Capstone Project: Malaria Detection

Final Submission

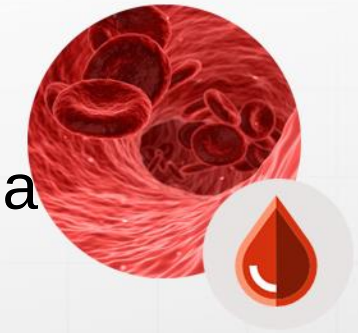
Mike Hankinson
April 21, 2022

Slide Members

Questions to Answer…

- Can we build a convolutional neural network to detect malaria?
- Is the available dataset sufficient to build an effective model?
- Can the computer vision model compete with the diagnostic accuracy of a trained professional?
- Can the computer vision model meet both capital and expense budget expectations?

Executive Summary



- A computer vision model has been developed that can detect malaria in a blood spear with 98% accuracy and recall.
- The following further work is warranted:

Business

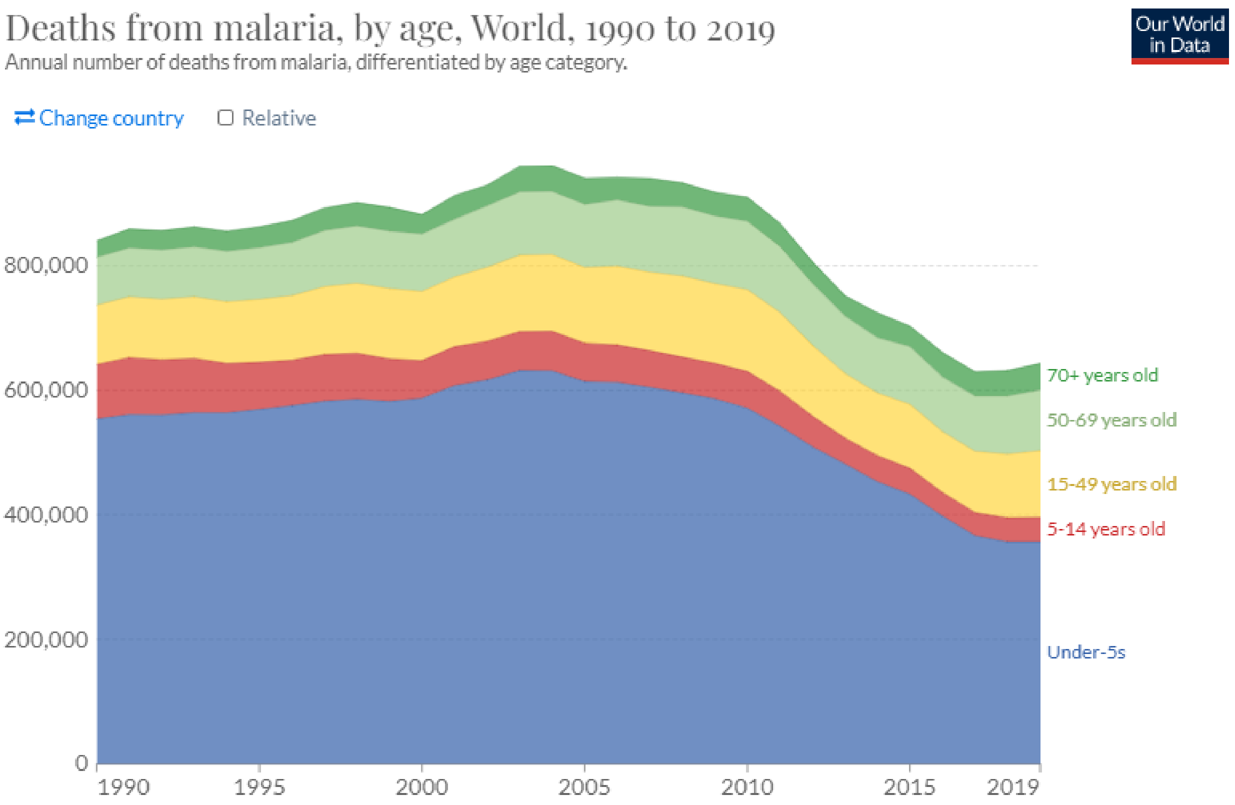
- Perform cost estimate to determine if Project Management Gate 4 **Capex** expenditures are still in alignment with previous stage 3 estimates.
- Develop rollout vision for market testing
- Complete expense analysis based upon rollout vision
- Schedule end of Q2 review with the management board

Technical

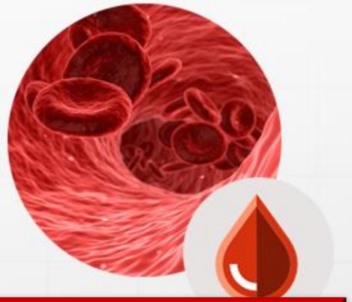
- Continue development of state-of-art Transfer Model. Target 99% accuracy/recall.
- Verify diagnostic accuracy of current technology in our market testing location.

