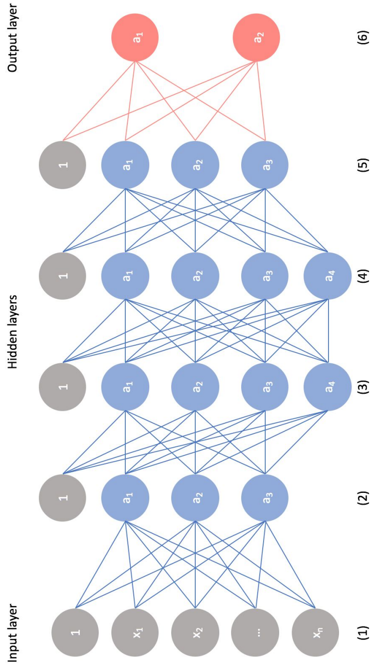


https://www.jeremyjordan.me/convolutional-neural-networks/

Convolutional Neural Network Team

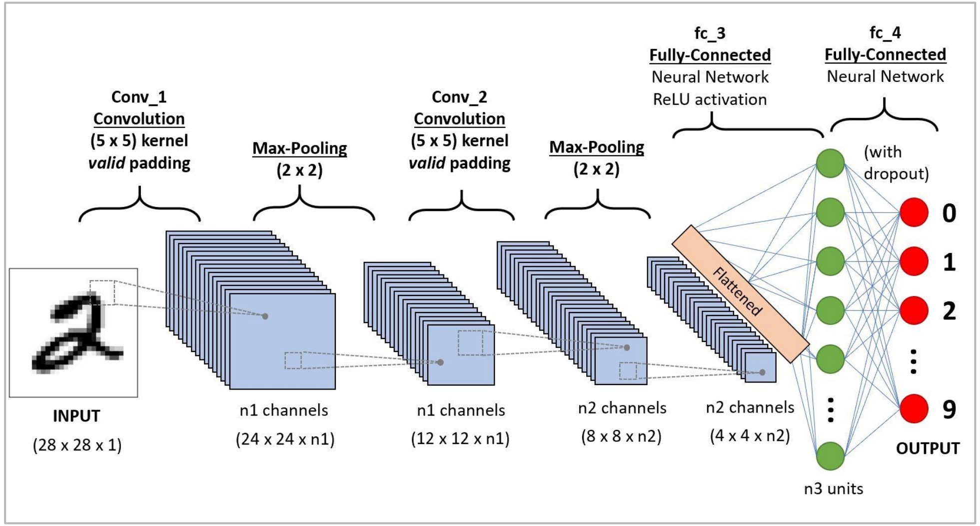
Stacie Barbarick, Cass Bliss, Joanna Harden

https://www.istockphoto.com/photos/neurons-white-background

Review:

- CNN: neural network that typically contains several types of layers, including a convolutional layer, a pooling layer, and activation layers.
- Can find and extract useful features from these images
- Very specific features can be detected anywhere on input images
- Different Models of CNNs we explored:

  EX: VGG16, ResNet50, Custom CNN



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Stacie

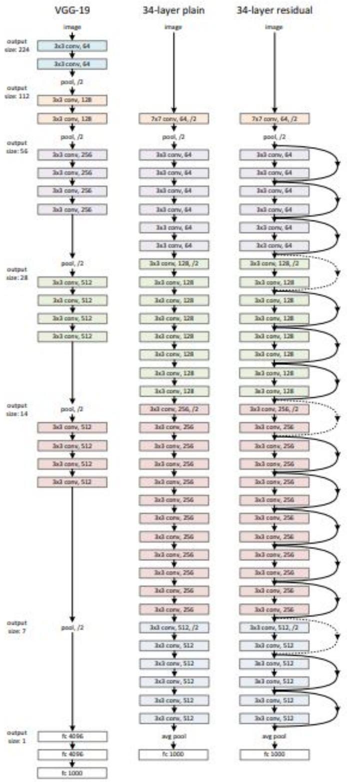


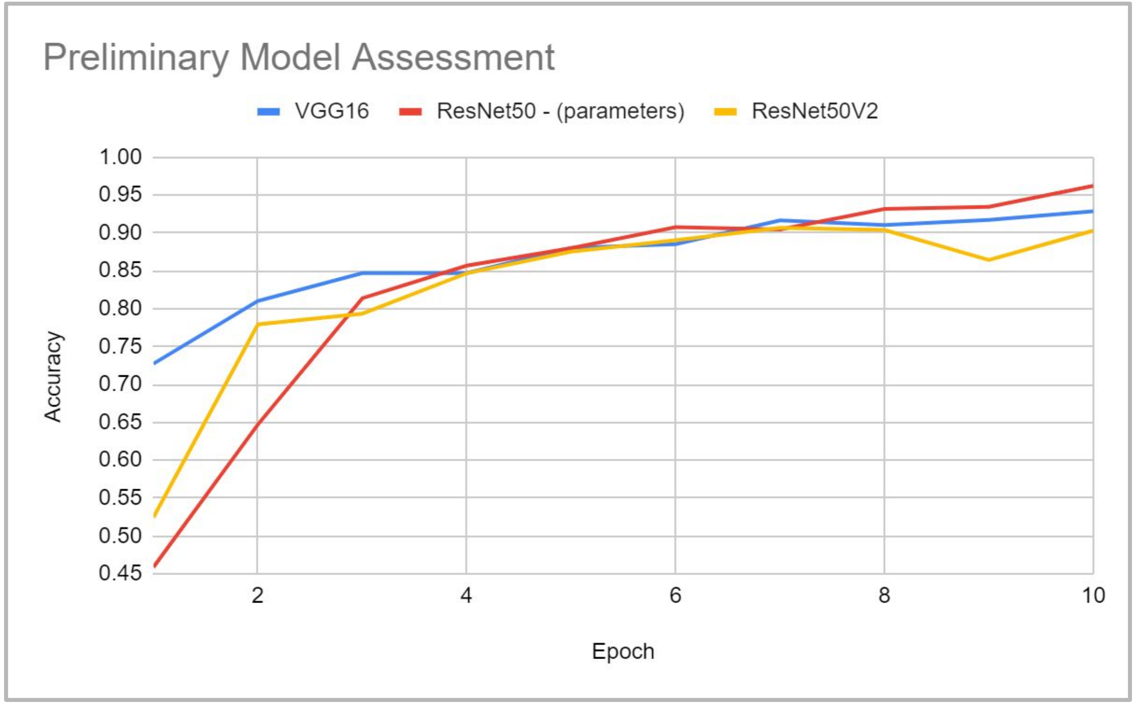
Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

Residual Network (ResNet)

- Allowed training of deep neural networks (150+layers) successfully.
- The main benefit of a very deep network: can represent very complex functions and learn features at many different levels of abstraction.
- However, when the network depth increasing, accuracy gets saturated
- Explicitly let stacked layers fit a residual mapping
- Plain baselines are mainly inspired by the philosophy of VGG nets.
- The shortcut connections perform identity mapping, and their outputs are added to the outputs of the stacked layers. Identity shortcut connections add neither extra parameter nor computational complexity.

He, Kaiming, et al. "Deep Residual Learning for Image Recognition." *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 10 Dec. 2015, doi:10.1109/cvpr.2016.90.

