Machine Learning with PyTorch

By: Justin (2109774), Kelvin (2223841), Xiuping (2223742)

Fruits

Apples

- Kelvin

Watermelon

- Xiuping

and **Unknown**

Banana

- Justin

Data Collection

- We used images of the fruits we already have at home, as well as images taken at a supermarket
- We used the “burst” mode to rapidly take images of the fruits at different angles, lighting and backgrounds
- Total images:
  - Apple: 3049
  - Watermelon: 1349
  - Banana: 2305

Sample Images (raw)

Apples

Watermelons

Bananas

Data Preparation

- Before training, we have to prepare our dataset by standardizing all the images by resizing them
- To speed up the training we also reduced the pixel count while resizing to reduce their file size
- We also split and shuffled the images into Train, Valid and Test folders to ensure separate data for training and evaluating the model as well as consistency
- We used 2 simple Python scripts to automate these tasks
- This is a one-time processing of the images to reduce their size

```python
# Calculate the new height to maintain the aspect ratio
width_percent = (new_width / float(img.size[0]))
new_height = int((float(img.size[1]) * float(width_percent)))

# Resize the image
img = img.resize((512, 512), Image.Resampling.LANCZOS)

# Save the resized image, overwriting the original file
img.save(img_path)
```

Resize images

```python
    # Get a list of all files in the input folder
    all_files = [f for f in os.listdir(input_folder) if os.path.isfile(os.path.join(input_folder, f))]

    # Shuffle the files randomly
    random.shuffle(all_files)

    # Calculate the number of files for each split
    total_files = len(all_files)
    train_count = int(total_files * split_ratio[0])
    val_count = int(total_files * split_ratio[1])
    test_count = total_files - train_count - val_count

    # Split the files into the respective folders
    train_files = all_files[:train_count]
    val_files = all_files[train_count:train_count + val_count]
    test_files = all_files[train_count + val_count:]

    # Copy the files to the respective folders if they don't already exist there
    for f in train_files:
        if not os.path.exists(os.path.join(output_folders[0], f)):
            shutil.copy(os.path.join(input_folder, f), os.path.join(output_folders[0], f))
    for f in val_files:
        if not os.path.exists(os.path.join(output_folders[1], f)):
            shutil.copy(os.path.join(input_folder, f), os.path.join(output_folders[1], f))
    for f in test_files:
        if not os.path.exists(os.path.join(output_folders[2], f)):
            shutil.copy(os.path.join(input_folder, f), os.path.join(output_folders[2], f))

# Define the input and output folders and the split ratio
input_base_folder = 'sorting'
output_base_folder = 'dataset'

fruits = ['apple', 'watermelon', 'banana']
split_ratio = [0.70, 0.15, 0.15]
```

Shuffle images in the folder + split into Train/Valid/Test 70/15/15%

Sample Images (processed)

Apples

Watermelons

Bananas

Image Pre-processing

- Generate Image Data for training with different augmented images for each epoch, as well as Image Data for Validation and Test

```python
# Define dataset class
class CustomDataset(Dataset):
    def __init__(self, data_dir, labels, transform=None):
        self.data = []
        self.labels = []
        self.transform = transform
        self.img_size = 100
        self.labels_map = labels
        for label in labels:
            path = os.path.join(data_dir, label)
            class_num = labels.index(label)
            for img in os.listdir(path):
                try:
                    img_arr = cv2.imread(os.path.join(path, img))[..., ::-1]  # Convert BGR to RGB format
                    resized_arr = cv2.resize(img_arr, (self.img_size, self.img_size))  # Reshaping images to preferred size
                    self.data.append(resized_arr)
                    self.labels.append(class_num)
                except Exception as e:
                    print(e)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        image = self.data[idx]
        label = self.labels[idx]
        if self.transform:
            image = self.transform(image)
        return image, label
```

Set image size to 100

Label the images

```python
# Define transforms
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.RandomRotation(30),
    transforms.RandomResizedCrop(100, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Create datasets
labels = ['apple', 'watermelon', 'banana']
train_dataset = CustomDataset(os.path.join('dataset', 'train'), labels, transform=transform)
val_dataset = CustomDataset(os.path.join('dataset', 'val'), labels, transform=transform)
test_dataset = CustomDataset(os.path.join('dataset', 'test'), labels, transform=transform)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Implement Data Augmentation to the dataset by randomly: rotating, cropping and flipping

Generate Image Datasets

Build model layers