Why this Project…

Malaria is a life-threatening disease caused by the plasmodium parasite and transmitted by the bite of infected mosquitoes.



https://ourworldindata.org/malaria#



BAD NEWS

- In 2019, 645,000 worldwide died from Malaria
- 55% of deaths are under the age of 5
- Late treatment can be fatal
- Clinical diagnosis requires a blood smear interpreted by an experienced specialist. (Expensive and time consuming)

….and the good

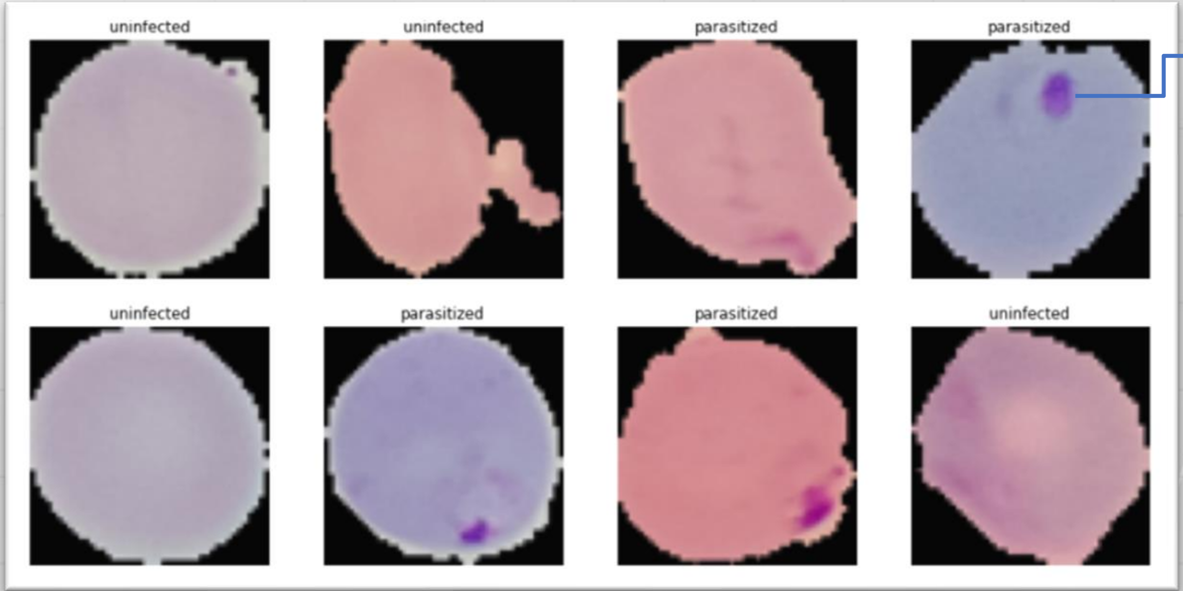
- A 33% reduction in deaths from malaria since 2004.
- Treatment regimens have improved including more effective and specifically tailored (to age, type of malaria, degree of sickness) medications.

Source Data

- ➢The data set was provided in the form of zipped folders containing PNG files.
- ➢The dataset contains a total of 27,558 images segregated as:

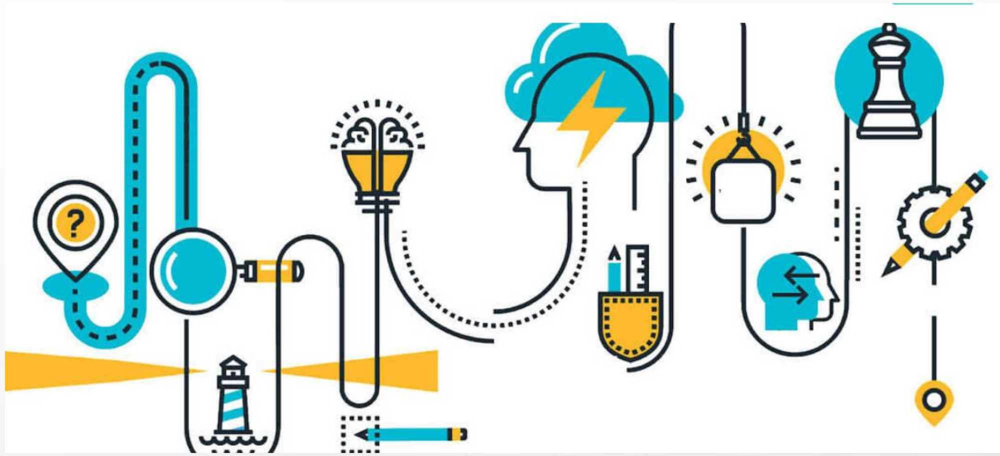
	Train Data	Test Data
Count of Parasitized	12,582	1,300
Count of Uninfected	12,376	1,300

Well balanced



Parasitized blood cells contain color ‘anomalies’

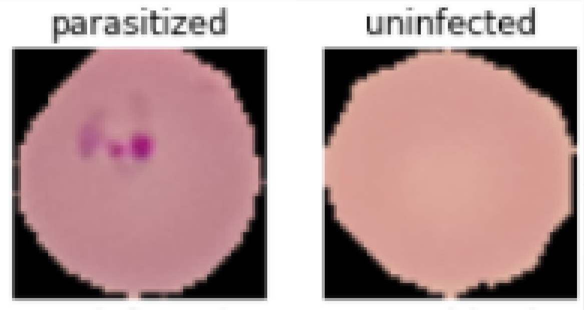
Data Preprocessing



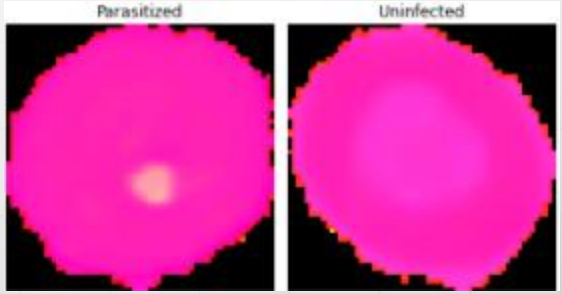
- ➢Read the picture files
- ➢Encode the PNG image to RGB grids of pixels with channels.
- ➢Convert into floating-point tensors (for CNN) input.
- ➢Normalize the pixel values (between 0 and 255) to the [0, 1] interval.

