Brain Hemorrhage Classification of CT Scan Images Using CNNs

Matrix Methods in Data Analysis & Machine Learning

Team 2: Josue Antonio, Theodore Hattenbach,
Mara Hubelbank, Sahil Sangwan, and Yinglin Wang

Project Introduction

- **Goal:** To train a Convolutional Neural Network (CNN) model for image classification of CT scan images.
- **Data Source:** Zeta Surgical
- **Application:** When provided with an unlabeled brain CT image, the model will show us an accurate prediction of the class that has bleeding.

	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)

Methodology

1. Load the labeled dataset.
   - Use a random sample of 1000 for intraventricular class.
2. Convert the images to grayscale.
3. Split the dataset into training, testing, and validation sets (80, 15, 5).
4. Down-sample by factor of 16.
   - (512, 512) -> (128, 128)
5. Analyze the results on CNN, testing the performance using various optimizers.
   - Model structure kept constant.
   - 3 epochs due to hardware constraints.

Implementation

CNN Model Design

Keras Model Structure

- First model had 1.8 million parameters
  - Reduced complexity
- Input shape of (128, 128)
- 6 classes (5 classes plus multi)
- Drop-out: 50%

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 57600)             0
_________________________________________________________________
dense (Dense)                (None, 16)                921616
_________________________________________________________________
dropout (Dropout)            (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 102
=================================================================
Total params: 940,534
Trainable params: 940,534
Non-trainable params: 0
_________________________________________________________________
```

```python
model_cnn_adam = keras.models.Sequential()
model_cnn_adam.add(keras.layers.Conv2D(32, (3,3), padding = 'valid', input_shape=(128,128,1), activation = 'relu'))
model_cnn_adam.add(keras.layers.MaxPooling2D((2,2)))
model_cnn_adam.add(keras.layers.Conv2D(64, (3,3), padding = 'valid', activation = 'relu'))
model_cnn_adam.add(keras.layers.MaxPooling2D((2,2)))
model_cnn_adam.add(keras.layers.Flatten())
model_cnn_adam.add(keras.layers.Dense(16))
model_cnn_adam.add(keras.layers.Dropout(0.5))
model_cnn_adam.add(keras.layers.Dense(6))
```

Model Results

```
model_cnn_adam.evaluate(X_test, yds_test)
```

```
423/423 [==============================] - 28s 66ms/step - loss: 0.3243 - accuracy: 0.9799
[0.3242693841457367, 0.9798816442489624]
```

```
model_cnn_sgd.evaluate(X_test, yds_test)
```

```
423/423 [==============================] - 18s 43ms/step - loss: 0.3243 - accuracy: 0.9799
[0.3242693841457367, 0.9798816442489624]
```

```
model_cnn_adamax.evaluate(X_test, yds_test)
```

```
423/423 [==============================] - 19s 44ms/step - loss: 0.1120 - accuracy: 0.9393
[0.11202788352966309, 0.9393491148948669]
```

```
model_cnn_adadelta.evaluate(X_test, yds_test)
```

```
423/423 [==============================] - 18s 43ms/step - loss: 0.7936 - accuracy: 0.9385
[0.7935603260993958, 0.9385355114936829]
```

```
model_cnn_adagrad.evaluate(X_test, yds_test)
```

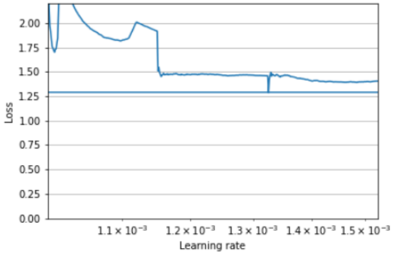
```
423/423 [==============================] - 18s 43ms/step - loss: 0.1121 - accuracy: 0.9375
[0.1120600700378418, 0.9375]
```

```
model_cnn_nadam.evaluate(X_test, yds_test)
```

```
423/423 [==============================] - 18s 42ms/step - loss: 0.1236 - accuracy: 0.9375
[0.1236410140991211, 0.9375]
```

