

New Wave in Image Classification

Qizhen(Kurtis) Shen
George Washington
University
lootlecoop@gwu.edu
[du](#)

Changhong Zhang
George Washington
University
edwinzhang@gwu.edu
[.edu](#)

Hui Yang
George Washington
University
huiyang@gwu.edu

Abstract – The deep learning field has been gaining huge tractions ever since CUDA came out in 2007. Parallel computing on GPU released the full potential of deep neural networks, and accelerated design and testing cycles on orders of magnitude, enabling exploration of areas that had bottlenecks without meaningful progress for ages. With GPU computing, performance gap between CNN and traditional neural network had further widened. Alexnet, VGG, Resnet, etc., these big names crashed the scene as soon as they came out. Research had been gathered huge interests in beating these benchmarks. Improvements normally are based upon further increasing layers, complexify existing conv net instead of creating new ones. This paper experimented on a set of newly proposed architectures called transformers. Coupled with details tuning, our models yielded an improved classification performance.

Table of Contents

I. INTRODUCTION	3
II Data Collection	7
III Related Work	10
IV Project Flow and file Structure	11
Augmentation Results	13
V Modeling	13
VI Results	26
References	27

I. INTRODUCTION

Neural network is inspired by biological neural network. Alexander Bain (**Bain, 1873**) and William James (**James, 1890**) first proposed the preliminary theoretical base independently. Fully connected neural network is a network such that each node in the next layer is connected with all nodes in the previous layer. And to learn, these connections are updated through a process called backpropagation. In essence, it is a pattern finder. The problem however is that because of the fully connectivity, there are so many computations needed for pattern improving even though it is a magic box that can basically do any tasks. Thus, instead of focusing on generality, modifications had been made for specific tasks and the improvements are significant.

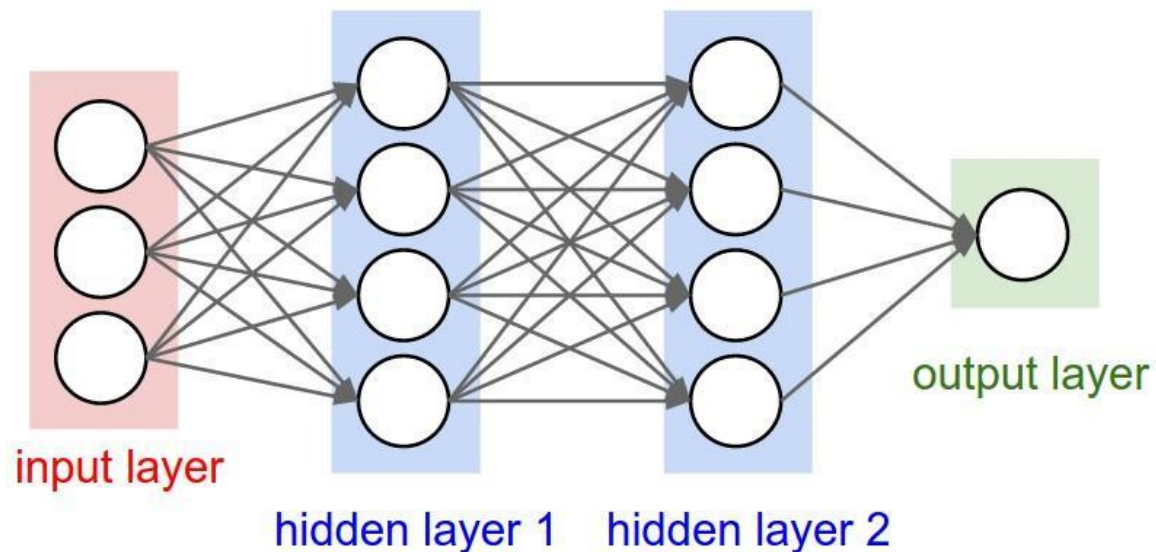


Figure 1 Fully Connected Neural Network

(CS231n Convolutional Neural Networks for Visual Recognition, 2022)

For image classification problem, again inspired by biological properties discovery from work by Hubel and Wiesel in the 1950s and 1960s (**Hubel & Wiesel, 1959**). Convolution neural network concept was developed by Kunihiro Fukushima in 1980 (**Fukushima, 1980**). He called the architecture neocognitron. Convolution neural network utilizes the mathematical concept of convolution, which is a mathematical operation on two functions to produce the third one with the operation being integral of the product of these two while inverting and shifting one of them (**Convolution, 2022**). It utilizes kernels as weight matrix and moves them across input plane independently, producing different feature maps. In essence, conv net acts as a focusing box before the fully connected layer, help reducing dimensions and computations while extracting important features for better classification.

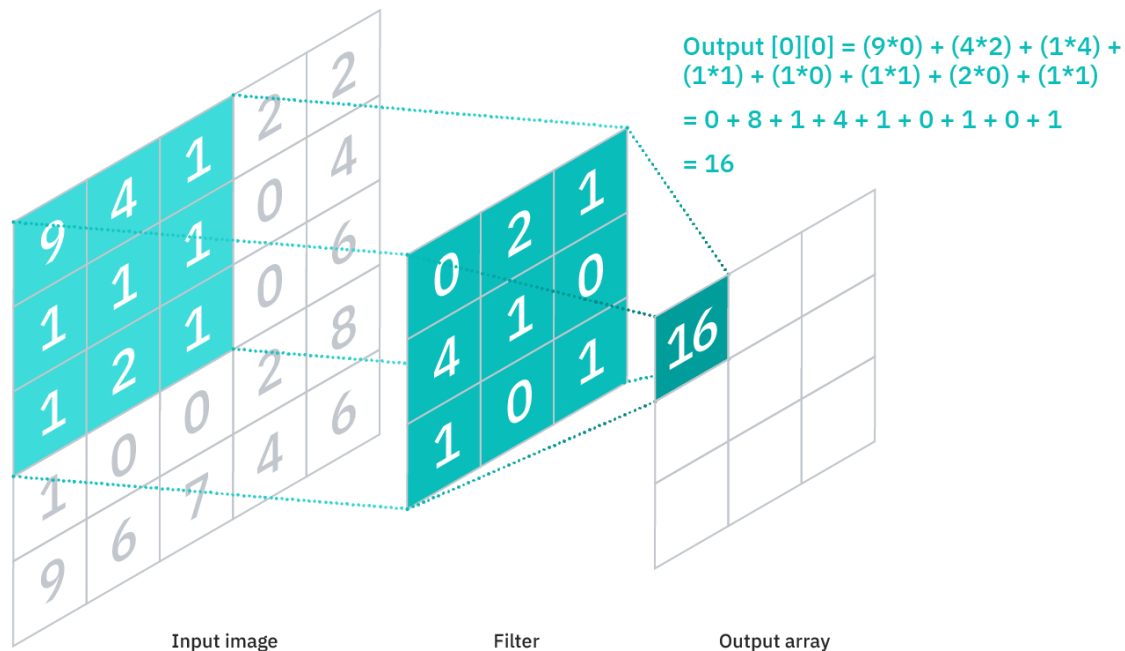


Figure 2 Convolution Neural Network

(IBM Cloud Education, 2020)

For prediction problem, concept called recurrent neural network was based on David Remelhart's work in 1986 (**Williams, Hinton, & Rumelhart, 1986**). Instead of previous mentioned memoryless network, recurrent one remembers the state of the hidden layers thus greatly improves the prediction accuracy of sequential dataset. It is widely used in speech recognition. In essence, recurrent net acts as memory holder inside fully connected layer.

