Machine Learning Engineer Nanodegree

Capstone Project

Alejandro Trujillo

May 27th, 2018

I. Definition

Project Overview

This project focuses on leveraging the power of convolutional neural networks (CNNs) and transfer learning models to predict the diagnosis of Optical Coherence Tomography (OCT) scans.

Optical coherence tomography is a non-invasive imaging test. OCT uses light waves to take cross-section pictures of your retina.

These diagnosis provide treatment guidance for multiple diseases, including:

- Choroidal neovascularization (CNV)
- Drusen
- Diabetic Macular Edema (DME)

The goal of this project is to develop a machine learning model that can enhance and aid physicians in their diagnosis of CNV, Drusen, and DME diseases.

The initial idea for this project was taken from kaggle's article "Retinal OCT Images (optical coherence tomography)"

The source of the full dataset for this project can be found at Mendeley's article "Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification"

The full dataset of 84,495 images was reduced to only 7,020 images given the limitations of the computing resources available at the time. There reduced dataset can be found here: OCT2017-RESIZED-V1.zip

Problem Statement

Problem

Approximately 30 million OCT scans are performed each year, and the analysis and interpretation of these images takes up a significant amount of time (Swanson and Fujimoto, 2017).

Well trained ophthalmologists can do an outstanding job at classifying any disease depicted in a OCT scan. However, long hours at work and eye fatigue can hinder their diagnosis. Even one misdiagnosed case can hinder the relationship with their patients and their reputation. Here I present an approach to aid ophthalmologists streamline their diagnosis and further enhance their relationship with their patients.

Intended Solution

The goal is to provide a machine learning model utilizing convolutional neural networks to predict the diagnosis of OCT scans with over 70% accuracy.

Strategy

The tasks involved in accomplishing this goal are the following:

1. Download and preprocess the OCT scans dataset
2. Create a CNN benchmark model with at least 50% accuracy to improve upon
3. Create a transfer learning model with at least 70% accuracy utilizing one of the following pretrained models:
    a. VGG-19
    b. ResNet-50
    c. InceptionV3
    d. Xception
4. Evaluate the models using their accuracy.

Metrics

The dataset is divided in training, validation and testing to prevent overfitting.

Each dataset has the following total number of images:

- Training has 5,082 images
- Validation has 938 images
- Test has 1,000 images

Additionally the following metrics are used to quantify the performance of the benchmark and optimized algorithms.

- **Training time:** Measures the time that the algorithm takes to learn the patterns within the training set.
    - Important to keep training time as small as possible without impacting accuracy to provide a quicker feedback loop and try other models or approaches.

- **Accuracy:** Indicates the degree to which the resulted output matches the correct label.
    - Important to keep as high as possible to provide to the user confidence that the model is producing a correct diagnosis.
- **Optimizer:** RMSprop divides the learning rate by an exponentially decaying average of squared gradients.

II. Analysis

Data Exploration

The original dataset is comprised of 84,495 total X-Ray images.

This large dataset had to be reduced to only 7,020 total images due to the limited resources available at the time. (8GB of RAM, Intel Core i7-4700HQ @ 2.40GHz and HDD Toshiba 1TB 5400 RPM)

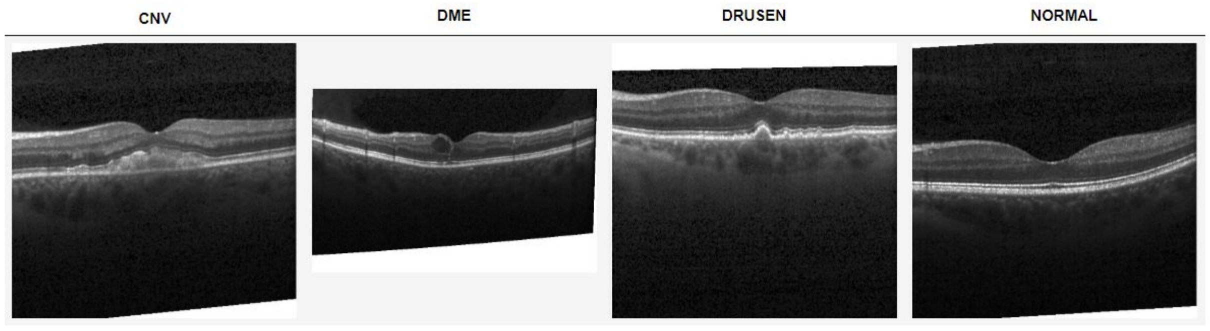
The reduced dataset of 7,020 total images was divided in the following sections

	CNV	DME	DRUSEN	NORMAL	TOTAL
Training	1,636	939	1,229	1,278	**5,082**
Validation	252	217	210	259	**938**
Testing	250	250	250	250	**1,000**

Exploratory Visualization

As seen in the images below the different categories could display similar traits.

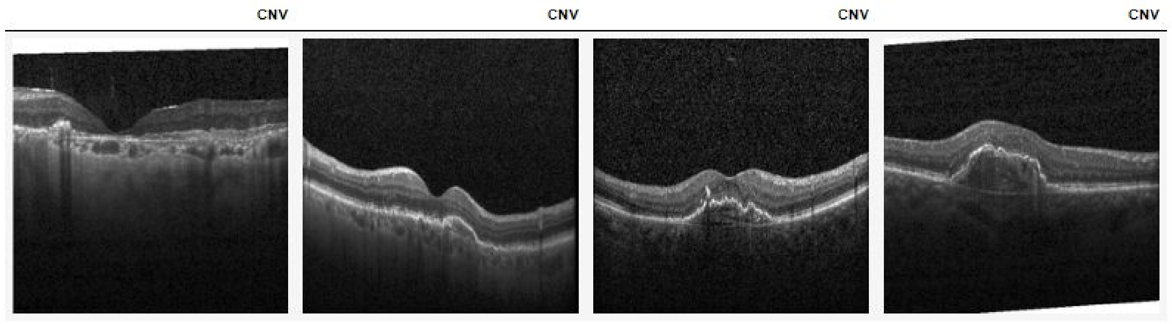
It is hard for someone without medical diagnosis background to identify outliers in the data, due to the specialized field of ophthalmology and radiology.



