Convolutional Neural Network Model for the Detection and Classification of Intracranial Hemorrhages Using Computerized Tomography Scans

George Barker[1, 2], Zach Francis[1, 3], Julio Hidalgo Lopez[1, 4], Andre Zeromski[1]

[1]Washington and Lee University Department of Computer Science; [2]Washington and Lee University Department of Chemistry and Biochemistry; [3]Washington and Lee University Department of Philosophy; [4]Washington and Lee University Neuroscience Program

Introduction

Intracranial Hemorrhage (IH) is defined as any bleeding inside the cranial cavity. IH can be caused by head trauma, hypertension, aneurysm, excessive drug use, and several other factors. IH is always considered a medical emergency due to the fragile nature of brain tissue and the potential for its compression within the skull. Depending on the IH's location, it is categorized as one of five subtypes: intraventricular, intraparenchymal (both occur within the brain; i.e. intracerebral and are also termed hemorrhagic stroke), subarachnoid, epidural, and subdural; each of which requires different treatment interventions. IH It is the fifth leading cause of death, the leading cause of disability, and accounts for about 10% of all strokes in the United States as of 2017. Diagnosis of IH typically includes interpretation of a Computerized Tomography (CT) scan by medical professionals, and can be time and resource intensive (1, 6, 8).

Here, we propose a deep learning model based on a Convolutional Neural Network (CNN) to aid in the diagnosis of IH. A trained deep learning model that is able to recognize the presence, or absence, of IH and categorize IH subtypes could be highly beneficial as screening software or a diagnostic aid to radiologists and other medical professions. It could be especially useful in emergency room environments where a radiologist may not be immediately available to interpret a patient's CT scan, and an emergency physician would like to verification their diagnosis. Using the Python programming language and several modules, we construct an environment that can read and manipulate CT data before using it to train any deep learning model architecture. Using this plug-and-play approach, we train several different models and compare their performances to determine which could best be used in a clinical setting.

Background

This project was inspired by a competition hosted on the Kaggle machine learning community website by the Radiological Society of North America (RSNA). The RSNA is an international, non-profit, organization of radiologists and medical physicists. Its mission is to "promote excellence in patient care and healthcare delivery through education, research, and technological innovation". The objective of the competition was to design an algorithm to accurately detect IH and its subtypes. Competitors were ranked based on their weighted mean column-wise log loss; however, instead of entering the competition, we decided to do a comparison study between different models.

The RSNA provided the training and testing datasets for this project. The training set alone consists of 752,800 2D CT scans collected from over 25,000 exams, and their labels. CT Data was collected from research institutions at Stanford University, Thomas Jefferson University, Unity Health Toronto, Universidade Federal de São Paulo, and the American Society of Neuroradiology. Each CT scan was hand-labeled by medical professional volunteers who are members of the RSNA. The dataset includes all IH subtypes, and all instances of IH are acute (10).

Knowing that we would be working with CT image data, we chose to approach the problem with CNN-based models. CNNs are the preferred architecture for image processing today, and can achieve remarkable performance with a smaller amount of training data, though this was not an issue in our case. CNNs are a powerful machine vision tool as they extract features from images in two-dimensional space. This allows for more robust feature extraction, and for the maintenance of spatial relationships between these features when compared to extracting features from a flattened image.

CT scans are stored in DICOM files, which we had to manipulate in order to form an input that could be used by our models. These files store substantial metadata, which is useful for the project, as well as a pixel array. The pixel array is comprised of greyscale Hounsfield Unit (HU) values which range from [-1000, 1000]. Since normal pixel values are within [0, 255], we lineralized the DICOM pixel arrays and passed these to a windowing algorithm which focuses the scope of the image to display certain tissue types. These processes are further described in the accompanying README (12, 13).

We chose to evaluate three previously defined models in addition to a hand-coded vanilla CNN: an AlexNet architecture, an InceptionV3 pre-trained model, a ResNet pre-trained model, and a DenseNet pretrained model. AlexNet is an 8-layer CNN based architecture that became a gold standard for machine vision in 2012 (3, 4, 5). DenseNet, ResNet, and InceptionV3 are keras models that have been pre-trained on the ImageSet dataset.. Further details on model implementation are described in the accompanying README (9, 11, 15).

Related Work

There has been substantial recent work on the application of deep learning models, especially those based on CNNs, to medical settings as diagnostic or analysis tools. Past studies have used CNN-based models to calculate the level of muscle atrophy in the supraspinatus fossa following rotator cuff injury and aid in the diagnosis of lung cancer.

