Authors: Ze Hong Wu, Ying Jie Mei

Introduction

This Stage 3 report describes the work we carried out between submitting the Project Proposal and submitting this report. It will be divided into three parts, with two parts corresponding to a team member's work and research and the third part corresponding to discussions on how we will combine our work for the next stage.

Part 1: Work by Ze Hong Wu

For Stage 3 of this project I trained variants of VGGNet, ResNet, XCeption, and DenseNet. A summary of relevant statistics will be presented in a table at the end of Part 1 of this document.

All models were trained for 20 epochs and with Model Checkpoint and Early Stopping callbacks. The model checkpoints save the model iterations with the lowest validation loss while the early stopping callbacks stop learning after 4 epochs of no improvement to validation loss.

VGG16

During the modeling and training process for the VGG16 model, I changed the model from the architecture described in the corresponding paper in the following ways:

- Reduced the filters count on all but the last stack of convolution layers by 50%
- Reduced unit count of the first dense layer from 4096 to 512
- Added a crop layer at the start
- Changed the output layer from 1000 units to 5

The first and second changes were motivated by a desire to reduce the amount of trainable parameters and GPU RAM usage. This may have been a short-sighted decision and I probably should have attempted to train a full VGG16 model.

The change to add a cropping layer was partially motivated by a desire to cut down on parameters and also on the following line from the VGG16 paper:

"To obtain the fixed-size 224×224 ConvNet input images, [the inputs] were randomly cropped from rescaled training images (one crop per image per SGD iteration)."

In retrospect, this decision to add a crop layer instead of modifying the model architecture to be compatible with 512x512 images was counterproductive. Such a careless cropping removed important features that the model could have used to classify the images. However, due to my

failure to respect previous students' advice to start work earlier, I did not have the time to re-train the VGG16 model.

The final change compels the model to classify five categories instead of 1000 and cannot be avoided.

The VGG16 model was trained for 8 epochs out of 16 (stopped at 12) with early stopping and model checkpoints. The model had a loss of 0.8977, an accuracy of 0.6436, and an F1 of 0.6479 when evaluated on 32 images from the testing set.

