CS 49377 Project Presentation

Team ExpoGeek
Authors: Ze Hong Wu, Ying Jie Mei

Project Descriptions

Dataset:

- Pictures dataset, 25,000 images, 5 categories
- Holdout set “Real World” in Gradescope

Objectives

- Train a CNN model to classify images from Pictures dataset
- Model must achieve an F1-Macro score of >79% on Real World

Project Proposal

In our project proposal we planned to:

- Stage III: test out 7 model architectures on the dataset
  - Ze Hong tested VGGNet, ResNet, XCeption, Densenet
  - Ying Jie tested AlexNet, GoogLeNet, EfficientNet
- Stage IV: Using the best architecture identified, test out various tools
  - Ze Hong tested activation functions and gradient clipping
  - Ying Jie tested optimizers, regularization layers, dropout layers
- Architecture changes not discussed
  - Depends on what architecture we select

Stage III: Ze Hong’s Work

Models are judged by their testing accuracy.

Experiments and Observations

- Tested four architectures: VGGNet, ResNet, XCeption, DenseNet
  - 20 epochs, SGD optimizer
  - Model checkpoints (validation loss), early stopping (4 epochs of no improvement)
  - Changed layer counts due to Google Colab resource limits
  - Mistake made: unnecessary and probably harmful cropping layers

Quick Explanation: VGGNet

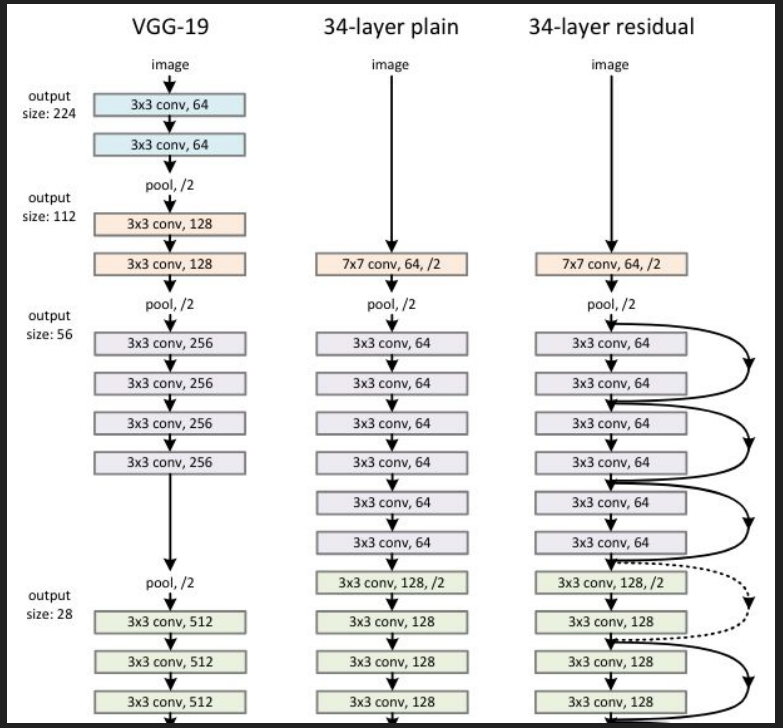
- VGGNet has blocks of conv layers
  - Separated by max-pool layers
- We implemented VGG16 with 16 conv layers
- Implemented a cropping layer at the start since the VGG paper mentioned it
  - Bad idea: Eliminated useful features from image, almost certainly reduced accuracy

Source: https://arxiv.org/pdf/1409.1556

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Quick Explanation: ResNet

- ResNet uses residual blocks with skip connections
  - Skips connect a block's input with its output
- We implemented and tested a ResNet-34 architecture