```
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.dropout = nn.Dropout(0.4)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(64 * 12 * 12, 128)
        self.fc2 = nn.Linear(128, 3)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.dropout(x)
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = CNNModel()
```

1. Create a PyTorch CNN model
2. Add the first Convolutional Layer and a Max pooling layer
3. Add another three more Convolutional Layers
4. Add Dropout and Flatten layers
5. Add two Fully Connected layers, with the final layer outputting 3 classes
6. Apply ReLU activation for the first Fully Connected Layer

Set parameters and load data

- Set variables for training the model

```
classes = np.unique(train_dataset.labels)
print (classes)
# Calculate class weights
class_weights = compute_class_weight(class_weight='balanced', classes=classes, y=train_dataset.labels)
print (class_weights)
#class_weights = [0.10, 1.42167256, 1.44033413]
class_weights = [0.10, 1.65607345, 0.96980976] # best performing class weights
# class_weights = [0.10, 2, 0.96980976]

# Convert class weights to a tensor
class_weights_tensor = torch.tensor(class_weights, dtype=torch.float).to(device)

# Define loss function with class weights
criterion = nn.CrossEntropyLoss(weight=class_weights_tensor)

# Optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Parameters for model training
num_epochs = 200
early_stopping_patience = 10
early_stopping_counter = 0
best_val_loss = float('inf')
```

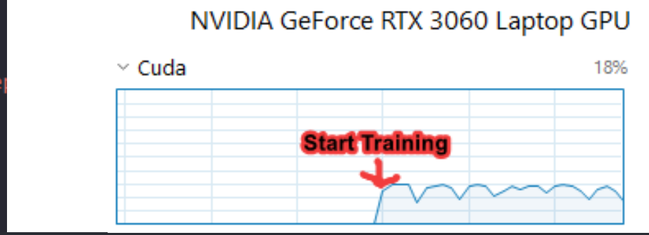
1. Calculate class weights using number of images of each class
2. Adjust the class weight as necessary by trial and error (Apple 0.10, Watermelon 1.656, Banana 0.969)
3. Adjust the learning rate as necessary by trial and error using the default (0.001) as a base
4. Used default Max Epochs (200) and default Early Stopping Patience (10)

Train the Model

- Train the model with either CPU or GPU (Nvidia CUDA) if available
- Training stops when Early Stopping Patience is met, or when all Epochs are completed

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```python
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()

        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(train_loader)
    epoch_acc = correct / total
    train_loss.append(epoch_loss)
    train_acc.append(epoch_acc)

    model.eval()
    val_running_loss = 0.0
    val_correct = 0
    val_total = 0

    with torch.no_grad():
        for val_images, val_labels in val_loader:
            val_images, val_labels = val_images.to(device), val_labels.to(device)
            val_outputs = model(val_images)
            val_loss_item = criterion(val_outputs, val_labels)

            val_running_loss += val_loss_item.item()
            _, val_predicted = torch.max(val_outputs.data, 1)
            val_total += val_labels.size(0)
            val_correct += (val_predicted == val_labels).sum().item()

    val_epoch_loss = val_running_loss / len(val_loader)
    val_epoch_acc = val_correct / val_total
    val_loss.append(val_epoch_loss)
    val_acc.append(val_epoch_acc)

    print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {epoch_loss:.4f}, Train Acc: {e

    if val_epoch_loss < best_val_loss:
        best_val_loss = val_epoch_loss
        torch.save(model.state_dict(), os.path.join('model', 'real_chatgpt.pth'))
        early_stopping_counter = 0
    else:
        early_stopping_counter += 1

    if early_stopping_counter >= early_stopping_patience:
        print("Early stopping")
        break

# Plotting
epochs_range = range(len(train_acc))
```




```
Epoch 1/200, Train Loss: 0.2692, Train Acc: 0.7113, Val Loss: 0.3300, Val Acc: 0.8914
Epoch 2/200, Train Loss: 0.0538, Train Acc: 0.9405, Val Loss: 0.1497, Val Acc: 0.9532
Epoch 3/200, Train Loss: 0.0469, Train Acc: 0.9497, Val Loss: 0.0695, Val Acc: 0.9831
Epoch 4/200, Train Loss: 0.0418, Train Acc: 0.9625, Val Loss: 0.1319, Val Acc: 0.9651
Epoch 5/200, Train Loss: 0.0275, Train Acc: 0.9680, Val Loss: 0.0618, Val Acc: 0.9811
Epoch 6/200, Train Loss: 0.0636, Train Acc: 0.9576, Val Loss: 0.1334, Val Acc: 0.9631
Epoch 7/200, Train Loss: 0.0293, Train Acc: 0.9640, Val Loss: 0.1935, Val Acc: 0.9462
Epoch 8/200, Train Loss: 0.0276, Train Acc: 0.9650, Val Loss: 0.0778, Val Acc: 0.9771
Epoch 9/200, Train Loss: 0.0258, Train Acc: 0.9755, Val Loss: 0.0564, Val Acc: 0.9811
Epoch 10/200, Train Loss: 0.0104, Train Acc: 0.9876, Val Loss: 0.1697, Val Acc: 0.9612
Epoch 11/200, Train Loss: 0.0108, Train Acc: 0.9853, Val Loss: 0.1811, Val Acc: 0.9502
Epoch 12/200, Train Loss: 0.0331, Train Acc: 0.9716, Val Loss: 0.0617, Val Acc: 0.9801
Epoch 13/200, Train Loss: 0.0079, Train Acc: 0.9883, Val Loss: 0.0249, Val Acc: 0.9920
Epoch 14/200, Train Loss: 0.0045, Train Acc: 0.9923, Val Loss: 0.0395, Val Acc: 0.9880
```