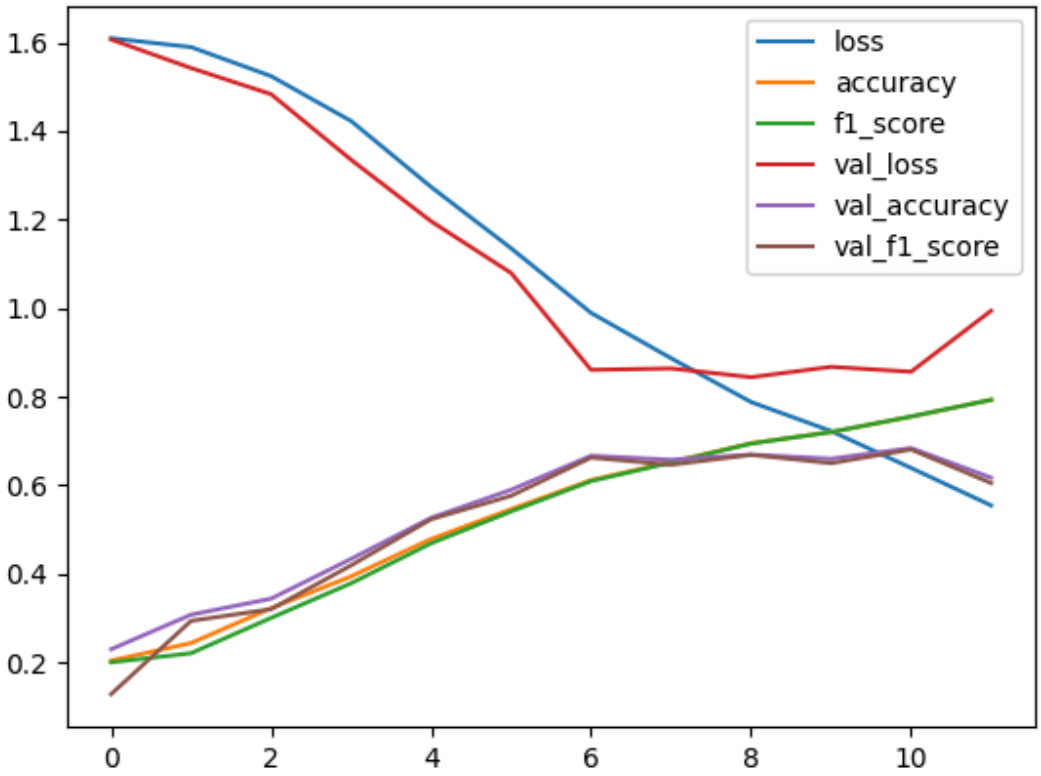

Figure 1: Loss, accuracy, and F1 of VGG13 model during training. Training seems to plateau by epoch 6, however this might just be a cause of chaotic behavior at low training levels.

ResNet34

During the modeling and training process for the ResNet34 architecture model, I changed the model from the architecture described in the corresponding paper in the following ways:

- Changed final dense layer from 1000 units to 5

Besides the singular change to predict 5 categories, the model was otherwise unchanged.

The ResNet34 model was trained for 6 epochs out of 16 (stopped at 10) with early stopping and model checkpoints. It had a loss of 0.9376, an accuracy of 0.7041, and an F1 of 0.6992 when evaluated on 32 images from the testing set.
Figure

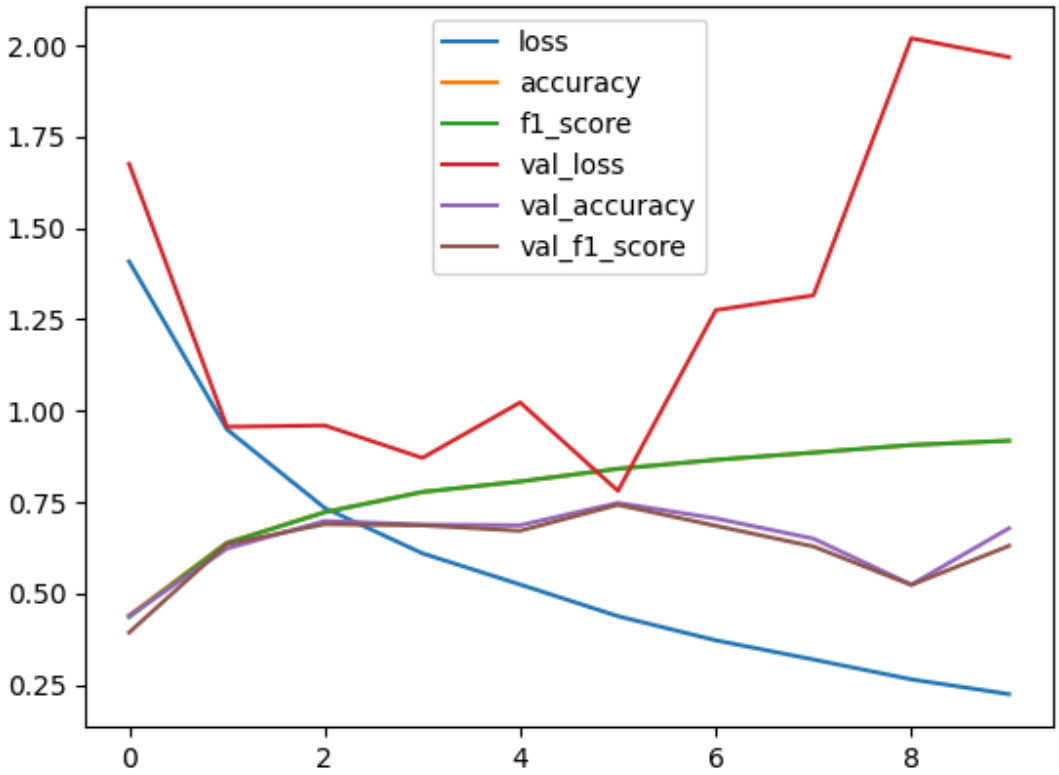

Figure 2: Loss, accuracy, and F1 of ResNet34 model during training. The model seemingly went haywire past epoch 5. Possible explanations include early training chaotic movements, rapid overfitting past that point (unlikely), some sort of double descent, and lack of regularization in the model architecture.

