Analysis of Dense Pixel Prediction Tasks Project Report : CS 7643

Yuemin Zhou, Brandon Sheffield, and Saheon Kim
Georgia Institute of Technology
{yzhou43, bsheffield7, skim935}@gatech.edu

Abstract

In this group project report, we analyze dense pixel prediction performance such as object detection and semantic segmentation performance on a subset of the COCO 2017 data set using limited computational resources. We distribute experiments among team members using distinct deep neural network architectures and collectively analyze the results. Future work proposes an architectural enhancement inspired by recent work done by Meta Research.

1. Introduction/Background/Motivation

The underlying motivation behind this project for the team is to gain practical experience with handling 'large' data sets, modern deep neural network architectures, popular frameworks, and build up intuition through experimental results and analysis. The tasks we decided to focus on are common in dense prediction tasks[16] found in computer vision such as object detection[20] and semantic segmentation[7]. The object detection task involves identifying a particular class in an image and drawing a bounding box around it. The semantic segmentation task involves pixel level identification of similar objects as a single class.

To equally distribute contributions among the team, we each chose a deep neural architecture that is relevant to the tasks of object detection and semantic segmentation. The team ended up independently choosing two types of general neural network architectures, the Convolutional Neural Network(CNN) and the Vision Transformer(ViT)[3]. Furthermore, we chose a third option, a type of network design space called RegNet, and applied this to the CNN to see whether this brings significant changes. Also a common framework is used among the team where each team member would have access to common building blocks, tools for analysis, and similar pipelines for training and testing models. This early decision would prove beneficial for assisting one another in troubleshooting and comparing methods. We leveraged the popular MMDetection[2] toolbox which has PyTorch deep learning framework support. Further details on deep learning frameworks, existing code, source code for experiments, and models are detailed in the Appendix section A.

For Faster R-CNN, we want to test it on the truncated Coco 2017 data set and see how compressed images effect the accuracy of the model. We will be using Principal Component Analysis(PCA) to compress the images so there will be 3 data sets, one baseline and one that has 50 percent of the principle components and one that has 25 percent of the principle components. We will train the model on the baseline and compressed images and see how each model performs on the actual uncompressed images in test. We will also test pre-trained models and models with slight architectural differences on the same data set.

1.1. How It Is Done Today

1.1.1 Faster R-CNN

CNNs have seen an explosion of success in computer vision tasks in the last decade. CNNs are known to exhibit inductive biases that make them adept at analyzing images for example. Such inductive biases include: locality, translation equivariance, and translation invariance.

Currently many modern CNN's used in object detection add a Feature Pyramid Network(FPN) [8] and a Region Proposal Network(RPN) on to the backbone CNN network. Faster R-CNN[13] is a form of this approach. The first layer is a convolution layer from Resnet. It has four stages. The results of this layer are connected to a Feature Pyramid Network[8]. The FPN is a feature extractor which extracts the necessary feature maps that are needed for object detectors. At each stage of the Resnet from the bottom up, a feature pyramid network will decrease spatial resolution and increase the semantic value. Therefore at the top layer the semantic value is highest. This top layer is then used to reconstruct layers or feature maps which are then used in the next step which is the region proposal network.

In the RPN, first anchor points are generated. Every point in the feature map is an anchor point. Then anchor boxes are generated for every anchor point. These anchor boxes are generated with the scales and ratio set; they determine the sizes of the boxes. Initially these boxes are dummy boxes so the model has to learn whether the given box is

foreground or background and at the same time it needs to learn the offsets to adjust the boxes so they fit the objects. This learning is done through regression. Once the foreground, background scores and offsets of anchor boxes are learned then the anchor boxes are processed again using proposal generation. Final proposals for the bounding box of an object are done through a region of interest pooling layer.

1.1.2 RegNet

One major flaw for CNN (or any other types of networks) is that sometimes, it can be a bit hard to generalize this to many different types of tasks and data. This is because each of those tasks and data will likely require different sets of hyperparameter combinations. Because of this inconvenience, there has been a recent attempt to focus not on the design and tuning of individual networks, but rather combine the effort of tuning an individual network and finding the best architecture of the network for the given task. This is where RegNet was formed. RegNet came from an effort to find a general set of network designing principles that would not give us not just one best architecture for a single network, but find a design space of certain types of networks that would be the most efficient in doing different tasks [11]. On certain studies, the performance of RegNet was found to be better and faster in similar training settings. This is why RegNet was also explored in addition to the CNN. One caveat is that RegNet is optimizing a much broader space, so it may provide slightly worse results compared to tuning an individual network for a particular task.

