

Classifying Brain Hemorrhages


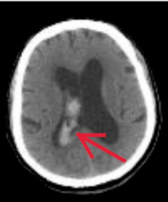

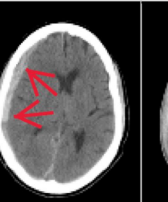



Kevin Ciardelli
Alicia Doung
Saskriti Neupane



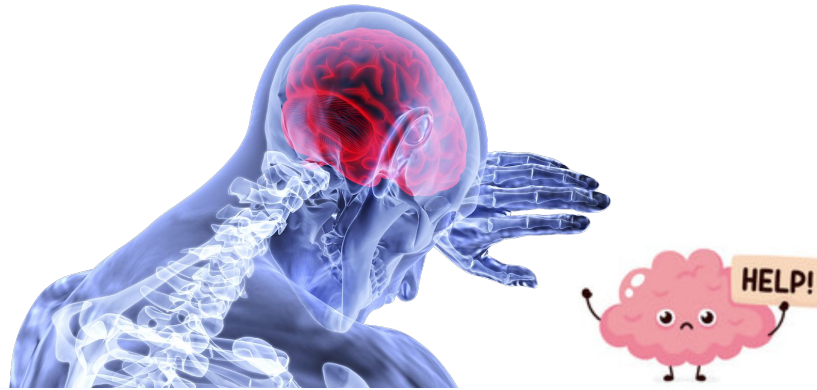
The task

- Obtain data set pertaining several types of hemorrhages
- Find source code that deals with cleaning 2D Images
- Find a model that can classify hemorrhages within a CT Scan
- Find a way to improve this model

	Intraparenchymal	Intraventricular	Subarachnoid	Subdural	Epidural
Location	Inside of the brain	Inside of the ventricle	Between the arachnoid and the pia mater	Between the Dura and the arachnoid	Between the dura and the skull
Imaging					
Mechanism	High blood pressure, trauma, arteriovenous malformation, tumor, etc	Can be associated with both intraparenchymal and subarachnoid hemorrhages	Rupture of aneurysms or arteriovenous malformations or trauma	Trauma	Trauma or after surgery
Source	Arterial or venous	Arterial or venous	Predominantly arterial	Venous (bridging veins)	Arterial
Shape	Typically rounded	Conforms to ventricular shape	Tracks along the sulci and fissures	Crescent	Lentiform
Presentation	Acute (sudden onset of headache, nausea, vomiting)	Acute (sudden onset of headache, nausea, vomiting)	Acute (worst headache of life)	May be insidious (worsening headache)	Acute (skull fracture and altered mental status)

Motivation

- Intracranial Hemorrhages (ICH), or bleeding within the skull, is one of the top five causes of fatal health complications that accounts for approximately 10 percent of strokes in the United State
- ICH varies in severity and impact, depending on factors like the location, size, and type of hemorrhage
- Traditional diagnostic methods are complex and time-consuming which is undesirable since rapid diagnosis is critical for the health, or even survival, of patients
- Automating the preliminary assessment of intracranial hemorrhages may expedite the diagnostic process and reduce human error, enabling doctors to make faster, more accurate decisions



