

Project Report

# Multi-class Image Classification Using Deep Convolutional Neural Networks

---

Pulkit Saxena  
([ps6104@rit.edu](mailto:ps6104@rit.edu))

Kritya Shree Sivasakthi  
([ks7612@rit.edu](mailto:ks7612@rit.edu))

Lokesh Kumar  
Rudhramurthi  
([lr4157@rit.edu](mailto:lr4157@rit.edu))

Eashan Joshi  
([ej2637@rit.edu](mailto:ej2637@rit.edu))

Xinyu Hu  
([xh1165@rit.edu](mailto:xh1165@rit.edu))

1<sup>st</sup> December, 2022



## Development of question / hypothesis

We initially wanted to use transfer learning using pre-trained image classification models, like Resnet50, but due to unavailability of GPU based servers as originally planned, we decided to further our understanding of deep convolutional neural networks by implementing our own model from scratch and comparing the accuracy of our model to the several other popular pre-trained models. We also decided to work on object detection along with multi-class image classification.



## Data research

The DARPA dataset was our original selection. We tried to use different methods to research the dataset. For example, what is the dataset about, what are the features, and what we can do with the dataset. In order to tackle the problem of dataset being too big, we tried using ijson library, pytorch, and other libraries to break down the large json files. So that we can understand the data. When we figured out the way to examine the data, we chose to exclude the dataset because it was mostly categorical and we lacked the necessary domain expertise. We also looked into the MNIST Fashion, COCO and CIFAR100 dataset, but they did not fit our criteria. Ultimately, we finalized the ImageNet dataset which was provided by the Professor but due to lack of resources we decided to work on a subset of the dataset called Tiny ImageNet.

Tiny-ImageNet is a subset of ImageNet. The dataset contains 100,000 images, 200 classes (500 image/class) and 64×64 colored images. The testing data are images without labels.

## Literature review

1. **Base Model:** Our base model is the best model for the dataset as it produced the best result. The model consists of 10- layers( 5-Convolutional layer, 4-MaxPool layer, 1-Fully Connected layer)
2. **VGG16** is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In ImageNet, the model obtains a 92.7% top-5 test accuracy. VGG16 is renowned for its ease of use since it stacks multiple 3×3 kernel-sized filters one after another instead of large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively).
3. **Resnet50** is a 50-layer convolutional neural network that was first introduced in the paper “Deep Residual Learning for Image Recognition”. It consists of 48 convolutional layers, one MaxPool layer, and one average pool layer. This was our choice because, although the model size was small, it is deeper than VGG-16.
4. **Xception** is a convolutional neural network that is 71 layers deep proposed by Chollet François. This architecture draws inspiration from Inception architecture (GoogLeNet). It has more efficient use of the model parameters and outperforms other models ResNet-50, ResNet-101 and ResNet-152.
5. **EfficientNetV2B3** is known to be one of the most effective models introduced in 2019. In comparison to the older version, EfficientNetV2 has a quicker training time and improved efficiency.



## Analysis strategy


We first attempted to download the Imagenet dataset which was around 500 GB. We investigated various methods for downloading the data and training the model in batches of 50 - 100 GB, but extracting data caused our PCs to crash. Therefore, we chose the subset of the data called tiny imagenet dataset which has around 1 GB of data. Using tensorflow and keras, we imported the dataset.

When we first ran our model we discovered that it reached its maximum accuracy after 10 to 12 epochs and then began depreciating. So we decided to use Early stopping - call backs, but after some processing, we found that the validation accuracy of the algorithm goes up again so we removed it. We got an accuracy of approximately 17% in our first run, next we analyzed ways to increase the accuracy of the base model.

First, we experimented by adding layers and neurons to increase the depth of the model. Then, we tested the model by experimenting with the existing layers, increasing the accuracy by 2 to 3% during each experiment. We finally reached an accuracy of 64.46% on our training set, 57.15% on our validation data and 13% on our test set after numerous subtle hyperparameter tuning and optimization.

