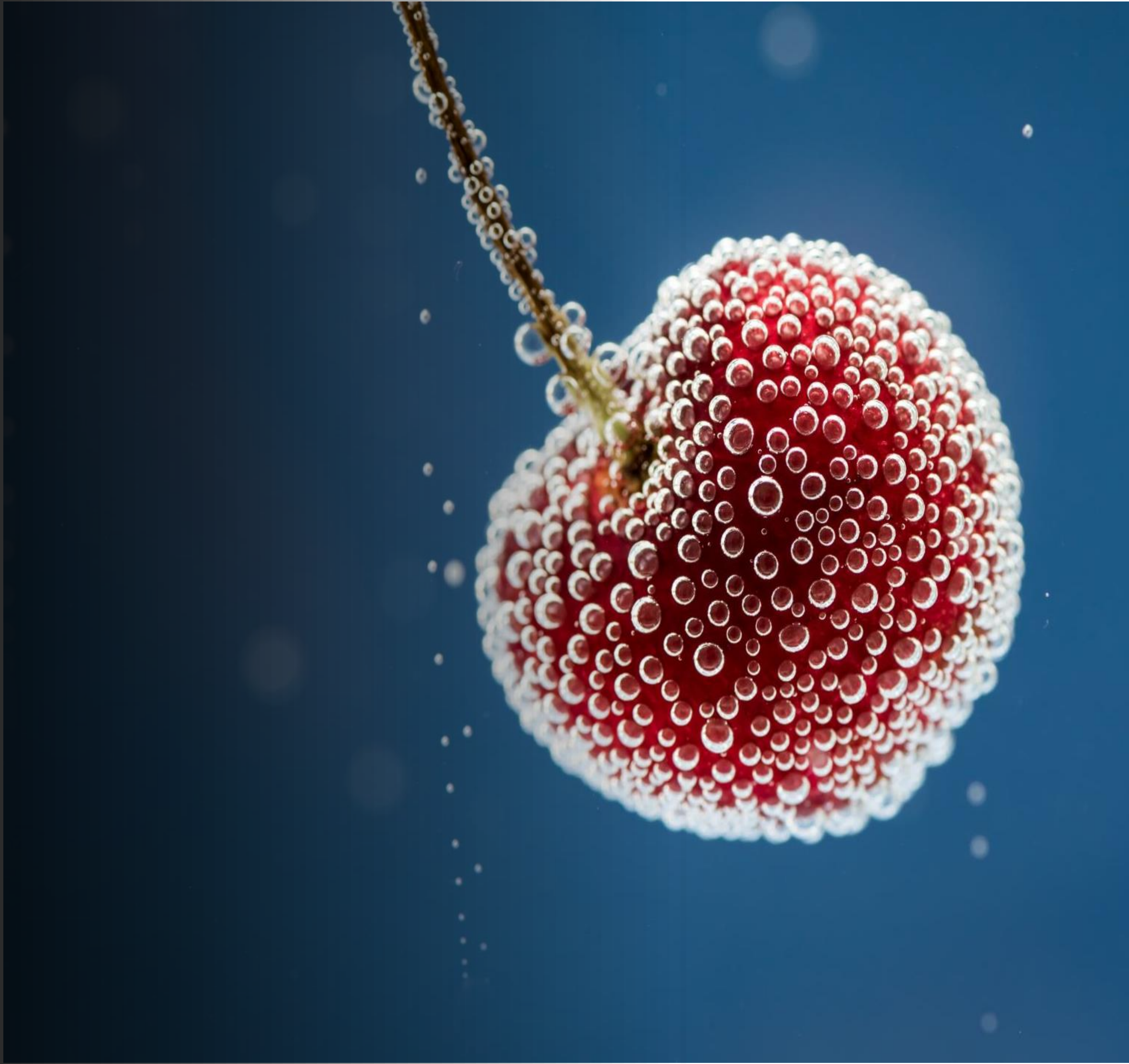
Fruit Recognition with Deep Learning

Afia Aziz kona

University of North Carolina at Charlotte



Project Overview

Objective: To design a deep learning model that can classify images of fruits as either fresh or rotten