1.1.3 Swin Transformer

Inspired by the general learning architecture of transformers[17] that have made breakthroughs in natural language processing, vision transformers were introduced as a competitive architecture for computer vision tasks such as image classification[3] with reliance on large-scale pre-training data. Shortly after the introduction of the ViTs that showed to achieve comparable or even superior performance on image classification tasks, the Google Brain team analyzed how ViTs are solving these tasks and how do their learned representations compare to CNNs[12](inspired from the third graded discussion readings in this class). Their findings discovered that ViTs show strong preservation of spatial information which is useful in other computer vision tasks such as object detection. Since then, vision transformers have made their way into object detection setting key milestones[1]:

The major strength of ViTs is also it's disadvantage in its abilility to establish long relationships. As a result, ViTs require vasts amount of data, time, and computational memory due to the quadratic cost of self-attention that is dependent on the number of patches. In object detection and segmentation tasks, a wxh image has a complexity of $O(w^2h^2)$. Needless to say, processing high-resolution images come at high cost often intractable. As a consequence, the general-purpose transformer backbone called the Swin Transformer[10] was proposed which constructs hierarchical feature maps and has linear computational complexity to image size. It inherits the advantages of the CNN as it fully considers the size invariance and relation between receptive fields.

1.2. Limitations of Current Practice

Object detection takes a lot of computational capacity and space. One area specifically is the images themselves; high quality images take up a lot of space. Therefore its worthwhile to explore methods that allow for a decrease in the space needed to store these images. We will decrease the quality of the images by taking smaller principle components of each image. This approach will simulate the use of lower quality images in training.

Limitations of the Swin Transformer[10] is that it does not not have as many inductive biases as the CNN such as translation invariance. This in turns make learning very difficult often requiring larger data sets and/or stronger data enhancements to achieve better learning performance. The impact on the amount of data required to train effective transformers can be see with GradCam[14]. Appendix C explores various Swin Transformer sizes trained on Imagenet1K[6] and ImageNet21k data sets. Additionally, applied researchers in the field of remote sensing applications[19] report that the architecture is limited on its ability to encode context information which affects detection accuracy of small-scale objects.

1.3. Who Cares?

The scope of our work is aimed at practitioners such as applied research scientists, machine learning engineers, robotics engineers, and anyone interested in practical performance of CNN and ViT based models perform. We motivate this project by diving deeper into the strengths and weaknesses of CNNs and ViTs through controlled experiments that compare performance on object detection and semantic segmentation tasks.

1.4. Data Set

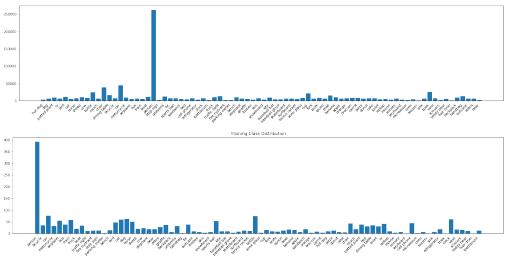
Here we briefly discuss the the most important aspects[4] of the data set used for experiments. We use the popular Microsoft COCO 2017 data set[9]. The name COCO standards for Common Objects in Context(COCO) as it was designed to represent a vast array of everyday objects. It is widely considered the gold standard for tasks such as object detection, semantic segmentation, and key point detection. The data set consists of 121,408 images, 883,331 object annotations, 80 classes, and an average image ratio of 640x480.

For the experiments, we are interested in object detection and semantic segmentation performance.

Due to computational constraints and limitations of the models we chose we eventually settled on a truncated version of the COCO 2017 data set. This data set has the bounding box (bbox) annotations and the mask annotations needed for Faster R-CNN and Mask R-CNN. This data was further truncated to 1977 images for train, 500 for test, and 494 for validation. These images were all taken from the original training data set. This drastic truncation is needed due to our limited time and computational hardware.

For the image compression experiments the images were converted to two compressed image sets. One set that has 50 percent of the principle components and another set that has 25 percent of the principle components. The training and validation data was converted this way but the test data was left the same to simulate how modeled trained on compressed images would perform in the real world.

