

New Wave in Image Classification

Qizhen(Kurtis) Shen
George Washington University
lottlecoop@gwmail.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. Project Overview	3
II Individual Work	3
Dataset	8
Related Work	12
Augmentation	13
III Results	15
IV Summary	15
V Code Percentage	15
References	16

I. Project Overview

Our project is about new waves in image classification.

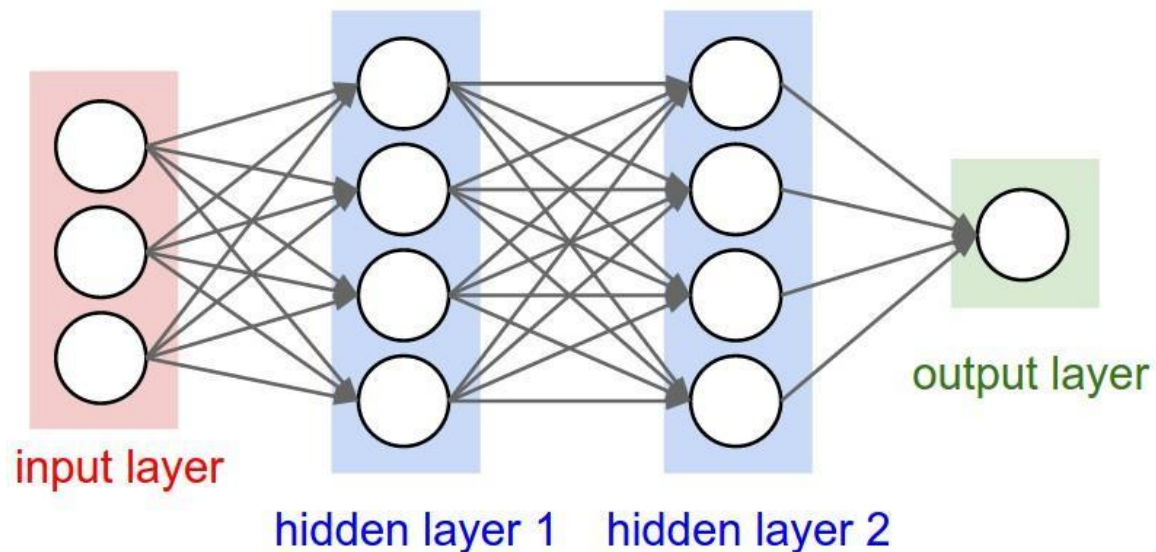
First, we give a comprehensive neural network history introduction. Explained all different fundamental deep learning architectures. Then we moved onto some latest developments in the field which is where our project gets inspiration from. We further detailed where the dataset is from, did some EDA, and how we processed the data. Later, we explored different models and drew our results and conclusions. At the end, we detailed some future work to research on.

Despite of clear tasks distribution, as me as the guider, constructing project idea, project flow, distributing tasks, constructed code blueprint that runs a baseline; Hui Yang working on CNN, VGG, and ResNet architecture explanation and model building and CNN visualization; and finally Changhong Zhang dived deep into Vision Transformer original paper, educated on the concepts and implemented in pytorch and visualize all our final results in tensorboard, a lot of our work are intertwined which I believe are the best form of a successful project group. Everyone contributed a huge amount and I think the evidence is obvious from our presentation.

II Individual Work

For my part, I did a deep research into the history of neural networks.

For example, I learned Neural network is inspired by biological neural network. Alexander Bain (**Bain, 1873**) and William James (**James, 1890**) first proposed the preliminary theoretical base independently. In my understanding, fully connected layers acted a magic pattern finder box.