The CNN model used by Young et al. for their muscle atrophy calculation task was composed of five convolution layers each of which was followed by a pooling layer. The output of the final pooling layer was passed to a fully connected layer and then a deconvolution and output layer. They achieved an accuracy of 99.8% and all calculations agreed with clinician calculations (2).

The CNN model used by Ardila et al. detects cancerous lung nodules from a 3D CT-scan and predict a lung cancer diagnosis. Their end-to-end model used a lung segmentation model to first analyze the CT scan for the region of lungs. This reduces their search spaces and prevents the model from searching for likely misleading patterns outside of the lung search space. The model uses a CNN with inception layers to extract features from a global model and region of interest detection model. The region of interest detection model finds 2 of the most likely region of interests for a tumor. The global view model extracts features from a full view of the CT scan. The features extracted from the global model and region of interest model are concatenated to predict a lung cancer diagnosis (14).

In addition to the studies described above, we took inspiration from some of the implementations used by competitors in the kaggle RSNA brain hemorrhage detection competition. Primarily, this helped us gain an understanding for the nature of DICOM files and the manipulations we would need to perform on this data type in order to use a CNN model to analyze it (12, 13).

Methods

Training Set and Testing Set

The dataset provided by the RSNA society is made up of 752,800 DICOM images. The filenames of each image is identified by a patient ID, which corresponds to a label in a .csv file for all patient IDs. We chose to split up the provided dataset 80% into a training set (602,240 images) and 20% into a testing set (150,560 images). We assessed all models based on this split of the training and testing set. We did not make a validation set due to the large size of our training data and the low probability of overfitting on one epoch of training.

HU Transformation and Windowing

Windowing is important because it allows the model to focus on a specific range of values in the DICOM image to analyze. These windows we apply allow us to see different tissues and the architecture of the brain, so hemorrhages can be more easily detected with the human eye. As shown in Figure 1, the raw dicom images are entirely grey, with no detectable distinctive features. We applied a linear transformation to our CT scans to convert the pixel values into densities that accurately reflect the internal objects of the human body. We transformed our brain scans into 3 brain windows and concatenated these along 3 channels. There were a variety of windowing techniques available for use. We chose to use a 3 channel windowing technique so that we could still use a 2D CNN and provide the model with additional details about the brain image. Our windowing method separated each channel of our image into a brain, vascular, and subdural window.

We originally debated whether using a bone window or a vascular window in tandem with a brain and subdural window would yield better results. The reasoning for this was that the brain and subdural windows would be crucial for IH detection and categorization as these are the tissues that are most impacted by IH. In order to test which of these two windows would best complement the brain and subdural windows, we trained and tested our basic vanilla CNN model, due to its shorter training time, with either bone/brain/subdural windowed images or vascular/brain/subdural windowed images (Figure 1) and compared its performance on both image types (see Results section below). We later used the best performing window combination to train our larger models.