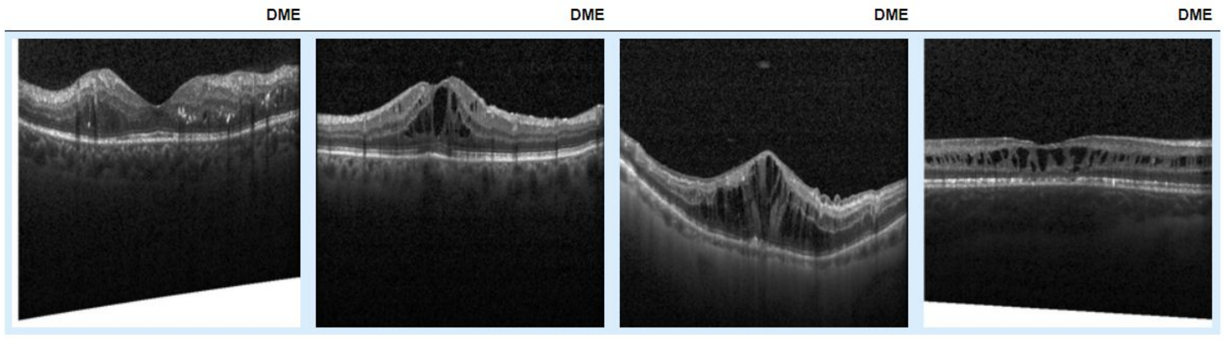
Nonetheless, when looking at many images of the different categories, one can see some distinct traits between the different categories.

Here is an example of the previous statement.

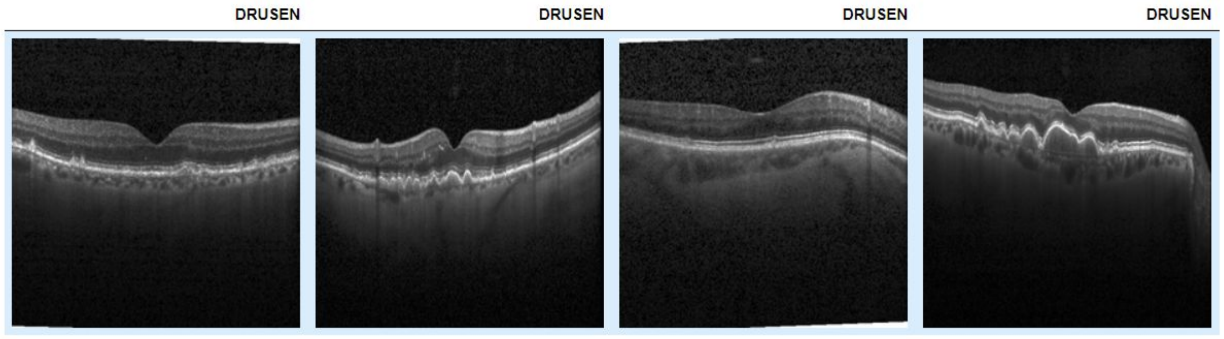
Distinctive CNV scans



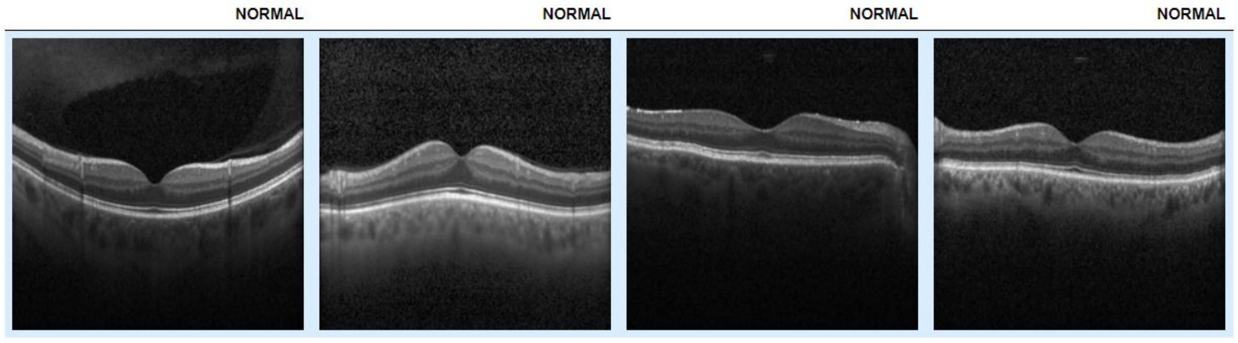
Distinctive DME scans



Distinctive DRUSEN scans



Distinctive NORMAL scans



A convolutional neural network should be able to analyse these images, identify the different patterns and produce an architecture robust enough to produce at least 70% accuracy.

Algorithms and Techniques

Processing the data.

After downloading the data. The initial step for data processing is to resize each image to 224x244 pixels on disk. This is with the purpose of optimizing computing resources and to accelerate training time. In a future step, the model will utilize this data dimension as its input shape to produce the CNN architecture.

Convolutional Neural Network (CNN)

I chose to use a CNNs architecture for my image classification project because of their outstanding image classification performance and frameworks available that facilitate the use of it, such as Keras and TensorFlow.

As stated by multiple sources and research papers on the internet:

"Convolutional neural networks (CNN) have recently shown outstanding image classification performance in the large scale visual recognition challenge (ILSVRC2012). The success of CNNs is attributed to their ability to learn rich mid-level image representations as opposed to hand-designed low-level features used in other image classification methods." [1]

Additionally, an interesting fact to ponder is that a few years ago the following statement was made by Microsoft researchers.

“Based on a survey of machine vision literature and vision experts at Microsoft Research, we believe classification accuracy of better than 60% will be difficult without a significant advance in the state of the art.”[2]

Even though that statement was made for CAPTCHA mechanisms a few years ago, it is interesting to see the improvements that have happened in the machine learning field.