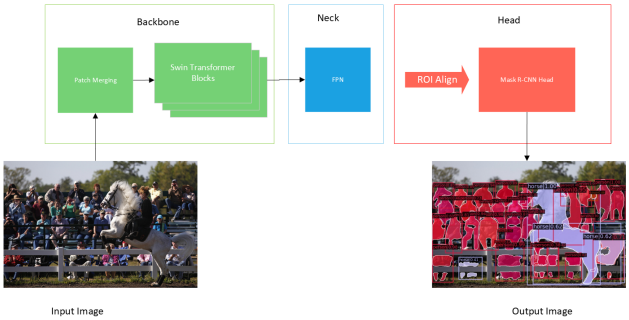
Also for coco we focused on one class classification because the classes in the data set was quite imbalanced as can be seen from the following training class distribution figures:



2. Architecture Designs

2.0.1 Mask R-CNN Swin Transformer

As illustrated in the following diagram, the deep neural network architecture presented is composed of 3 main components: backbone(Swin Transformer), neck(Feature Pyramid Network), and model(Mask R-CNN).



An image of size 256x256 is expected of the model but not a strict requirement. Before an image is fed into the architecture above, a set of pre-processing stages occur after loading the image: the image is resized to 1333x800, randomly flipped, normalized, and padding is added.

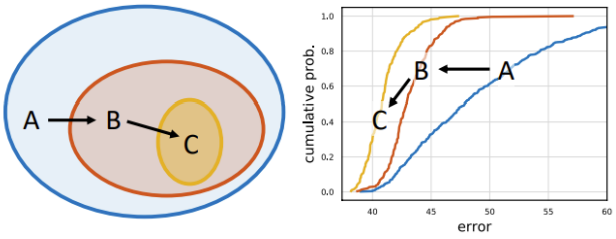
The following is a table of a limited set of hyper-parameters in the architecture components that are used to tune model performance:

Table 1. Highlighted Architecture Hyper-parameters

Component	Values	Selected Values
Backbone	Window Size	7
Backbone	Activation	GELU
Backbone	Layers	LayerNorm, 2-layer MLP
Backbone	Attn Heads	3, 6, 12, 24
Optimizer	AdamW	0.0001
Neck	In Channels	96, 192, 384, 768
Method	Loss	Cross Entropy
Method	Mask Size	28

2.0.2 RegNet

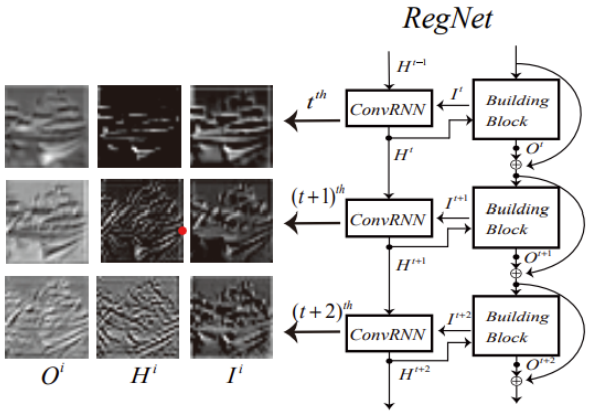
RegNet came from an effort to not find the best network, but the best network design space. For our case, we are specifically talking about CNNs. The process was to first choose a space called AnyNet, which is virtually a space without any conditions, and then add conditions to see if the error distributions are improved optimally. The below image shows that as we go from space A to space C, our error distributions get better, making C a better space than A and B. Do note that most of the search was done in a setting that required low computational resources.



The key to the RegNet space is that the width and depth of the network is determined by a quantized linear function [11]. We also have to quantize this, and this lets us have a quantized linear parameterization of width of block j to be $u_j = w_a(j+1)$ where w_a is simply the slope.

From here, we can build a more specific design space called RegNetX that was utilized in our experiments. The name comes from the so called X block, which includes bottleneck layers. Bottleneck layers are simply layers with fewer neurons than adjacent layers, and this is done so that we can compress feature representations to operate in limited computation spaces. This block has 1-by-1 convolution layers at the start and at the end, and it has a 3-by-3 group convolution layer in between. The block has 3

parameters: width, group width, and the bottleneck ratio that determines how much fewer layers the bottleneck layer will have [11]. The below diagram would represent how a RegNetX applied on CNN would look like roughly [18].



I^i / H^i: The input/ouput of ConvRNN at i^{th} building block

3. Approach

3.1. What We Did

The main objective we wanted to study was how well do the CNN and Transformer deep neural networks compare in controlled experiments on the same data set. We foresaw having different architectures being an issue when it came to comparing results and performance which motivated the idea of having a common framework. Upon doing research on object detection methods, we found a popular toolbox in which to base our experiments on, MMDetection[2]. See Appendix A regarding information related to deep learning frameworks, existing code, source code changes, and existing models used to produce experimental results.