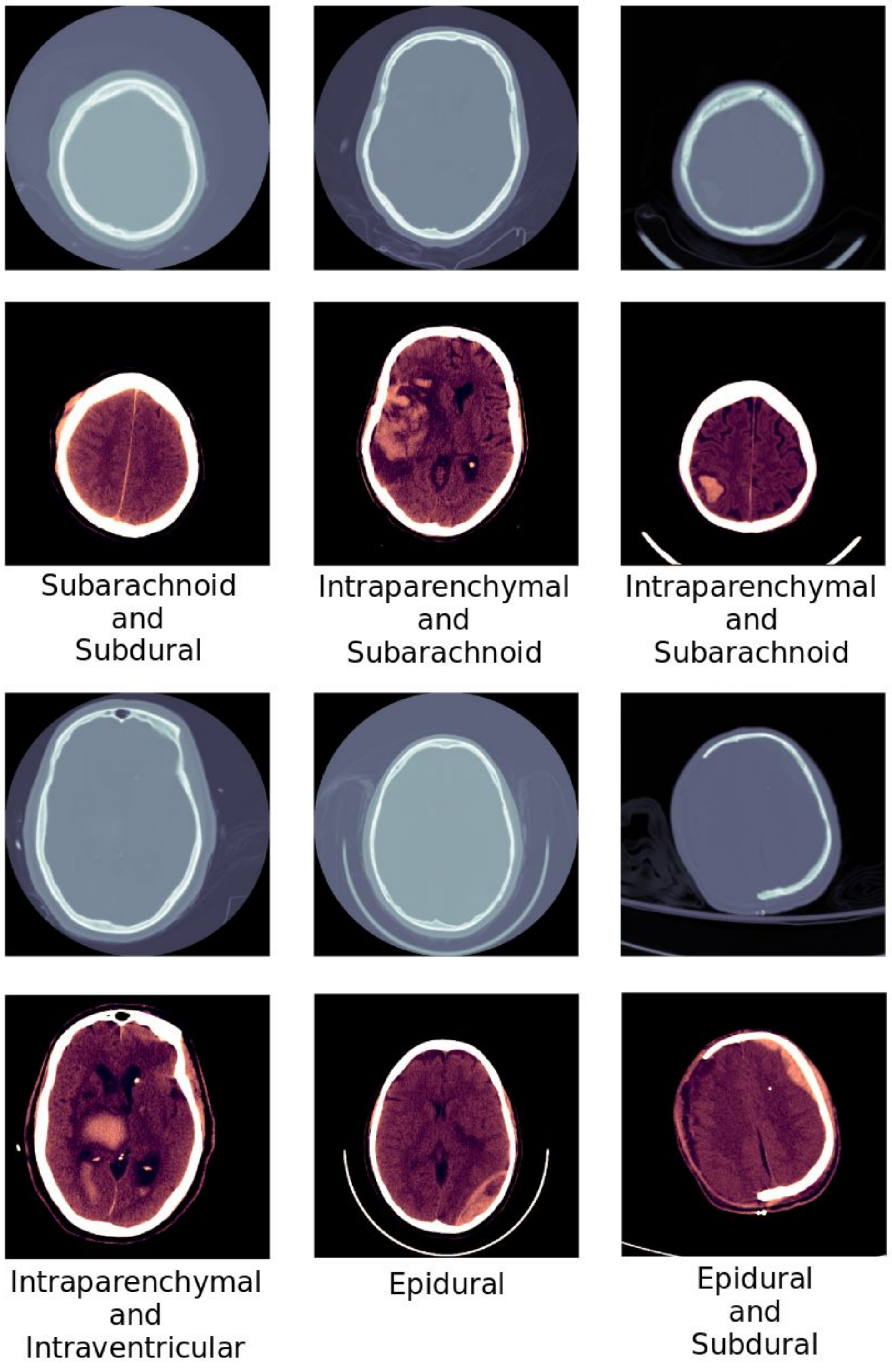

Subarachnoid and Subdural
Intraparenchymal and Subarachnoid
Intraparenchymal and Subarachnoid
Intraparenchymal and Intraventricular
Epidural
Epidural and Subdural

Figure 1 - Raw CT images (top) and the same images after compound windowing (bottom). The windows used were brain, subdural, and vascular. The IH subtype is stated below the windowed image.

Padding, Truncating, and Removing Corrupted Files

The DICOM files provided by the RSNA society are 512 x 512 x 1. We chose to keep these dimensions to retain the best quality of data. Some images were less than 512 x 512. We padded these images with 0s. 0s represent the border of our CT scan and thus do not represent any feature that should be extracted by the model. We also truncated the end of images larger than 512 x 512. This is a point of weakness as we are losing information in these images, it is possible, albeit unlikely that some images of the brain are cut off. However, << 1% of our data needed to be padded or truncated. Truncation occurred much more rarely than padding. There were also 1 in the dataset that is corrupted. We had to make code that could adequately handle this cases, and future case, to ignore the file during training.

Preprocessing and Feeding Data to the Model

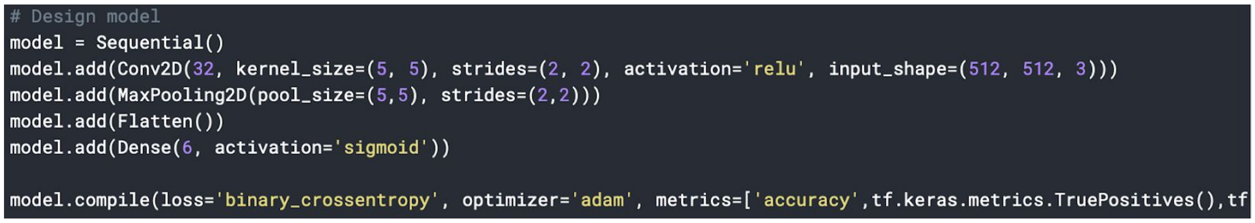
We created a DataGenerator class to feed images into our model for training so that we did not fill up the RAM too quickly by loading in the entire training or testing dataset at once. We also preprocess our data within the data generator. As the data generator retrieves each batch, it transforms the raw pixel values of the DICOM file into the corresponding HU units and creates the specified windows as channels. Further details are in the README.