XCeption

During the modeling and training process for the XCeption architecture model, I changed the model from the architecture described in the corresponding paper in the following ways:

- Changed final output layer from 1000 units to 5

The XCeption model was trained for 8 out of 16 epochs (stopped at 12) with early stopping, model checkpoints, and learning rate scheduling. Following instructions from the XCeption

paper, I trained the model with a scheduler that reduced the learning rate by 6% (multiplication by 0.94) every two epochs. The model had a loss of 0.6541, an accuracy of 0.7734, and an F1 of 0.7782 when evaluated on 32 images from the testing set.

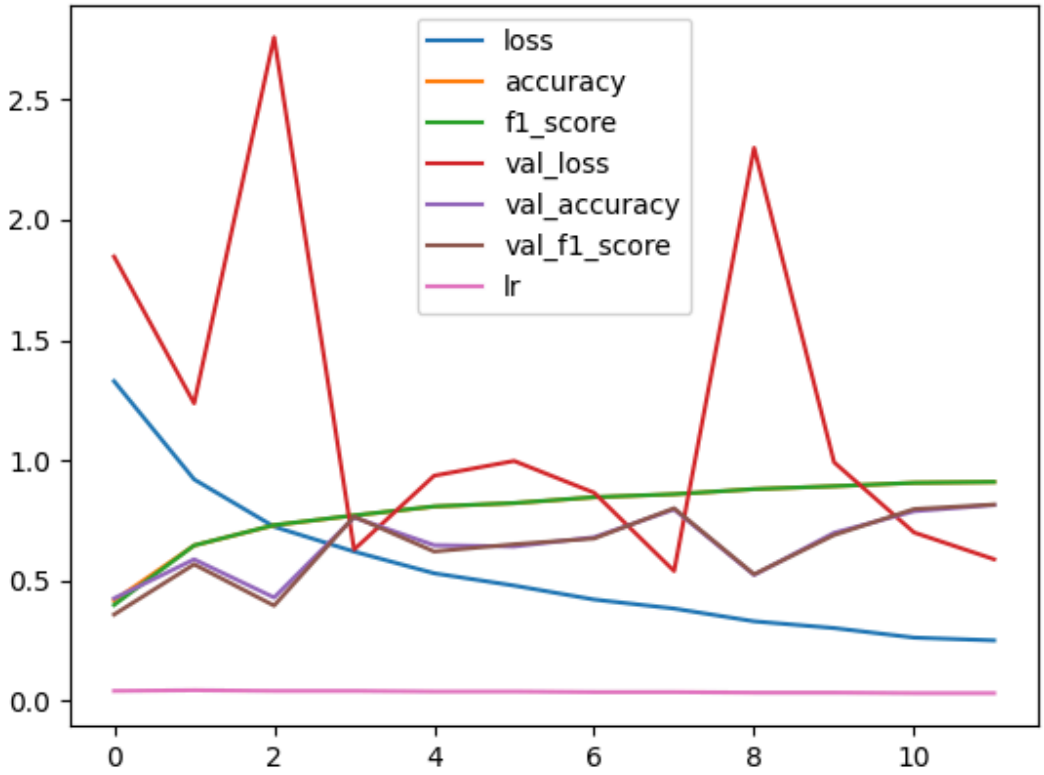

Figure 3: Loss, accuracy, and F1 of XCeption model during training. The validation loss saw two large peaks but appeared to be dropping near the last epochs prior to stopping. The learning rate change was relatively insignificant relative to that of the other variables. Further training without or with a more lenient early stopping may yield better results.

DenseNet

During the modeling and training process for the DenseNet model (based on the DenseNet121 architecture), I changed the model from the architecture described in the corresponding paper in the following ways:

- Changed final output layer from a logistic regression to a Dense layer with softmax output
- Reduced number of residual blocks in each dense block by 50%

The second change was motivated by hardware limitations. When training the model based on a full number of convolution layers, the GPU (a V100 device) exceeded its RAM limit and crashed the session during the first few seconds of training. I halved the convolution layer counts to reduce RAM usage. This shrunk-down DenseNet model was able to be trained without issue.

The DenseNet model was trained for a full 16 epochs with early stopping, model checkpoints, and learning rate scheduling. Following instructions from the DenseNet paper, I trained the model with a scheduler that halved the learning rate after 50% and 75% of the training epochs occurred, respectively.. The model had a loss of 0.6446, an accuracy of 0.7686, and an F1 of 0.7696 when evaluated on 32 images from the testing set.