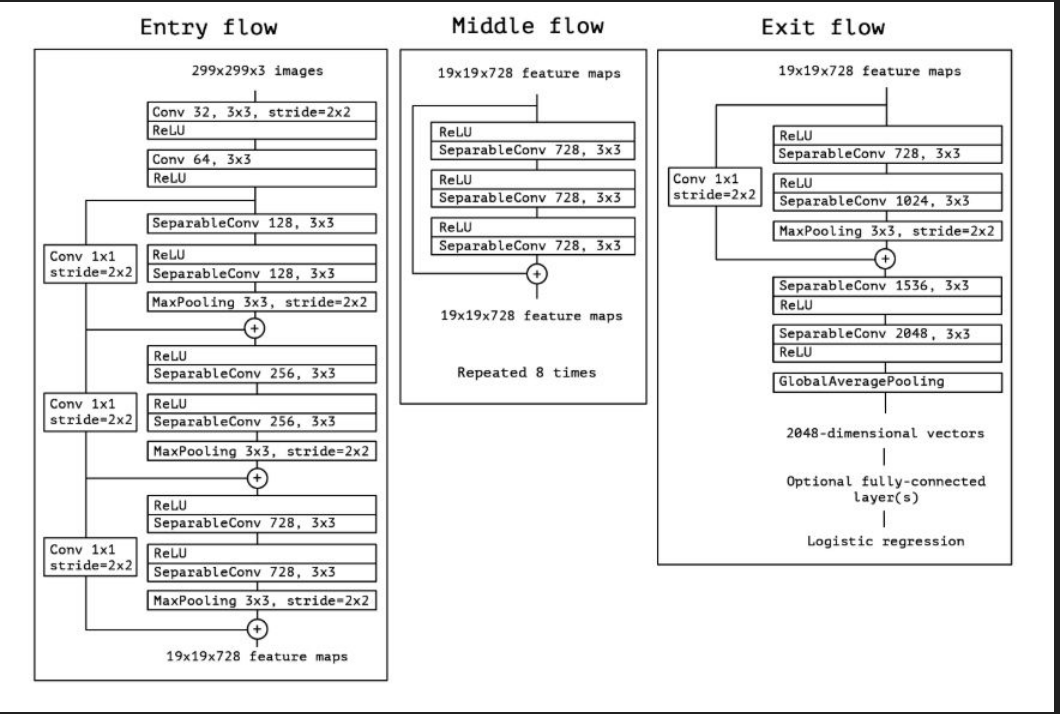


Source: https://arxiv.org/pdf/1512.03385

Quick Explanation: XCeption

- XCeption uses Separable Convolution layers
  - Depth-wise SConv + Point-wise SConv
- Many residual blocks

Source:
https://arxiv.org/pdf/1610.02357



Quick Explanation: Depth-wise and Point-wise

Depth-wise Separable Conv

- Uses one kernel per input filter
- Outputs one filter per input filter

Point-wise Separable Conv

- Uses X 1x1 kernel going through layers on all input filters
- Outputs X filters

Tensorflow Separable Conv uses depth-wise followed by point-wise

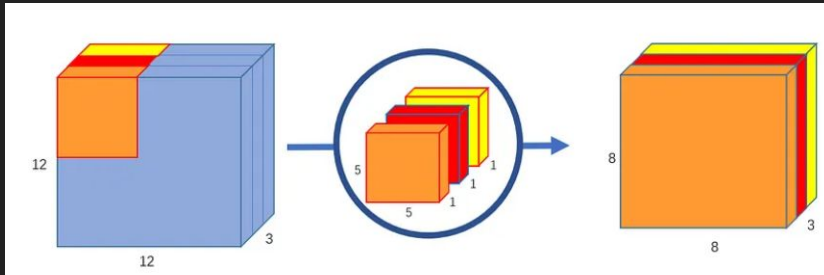


Image 6: Depthwise convolution, uses 3 kernels to transform a 12×12×3 image to a 8×8×3 image

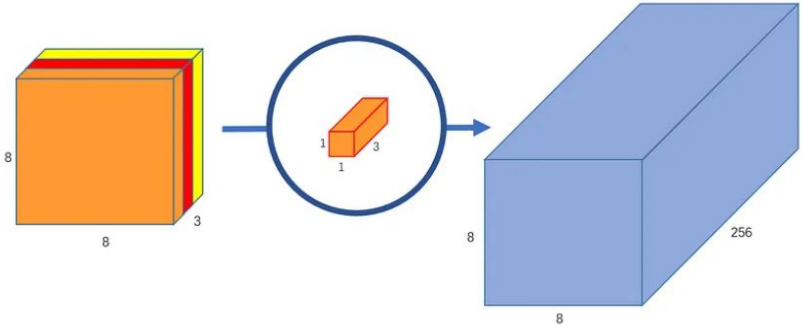


Image 8: Pointwise convolution with 256 kernels, outputting an image with 256 channels

Source: Depth-wise and Point-wise

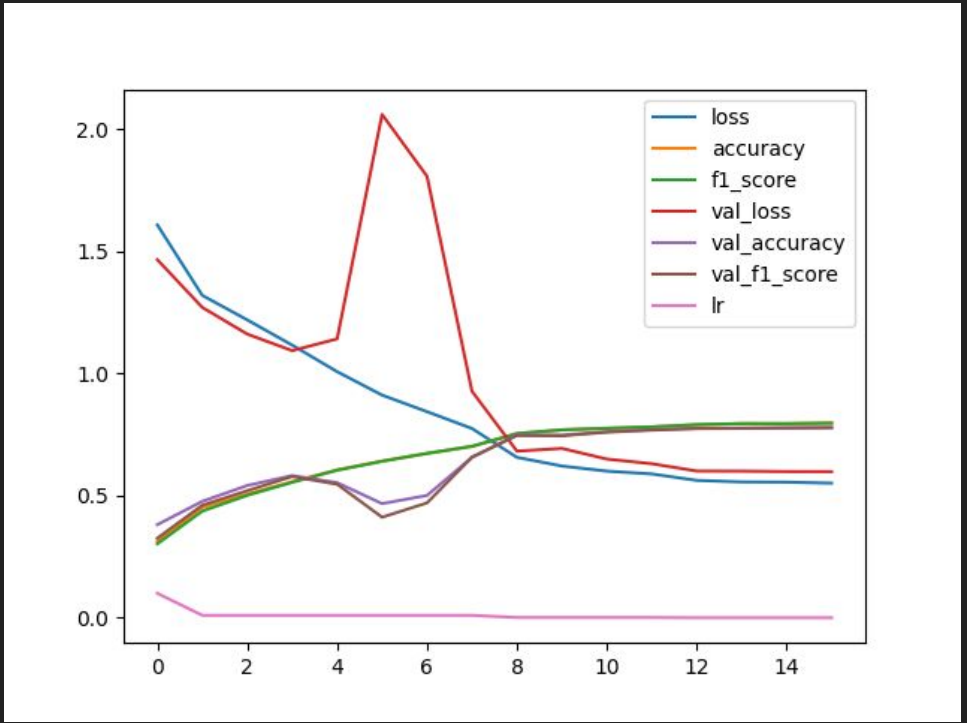
Images from previous slides are taken from here:

