Image Classification using Various CNNs

Jason Bard
Bavithra Lakshmanasamy
John Aguinaga
Justin Pham

Datasets

Rock-Paper-Scissors Classification

Objective: Classify if an image is Rock, Paper, or Scissors

https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors

- Description: Use images of Rock-Paper-Scissors gameplay to label pictures accordingly.

Data:

- 2188 images of individual hand signals against a green background
  - 726 Rocks, 710 Papers, 752 Scissors
- Stored in PNG format (300 x 200)
- Data size: 160* MB

- Task:
  - Build a classifier that successfully separates these three categories
  - Metric: accuracy, although precision and recall could be useful

- Resources:
  - Many notebooks associated with this dataset on Kaggle, specifically tackling this problem

*if each picture is only counted once. Each picture appears twice, making the total size 321 MB.

Natural Scenery Classifier

Objective: Predict image type based on input image

https://www.kaggle.com/datasets/puneet6060/intel-image-classification

- **Description:** *Use a machine learning algorithm to predict which category of scenery the provided image belongs to with highest possible accuracy*
- **Source:** *Kaggle (Dataset)*
- **Data:**
  - **What is a data point?** *Data point is a single 150 x 150 image of some natural scene*
  - **What are the types?** *There are 6 types of images: Building(s), Forest, Glacier, Mountain, Sea and Street. All images are of ".jpg" format and contained within separate folders by image type*
  - **How many instances?** *~ 24,000 images. Divided into: 14,034 for training, 3000 for testing and 7300 for prediction (?)*
    - **How large?** *399 MB total*
- **Task:** *Classification: predict which category an image will be*
  - **Metric:** *Accuracy: Determine total correct predictions over totals in given data.*
- **Resources:** *Kaggle users have performed challenge on dataset and provide explanations. Also, dataset was originally created by Intel as part of an image classification challenge (discontinued).*

Driver Drowsiness Detection System with Neural networks

Objective: build a drowsiness detection system that will detect that a person's eyes are closed for a few seconds.

https://www.kaggle.com/datasets/serenaraju/yawn-eye-dataset-new

- Description: Drowsiness detection system, build a model using deep neural network to classify images and detect if that person's eyes are closed for a few seconds or not
- Source: Kaggle
- Data:
- Data point: images with varied eye patterns like yawning, closed, opened.
- Instances: 2467 images in total which are of different dimensions and the format is of .jpg
- Size: 169MB
- Task: Classification (Classifier will categorize whether eyes are open or closed)
- Metric: Accuracy
- Resource:

  https://www.kaggle.com/datasets/serenaraju/yawn-eye-dataset-new

Neural Networks

Overview

- Wanted to get models close to 40 million parameters for each of them
- Using all models on each dataset, reporting results
  - Accuracy, precision, recall, F1

- Each of us used a different CNN for the project
  - VGGNet, AlexNet, DenseNet121, custom architecture

VGGNet

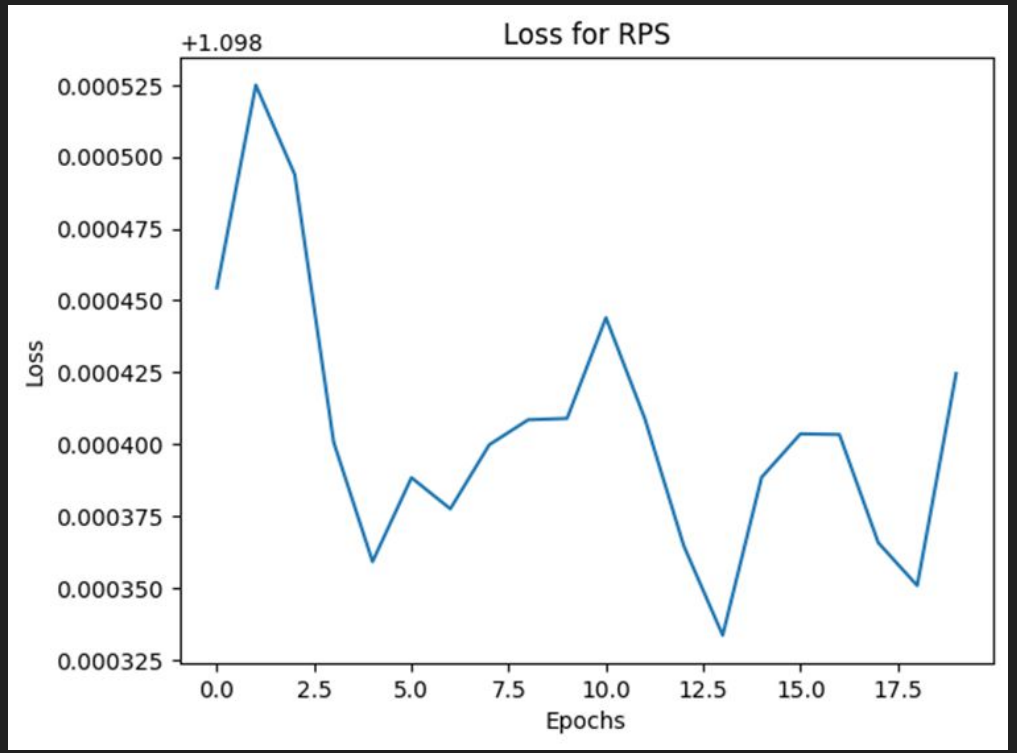
- Initially proposed by Simonyan and Zisserman in 2014
  - Works by passing through a stack of convolutional layers three times, max pooling in between each time
  - Flattens it out after this, feeding into a regular DNN