```
# finding best learning rate
plt.plot(expon_lr_cnn_sgd.rates, expon_lr_cnn_sgd.losses)
plt.gca().set_xscale('log')
plt.hlines(min(expon_lr_cnn_sgd.losses), min(expon_lr_cnn_sgd.rates), max(expon_lr_cnn_sgd.rates))
plt.axis([min(expon_lr_cnn_sgd.rates), max(expon_lr_cnn_sgd.rates), 0, expon_lr_cnn_sgd.losses[0]])
plt.grid()
plt.xlabel("Learning rate")
plt.ylabel("Loss")
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
min(expon_lr_cnn_sgd.losses)
```

```
1.286063313484192
```

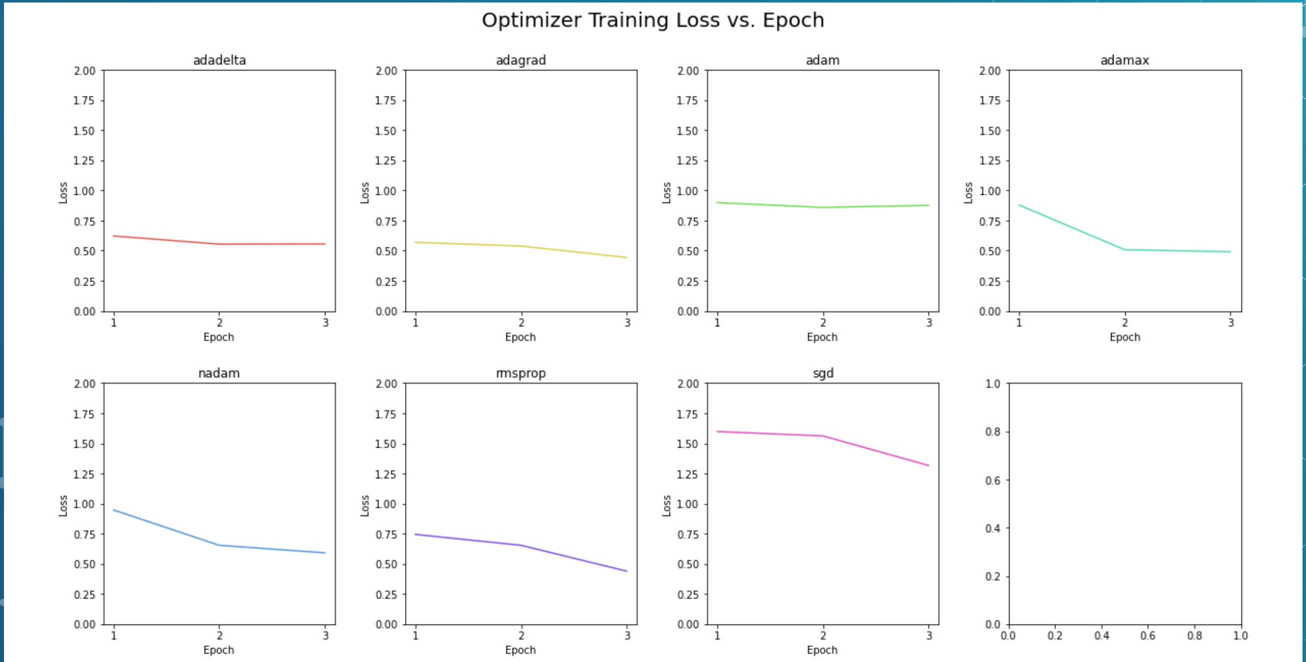
```
expon_lr_cnn_sgd.rates[np.argmin(expon_lr_cnn_sgd.losses)]
```