https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

Stage III: Ze Hong's Work: DenseNet

Diagram 1: DenseNet training history

- Loss: 0.6446
- Accuracy: 0.7686
- F1: 0.7696
- Early-stopped with 16 epochs
- Learning rate scheduler: 0.01 initial, 0.001 after 12 epochs, 0.0001 after 18
- Architecture explanation later



Stage III: Ying Jie’s Work

Architecture tested:

- AlexNet
    - Convolution layers
    - Dropout layers
- GoogLeNet
    - Inception module
- EfficientNet
    - Compound Scaling

- 20 epochs
- SGD Optimizer
- Simplified version due to size limit

AlexNet

- Similar to LeNet-5
- Replicated a simplified version of AlexNet architecture
- Reduced units and filters
- Reduce overfitting by dropout
- Data augmentation

Table 14-2. AlexNet architecture

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	–	1,000	–	–	–	Softmax
F10	Fully connected	–	4,096	–	–	–	ReLU
F9	Fully connected	–	4,096	–	–	–	ReLU
S8	Max pooling	256	6 × 6	3 × 3	2	valid	–
C7	Convolution	256	13 × 13	3 × 3	1	same	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	same	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	same	ReLU
S4	Max pooling	256	13 × 13	3 × 3	2	valid	–
C3	Convolution	256	27 × 27	5 × 5	1	same	ReLU
S2	Max pooling	96	27 × 27	3 × 3	2	valid	–
C1	Convolution	96	55 × 55	11 × 11	4	valid	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

```
# 5 Convolutional Layer
Conv2D(filters=16, kernel_size=(3, 3), strides=(2, 2), padding='valid', activation='relu', input_shape=(227, 227, 3)),
MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),

Conv2D(filters=32, kernel_size=(3, 3), padding='same', activation='relu'),
MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),

Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'),

Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'),

Conv2D(filters=32, kernel_size=(3, 3), padding='same', activation='relu'),
MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),

Flatten(),

# Fully Connected Layer
Dense(512, activation='relu'),
Dropout(0.25),

Dense(256, activation='relu'),
Dropout(0.25),

# Output Layer
Dense(5, activation='softmax'),
```

GoogLeNet

- Inception module
- Different kernel sizes allow capture patterns at different scales
- Concatenate all outputs in depth concatenation layer
- Able to capture complex patterns across channels
- Bottleneck layers (Reduce dimensionality)

```
def inception_module_simple(x, filters):
    path1 = Conv2D(filters[0], (1, 1), padding='same', activation='relu')(x)
    path2 = Conv2D(filters[1], (3, 3), padding='same', activation='relu')(path1)
    path3 = Conv2D(filters[2], (5, 5), padding='same', activation='relu')(path1)
    path4 = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(x)
    path4 = Conv2D(filters[3], (1, 1), padding='same', activation='relu')(path4)

    # Concatenate the inception paths
    return concatenate([path1, path2, path3, path4], axis=-1)
```