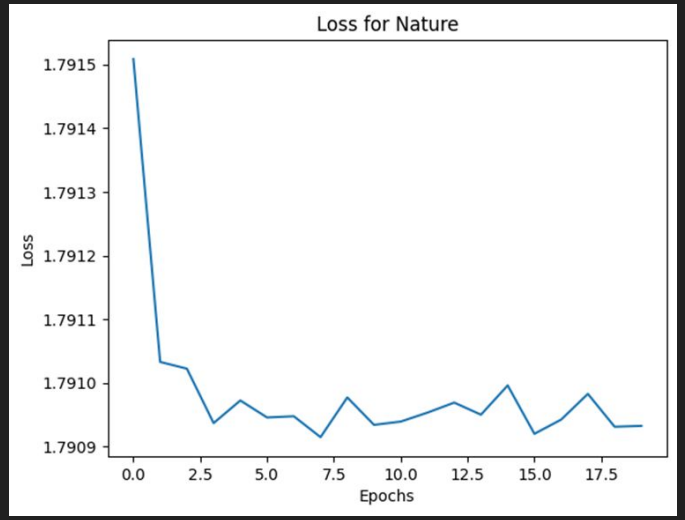
- Customizations
  - Added more dense layers
  - Changed last dense layer depending on dataset

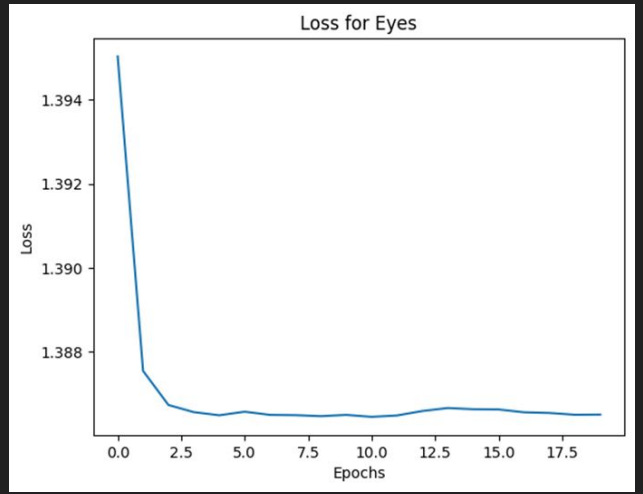
Results

Dataset	Accuracy	Precision	Recall	F1
RPS	0.337386	0.113829	0.337386	0.170227
Nature	0.175000	0.030625	0.175000	0.052128
Eyes	0.251732	0.063369	0.251732	0.101250

Loss Curves


Loss for RPS
+1.098
Loss
0.000525
0.000500
0.000475
0.000450
0.000425
0.000400
0.000375
0.000350
0.000325
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs


Loss for Nature
Loss
1.7915
1.7914
1.7913
1.7912
1.7911
1.7910
1.7909
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs


Loss for Eyes
Loss
1.394
1.392
1.390
1.388
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

AlexNet

- AlexNet is a Convolutional Neural Network (CNN) architecture that was made by Alex Krizhevsky, along with Ilya Sutskever and Geoffrey Hinton. It was used and popularized after the ImageNet Large Scale Visual Recognition Challenge in September of 2012.
- Why did AlexNet perform well for its time?
    - The Relu function: Enabled the CNN to be trained faster than tanh and sigmoid-based networks.
    - Local Response Normalization: The effect of active neurons on its surrounding neurons.
    - Dropout: For some random layer of neurons, some neurons are “shut off” essentially putting on the brakes. This effect helps prevent overfitting of Neural networks.
- Customizations: Changed the values in the 2D Convolution layers (Number of filters, Kernel size). Helped reach target for number of parameters.