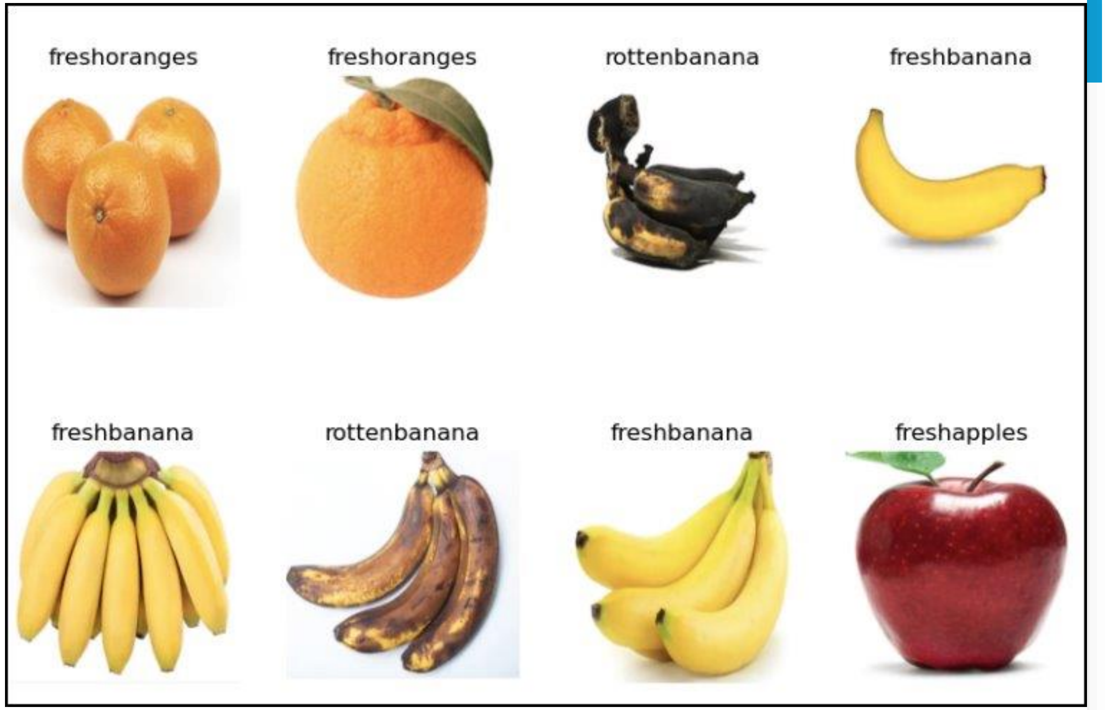
Significance: Helps in quality control in food processing and retail industries by automating the sorting of fruits, thus reducing waste and ensuring quality.

Outcome: A CNN model trained to identify five different categories of fruit conditions with high accuracy.

Dataset Composition

- **Volume:** 1212 images, split into 5 classes.
- **Class Labels:** Fresh Apples, Fresh Bananas, Fresh Oranges, Rotten Apples, and Rotten Banana.

Visualize some of the images from our dataset



CNN Model Architecture - Overview

- ***Design Philosophy:***

Layered approach to extract features and reduce dimensionality while preserving spatial hierarchy.

- ***Key Components:***

➢ Convolutional layers for feature detection.

➢ Pooling layers for downsampling.

➢ Dropout layers for regularization to combat overfitting.

Model Architecture - In Depth

- **Input Layer:**

300×300×3 corresponding to the height, width, and RGB channels.

- **Convolutional Layers:**

➢ 1st Conv Block: 32 filters, 3×3 kernel, ReLU activation, followed by batch normalization and 2×2 max pooling.

➢ 2nd Conv Block: 64 filters, increased depth for complex patterns.

➢ 3rd and 4th Conv Blocks: 128 and 256 filters respectively, increasing the model's capacity to learn finer details.

- **Flattening Layer:**

Converts the 2D matrix data to a vector for the dense layer.

- **Dense Layers:**

128 neurons for high-level reasoning.

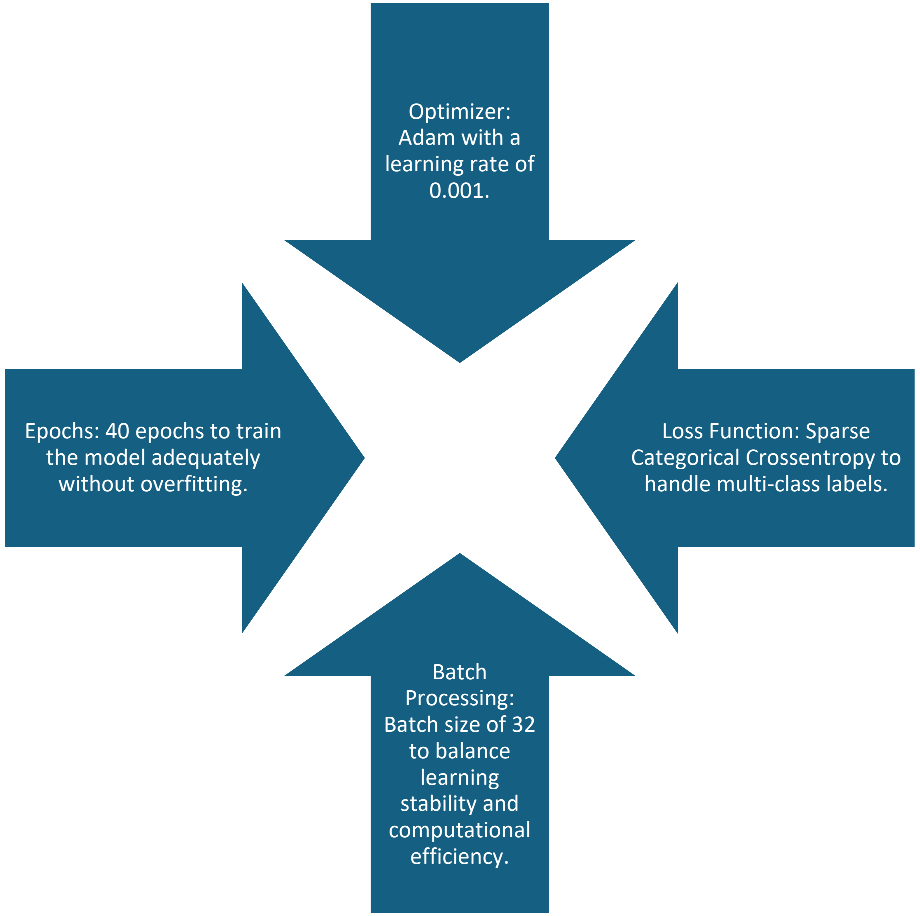
- **Output Layer:**

5 neurons with softmax activation for classification.

Model Architecture

```
model = models.Sequential([
    # First, the preprocessing layers
    layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
    layers.experimental.preprocessing.Rescaling(1./255),
    # First convolution block
    layers.Conv2D(32, (3, 3), padding='same', activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.2),
    # Second convolution block
    layers.Conv2D(64, (3, 3), padding='same', activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.3),
    # Third convolution block
    layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.4),
    # Fourth convolution block
    layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.5),
    # Flattening the output to feed into a Dense layer
    layers.Flatten(),
    # Dense Layer
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    # Output Layer
    layers.Dense(n_classes, activation='softmax'),
])
```

Training Strategy



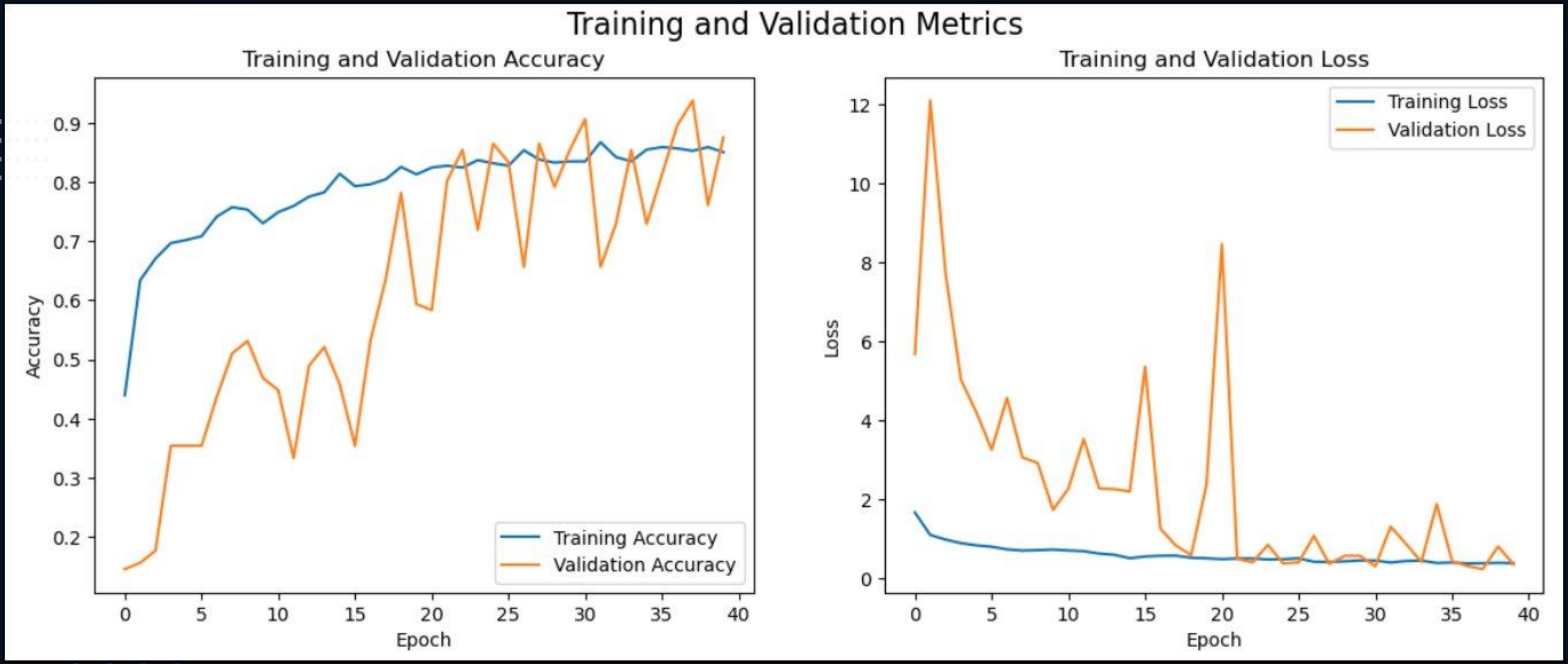
Model Training and Validation

- Observing our test dataset, we find that our accuracy is 81.00%. It's thought that this precision is really good.

```
scores = model.evaluate(test_ds)
```

```
5/5 [==============================] - 9s 1s/step - loss: 0.4900 - accuracy: 0.8188
```

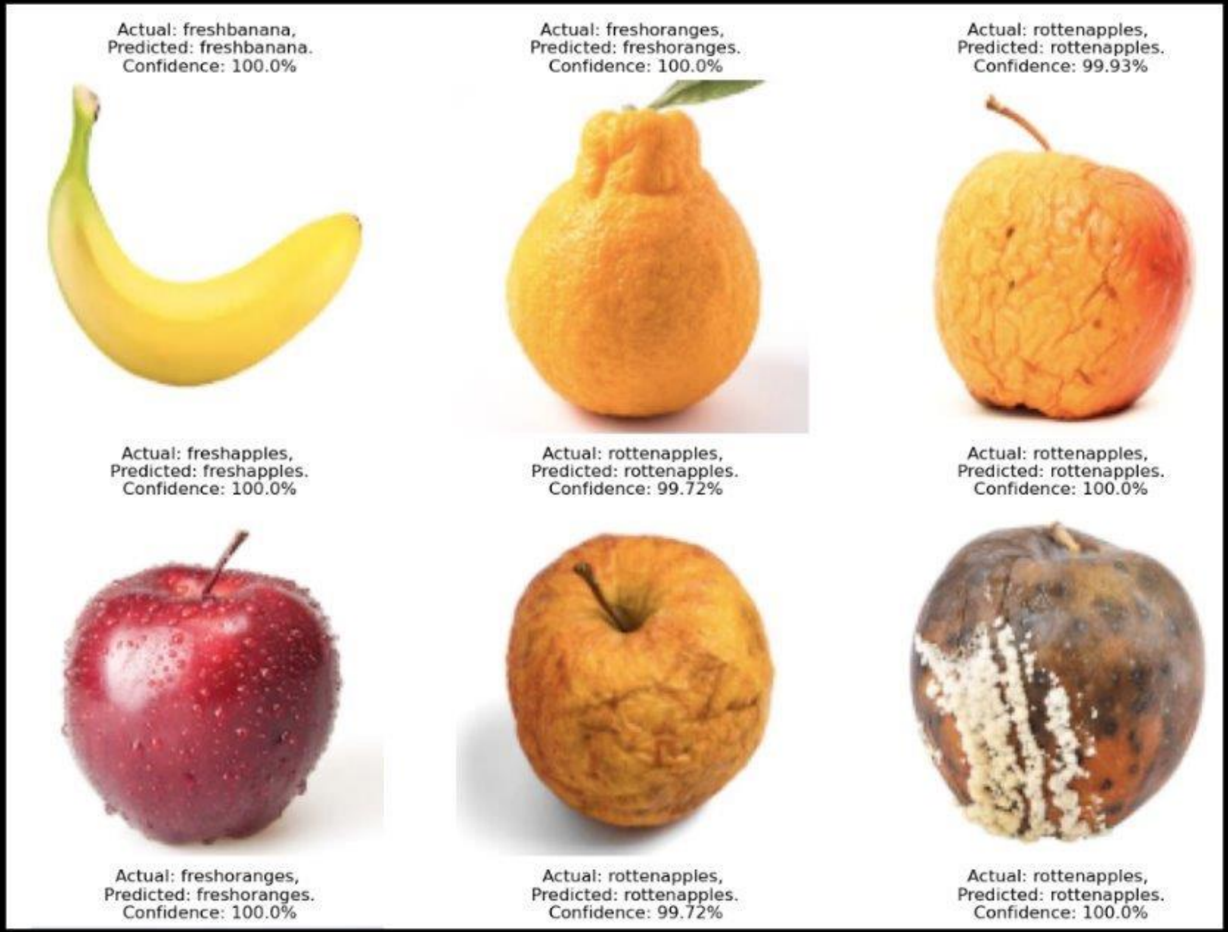
Evaluation and Testing



Sample Predictions

```
def predict(model, img):

    # Convert the image to a float32 tensor, normalize it, and resize
    img = tf.convert_to_tensor(img, dtype=tf.float32)
    img = img / 255.0
    img = tf.image.resize(img, [IMAGE_SIZE, IMAGE_SIZE])
    img = tf.expand_dims(img, axis=0)  # Add a batch dimension

    # Predict the class of the image
    predictions = model.predict(img)
    predicted_class_index = np.argmax(predictions[0])
    predicted_class = class_names[predicted_class_index]
    confidence = round(np.max(predictions[0]) * 100, 2)

    return predicted_class, confidence

# Visualization and running inference on a few sample images
plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        # Correctly pass the image to the prediction function
        predicted_class, confidence = predict(model, images[i].numpy(
        actual_class = class_names[labels[i].numpy()]  # Ensure to ge

        plt.title(f"Actual: {actual_class},\nPredicted: {predicted_cl
        plt.axis("off")
plt.show()
```

Predictions



Conclusion

- **Model Learning:** The model exhibits strong learning capabilities, with a steady increase in training accuracy.
- **Generalization Gap:** A noted difference between training and validation accuracy suggests a need for better model generalization.
- **Validation Volatility:** Validation loss volatility points to model sensitivity, requiring further optimization
- Here are some strategies to consider: Data Augmentation, Model Architecture, Hyperparameter Tuning,Advanced Optimization Techniques, and Batch Normalization.

Thank You