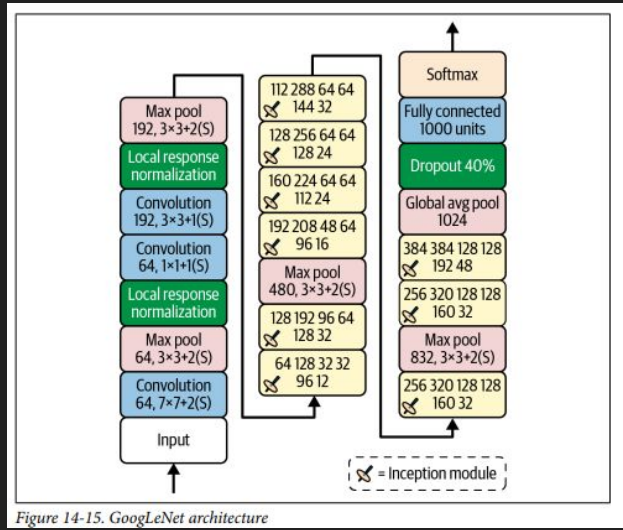


Figure 14-15. GoogLeNet architecture

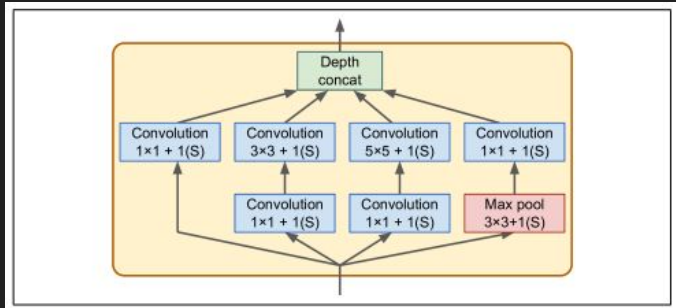


Figure 14-14. Inception module

EfficientNet

Compound Scaling

- Scaling up a baseline model for depth, width and image resolution
- Scale all three dimension of network
- High resolution lead to learning new complex patterns

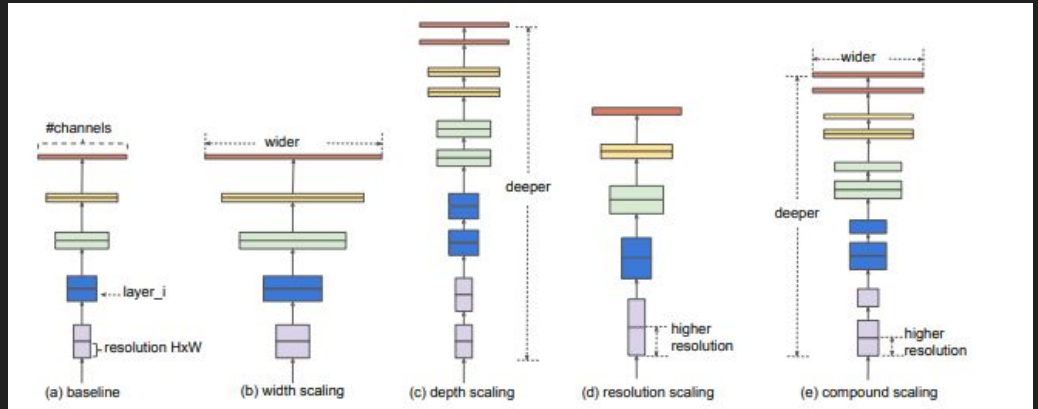


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

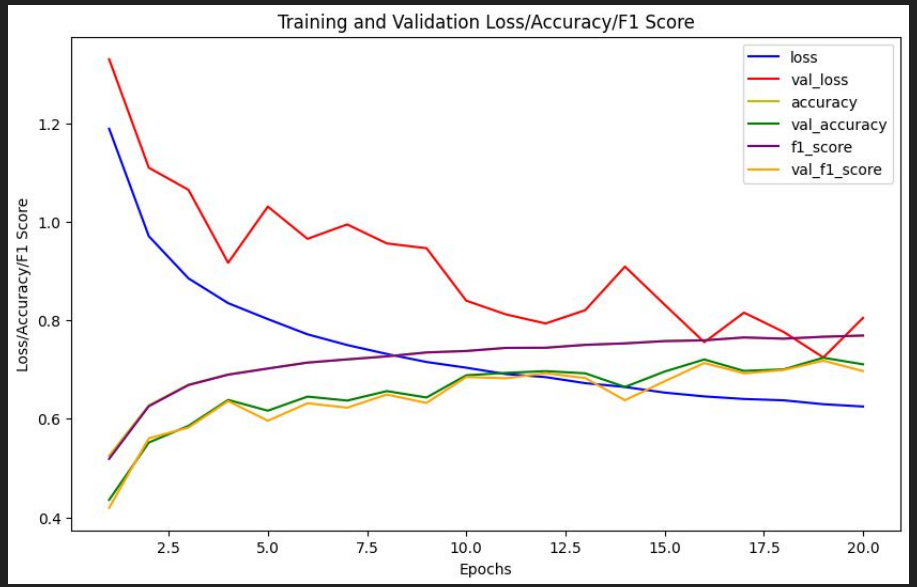
```
def simplified_efficientnet(input_shape, num_classes, width_coefficient, depth_coefficient, resolution_coefficient):
    def scale_filters(filters, width_coefficient):
        return max(1, int(filters * width_coefficient))

    def scale_repeats(repeats, depth_coefficient):
        return max(1, int(repeats * depth_coefficient))

    def scale_resolution(resolution, resolution_coefficient):
        return max(1, int(resolution * resolution_coefficient))
```