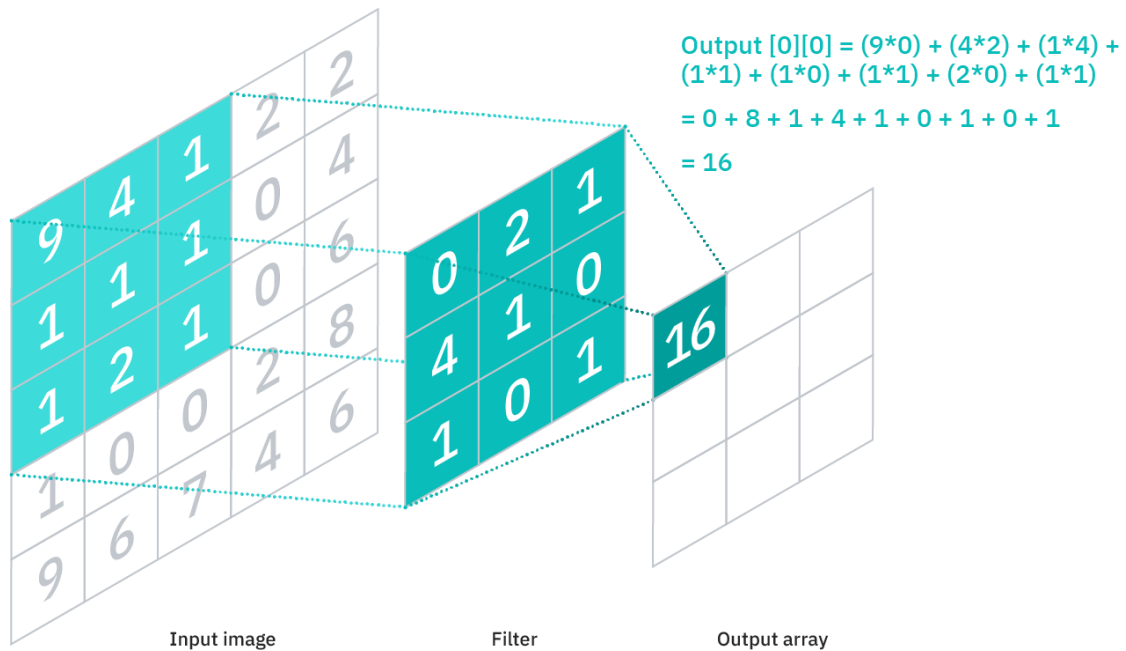


(CS231n Convolutional Neural Networks for Visual Recognition, 2022)

I also learned different architecture are created tailored to specific tasks for better performance.

CNN is created for image classification problem. It utilized different kernels doing dot product across the input space to extract different features and pooling layers to enhance the extraction meanwhile reduce sizes. CNN for me acted as a focus box in front of the pattern finder for better feature extraction and dimension reduction for fully connected layers. For example, use

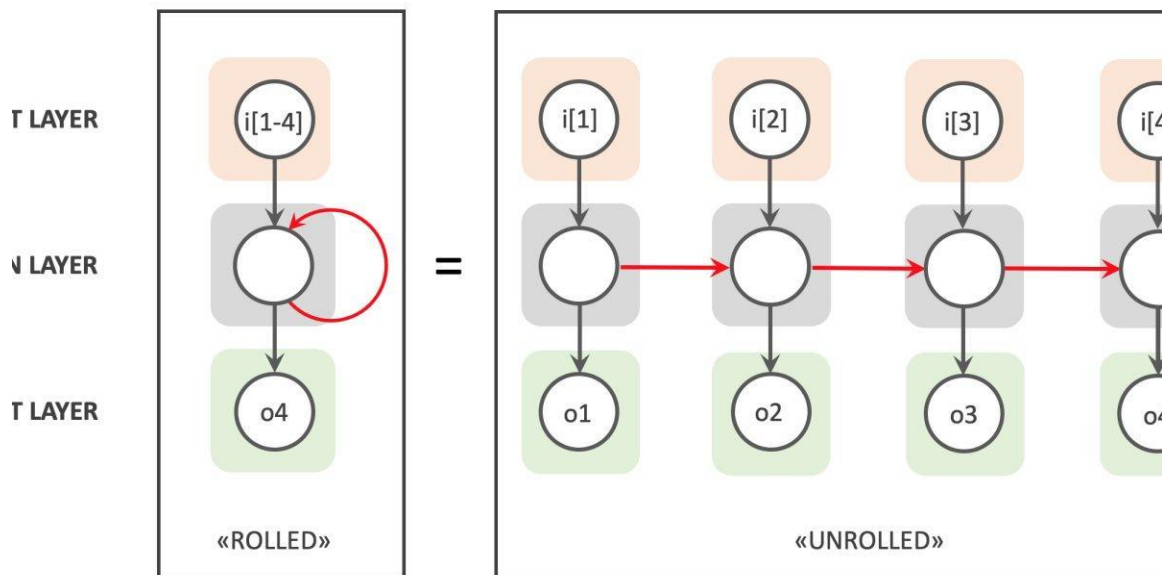
different kernels, it is very easy to separate letter “x” and letter “T”, since letter “x” consists of diagonal lines while letter “T” consists horizontal and vertical lines.



(IBM Cloud Education, 2020)

RNN is created for prediction problem. To me, it acts as a magic memory box inside the pattern finder box since previous state of the hidden layer can be passed onto the next one. For

example, for text autocompletion, hidden layer neuron can be how many steps since a certain alphabet appeared. After every update, the steps are stored inside the neuron.

