in the previous section, which improves feature selection by highlighting channel and spatial-wise patterns. And as we go deeper, the attention models prioritize certain features over others. When we add FPN, it merges these high-quality features picked up from the attention module with more general and noisy features from earlier layers. As a result of this blending, the model tends to lose the advantage of the attention module and weakens its ability to emphasize the most relevant part of the input.

Interestingly, we witnessed a marginal change in the performance of ResNet-18, even after introducing FPN. It's probably because ResNet-based models naturally combine features across layers through skip connections. This means, these shortcut connections, by default, allow the flow of information from earlier to later layers, which helps preserve both fine and abstract details of input. Hence, even though FPN didn't bring any advantage to the ResNet-based model, it didn't even harm the performance.

5.6 Pretrained Models

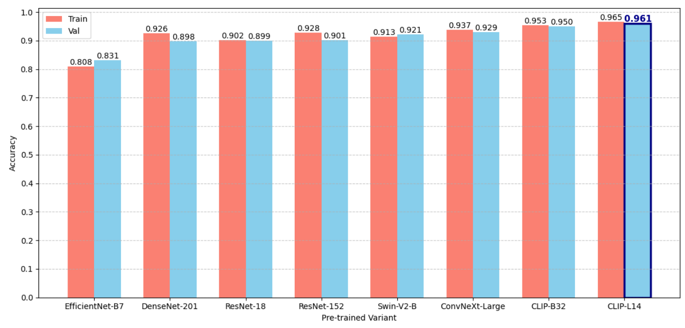

Figure 9: Frozen Pre-trained Models Performance

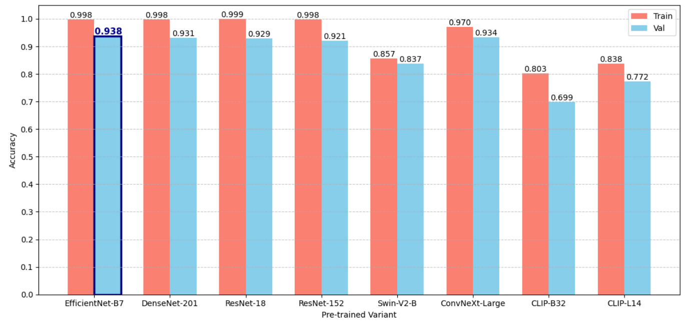

Figure 10: Trainable Pre-trained (Unfrozen) Models Performance

As shown in Figures 9 and 10, we leveraged a diverse set of state-of-the-art pretrained models spanning convolutional, such as ResNet, DenseNet, etc., and transformer-based architectures, such as CLIP, SWin, etc., to assess their adaptability to our downstream task through transfer learning. Since the hyperparameter tuning for such models is very costly, that's why, we adhere to the commonly used default values for the learning rate (0.001), optimizer (Adam) and weight decay (0.0001), which are often sufficient for achieving a good convergence in fine-tuning tasks.

We implemented two main variants for each model, i.e., frozen and trainable (unfrozen). Under frozen, all the layers of the pre-trained model were kept constant, and only an appended linear classification layer was trained. This method assesses how well the pretrained features generalize to our dataset. While trainable involves not only the appended classification layer, but also fine-tunes the entire pretrained backbone.

EfficientNet-B7, the deepest EfficientNet variant pretrained on the ImageNet, showed a great improvement when moving from the frozen setup to the unfrozen setup, gaining a total of over 10% increase in the validation accuracy. This method essentially scales all dimensions of depth, width, and resolution [15], allowing it to capture complex patterns and fine-grained features in the dataset. As it's the deepest variant (with 813 layers) and outperformed its other variants on the ImageNet dataset, we planned to test it on our dataset, despite being a computationally expensive option. However, the performance gains in our task justified the additional training cost. When frozen, only the appended classification layer adapts to the dataset, and the validation accuracy (83.1%) shows that the features, originally trained on ImageNet, didn't align fully with the diverse natural scenes in our dataset. However, unfreezing allowed the whole network to update its feature representation based on our domain, which significantly boosted the accuracy to almost 94%. This big boost is probably because EfficientNet balances depth, width, and resolution uniformly, which gives this model enough capacity to effectively learn complex features, retain diverse features, and capture fine details even in a new domain.