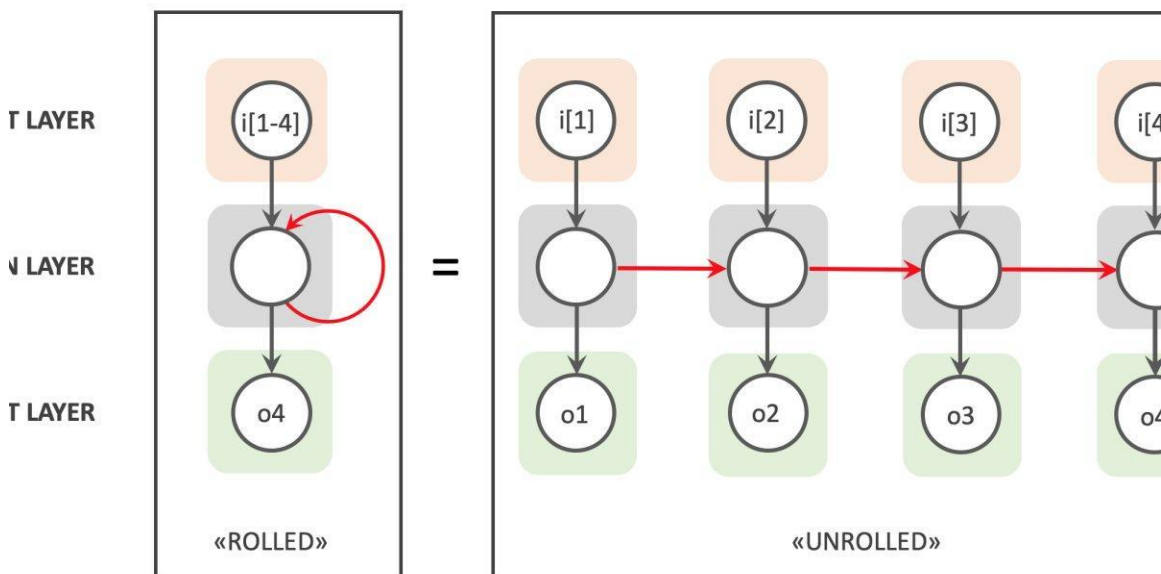


Figure 3 Recurrent Neural Network

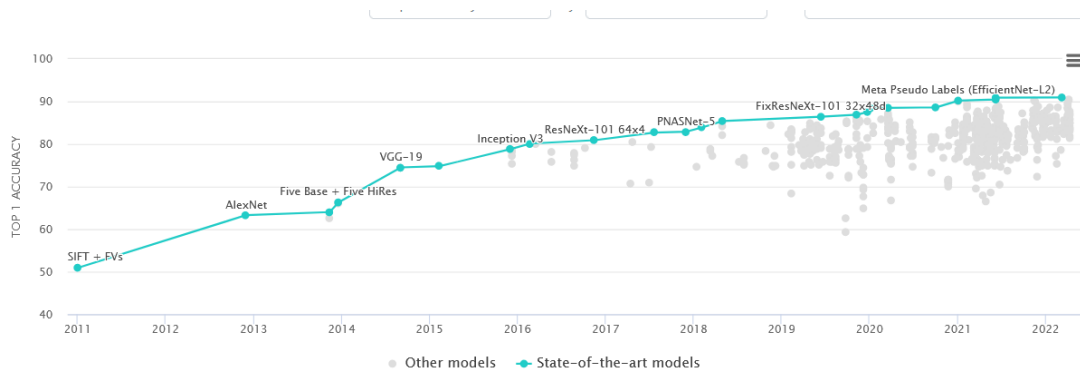
(West, 2019)

As can be seen, concepts of these networks were introduced quite early, and each new concept injected new blood into their respective field when introduced, generating great hype. However, performance staled after a short period as well.

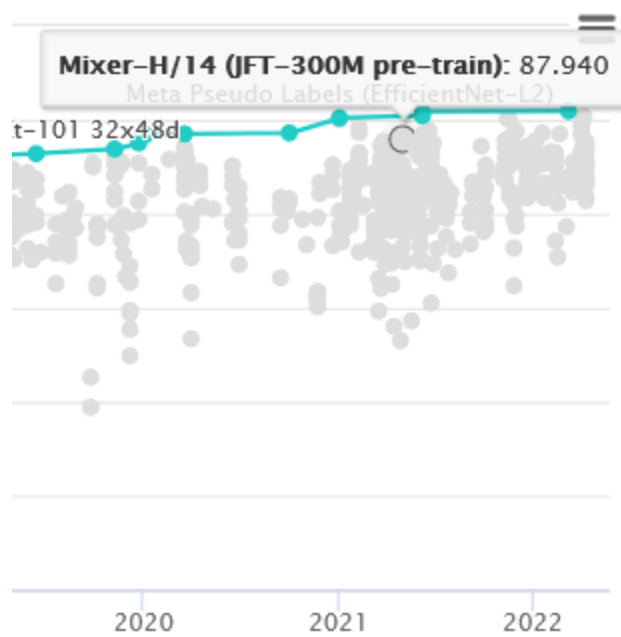
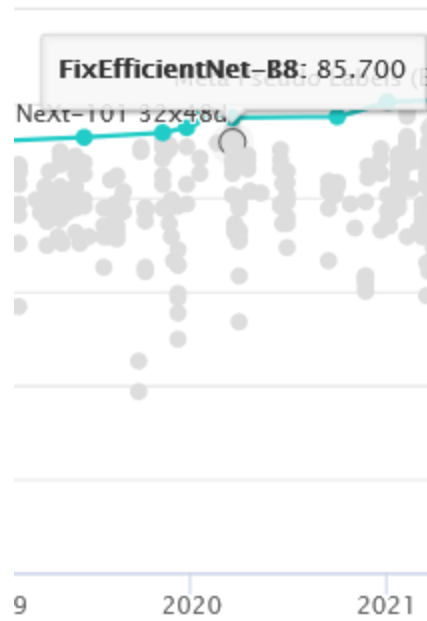
It was not until CUDA came out that a new wave of deep learning regenerated public interest. Compute Unified Device Architecture, or CUDA in short, is a parallel computing platform and application programming interface (API) that allows software to use specific types of graphic processing units (GPUs) for general purpose of processing (**CUDA, n.d.**). Development of new architectures took several years and in 2012, a CNN architecture named Alexnet shocked the world in the 2012 ImageNet Large Scale Visual Recognition Challenge, achieved a top-5 error of 15.3%, with the runner-up yielding more than 10.8% higher error percentage points (**Stanford Vision Lab , 2012**). In succeeding years, new architectures came one after another. GoogleNet, VGG, ResNet, EfficientNet all have impressive performances.

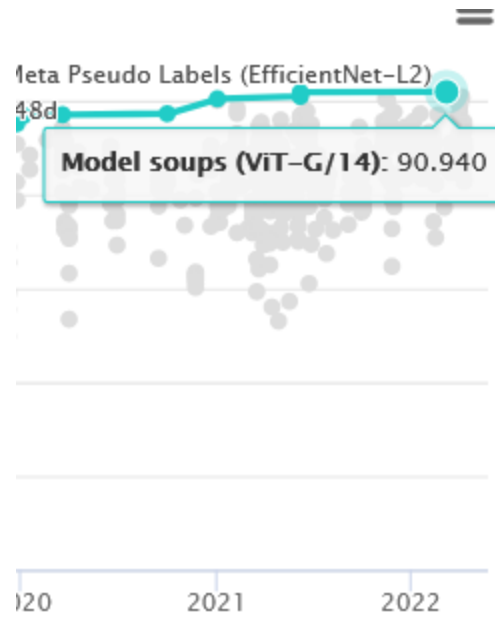
These architectures all competed based on ImageNet, which has been used as the industry standard testing bed for comparing deep learning architectures.