```
0.0013256102
```

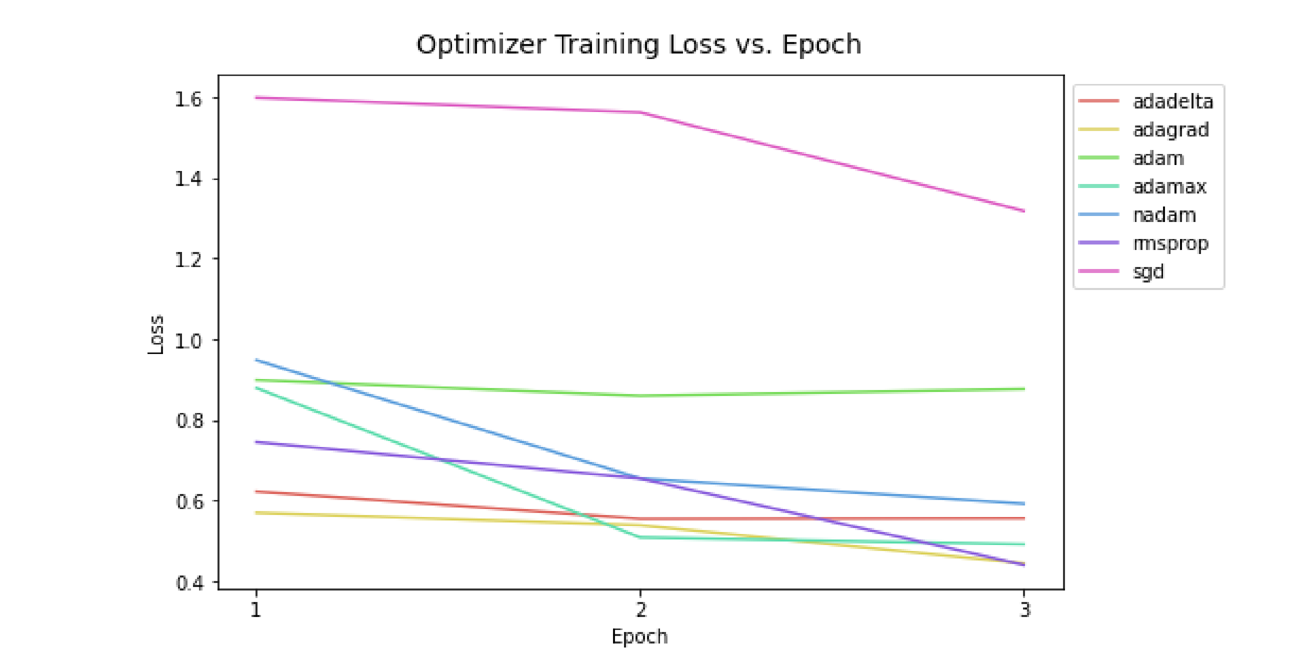
Data Visualization

Training & Validation

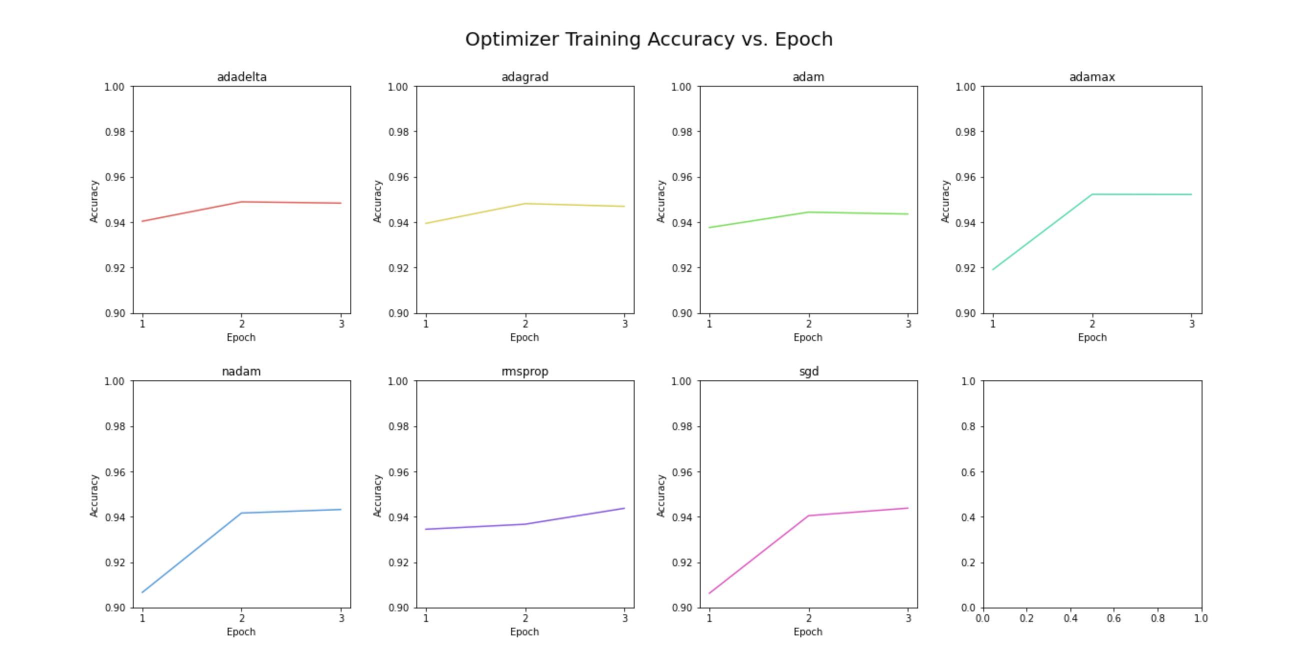
Loss vs. Epoch by Optimizer



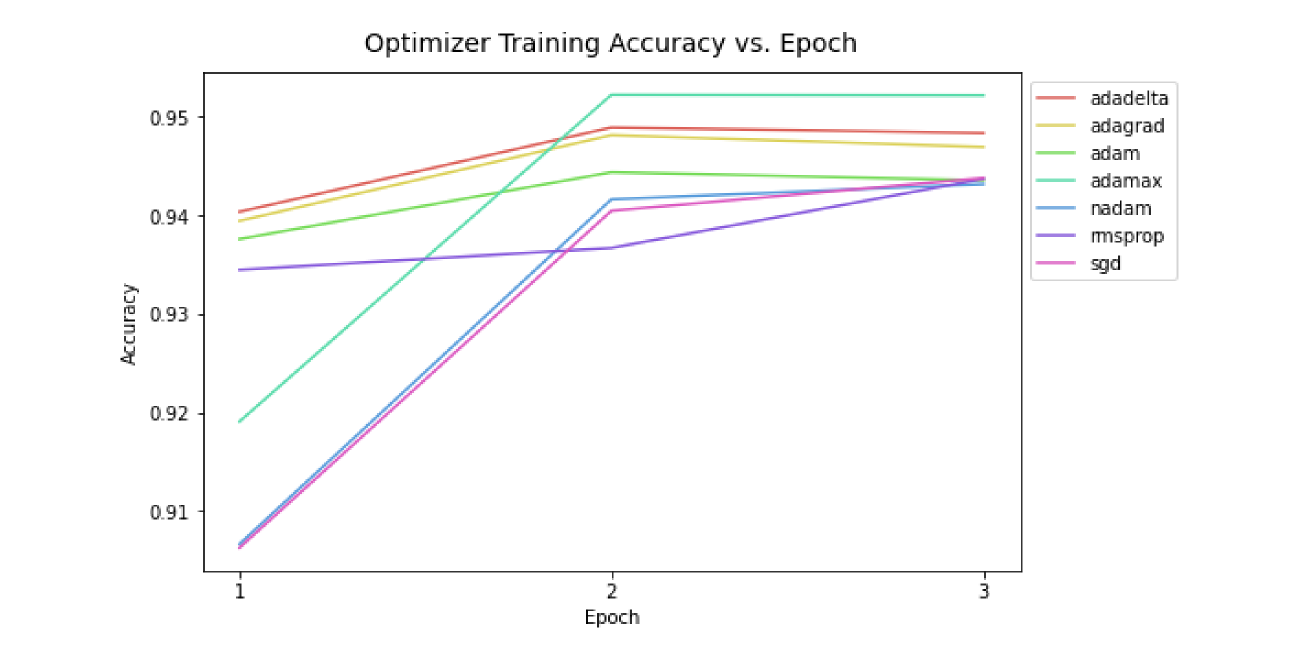
Loss vs. Epoch, All Optimizers



Accuracy vs. Epoch by Optimizer



Accuracy vs. Epoch, All Optimizers