Data Preprocessing

- Decide on Color Space - (Contrast Enhancement)
  - ✓RGB

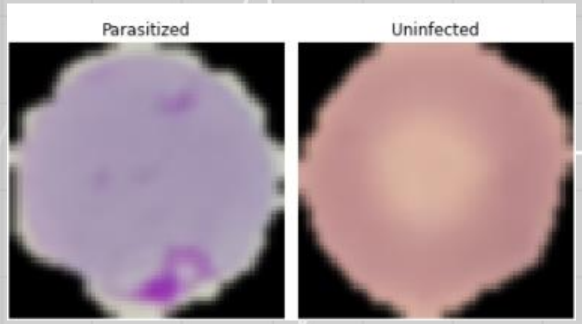


  - ✓HSV



Parasitized blood cells maintain color ‘anomalies’ In both RGB and HSV

  - ✓YCbCr



- Apply Gaussian Blurring/Smoothing (Reduce Noise)
- One Hot Encode labels

Model Development

Ideal Model for Feature Learning:
- Spatial locality detectors
- Weight sharing (requires << fully connected NN)
- Pooling (reduces output dimension)

Define the model: Convolutional Neural Network

- ✓ Activation Function
- ✓ Number of Layers
- ✓ Number of Neurons per Layer
- ✓ Neuron Dropout
- ✓ Max Pooling
- ✓ Flattening
- ✓ Define Loss Function
- ✓ Define Optimizer
- ✓ Determine the number of epochs
- ✓ Success Metric –Iterate/min. loss function

How to Build the Model: What different techniques should be explored?

Color Space
RGB
HSL
HSV, etc.

Data Augmentation
Rotation
Cropping
Noise, Shear, etc.

Transfer Learning
AlexNet
VGGNet
Inception, etc.

Optimizer
Adam
Adamax
SGD, etc.

Hyperparameter Tuning
If so, which parameters

Weight Initialization
Random
Xavier distribution
He Gaussian method

Learning Rate

Comparison of Results

Top Performing Models

Model	Augment	Feature Learning	Classifier	Test Accuracy	Recall
Base		3 Convolutional layers (relu) each containing max-pooling layers and 20% dropout	Fully connected dense layer (512) with 40% dropout Fully connected output layer (2) with softmax activation	0.97	0.97
Model 1		4 Convolutional layers (tanh) each containing max-pooling layers and 20% dropout	Fully connected dense layer (512) with 40% dropout Fully connected output layer (2) with softmax activation	0.95	0.95
Model 2		4 Convolutional layers (LeakyReLU 0.1) each containing max-pooling layers and 20% nueron dropout	Fully connected dense layer (512) with 40% dropout Fully connected output layer (2) with softmax activation	0.98	0.98
Model 3	☑	**Same as Model 2**	**Same as Model 2**	0.98	0.98
Model 4		**Transfer Learning**: VGG16 model Flatten the output from the 5th block of the VGG16 model (Models 4&5 use identical feature learning)	Fully connected dense layer (256) Fully connected dense layer (128) with 30% dropout Fully connected dense layer (64) Fully connected output layer (2) with softmax activation Vary patience and optimization functions (Models 4-7 use identical Classifiers)	0.95	0.95
Model 5	☑	**Transfer Learning**: VGG16 model Flatten the output from the 5th block of the VGG16 model (Models 4&5 use identical feature learning)	Vary optimization functions (Models 4-7 use identical Classifiers)	0.92	0.96
Model 6	☑	**Transfer Learning**: InceptionV3 model Flatten the output	Vary optimization functions (Models 4-7 use identical Classifiers)	0.94	0.94
Model 7	☑	Hyperparameter Tuning with KerasTuner		0.90	0.97

Increasing Model Complexity

Transfer Learning: Further Development

*** See Appendix for full model details**

Insights

- Most models performed to acceptable levels for field implementation.
- Sometimes the simplest technical solutions are best.
- Models 2 and 3 are currently the optimum choice for the current implementation.
- Further study should be performed incorporating more recent advancements in Transfer Learning Models.
- See Appendix for more detailed insights into data augmentation, transfer learning, transfer learning patience and optimizer performance comparison.