The accuracy progress graph over the years is displayed below (**Image Classification on ImageNet, n.d.**).



As can be observe, since 2020, architecture based on resnet no longer secured top positions. And when zoomed in on current performance, all top architectures are variants of Vision Transformers, with the concept transformer coming out in 2017. The dominate performance was the prime factor motivating us to conduct this research.





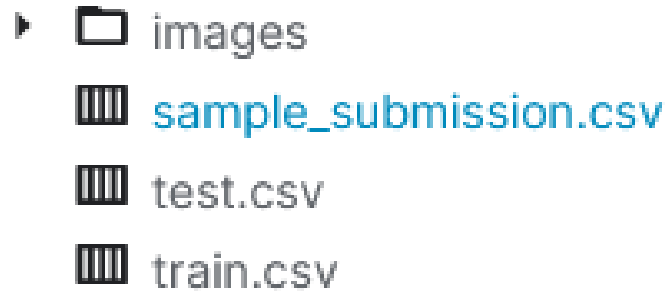
II Data Collection

Our dataset is from a recent Kaggle competition Classify Leaves. (**Classify Leaves, 2021**). It is a collection of leaf images.



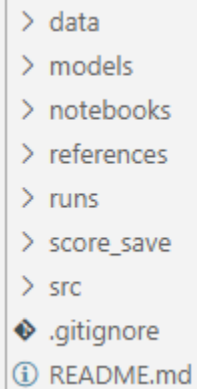
Images are split into 18353 training ones and 8800 testing ones, with each class having at least 50 images. Upon quick scan, each image is 224 by 224, and contains precisely one leaf, sitting

completely inside the frame which could explain for the impressive results (**Classify Leaves, 2021**).



(Classify Leaves, 2021)

According to the original data source, all images are contained inside the images folder, and labeling files are separated. Thus, we created our project structure as follows.

A screenshot of a file explorer showing the project structure. The files and folders listed are: data, models, notebooks, references, runs, score_save, src, .gitignore, and README.md. The 'data' folder is highlighted in blue.

All images are stored inside the data folder. The two labeling files are stored inside references folder. To explore the dataset, a jupyter notebook inside the notebooks folder is written to do some EDA. Below is what our training set looks like.

	image	label
0	images/0.jpg	maclura_pomifera
1	images/1.jpg	maclura_pomifera
2	images/2.jpg	maclura_pomifera
3	images/3.jpg	maclura_pomifera
4	images/4.jpg	maclura_pomifera
5	images/5.jpg	maclura_pomifera
6	images/6.jpg	ulmus_rubra
7	images/7.jpg	broussonettia_papyrifera
8	images/8.jpg	maclura_pomifera
9	images/9.jpg	broussonettia_papyrifera
...
18343	images/18343.jpg	prunus_virginiana
18344	images/18344.jpg	quercus_shumardii
18345	images/18345.jpg	ptelea_trifoliata
18346	images/18346.jpg	quercus_montana
18347	images/18347.jpg	ptelea_trifoliata
18348	images/18348.jpg	aesculus_glabra
18349	images/18349.jpg	liquidambar_styraciflua
18350	images/18350.jpg	cedrus_libani
18351	images/18351.jpg	prunus_pensylvanica
18352	images/18352.jpg	quercus_montana

18353 rows × 2 columns

Figure 4 Training Dataset

In total, there are 176 different kinds of leaves and each leaf corresponds to exactly one class, making our task a multiclass classification problem.

maclura_pomifera	353
ulmus_rubra	235
prunus_virginiana	223
acer_rubrum	217
broussonettia_papyrifera	214
prunus_sargentii	209
ptelea_trifoliata	193
ulmus_pumila	189
abies_concolor	176
asimina_triloba	174
...	
aesculus_flava	68
amelanchier_arborea	68
pinus_thunbergii	67
acer_griseum	64
ulmus_procera	58
cedrus_deodara	58
ailanthus_altissima	58
crataegus_crus-galli	54
evodia_daniellii	53
juniperus_virginiana	51
Name: label, Length: 176, dtype: int64	

Figure 5 Training Dataset Class Distribution

III Related Work

Since the dataset came from a kaggle competition, a lot of related work was available

(Classify Leaves, 2021).

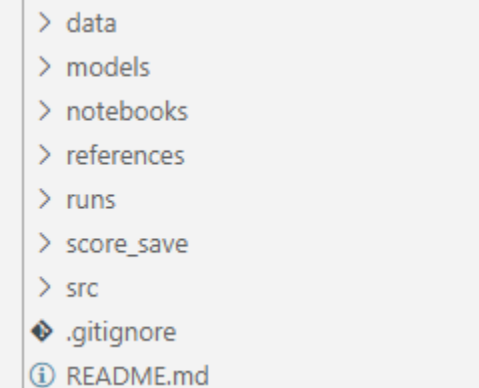
For example, fifth place submission achieved an astronomical accuracy score of 0.98704. It used resnet50 and seresnext50 architecture for training. As for image augmentation, the publisher utilized horizontal flip, vertical flip, rotate, random brightness contrast, shift scale rotate and normalization **(tf96, 2021).**

Another example is from seventh place finisher (**Charlesyyun, 2021**), who performed CutMix augmentation and test time augmentation to achieve this high score. Cutmix is a replacement strategy for cutout which randomly cut a square of the image. Cutmix replaced the removed pixels with patch from another image. The label is also mixed into the image with the proportion of the pixels size. All top scores utilized normalization with mean (0.485, 0.456, 0.406) and standard deviation (0.229, 0.224, 0.225). This mean and standard deviation are that of the ImageNet. Since ImageNet is so huge, this normalization could help accelerate training process.

IV Project Flow and file Structure

As briefly introduced in the beginning, the new powerhouse in the image classification is a set of architecture called transformers. First our team explored the base CNN model. Resnet18 being a famous introductory complex CNN model is also explored, and the main object of the project - the transformer model - was tested and evaluated.

Base code templates are provided by Professor Amir which utilizes pytorch framework.



```
> data
> models
> notebooks
> references
> runs
> score_save
> src
. .gitignore
i README.md
```

Our file hierarchy is again displayed above. Runs folder is for tensorboard loading while score_save recorded model performances. Source folder being the main folder includes all our training files.

The backbone file was rewritten removing global variables. Then these variables were moved into the utility.py file. And inside the training file, I simply initialized the Utility class and get these instance attributes since I decorated the class with the dataclass function from dataclasses module. The main benefit is making the workflow more intuitive. Additionally, it is much easier to navigate changing these hyperparameters and it is less error prone. Later on, our team create utility classes for different models to easily tune hyperparameters separately for different models. Hyperparameters to tune are number of epochs, batch sizes, learning rate, image channels, image sizes. To encoding leaf labels, for multiclass problem, label encoder should be used. The generated label column is an integer ranger from 0 to 175.

Training and validation set are split from the raw training set using stratified method to insure proper class representation as a result of large class numbers.

Our dataset was built on top of pytorch util package data Dataset abstract class. For the main file, we did not implement these augmentations because our goal was to compare across the architectures not to aim for the highest score. Another reason is some of the transformations cannot be applied across the board for violating the base of some of the network structures.

However, we did test if some of these arguments would bring higher scores.