Previous Work / Challenges



Previous Works

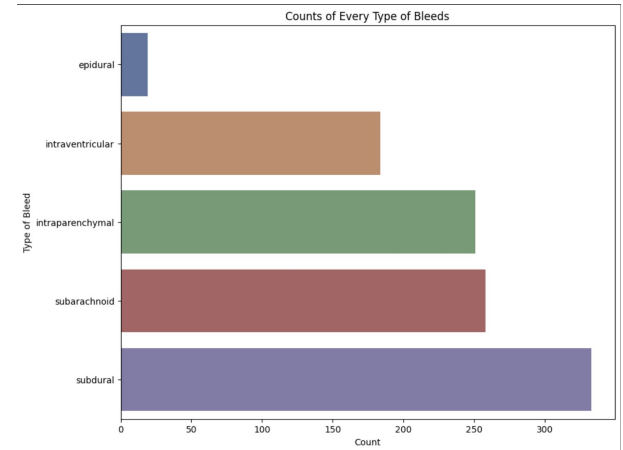
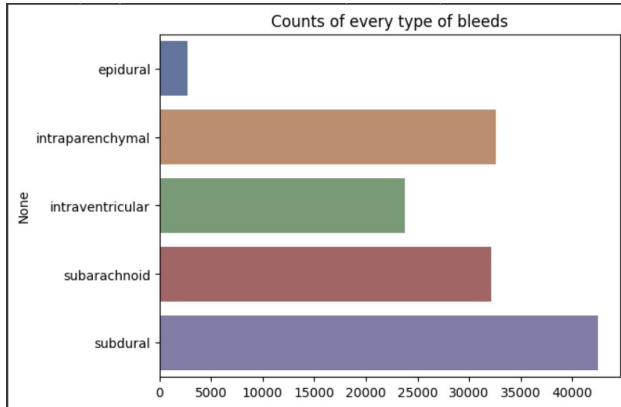
- Naive Bayes, K-Means clustering, image segmentation, and image thresholding
 - Issues: Due to the non-uniform illumination and interference of similar objects such as blood vessels, to detect the hemorrhages by using color information alone is not robust enough. Not only the classification accuracy but also the segmented boundaries of hemorrhages are not very desirable.
- Deep Learning: CNN
 - Strengths: Can model the intricate variations and subtleties in different hemorrhage types, combined with the ability to leverage 3D spatial contexts from medical volumes. It enables end-to-end multi-class learning, jointly optimizing feature extraction and classification in a unified framework.

Challenges

- Converting identification to classification
- Data cleaning

Dataset

- Supplied by four universities: Stanford, Thomas Jefferson, UHT, UNIFESP
- Consists of 194,082 CT scans varying in classification
- Non-evenly split
- Diluted to 1,500 scans for time-efficiency purposes
- 1238 test, 262 validation



Baseline Model - FastAi / CNN

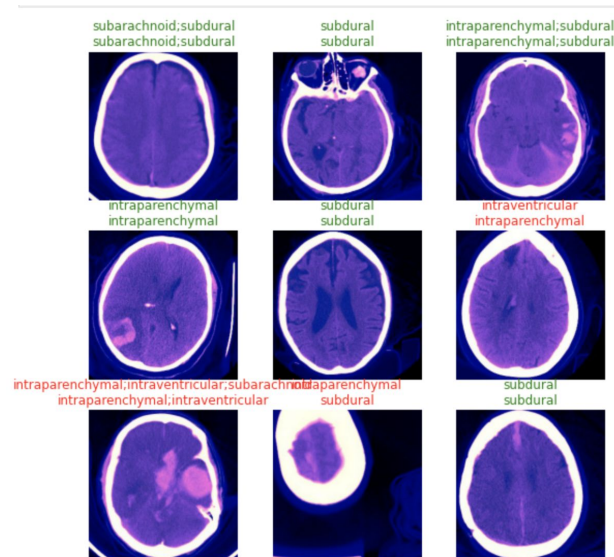
FastAi

- Deep learning library built on top of pyTorch; it includes some pretrained models like ResNet, DenseNet
- Automates much of the boilerplate code required in data preparation and augmentation. It provides comprehensive support for automatically handling typical tasks in processing images, text, tabular data, and more.