Evaluate the Model

```python
model.eval()

# Initialize variables to track test performance
test_loss = 0.0
test_correct = 0
test_total = 0

# No gradient needed for evaluation
with torch.no_grad():
    for test_images, test_labels in test_loader:
        test_images, test_labels = test_images.to(device), test_labels.to(device)

        # Forward pass
        test_outputs = model(test_images)
        loss = criterion(test_outputs, test_labels)

        # Update test loss
        test_loss += loss.item()

        # Calculate accuracy
        _, predicted = torch.max(test_outputs.data, 1)
        test_total += test_labels.size(0)
        test_correct += (predicted == test_labels).sum().item()

# Calculate average loss and accuracy
test_loss /= len(test_loader)
test_accuracy = test_correct / test_total

print(f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}')

# Optional: Generate a confusion matrix
all_preds = []
all_targets = []
with torch.no_grad():
    for test_images, test_labels in test_loader:
        test_images, test_labels = test_images.to(device), test_labels.to(device)
        outputs = model(test_images)
        _, preds = torch.max(outputs, 1)
        all_preds.extend(preds.cpu().numpy())
        all_targets.extend(test_labels.cpu().numpy())

conf_matrix = confusion_matrix(all_targets, all_preds)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
# Display the summary
#  (3, 100, 100) refers to the input size, with the input images being of 100x100 with 3 channels (RGB).
summary(model,(3,100,100))
```

- Evaluate the performance of the model by testing the model against the Test Dataset

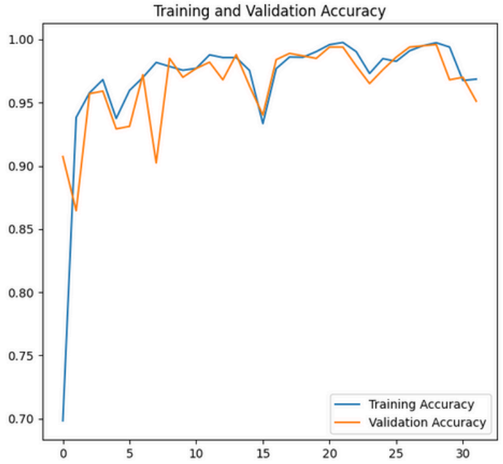
Real-time detecting

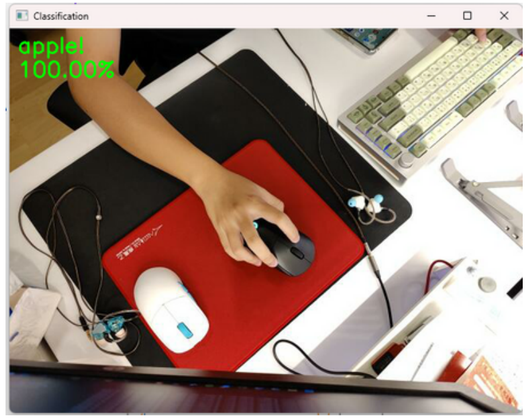
- Uses OpenCV to capture live video feed from webcam for real-time detecting
- 4 categories total: the 3 fruits and “unknown” are shown with their confidence levels
- “unknown” is shown when the confidence of any fruit is below 70% (percentage chosen from trial and error of accuracy)


```python
while (True):
    ret, original = cap.read()

    frame = cv2.resize(original, (100, 100))
    cv2.imwrite(filename='img.jpg', img=original)
    image = Image.open('img.jpg')

    prediction = import_and_predict(image, model)

    confidence_threshold = 0.8  # Adjusted based on model evaluation

    max_confidence_score = np.max(prediction)
    predicted_class_idx = np.argmax(prediction)

    # Check if the highest confidence score is below the threshold
    if max_confidence_score < confidence_threshold:
        predict = "unknown"
        confidence_text = "unknown"
    else:
        predict = class_labels[predicted_class_idx]
        confidence_text = f"{max_confidence_score*100:.2f}%"

    # Display all class confidences
    y_position = 30
    for i, (label, score) in enumerate(zip(class_labels, prediction[0])):
        text = f"{label}: {score*100:.2f}%"
        cv2.putText(original, text, (10, y_position), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
        y_position += 30

    cv2.putText(original, f"Prediction: {predict}", (10, y_position), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
    y_position += 30
    cv2.putText(original, f"Confidence: {confidence_text}", (10, y_position), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

    cv2.imshow("Classification", original)
```