Loss Function, Optimizer, and Metrics

For all of our tested models, we used the same optimizer, metrics, and loss function. The loss function we used was binary cross entropy. Binary cross entropy was useful for our 6 node output layer as each node is a label and is not a mutually-exclusive classification. The optimizer we used was Adam. Adam is useful as it has an adaptive learning rate that makes it efficient to use on larger datasets and able to avoid local minimums.The metrics we used were accuracy, true positives, false positives, true negatives, and false negatives. Our accuracy metric predicted out of the 6 output nodes, how many of the labels for each of these nodes were correct over the total amount of labels. True positives, false positives, true negatives, and false positives were also used as metrics to provide a more detailed picture of the model predictions. Area under the curve (AUC) was used to show how well, or poorly, the model is making it's predictions. Specifics on how these metrics work are in the README.

Model Implementation

The architecture for our basic vanilla CNN is relatively simple. It consists of a convolution layer followed by a max pooling layer followed by a fully connected output layer (Figure 2). The Alexnet was hand-coded and consists of five convolution layers, the first two and the last of which are followed by max pooling and dropout layers, followed by three fully connected layers and a fully connected output layers. The InceptionV3 and DenseNet were loaded from keras's pre-trained models. Further specifics about our implementations can be found in the accompanying README.



```
# Design model
model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), strides=(2, 2), activation='relu', input_shape=(512, 512, 3)))
model.add(MaxPooling2D(pool_size=(5,5), strides=(2,2)))
model.add(Flatten())
model.add(Dense(6, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy',tf.keras.metrics.TruePositives(),tf
```

Figure 2 - Python implementation of our basic vanilla CNN using the keras module.

Evaluation and Discussion

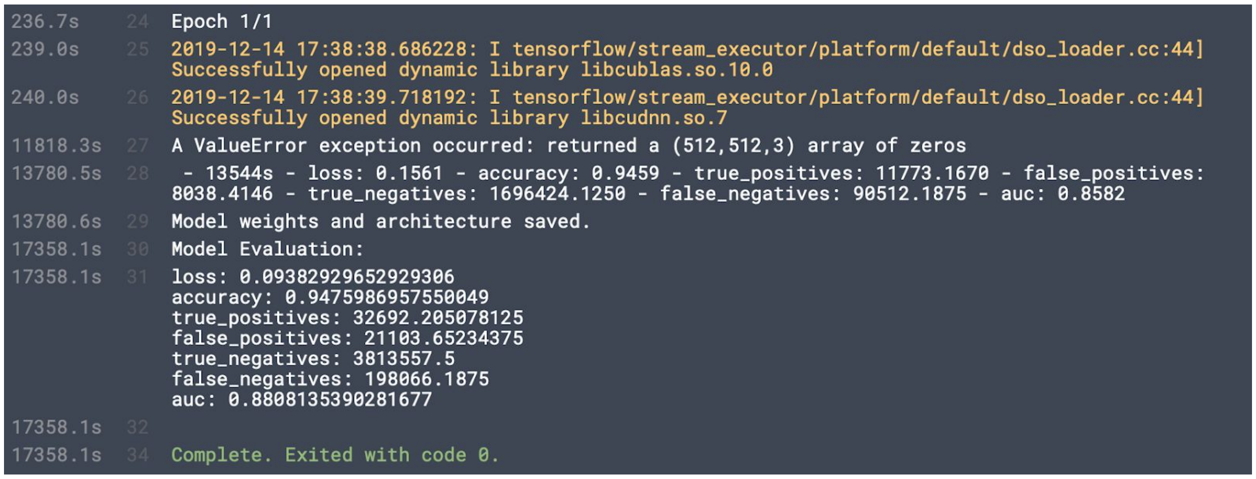
To begin we will discuss a comparison between the window combinations bone/brain/subdural and vascular/brain/subdural. We found that the vascular/brain/subdural windowed images led to similar performance when used as the input to a basic vanilla CNN. (Table 1; Figures 3, 4)..We ended up using the vascular window as we felt that the mode advanced models would be able to extract more abstract features from this window as it seemed to have more soft-tissue granularity.