CNN

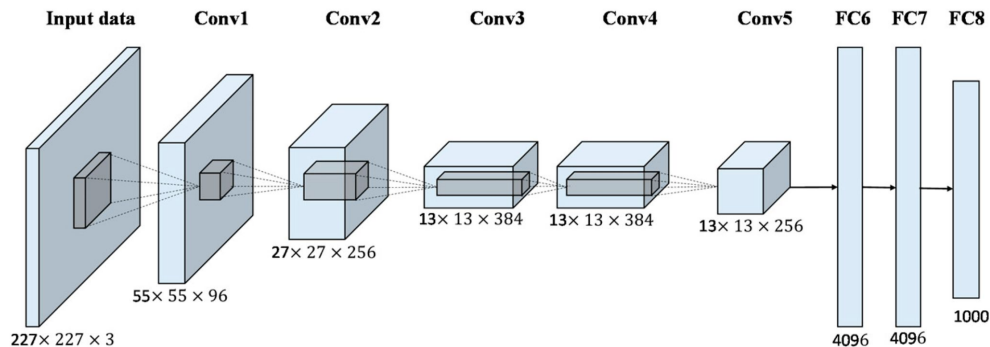
- Using a baseline model of Resnet18
- Binary Cross-Entropy Loss Function instead of categorical

epoch	train_loss	valid_loss	accuracy_multi	time
0	0.664726	0.739535	0.650000	05:29
1	0.575483	0.689729	0.676316	05:29

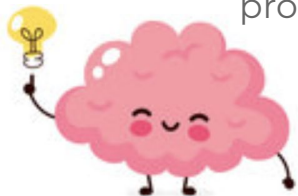


Alexnet Initial Results

- Duplicate the gray-scale images to fit the 3 channels needed
- Brush up
 - Conv. layers depict low-level edges and textures which gets more specific as it continues through more layers
- Fully connected layers integrate
- Final output gives probabilities for each neuron



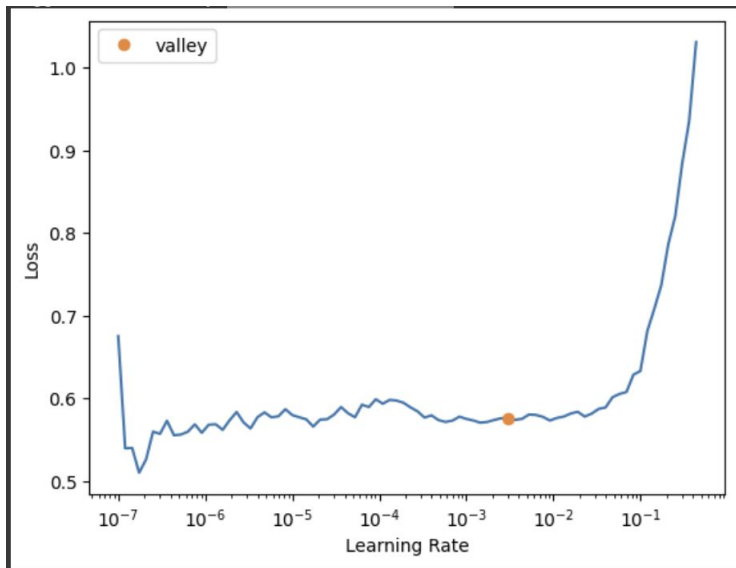
epoch	train_loss	valid_loss	accuracy_multi	time
0	0.363643	0.554391	0.726316	01:42
1	0.338968	0.555094	0.739474	02:01



AlexNet - Optimizing with Learn Rate

Leslie Learner's Rate

- Allows us to train as fast as possible without running into instability
- Increases the learning rate from a lower bound to an upper bound over several iterations while training on the dataset.
- Monitors the loss and plots it against learning rate
- The ideal learning rate is often chosen from a point where the loss starts to decrease and before it becomes unstable or increases sharply



epoch	train_loss	valid_loss	accuracy	time
0	0.472959	0.658210	0.723333	01:18
1	0.508787	0.589581	0.740000	01:28



