Module 4: Final Project

Pediatric Pneumonia Chest X-Ray Image Classification

by Steven Contreras

The problem

Dataset Context

1. From the *Cell* project: Deep Learning multi-class classification of pediatric X-Ray images into classes:
    a. Normal
    b. Pneumonia (Bacterial)
    c. Pneumonia (Viral)
2. The primary purpose of the CNN built in the *Cell* project was to classify OCT (Optical Coherence Tomography). The solution, in general, provides multi-class classification of medical imaging.
3. The approach in the *Cell* project is based on Transfer Learning.

Problem statement

This is a classic binary classification machine learning problem.

1. Build a CNN from scratch, NOT based on Transfer Learning.
2. Classify the X-Ray images (for pediatric pneumonia diagnosis) as either Normal (negative, 0) or Pneumonia (positive, 1) with high accuracy.

Training Data (X-Ray Images)

Dataset Size

5232 total images to train from

- 1349 “Normal” (ground negative)
- 3883 “Pneumonia” (ground positive)

Thus, we have **class imbalance** in the training data.

This is accounted for in the CNN by giving the ground-negative class more weight during training.

Varying Shapes

Ranging

- From 127 x 384, rayscale (one channel)
- To 2583 x 2916, RGB (three channel)

The distribution is, therefore, desirable.

This is leveraged/exploited via Data Augmentation.

Data Augmentation

“Fake” (augmented) data is generated from this desirable distribution (real training data) at a ratio of 25% with the following transformations:

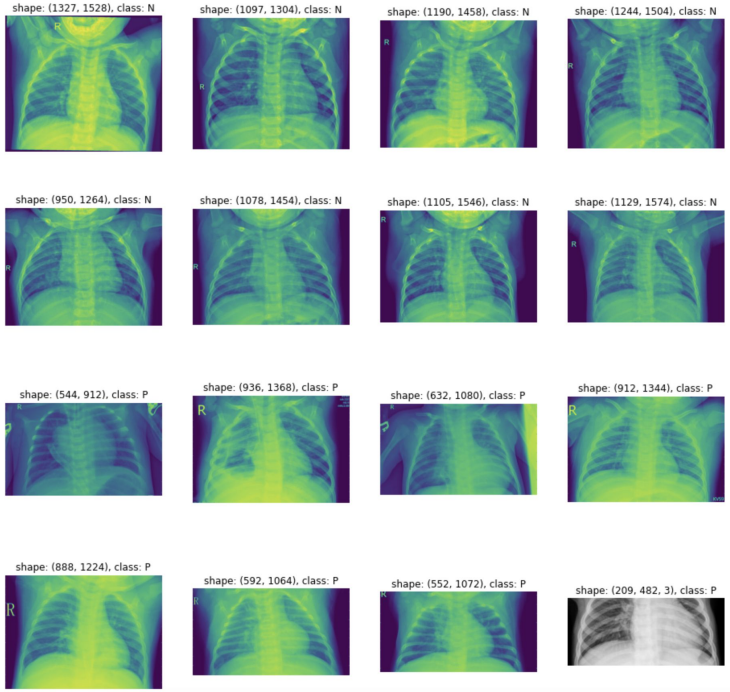
- Rotation (about some fixed point in the x-y plane)
- horizontal translation (along the x-axis)
- vertical translation (along the y-axis)
- Shearing
- Zooming
- Flipping horizontally (along the vertical axis)

The CNN Architecture: Part 1

As can be seen in the sample to the right, images are similar from a high level.

Ground-negative (“Normal”) images are in the top two rows. Similarly, ground-positive (“Pneumonia”) images are in the bottom two rows. To the untrained human eye, at a high level, the differences may not be immediately obvious.