Nowadays, the results that can be accomplished in the image classification reign with a limited amount of resources are impressive.

The CNNs architecture presented in this project is able to obtain at least 70% accuracy running on a 3 year old laptop with 8GB of RAM, Intel Core i7-4700HQ @ 2.40GHz and HDD Toshiba 1TB 5400 RPM .

Benchmark models

I took 3 different approaches for building a benchmark model

I will then evaluate the accuracy score of all 3 and utilize the benchmark model with the highest accuracy score to compare against the optimized transfer learning model.

These 3 benchmark models can be found in the jupyter notebook Benchmark.ipynb

Random Prediction Algorithm [3]

The concept of this algorithm goes as follow: assign a random prediction for each input in the input set and then evaluate its overall accuracy.

This algorithm is in the “Random Prediction Algorithm” section in the Benchmark.ipynb jupyter notebook

This algorithm achieved an accuracy score of **24.9115%**.

It is somewhat impressive that a random algorithm achieved 24.9115% accuracy in a dataset of 5,082 images. I think this high accuracy score in the random algorithm is due to the small number of possible outcomes, meaning each image can only have 1 or 4 categories. However, I assume a random person would score much lower than this without any radiology training.

Zero Rule Algorithm [3]

I was introduced to this concept by the article How To Implement Baseline Machine Learning Algorithms From Scratch With Python

This article states *"For classification problems, the one rule is to predict the class value that is most common in the training dataset. This means that if a training dataset has 90 instances of class "0" and 10 instances of class "1" that it will predict "0" and achieve a baseline accuracy of 90/100 or 90%."*

This algorithm is in the "Zero Rule Algorithm" section in the Benchmark.ipynb jupyter notebook

This algorithm achieved an accuracy score of **32.1921%**

In my opinion this accuracy score is pretty high for such a basic algorithm. As I mentioned before I assume a random person would score much lower than this without any radiology training.

We will now move on to explore the power of a basic CNN architecture from scratch to build a basic CNN benchmark model.

Basic CNN architecture from scratch

My last benchmark model is a basic CNN architecture built from scratch.

This model's initial goal was to only achieve more than 1% accuracy, but given the predictive power of CNNs this benchmark model was achieving an accuracy score above 50% consistently.

The benchmark architecture will be as follows.

1. Convolutional 2D layer with kernel size 2 and ReLU activation function.
2. Max pooling 2D layer with pool size 2.
3. Convolutional 2D layer with kernel size 2 and ReLU activation function.
4. Max pooling 2D layer with pool size 2.
5. Convolutional 2D layer with kernel size 2 and ReLU activation function.
6. Max pooling 2D layer with pool size 2.
7. Dropout layer with 0.3 probability.
8. Flatten
9. Dense with 4 nodes and softmax activation function.

This benchmark model is used to compare the results of the optimized transfer learning model.

This algorithm is in the “Train a CNN from Scratch” section in the Benchmark.ipynb jupyter notebook.

After its implementation, the benchmark model produced a test accuracy of **58.80%**

I believe it’s pretty impressive that CNNs are able to achieve over 50% accuracy in a dataset with only 5,082 images. This is benchmark model is only using about 6% of the original dataset of 84,495 images!

Parameters

The benchmark model is configured with the following parameters:

- Optimizer = rmsprop
- Loss = categorical_crossentropy
- Metrics = accuracy
- Epochs = 7

Time to train

The benchmark model took around 20 minutes to train on 5,082 images.

III. Methodology

Data Preprocessing

Minimal data preprocessing was required.

The only steps necessary to build the benchmark model and the transfer learning model are described below:

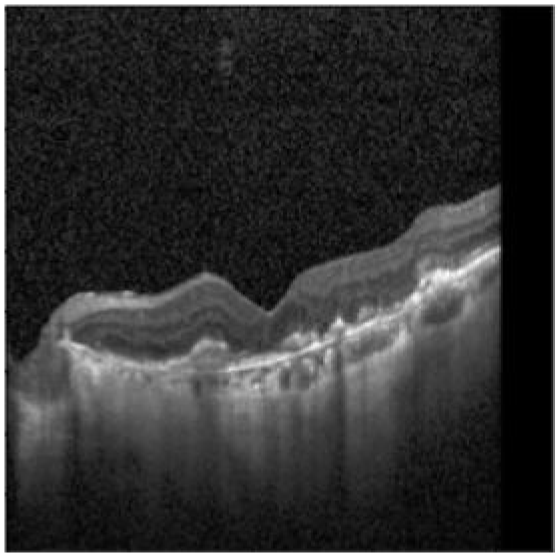
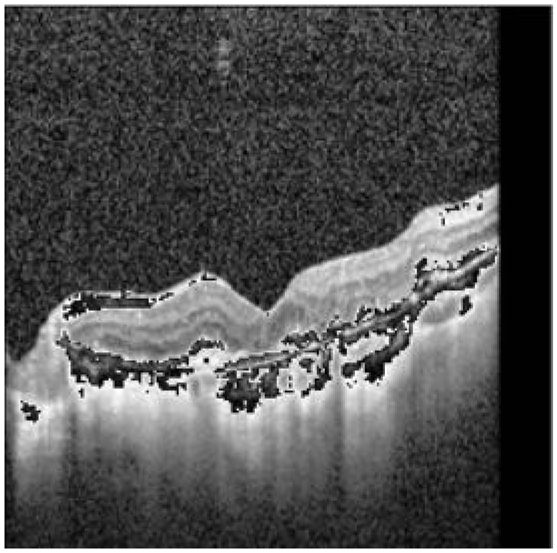
1. Resize all images to 224x244 on disk
2. Load the image from disk with a target size of 224x224 pixels
3. Finally convert the image into a 4D tensor with shape (1, 224, 224, 3).

Subtract the mean of the original dataset

Normalizing images by subtracting the ImageNet mean was explored, but it negatively impacted the model as it was not able to achieve above 25% accuracy.

Here are comparisons of the same image before and after preprocessing it via the Keras' *inception_v3.preprocess_input* method.