As a means to further improve the efficiency of the model, we explored and attempted transfer learning since pre-trained models are known to have better accuracy, but this did not produce fruitful results. There are several pre-trained models with excellent performance and accuracy for image processing. Thus, we tried various models (EfficientNetV2B3, ResNet50, Xception and VGG-16) but our model gave better results compared to a few of the pre-trained models for the Tiny ImageNet dataset. Therefore, we tried freezing the layers which gave us better accuracy. We decided to execute the pre-trained models for 20 epochs considering the fact that the pre-trained models are fast and are trained to produce accurate results. When we noticed the fall in accuracy, we decided to stop the model which eventually landed at a round number of 20 epochs. The models Xception and VGG-16 gave better results than our model, hence we tried transfer learning by adding a few layers from those models to our base models. During transfer learning, each epoch had an ETA of 4 to 5 hours and without GPU; since our Colab GPU ran out, we decided to pivot and concentrate on our base model. After all the struggles and experiments, we achieved an accuracy of 57.15% and 80.08% for the validation data and 64.46% and 90.91% for the training data for our base model and VGG-16 respectively.

During this entire process, we hit several roadblocks where we had to pivot and find other ways to make our model work. Some of these included the RAM crashing quite frequently because of



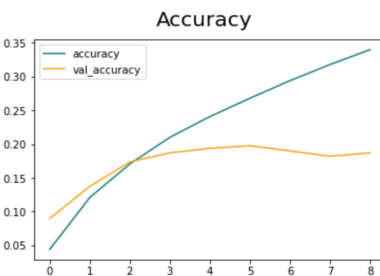
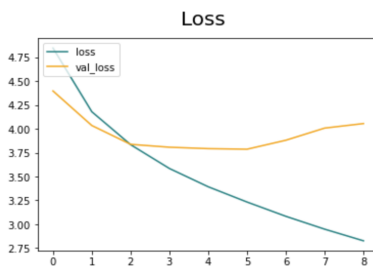
the size and finding that our images were imported incorrectly as the default size for ImageNet is 256 x 256 but Tiny ImageNet was 64 x 64. In addition, each epoch execution took 30 to 40 minutes. While researching on reducing the time taken for each epoch, we discovered that Google Colab had an option to switch to GPU. This made a significant difference because it reduced the duration from 30 mins to 3 mins, allowing us to experiment more with the model.

On the other hand, working with Colab caused more problems. After continuous usage Google Colab stopped working since it ran out of GPU and prompted us to work with the paid version of Colab. Hence, we switched to our local GPU and decided to use Pulkit's laptop as it had the M1 Pro chip. Even then, we encountered an expected error where the M1 does not support the tensorflow used by all systems (we had to download a special version of Anaconda and tensorflow for the machine). We also noted that our model worked better on a local server in comparison to Colab making it machine dependent.

## Analysis code

### Base Version

```
Model: "sequential"
Layer (type)                 Output Shape              Param #
-----
conv2d (Conv2D)              (None, 254, 254, 16)     448
max_pooling2d (MaxPooling2D) (None, 127, 127, 16)     0
conv2d_1 (Conv2D)            (None, 125, 125, 32)     4640
max_pooling2d_1 (MaxPooling2D) (None, 62, 62, 32)     0
conv2d_2 (Conv2D)            (None, 60, 60, 64)       18496
max_pooling2d_2 (MaxPooling2D) (None, 30, 30, 64)       0
conv2d_3 (Conv2D)            (None, 28, 28, 128)      73856
max_pooling2d_3 (MaxPooling2D) (None, 14, 14, 128)     0
conv2d_4 (Conv2D)            (None, 12, 12, 256)      295168
max_pooling2d_4 (MaxPooling2D) (None, 6, 6, 256)       0
flatten (Flatten)            (None, 9216)              0
dense (Dense)                (None, 256)               2359552
dense_1 (Dense)              (None, 200)               51400
Total params: 2,803,560
Trainable params: 2,803,560
Non-trainable params: 0
```