AlexNet Results

AlexNet on Scenery

Metric	Value
Accuracy	0.764
Precision	0.796
Recall	0.764
F1 Score	0.763

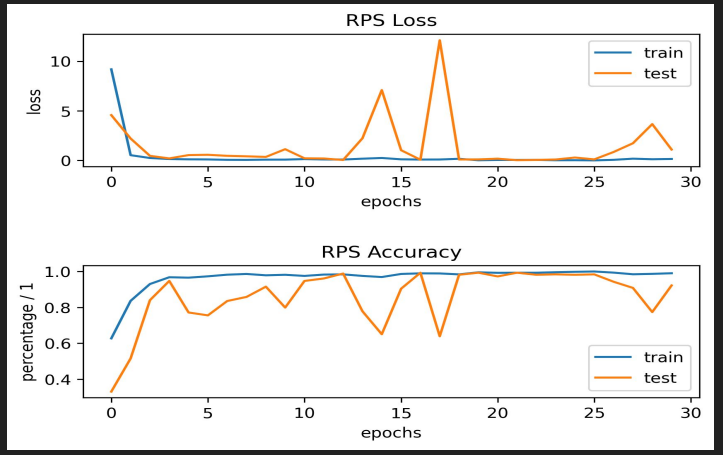
AlexNet on RPS

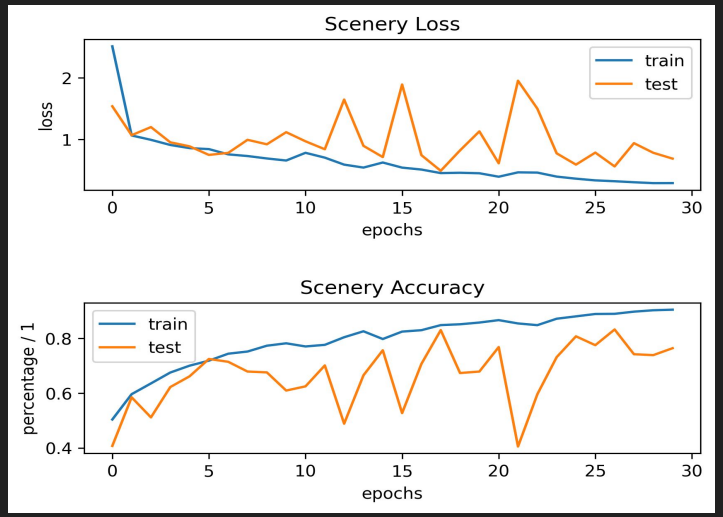
Metric	Value
Accuracy	0.922
Precision	0.933
Recall	0.922
F1 Score	0.920

AlexNet on Eyes

Metric	Value
Accuracy	0.776
Precision	0.849
Recall	0.776
F1 Score	0.736

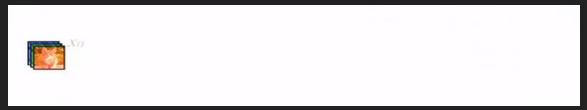
AlexNet
Loss Curves





DenseNet121

- DenseNet architecture is jointly invented by Cornwell University, Tsinghua University and Facebook AI Research (FAIR).
- DenseNet is a Convolutional Neural Network which connects each layer to every other layer directly in a feed-forward fashion.
- Types: DenseNet121, 169, 201 and so on.
- In DenseNet-121, there are four dense blocks, each containing a specific number of layers (6, 12, 24, and 16 layers, respectively), for a total of 121 layers in the entire network. These layers can be modified as per our needs.
- Each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers.
- Steps:



- Conv layers(extracts features)–––Denseblock 1(Previous layers concatenated)—--Transition layer()-----Densebl 2()
- Why this model? - is designed to address the vanishing gradient problem
- - Reduced Overfitting
- Customizations: - increased the growth rate of the dense blocks in the network
- - changed the number of layers within each dense block
- - changed the number of filters in the transition layers