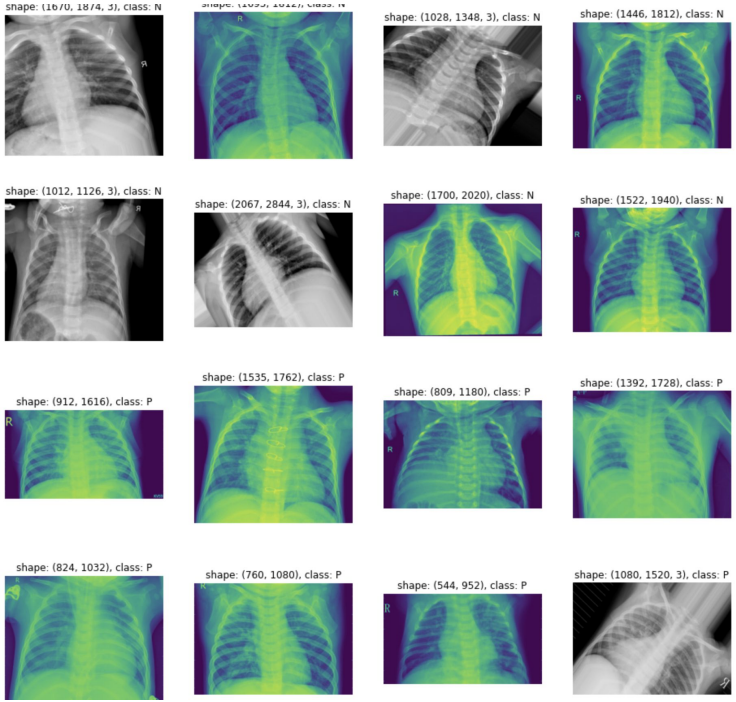
It seems reasonable to create a Convolution Neural Network architecture such that **features and details become increasingly discernible with the “magnification” of/by each subsequent convolution**, as we delve deeper into the network.



The CNN Architecture: Part 2

This is emphasized with the use of the augmented (“fake”) images generated (transformed) from the existing distribution of training images, as seen to the right.

Note:
It’s a bit unnerving how “real” these generated images are.



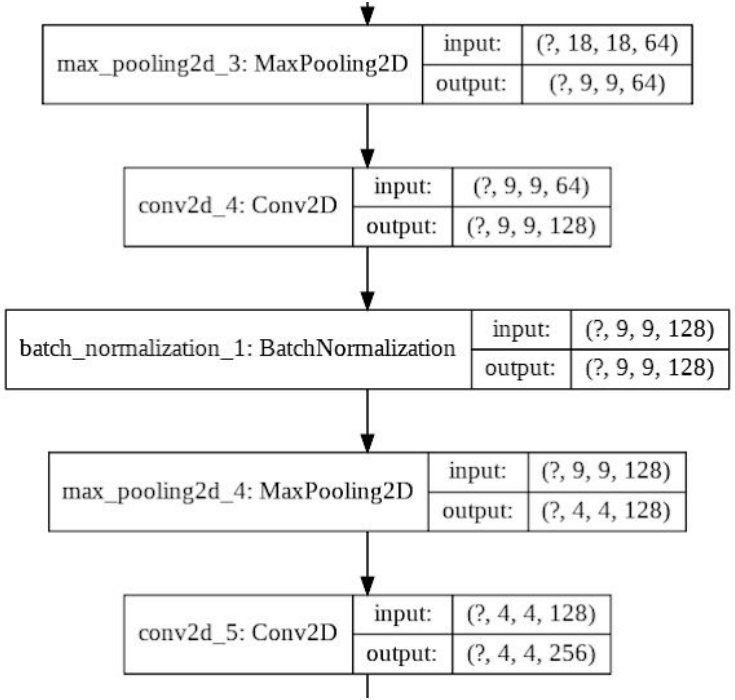
The CNN Architecture: Part 3

A section of final architecture is visualized to the right - the network is too big to fit on a single slide.

But here are the general organizing principles:

1. Input (images) are scaled and normalized to 150x150x3.
2. The first convolution begins with 8 3x3 “same” padding filters.
3. Every layer ends with Max 2x2 pooling - thus, its output is halved.
4. Because of this, the number of filters in the subsequent layer is doubled; e.g. the layer 2 convolution has 16 3x3 “same” padding filters

(continued on next slide)



The CNN Architecture: Part 4

(continued) organizing principles:

5. This pattern continues until the final convolution layer, which consists of 512 3x3 “same” padding filters
6. Variance becomes an issue the deeper into the network we go, so BatchNormalization occurs in between Max Pooling output, beginning with layer 4 (64 3x3 “same” padding filters)
7. The output of the final convolutional layer, layer 7 (512 3x3 “same” padding filters, BatchNormalization, MaxPooling) is flattened for input to the fully connected, dense layers following

(continued on next slide)