In the CNN model set we picked the Faster R-CNN to do image compression experiments on. The compression experiments would be done on 3 different models; one without any pre-tuning, one with pre-tuning and one that has a slight architecture change and has no pre-tuning. The model that has the slight architecture change has the scales doubled from 8 to 16. Scales is one of the parameters that determine the anchor box sizes therefore we were hoping that large anchor boxes sizes could overcome the blurriness that comes with compressed images and therefore give better results. All three models will be trained and validated on three different data sets but will be tested on one data set. The three data sets used for training and validation are; a set of uncompressed images, and a set of compressed images that have 50 percent principle components and a set of images that have only 25 percent principles components. All three data sets are the same truncated coco images and are essentially the same, other than the compression done by PCA. Testing data will only be on uncompressed images. Essentially we want to see how models perform on the original images when trained on compressed images.

For comparing models, CNN (and RegNet) to the Swin Transformer, we generated baseline performances on the truncated coco data set then did some fine-tuning of the model parameters. Existing pre-trained models were used as baselines. Fine tuning was done on model parameters and some architecture parameters as time allowed. An enhancement to the Swin Transformer is proposed in Appendix B that serves as future work.

3.2. Problems Anticipated and Encountered

The types of architectures that we experimented with and analyzed have an inherit difficulty in requiring lots of hardware resources such as storage and GPUs for training models from scratch. It was learned early that using the COCO 2017 data set would involve several days of training without hyperparameter tuning as well as exceed current GPU resources which motivated the idea of reducing the data set size. Even for fine-tuning training jobs, the authors experienced varying memory usages of 3GB to 10GB of GPU for small batch sizes which was surprising.

4. Experiments and Results

4.1. Measuring Success

Summary of metrics referenced in the following experiments use to gauge success:

- box mAP - measures how 'tight' the predicted bounding box on average is to the ground truth label, It is the mean of the all the average bounding box precision.
- mask mAP - measures how accurate the intersection of pixels are on average to the ground truth label
- Floating Point Operations(FLOPs) - Lower values are better as it means the model uses less floating point calculations
- Parameters(Params) - measured in million(M), this metric is useful for practitioners interested in models that can fit into low memory compute environments like edge computing devices
- Frames Per Second(FPS) - how many images of size 1280x720, 720p resolution, a model can process in a second

4.2. Overfitting and Generalization

When assessing models for overfitting and how well they generalized to new data, we examined each model's training accuracy and validation bbox average precision. We found that for all model experiments that both metrics were increasing like in Figure 1 therefore we can say that we are not over-fitting. Below is an image that shows a sample of

the training losses and validation losses for RegNet. For brevity, we do not show the Swin Transformer and ResNet Faster R-CNN as they exhibit the same behavior.

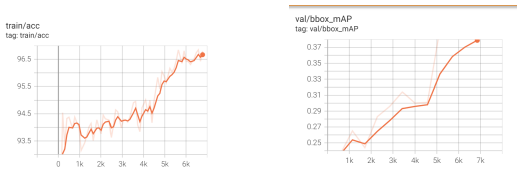


Figure 1. RegNet: Train/Validation accuracy

4.3. Experiment 0: Baseline

Backbone	Method	Box mAP	Mask mAP
Swin-T	Mask R-CNN	0	0
Resnet50	Faster R-CNN	0.1	N/A
RegNet	Faster R-CNN	0	N/A

Table 2. Baseline performance

In order to gauge performance of experiments, we first established a baseline from the pre-trained ImageNet-1K models. More specifically, we want to know the performance of the pre-trained ImageNet-1K model on it's ability to generalize to the truncated COCO 2017 data set. One of the expectations we had for the experiment going in is that the pre-trained models would be able to reasonably generalize in testing since there are common classes such as person. However, results were abysmal to say the least. We suspect the inherit difficulty bias in COCO coupled with overfitting to the ImageNet pre-trained models that were trained on 256x256 images are the main culprits for poor model performance. Other ideas we brainstormed were not enough hyperparameter tuning using optimization techniques such as grid search, random search, or Bayesian search which is a consequence of limited time and computational power.

4.4. Experiment 1: Fine-Tuned Performance

Backbone	Method	Box mAP	Mask mAP
Swin-T	Mask R-CNN	42.7	39.3
Resnet50	Faster R-CNN	54.6	N/A
RegNet	Faster R-CNN	45.0	N/A

Table 3. Fine-tuned performance

One of the surprising outcomes when comparing model performance was how underperforming the Swin Transformer was in comparison. We speculate that the under performance can be explained by the structure of the model type, Mask R-CNN in this case, to the structure of the data which contains annotations for Box and Mask mAP values. One motivating factor for using the Swin Transformer Mask R-CNN model is that it is the only one capable of performing semantic segmentation and object detection tasks which makes it a versatile model.