Appendix Slides

Feature Learning

Classifier

Model 1

Sequential

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	416
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
dropout (Dropout)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 32, 32, 32)	4128
max_pooling2d_1 (MaxPooling 2D)	(None, 16, 16, 32)	0
dropout_1 (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 32)	4128
max_pooling2d_2 (MaxPooling 2D)	(None, 8, 8, 32)	0
dropout_2 (Dropout)	(None, 8, 8, 32)	0
conv2d_3 (Conv2D)	(None, 8, 8, 32)	4128
max_pooling2d_3 (MaxPooling 2D)	(None, 4, 4, 32)	0
dropout_3 (Dropout)	(None, 4, 4, 32)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 512)	262656
dropout_4 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 2)	1026

Total params: 276,482
Trainable params: 276,482
Non-trainable params: 0

Model 2

Sequential Adding Batch Normalization

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	896
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
dropout (Dropout)	(None, 32, 32, 32)	0
batch_normalization (BatchN ormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_1 (MaxPooling 2D)	(None, 16, 16, 32)	0
dropout_1 (Dropout)	(None, 16, 16, 32)	0
batch_normalization_1 (Batc hNormalization)	(None, 16, 16, 32)	128
conv2d_2 (Conv2D)	(None, 16, 16, 32)	9248
max_pooling2d_2 (MaxPooling 2D)	(None, 8, 8, 32)	0
dropout_2 (Dropout)	(None, 8, 8, 32)	0
batch_normalization_2 (Batc hNormalization)	(None, 8, 8, 32)	128
conv2d_3 (Conv2D)	(None, 8, 8, 32)	9248
max_pooling2d_3 (MaxPooling 2D)	(None, 4, 4, 32)	0
dropout_3 (Dropout)	(None, 4, 4, 32)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 512)	262656
dropout_4 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 512)	262656
dropout_5 (Dropout)	(None, 512)	0
dropout_6 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 2)	1026

Total params: 555,362
Trainable params: 555,170
Non-trainable params: 192

Model 3

Sequential Adding Data Augmentation

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	896
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
dropout (Dropout)	(None, 32, 32, 32)	0
batch_normalization (BatchN ormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_1 (MaxPooling 2D)	(None, 16, 16, 32)	0
dropout_1 (Dropout)	(None, 16, 16, 32)	0
batch_normalization_1 (Batc hNormalization)	(None, 16, 16, 32)	128
conv2d_2 (Conv2D)	(None, 16, 16, 32)	9248
max_pooling2d_2 (MaxPooling 2D)	(None, 8, 8, 32)	0
dropout_2 (Dropout)	(None, 8, 8, 32)	0
batch_normalization_2 (Batc hNormalization)	(None, 8, 8, 32)	128
conv2d_3 (Conv2D)	(None, 8, 8, 32)	9248
max_pooling2d_3 (MaxPooling 2D)	(None, 4, 4, 32)	0
dropout_3 (Dropout)	(None, 4, 4, 32)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 512)	262656
dropout_4 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 512)	262656
dropout_5 (Dropout)	(None, 512)	0
dropout_6 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 2)	1026

Total params: 555,362
Trainable params: 555,170
Non-trainable params: 192

Feature Learning

Classifier

Model 4

VGG16 Transfer Learning Varying Patience and Optimization Function

```
Model: "vgg16"

Layer (type)                 Output Shape              Param #
===============================================================
input_1 (InputLayer)         [(None, 64, 64, 3)]       0

block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792

block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928

block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0

block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856

block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584

block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0

block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168

block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080

block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080

block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0

block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160

block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808

block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808

block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0

block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808

block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808

block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808

block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0

===============================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

```
layers.Flatten()(base_model.output)
layers.Dense(256, activation='relu')(x)
layers.Dense(128, activation='relu')(x)
layers.Dropout(0.3)(x)
layers.Dense(64, activation='relu')(x)
layers.Dense(2, activation='sigmoid')(x)
```

Model 5

VGG16 Transfer Learning with Augmentation and Varying Optimization

```
Model: "vgg16"

Layer (type)                 Output Shape              Param #
===============================================================
input_1 (InputLayer)         [(None, 64, 64, 3)]       0

block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792

block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928

block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0

block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856

block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584

block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0

block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168

block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080

block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080

block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0

block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160

block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808

block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808

block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0

block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808

block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808

block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808

block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0

===============================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

```
layers.Flatten()(base_model.output)
layers.Dense(256, activation='relu')(x)
layers.Dense(128, activation='relu')(x)
layers.Dropout(0.3)(x)
layers.Dense(64, activation='relu')(x)
layers.Dense(2, activation='sigmoid')(x)
```

Model 6

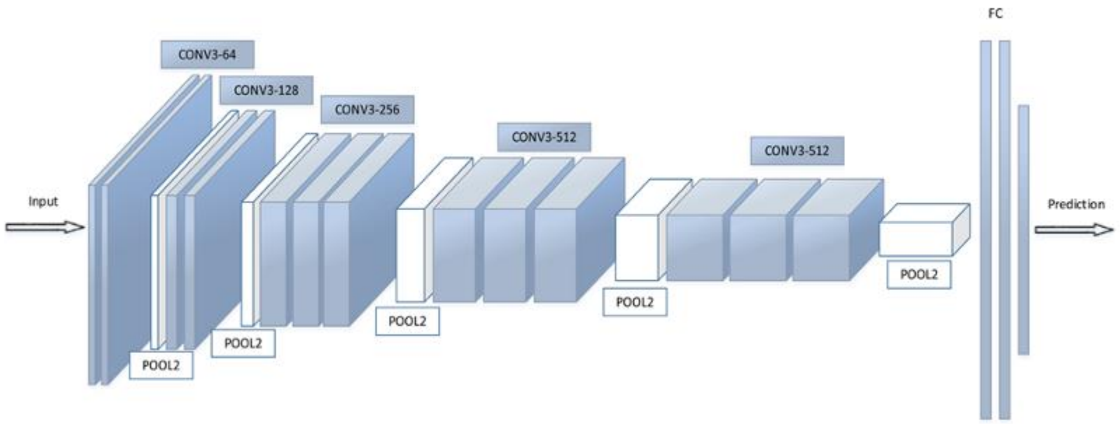
InceptionV3 Transfer Learning with Augmentation and Varying Optimization