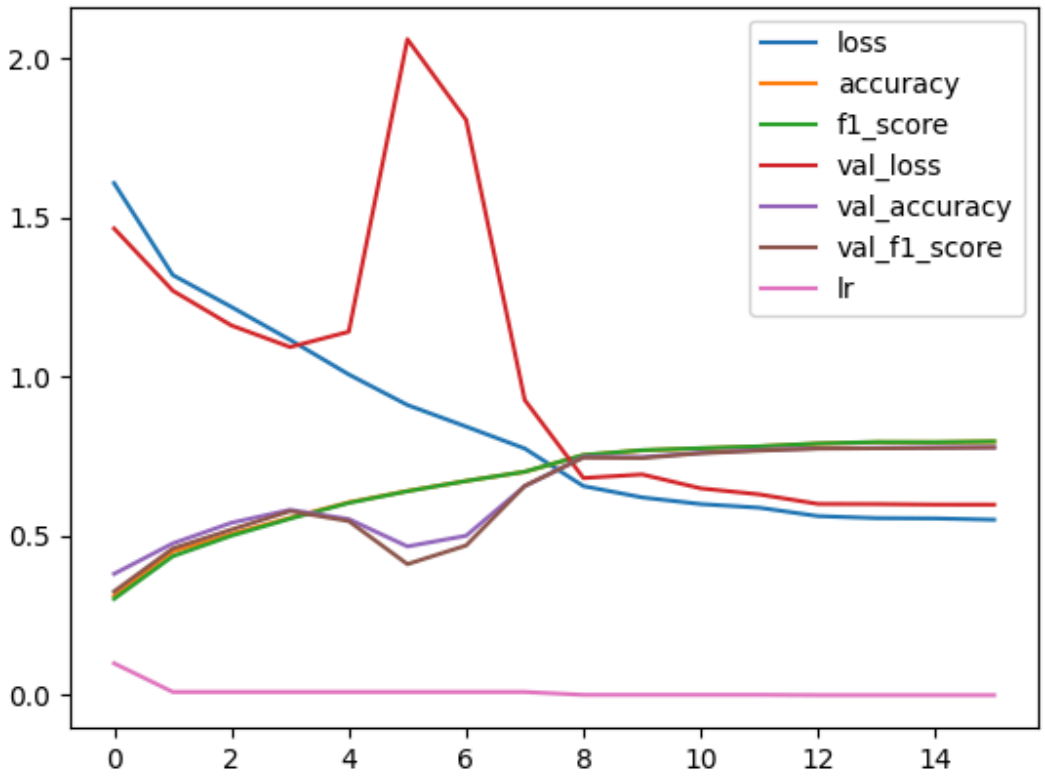

Figure 4: Loss, accuracy, and F1 of DenseNet model during training. The sudden drop in learning rate at the start is an error caused by a typo in the learning scheduler code that was not noticed until now. The validation loss curve is more consistent with conventional model training results.

One notable feature of the DenseNet model is that it is surprisingly small in terms of file size; while the VGG, ResNet, and XCeption models fell between 80 to 120 megabytes, the DenseNet model reached an unexpectedly low size of 19.6 megabytes.

Summary of Experiences

Table 1: A summary of the results of training each of the four 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

There are a few important facts to keep in mind here:

- The “changed final dense layer to use 5 units” is omitted since it was required for all models in order to predict the correct number of categories.
- Loss, Accuracy, and F1 scores are derived from evaluating on the testing set.
- “Epochs” values do not count epochs rolled back by early stopping.

Part 2: Work by Ying Jie Mei:

For stage III of the project, I have trained models of the inspired form of AlexNet, GoogLeNet, and EfficientNet architectures. A summary statistic table is at the bottom of part 2.

During training, I tried to train the model to its fullest potential, but the file size was too large to submit on Gradescope. I was forced to decrease its size by significantly reducing layers and unit counts to successfully create the model in under 100 MB and upload it to Gradescope.

For all the models, I have modified the model from its original architecture and trained with the most basic tools for a simplified version of itself or using its core features.

All the models used the standard batch size of 32, and Adam was the optimizer.