DenseNet-201, the second deepest architecture of the DenseNet family [4], performed even better under frozen configuration, achieving a validation accuracy of 89.8%. This performance boost, compared to the EfficientNet counterpart, is probably because of DenseNet's unique connectivity pattern. Each layer in dense blocks receives input from all preceding layers, which facilitates feature reuse and efficient flow of gradients across the network, and also helps avoid the vanishing gradient problem. Due to this connectivity pattern, the network learn rich and generalizable features that can be transferred across other domains. This is why we gain almost 90% accuracy on the validation set even under frozen settings. However, when we unfroze the backbone and trained the whole architecture, the model's performance increased to over 93%, which shows this model's ability to adapt the already robust feature representations to a more dataset-specific one, helping to gain additional performance. However, the performance gain was not as big as EfficientNet, suggesting pretrained features can generalize very well across different domains.

ResNet-18, a convolutional neural network with residual learning, utilizes skip connections that enable stable gradient flow through its layers. This architectural design makes ResNet particularly robust for transfer learning applications. In the frozen configuration, ResNet-18 achieved 89.9% validation accuracy as shown in Figure 9, showing solid baseline performance with just the classification head trained. In the trainable (unfrozen) configuration, as shown in Figure 10 the validation accuracy improved to 92.9% when fully fine-tuned. This stable improvement demonstrates that residual connections effectively preserve pretrained features while allowing task-specific adaptation. The 3% gain without the performance drops highlights the architectural advantages of residual learning for transfer learning on smaller datasets. With regards to ResNet-18 implemented from scratch in 5.4, the frozen and trainable variants of pretrained outperformed by almost 3% and 6%, respectively. This difference in performance basically shows how exposure to a large image set like ImageNet can help a model's generalization and transfer learning capabilities, and outperform its from-scratch counterparts by such a margin.

ResNet-152, with its deeper residual architecture, follows a similar pattern but with an interesting slight difference. The frozen model achieved 90.1% validation accuracy, as we can see in Figures 9, marginally outperforming ResNet-18 due to richer feature representations from additional layers. When fully fine-tuned, we notice in Figure 10 that ResNet-152 reached 92.1% validation accuracy, slightly underperforming ResNet-18 despite having significantly more parameters. The performance between shallow and deep ResNet variants suggests that for our dataset size, the depth of ResNet-18 is sufficient and deeper models offer no additional advantage. The performance increase during trainable configurations for both variants, compared to their respective frozen variants, confirms that residual architectures are inherently more robust to full network fine-tuning. Similarly to ResNet-18, both variants achieved a significant boost in performance compared to their from-scratch counterpart.

Swin-V2-B [8], a Swin Transformer, uses a hierarchical vision transformer with local window-based self-attention and shifted windows to efficiently capture both fine-grained details and global context of the image. The results from this model were completely opposite compared to what we have seen so far under the pretrained category. This model showed an excellent performance of 92.1% on the validation set. While during end-to-end fine-tuning, the accuracy significantly drops, reaching around almost 84%. This suggests that fine-tuning end-to-end has probably disrupted the prominent internal representations that the model had during large-scale pretraining. As we see in the upcoming paragraph, the same trend has also been witnessed for the CLIP model (a vision transformer with self-attention). Essentially, this suggests that, unlike CNN, training these architectures on a small and less diverse dataset could really degrade the highly informative features learned from pretraining and may require quite specific fine-tuning strategies.

ConvNeXt-Large [9], a CNN architecture, is designed to bridge the performance gap between the transformers and the CNN architecture. Inspired by transformers, they adopt elements such as large kernel sizes, inverted bottleneck blocks, and layer normalization, etc., to enhance their representational power. When only the classification head was trained, the model reached an accuracy of 92.9%, which suggests the pretrained model provides semantically rich features and features can be generalized across different domains. When we unfroze the backbone and trained the model end-to-end, the performance only increased by 0.4%, which shows that it slightly improved its feature representation and was slightly better suited to our dataset. However, the important thing that we noticed here, even though it was inspired by transformer architecture, it still didn't show any performance drop when end-to-end fine-tuned, suggesting it preserves the same robustness as StandardCNN, while also bringing the transformer-based architecture benefits to the pipeline.