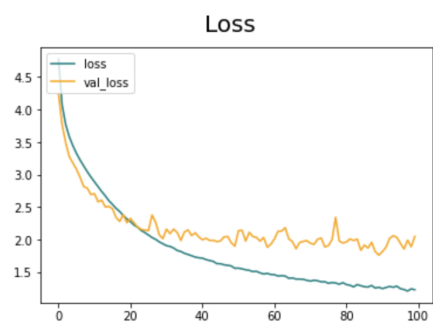
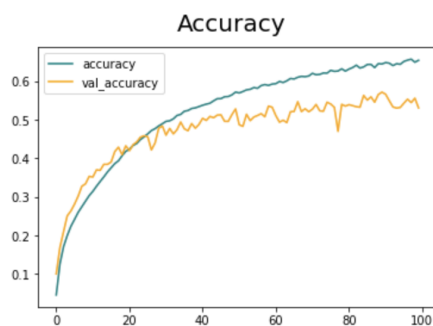


- Input Shape: 256 x 256 x 3
- Number of layers: 12 weighted layers and 1 output layer
- Model Layers:
  - Conv2D - filters: 16, 32, 64, 128, 256; Kernel size: (3,3)
  - MaxPooling
  - Flatten
  - Fully Connected (Dense)- Units 256
  - Output (Dense) - 200
- Optimizer: Adam
- Loss: SparseCategoricalCrossEntropy
- Epochs: 100
- Activation Function: ReLU,
- Output Activation: Softmax

## Final Version

Layer (type)	Output Shape	Param #
conv2d_47 (Conv2D)	(None, 62, 62, 64)	1792
max_pooling2d_45 (MaxPooling2D)	(None, 31, 31, 64)	0
conv2d_48 (Conv2D)	(None, 29, 29, 128)	73856
max_pooling2d_46 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_49 (Conv2D)	(None, 12, 12, 256)	295168
max_pooling2d_47 (MaxPooling2D)	(None, 6, 6, 256)	0
conv2d_50 (Conv2D)	(None, 4, 4, 128)	295040
max_pooling2d_48 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten_13 (Flatten)	(None, 512)	0
dense_26 (Dense)	(None, 64)	32832
dense_27 (Dense)	(None, 200)	13000

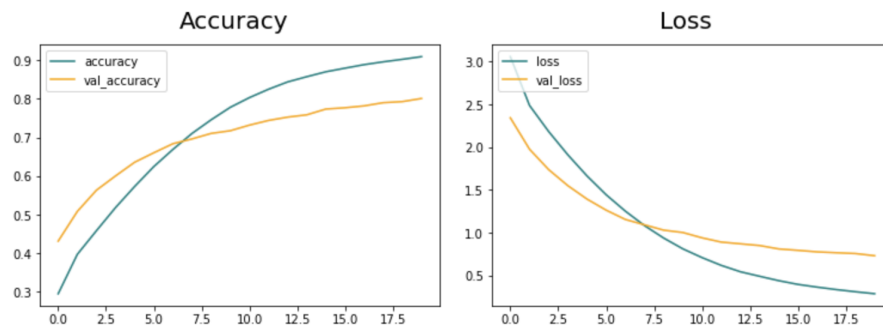
=====  
Total params: 711,688  
Trainable params: 711,688  
Non-trainable params: 0



- Input Shape: 64 x 64 x 3
- Number of layers: 10 weighted layers and 1 output layer
- Model Layers:
  - Conv2D - filters: 64, 128, 256, 128; Kernel size: (3,3)
  - MaxPooling
  - Flatten
  - Fully Connected (Dense)- Units 256
  - Output (Dense) - 200
- Optimizer: Adam
- Loss: SparseCategoricalCrossEntropy
- Epochs: 100
- Activation Function: ReLU
- Output Activation Function: softmax