Resnet-18 to Resnet-34

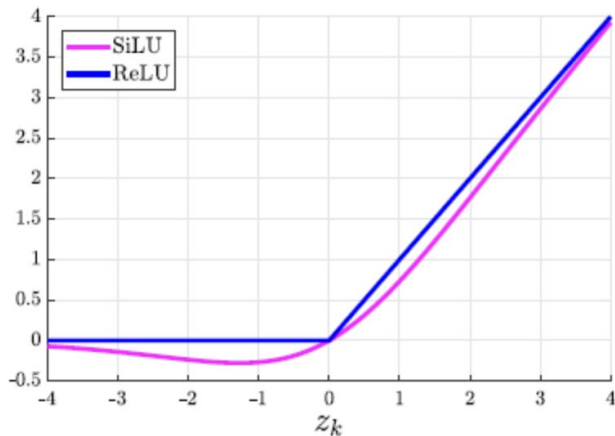
- Difference lies in the number and configuration of residual blocks within each stage
- ResNet-18 each stage comprises 2 blocks. Each block has two layers of 3x3 convolutions
- Depth in ResNet-34 allows it to learn more complex patterns and finer details in the data

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet: SiLU instead of ReLU



Silu (Sigmoid Linear Unit)



$$\text{SiLU}(x) = x \left(\frac{1}{1 + e^{-x}} \right)$$

- ReLU: Neurons output zeros regardless of the input if their weights adjust to always produce negative sums. No gradient flows through them during backpropagation
- SiLU: Maintains a self-normalizing property as it pushes data through multiple layers without losing the mean and variance
 - Deals with the vanishing gradient problem
- Activation Function is calculated by multiplying the input value (x) by the sigmoid of that same input value

SiLU Results

Initial Result:

epoch	train_loss	valid_loss	error_rate	time
0	1.256667	0.881993	0.283262	11:52
1	0.737067	0.622541	0.248927	11:59
2	0.473540	0.583938	0.261803	11:58



Result:

epoch	train_loss	valid_loss	error_rate	time
0	1.028587	0.882539	0.357500	20:12

- Unfortunately with SiLU, the model performed worse
 - Increase in validation loss and error rate

```
#SiLU
import torch.nn as nn
import torchvision.models as models

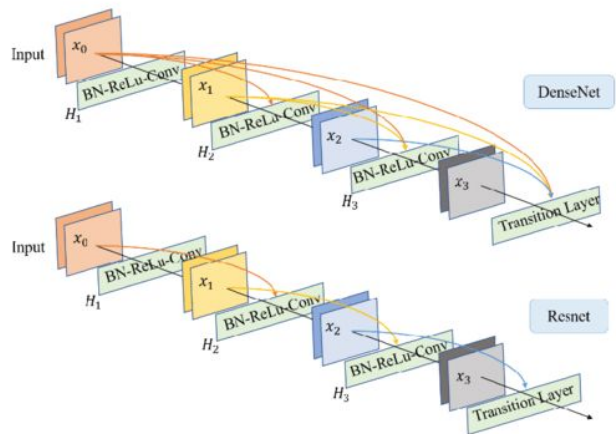
def resnet34_silu(pretrained=True):
    # Load the pretrained ResNet34 model
    model = models.resnet34(pretrained=pretrained)

    # Function to replace ReLU with SiLU
    def replace_relu_with_silu(module):
        for name, child in module.named_children():
            if isinstance(child, nn.ReLU):
                setattr(module, name, nn.SiLU())
            else:
                replace_relu_with_silu(child)

    # Replace all ReLU activations with SiLU
    replace_relu_with_silu(model)
    return model
```

DenseNet Initial Results

epoch	train_loss	valid_loss	accuracy	time
0	0.878568	0.614177	0.736667	12:53
1	0.641335	0.471099	0.786667	12:33

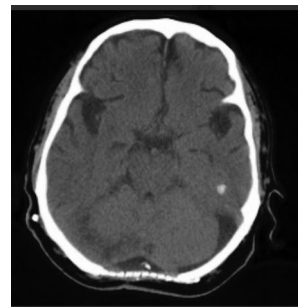


- In DenseNet161, each layer (161) receives input from all previous layers and passes its feature maps to all subsequent layers
- Composed of dense blocks, where each block contains multiple layers, each connected to every other layer
- Benefits:
 - improved information flow
 - mitigates the vanishing gradient problem
 - enhances feature reuse