We then trained our basic vanilla CNN on two epochs to see if we could still obtain an increase in performance via increased training without the model overfitting our training data. We saw a slight increase in the area under the curve metric compared to an identical model ran on one epoch of the same training data (data not shown). However, we felt that running the more complex models on multiple epochs would be inefficient for our purposes, and therefore only trained these models for one epoch. As an example the DenseNet was trained on 25,000 samples and 50,000 samples and showed very similar metrics (figure 8 a,b).

Surprisingly, the AlexNet model performed worse than the basic vanilla CNN with a smaller area under the curve, and a true positives four orders of magnitude below the basic vanilla CNN. The only performance metric where the AlexNet slightly outperformed the basic vanilla CNN was true negatives (Table 2; Figures 4, 5). This is most likely due to AlexNet's huge architecture requiring more epochs to train properly than a basic vanilla CNN which can obtain generally good results on a smaller dataset.

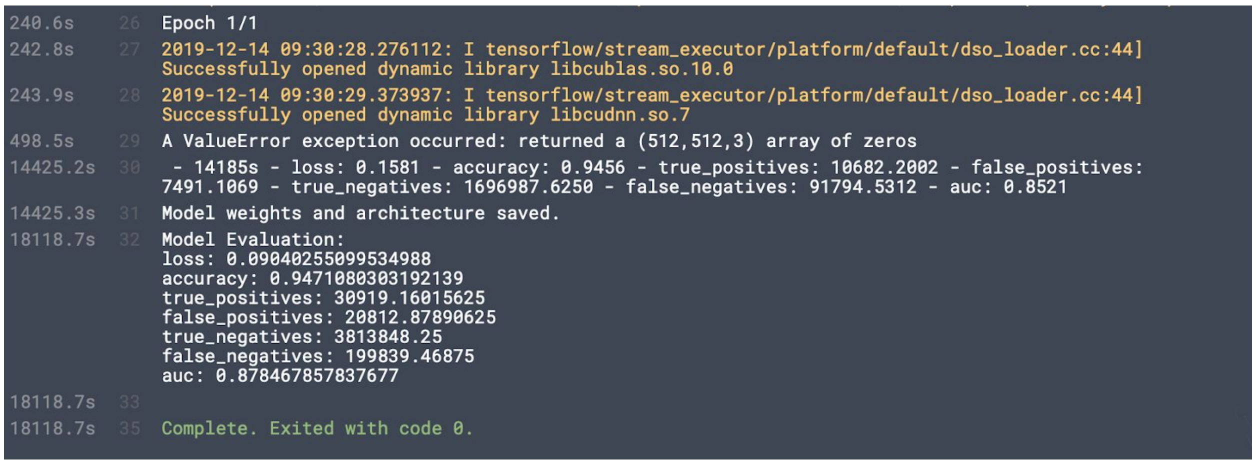
Additionally, our InceptionV3, ResNet, and DenseNet models also performed worse than the vanilla CNN. The area under the curve was lower, the loss was much higher, and our metrics of false positives/negatives were worse as well. The model had higher true negatives, but false negatives were higher as well. The predicted positives were very low in general, and we think the reason for this is because the model is very deep, and potentially extracting features where none exist. This could be a reason why the vanilla CNN performs better as well, since it is extracting only the abstract features it needs for classification.

We believe our models can be made more robust by training on more epochs, however, given the size of our training set this was very difficult. We could have broken our dataset of 700,000 down into a dataset of 100,000 or 50,000 to train on multiple epochs. We did not see a large difference between the vanilla CNN model trained on 20,000 images and the whole dataset of 750,000 images. Therefore, training on the whole dataset may have limited our efficiency in training our models. In the future, it would be beneficial to train on a smaller dataset. This would allow us to feasibly train complex models to optimum levels and efficiently hyperparameter tune. Further, it may be beneficial to define a custom loss function to increase the cost of false negatives exhibited by our complex models.