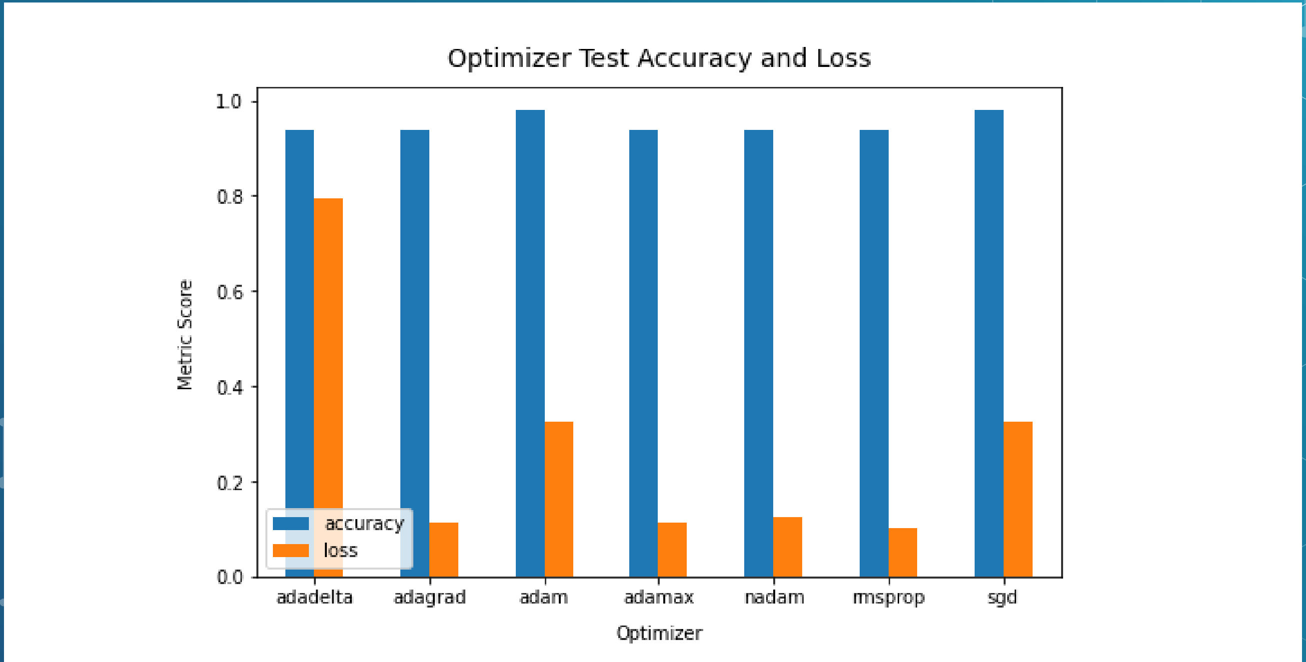


Accuracy vs. Val Accuracy Over Epoch, by Optimizer

Data Visualization

Testing Dataset Performance

Test Accuracy and Loss by Optimizer



Conclusion

Conclusion & Reflection

What we achieved:

- Best optimizer found to be Adam.
    - Training & training accuracy: 0.98
- SGD is close, but loss is slightly larger across multiple runs.

What we can improve on:

- Testing and validation accuracy is higher than training accuracy.
- Train more epochs using more data.
- Expand the scope to perform image segmentation on hemorrhages.
- Active Contour segmentation (supervised)
- Simple Linear Iterative Clustering (unsupervised)