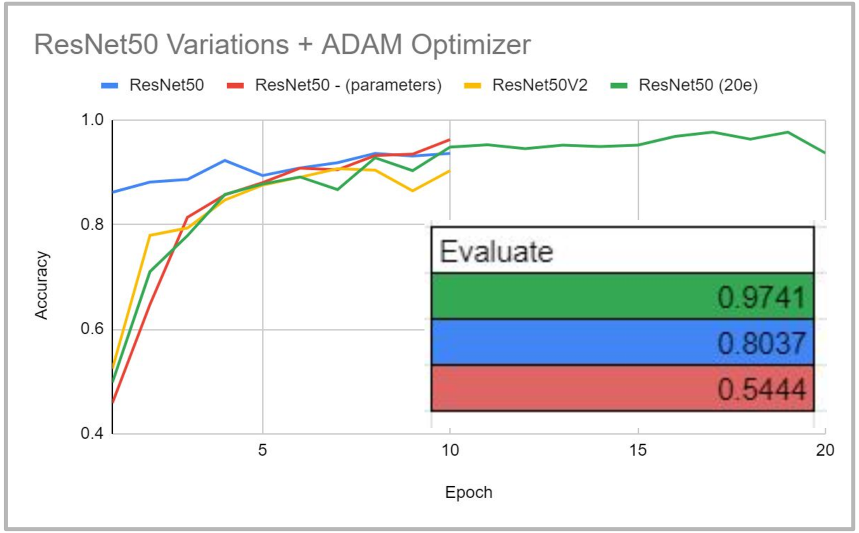
Stacie

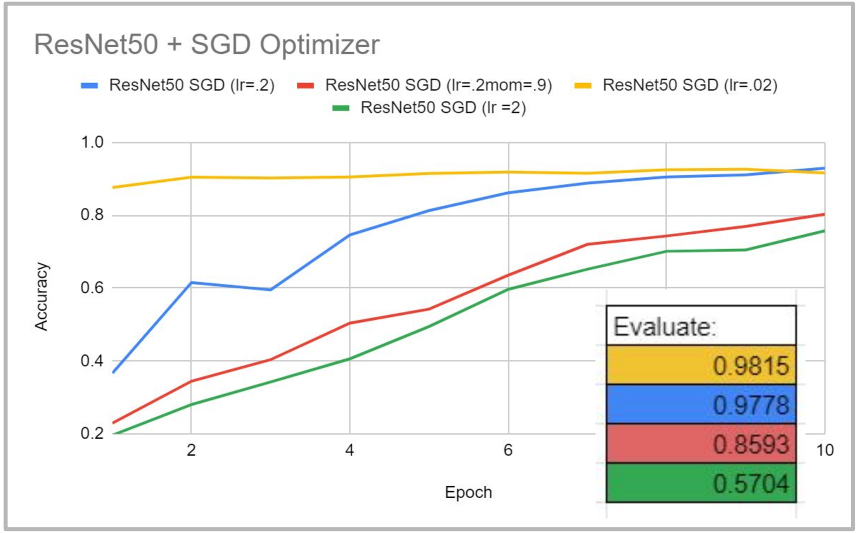


Preliminary training with variety of architecture like VGG16, ResNet50 (w/o most parameters), ResNet50v2