AlexNet

For this model, I have kept the essentials of AlexNet, even though it is similar to LeNet, AlexNet is known for its deep convolutional layers; unlike LeNet, AlexNet contains convolutional layers that stack directly on top of each other. I have retained the feature of the input of 227x227 image with five convolutional layers and three fully connected layers, including the output layer of five units (since we only have five classes), a 50% dropout rate for each of the two fully connected layers, and corresponding activation.

The following are the modifications I have done:

-Reduced the unit size to 512 for a fully connected layer and 256 for another fully connected layer
-Reduced the filters significantly
-Changed the output layer unit to 5 since we only have five classes.
-Standardized kernel sizes and pool sizes

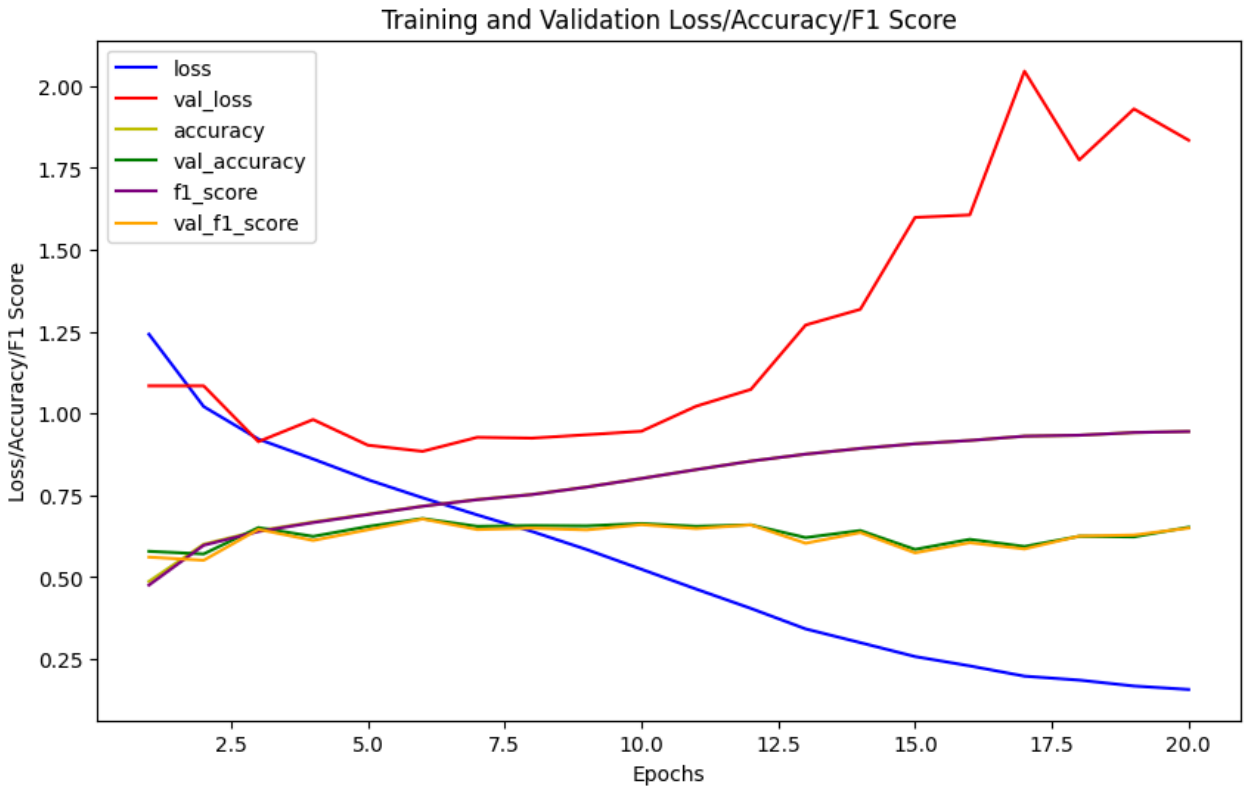

Figure 5.

The graph shown suggests as the model trains with more and more epochs, it is not improving but getting worse. There's a divergence at around six epochs, which is an indication of overfitting. As the training goes on, the validation loss increases significantly due to overfitting; learning the data too well is a problem.

GoogLeNet

The architecture uses the inception module, and I have modified the model to be significantly lower than the values from the original GoogLeNet architecture shown in the textbook. I used a simpler inception module and used 1x1 convolutions in the module while also incorporating the usage of global average pooling.