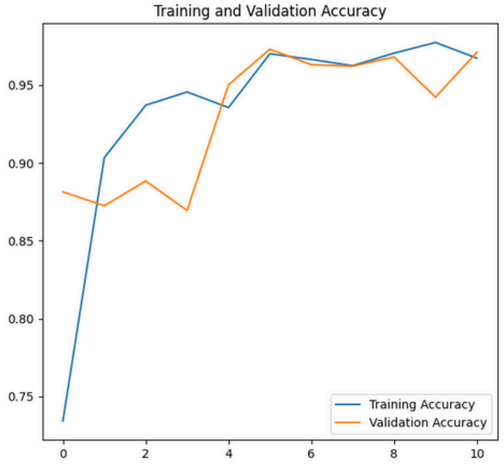
Previous Models

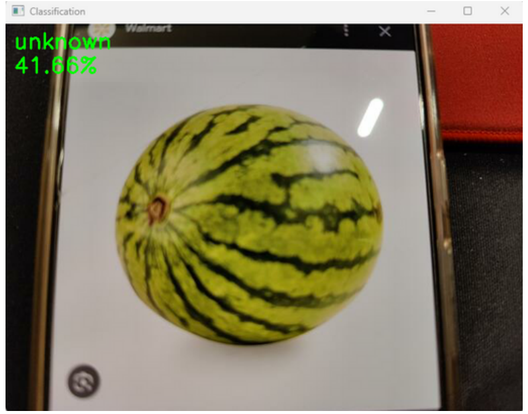
- Notable models trained during our project, and their downsides





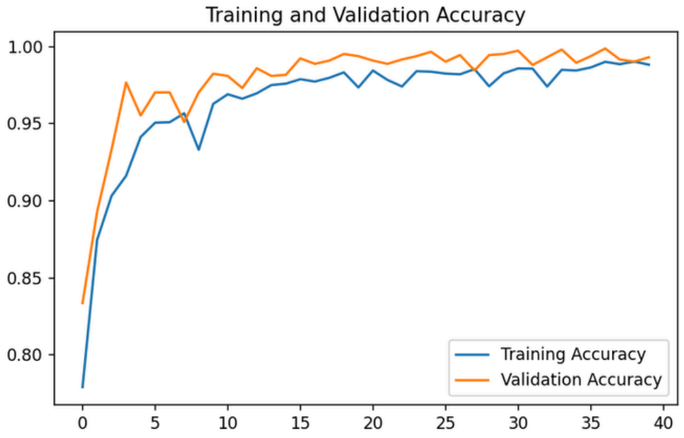
- class_weights = [0.08, 1.8, 0.96980976]
- early_stopping_patience = 10
- learning_rate=0.001
- last epoch=32
- model was overfitted and detects everything as an apple
- model was used to find the perfect class weight