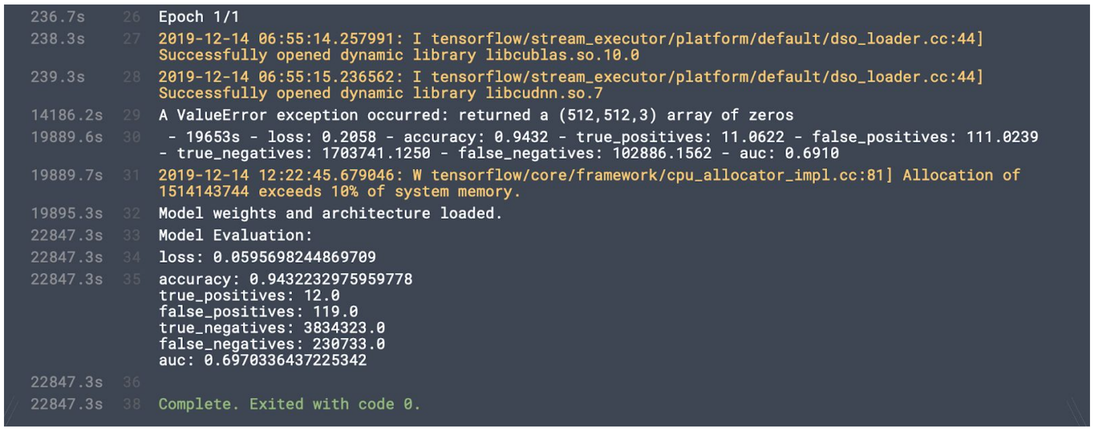
```
236.7s    Epoch 1/1
239.0s    2019-12-14 17:38:38.686228: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
          Successfully opened dynamic library libcublas.so.10.0
240.0s    2019-12-14 17:38:39.718192: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
          Successfully opened dynamic library libcudnn.so.7
11818.3s  A ValueError exception occurred: returned a (512,512,3) array of zeros
13780.5s   - 13544s - loss: 0.1561 - accuracy: 0.9459 - true_positives: 11773.1670 - false_positives:
          8038.4146 - true_negatives: 1696424.1250 - false_negatives: 90512.1875 - auc: 0.8582
13780.6s  Model weights and architecture saved.
17358.1s  Model Evaluation:
17358.1s  loss: 0.09382929652929306
          accuracy: 0.9475986957550049
          true_positives: 32692.205078125
          false_positives: 21103.65234375
          true_negatives: 3813557.5
          false_negatives: 198066.1875
          auc: 0.8808135390281677
17358.1s
17358.1s  Complete. Exited with code 0.
```

Figure 3 - Evaluation output of our **basic vanilla CNN** after one epoch of training and testing with **bone**/brain/subdural compound windowed images.



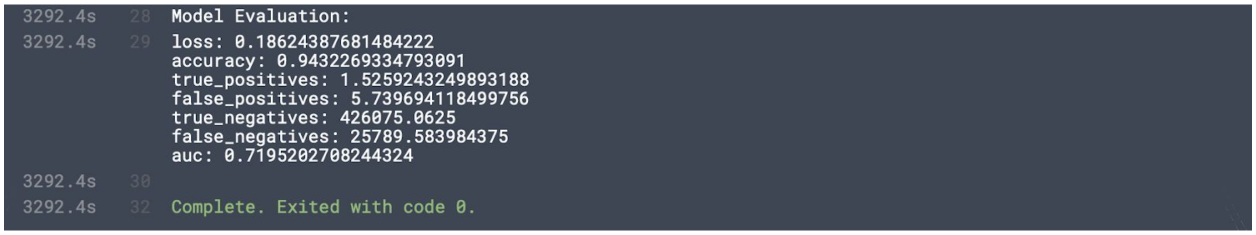
```
240.6s    Epoch 1/1
242.8s    2019-12-14 09:30:28.276112: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
          Successfully opened dynamic library libcublas.so.10.0
243.9s    2019-12-14 09:30:29.373937: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
          Successfully opened dynamic library libcudnn.so.7
498.5s    A ValueError exception occurred: returned a (512,512,3) array of zeros
14425.2s   - 14185s - loss: 0.1581 - accuracy: 0.9456 - true_positives: 10682.2002 - false_positives:
          7491.1069 - true_negatives: 1696987.6250 - false_negatives: 91794.5312 - auc: 0.8521
14425.3s  Model weights and architecture saved.
18118.7s  Model Evaluation:
          loss: 0.09040255099534988
          accuracy: 0.9471080303192139
          true_positives: 30919.16015625
          false_positives: 20812.87890625
          true_negatives: 3813848.25
          false_negatives: 199839.46875
          auc: 0.878467857837677
18118.7s
18118.7s  Complete. Exited with code 0.
```

Figure 4 - Evaluation output of our **basic vanilla CNN** after one epoch of training and testing with **vascular**/brain/subdural compound windowed images.