TYPE	PATCH / STRIDE SIZE	INPUT SIZE
Conv	3×3/2	299×299×3
Conv	3×3/1	149×149×32
Conv padded	3×3/1	147×147×32
Pool	3×3/2	147×147×64
Conv	3×3/1	73×73×64
Conv	3×3/2	71×71×80
Conv	3×3/1	35×35×192
3 × Inception	Module 1	35×35×288
5 × Inception	Module 2	17×17×768
2 × Inception	Module 3	8×8×1280
Pool	8 × 8	8 × 8 × 2048
Linear	Logits	1 × 1 × 2048
Softmax	Classifier	1 × 1 × 1000

```
layers.Flatten()(base_model.output)
layers.Dense(256, activation='relu')(x)
layers.Dense(128, activation='relu')(x)
layers.Dropout(0.3)(x)
layers.Dense(64, activation='relu')(x)
layers.Dense(2, activation='sigmoid')(x)
```

Transfer Learning: VGG16 and InceptionV3

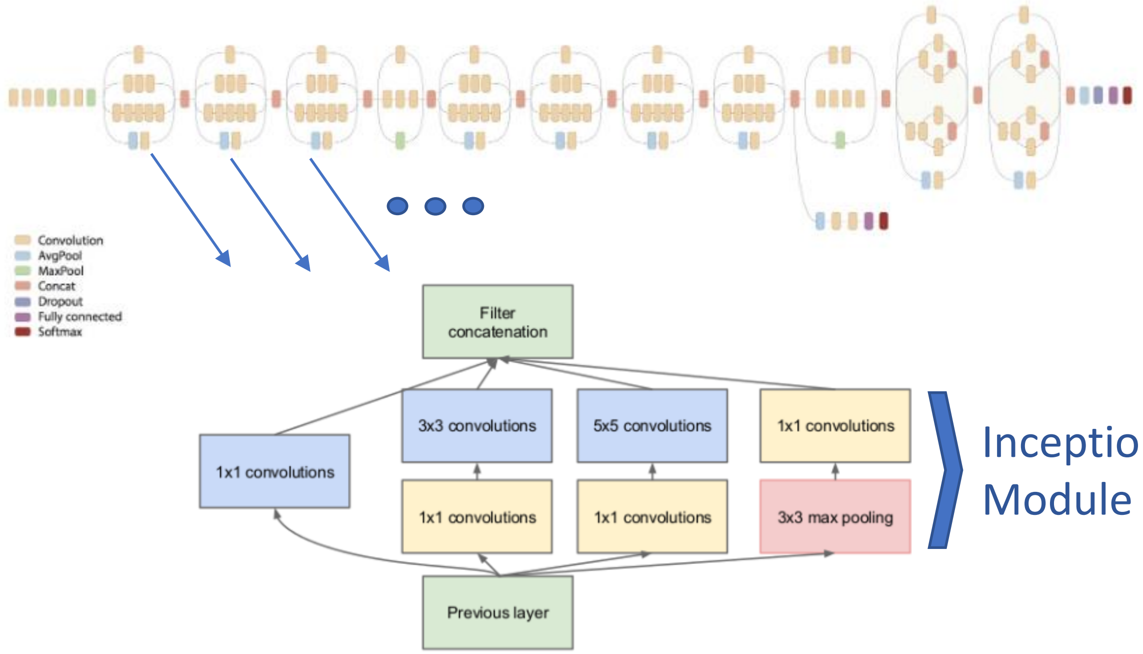
VGG16
16 Layers



Feature Learning	13 Convolutional Layers (3x3 Filter, rely, padding=same)
	Increasing # number of filters (64, 128, 256 and 512)
	MaxPooling2D(2, 2) (after conv 2, 4, 7, 10 and 13)

Classification	Flatten 3 Dense Layers (4096, 4096 and 1000 nodes w/relu, relu, softmax)