For example, resnet batch size 100, learning rate 3×10^{-4} , 30 epochs builds an extremely strong base line showing below.

```
Validation acc 0.95859 Validation f1_macro 0.95877 Validation coh 0.95832 Validation sum 2.87569
```

Upon testing, batch size works best around 120, and the highest batch size allowed under memory is around 320. Higher batch size converges much quick, but the top score is significantly lower. Our understanding is that large batch sizes eliminated the purpose of shuffling our data, resulting in overfitting problem.

To make the template work for multiclass problem, the following encoding inside dataclass `__get__` item function should be removed because we don't want our labels to be hot encoded. Otherwise, we couldn't load the data and label in batches from dataloader. And the error message would be dataloader is not scriptable.

```
labels_mc:np.ndarray = np.zeros(self.OUTPUTS_a)
for _idx, _label in enumerate(range(self.OUTPUTS_a)):
    if _label == y:
        labels_mc[_idx] = 1
y_tensor = torch.FloatTensor(labels_mc)
```

Other parts altered are changing loss function cross entropy and like below, argmax was added to the label results.

Additionally, we changed the valuation metrics to f1_macro and cohen's kappa on top of the accuracy score and the best model is saved on highest summation score instead of default accuracy to get a more balanced model.

```
pred_logits = np.vstack((pred_logits, np.argmax(output.detach().cpu().numpy())))
real_labels = np.vstack((real_labels, np.argmax(xtarget.cpu().numpy())))

pred_labels = pred_logits[1:]
```

```
list_of_metrics = ["acc", 'f1_macro', 'coh']  
list_of_agg = ['sum']
```

Augmentation Results

The base line best results are as follows.

```
Validation acc 0.95859 Validation f1_macro 0.95877 Validation coh 0.95832 Validation sum 2.87569
```

When increasing image size to larger than the original one to 320 from 224, the results improved by almost 1 percent, which is a significant jump when margin of increase is obnoxiously low.

```
Validation acc 0.96295 Validation f1_macro 0.96373 Validation coh 0.96271 Validation sum 2.88939
```

Resnet advancement resnext50 is also tested. Because of the memory restraint, batch has to be lowered to highest allowed 55. The results are below. This proves the deeper the architecture goes, the better the performance.

```
Validation acc 0.96949 Validation f1_macro 0.97130 Validation coh 0.96929 Validation sum 2.91009
```

Finally, despite of not using transform across the board, I still test if results improve. I added RandomHorizontalFlip with 50% and RandomRotation of 45 degrees. To no surprise, the results improved tremendously from 0.95877.

```
Validation acc 0.96595 Validation f1_macro 0.96680 Validation coh 0.96573 Validation sum 2.89848
```

V Modeling

1. CNN baseline Model

We built a Convolutional Neural Network as a baseline for our modeling whose architecture is described in Table 1.

We also visualized the filters and feature maps to better understand how the CNN model works in Sect. 1.a.

Our CNN baseline model has only 4 Conv2d layers and to test the effects of deeper networks, we also run the VGG-16 pretrained model for comparison which will be described in Sec 1.b.

Layer	Output Shape
	[3, 224, 224]
Conv2d(3, 16, kernel_size=(7, 7), stride=(2, 2))	[16, 109, 109]
ReLU	[16, 109, 109]
BatchNorm2d	[16, 109, 109]
MaxPool2d(kernel_size=(3, 3), stride=2)	[16, 54, 54]
Conv2d(16, 64, kernel_size=(5, 5), stride=(2, 2))	[64, 25, 25]
ReLU	[64, 25, 25]
BatchNorm2d	[64, 25, 25]
MaxPool2d(kernel_size=(3, 3), stride=2)	[64, 12, 12]
Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))	[128, 10, 10]
ReLU	[128, 10, 10]
BatchNorm2d	[128, 10, 10]
Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))	[256, 8, 8]
ReLU	[256, 8, 8]
AdaptiveAvgPool2d	[256, 1, 1]
Linear	[4096]
ReLU	[4096]
Linear	[176]
BatchNorm1d	[176]

Table 1 The CNN baseline model architecture and the Output size of each layer

a. Visualization of Filters and Feature maps

Filters are the learned weights of the kernels that are used to convolve with the feature maps whereas the feature map, also called Activation Map, is obtained with the convolution operation, applied to the input data or the feature map from the previous layer using the filter/kernel.

We plot the first layer filter for the trained CNN baseline model in Fig. 7 which has the size of [3, 16, 7, 7]. The first dimension of size 3 is represented by RGB colors shown in the figure. The second layer of filters are shown in Fig. 8 which we only plot the first filter of 16 filters with the size of [16, 64, 5, 5] since the size 16 in the first dimension can not be present in any color map like what we do for the first layer filters. We don't plot the rest 2 layers of filters in this main report but they will be present in the individual report.

We also plot some feature maps of the raw image of a maclura pomifera type leaf shown in Fig. 9. Fig. 10 and Fig. 11 show the feature maps after the Conv2d and ReLU/BatchNorm/MaxPool in the first convolutional layer respectively whereas Fig 12 shows the feature maps after the second Conv2d.

The flowchart of how the first two convolutional layers work with the filters and feature maps is shown in Fig. 13.

We don't plot all other filters and feature maps in the last two layers as the channel numbers grow up in deeper neural layers and it becomes more difficult to make the plots in a meaningful way and to comprehend. This is always a dilemma in deep learning that as the model becomes more complex, the performance may get better whereas the interpretability becomes less comprehensive.

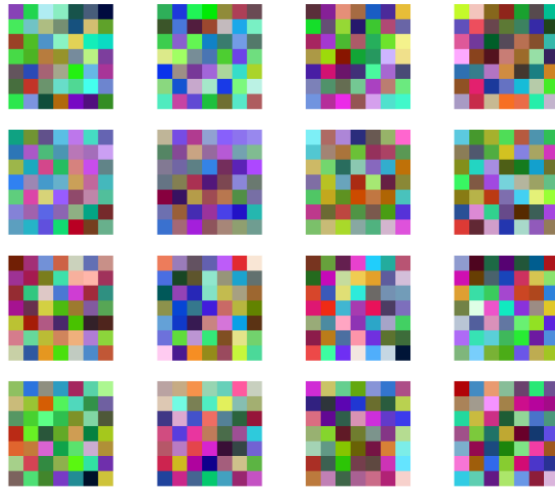


Figure 7 The filters of the first layer kernels with size of $[3, 16, 7, 7]$

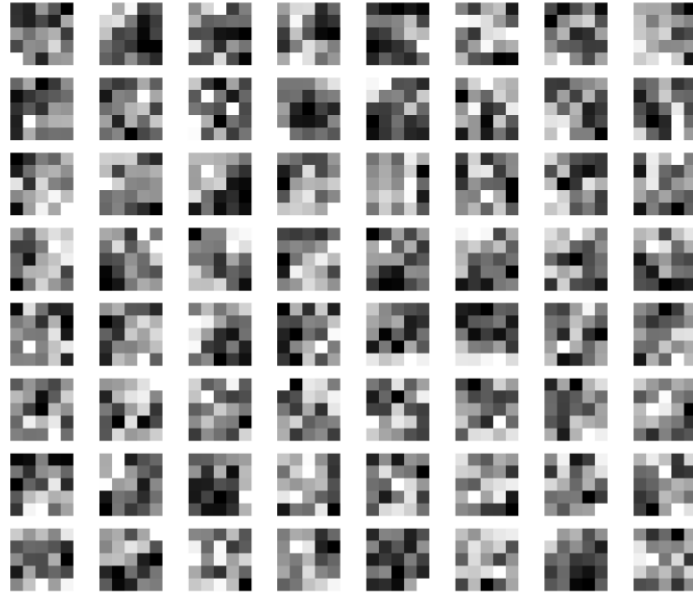


Figure 8 The first of the 16 filters of the second layer kernels with size of [16, 64, 5, 5]