Results

RPS:

Metric	Value
Accuracy	0.998
Precision	0.998
Recall	0.998
F1 Score	0.998

Eyes:

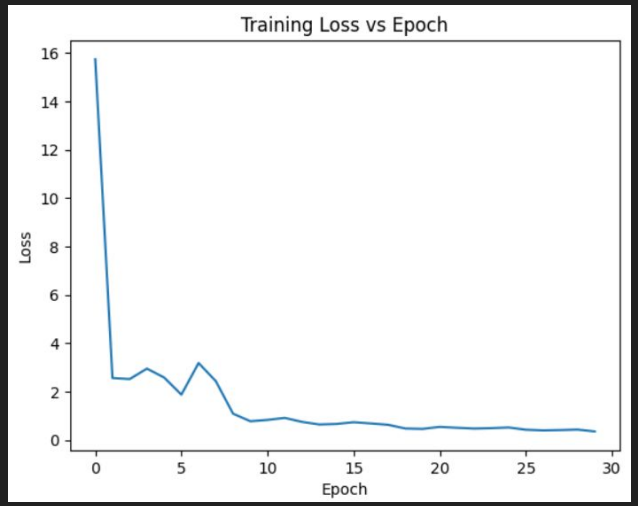
Metric	Value
Accuracy	0.769
Precision	0.774
Recall	0.769
F1 Score	0.770

Nature:

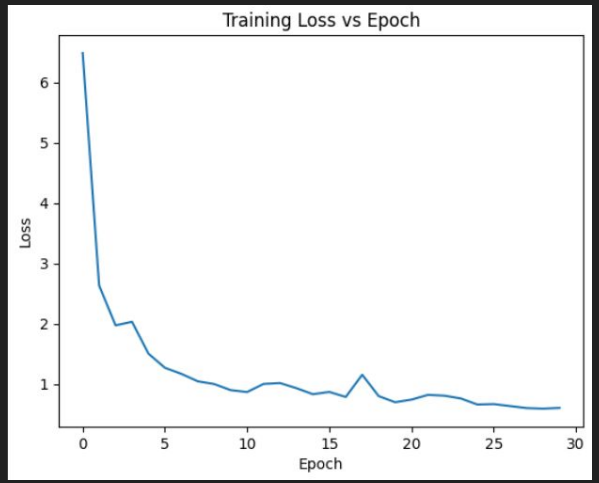
Metric	Value
Accuracy	0.752
Precision	0.773
Recall	0.752
F1 Score	0.748

Loss Curves:

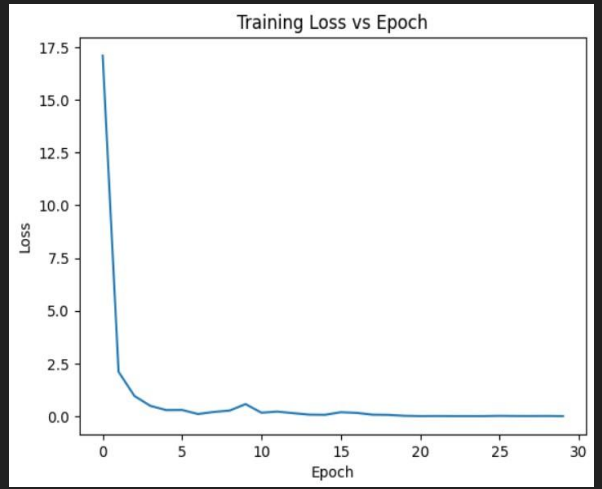
Eyes:



Nature:



RPS:



Custom Architecture

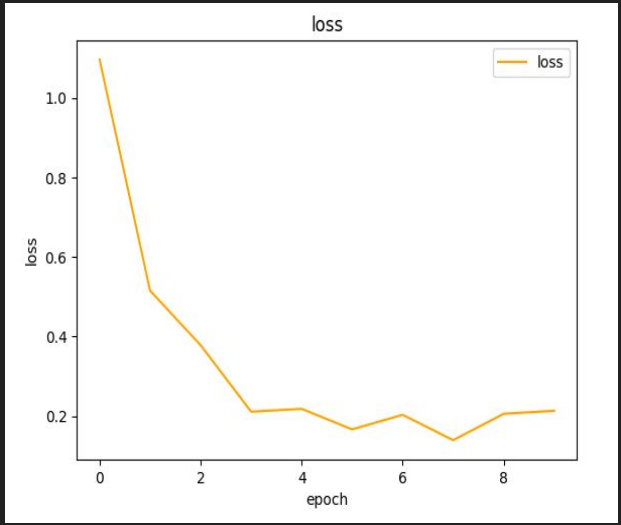
- With modifications from my individual project….
  - Used 2 stacks of convolutional layers + max pooling
  - Flattens out and feeds into 2 dense + dropout
  - To compile, the loss function used: categorical_crossentropy, optimizer: adam, and metrics: accuracy