The following are the modifications I have done:

-Simplified inception module
-Reduced layers
-Reduced unit size
-Reduced units for the output layer to five

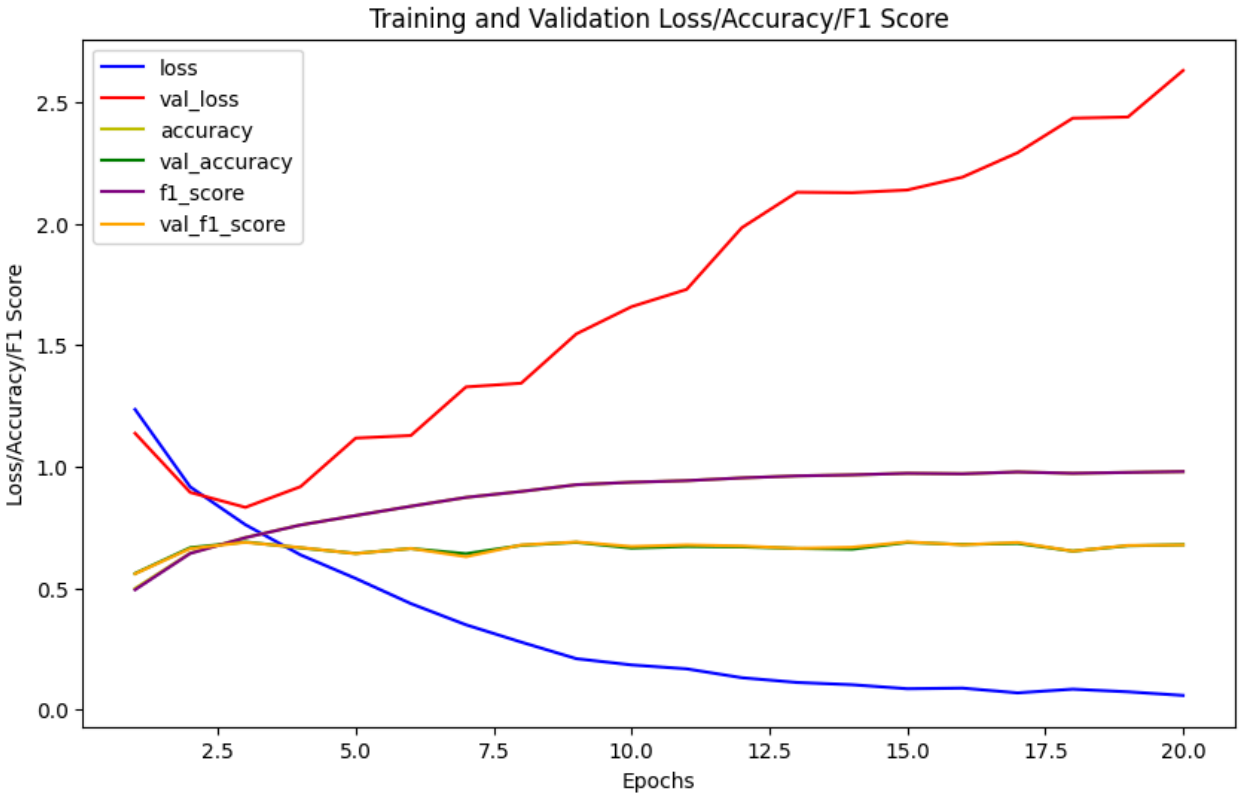

Figure 6.

The graph shown suggests as the model trains with more and more epochs, it is not improving but getting worse. There's a divergence at around three epochs, which is an indication of overfitting. Similar to AlexNet, this model learns the data too well and goes overfitting at a certain point.

EfficientNet

This architecture was complicated to make since I wasn't able to reproduce something similar to EfficientNetB0 or any other architectures from B0-B7. I was able to make something similar to its general features, like the use of depthwise separable convolutions, batch normalization, ReLU activation, and global average pooling. However, the original EfficientNet architectures contain compound scaling methods and squeeze and excitation blocks, which I wasn't able to incorporate into the model I created.