Introducing CutMix

- Data augmentation technique that involves cutting a rectangular patch from one image and pasting it onto another
- Used to create additional training samples to:
 - improve the diversity of the training dataset
 - reduce overfitting
 - reduce reliance on specific features
- Overall, can enhance the robustness and accuracy of models during the training process



Original Image

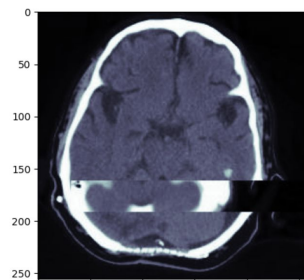


Image modified with
CutMix

```
import os
import random
from PIL import Image

def cutmix_image(source_folder, destination_folder, alpha=1.0):
    # Ensure the destination folder exists
    os.makedirs(destination_folder, exist_ok=True)

    # List all images in the source folder
    images = [os.path.join(source_folder, f) for f in os.listdir(source_folder) if f.endswith(('.png', '.jpg', '.jpeg'))]
    total_images = len(images)

    for idx, image_path in enumerate(images):
        # Open the base image
        img1 = Image.open(image_path)
        # Randomly select another image
        img2 = Image.open(random.choice(images))

        # Generate a random rectangular mask
        width, height = img1.size
        rx, ry = random.randrange(0, width), random.randrange(0, height)
        rw, rh = int(random.gauss(width / 2, width / alpha)), int(random.gauss(height / 2, height / alpha))
        rw, rh = max(1, min(rw, width - rx)), max(1, min(rh, height - ry))

        # Create a new image by blending img1 and the patch from img2
        patch = img2.crop((rx, ry, rx + rw, ry + rh))
        img1.paste(patch, (rx, ry))

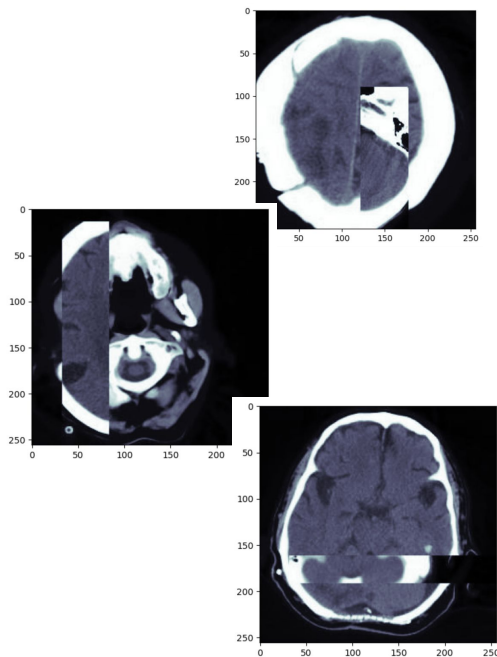
        # Save the new image with the same original name in the destination folder
        original_name = os.path.basename(image_path)
        img1.save(os.path.join(destination_folder, original_name))

        # Print progress
        print(f'Processed {idx + 1}/{total_images} images.')

# Example usage
source_folder = 'shorter_data'
destination_folder = 'shorter_data/mixed2'
cutmix_image(source_folder, destination_folder)
```

DenseNet + CutMix

- Enhanced robustness to noise and perturbations in the input data
- blends different images (form of regularization) encouraging the model to learn more general features instead of memorizing specific ones
- augments datasets by generating new examples/variations during training through mixing



epoch	train_loss	valid_loss	accuracy	time
0	0.743620	0.510875	0.726667	12:24
1	0.540349	0.478400	0.753333	12:39

Future

- Comparing all 3 models:
 - moving forward with Alexnet due to efficiency of accuracy
- Optimization:
 - Increase training epochs
 - Continue to use optimal learning rates
- Data:
 - Test Alexnet with CutMix
 - Further augment data using transpositions
 - Use more of the dataset