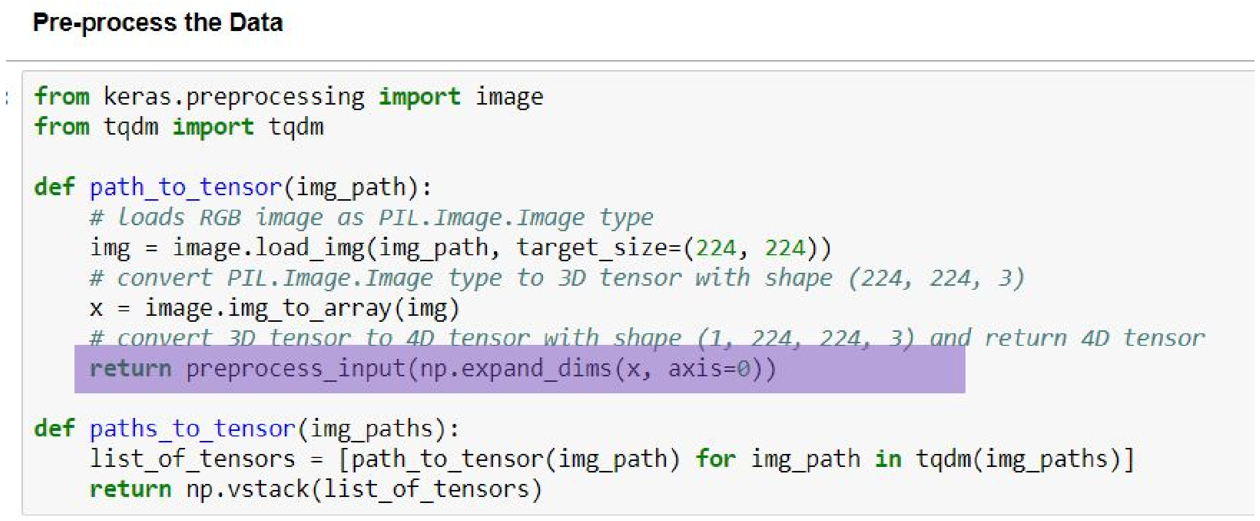
In a nutshell, I will use the InceptionV3 pretrained model on ImageNet provided by Keras. Please see the "Implementation" section for more information regarding this model.

Original Image	Preprocessed Image
	

The transfer learning model was not able to achieve an accuracy above 25% after executing image normalization on the entire dataset.

It appears that the original image contains more relevant information than the preprocessed image to properly predict its correct label.

Here the code snippet I utilized for applying image normalization (Please see the “Explore feasibility of mean subtraction algorithm” in the InceptionV3.ipynb notebook for more information)



Pre-process the Data

```
from keras.preprocessing import image
from tqdm import tqdm

def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(224, 224))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
    return preprocess_input(np.expand_dims(x, axis=0))

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
    return np.vstack(list_of_tensors)
```

Implementation

Transfer Learning

I chose to utilize the pretrained Inception V3 model, with weights pre-trained on ImageNet for the transfer learning model.

In a deep learning architecture higher layers identify the most general features of the dataset, such as edges.

Deep layers identify more distinctive features, as the deep learning architecture grows

The premise for utilizing transfer learning is to reduce training time and improve accuracy. This should be possible by utilizing the pre-trained layers and focus on the deeper layers that identify most specific traits.

An image of the full InceptionV3 architecture can be found in the following link: [InceptionV3 Architecture]

Parameters

The transfer learning model is configured with the following parameters:

- Optimizer = rmsprop

- Optimizer = Stochastic Gradient Descent:
    - Learning rate = 0.0001
    - Momentum = 0.9
- Loss = categorical_crossentropy
- Metrics = accuracy
- Epochs = 7

Time to train

The time to train was roughly 20 minutes, slightly better than the benchmark model.

Accuracy

The transfer learning model produced a test accuracy of **81.90%**

Complications

- Creating an Amazon EC2 g3.8xlarge instance was the initial option to explore for creating the model. This would have enabled the model to be trained in much more data, and possibly output an accuracy of over 95%. However, a couple days were spent requesting Amazon to allow me to have such instance. Amazon’s first response was to decline my request. Only after appealing their decision and after 4 days in total my request was granted but by then I had already created the model and achieved the expected results locally.

- It is unfeasible to utilize all the images of the original dataset given the computing resources available at the time.
    - The implementation phase utilized 100% of ram and disk causing the computer to become unresponsive many times and requiring restart.
    - The dataset had to be reduced to fit in the computer resources and to train the model.
    - Also, each image in the reduced dataset had to be resized on disk to the expected dimensions to optimize the disk read time.

- The first 200 layers of the InceptionV3 architecture had to be freezed, meaning utilize the existing weights, to achieve the above 70% accuracy. The top 110 layers of the architecture where trained on the OCT dataset. It is unclear why training the top 110 layers of this architecture was still faster than the benchmark model.

- Several days were spent preparing the computing resources, gathering data, researching articles about transfer learning, building prototypes. All this effort has enabled this project to be successful in building a model that can make predictions with over 70% accuracy.

Coding process

Getting familiar with Keras pretrained models can take some effort, research and trial and error. I decided to focus on their pretrained InceptionV3 model on the ImageNet dataset. I spent several hours in the "Utilize InceptionV3 pretrained model to predict OCT diagnosis" section of my InceptionV3 jupyter notebook trying different approaches until I was able to compute the expected accuracy.

Some of the approaches I researched were:

- Image Augmentation.
- Freezing and Unfreezing different InceptionV3 pretrained layers
- Increasing or reducing the dataset
- Increasing the number of epochs from 10 to 100.

A rough draft of my prototyping can be found in here: Prototyping

Refinement

The following parameters were refined and explored during implementation until the expected accuracy result was achieved.