CLIP-B32, a vision transformer pretrained via contrastive learning on 400-million image-text pairs [10], encodes visual features in a global, semantic-rich embedding space. In the frozen setup, where the whole backbone remains the same except for the final classification layer, the model performed quite well, achieving around 95% accuracy, showing the learned visual representation of this model transferred well to our task. However, when we made the entire model trainable (fine-tuning), hence allowing the flow of gradients through the entire architecture, the validation accuracy significantly dropped by over 25%, along with some signs of overfitting. This sudden drop in performance suggests that fine-tuning all the layers on the small dataset, like ours, could severely disrupt the strong visual features already learned by the model, causing overfitting. Additionally, as it has been trained for a different objective, i.e., image-text alignment, so updating all its weights altogether on such a small dataset is highly likely to be the reason for this performance drop.

CLIP-L14 showed a similar trend to CLIP-B32. The frozen variant yielded even stronger results on our dataset, achieving over 96% accuracy, which is the highest among all the models and configurations. Since CLIP-L14 is a larger and more expressive model than CLIP-B32 and accepts an input size of 336 x 336 px, allowing it to capture more detailed spatial features and, in turn, richer semantic embeddings. Although fine-tuning again hurt the strength of this model compared to its frozen setting, however, it still outperformed B32, which suggests this model's higher capacity allowed it to fine-tune more effectively.

5.7 Generalization on Test Set:

After experimenting with different models and hyperparameter configurations, we leveraged the best-performing models on the unseen data to evaluate the generalization. To keep the analysis concise and within the report's page limit, we only reported the best results from each category (Scratch or Pre-trained), as shown in Table 1.

As the results demonstrate, among all the models that are trained from scratch, StandardCNN has performed the best overall, achieving an accuracy of almost 91%, with balanced precision, recall and f1 score. Even though the Attention_CNN model has achieved the highest validation, but it could not generalize the same on the test set, achieving a little lower accuracy of 90.40% compared to StandardCNN. The AutoEncoder with Classification head, however, performed the worst out of all, even after all combinations of hyperparameter tuning. For the ResNet-18 variant, despite the model robustness and generalization ability, it didn't outperform the StandardCNN, likely because of the small size of the dataset which could not effectively leverage the deep architecture of the ResNet-based models. Similarly, the case with FPN, it also could not outperform the StandardCNN.

Table 1: Test set Performance of models trained from scratch and pretrained models on the Intel Image Classification. "-" indicates no metric result available for a particular model. **Bold red** indicates the highest performance, **bold blue** the second highest, and **bold orange** the third highest.

		Generalization on Test Set			
Training Type	**Model**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Scratch	StandardCNN	**90.97%**	**91.10%**	**91.11%**	**91.10%**
	AE_with_Classifier	80.00%	80.41%	80.00%	80.13%
	Attention_CNN	90.40%	90.57%	90.57%	90.56%
	ResNet-18	86.20%	86.38%	86.41%	86.38%
	FPN+StandardCNN	88.67%	89.24%	89.09%	88.81%
Pre-trained	CLIP-L14 (Frozen)	**95.27%**	**95.34%**	**95.34%**	**95.34%**
State-of-the-Art	CNN [7]	**99.92%**	–	–	–

On the other hand, the frozen variant of CLIP-L14 outperformed all the scratch-trained models and also the pre-trained models, achieving an accuracy of 95.27%, while precisely balancing precision and recall. This shows the generalization ability of this model across different domains.

While CLIP-L14 performed the best across all our experiments. However, the State-of-the-Art benchmark has reported 99.92% accuracy on this dataset likely representing the results that achieved under highly-optimized conditions, with no reported precision, recall, and F1 score.