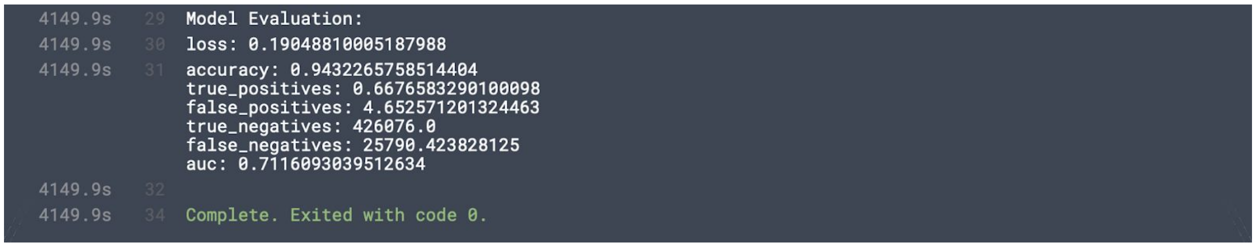
```
236.7s     Epoch 1/1
238.3s     2019-12-14 06:55:14.257991: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
           Successfully opened dynamic library libcublas.so.10.0
239.3s     2019-12-14 06:55:15.236562: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
           Successfully opened dynamic library libcudnn.so.7
14186.2s   A ValueError exception occurred: returned a (512,512,3) array of zeros
19889.6s    - 19653s - loss: 0.2058 - accuracy: 0.9432 - true_positives: 11.0622 - false_positives: 111.0239
           - true_negatives: 1703741.1250 - false_negatives: 102886.1562 - auc: 0.6910
19889.7s   2019-12-14 12:22:45.679046: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of
           1514143744 exceeds 10% of system memory.
19895.3s   Model weights and architecture loaded.
22847.3s   Model Evaluation:
22847.3s   loss: 0.0595698244869709
22847.3s   accuracy: 0.9432232975959778
           true_positives: 12.0
           false_positives: 119.0
           true_negatives: 3834323.0
           false_negatives: 230733.0
           auc: 0.6970336437225342
22847.3s
22847.3s   Complete. Exited with code 0.
```

Figure 5 - Evaluation of our **AlexNet model** after one epoch of training. All training and testing images were composed from their vascular, brain, and subdural windows.



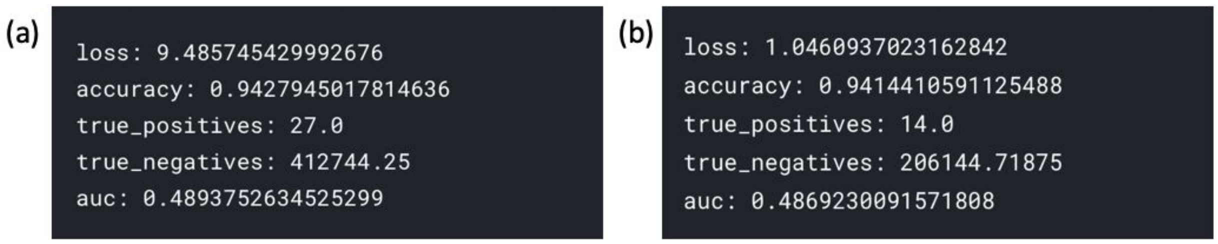
```
3292.4s    Model Evaluation:
3292.4s    loss: 0.18624387681484222
           accuracy: 0.9432269334793091
           true_positives: 1.5259243249893188
           false_positives: 5.739694118499756
           true_negatives: 426075.0625
           false_negatives: 25789.583984375
           auc: 0.7195202708244324
3292.4s
3292.4s    Complete. Exited with code 0.
```

Figure 6 - Evaluation of our **Inception model** with pre-trained CNN weights and all layers frozen except the output layer, after one epoch of training. All training and testing images were composed from their vascular, brain, and subdural windows.



```
4149.9s    Model Evaluation:
4149.9s    loss: 0.19048810005187988
4149.9s    accuracy: 0.9432265758514404
           true_positives: 0.6676583290100098
           false_positives: 4.652571201324463
           true_negatives: 426076.0
           false_negatives: 25790.423828125
           auc: 0.7116093039512634
4149.9s
4149.9s    Complete. Exited with code 0.
```

Figure 7 - Evaluation of our **Inception model** with pre-trained CNN weights and all layers frozen except the output layer and sigmoid dense layer, after one epoch of training. All training and testing images were composed from their vascular, brain, and subdural windows.