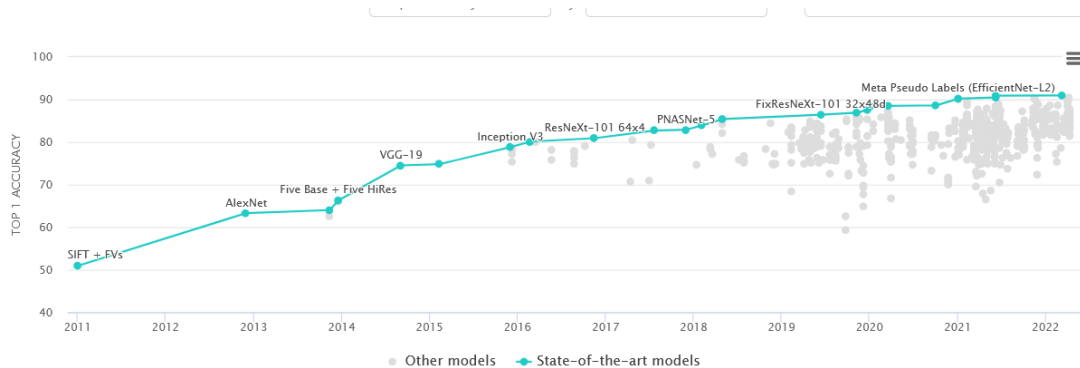


(West, 2019)

Through this project, I learned about those big-name architectures. GoogleNet, VGG, ResNet, EfficientNet, you name it.

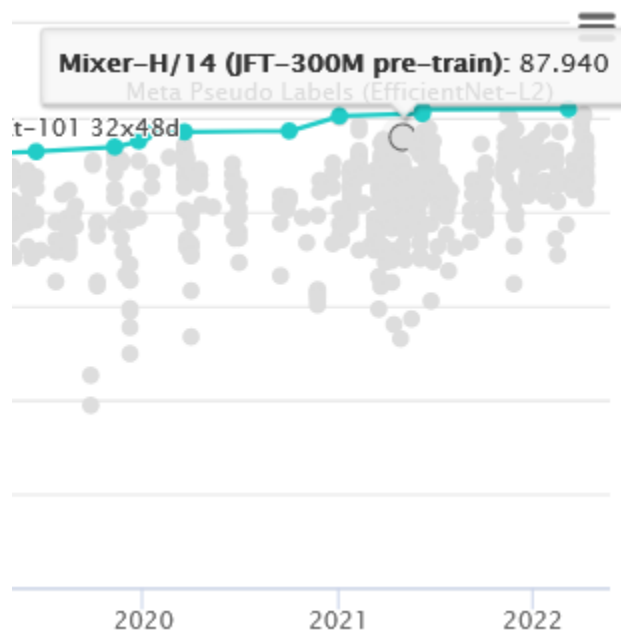
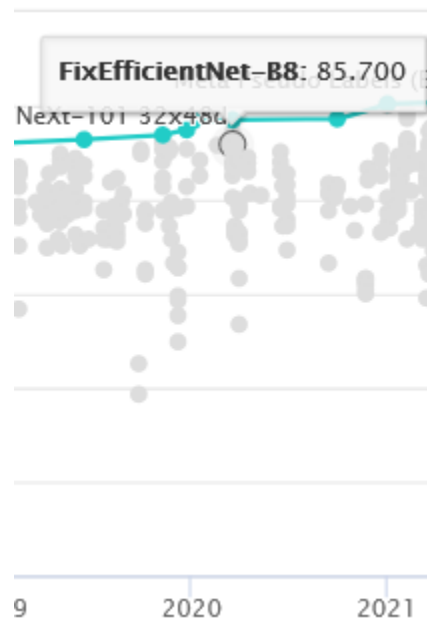
I learned that these architectures all competed based on ImageNet, which has been used as the industry standard testing bed for comparing deep learning architectures.

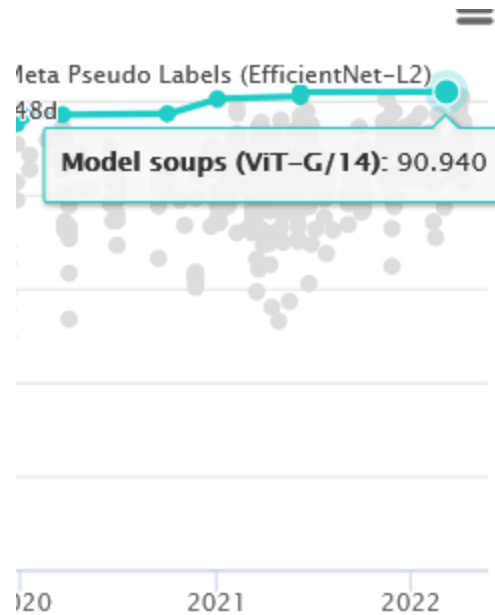
The accuracy progress graph over the years is displayed below (Image Classification on ImageNet, n.d.).