6 CONCLUSION/FUTURE WORK

This study evaluated six DL approaches on the Intel Image Classification Dataset, comparing models trained from scratch with pre-trained alternatives. Pre-trained models significantly outperformed scratch-trained models, with CLIP-L14 achieving 96% validation accuracy, followed by CLIP-B32 at 95%. Among scratch-trained models, attention mechanisms proved most effective, with BAM achieving 91.5%. Our findings indicate that frozen pre-trained models, particularly CLIP variants, should be prioritized for optimal performance. When unavailable, attention-based CNNs offer the best alternative. Transfer learning consistently outperformed architectural complexity, as deeper ResNets and feature pyramid networks did not improve performance and may cause overfitting on limited datasets. This work demonstrates the value of transfer learning over architectural complexity in image classification. Future research should explore advanced fine-tuning techniques for pre-trained transformers and dimensionality reduction methods like UMAP for feature visualization.

7 STATEMENT OF CONTRIBUTION:

Nehme Haidura:Implemented StandardCNN, AE, CNN+Attention. Wrote: Abstract,1,4,5.3,5.4,conclusion

Atul Kumar: Implemented RL+Skip, FPN, Pre-trained models. Wrote: 2,3,5.1,5.2,5.5,5.6,5.7

REFERENCES

[1] Puneet Bansal. 2019. Intel Image Classification. (2019). https://www.kaggle.com/datasets/puneet6060/intel-image-classification.

[2] Ajay Uppili Arasanipalai Qibin Hou Diganta Misra, Trikay Nalamada. 2020. Rotate to Attend: Convolutional Triplet Attention Module. (2020). https://arxiv.org/pdf/2010.03045.

[3] Mahmudul Hasan. 2024. Vision Eagle Attention: a new lens for advancing image classification. (2024). arXiv:cs.CV/2411.10564 https://arxiv.org/abs/2411.10564

[4] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2261–2269. https://doi.org/10.1109/CVPR.2017.243

[5] Li Shen Jie Hu and Gang Sun. 2017. Squeeze-and-Excitation Networks. (2017). https://arxiv.org/pdf/1709.01507.

[6] Joon-Young Lee In So Kweon Jongchan Park, Sanghyun Woo. 2018. BAM: Bottleneck Attention Module. (2018). https://arxiv.org/abs/1807.06514.

[7] Athanasios Kanavos, Orestis Papadimitriou, Khalil Al-Hussaeni, Ioannis Karamitsos, and Manolis Maragoudakis. 2024. Analyzing Deep Learning Techniques in Natural Scene Image Classification. In *2024 IEEE International Conference on Big Data (BigData)*. 5682–5691. https://doi.org/10.1109/BigData62323.2024.10824948

[8] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. *CoRR* abs/2103.14030 (2021). arXiv:2103.14030 https://arxiv.org/abs/2103.14030

[9] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. A ConvNet for the 2020s. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11966–11976. https://doi.org/10.1109/CVPR52688.2022.01167

[10] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. *CoRR* abs/2103.00020 (2021). arXiv:2103.00020 https://arxiv.org/abs/2103.00020

[11] Mohammad Rahimzadeh, Shaghayegh Parvin, Alireza Askari, et al. 2024. Wise-SrNet: a novel architecture for enhancing image classification by learning spatial resolution of feature maps. *Pattern Analysis and Applications* 27, 30 (2024). https://doi.org/10.1007/s10044-024-01211-0 Published: March 9, 2024.

[12] Ruman. 2025. Feature Pyramid Network for Multi-Scale Detection. https://rumn.medium.com/feature-pyramid-network-for-multi-scale-detection-f573a889c7b1. (2025). Accessed: 2025-06-29.

[13] Joon-Young Lee In So Kweon Sanghyun Woo, Jongchan Park. 2018. CBAM: Convolutional Block Attention Module. (2018). https://arxiv.org/pdf/1807.06521.

[14] Aditya Sharma. 2018. Intel Autoencoder as a Classifier using Fashion-MNIST Dataset Tutorial. (2018). https://www.datacamp.com/tutorial/autoencoder-classifier-python.

[15] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 6105–6114. https://proceedings.mlr.press/v97/tan19a.html

[16] Jianshu Li Shuicheng Yan Jiashi Feng Yunpeng Chen, Yannis Kalantidis. 2018. A2-Nets: Double Attention Networks. (2018). https://arxiv.org/pdf/1810.11579.