- **Image Augmentation**: The following parameters were tried during prototyping but not implemented in the final solution because they didn't produce notable accuracy gains and the expected accuracy was achieved without requiring further image augmentation:
    - rotation_range=40,
    - width_shift_range=0.2,
    - height_shift_range=0.2,
    - rescale=1./255,
    - shear_range=0.2,

    - zoom_range=0.2,
    - horizontal_flip=True,
    - fill_mode='nearest'
    - Full list of parameters and definitions can be found here: https://keras.io/preprocessing/image/

- **Freezing and Unfreezing different InceptionV3 pretrained layers.**
    - Originally only 249 layers were frozen on the InceptionV3 model. This number had to be reduced to only 200 to allow the deeper layers of the model to be trained on the new dataset.

- **Increasing or reducing the dataset**
    - The total number of 84,495 images was reduced to only 5,082 images given the limited resources available.
    - Initially the number of images was reduced to only 1,000 images. The model was not able to identify specific features with such a reduced dataset. The model was only able to produce a top accuracy score of 53.1306% with this dataset.

- **Increasing the number of epochs from 10 to 100.**
    - Increasing the number of epochs didn’t had much impact in the final accuracy score. Even with just 4 or 5 epochs the model was able to compute the necessary traits to produce an acceptable accuracy.

IV. Results

Model Evaluation and Validation

Final Model

The final model utilizes the pretrained InceptionV3 architecture on the ImageNet dataset.

Only the First 200 layers of the model are frozen with their initial weights. The deeper 110 layers are trained with the OCT dataset.

The full architecture of the InceptionV3 model can be found here: InceptionV3 Architecture

Architecture

The top layer of the pretrained architecture was replaced by the following layers, to allow the model to make predictions on the new dataset:

1. GlobalAveragePooling2D
2. Dense layer with 1024 nodes and ReLU activation function
3. Dense layer with 4 final nodes and Softmax activation function to make classifications.

Model Compilation Process

The following class weights where computed by the scikit learn method compute_class_weight. These weights are utilized for fitting the model. These weights can be useful to tell the model to "pay more attention" to samples from an under-represented class[5]

CNV	DME	DRUSEN	NORMAL
0.77658924	1.35303514	1.03376729	0.99413146

Initially the InceptionV3 architecture was compiled using the following parameters:

- optimizer='rmsprop',
- loss='categorical_crossentropy'

- Class_weights from scikit compute_class_weight

The model went through a second compilation process producing better accuracy and less validation loss.

The second compilation process made use of the following parameters:

- Optimizer = Stochastic Gradient Descent with learning rate 0.0001 and momentum=0.9
- loss='categorical_crossentropy'
- Class_weights from scikit compute_class_weight

It is unclear why this second compilation process produced much better accuracy and less validation loss. However, changing the optimizer might have helped in producing a better accuracy score during the second compilation process.

After researching a few articles online and Keras documentation it is suggested that the RMSProp optimizer is recommended for recurrent neural networks[4].

Stochastic Gradient Descent seems to take advantage of its learning rate and momentum between each batch to optimize the model’s weights based on the information of the loss function in this case 'categorical_crossentropy'.

I suggest reading An overview of gradient descent optimization algorithms by Sebastian Ruder for additional information about optimization algorithms.

I found at least one more person online experiencing similar results. Please see the following question RMSProp and Adam vs SGD. I placed an answer hoping it will shed some light on the issue.

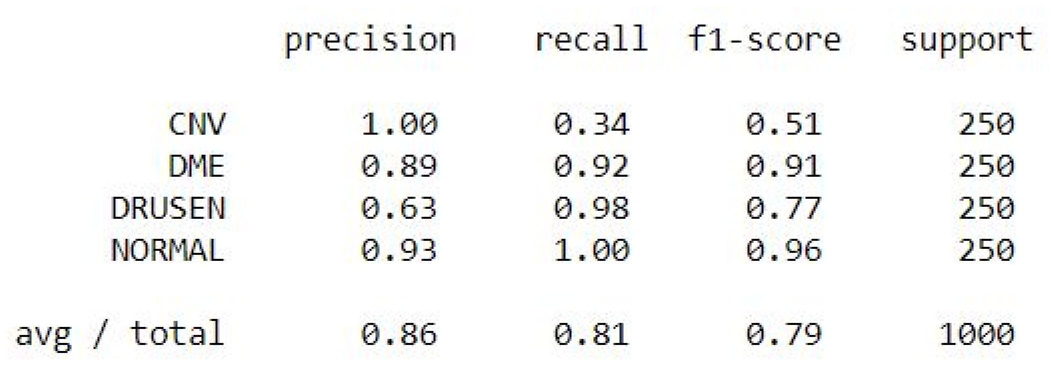
Final Results

The model only took about 20 minutes to train and it successfully produced a accuracy of over 70% consistently.

The highest accuracy score I have been able to produce is 81%

I believe this result is pretty impressive since this model is only using 6% of the original dataset.

Additionally, the precision, recall, f1-score metrics are acceptable:



```
             precision    recall  f1-score   support

        CNV       1.00      0.34      0.51       250
        DME       0.89      0.92      0.91       250
     DRUSEN       0.63      0.98      0.77       250
     NORMAL       0.93      1.00      0.96       250

avg / total       0.86      0.81      0.79      1000
```