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 150, 150, 3)]	0
conv2d (Conv2D)	(None, 150, 150, 8)	224
max_pooling2d (MaxPooling2D)	(None, 75, 75, 8)	0
conv2d_1 (Conv2D)	(None, 75, 75, 16)	1168
max_pooling2d_1 (MaxPooling2	(None, 37, 37, 16)	0
conv2d_2 (Conv2D)	(None, 37, 37, 32)	4640
max_pooling2d_2 (MaxPooling2	(None, 18, 18, 32)	0
conv2d_3 (Conv2D)	(None, 18, 18, 64)	18496
batch_normalization (BatchNo	(None, 18, 18, 64)	256
max_pooling2d_3 (MaxPooling2	(None, 9, 9, 64)	0
conv2d_4 (Conv2D)	(None, 9, 9, 128)	73856
batch_normalization_1 (Batch	(None, 9, 9, 128)	512
max_pooling2d_4 (MaxPooling2	(None, 4, 4, 128)	0

The CNN Architecture: Part 5

(continued) organizing principles:

8. There is only one dense layer, consisting of 1024 units, with dropout at a rate of 20%
9. Regulated Linear Unit (ReLu) activation is of course used for every layer (block) with the exception of the final output layer, which of course using sigmoid activation for binary classification

```
conv2d_5 (Conv2D)            (None, 4, 4, 256)         295168
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 256)         1024
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 2, 2, 512)         1180160
_________________________________________________________________
batch_normalization_3 (Batch (None, 2, 2, 512)         2048
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 1, 1, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 1024)              525312
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 1025
=================================================================
Total params: 2,103,889
Trainable params: 2,101,969
Non-trainable params: 1,920
_________________________________________________________________
```

Optimization, Learning Rate Adjustment, Etc.

Optimization

The **Adam** optimizer is used as it can address both bias and variance correction via “adaptive estimation of first-order and second-order moments”.

Default values of exponential decay rates of first/second moments are used, 0.9 and 0.999, respectively.

Learning Rate Adj.

The **ReduceLROnPlateau** callback reduces the learning rate geometrically by a factor of 0.8 whenever the validation loss does not increase for two consecutive epochs.

The idea is that the gradients in this case are on the “right track”, so we want to discourage overshooting to the wrong trajectory.

Accuracy Threshold

A custom callback monitors accuracy and validation accuracy. If BOTH metrics surpass the threshold (97.5%), training terminates early.

Note that this is NOT the same thing as using the EarlyStopping callback, which will halt training if a particular metric stops changing beyond some delta.

Results

1. **With GPU, training time averaged to just under 3 minutes!**
2. Surpassed accuracy threshold at epoch 10/20 (BatchNormalization facilitates fast convergence)

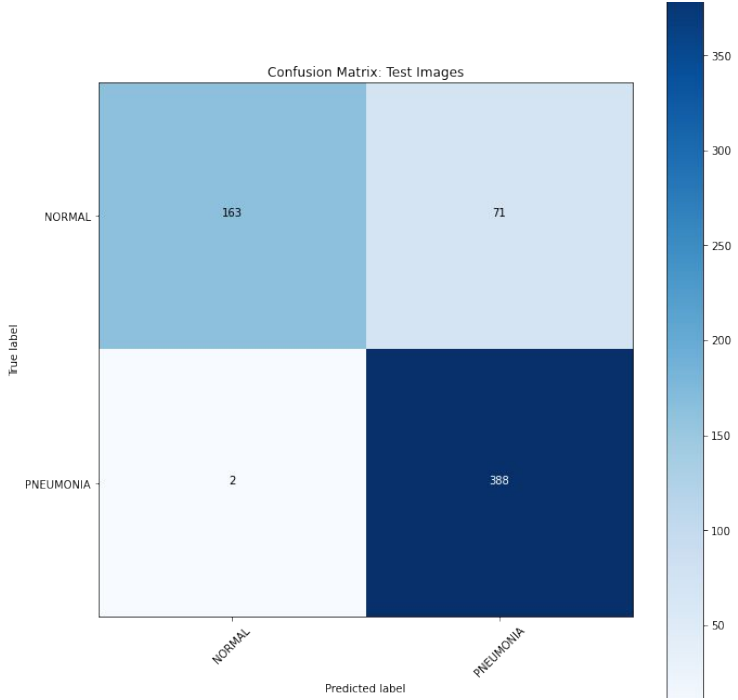
- *Training*
  - *Accuracy*: **99.37%**
  - *Loss*: **0.0146**
- *Validation*:
  - *Accuracy*: **97.69%**
  - *Loss*: **0.0942**
- ***TEST***:
  - *Accuracy*: **88.30%**
  - *Loss*: **0.5591**

False-Positives vs. False Negatives

Classification Report:

	precision	recall	f1-score	support
NORMAL	0.99	0.70	0.82	234
PNEUMONIA	0.85	0.99	0.91	390
accuracy			0.88	624
macro avg	0.92	0.85	0.87	624
weighted avg	0.90	0.88	0.88	624