Figure 9 The raw image of a maclura pomifera type leaf used for feature maps

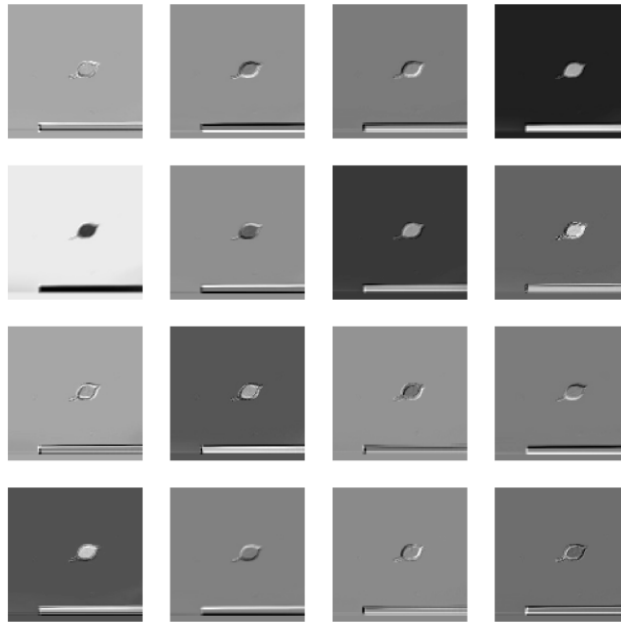


Figure 10 The feature maps after Conv2d in the first convolutional layer

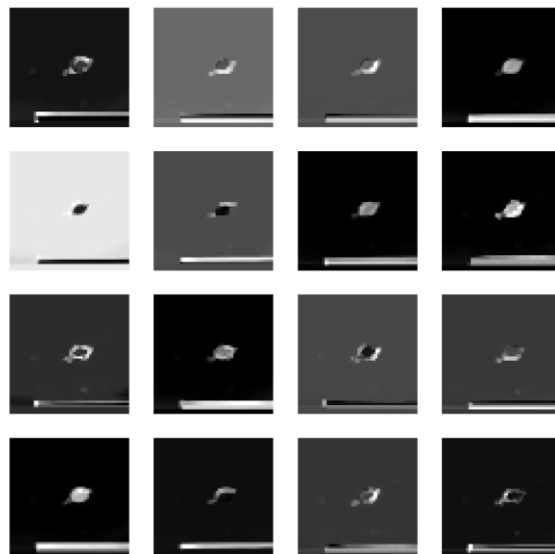


Figure 11 The feature maps after ReLU, BatchNorm and MaxPool in the first convolutional layer

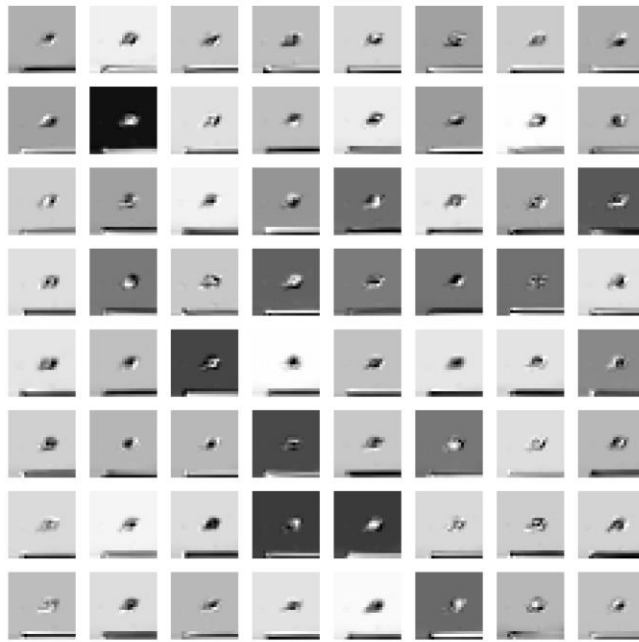


Figure 12 The feature maps after Conv2d in the second convolutional layer

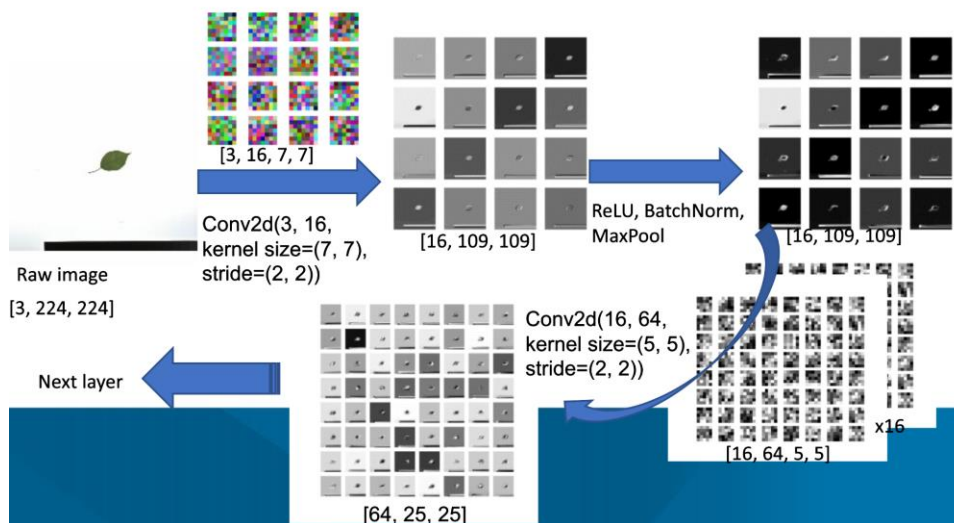


Figure 13 the flowchart of how the first two convolutional layers operate with the filters and feature maps.

b. A very deep CNN model: VGG16

To take another step further to test a deeper CNN model, we used VGG16 (Very Deep Convolutional Networks For Large-Scale Image Recognition, Simonyan, K., & Zisserman, A. 2014) on our dataset and compared its performance with our CNN baseline model. The VGG-16 model has a total number of 13 convolutional layers and 3 fully connected layers where the architecture is shown in Table 2.

Layer	Output Shape
Conv2d(3, 64)	[64, 224, 224]
ReLU	[64, 224, 224]
Conv2d(64, 64)	[64, 224, 224]
ReLU	[64, 224, 224]
MaxPool2d	[64, 112, 112]
Conv2d(64, 128)	[128, 112, 112]
ReLU	[128, 112, 112]
Conv2d(128, 128)	[128, 112, 112]
ReLU	[128, 112, 112]
MaxPool2d	[128, 56, 56]
Conv2d(128, 256)	[256, 56, 56]
ReLU	[256, 56, 56]
Conv2d(256, 256)	[256, 56, 56]
ReLU	[256, 56, 56]
Conv2d(256, 256)	[256, 56, 56]
ReLU	[256, 56, 56]
MaxPool2d	[256, 28, 28]
Conv2d(256, 512)	[512, 28, 28]
ReLU	[512, 28, 28]
Conv2d(512, 512)	[512, 28, 28]
ReLU	[512, 28, 28]
Conv2d(512, 512)	[512, 28, 28]
ReLU	[512, 28, 28]
MaxPool2d	[512, 14, 14]
Conv2d(512, 512)	[512, 14, 14]
ReLU	[512, 14, 14]
Conv2d(512, 512)	[512, 14, 14]
ReLU	[512, 14, 14]
Conv2d(512, 512)	[512, 14, 14]
ReLU	[512, 14, 14]
MaxPool2d	[512, 7, 7]
AdaptiveAvgPool2d	[512, 7, 7]
Linear	[4096]
ReLU	[4096]
Dropout	[4096]
Linear	[176]

Table 2 The VGG-16 model architecture and the Output size of each layer

The comparison of performance of the baseline model and the VGG-16 model is shown in Table 3. We see that VGG-16 doesn't perform better than the CNN baseline model while the depth and the complexity of the VGG-16 model is much higher. This may relate to the degradation problem that even for a deeper network starts to converge, the accuracy gets saturated and degrades as the network depth keeps increasing. This leads to our consideration of the ResNet models which will be introduced in the next section.