Some modifications I have done:

-Reduced filters
-Reduced layers
-Reduced output layer units to five because we only have five classes

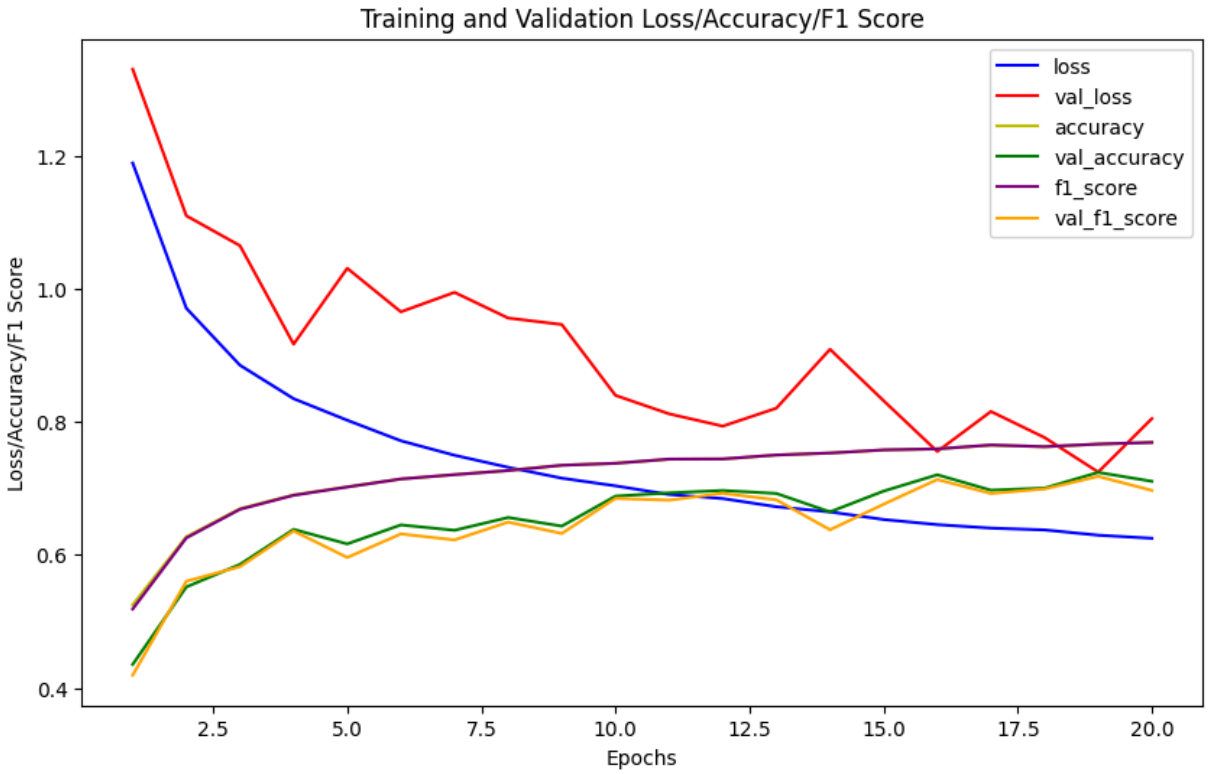

Figure 7.

This graph of EfficientNet seems to perform better than the previous two architectures, as it still diverges but not as much as the last two, but compared to the other two, this model seems not to overfit as much, and the model is learning from the training data as we can see from the graph.

Summary

Table 2

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

The best model out of the three would be EfficientNet since it has better accuracy and f1 score and didn't overfit as severely as the other two architectures. Still, because I didn't fully utilize the methods from the EfficientNet architecture, I am not sure whether this architecture would actually best AlexNet, GoogLeNet, and other architectures. And because I reduced all three models' techniques to retaining only the core features while reducing further for the layers and unit size, they took way less time to train; I wouldn't say they are the best models for training the data we have now since these models are technically designed for enormous datasets. Overall, I will not choose any of the three models for the following stage IV model.

Part 3: Stage IV plans

For stage 4, after some discussion, we have chosen to continue with the DenseNet model. A detailed description of the model's architecture is presented below.