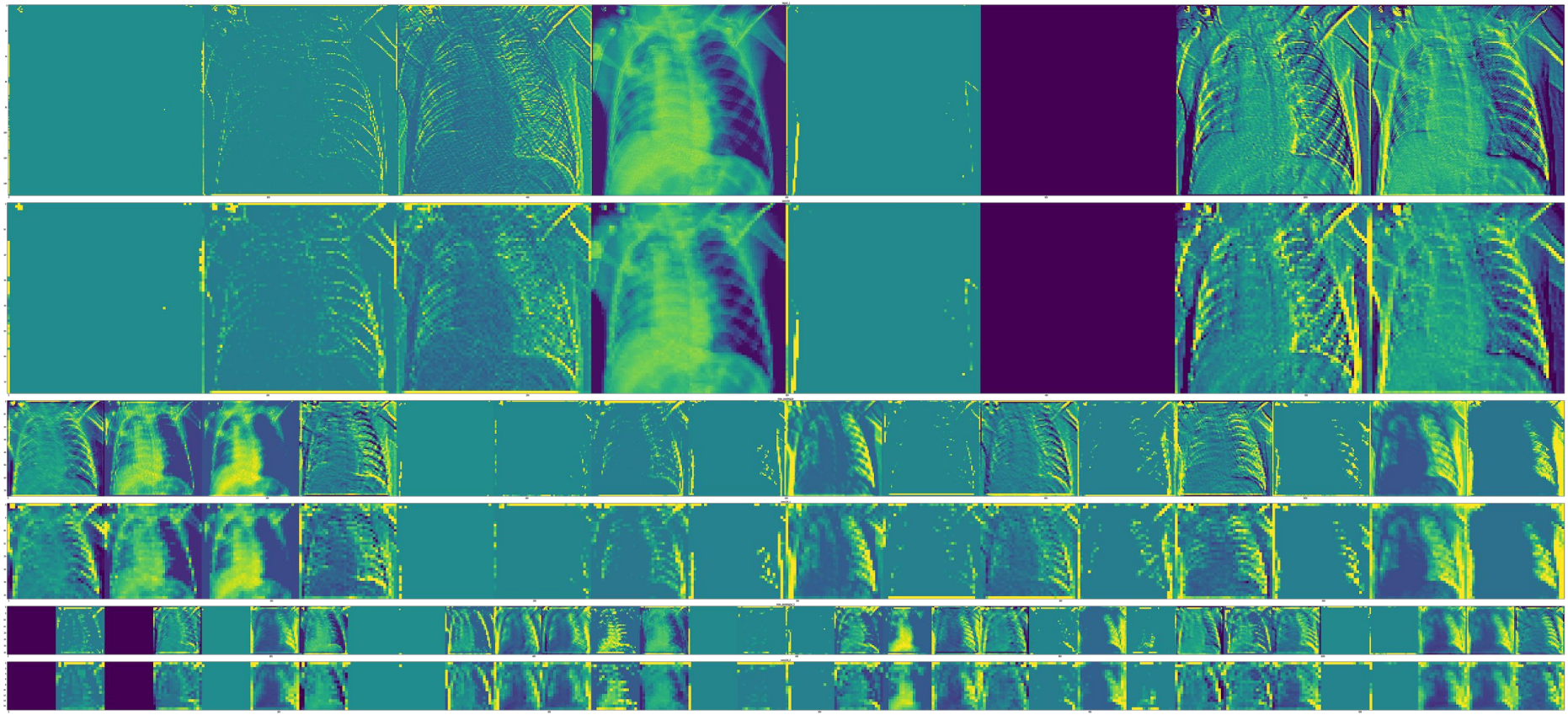
As you can see above and to the right, this model performs very well, predicting only 2 False out of 234 True Negatives and 71 False out of 390 True Positives from the Test set. But, in this case, because a False-Negative is far worse than a False-Positive, this model is considered superior.



What the CNN “sees”

Convolutional Filtering

The First 3 Layers (Positive Example)



Conclusion and Future Work Consideration

Thank you so much for your time!

Possible Future Work:

- **Investigate techniques to improve performance on unseen data; in particular, dive deeper into data augmentation**
  - **Try a larger ratio**
  - **More aggressive data augmentation hyper-parameters (e.g. rotation angle range)**
- **Integrate Transfer Learning (e.g. ImageNet)**