InceptionV3
22 Layers

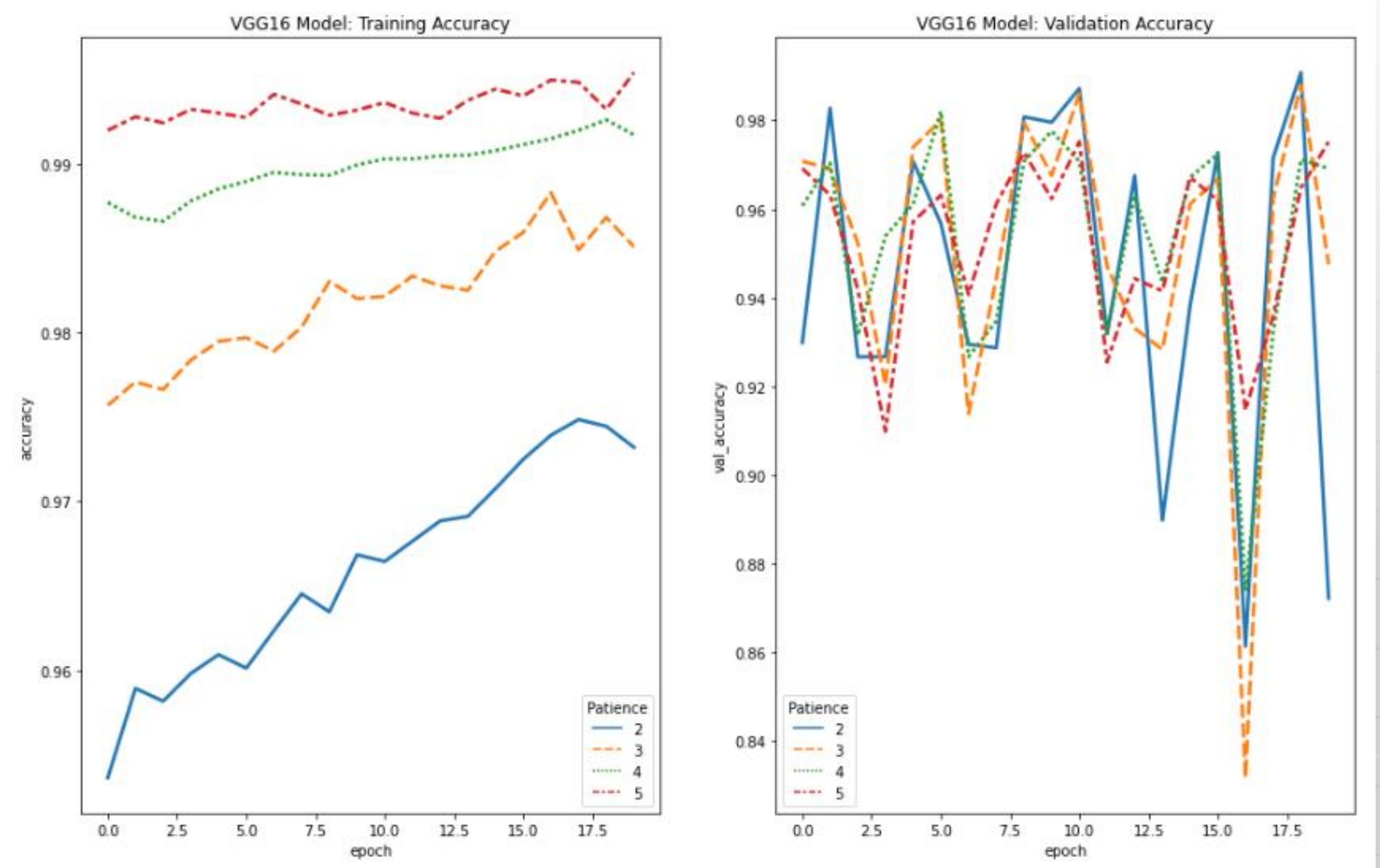


Inception Module

Inception Modules	Multi-level feature extractor Convolutions of different sizes obtain a diversified feature map 1 x 1 convolution blocks perform dimension reduction
Classification	Contains output layer but alson has 2 additional classification outputs which are used to inject gradients at lower layers

Reference:
https://www.kaggle.com/code/shivamb/cnn-architectures-vgg-resnet-inception-tl/notebook

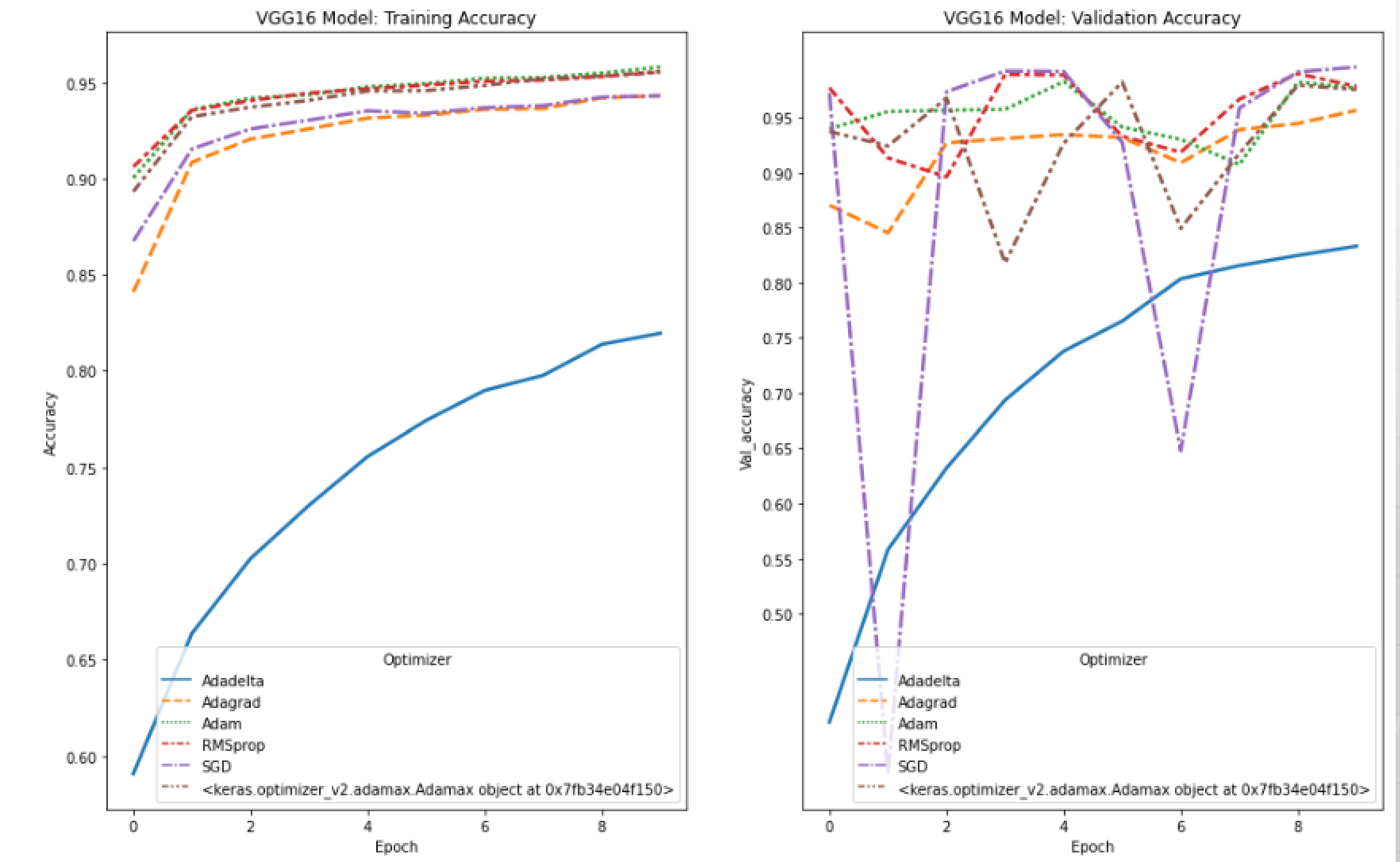
Model 4 Insight: Transfer Learning (VGG16)