However, no evaluate function, no validation data evaluation included

Stacie

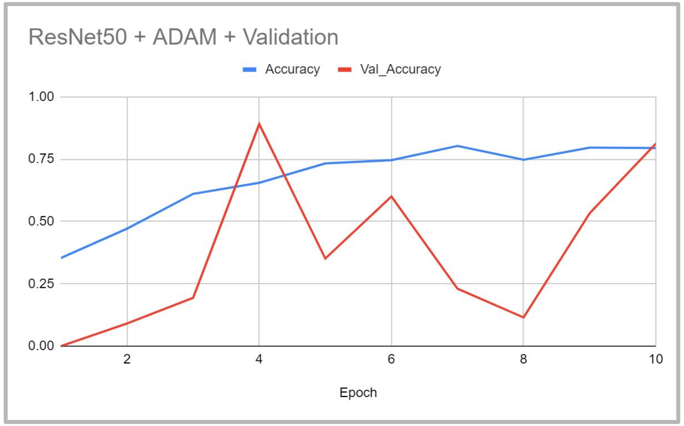


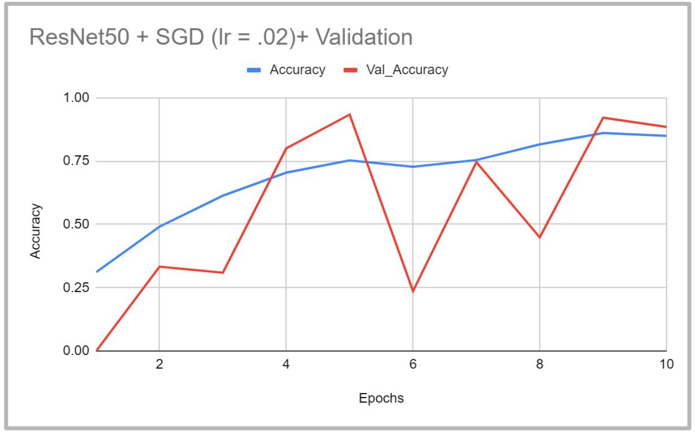


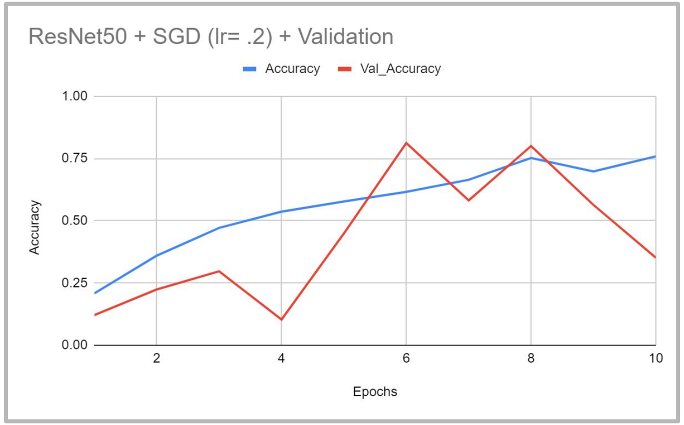
ResNet50 with varied parameters and optimizers and variations within the optimizer (SGD) - Evaluate function applied but no Validation Data

Stacie

ResNet 50 with differing optimizers; plotting accuracy and validation







ADAM
Accuracy: .7946
Validation Accuracy: .8121
SGD (lr =.2)
Accuracy: .7584
Validation Accuracy: .3515
SGD (lr =.02)
Accuracy: .8497
Validation Accuracy: .8848

Cass

My Chaotic & Obsessive Process:

- Taking the general CNN model, rerouting data input
- Taking away layers
- Adding Conv 2D layers (smaller and bigger filters)
- Trying different batch values
- Trying different epoch values
- Really didn’t touch the dense/ flatten layers
- Looked for higher starting accuracy before committing to full run throughs
- Ended up with a funnel shaped conv2d and pooling layer structure
- Accidentally double trained model

Cass

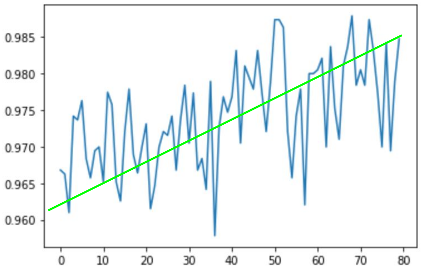
Top Model:

“Miraculum”

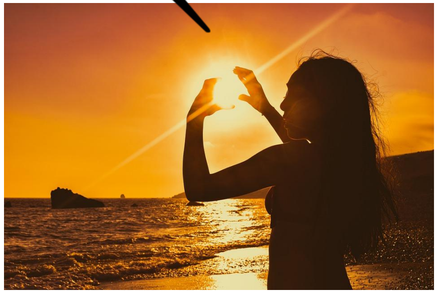
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param
=================================================================
conv2d (Conv2D)              (None, 296, 296, 16)      416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 148, 148, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 144, 144, 32)      12832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 72, 72, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 68, 68, 64)        51264
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 34, 34, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 73984)             0
_________________________________________________________________
dense (Dense)                (None, 100)               73985
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                2020
_________________________________________________________________
dense_2 (Dense)              (None, 10)                210
=================================================================
Total params: 7,465,242
Trainable params: 7,465,242
Non-trainable params: 0
```

Epochs	Batch size	Accuracy
80	15	100%

****Double Trained****



You can call me Osirus



Cass

2nd Place Model: “Nuper Nocte”

Epochs	Batch Size	Accuracy
80	20	99.81%