And exploring recent performance results were where the whole project idea is from.

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.





Even though transformers modeling part was my teammate's responsibility, with my own research and my teammates fantastic explanation, I think I got a grasp of at least how base transformers work.

For me, transformers is a data converter consisting of CNN which is focus and RNN which is memory. It also consisted linear transform.





In the author's term, the linear transform is called value matrix since transformation is a weight matrix. Another two matrix, query matrix and key matrix, which are both used for focus and memory, are combined into an attention matrix. And finally, the attention matrix is combined with the value matrix to get the output.

[Dataset](#)

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

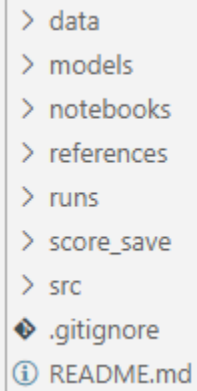


Images are split into 18353 training ones and 8800 testing ones, with each class having at least 50 images. Upon quick scan, each image is 224 by 224, and contains precisely one leaf, sitting perfectly inside the frame which could explain for the impressive results (**Classify Leaves, 2021**)

- ▶  images
-  [sample_submission.csv](#)
-  test.csv
-  train.csv

(Classify Leaves, 2021)

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



A screenshot of a file explorer window showing a project directory. The directory contains several subfolders and two files. The subfolders are: data, models, notebooks, references, runs, score_save, and src. The files are: .gitignore and README.md. The .gitignore file is marked with a diamond icon, and the README.md file is marked with an 'i' icon.

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

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

Related Work

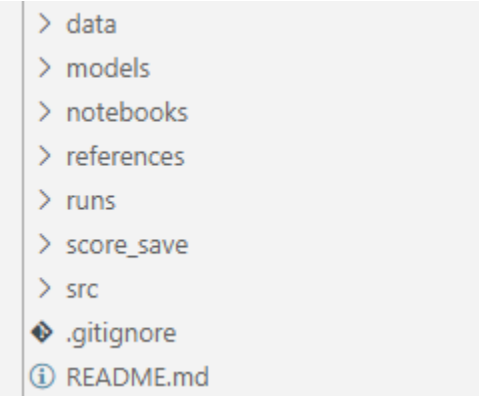
Since the dataset came from a kaggle competition, a lot of related work was available (**Classify Leaves, 2021**).

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.

Augmentation

Base code templates are provided by Professor Amir which utilizes pytorch framework. Now I will rerefer to the structure.



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

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. 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, I did test if some of these augmentations would bring higher scores.

For example, resnet batch size 100, learning rate 3×10^{-4} , 30 epochs.

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, I added argmax to the label results. This particular bug took forever to diagnose since after deleting the threshold 1 and 0 encoding, the results can be passed into accuracy metrics if the label encoding part was not fixed. However, the results were extremely low, being around 0.05. Later on, I figured the reason being, without argmax, each image has 176 probabilities. Only when probabilities are matched exactly, will the accuracy metrics classify as correct.

Additionally, I 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']
```

Other parts I explored are dataloader parameter sampler. There is an up-to-date class Imbalanced Sampler. When combining with image oversampling methods, it could fix the weight imbalances between classes.

III 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
```

I also tested resnet advancement resnext50. 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
```

IV Summary

From my experiment, resnet with only 18 layers performed exceptionally well. Deepen the layers gives us the largest overall score bump. Somethings I would like to try is implementing the cutmix augmentation since the fifth-place submitter did not augment this but still beat those who used. In addition, architecture of vision transformer is quite complicated. To fully grasp, it helped me remembered linear algebra much deep since transformer in essence are three matrices. In addition, being the better programmer in the group helping successfully alter and deploy vision transformer script revised myself some of the programming nuance.

V Code Percentage

As been mentioned, the template is from Professor Amir. However, it was completely hauled and renovated. Since my part is run the base line, there is a limit of things I don't need to change. However, I also participated extensively in help debugging the vision transformer modeling part, like the static method function of the dataset, and called by the dataloader

collateral function. The weight decay parameters is taken directly from kaggle competition since top runners all used this factor.

References

- 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.