Increasing model callback patience…

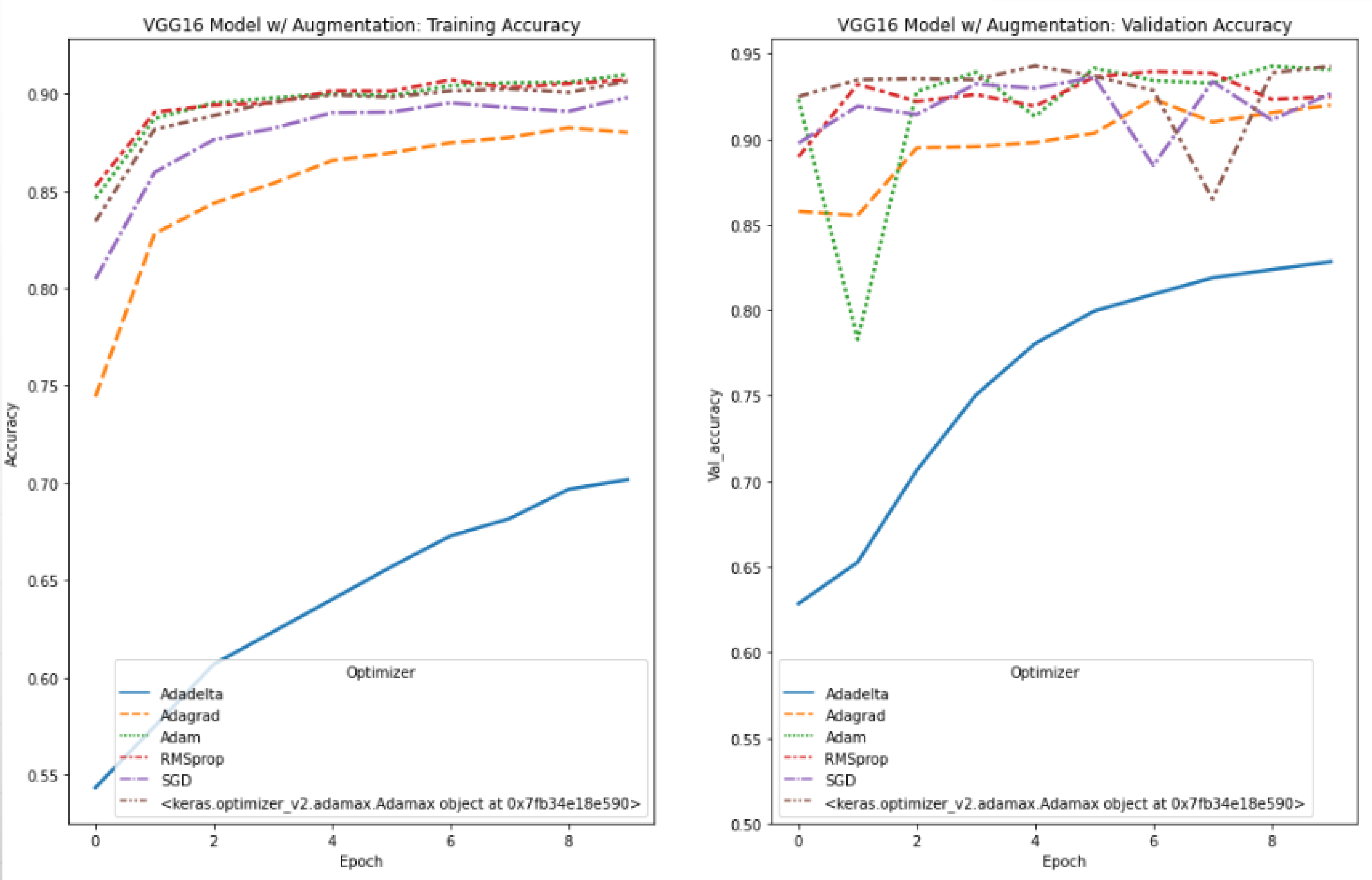
- increased training performance
- did not increase validation accuracy on an already overfit model