For Faster RCNN the model had hyper parameters of learning rate, epochs, warm ups, and evaluation metric. These were all set from a baseline configuration; epoch is 12, warmup is linear and evaluation metric is linear. The linearing rate was also automatically set with respect to how many GPU's the computers was using.

For the RegNet, the additional X blocks that we have are not really possible to tune due to the fact that much of these building blocks are set. The studies have also shown that bottleneck ratio of 1 also gives the best results. Therefore, in order to fine tune this, we have to tune the simpler Faster R-CNN component, tuning the hyper-parameters stated above. This still gives a different result than simply tuning the faster R-CNN by itself. Results are shown in Figure 1. There does not seem to be any overfitting based on the figure.

For all models the loss function is cross entropy and the optimizer for Faster RCNN and Regnet is SGD. For Swin the optimizer is AdamW.

4.5. Experiment 2: Model Performance and Complexity

Backbone	Method	FLOPs (G)	Params (M)	FPS
Swin-T	Mask R-CNN	263	48	23.8
Resnet50	Faster R-CNN	202	41	15.4
RegNet	Faster R-CNN	183	31	7.4

Table 4. Model performance and complexity with 720p images

Scalability of models is a performance characteristic of interest to practitioners so we focus on computational complexity of the models in this experiment. As one can see, there is a performance gap between the Swin-T, RegNet, and Resnet50 Faster R-CNN model. One can observe a correlation between FLOPS and the number of parameters for each model. This makes sense intuitively that a larger complex model implies that there are more floating point operations to perform. What we found surprising is that the Swin-T model is able to process more images per second than the Faster R-CNN model while having a higher count of FLOPs and parameters.

Our RegNet model is also a simplified version of faster R-CNN but has the additional RegNetX component. RegNet also already attempts to choose the simplest model that optimizes the error distribution, so it was fully expected to see the lower number of parameters. Because RegNet is a model that attempts to simplify the model as much as possible, it is fully expected that RegNet has least number of parameters, and therefore, the least amount of FLOPs.

4.6. Experiment 3: Faster R-CNN Results for image compression

This set of experiments will consist of three experiments for image compression. The first experiment in this set is for a baseline model without any pre-trained model loaded. We will train this baseline model on the three data sets. The charts below show the test results.

Data used	Box mAP
uncompressed	32.3
compressed, 50 percent PCA used	31.7
compressed, 25 percent PCA used	30.1

Table 5. Resnet 50 Faster R-CNN Baseline performance

The second experiment in this set is with a pre-trained model. We will retrain this pre-trained model on the three data sets. The charts below show the test results. The results here was much better than the other experiments in this set. This is mainly due to loading a pre-trained model that already saw some configurations of people.

Data used	Box mAP
uncompressed	54.6
compressed, 50 percent PCA used	54.5
compressed, 25 percent PCA used	53.0

Table 6. Resnet 50 Faster R-CNN pre-trained performance

The third experiment in this set is with a model that has a slight architecture change. These results are test data taken from a model that has a slight architecture adjustment which is doubling the scales used to generate the anchor boxes. The scales are used to generate the sizes of the anchor boxes; the idea is to increase them so that any blurriness from the compressed images can be overcome. No pre-tuning was done.

Data used	Box mAP
uncompressed	31.1
compressed, 50 percent PCA used	30.9
compressed, 25 percent PCA used	29.2

Table 7. Faster R-CNN architecture adjustment performance

The next figure is from the baseline model which shows the training and validation accuracy. Both are increasing throughout the training regime, therefore it shows that the model is not over fitting. The whole set of figures for this experiment on compressed images can be found in the appendix D.

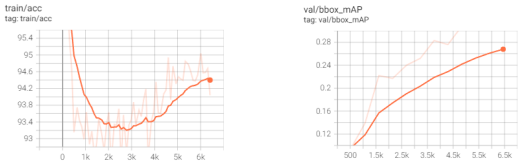


Figure 2. Faster R-CNN baseline training accuracy and validation accuracy

Overall for the Faster R-CNN experiments done on image compression, the results showed that in all cases the models trained on uncompressed images did the best. However the 50 percent compressed image experiments did almost as good. Performance did seem to drop off much more when only 25 percent of the components are used. Therefore potentially we could train models on compressed images that only have 50 percent of the principle components and not get a large performance hit. Also in all experiments the same hyper-parameters were used to make sure the model itself stayed the same and the only thing that changed was the images.