Models	Epochs	Batch size	Learning rate	accuracy	f1_macro	coh	sum
CNN-baseline	20	50	3e-4(mode='min', factor=0.5, patience=5)	0.846	0.840	0.845	2.530
VGG-16	20	50	3e-04(mode='min', factor=0.5, patience=5)	0.826	0.822	0.825	2.472

Table 3 The comparison of the performance between CNN baseline model and VGG-16 model

2. Resnet (Deep Residual Learning for Image Recognition)

Recent works have shown that image classification tasks can benefit from very deep models. However, the vanishing/exploding gradients problem may come with a deeper network. This problem has been largely mitigated by the normalization initialization and intermediate normalization layers, which enable networks with tens of layers to start converging with stochastic gradient descent (SGD). Another problem, called the degradation problem, has been exposed, as deeper networks start to converge, where the accuracy gets saturated and then degrades rapidly as the network depth keeps increasing.

The ResNet model was proposed to address the degradation problem by introducing a deep residual learning framework. The building block of the residual learning can be realized by feedforward neural networks with "shortcut connections" (see Fig. 9). In ResNet models, the shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layers. Identity shortcut connections add neither extra parameter nor computational complexity.

There are a number of Resnet variants with different numbers of layers. And here we plot the model architecture of one of them – the ResNet18 model in Figure 10 which performs the best among all the models that we have tried in this project.

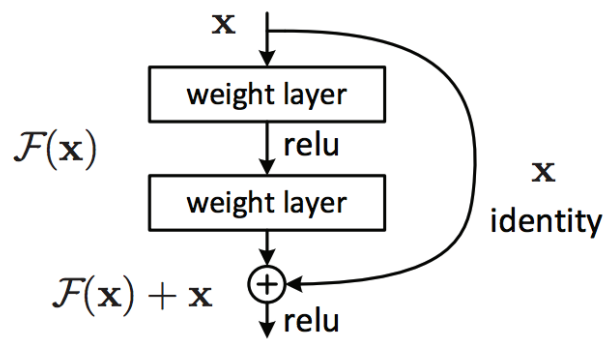


Figure 9 The building block of Residual learning

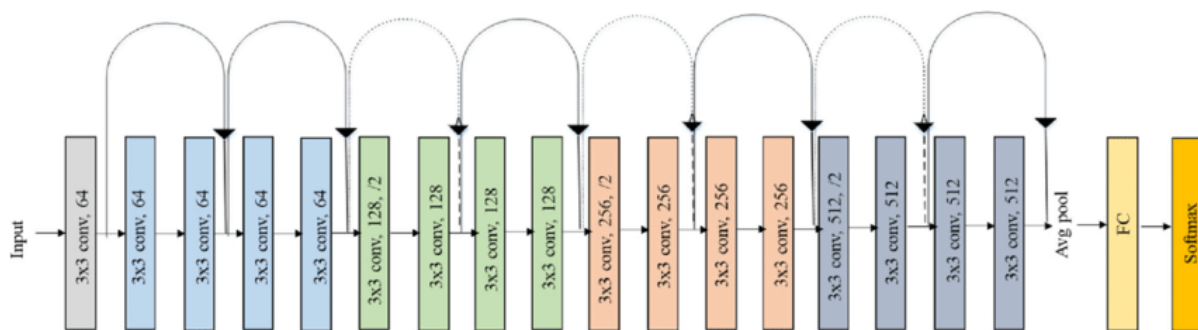


Figure 10. The architecture of the Resnet18 model. from He et al., 2015

3. Vision Transformer(Vit)

In the previous work, we have used the basic CNN model and Resnet model to do this classification work. However, a brand-new framework named “Transformer” is created and developed overwhelmingly among the NLP area. After that, some researchers are trying to use the model to do some CV tasks with as few changes as possible. The model we used in the project is named “Vision Transformer”, which is really similar to the Bert model (one of the latest Transformer models in the NLP area).

a. Main idea of Vision Transformer model – Attention Mechanism

In fact, the Vit model uses attention mechanisms to capture the characteristics of an image.

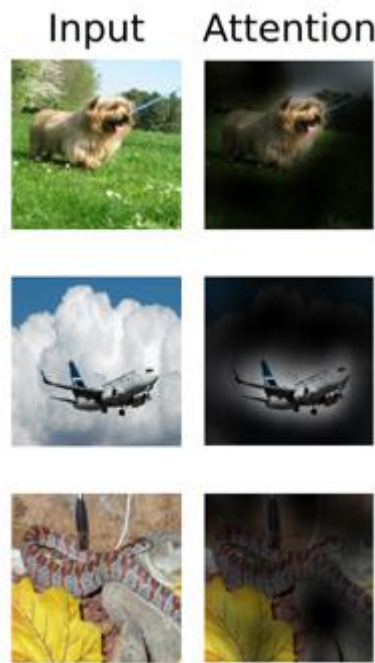


Figure 1: Representative examples of attention from the output token to the input space

Basically, attention is weight for the pixel of the image in the CV area, which is similar to the attention in the NLP area. But the difficulty for using attention is that we can't directly change the image from a high dimension matrix into a vector for computing, because there is a limit for the memory (RAM) of computers now. So, some researchers divided the image into patches, and these patches can be seen as words in NLP, which overcome the computing limits problems. ^[1]

b. Structure/architecture of transformer

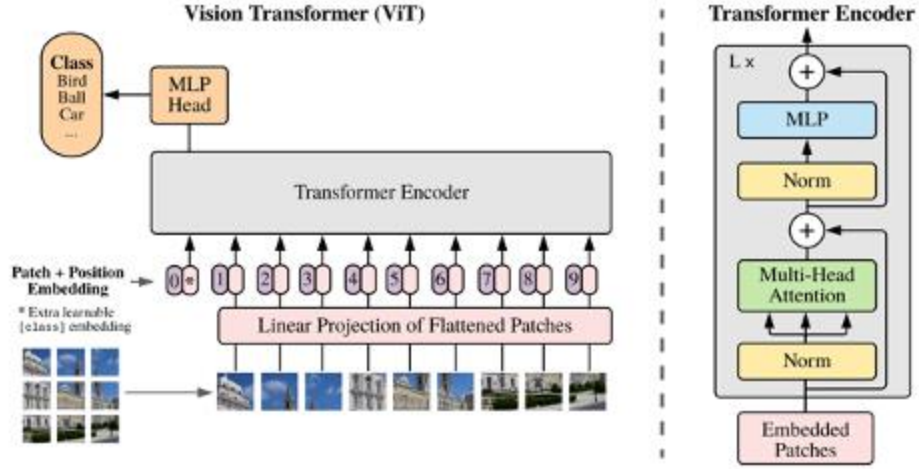


Figure 2: Model overview

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$

We split an image into fixed-size patches (for this model is 16x16 size), linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. (Which makes the model as much as similar to the transformer model we used in NLP area)

c. How we train the model

i. Input size:

We use the 224x224 size as the input size, which is also the original size of our data. Since the patch size is fixed to 16x16, when we implement the position embedding process, the image must keep the same input size as the pretrained model, otherwise the position information of the pretrained model will be affected.

ii. Training based on pretrained model:

Like using a pre-trained model to train a Resnet model, we only unfreeze the final layer of the ViT model (MLP Head layers). In this part, we passed the final outputs of the Transformer Encoder layer to a fully connected layer with 512 neural, and use ReLu function as the activation function. After that, we did a dropout process at 0.3 level to improve the model’s robustness. Finally, we passed the vectors to a fully connected layer, whose neural number is the number of the classification categories of our data set (176 kinds of tree leaves).

```

elif model_name == 'ViT':
    # model = vit_base_patch16_224_in21k(num_classes=OUTPUTS_a, has_logits=False).to(device)
    model = torch.hub.load('facebookresearch/deit:main', 'deit_tiny_patch16_224', pretrained=True)
    for param in model.parameters(): #freeze model
        param.requires_grad = False

    n_inputs = model.head.in_features
    model.head = nn.Sequential(
        nn.Linear(n_inputs, 512),
        nn.ReLU(),
        nn.Dropout(0.3),
        nn.Linear(512, OUTPUTS_a)
    )

```

Figure 3: The Vision Transformer definition part of our code

d. Why we didn't perform better than Resnet model

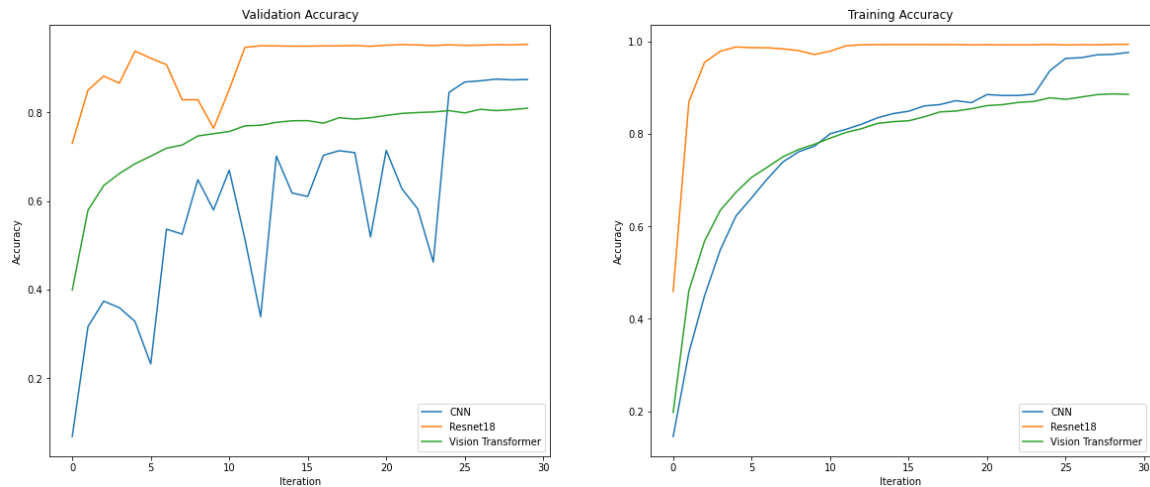


Figure 5: The accuracy of models in the tree leaves data set classification task

From figure 5, we can easily find that the Vision transformer model didn't perform better than Resnet, even worse than the CNN model. However, the latest ranking of models in the image classification area shows that 8 of the top 10 models are using transformer architecture. So why did this happen?

If we ignore the hyperparameter tuning temporally (our model results had converged), the main reason Vit model didn't achieve a better score is that the data set size is too small to offer enough data to train the model. As the paper of Vit model said, Transformers lack some of the inductive biases inherent to CNNs, such as translation **equivariance and locality**, and therefore do not generalize well when trained on insufficient amounts of data. If we can get more data, we may train a better Vit model.

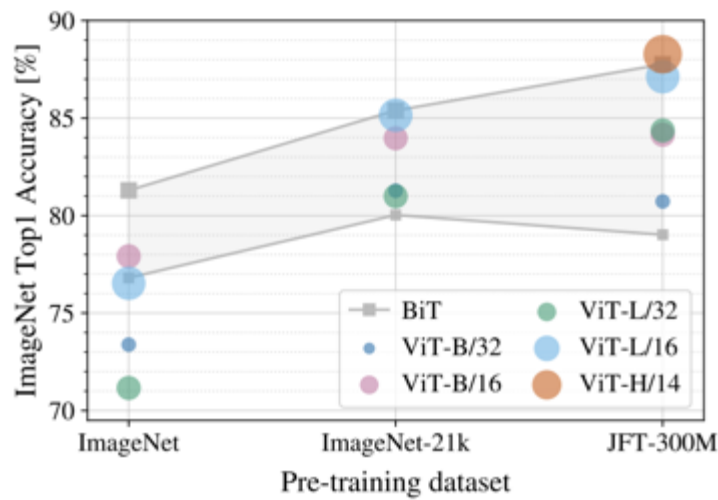


Figure 5: Transfer to ImageNet.

According to the paper of Vision Transformer, the authors point out that, all the ViT models performed worse than BiT ResNets (shaded area) on a relatively small size data(ImageNet), but with the data set size getting bigger, ViT models gradually show a comparable score. What's more, ViT models even perform better than ResNet if the data set size achieves a very large scalar, for example, JFT-300M.