Model architecture

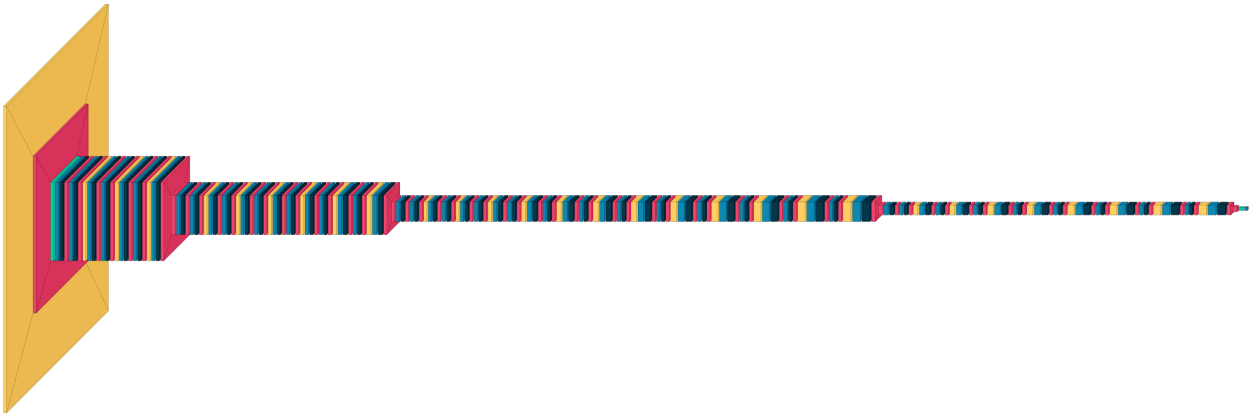

Figure 8: Architecture of the DenseNet model, generated using the visualkeras package for Python. Due to difficulties with working with PlotNeuralNet, I elected to not use it to generate this image.

The DenseNet architecture contains a number of residual blocks with the following architecture. A number of residual blocks stacked end to end form a dense block, and the model contains four dense blocks of different lengths. The model also contains a number of transition layers, placed after each dense block except the last, that controls the size of the model.

In the figure above, each dense block corresponds to a long stack of layers with similar (x,y) dimensions. Each residual block corresponds to a regular pattern of colored layers in each dense block. Each transition layer is located at the end of each dense block, where it meets the next block. The output dense layer is represented by the few green pixels at the far right, while the input image is the yellow layer on the far left.

The architectures of the residual blocks and transition layers are presented in the tables below.

Table 3: Residual Block architecture

Layers	Parameters and Notes	Output shape
Input = a	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	concat(b,a)	x,y,filters+32

Table 4: Transition Layer architecture

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/2
AveragePooling2D	2x2 pooling, stride 2, same padding	x/2,y/2,filters/2

With the residual block architecture described properly, the main architecture can be described in depth.

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 Layer		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 Layer		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 Layer	Unintentionally added to the model as a result of forgetting to update a check when reducing residual block counts. This layer will be removed in Stage IV.	8,8,258
GlobalAveragePooling2D		None,258
Output	Dense layer, 5 units, softmax activation	None,5

Notice that the filter count grows by 32 per residual block and the dimension values are reduced by 50% per transition layer. The authors of the DenseNet paper describe these values as the growth rate and the compression factor. These values can be freely modified to change the rate at which filters increase per residual block and the amount of compression each transition layer imparts upon the model. Additionally, the number of residual blocks per dense block can be modified as well.

Recall that, as shown in Figure 4 and Table 1, DenseNet outperformed nearly all of the other models we trained for Stage 3 (barring XCeption which was marginally better). However, unlike XCeption, DenseNet did not exhibit the massive validation loss fluctuations that XCeption did and displayed a smooth loss decrease after overcoming an initial spike. Additionally, the DenseNet model took up only 19.6 megabytes of space, a massive improvement over the 159.6 megabytes of the XCeption model. Because of these reasons, we have chosen to continue to Stage 4 with our DenseNet model.

Work Distribution

In our project proposal, we proposed the following work distribution for Stage IV:

Ze Hong Wu: Activation Functions, Gradient Clipping
Ying Jie Mei: Optimizer, Regularization, Dropout, Early Stopping

In addition, we propose testing the following architecture changes to the DenseNet model:

Ze Hong Wu: Partial revert of the residual block count reduction (from -50% to -25%); changes to the growth rate and compression factor; removal of the unintended transition layer
Ying Jie Mei: Adjusting the Depth of the Network, various input image resolutions, modifications on layers within dense blocks such as bottleneck layers, feature reduction in transition layer

We may choose to investigate other architecture changes or tools during our project work, if the idea comes to mind. We will report all such unplanned changes in the Stage IV report.