Also the slight architecture change of doubling the scales did not result in better performance. This means that increasing the anchor box sizes did not result in better performance. This could be due to many factors, perhaps the scale used was already on the larger side for the image. Further analysis could be done with other parameters to change anchor box sizes however we did not have more time to do further experiments.

5. Future Work

With the rise of attention-based neural networks, wider architectures is once again being revisited[5]. In the work "Non-deep Networks", an architecture with several parallel branches is proposed that leads to more complex design. In the work, "Three things everyone should know about Vision Transformers", the authors propose three methods for ViTs that are much simpler and flexible alternative methods: parallel blocks, fine-tuning, patch pre-processing[15].

Future work motivated by the implementation of these three methods mechanisms into the Swin Transformer shows promise. In the appendix section B, it is illustrated how the parallel blocks can be implemented for future research opportunities.

6. Work Division

Student Name	Impl/Experiments/Analysis
Yuemin Zhou	Faster R-CNN
Brandon Sheffield	Swin Transformer
Saheon Kim	RegNet

Table 8. Contributions of team members.

References

[1] Ershat Arkin, Nurbiya Yadikar, Yusnur Muhtar, and Kurban Ubul. A survey of object detection based on cnn and transformer. In *2021 IEEE 2nd International Conference on Pattern Recognition and Machine Learning (PRML)*, pages 99–108. IEEE, 2021. 2

[2] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 1, 4

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1, 2

[4] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021. 2

[5] Ankit Goyal, Alexey Bochkovskiy, Jia Deng, and Vladlen Koltun. Non-deep networks. *arXiv preprint arXiv:2110.07641*, 2021. 6

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 2

[7] Fahad Lateef and Yassine Ruichek. Survey on semantic segmentation using deep learning techniques. *Neurocomputing*, 338:321–348, 2019. 1

[8] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 1

[9] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 2

[10] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021. 2, 9

[11] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. *CoRR*, abs/2003.13678, 2020. 2, 3, 4

[12] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *Advances in Neural Information Processing Systems*, 34, 2021. 2

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. 1

[14] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017. 2

[15] Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Jakob Verbeek, and Hervé Jégou. Three things everyone should know about vision transformers. *arXiv preprint arXiv:2203.09795*, 2022. 6, 9

[16] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Multi-task learning for dense prediction tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021. 1

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2

[18] Jing Xu, Yu Pan, Xinglin Pan, Steven Hoi, Zhang Yi, and Zenglin Xu. Regnet: Self-regulated network for image classification, 2021. 4

[19] Xiangkai Xu, Zhejun Feng, Changqing Cao, Mengyuan Li, Jin Wu, Zengyan Wu, Yajie Shang, and Shubing Ye. An improved swin transformer-based model for remote sensing object detection and instance segmentation. *Remote Sensing*, 13(23):4779, 2021. 2

[20] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Asghar, and Brian Lee. A survey of modern deep learning based object detection models. *Digital Signal Processing*, page 103514, 2022. 1

A. Existing Code and Models

References	Online Sources
Deep Learning Framework	https://pytorch.org/
Detection Framework	https://github.com/open-mmlab/mmdetection
Brandon Sheffield	https://github.com/mastash3ff/cs7643-GroupProject
Swin Transformer Models	https://github.com/open-mmlab/mmdetection/tree/master/configs/swin
ViT Gradcam	https://github.com/jacobgil/pytorch-grad-cam

B. Proposed Swin Transformer Enhancement

An enhancement that is proposed for the Swin Transformer is the use of parallel blocks that results in an architecture that has the same number of parameters and computational complexity while being wider and shallower. This insight was realized while surveying ViTs[15] and realized not to exist in Swin Transformer implementations as far as we know.

Given a sequence of transformer blocks:

$$x^{'}_{i+1} = x_1 + mhsa_l(x_l) \tag{1}$$

$$x^{'}_{i+1} = x^{'}_{1+1} + ffn_l(x^{'}_{l+1}) \tag{2}$$

$$x^{'}_{i+2} = x_{1+1} + mhsa_{l+1}(x_{l+1}) \tag{3}$$

$$x_{i+2} = x^{'}_{1+2} + mhsa_{l+1}(x^{'}_{l+2}) \tag{4}$$

these transformer blocks are combined into two parallel operations:

$$x_{l+1} = x_l + mhsa_{l,1}(x_l) + mhsa_{l,2}(x_l) \tag{5}$$

$$x_{l+2} = x_{l+1} + ffn_{l,1}(x_l + 1) + ffn_{l,2}(x_{l+1}) \tag{6}$$

The two parallel operations effectively result in twice the amount of processing that is performed in parallel while reducing the number of layers by two.