```
[25]: model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 296, 296, 16)      416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 148, 148, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 144, 144, 32)      12832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 72, 72, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 68, 68, 64)        51264
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 34, 34, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 73984)             0
_________________________________________________________________
dense (Dense)                (None, 100)               7398500
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                2020
_________________________________________________________________
dense_2 (Dense)              (None, 10)                210
=================================================================
Total params: 7,465,242
Trainable params: 7,465,242
Non-trainable params: 0
_________________________________________________________________
```

Using `model.evaluate` on the generator

```
[31]: # Set the step size based on the amount of data and the batch size.
STEP_SIZE_TEST=test_generator.n//test_generator.batch_size

# Reset the gererator.
test_generator.reset()

# Evaluate and score all the data in the generator.
pred=model.evaluate(test_generator, steps=STEP_SIZE_TEST, verbose=1)

WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
1/1 [==============================] - 67s 67s/step - loss: 0.0110 - accuracy: 0.9981
```

```
For wavelength 131 : (array([0]), array([23]))
For wavelength 1600 : (array([1]), array([70]))
For wavelength 1700 : (array([2]), array([71]))
For wavelength 171 : (array([3]), array([70]))
For wavelength 193 : (array([4, 5]), array([68,  1]))
For wavelength 211 : (array([5]), array([49]))
For wavelength 304 : (array([6]), array([71]))
For wavelength 335 : (array([7]), array([23]))
For wavelength 4500 : (array([8]), array([71]))
For wavelength 94 : (array([9]), array([23]))
```

Cass

3rd place model: “Curiositas”

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 296, 296, 16)      416
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 148, 148, 16)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 144, 144, 32)      12832
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 72, 72, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 68, 68, 64)        51264
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 34, 34, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 30, 30, 128)       204928
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 15, 15, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 28800)             0
_________________________________________________________________
dense_3 (Dense)              (None, 100)               2880100
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_4 (Dense)              (None, 20)                2020
_________________________________________________________________
dense_5 (Dense)              (None, 10)                210
=================================================================
Total params: 3,151,770
Trainable params: 3,151,770
Non-trainable params: 0
```

Epochs	Batch Size	Accuracy
80	10	98.51

```
[49]: A = np.sum(np.unique(Y_, return_counts=True)[1])
      B = np.sum(np.abs(np.flip(np.unique(images_pull[1], return_counts=True, axis =0)[1]) - np.unique(Y_, return_counts=True)[1]))

      Score = 1 - B / (2 * A)
      print(Score)

      0.9851851851851852
```

```
For wavelength 131 : (array([0]), array([23]))
For wavelength 1600 : (array([1]), array([70]))
For wavelength 1700 : (array([2]), array([71]))
For wavelength 171 : (array([3]), array([70]))
For wavelength 193 : (array([3, 4, 5]), array([ 1, 67,  1]))
For wavelength 211 : (array([5]), array([49]))
For wavelength 304 : (array([6]), array([71]))
For wavelength 335 : (array([0, 7, 9]), array([ 5, 17,  1]))
For wavelength 4500 : (array([8]), array([71]))
For wavelength 94 : (array([9]), array([23]))
```