Model 4 Insight 2: Transfer Learning (VGG16)



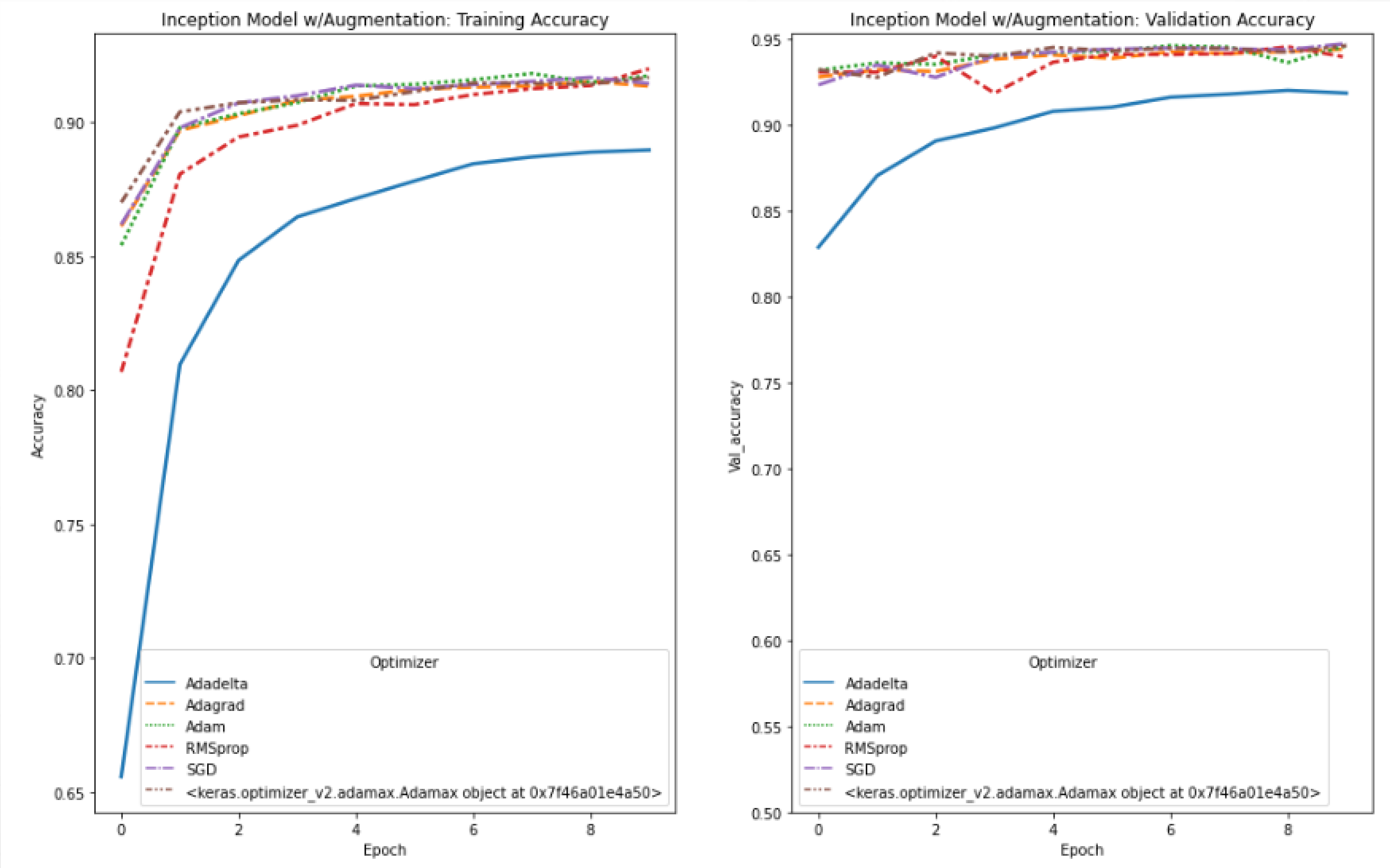
- Demonstrates performance of various optimizers

Model 5 Insight 1: Transfer Learning (VGG16)



- Demonstrates performance of various optimizers
- Addition of augmentation successfully reduced overfitting. (comparing previous slide Model 4 with Model 5 Validation accuracies)

Model 6 Insight: Transfer Learning (InceptionV3)



- Demonstrates performance of various optimizers
- Addition of augmentation with **InceptionV3** model successfully eliminated overfitting.

Referenced Code

- Optimizer Reference: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=An%20optimizer%20is%20a%20function,loss%20and%20improve%20the%20accuracy
- VGG16 Code Reference: https://keras.io/api/applications/vgg/
- InceptionV3: https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/
- KyrasTuner Code Reference: https://keras.io/keras_tuner/