Stage III: Ying Jie's Work: EfficientNet

- Loss: 0.8151
- Accuracy: 0.7196
- F1-Score: 0.7040
- Best one out of the three
- Did not overfit as severely



Stage III: Model Architecture for Next Step

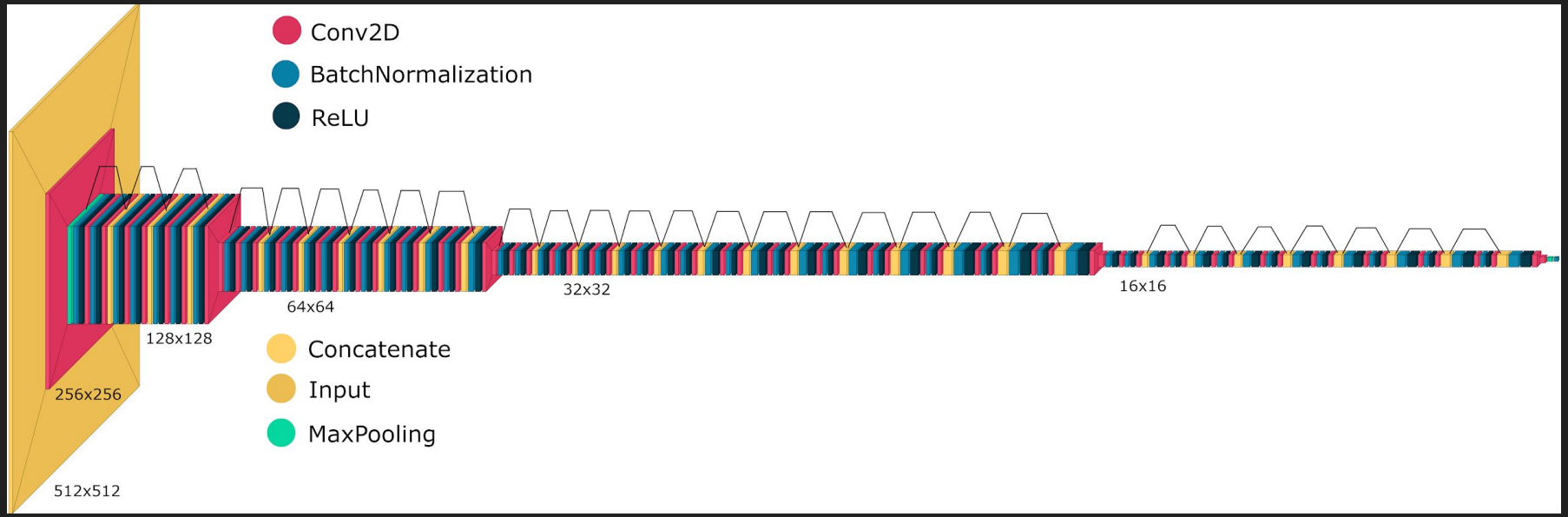


Diagram 2: Model architecture. Lines represent skip connections. Model has 2.4M weights. Architecture visualization was generated using visualkeras and edited using GIMP and Paint. Skip connection lines are drawn poorly; there should be a lot more of them.

Stage III: Model Architecture Discussion

- DenseNet based model
- Model consists of “dense stacks” each containing several residual blocks
  - Res-blocks use Conv2D to learn features
  - Each residual block implements skip connections
    - Each block connected to all previous blocks
  - Several blocks come one after another
- Residual blocks end in transition layers, see right
  - Uses Average Pooling to downscale layers
- Model uses 4 dense stacks
  - Stacks have length 3, 6, 12, 8
  - Length describes number of residual blocks in a stack

DenseNet: Residual Block

- Residual blocks implement skip connections
- DenseNet blocks connect to all previous blocks
  - ResNet only connects to previous residual block

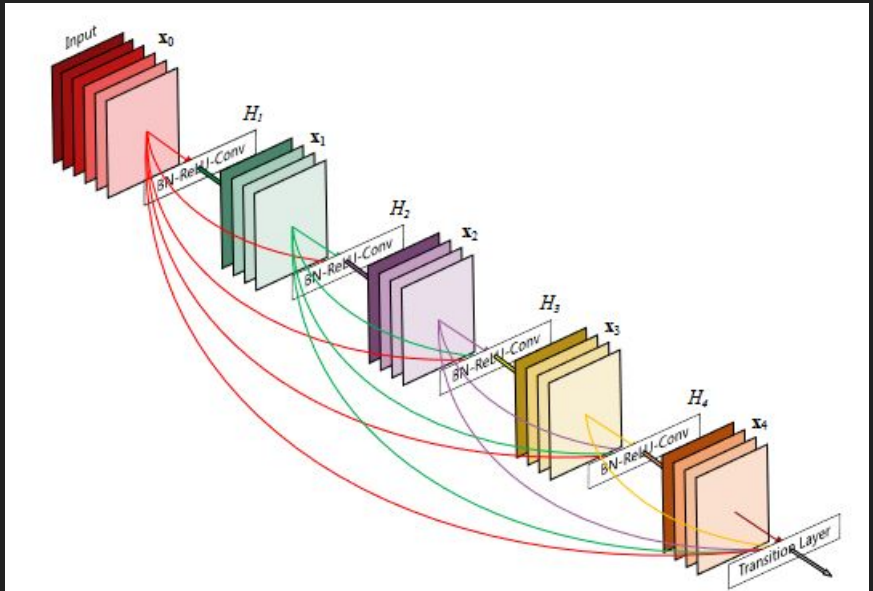


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Source: https://arxiv.org/pdf/1608.06993