## VGG-16

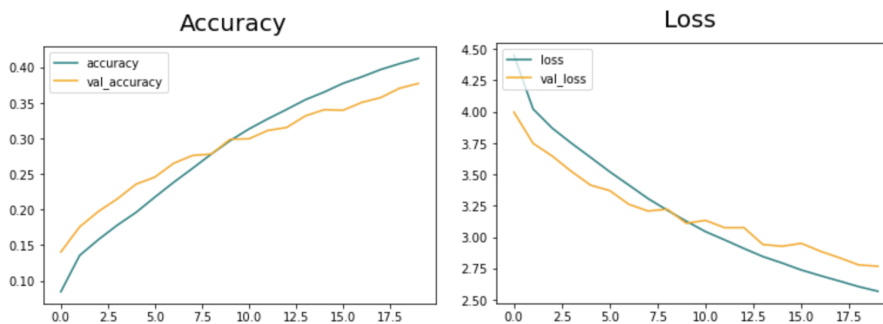


Accuracy and loss of VGG-16

Training Accuracy:90.91%

Validation Accuracy:80.08%

## Xception

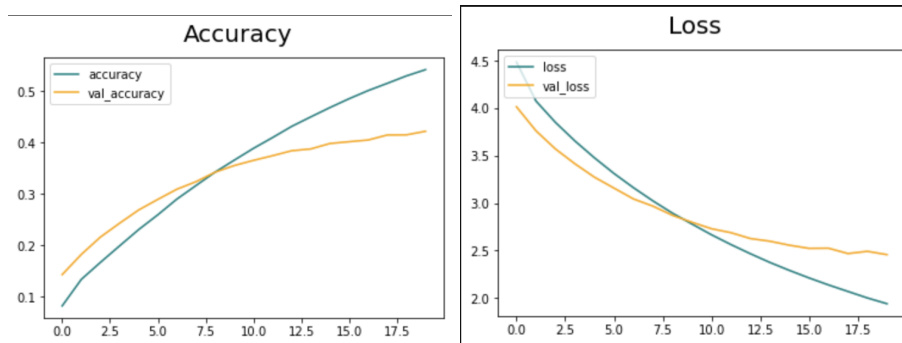


Accuracy and loss of Xception

Training Accuracy:41.24%

Validation Accuracy:37.72%

## ResNet50

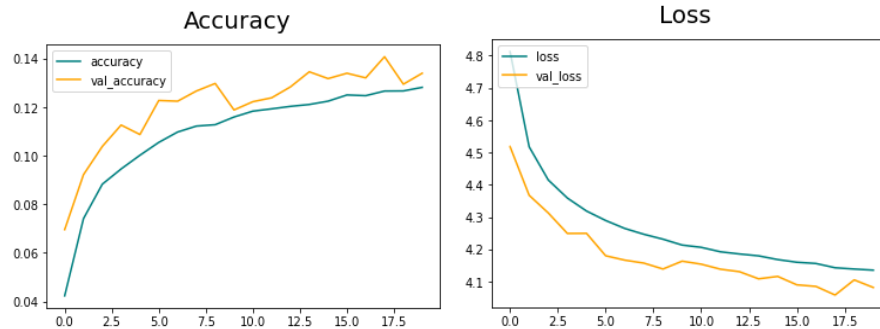


Accuracy and loss of ResNet50

Training Accuracy:54.12%

Validation Accuracy:42.16%

## EfficientNetV2B3



Accuracy and loss of EfficientNetV2B3

Training Accuracy:15.82%

Validation Accuracy:15.22%

## Work planning and organization of each team member

Pulkit and Kritya focused on implementing the base model and improving its validation accuracy by experimenting with different convolutional layers and hyperparameters.

Eashan worked on implementing the Object Detection on the image dataset.

Lokesh (VGG16, ResNet50, EfficientNetB7) and Xinyu (Xception, EfficientNetV2B3) worked on implementing the pre-trained models and transfer learning.

## My Contributions.....

We started the project by searching for ways to import the imagenet dataset and have a better understanding about the data that was stored in it. But the size of the dataset prohibited me and my teammates from doing that. So we looked for other datasets and none of them suited us. Then we came across tiny imagenet which is a subset of imagenet dataset. Till importing the dataset all of us worked together to make it easier and have a better understanding about the data. Me and Xinyu worked on pre-trained models which are basically used to get maximum accuracy with minimum time. While researching we came across a lot of pre-trained models like VGG-16, ResNet50, InceptionV3, MobileNet, EfficientNetB0 - EfficientNetB7 and many more.