Here we illustrate the sequential then proposed parallel block methods for ViTs. Given a sequence of transformer blocks:

$$x^{'}_{i+1} = x_1 + mhsa_l(x_l) \tag{7}$$

$$x^{'}_{i+1} = x^{'}_{1+1} + ffn_l(x^{'}_{l+1}) \tag{8}$$

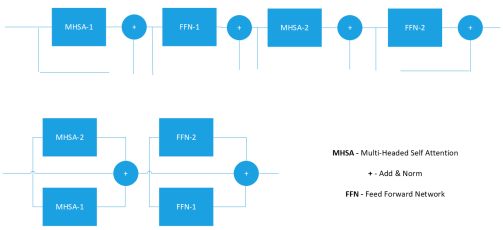
$$x^{'}_{i+2} = x_{1+1} + mhsa_{l+1}(x_{l+1}) \tag{9}$$

$$x_{i+2} = x^{'}_{1+2} + mhsa_{l+1}(x^{'}_{l+2}) \tag{10}$$

The original authors proposed the following parallel transformer blocks:

$$x_{l+1} = x_l + mhsa_{l,1}(x_l) + mhsa_{l,2}(x_l) \tag{11}$$

$$x_{l+2} = x_{l+1} + ffn_{l,1}(x_l + 1) + ffn_{l,2}(x_{l+1}) \tag{12}$$



The Swin Transformer is notably a variant of the original ViT therefore there are design changes to consider. For one, the Swin Transformer introduces the ideas of using the regular and shifted windows for computing attention which is not found in classic ViTs. The use of these windows avoids quadratic complexity of global attention with respect to the number of tokens as it would require an intractable amount of tokens for dense prediction tasks as well as computing on high-resolution images. The equations for computing these two windows are as followed:

$$\omega(MSA) = 4hwC^2 + 2(hw)^2C, \tag{13}$$

$$\omega(W-MSA) = 4hwC^2 + 2M^2hwC, \tag{14}$$

These windows can now be placed into the Swin Transformer blocks:

$$z^l = W-MSA(LN(z^{l-1})) + z^{l-1} \tag{15}$$

$$z^l = MLP(LN(z^l)) + z^l \tag{16}$$

$$z^{l+1} = W-MSA(LN(z^{l-1})) + z^{l-1} \tag{17}$$

$$z^{l+1} = MLP(LN(z^{l+1})) + z^{l+1} \tag{18}$$

The insight of parallel blocks can be adapted to Swin Transformers[10] which is a variant of ViTs. The original Swin Transformer two block layout is defined as:

$$x^{'}_{i+1} = x_1 + mhsa_l(x_l) \tag{19}$$

$$x^{'}_{i+1} = x^{'}_{1+1} + ffn_l(x^{'}_{l+1}) \tag{20}$$

$$x^{'}_{i+2} = x_{1+1} + mhsa_{l+1}(x_{l+1}) \tag{21}$$

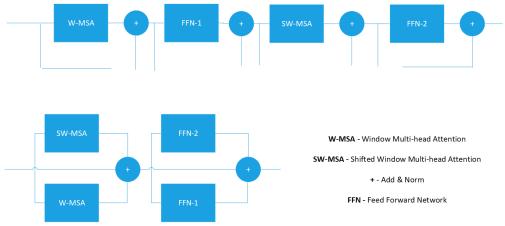
$$x_{i+2} = x^{'}_{1+2} + mhsa_{l+1}(x^{'}_{l+2}) \tag{22}$$

It can be observed that these two Swin Transformer blocks can also be placed in parallel fashion.