Results: Custom Architecture

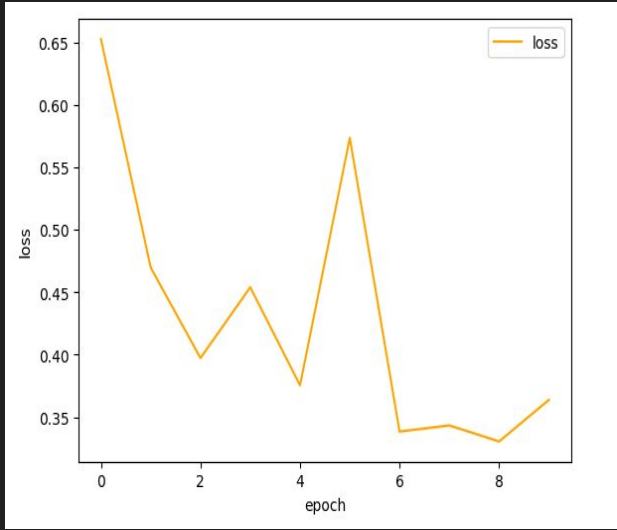
Data	Accuracy	Recall	Precision	F1
Nature	0.673333	0.675684	0.680738	0.675810
RPS	0.321918	0.333333	0.107306	0.162349
Eyes	0.849885	0.847996	0.867015	0.844401

Loss Curves: Custom Architectures

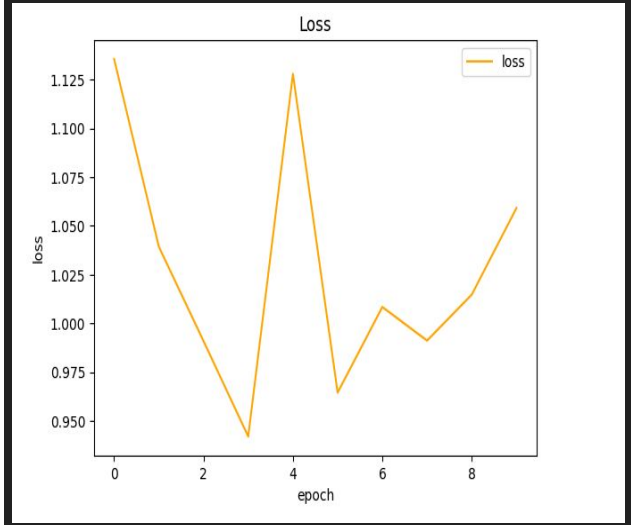
RPS:



Eyes:



Nature:



Conclusions

- Unsure about what happened with VGGNet.
  - All other datasets had accuracy that was substantially better than random guessing

- All other models are effective especially on the RPS dataset. The pixel values are more uniform.

- Densenet works better for RPS with 0.998 compared to Eyes and Nature.
- Alexnet and Dense121 worked best for Nature with ~0.76 accuracy.
- Custom Arch worked best for eyes with ~0.84 accuracy.

Challenges

- Creating the model to get around ~40 million parameters
- Resources and computing time
- Scheduling times to meet
- Coordinating the `datasetload.py` module

References

https://towardsdatascience.com/implementing-alexnet-cnn-architecture-using-tensorflow-2-0-and-keras-2113e090ad98
https://medium.com/analytics-vidhya/vggnet-architecture-explained-e5c7318aa5b6
https://www.kaggle.com/code/blurredmachine/alexnet-architecture-a-complete-guide/notebook
https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c
https://www.kaggle.com/code/dansbecker/deep-learning-from-scratch-daily
https://arxiv.org/abs/1608.06993
https://keras.io/api/applications/densenet/
https://iq.opengenus.org/architecture-of-densenet121/
https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803