Stage III: Reflection

Places for improvement

- Should have focused on only a couple of model architectures
  - We spread our efforts too thin
  - Could have freed up time for architecture tweaks and more in-depth testing
- Image cropping layer was a very bad move
  - Eliminated useful features from image, almost certainly reduced accuracy

Stage IV: Proposed Work

In the Stage III report we propose to do in Stage IV:

- Ze Hong: Activation functions, Gradient clipping
- Ying Jie: Optimizer, Regularization

We also proposed possible architecture tweaks, if there is time:

- Ze Hong Wu: modification of stack lengths

Stage IV: TFDatasets

- Encountered an issue with bad format TFDatasets in Stage III
- Labels for images did not match Real Life dataset labels
  - Write TFDatasets code generated labels by reading file directory using listdir
  - Auto-generated labels are in alpha order ['bell', 'dog', 'horse', 'house', 'tiger']
  - Labels for Real Life are ['dog', 'tiger', 'house', 'bell', 'horse']
- We fixed this at the start of Stage IV

```
# Get the list of all labels from the class directories in the dataset
# CHECK YOUR LABESL AND MAKE SURE THEY ARE CORRECT
# (the only file sin your data_path shoudl be the class directories)
labels = os.listdir(data_path)
```

Note: A Strange Image

Found this image during testing, re-discovered during slides building

- What category is this?
  - Bell? Dog? Horse? House? Tiger?

```
print("is loop running?")
image = batch[0]  # This gets the first image in the batch
print(type(image))

# The image tensor shape is (512, 512, 3), you can now visualize it
plt.imshow(image)
plt.show()
print(label[0])
```

```
is loop running?
<class 'tensorflow.python.framework.ops.EagerTensor'>
```


```
tf.Tensor([0. 1. 0. 0. 0.], shape=(5,), dtype=float32)
```

Stage IV: Ze Hong's work

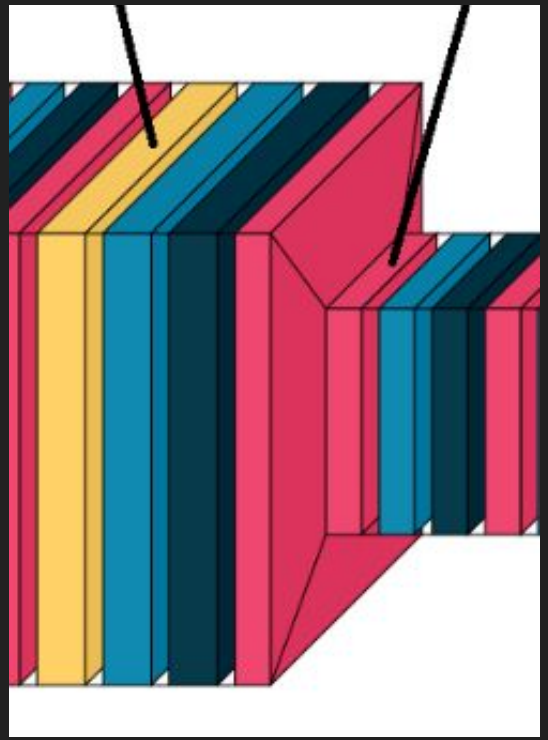
Experiments:

- Used Stage III submission as a baseline
- Trained for 24 epochs with SGD and checkpoints
- Changes:
  - Clipnorm: added clipnorm=1.0 to SGD
  - LeakyReLU: used aforementioned activation function
  - Compression factor: 0.5 to 0.66; explanation in next slide
  - Growth rate: 32 to 48; explanation in next slide
  - Learning rate: -50% lr instead of -90% in original scheduler

Quick Explanation: Compression Factor

Compression Factor (CF)

- Transition blocks reduce number of filters
- CF value describes the proportion of filters after block compared to before
  - DenseNet paper authors proposed using CF = 0.5
  - Transition block with CF = 0.5 receives 288 filters as input and outputs 144 filters



Quick Explanation: Growth Rate

Growth Rate (GR)

- Residual blocks concatenate input and output
  - Number of filters increase after each residual block
  - Total filters = input filters + residual block output filters
- GR describes filter increase amount after each residual block
  - DenseNet authors propose using GR = 32
  - Residual block takes in 112 filters input, outputs 32 filters, and concates
  - End result is 144 filters

Stage IV: Ze Hong's Work: Results

Table 1: Tools tested and results.

Compression model had best accuracy and was chosen for comparison.

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

Stage IV: Bugs?

“Removed transition layer, fixed learning rate bugs”

- Stage III submission model had a stray transition layer not specified in paper
- Learning rate initially planned to be static
  - Overwritten by a scheduler that wasn’t meant to get into final model
  - Scheduler removed during Ze Hong’s tests

Stage IV: Ying Jie’s Work

- DenseNet model as baseline from stage III
- Attempt modifications on:
    - Optimizer
    - Regularization(Dropout)
    - Increment base filter and units
    - Image shape

Stage IV: Ying Jie's Work: Results

- Full table of changes tested and experimental results.
- Dropout layer to prevent overfit fails
- Adam optimizer model had the best result

Baseline Model	Loss	Accuracy	F1_Score
my_densenet	0.5614	0.7959	0.7962

Initial Model	Model Name	Modification	Value	Loss	Accuracy	F1_Score
Baseline	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

Stage IV: Model Architecture for Next Step

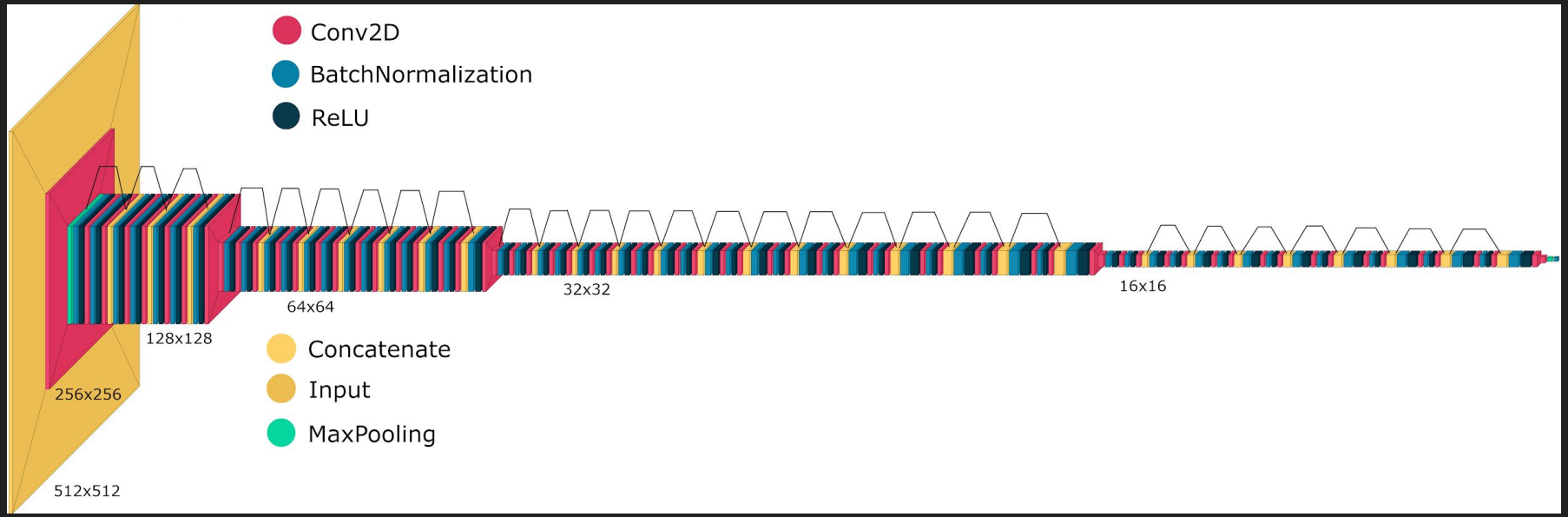


Diagram 2: Model architecture. Lines represent skip connections. Model has 2.4M weights. Architecture visualization was generated using visualkeras and edited using GIMP and Paint. Skip connection lines are drawn poorly; there should be a lot more of them.

Stage IV: Model Architecture Discussion

We went with Ying Jie’s model to advance into Stage V.

