Authors: Ze Hong Wu, Ying Jie Mei

Introduction

This final report summarizes and describes our work throughout the project.

This report will be structured to include the following sections:

- Part 1: Quick overview of project expectations, Stages I and II
- Part 2, 3, 4: Stage III, IV, and V
    - Proposed changes
    - Experiment done by both team members and outcomes, summarized
    - Model architecture and reasons for choosing it
    - Reflections on what could have been better
- Part 5: Final reflections

Part 1: Quick Overview

For this project, we were tasked to train a Convolutional Neural Network on the proprietary Pictures dataset. This dataset contains 25,000 images in five categories, with a Real World holdout test set on Gradescope that our models will be evaluated against in the end. Our deep learning model is expected to achieve a minimum of 79% F1-Macro score when tested against Real World.

For our initial project proposal, we offered the following roadmap:

- Stage III: Test different architectures
    - Ze Hong: VGGNet, ResNet, XCeption, DenseNet
    - Ying Jie: AlexNet, GoogLeNet, EfficientNet
- Stage IV: Test different tools
    - Ze Hong: Activation functions, gradient clipping
    - Ying Jie: Optimizers, Regularizers, Dropout
- Stage V: Possible architecture changes
    - This section was initially left vaguely defined, to be determined pending Stage III and IV results. Due to reasons discussed in the reflection sections, we did not test out any such changes.

Part 2: Stage III

During the Stage III work, we followed the plans offered in the project proposal and tested the aforementioned architectures.

Part 2.1: Stage III work and results

During our testing, we made a number of changes to these architectures. Some of these changes were necessary for this project, such as changing the output layer's number of units. Other changes, such as decisions to eliminate some layers, were motivated by hardware limitations.

Table 1: Ze Hong's work and results. Note that the "reduced output layer unit count" changes are not listed, since they are common to all models.

Model	Architecture changes (from paper specifications)	Loss	Accuracy	F1	Epochs	Time per epoch
VGGNet	Halved conv filter counts Added crop layer Reduced first dense layer neurons from 4096 to 512	0.8977	0.6436	0.6479	8	236.5s
ResNet	None	0.9376	0.7041	0.6992	6	267.4s
XCeption	None	0.6541	0.7734	0.7782	8	278.9s
DenseNet	Halved residual block counts	0.6446	0.7686	0.7696	16	270.4s

Table 2: Ying Jie's work from all three architectures.

Model	Architecture modifications	Loss	Accuracy	F1	Epochs	Training Time
AlexNet	Reduced filters significantly Reduced unit size Standardized kernel sizes and pool sizes Reduce output layer unit to 5	1.4645	0.6830	0.6800	10	516s; 8m36s
GoogLeNet	Simplified inception module Reduced layers Reduced unit size Reduce output layer unit to 5	2.3966	0.6866	0.6859	20	803s; 13m23s
EfficientNet	Reduced filters Reduced layers Reduce output layer unit to 5	0.8151	0.7196	0.7040	20	873s; 14m33s

Tables 1 and 2 indicate that the DenseNet architecture appeared to have the best accuracy when evaluated against the local training set. For this reason, we chose to proceed with the DenseNet model.

Part 2.2: Model architecture and Reflection

The three architectures, AlexNet, GoogLeNet, and EfficientNet, seem like good picks. However, in our case, a full replication of the three architectures would be difficult, and the network would be too deep and large. So, I (Ying Jie) have replicated a similar version but a simple model inspired by these three architectures. It still retains the core features that resemble the three architectures. As I trained the model, I came across a problem with the models, their sizes were too large for Gradescope to accept as they were way above the limit Gradescope allows.

EfficientNet would have been the go-to architecture if DenseNet had not emerged victorious from all the architecture models we trained. We also should have reduced our choice of architectures to train to only one or two per person since the decision to choose seven architectures actually made it difficult for us to complete training with different techniques.

In the end, after comparing our results, we chose to proceed with the DenseNet model.