(a)

```
loss: 9.485745429992676
accuracy: 0.9427945017814636
true_positives: 27.0
true_negatives: 412744.25
auc: 0.4893752634525299
```

(b)

```
loss: 1.0460937023162842
accuracy: 0.9414410591125488
true_positives: 14.0
true_negatives: 206144.71875
auc: 0.4869230091571808
```

Figure 8 - Evaluation of our **Densenet model** after one epoch of training through a subset of (a) 50,000 or (b) 25,000 samples. All training and testing images were composed from their vascular, brain, and subdural windows.

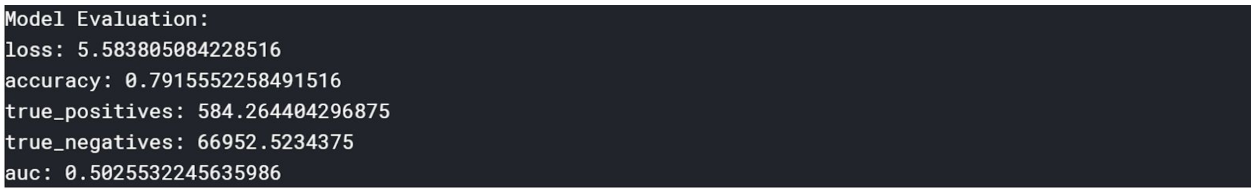

```
Model Evaluation:
loss: 5.583805084228516
accuracy: 0.7915552258491516
true_positives: 584.264404296875
true_negatives: 66952.5234375
auc: 0.5025532245635986
```

Figure 9 - Evaluation of our **ResNet** model after one epoch of training through a subset of 50,000 samples. All training and testing images were composed from their vascular, brain, and subdural windows.

Performance of Vanilla CNN on Differently Windowed Images

Windows	***Loss***	***Accuracy***	***True Positives***	***True Negatives***	***Area Under the Curve (u^2)***
Bone/ Brain/ Subdural	0.0938	94.76%	32,692	3,813,558	0.8808
Vascular/ Brain/ Subdural	0.0904	94.71%	30,919	3,813,848	0.8785

Table 1 - Comparison of the performance of a basic vanilla CNN on CT images composed of different window combinations. The vascular/brain/subdural window combination had better performance on every metric except for true negatives.

Performance Comparison of Different CNN-Based Deep Learning Models

Model	***Loss***	***Accuracy***	***True Positives*** (Sensitivity)	***True Negatives*** (Specificity)	***Area Under the Curve (u^2)***
Basic Vanilla CNN	0.0904	94.71%	30,919	3,813,848	0.8785
AlexNet	0.0596	94.32%	12	3,824,323	0.6970
InceptionV3	0.1905	94.32%	0.6677	426,076	0.7195
DenseNet*	9.4857	94.28%	27	412,744	0.4893
ResNet*	5.5838	79.15%	584	66,952	0.5026

Table 2 - Comparison of the performances of our basic vanilla CNN, AlexNet, InceptionV3, DenseNet, and ResNet implementations. All models were trained for one epoch on the full dataset on vascular/brain/subdural window. *These models were trained on a subset of 50,000 samples.

Sources

1. *Emerg Med Clin North Am*. 2012 August ; 30(3): 771–794. doi:10.1016/j.emc.2012.06.003

2. Kim, Joo Young, et al. “Development of an Automatic Muscle Atrophy Measuring Algorithm to Calculate the Ratio of Supraspinatus in Supraspinatus Fossa Using Deep Learning.” *Computer Methods and Programs in Biomedicine*, vol. 182, 2019.

3. Krizhevsky, Alex, et al. “ImageNet Classification with Deep Convolutional Neural Networks.”

4. https://www.mydatahack.com/building-alexnet-with-keras/

5. https://engmrk.com/alexnet-implementation-using-keras/

6. https://my.clevelandclinic.org/health/diseases/14480-intracranial-hemorrhage-cerebral-hemorrhage-and-hemorrhagic-stroke

7. https://en.wikipedia.org/wiki/Intracranial_hemorrhage

8. https://jovianlin.io/saving-loading-keras-models

9. https://towardsdatascience.com/densenet-2810936aeebb

10. https://www.kaggle.com/c/rsna-intracranial-hemorrhage-detection

11. https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

12. https://www.kaggle.com/gzuidhof/full-preprocessing-tutorial

13. https://www.kaggle.com/allunia/rsna-ih-detection-eda

14. https://www.nature.com/articles/s41591-019-0447-x

15. https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

16. https://cloud.google.com/tpu/docs/inception-v3-advanced

17. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy