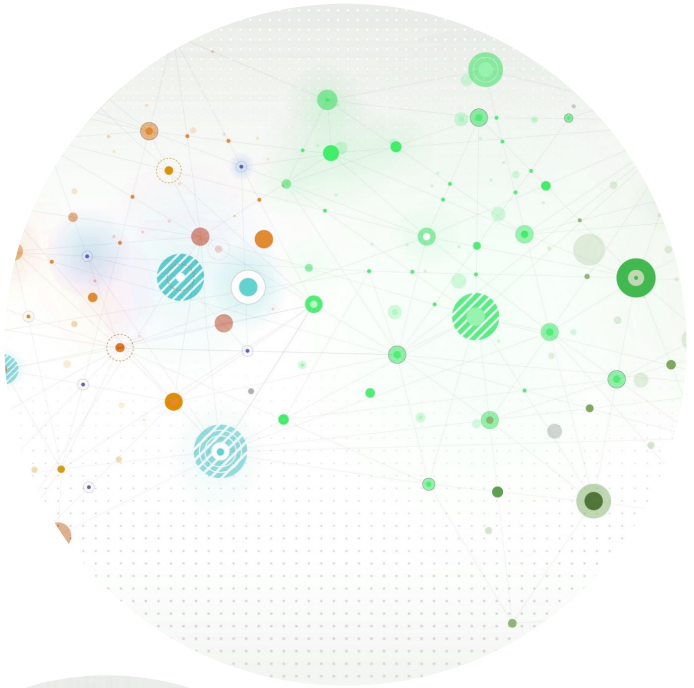
MALARIA DETECTION

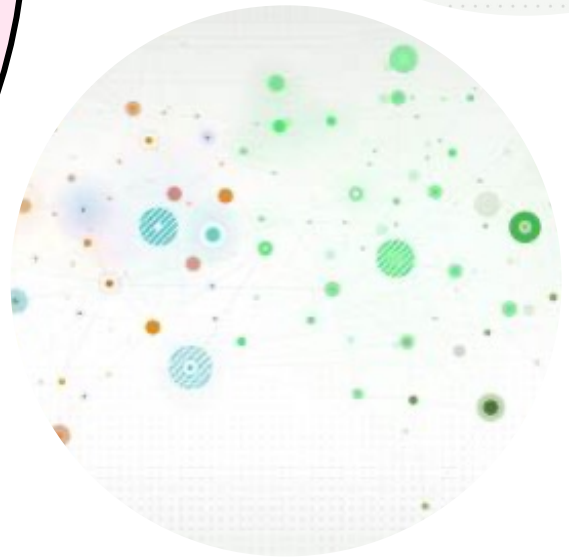
LIVE PRESENTATION

A MACHINE LEARNING APPROACH

BY

PERI S. PERIYASAMY





Executive Summary

- The goal is to find a *Computer Vision* based *Machine Learning* model to identify malaria infected cells quickly without human inspection
- We developed 7 different models, applying various machine learning techniques that are proven in the industry
- The model that used the original cell images dataset along with simulated (generated) images from them to cover unseen variations, has outperformed other models we built, and seems to be the best choice
- There is a potential to improve even this model further *(details captured in the slide for next steps)*, but that will take additional time and resources.
- Time to market is of essence here. Considering the dire situation to identify malaria infected patients early on, and in order to protect them and reduce the number of deaths due to malaria around the world, sooner we deploy a good solution the better it would be
- Given these, our recommendation is to:
    - Deploy our final model now
    - Continue to invest in fine-tuning the model
    - Rollout the updated model when we have significant improvement

Overview of the Problem

- Malaria is a contagious disease caused by **Plasmodium** parasites that are transmitted to humans through mosquitoes
- The lethal parasites can stay alive for more than a year in a human body without showing any symptoms
- Almost 50% of the world's population is in danger from Malaria.
- There were 229 million malaria cases and 400,000 malaria related deaths reported throughout the world in 2019 alone
- Manual inspection of red blood cells by experienced professionals to discriminate between healthy and infected cells is a tedious and time-consuming process. Further the accuracy of such diagnostics can be adversely impacted by inter-observer variability

Approach for the Solution

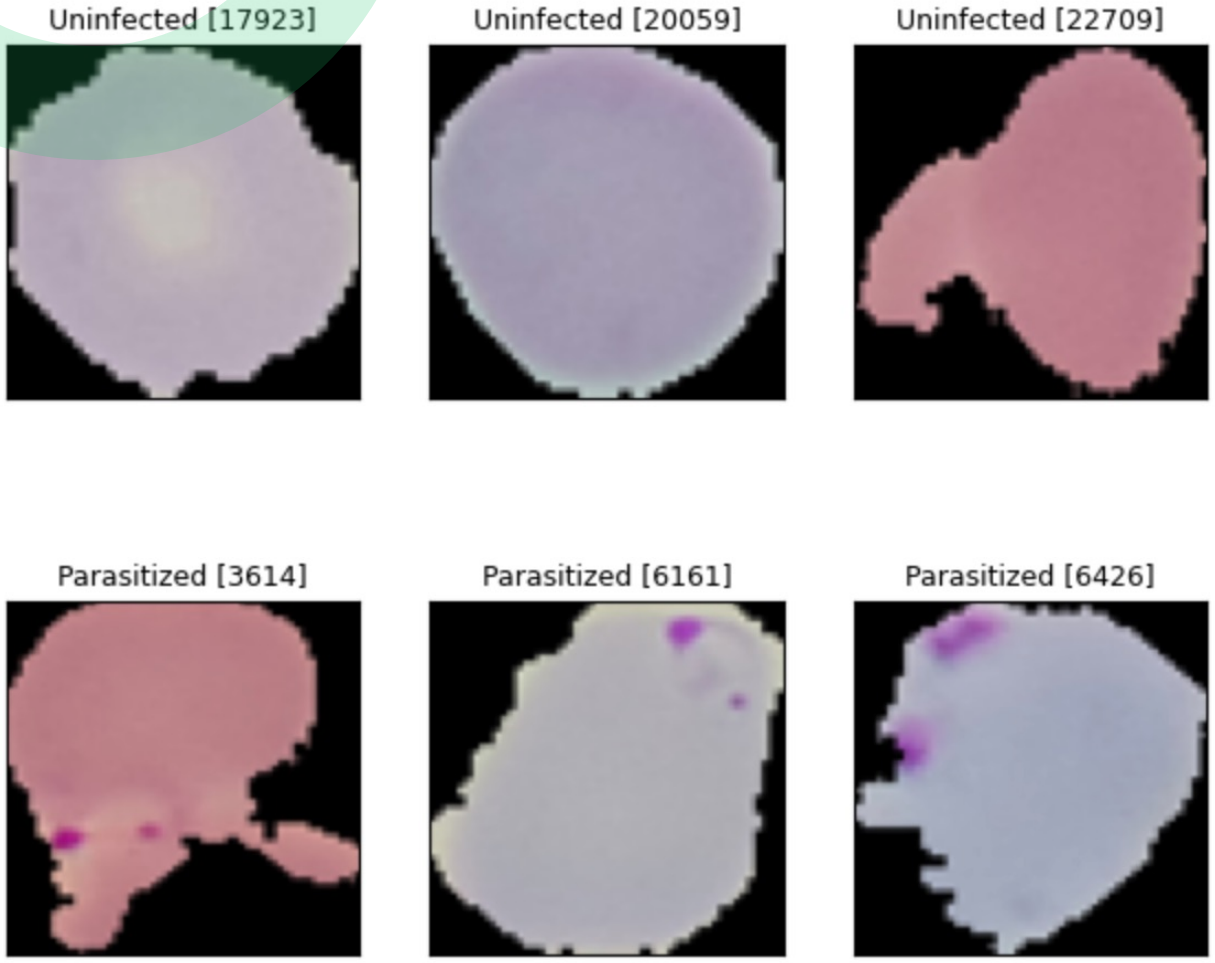
Two sets of color images of red cells, with one set containing *parasitized* cell images and the other set containing *uninfected* cell images are available

Use this collection of cell images to develop a *Computer Vision* based *Machine Learning* model that can speed up the identification of malaria infection of a given cell image without relying on human inspection

This model will help to identify malaria infected people during their early infection phase, so that they can be treated timely to avoid any hospitalizations or even death

Build multiple Machine Learning models using various techniques, compare their effectiveness and identify the final model that we can start using in the field ASAP – Time is of essence!

Key Findings & Insights



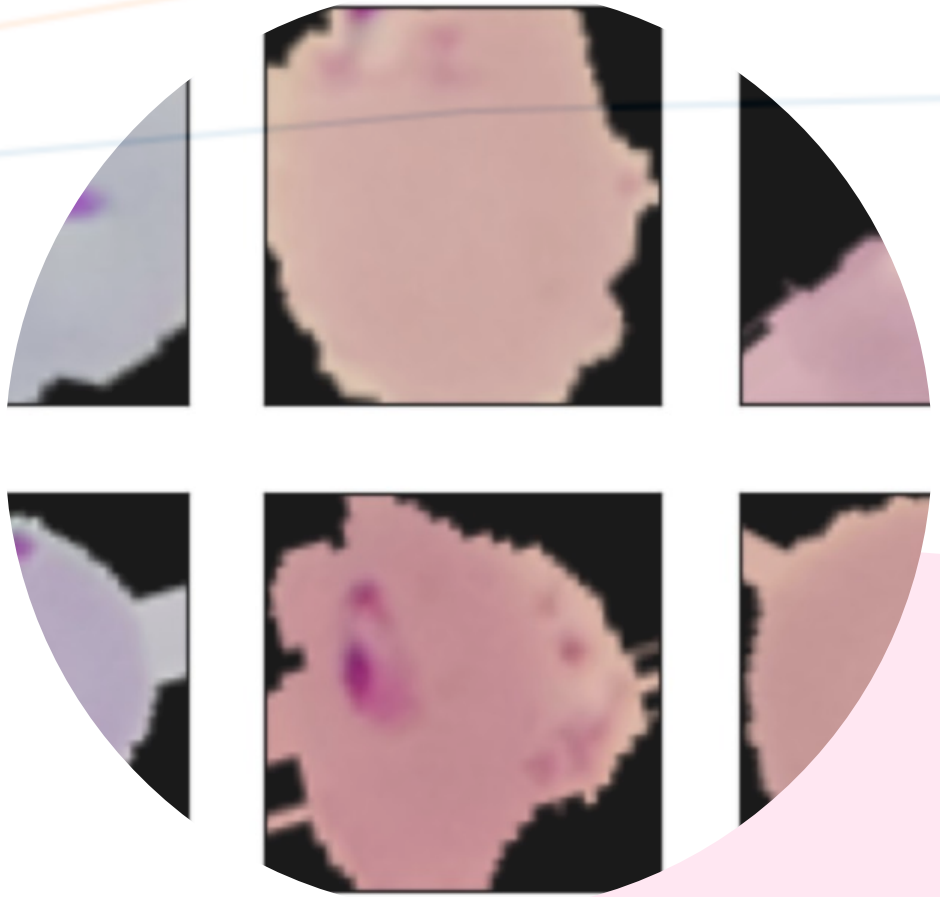
Cell images collection we have is well balanced between *parasitized* and *uninfected* samples – it helps our models to be very accurate

The differentiating patterns in parasitized images seem simple enough, containing spots of various shapes with different color and intensities. Computer Vision based Machine Learning models are good in recognizing them easily

Created multiple models and evaluated their merits using various sets of pre-processed images (for e.g., blurring of images, converting them to different color space, use of augmented images etc.)

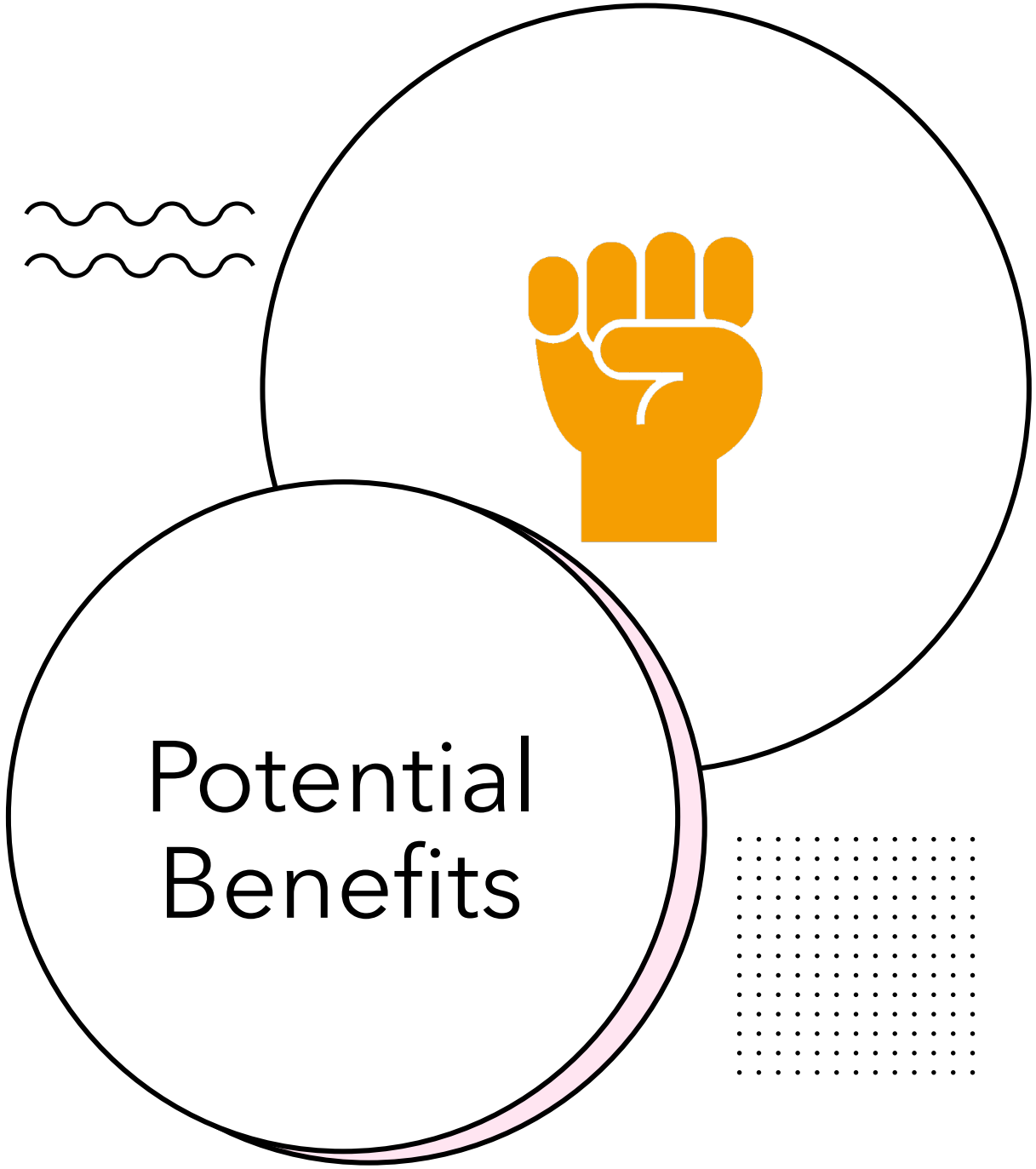
Proposed Final Model Solution

- All of our models performed reasonably well, but with various levels of accuracy
- Our Final Model makes use of augmented images that simulate variations of actual samples and has the best performance with an accuracy of 99%, when tested against unseen samples
- Still, there is scope for further improvement, as detailed in the next steps



Recommendations & Next Steps

- Deploy our Final Model to the world geographies ASAP as it can help to prevent hospitalization or even death of infected people
- Continue this study further to fine-tune our final model or to come up with an even better performing model
- Invest in collecting additional samples on an on-going basis from different parts of the world to enrich the samples dataset, and to re-train our model on the new samples that would make the model more efficient
- On a periodic basis, plan for rolling out of updated models whenever we have attained models with significant performance improvements
- The cost of developing and deploying these models are very insignificant when compared to the potential benefit of saving hundreds of thousands of human lives



Potential Benefits

- Saving a single human life itself is a great achievement, but using our final model in the field, we could potentially save hundreds of thousands of malaria infected people
- Use of our model would reduce the load and reliance on human specialists, and relieving them to use their expertise to solve other demanding problems
- With continued and prolonged use of the model, we can even dream about the possibility of completely eradicating malaria from this world!

Risks & Challenges

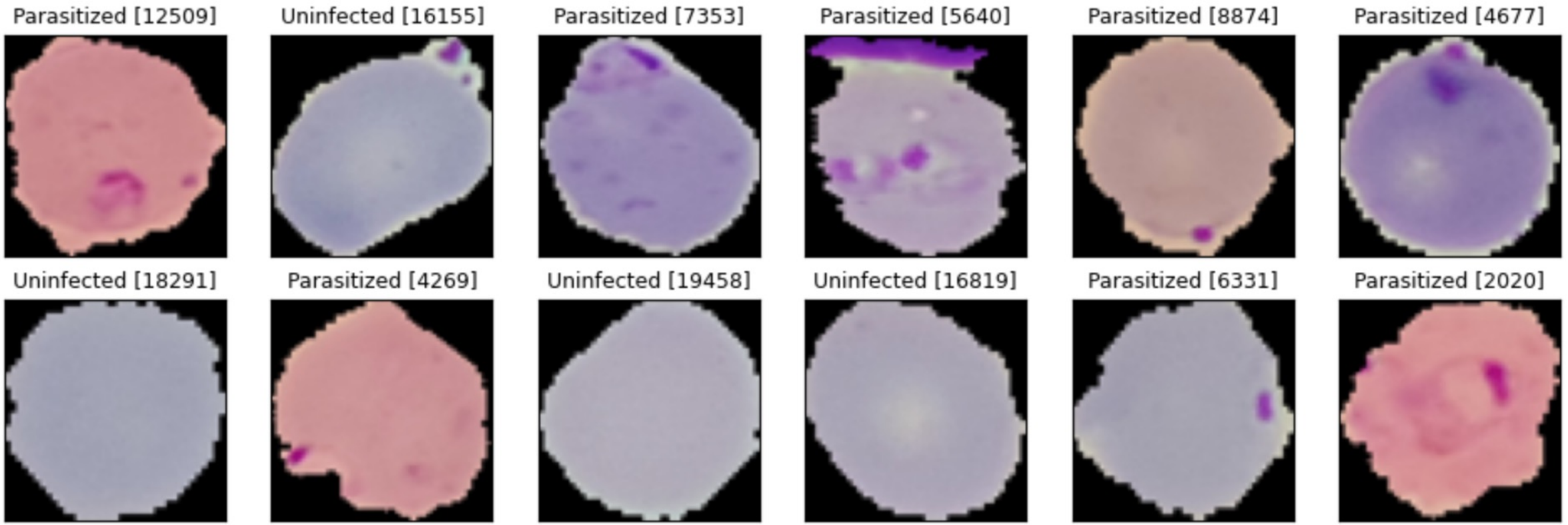
- A possible risk is that the model mis-diagnosing an infected person as non-infected, which has the potential of losing a human life. Fortunately, our final model's performance metrics are very high that the benefits out-weigh this insignificant risk
- We need to keep re-calibrating the model so that they are not outdated and can keep learning any variations that may be developing

Thank you!

Appendix

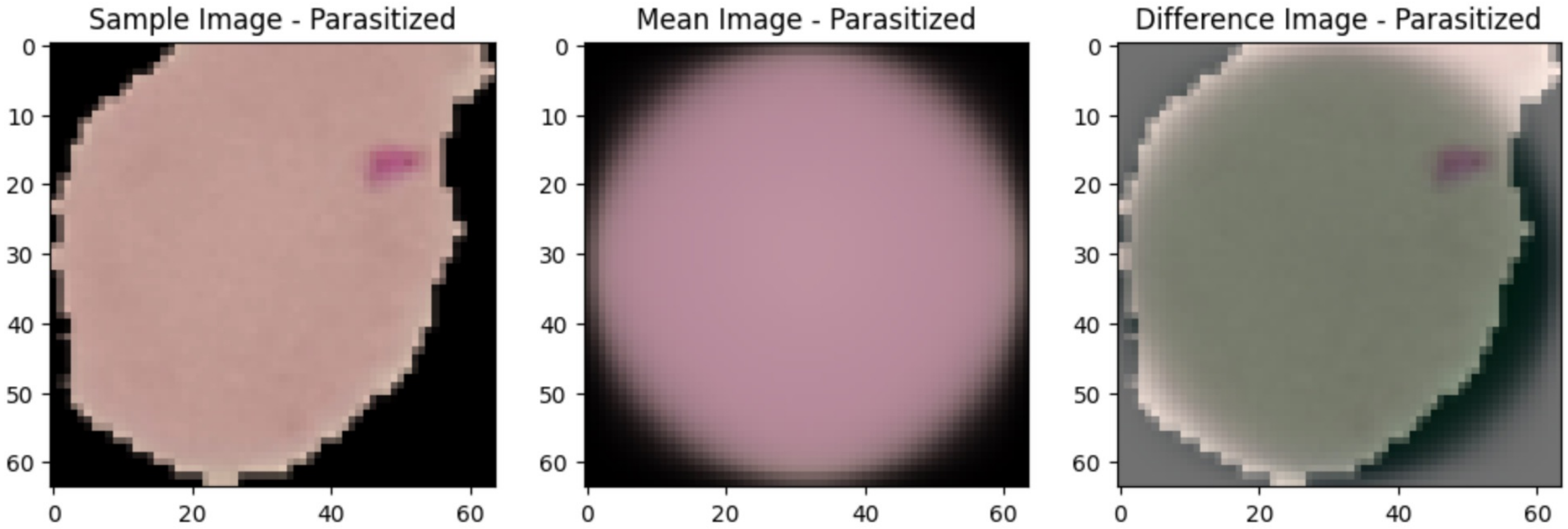
Images from training samples

```
# Display a random 36 images in a 6x6 plot as requested from the training dataset along with their labels

displayImages(X_train_normalized, 6, 6, (12, 12), True)
```



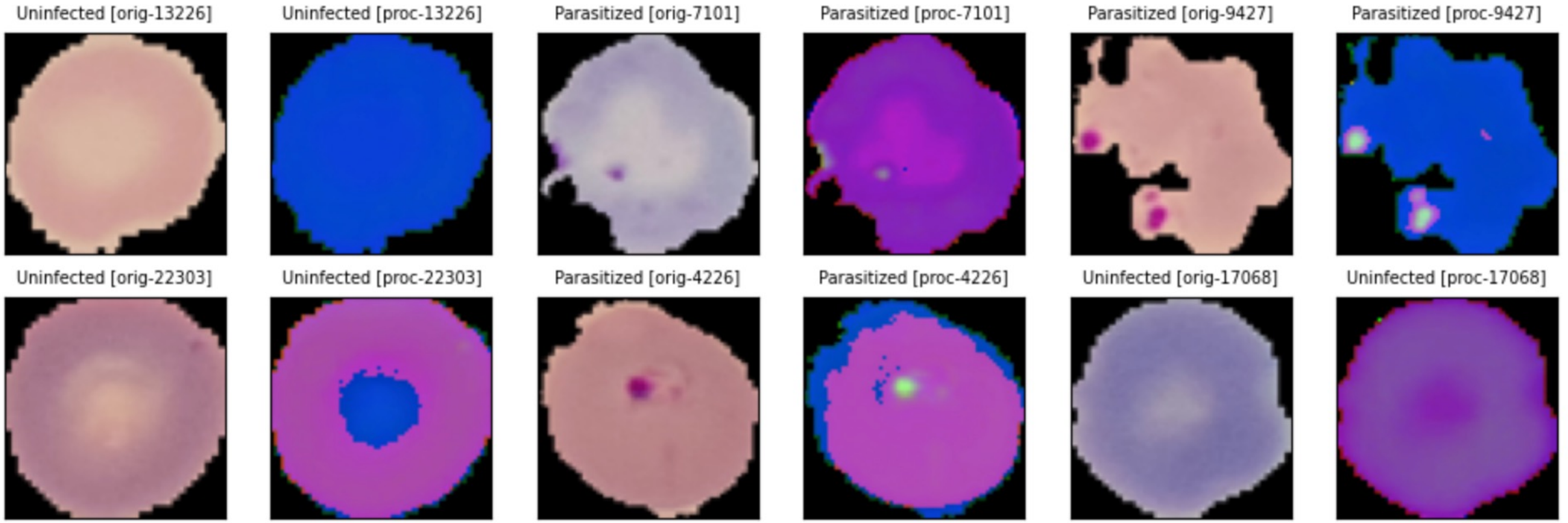
Original, Mean and Difference Images

```
# Display a random parasitized sample image and its difference image from its mean

img_to_display = np.random.randint(0, X_train_parasitized.shape[0])            # choose a random sample
DisplayMeanAndDifferenceImages(X_train_parasitized[img_to_display], mean_parasitized_image, 'Parasitized')
```



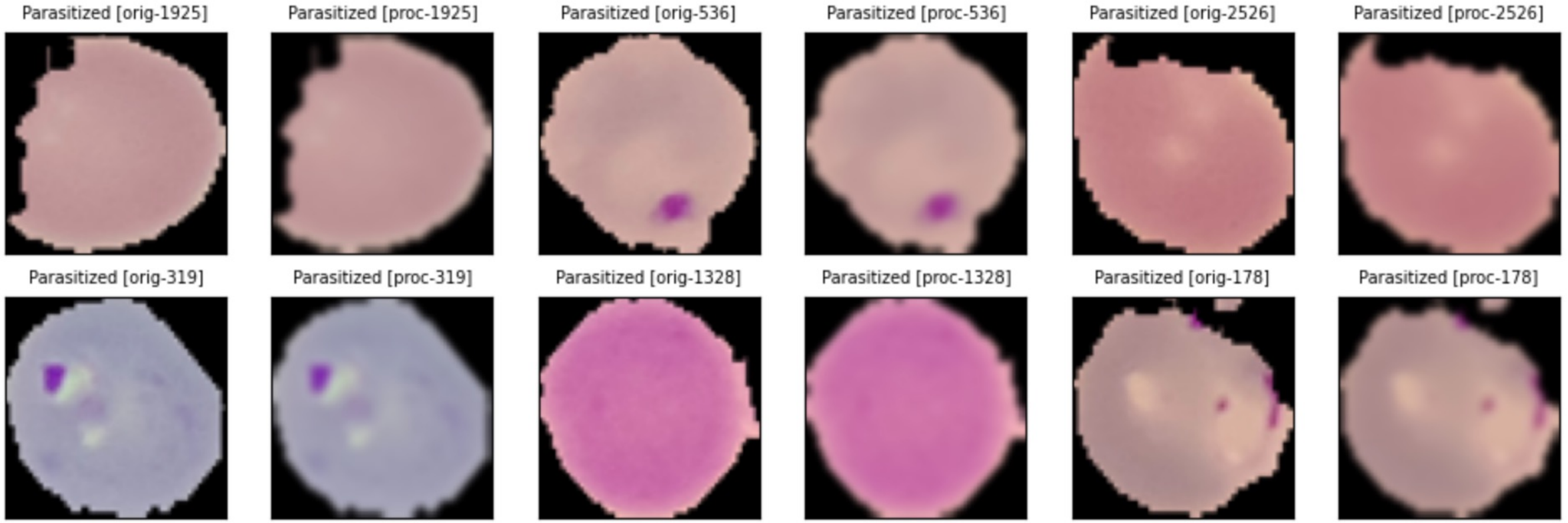
RGB Images and their HSV versions

```
# Compare a random set of RGB and their corresponging HSV images from the training dataset

displayOriginalAndProcessedImages(X_train_normalized, X_train_normalized_hsv)
```



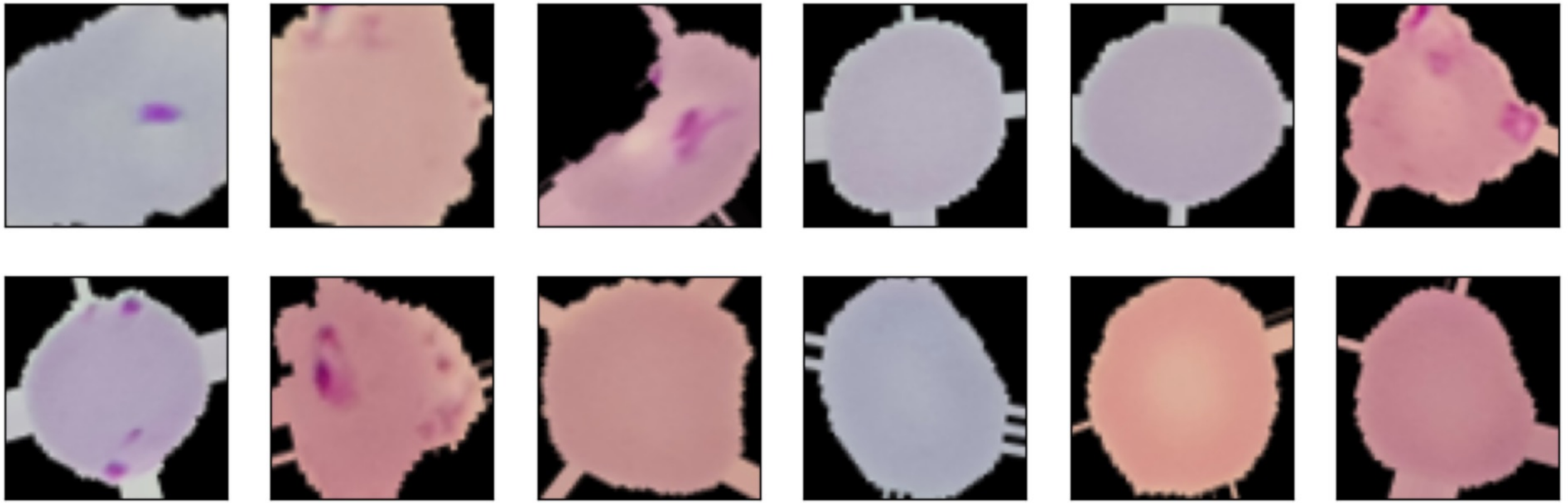
Gaussian Blurred Images

```
# Compare a random set of RGB and their corresponging Gaussain blurred images from the test dataset, visually

displayOriginalAndProcessedImages(X_test_normalized, X_test_normalized_blurred)
```

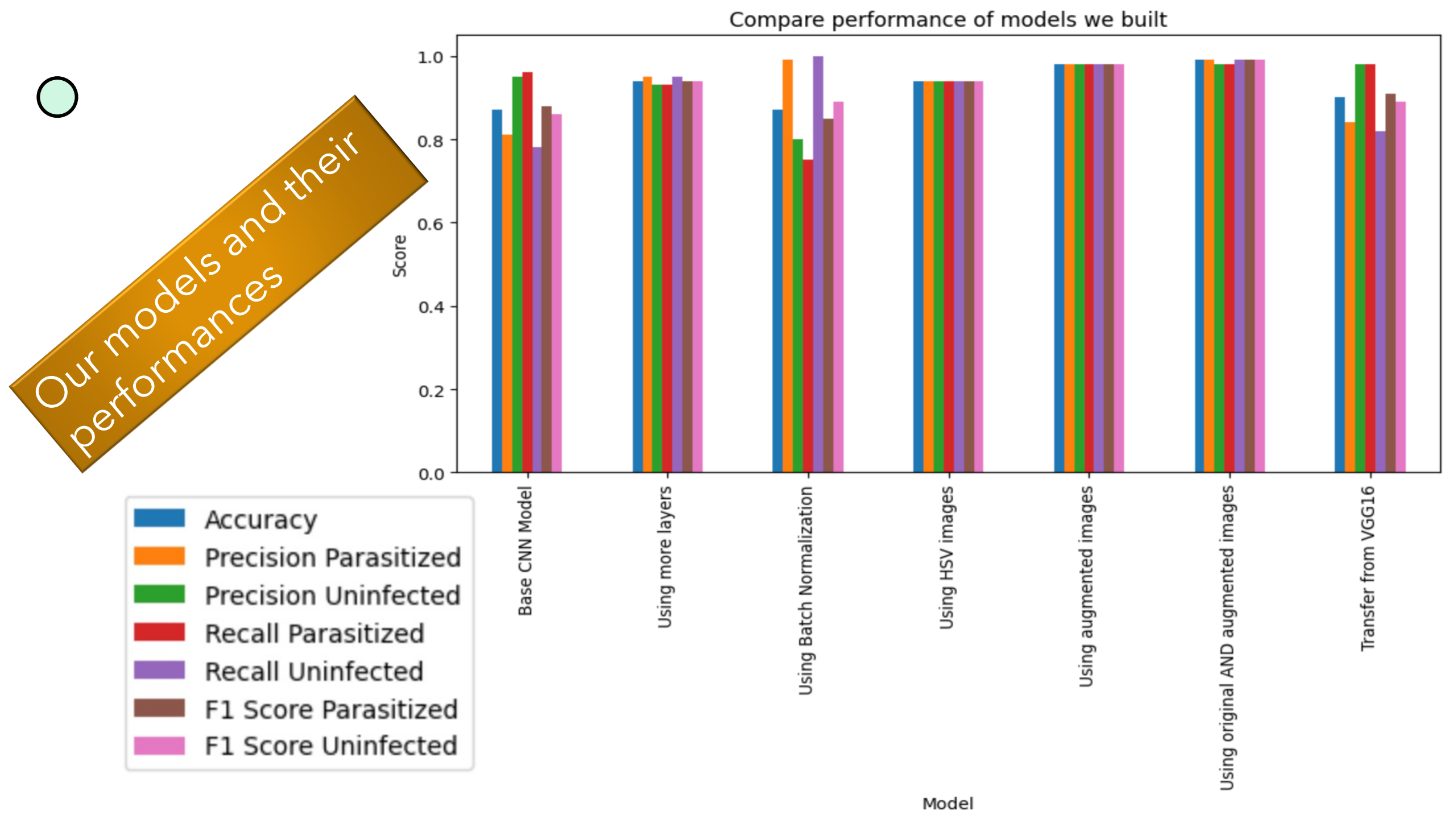


Augmented Images

```
# Create the image data generator using Keras

from keras.preprocessing.image import ImageDataGenerator

data_generator = ImageDataGenerator(rotation_range=45,        # rotate images randomly upto 45 degrees
                                    zoom_range=0.2,           # zoom the image upto 20%
                                    horizontal_flip=True,     # flip the image horizontally
                                    vertical_flip=True,       #       and verticaly, randomly
                                    fill_mode='nearest')
```

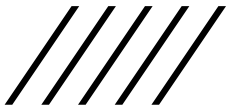


Our models and their performances



Models' performance metrics

Model	Accuracy	Precision Parasitized	Precision Uninfected	Recall Parasitized	Recall Uninfected	F1 Score Parasitized	F1 Score Uninfected
Base CNN Model	0.87	0.81	0.95	0.96	0.78	0.88	0.86
Using more layers	0.94	0.95	0.93	0.93	0.95	0.94	0.94
Using Batch Normalization	0.87	0.99	0.80	0.75	1.00	0.85	0.89
Using HSV images	0.94	0.94	0.94	0.94	0.94	0.94	0.94
Using augmented images	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Using original AND augmented images	0.99	0.99	0.98	0.98	0.99	0.99	0.99
Transfer from VGG16	0.90	0.84	0.98	0.98	0.82	0.91	0.89



Using Original & Augmented Images

Use combined image data generators

```
# Let's use both original dataset along with the augmented images generated

# we already have a ImageDataGenerator for augmented images, we will re-use it here

# Create a genearotr for original images
original_generator = ImageDataGenerator()

# collect flow'ed images from both original and augmented/generated images
X_train_original = original_generator.flow(X_train_normalized, y_train_encoded, batch_size=32)
X_train_augmented = data_generator.flow(X_train_normalized, y_train_encoded, batch_size=32)

# concatenate the generaotrs
def CombineGenerators(gen1, gen2):
    while True:
        x1, y1 = gen1.next()
        x2, y2 = gen2.next()
        yield (np.concatenate((x1, x2), axis=0), np.concatenate((y1, y2), axis=0))

X_train_combined = CombineGenerators(X_train_original, X_train_augmented)
```

Architecture of our Final Model

Building the Model

```
# We can use the same architecture as the last model we built to use with augmented training images

model_orig_aug = Sequential()
model_orig_aug.add(Conv2D(16, (3, 3), padding="same", input_shape=(64, 64, 3)))
model_orig_aug.add(LeakyReLU(alpha=0.1))
model_orig_aug.add(Conv2D(32, (3, 3), padding="same"))
model_orig_aug.add(LeakyReLU(alpha=0.1))
model_orig_aug.add(MaxPooling2D(pool_size=(2, 2)))
model_orig_aug.add(BatchNormalization())
model_orig_aug.add(Conv2D(32, (3, 3), padding="same"))
model_orig_aug.add(LeakyReLU(alpha=0.1))
model_orig_aug.add(Conv2D(64, (3, 3), padding="same"))
model_orig_aug.add(LeakyReLU(alpha=0.1))
model_orig_aug.add(MaxPooling2D(pool_size=(2, 2)))
model_orig_aug.add(BatchNormalization())
model_orig_aug.add(Flatten())
model_orig_aug.add(Dense(32))
model_orig_aug.add(LeakyReLU(alpha=0.1))
model_orig_aug.add(Dropout(0.5))
model_orig_aug.add(Dense(2, activation='softmax'))

model_orig_aug.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])
```

Training of our Final Model

Fit and Train the model

```
# Train the model using prepared iterators that use training and test images

model_history = model_orig_aug.fit(X_train_combined,
   steps_per_epoch=len(X_train_normalized) * 2 // 32,
   epochs=50, callbacks=[early_stopping_cb],
   validation_data = val_iterator,
   validation_steps=X_test_normalized.shape[0] // 32
   )
```

```
Epoch 1/50
1559/1559 [==============================] - 59s 36ms/step - loss: 0.2557 - accuracy: 0.9064 - val_loss: 0.0605 - val_accuracy: 0.9865
Epoch 2/50
1559/1559 [==============================] - 55s 35ms/step - loss: 0.0902 - accuracy: 0.9714 - val_loss: 0.0673 - val_accuracy: 0.9807
Epoch 3/50
1559/1559 [==============================] - 55s 35ms/step - loss: 0.0797 - accuracy: 0.9746 - val_loss: 0.0467 - val_accuracy: 0.9869
Epoch 4/50
1559/1559 [==============================] - ETA: 0s - loss: 0.0755 - accuracy: 0.9756Restoring model weights from the end of the best epoch: 1.
1559/1559 [==============================] - 55s 35ms/step - loss: 0.0755 - accuracy: 0.9756 - val_loss: 0.0467 - val_accuracy: 0.9846
Epoch 4: early stopping
```