$$x_{l+1} = x_l + mhsa_{l,1}(x_l) + mhsa_{l,2}(x_l) \tag{23}$$

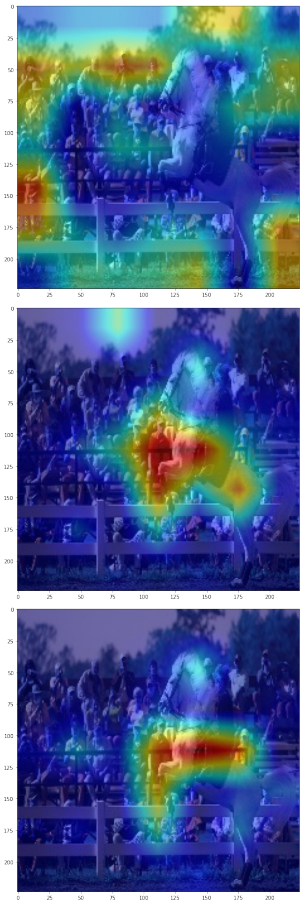
$$x_{l+2} = x_{l+1} + ffn_{l,1}(x_l + 1) + ffn_{l,2}(x_{l+1}) \tag{24}$$

The following diagram shows the newly paralleled blocks for the Swin Transformer.



C. Swin Gradcam

Visual Transformers are sensitive to the amount of data used to train them. This can be seen when the size of the models grow. The gradients can be visualized of where the network pays the most attention to in the following series of images.



D. additional charts for Faster R-CNN image compression experiments.

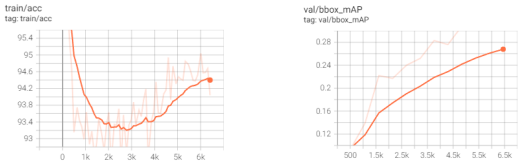


Figure 3. Faster R-CNN baseline training accuracy and validation accuracy

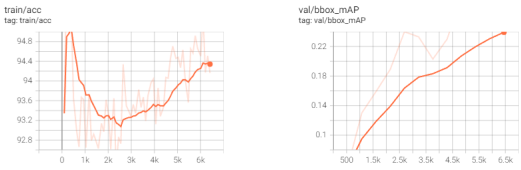


Figure 4. Faster R-CNN 50 percent PCA components training accuracy and validation accuracy

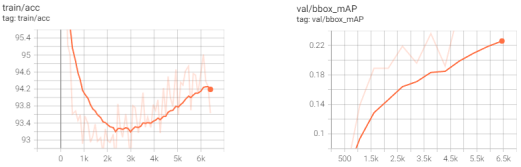


Figure 5. Faster R-CNN 25 percent PCA components training accuracy and validation accuracy

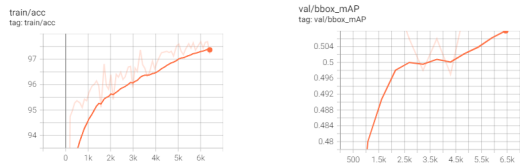


Figure 6. Faster R-CNN pre-trained training accuracy and validation accuracy

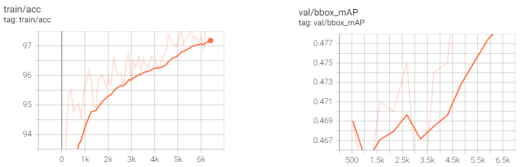


Figure 7. Faster R-CNN pre-trained 50 percent PCA components training accuracy and validation accuracy

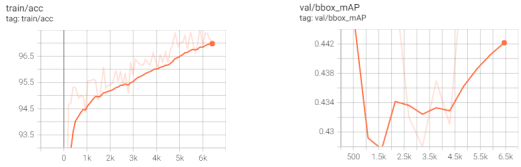


Figure 8. Faster R-CNN pre-trained 25 percent PCA components training accuracy and validation accuracy

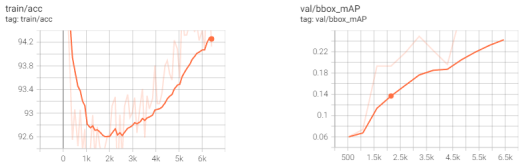


Figure 9. Faster R-CNN architecture edit training accuracy and validation accuracy

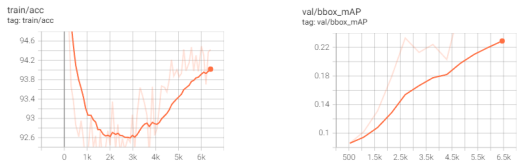


Figure 10. Faster R-CNN architecture edit 50 percent PCA components training accuracy and validation accuracy

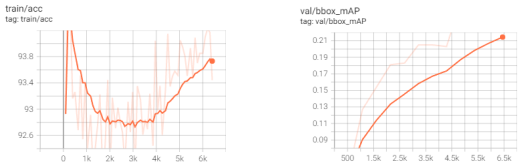


Figure 11. Faster R-CNN architecture edit 25 percent PCA components training accuracy and validation accuracy