Joanna - VGG16 Network

- Started with full build of VGG 16 Model
- Pulled in Keras VGG Model
- Trained four different attempts:
  - 300x300, Epochs=10, Batch Size=5
  - 32x32, Epochs=10, Batch Size=50
  - 32x32, Epochs=30, Batch Size=100
  - 32x32, Epochs=80, Batch Size=100
- Accuracy hovered around .08 - .10

```
model.add(Conv2D(64, kernel_size=(3,3),activation='relu',input_shape=(300, 300, 1)))
model.add(Conv2D(64, kernel_size=(3,3),activation='relu',input_shape=(300, 300, 1)))
model.add(MaxPooling2D(pool_size=(3, 3),strides=2))

model.add(Conv2D(128, kernel_size=(3,3),activation='relu'))
model.add(Conv2D(128, kernel_size=(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Conv2D(256, kernel_size=(3,3),activation='relu'))
model.add(Conv2D(256, kernel_size=(3,3),activation='relu'))
model.add(Conv2D(256, kernel_size=(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Conv2D(512, kernel_size=(3,3),activation='relu'))
model.add(Conv2D(512, kernel_size=(3,3),activation='relu'))
model.add(Conv2D(512, kernel_size=(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Conv2D(512, kernel_size=(3,3),activation='relu'))
model.add(Conv2D(512, kernel_size=(3,3),activation='relu'))
model.add(Conv2D(512, kernel_size=(3,3),activation='relu'))

model.add(Flatten())

model.add(Dense(100, activation='relu')) # 4096 is number of nodes
#model.add(Dropout(.5))
model.add(Dense(20, activation='relu'))
```

Joanna - Dense Network v1

- Made up of one Flatten layer and four Dense layers (27 mil. parameters)
- Trained two attempts
  - 300x300, Epochs=10, Batch Size=5 — Accuracy ~.80
  - 300x300, Epochs=30, Batch Size=5 — Accuracy ~.99
- Saved model and tested it — Acc. ~99.26%
- Misclassified 4 of the 540 testing images:

DenseNet_v1

Layer (type)	Output Shape
flatten (Flatten)	(None, 90000)
dense (Dense)	(None, 300)
dense_1 (Dense)	(None, 200)
dense_2 (Dense)	(None, 100)
dense_3 (Dense)	(None, 10)

```
For wavelength 193 : (array([3, 4, 5]), array([ 3, 65,  1]))
```

Joanna - Dense Network v2 + v3

- Trained each
  - V2 - Epochs=30, Batch Size=5 — Accuracy ~.98
  - V3: Double-Trained, Epochs=10, Batch Size=10 — Accuracy ~.99
- Saved both models and tested them
  - V2 — Acc. ~99%
  - V3 — Acc. ~99%
- Both misclassified 6 of the 540 testing images

DenseNet_v2

```
Layer (type)                 Output Shape
==========================================
flatten_2 (Flatten)          (None, 900)
__________________________________________
dense_8 (Dense)              (None, 30)
__________________________________________
dense_9 (Dense)              (None, 20)
__________________________________________
dense_10 (Dense)             (None, 10)
__________________________________________
dense_11 (Dense)             (None, 10)
------------------------------------------
```

DenseNet_v3

```
Layer (type)                 Output Shape
==========================================
flatten (Flatten)            (None, 900)
__________________________________________
dense (Dense)                (None, 300)
__________________________________________
dense_1 (Dense)              (None, 200)
__________________________________________
dense_2 (Dense)              (None, 100)
__________________________________________
dense_3 (Dense)              (None, 10)
```

Conclusions

- CNNs are ideal for scanning images for incredible details (i.e. sun spots) but may be unnecessary for tasks like identifying solar wavelength data. (i.e. dense layers only result)
- Network optimization can be incredibly time consuming: payoff between initial network training and accuracy?
- Different architectures and optimization parameters may be more suited for different image training goals

Recommendations

Image Preprocessing:

- Highly recommend having code capable of preprocessing any solar image in the following ways for applicability
  - Greyscale
  - Similar resolution (maybe .nearest() function) to what we have been working with
- Otherwise applicability is very limited & models do not work