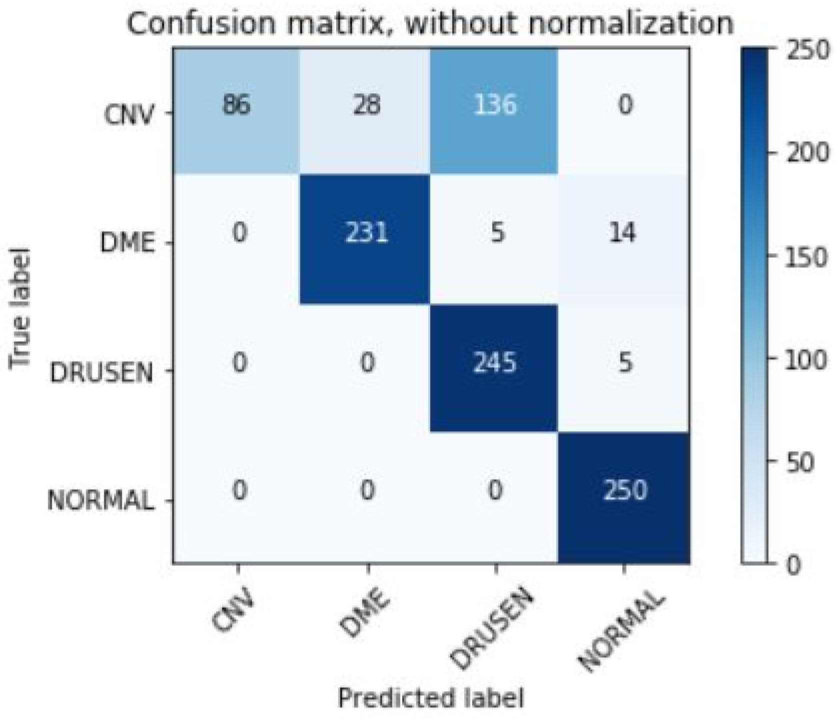
There are a few imbalances in the precision of DRUSEN and CNV recall, even after using normalized class weights.

These metrics are in the “Sklearn classification_report” section in the InceptionV3.ipynb notebook

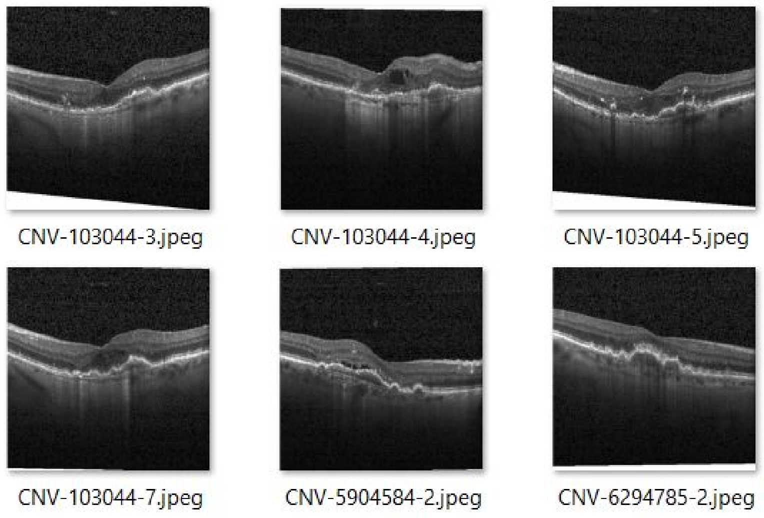
In layman's terms, high precision means that the model leans more on the premise that is ok if not all items have of such class but it should find all items with the given class. High recall means that the model leans more on the premise that it doesn't necessarily finds all items in the given class but they better be in such class.

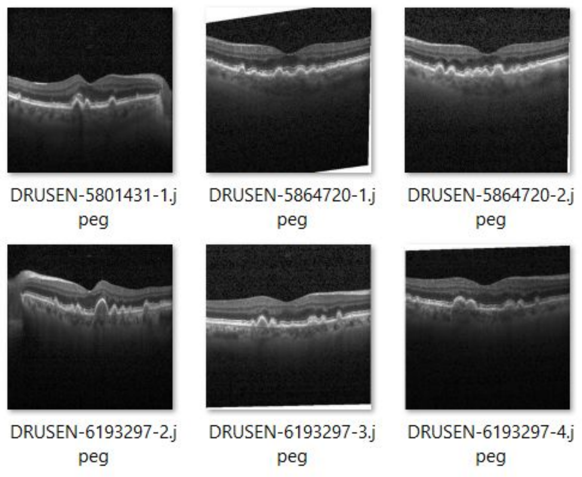
Confusion Matrix

This code can be found in the “Confusion matrix” section of the InceptionV3.ipynb notebook



Most of the misclassifications occurred by classifying DRUSEN as CNV. I believe this is reasonable given the similarities between these classes.

CNV	 CNV-103044-3.jpeg CNV-103044-4.jpeg CNV-103044-5.jpeg CNV-103044-7.jpeg CNV-5904584-2.jpeg CNV-6294785-2.jpeg

DRUSEN	 DRUSEN-5801431-1.jpeg DRUSEN-5864720-1.jpeg DRUSEN-5864720-2.jpeg DRUSEN-6193297-2.jpeg DRUSEN-6193297-3.jpeg DRUSEN-6193297-4.jpeg

Expectations

The final model has been tested with various inputs and it has shown to generalize well to unseen data.

The model is robust enough to produce over 70% accuracy consistently as long as is trained with at least 5,082 training images.

The results from the model can be trusted for this exercise. However, consulting with an experienced opthamologist is necessary to take this model to a production environment.

Justification

Benchmark vs Transfer Learning Model

The following table shows the difference between the benchmark and the transfer learning model.

During trials. The benchmark model accuracy oscillates between 55% and 70% accuracy.

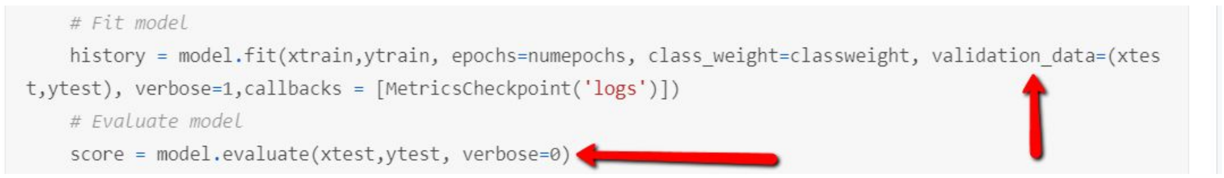
	Total Training Images	Epochs	Time to Train	Accuracy
Random Guess	5,082	N/A	Trivial	**24.9115%**

Zero Rule	5,082	N/A	Trivial	**32.1921%**
Benchmark	5,082	7	12 min 33sec	**58.80%**
Transfer Learning Model	5,082	5	22 min 46 sec	**81.90 %**