- Reason: it performed the “best” when judging by testing accuracy

Details:

- Adam optimizer with ClipNorm 1.0
  - Initial learning rate 0.01, reduction by 90% at 12 and 18 epochs
- Growth Rate = 32, Compression Factor = 0.5
- ReLU activation
- No change to layer types or count

Stage IV: Reflection

Places for improvement

- Once again, we spread efforts too thin
  - Activation functions: only tested Leaky ReLU, not others such as SELU and ELU
  - If we focused on fewer things, we could have tested more in depth

Stage V: Proposed Work

In the Stage IV report we proposed to do in Stage V:

- Both: combine best work from both team members into a new model
- Ying Jie: test compression factors of 0.75 and 1.0 (current CF is 0.66)
- Ze Hong: test a number of new learning rate schedulers
    - Details on next slide

Stage V: Ze Hong's Work

- We tried a few more than expected
- "Custom Schedulers" to be discussed in next slide
- Goal is to "gradually spiral into" the local minima

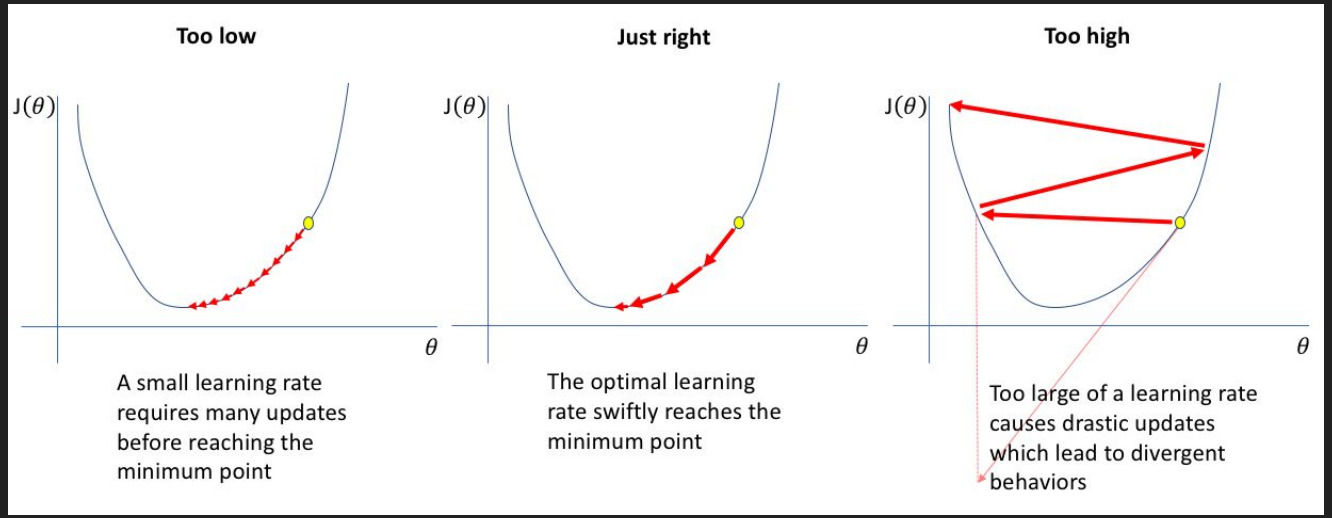
Table 3: Loss - Accuracy - F1 Macro values for various scheduler models

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

Spiraling into Minima: Visual Aid

Attempts to avoid “too high” and “too low” by gradually lowering lr over time

Source: https://www.jeremyjordan.me/nn-learning-rate/



Stage V: Ze Hong's Findings

Learning-5 model did best

- Uses Learning-1 scheduler
  - 0.01 lr, then 0.001, finally 0.0001
- Reduces compression ratio
  - 0.66 -> 0.5
- Reasons for choosing L5 not L1
  - Loss, Accuracy about similar
  - Much better F1 Macro

Table 3: Loss - Accuracy - F1 Macro values for various scheduler models

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

Stage V: Custom Schedulers

Custom schedulers that scale down lr gradually

- Didn’t quite work, but might have improved accuracy if tested further

```
num_epochs=24
for i in range(24):
    print(schedule_func4(i, 3))
```

```
0.01125
0.01
0.00875
0.0075
0.00625
0.005
0.00375
0.0025
0.00125
0.001
0.000875
0.00075
0.000625
0.0005
0.000375
0.00025
0.000125
0.0001
8.7e-05
7.5e-05
6.3e-05
5e-05
3.8e-05
2.5e-05
```

```
[ ] for idr in range(1, 25):
      print(schedule_func3(idr, 3))
```

```
0.01
0.009
0.008
0.007
0.006
0.005
0.0042
0.0036
0.003
0.0024
0.0018
0.0012
0.0011
0.001
0.0009
0.0008
0.0007
0.0006
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
```

Stage V: Ying Jie’s Work

- Compression factors
- Original: 0.66 (Later reduced to 0.5)
- Attempt modifications on:
    - 0.75
    - 1.00

Stage V: Ying Jie's Findings

- Final Results
- As compression factor increases accuracy and f1 score decreases accordingly
- Loss also increases as indication of worse performance

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

Stage V: Model

- Same model architecture as from Stage IV
- Same learning rate scheduler
  - Initial learning rate 0.01, reduction by 90% at 12 and 18 epochs
  - All other learning rates did not live up to it
- Trained for 48 epochs, no early stopping
- Adam optimizer with ClipNorm 1.0
- Growth Rate = 32, Compression Factor = 0.5

Final Reflections

Time management and scope

- Focusing on less things was a better idea, should have applied it earlier