The main idea about these pre-trained models that fascinated me was the number layers and filters that they had and the time they took was almost inversely proportional like ResNet50 has 50 layers and VGG-16 has 19 layers and of which 16 are weighted, but ResNet50 is supposedly faster than VGG-16. So after some research, I found that VGG-16, ResNet50, EfficientNet were some popular models with very high accuracy. But we did go ahead and try ResNet-34, InceptionV3, MobileNet as they had their perks like it was better while identifying dog breeds among dogs.

While implementing these pre-trained models, we found the biggest flaw was in our importing where it was getting imported in the size 256 x 256 which is default instead of 64 x 64 which was the size of the image. . Another problem that I faced during this process was while running the model, it took almost 30-40 minutes an epoch. To counter it we were searching for other solutions, that's when we found that there is an option in colab to change to GPU from the RAM, this gave extraordinary results where we were able to run an epoch 3-4 minutes. This gave us the chance to run our model more number of times with a lot of subtle changes to produce better results.

Initially I tried several optimizers for our model to find which was better, so I started with "RMSprop" as I could understand it as it was similar to Gradient descent in the aspect of taking average of roots and dividing it, but it did not do good to the accuracy. Next, I tried "adam" as it was supposed to be the best for image classification and it yielded some good results. To try different varieties, I added pretrained Imagenet dataset's weights which again tweaked the model's performance and for loss I had to go with sparse categorical cross entropy as our dataset had 200 classes. But all these experiments caused another fatal shortcoming where my colab suddenly stopped working and displayed the message that I ran out of GPU. But when I tried to run my model in a jupyter notebook I faced another issue where I was not able to use tensorflow in my laptop as it had an M1 chip and it was the same problem for pulkit as it had M1 pro chip.

When we googled this problem, we came to know that there is a separate version of anaconda for macbooks of M1 chip and a different procedure to download tensorflow. But even with all these attempts, results produced by my teammate's device were much better than my device for the same code.

Even after all these problems, accuracy produced by some of the pre-trained models were pretty good. I worked on VGG-16, ResNet50, EfficientNetB7. The validation accuracies were as follows 80.08%, 42.16%, 0.58% respectively. These accuracies were obtained by freezing layers to not lose the data about the weights. The reason that there is such a difference in accuracy is because it starts with a minimum number of layers and it keeps increasing(for example, VGG-16 has 16 weighted layers in comparison with 813 for EfficientNetB7) but when we add more layers and filters, the accuracy of the model should generally increase but it decreases as the image quality in our dataset is less. After which we tried transfer learning but since our base model had an accuracy of 57.15% we decided to try taking layers from VGG-16, Xception and the results were not better than our base model and the GPU limitation stopped me from experimenting with it.

## References

- ❖ Recht, B., Roelofs, R., Schmidt, L. and Shankar, V., 2019, May. Do imagenet classifiers generalize to imagenet? *International Conference on Machine Learning* (pp. 5389-5400). PMLR.
- ❖ Tammina, S., 2019. Transfer learning using vgg-16 with deep convolutional neural network for classifying images. *International Journal of Scientific and Research Publications (IJSRP)*, 9(10), pp.143-150.
- ❖ Benbrahim, H. and Behloul, A., 2021, November. Fine-tuned Xception for Image Classification on Tiny ImageNet. In *2021 International Conference on Artificial Intelligence for Cyber Security Systems and Privacy (AI-CSP)* (pp. 1-4). IEEE.
- ❖ Le, Y. and Yang, X., 2015. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7), p.3.
- ❖ Yu, H., 2017. Deep Convolutional Neural Networks for Tiny ImageNet Classification. *Stanford*.
- ❖ Kumar, P., Singh Singh, R., Singh, N.K. and Agarwal, H., 2020. Resizing Tiny Imagenet: An Iterative Approach Towards Image Classification.
- ❖ Sun, L., 2016. Resnet on tiny imagenet. *Submitted on*, 14.
- ❖ Basha, S.S., Dubey, S.R., Pulabaigari, V. and Mukherjee, S., 2020. Impact of fully connected layers on performance of convolutional neural networks for image classification. *Neurocomputing*, 378, pp.112-119.
- ❖ <https://neurohive.io/en/popular-networks/vgg16/>
- ❖ <https://datagen.tech/guides/computer-vision/resnet-50/>