VI Results

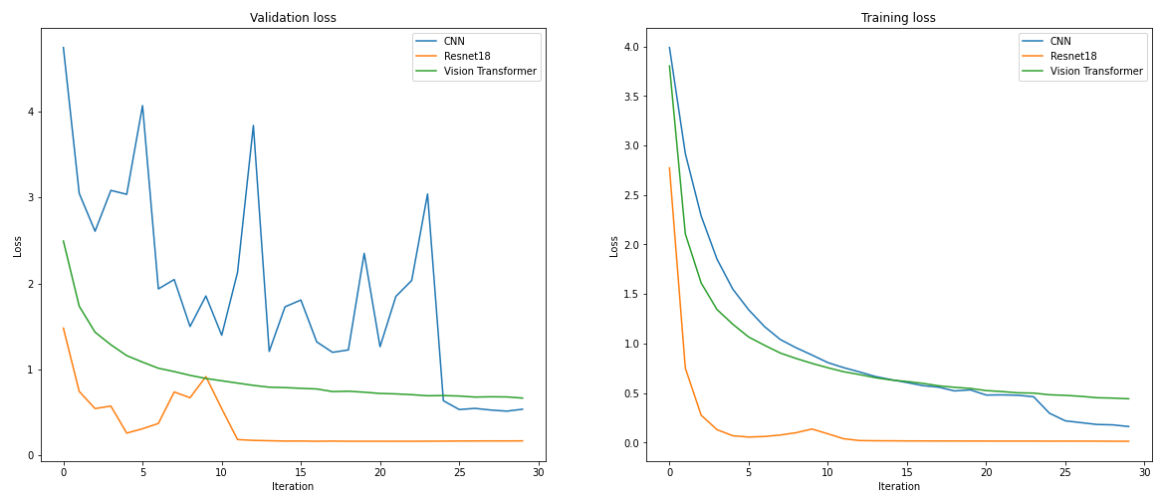


Figure : Loss Comparison

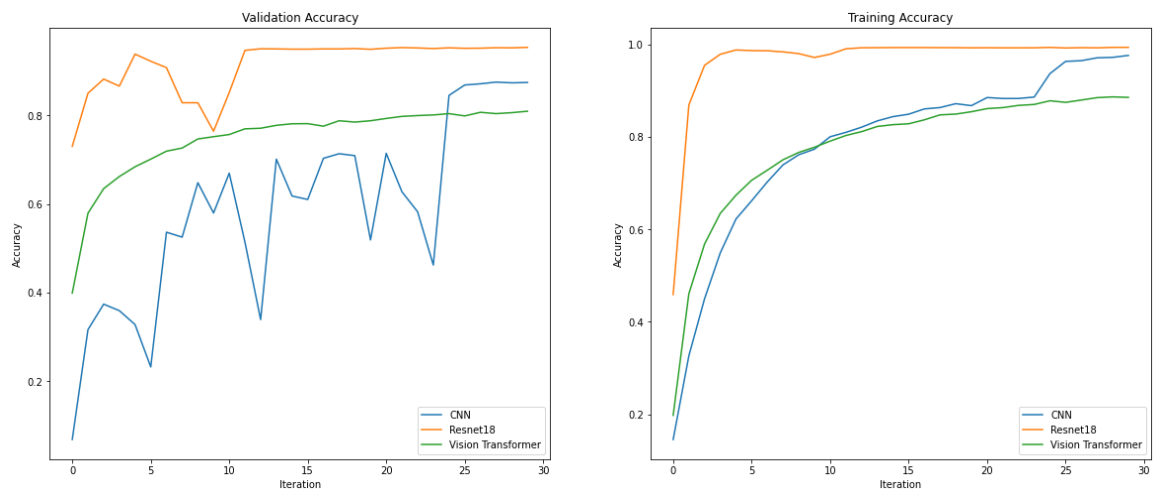


Figure : Accuracy Comparison

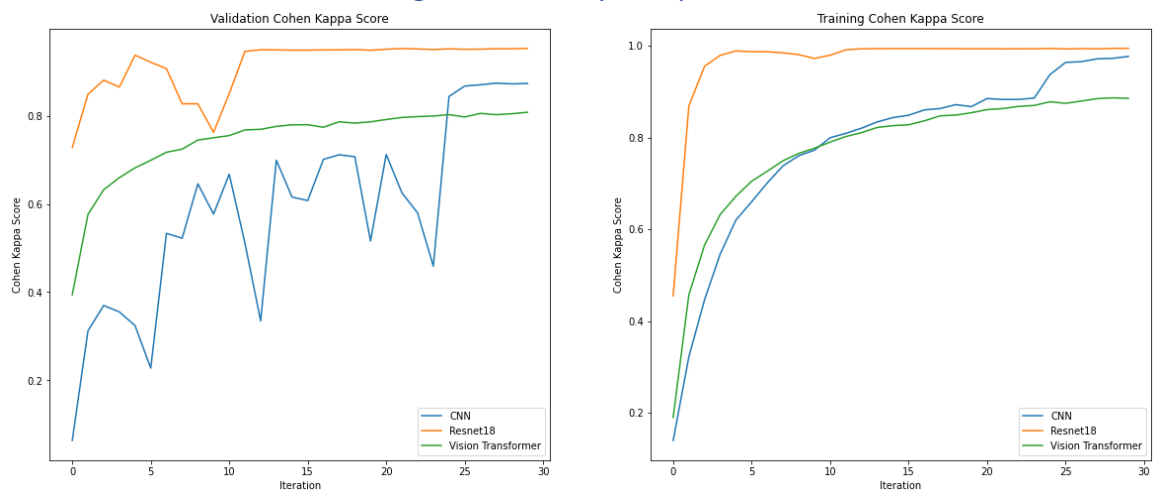


Figure : Cohen Kappa Comparison

As can be inferred from the loss graph, all three converged eventually. However, the progress evidently differed, with CNN displaying a much more significant volatile curve than the other two.

At a first glance, one might feel hard to interpret which metrics corresponded to which graph. However, it further supported the superior performance of Resnet18 over the other two, on account of the three metrics supplementing each other without the sole determining factor being the accuracy.

In summary, out of the three architectures we tested, resnet 18 scored the highest accuracy, highest f1 macro score and cohen kappa score with CNN being the runner up.

In spite of the performance edge transformers have in image classification task over resnet, the best of CNN architecture, in our dataset, the score of the resnet beat that of vision transformer model. The reason being that transformers lack some of the inductive biases that are inherent to CNNs, such as translation equivariance and locality. Thus, they do not generalize well when trained on insufficient amounts of data. If we can get more data, we may train a better ViT model.

References

References

- [1] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.
 - [2] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv e-prints*, Article arXiv:1512.03385, arXiv:1512.03385.
 - [3] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv e-prints*, Article arXiv:1409.1556, arXiv:1409.1556.
- IBM Cloud Education. (2020, 10 20). *Convolutional Neural Networks*. Retrieved from ibm: <https://www.ibm.com/cloud/learn/convolutional-neural-networks>

- Bain, A. (1873). *Mind and Body: The Theories of Their Relation*. New York: D. Appleton and Company.
- Charlesyyun. (2021, 6 28). *7th: ResNeSt+ResNeXt+DenseNet (0.98840)*. Retrieved from kaggle: <https://www.kaggle.com/code/charlesyyun/7th-resnest-resnext-densenet-0-98840>
- Classify Leaves*. (2021, 5). Retrieved from kaggle: <https://www.kaggle.com/competitions/classify-leaves/data>
- Classify Leaves*. (2021, 6 26). Retrieved from kaggle: <https://www.kaggle.com/competitions/classify-leaves/code?competitionId=29193&sortBy=scoreDescending>
- Classify Leaves*. (2021, 6 28). Retrieved from kaggle: <https://www.kaggle.com/competitions/classify-leaves/data>
- Convolution*. (2022, 4 11). Retrieved from Wikipedia: <https://en.wikipedia.org/wiki/Convolution>
- CS231n Convolutional Neural Networks for Visual Recognition*. (2022, 4 19). Retrieved from github.io: <https://cs231n.github.io/convolutional-networks/>
- CUDA*. (n.d.). Retrieved from Wikipedia: <https://en.wikipedia.org/wiki/CUDA>
- Fukushima, K. (1980). Neocognitron: A Self-organizing Neural Network Model. *Biological Cybernetics*, 193-202.
- Hubel, D. H., & Wiesel, T. N. (1959). Receptive Fields of Single Neurones in the cat's striate cortex. *The Journal of Physiology*, 574-591.
- Image Classification on ImageNet*. (n.d.). Retrieved from papers with code: <https://paperswithcode.com/sota/image-classification-on-imagenet>
- James, W. (1890). *The Principles of Psychology*. New York: H. Holt and Company.
- Stanford Vision Lab . (2012, 9 30). *Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)*. Retrieved from image net: <https://image-net.org/challenges/LSVRC/2012/results.html>
- tf96, w. (2021, 6 28). *[0.98909] 5th Place Solution*. Retrieved from kaggle: <https://www.kaggle.com/code/wangtf96/0-98909-5th-place-solution>
- West, M. (2019, 7 11). *Explaining Recurrent Neural Networks* . Retrieved from bouvet: <https://www.bouvet.no/bouvet-deler/explaining-recurrent-neural-networks>
- Williams, R. J., Hinton, G. E., & Rumelhart, D. E. (1986). Learning representations by back-propagating errors. *Nature*, 533-536.