Additional Kaggle Kernels

There are other models online claiming to achieve around 91% accuracy, but after researching their models I have reason to believe that they are using their testing set as their validation set. This approach breaks a critical rule in Machine Learning: “Never use your testing set for training”

Here is the basis for the previous argument (Source link):



```
    # Fit model
    history = model.fit(xtrain,ytrain, epochs=numepochs, class_weight=classweight, validation_data=(xtes
t,ytest), verbose=1,callbacks = [MetricsCheckpoint('logs')])
    # Evaluate model
    score = model.evaluate(xtest,ytest, verbose=0)
```

Final Solution

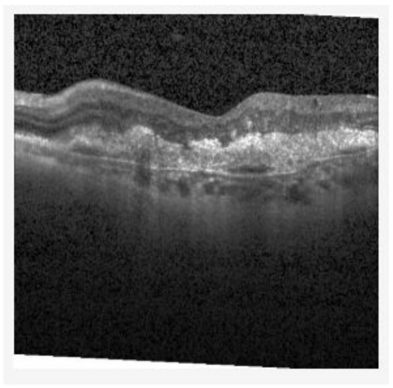
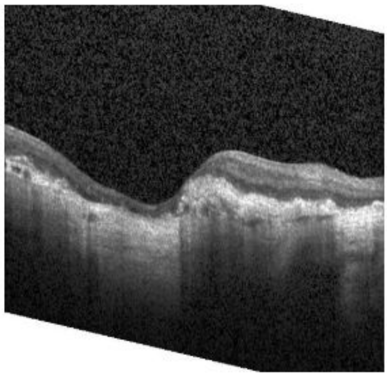
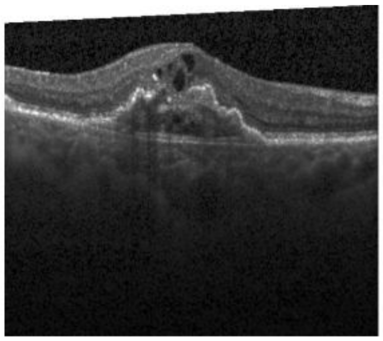
The final transfer learning solution achieves over 70% accuracy consistently. This final solution is robust enough to have solved the problem statement for this project.

V. Conclusion

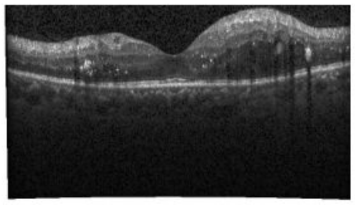
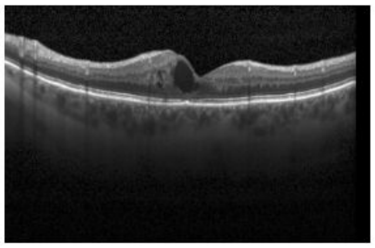
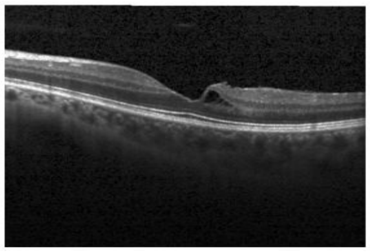
Free-Form Visualization

An important quality of the project is that it achieves a good level of accuracy. Here are a few examples where the project made the correct diagnosis prediction for a OCT scan.

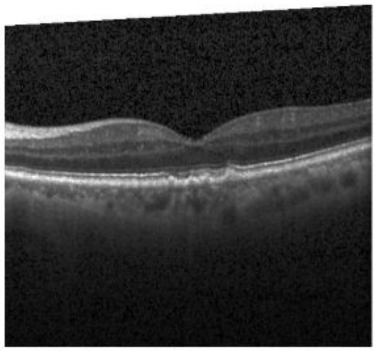
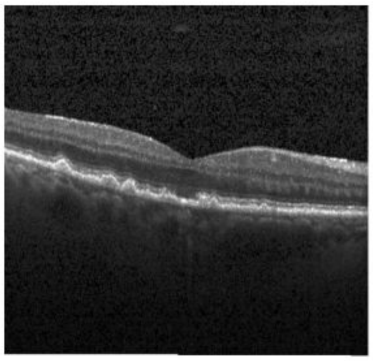
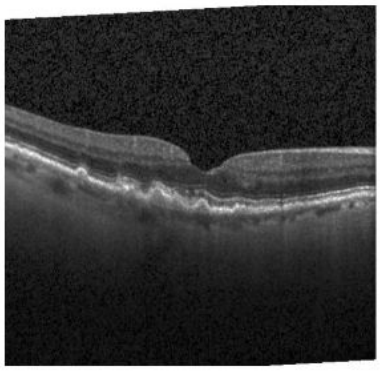
CNV

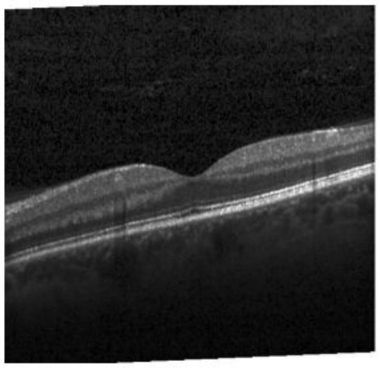
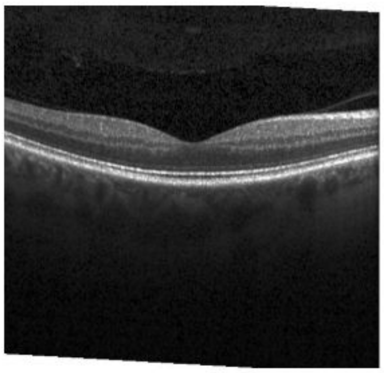
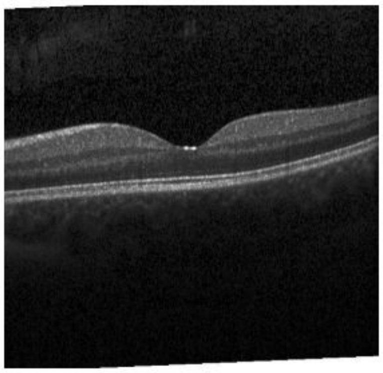
DME

DRUSEN

NORMAL

Reflection

I believe there is a lot of potential for machine learning models in the medical field, and I have been interested in building one for a long time; thankfully, Udacity has built a useful machine learning course that allowed me to create my own predictive model.