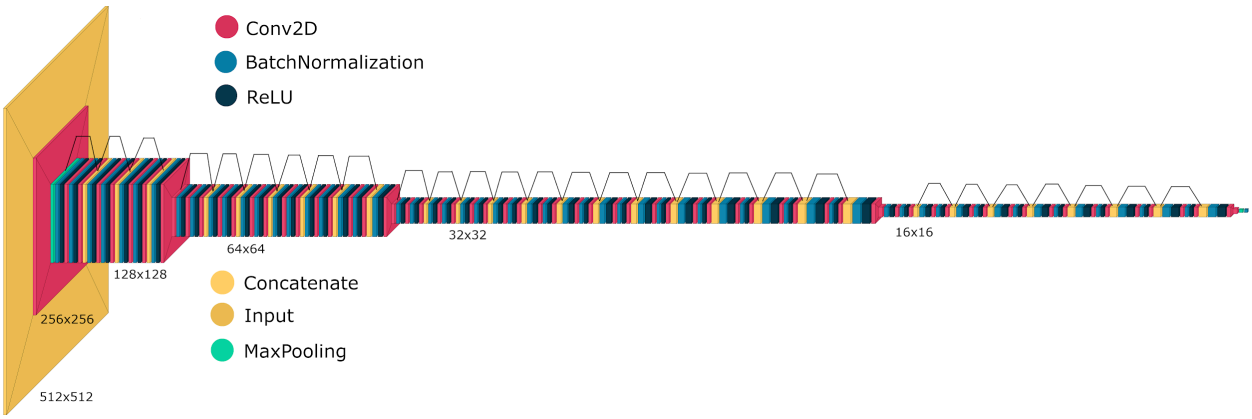

Figure 1: Architecture of the chosen model, generated using the visualkeras package for Python and edited using GIMP and Paint. Black lines connecting different layers are skip connections. Note that there should be many more skip connection lines than is drawn, due to the DenseNet architecture implementation of skip connections.

At the end of Stage III, we proceeded with a DenseNet model as described in Figure 1. It is derived from the DenseNet-121 architecture with half of its residual blocks removed.

The DenseNet architecture is primarily made up by a number of dense stacks, each containing a number of residual blocks, each connected to the outputs of all previous residual blocks using skip connections. Each dense stack is followed by transition layers that downscale and downsize the input, except for the last dense stack which leads directly to the final layers.

Table 3: Residual Block architecture. Note that x, y, filters are variables and that each residual block has an input of a different size.

Layers	Parameters and Notes	Output shape
Input	Residual Block begins here	x,y,filters
BatchNormalization		x,y,filters
ReLU		x,y,filters
Conv2D	128 filters, 1x1 kernels, stride 1, same padding	x,y,128
BatchNormalization		x,y,128
ReLU		x,y,128
Conv2D	32 filters, 3x3 kernels, stride 1, same padding, output=b	x,y,32
Concatenate	Skip connection, links this block with outputs of previous blocks	x,y,filters+32

Table 3 describes the architecture of each individual residual block. The input is passed through a number of BatchNormalization and Conv2D layers before being concatenated with the outputs of all previous residual blocks.

Table 4: Transition Block architecture. Once again, x, y, filters are variables and reflect the fact that each transition block has a different input.

Layers	Parameters and Notes	Output shape
Input	Transition layer begins here	x,y,filters
BatchNormalization		x,y,filters
ReLU		x,y,filters
Conv2D	filters/2 filters, 3x3 kernel, stride 1, same padding	x,y,filters*0.5
AveragePooling2D	2x2 pooling, stride 2, same padding	(x,y,filters)*0.5

Table 4 describes the architecture of each transition block. The AveragePooling2D layer is the most important layer, as it down-sizes the input by reducing the dimensions of the input.

Table 5: DenseNet model architecture

Layers	Parameters and Notes	Output shape
Input		512,512,3
Conv2D	64 filters, 7x7 kernel, stride 2, same padding, relu	256,256,64
MaxPooingl2D	3x3 pooling, stride 2, same padding	128,128,64
Residual Block		128,128,96
Residual Block		128,128,128
Residual Block		128,128,160
Transition Block		64,64,80
Residual Block x6	Filter counts after each block are: 112, 144, 176, 208, 240, 272. (x,y) dims remain at (64,64).	64,64,272
Transition Block		32,32,136
Residual Block x12	Filter counts after each block are: 168, 200, 232, 264, 296, 328, 360, 392, 424, 456, 488, 520. (x,y) dims remain at (32,32).	32,32,520
Transition Layer		16,16,260
Residual Block x8	Filter counts after each block are:290, 324, 356, 388, 420, 458, 484, 516. (x,y) dims remain at (16,16).	16,16,516
Transition Block	Unintentionally added to the model during Stage III. It was removed in Ze Hong's tests but not in Ying Jie's tests. It will be removed in Stage V.	8,8,258
GlobalAveragePooling2D		None,258

Table 5 describes the overall architecture of the DenseNet model we chose, using the residual block and transition block architectures mentioned in Tables 3 and 4. Note that a number of the residual blocks are listed together to cut down on redundant text.

After consultation we conclude that we should have focused on fewer architectures in order to gain a more in-depth understanding of them, which would in turn allow us to test more tools in depth in subsequent stages.

Part 3: Stage IV

As we described in our Stage III report, we proposed to investigate a number of tools and hyperparameters that can improve the performance (measured by accuracy on the testing set) of our models. In addition to those specified in the project proposal, we also tested a couple of hyperparameters unique to the DenseNet architecture.
Ze Hong: Activation functions, gradient clipping
Ying Jie: Optimizers, Regularizers, Dropout

During this portion, we discovered that we had incorrectly formatted our TFDatasets and that the labels on our local sets did not match those on the Real World holdout set on Gradescope. We corrected this error and continued on.

Part 3.1: Stage IV work and results

During Stage IV, Ze Hong tested a number of changes on the Stage III model. Two important variables mentioned in the tables below require some explanation.

Compression Factor (CF) is a number that determines how many filters are preserved when going through a transition block. For example, with a CF of 0.75, the number of filters after passing through the transition block will change from X to 0.75X.

Growth Rate (GR) is a number that determines the rate at which the number of filters change after each residual block. A residual block with a GR of 32 might take in an input of 128 filters, pass it through its convolution layers to produce an output of 32 filters, then concatenate the two to produce a true output of (32+128)=160 filters.

Table 5: Table of Ze Hong's changes tested and experimental results. Loss, Accuracy, and F1 Macro scores are derived from evaluating the models against the local test set.

Model	Based On	Changes	Loss	Accuracy	F1 Macro
Baseline	none	Baseline from Stage III	0.6446	0.7686	0.7696
Initial	Baseline	Removed transition layer Fixed learning rate bug Trained for 24 epochs	0.5793	0.8049	0.8041
ClipNorm	Initial	Added clipnorm=1.0 to SGD	0.5196	0.8428	0.8404

LeakyReLU	ClipNorm	Changed ReLU for Leaky ReLU	0.4755	0.8263	0.8245
Compression	ClipNorm	Compression Factor changed from 0.5 to 0.66	0.4658	0.8433	0.8422
Growth Rate	Compression	Growth Rate changed from 32 to 48	0.5159	0.8277	0.8252
Learning Rate	Compression	Learning Rate changed from -90% at 12 and 18 epochs to -50% at 12 and 18 epochs	0.5168	0.8134	0.8141

Table 6: Ying Jie's experimentation and results for Stage IV.

Initial Model	Model Name	Modification	Value	Loss	Accuracy	F1_Score
Baseline	my_densenet			0.5614	0.7959	0.7962
my_densenet	M1	Optimizer	Adam	0.3813	0.8574	0.8589
M1	M2	Regularization	Dropout layer after activation function	0.9873	0.5918	0.5835
M1	M3	Filter	Increment base filter after each dense block	0.4921	0.8125	0.8157
M1	M4	Regularization	Dropout layer after convolution layers	0.8932	0.6523	0.6515
M1	M5	Image shape	Changed image shape to (256,256,3)	0.4403	0.8216	0.8243
M1	M6	Regularization Filter	Dropout layer after activation, before and after convolution, before and after concatenate, before and after transition layers, also filter scaling by 16 at end of each dense block	1.6273	0.3818	0.3372

As a result of comparing these results, we concluded that Ying Jie's Adam model (which implements the Adam optimizer) performed the best. We chose to submit that model for Stage IV and continue into Stage V with it.

Part 3.2: Model architecture and Reflection

For stage IV there wasn't much interesting to do besides trying out different techniques and trying to improve the previous stage model accuracy and f1 score. The problem in this case was that trying out the regularization method wasn't helping as we trained, and other modifications on depth size and width size such as adding more layers and adding more units also wasn't helping in this case but the accuracy and f1 score unfortunately failed our expectation. We then drop these modifications and continue with other techniques.

The model architecture for Stage IV is largely similar to the Stage III model, with the exception of using the Adam optimizer instead of the SGD optimizer. The Adam optimizer uses a ClipNorm of 1.0.

Additionally, the chosen model itself uses these hyperparameters:

- Growth Rate: 32 filters
- Compression Factor: 0.5
- Activation function: ReLU
- Metrics: Accuracy, F1 Macro
- Loss: Categorical Cross-entropy
- Epochs trained: 16
- Initial Learning Rate (lr): 0.1
- Learning Rate schedule: Reduce lr by 90% at 8 and 12 epochs

As the model architecture and layers makeup is unchanged, we have elected to not re-describe it in depth.

After consulting with each other during the writing of this document, we concluded that in Stage IV we also suffered from the same issue in Stage III - trying to test too many different things and thus not being able to go in depth on any of them.

Part 4: Stage V

In our Stage IV report, we described our plans for Stage V. We combined the Compression + Learning model and the Adam model. Then Ying Jie investigated compression factor values of 0.75 and 1.0, while Ze Hong tested a number of learning rate schedulers.

Part 4.1: Stage V work and results

During the initial portions of Stage V work, Ze Hong discovered that the Adam optimizer destructively interfered with the Compression change and that combining these two changes led to a model with less accuracy than either of its two predecessors. We made attempts to back-track our work and ultimately found that following Ying Jie's model and abandoning attempts at merging the previous two models was the best course of action.

Table 7: Loss - Accuracy - F1 Macro values for the combined model and sequentially backtracked versions.

Model	Based on	Changes	Loss	Accuracy	F1 Macro
Adam	Ying Jie's work	n/a	0.3813	0.8574	0.8589
Compression	Ze Hong's work	n/a	0.4658	0.8433	0.8422
Combined	Adam + Compression	Merges the two above models	0.5318	0.8143	0.8164
No-Compress	Combined	Reset compression value to 0.5	0.5311	0.8147	0.8168
Last-transition	No-compress	Re-added removed transition layer	0.5304	0.8121	0.8177

After choosing to fully commit to Ying Jie's model, we continued testing our assigned tools.

Table 8: Loss - Accuracy - F1 Macro values for various scheduler models tested by Ze Hong.

Model	Based on	Changes	Loss	Accuracy	F1 Macro
Combined	Adam + Compression	Merges the two above models	0.5318	0.8143	0.8164
Learning-1	Combined	-90% lr at 50% and 75% epochs; initial lr of 0.01	0.4581	0.8464	0.8164
Learning-2	Learning-1	-50% lr instead of -90%	0.5171	0.8304	0.8275
Learning-3	Combined	-0.0005 lr at 25%, 50%, and 75% epochs; initial lr of 0.002	0.5068	0.8384	0.8365
Learning-4	Combined	-0.001 lr per epoch; initial lr of 0.025	0.6234	0.7937	0.7933
Learning-5	Learning-1	Set compression to 0.5	0.4495	0.8455	0.8438

Learning-6	Learning-5	Used custom scheduler 1	0.5124	0.8192	0.8179
Learning-7	Learning-6	Used custom scheduler 2	0.4880	0.8326	0.8316

Table 9: Ying Jie's work and results in Stage V.

Model	Based on	Changes	Loss	Accuracy	F1 Macro
Baseline	Stage IV Model	Adam Optimizer	0.3813	0.8574	0.8589
Compression-1	Adam	Changes compression from 0.5 to 0.75	0.4153	0.8525	0.8552
Compression-2	Adam	Changes compression from 0.5 to 1.00	0.4417	0.8418	0.8443

As a result of comparing our results for Stage V, we concluded that the initial learning rate scheduler from Stage IV remained the best option we have. In light of this, Ying Jie selected the Baseline model from Table 9 and trained it for 48 epochs.

Part 4.2: Model architecture

The model architecture for Stage V is nearly identical to the Stage IV model, with the exception of training for an increased number of epochs. As no other changes occurred here, we have chosen to not re-describe it in depth.

Part 5: Final reflections

Overall, we made progress from Stage III to V, but we could have done better by reducing architecture selections during Stage III. We could have either chosen a single architecture to work on or worked on one architecture for each person. This way, we would have been able to make further progress and use more techniques.

In general, we suffered from an issue of trying to do too many things. Had we focused on a fewer number of architectures and tools, we may have achieved better results or learned more from this project.

Word count: 2783