Initially, I went to kaggle.com to look for interesting datasets in the medical field and fortunately I found this idea for creating a classifier of OCT scans.

The initial article can be found here: (Kaggle OCT Scans)

I decided to use this article as a starting point for my capstone project because it states a good objective and it piqued my interest.

Once I decided on this objective and direction for my capstone project I began researching, designing and implementing it.

There were many difficulties along the way to complete this project.

The tools required where CUDA, TensorFlow and Keras. The whole setup took roughly 8hrs. Fortunately, I had the opportunity to setup these tools at one of my Udacity Connect sessions.

It that long because some installation links were broken or I installed newer versions of CUDA which are not yet supported by TensorFlow.

The first difficulty started in getting the full dataset. The kaggle dataset is incomplete and it is unorganized. However, after reading the full article, I found references that pointed to the source Mendeley.com. Fortunately, this source contains the full dataset already labeled.

I then began processing the data. It was evident from the beginning that my local computer didn’t had the necessary resources to process the whole dataset as it was too large.

I wanted to train the model using the full dataset but the resources available at the time didn’t allow me to make use of the full dataset.

Therefore, I began looking to host my algorithm in a cloud provider such as Amazon’s AWS. I requested an EC2 instance capable of such computing power but my initial request was denied by Amazon. My request was granted only after appealing the initial decision, but by then 4 days had gone by and I had already built a successful model locally.

I spent many days researching how to build a proper model using transfer learning by reading many articles online, spending time in Udacity’s classroom and reading Keras documentation and examples about the topic. After all that research, I began prototyping, and here is were a lot of my time was spent trying different configurations and approaches.

Training and testing the benchmark and transfer learning models took about 1 hr for both. So I had to make sure that I was implementing a reasonable CNN architecture and using the correct parameters to optimize my time.

I found it very interesting that creating an acceptable classifier for OCT scans, and other X-Ray images is very feasible nowadays. I would be very interested in seeing the positive impact that models like this can have in enhancing the medical field.

I would assume physicians will be open-minded to adopt machine learning models, since they can streamline their diagnosis and optimize their schedule by providing an accurate diagnosis in almost real-time.

Using the necessary computing power, sophisticated deep learning methods, and image augmentation could potentially bring the accuracy up to 95%. This accuracy could probably match that of a professional opthamologist.

The final transfer learning model presented here fits my expectations for this exercise, since it achieves above 70% accuracy and it only takes about 20 minutes to train.

The benchmark model also exceeded expectations by achieving over 50% accuracy. I initially thought that it would be difficult to get over 10% accuracy in such a basic CNN architecture and with only 5,082 training images, but this project showed otherwise.

I believe that to take this model to a production setting a professional opthamologist should be consulted to see what level of accuracy is necessary to produce production-ready diagnosis.

I believe there is a lot of potential for machine learning algorithms in the medical field and undoubtedly new business will be built in this field.

Improvement

Image augmentation should be explored to a greater extent to test if it helps achieve better accuracy.

Training the model on the full dataset should also be explored, as it should provide an accuracy improvement. Many cloud providers have instances specialized for machine learning, this option should be explored to possibly utilize the full dataset for training.

Additional classification capabilities for predicting other conditions, such as bone fractures and MRIs, could enhance the user experience as well.

This algorithm could be used to build a website. This website could help in exposing the the power of machine learning to ophthalmologists, and explore the enhancement of assisted diagnosis by a machine learning algorithm.

References:

- [1] Oquab, M., Bottou, L., Laptev, I. and Sivic, J. (2018). Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks. [online] Openaccess.thecvf.com. Available at: http://openaccess.thecvf.com/content_cvpr_2014/papers/Oquab_Learning_and_Transferring_2014_CVPR_paper.pdf [Accessed 28 May 2018].
- [2] Elson, J., Howell, J., Douceur, J. and Saul, J. (2018). Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization. [online] Microsoft.com. Available at: https://www.microsoft.com/en-us/research/wp-content/uploads/2007/10/CCS2007.pdf [Accessed 28 May 2018].

- [3] Brownlee, J. (2018). How To Implement Baseline Machine Learning Algorithms From Scratch With Python. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/implement-baseline-machine-learning-algorithms-scratch-python/ [Accessed 30 May 2018].
- [4] GitHub. (2018). keras-team/keras. [online] Available at: https://github.com/keras-team/keras/blob/master/keras/optimizers.py#L209 [Accessed 30 May 2018].
- [5] GitHub. (2018). keras-team/keras. [online] Available at: https://github.com/keras-team/keras/blob/d89afdfd82e6e27b850d910890f4a4059ddea331/keras/engine/training.py#L1392 [Accessed 30 May 2018].
- Turbert, D. (2018). What Is Optical Coherence Tomography?. [online] American Academy of Ophthalmology. Available at: https://www.aao.org/eye-health/treatments/what-is-optical-coherence-tomography [Accessed 28 May 2018].
- En.wikipedia.org. (2018). Choroidal neovascularization. [online] Available at: https://en.wikipedia.org/wiki/Choroidal_neovascularization [Accessed 28 May 2018].
- Porter, D. (2018). What Are Drusen?. [online] American Academy of Ophthalmology. Available at: https://www.aao.org/eye-health/diseases/what-are-drusen [Accessed 28 May 2018].
- Bausch.com. (2018). Diabetic Macular Edema – Information about Diabetic Macular Edema or DME : Bausch + Lomb. [online] Available at: http://www.bausch.com/your-eye-concerns/diseases-and-disorders/diabetic-macular-edema [Accessed 28 May 2018].