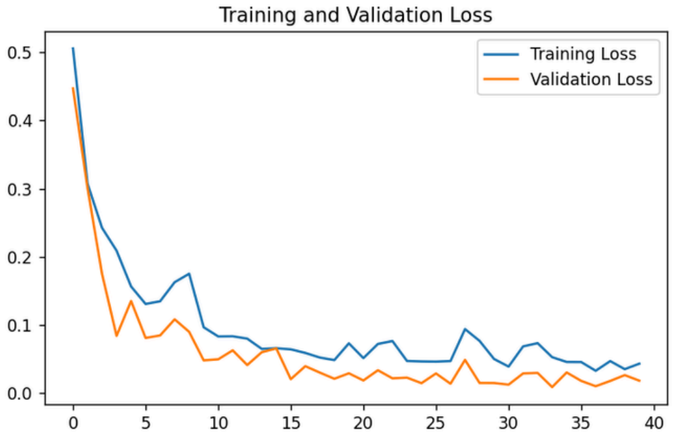




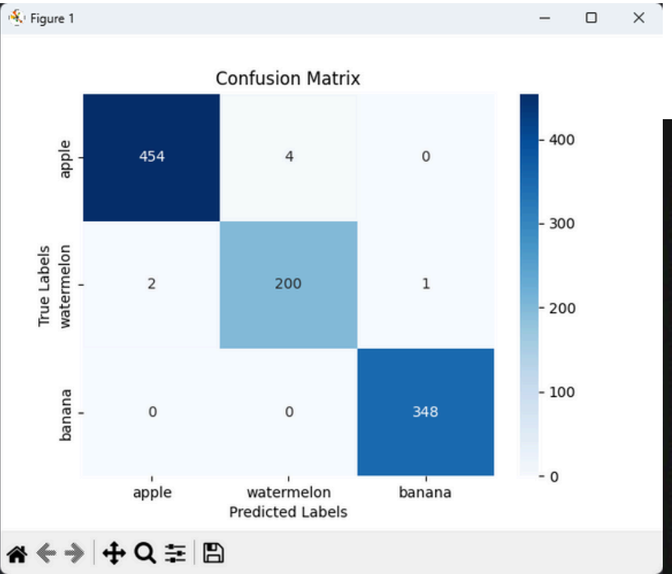
- class_weights = [0.08, 2.2, 0.96980976]
- early_stopping_patience = 5
- learning_rate=0.001
- last epoch=11
- model was slightly overfitted and was unable to detect watermelons
- model was used to find the perfect early stopping patience

Best Model





- class_weights = [0.10, 1.65607345, 0.96980976]
- early_stopping_patience = 10
- learning_rate=0.001
- last epoch=39
- model appears to be overfitted but it is able to recognize unseen images well
- this model is therefore chosen as our final model




```
Epoch 30/200, Train Loss: 0.0354, Train Acc: 0.9876, Val Loss: 0.0218, Val Acc: 0.9933
Epoch 31/200, Train Loss: 0.0303, Train Acc: 0.9903, Val Loss: 0.0162, Val Acc: 0.9955
Epoch 32/200, Train Loss: 0.0321, Train Acc: 0.9903, Val Loss: 0.2858, Val Acc: 0.9472
Epoch 33/200, Train Loss: 0.0509, Train Acc: 0.9847, Val Loss: 0.0397, Val Acc: 0.9866
Epoch 34/200, Train Loss: 0.0227, Train Acc: 0.9916, Val Loss: 0.0589, Val Acc: 0.9829
Epoch 35/200, Train Loss: 0.0362, Train Acc: 0.9874, Val Loss: 0.0262, Val Acc: 0.9918
Epoch 36/200, Train Loss: 0.0284, Train Acc: 0.9916, Val Loss: 0.0230, Val Acc: 0.9896
Epoch 37/200, Train Loss: 0.0299, Train Acc: 0.9914, Val Loss: 0.0339, Val Acc: 0.9851
Epoch 38/200, Train Loss: 0.0235, Train Acc: 0.9914, Val Loss: 0.0130, Val Acc: 0.9941
Epoch 39/200, Train Loss: 0.0170, Train Acc: 0.9949, Val Loss: 0.0324, Val Acc: 0.9874
Early stopping
```

Summary of the Model

```
Test Loss: 0.0447, Test Accuracy: 0.9921
----------------------------------------------------------------
       Layer (type)               Output Shape         Param #
================================================================
           Conv2d-1        [-1, 32, 100, 100]             896
         MaxPool2d-2          [-1, 32, 50, 50]               0
           Conv2d-3          [-1, 32, 50, 50]           9,248
         MaxPool2d-4          [-1, 32, 25, 25]               0
           Conv2d-5          [-1, 64, 25, 25]          18,496
         MaxPool2d-6          [-1, 64, 12, 12]               0
           Dropout-7          [-1, 64, 12, 12]               0
           Flatten-8                [-1, 9216]               0
            Linear-9                 [-1, 128]       1,179,776
           Linear-10                   [-1, 3]             387
================================================================
Total params: 1,208,803
Trainable params: 1,208,803
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.11
Forward/backward pass size (MB): 4.33
Params size (MB): 4.61
Estimated Total Size (MB): 9.06
----------------------------------------------------------------
```

- 3 Convolutional Layers, of which the first layer has a Max Pooling layer
- Applied Dropout layer to reduce overfitting
- 2 Fully-Connected layers in total at the end with the last one outputting 3 Classes

Test Results

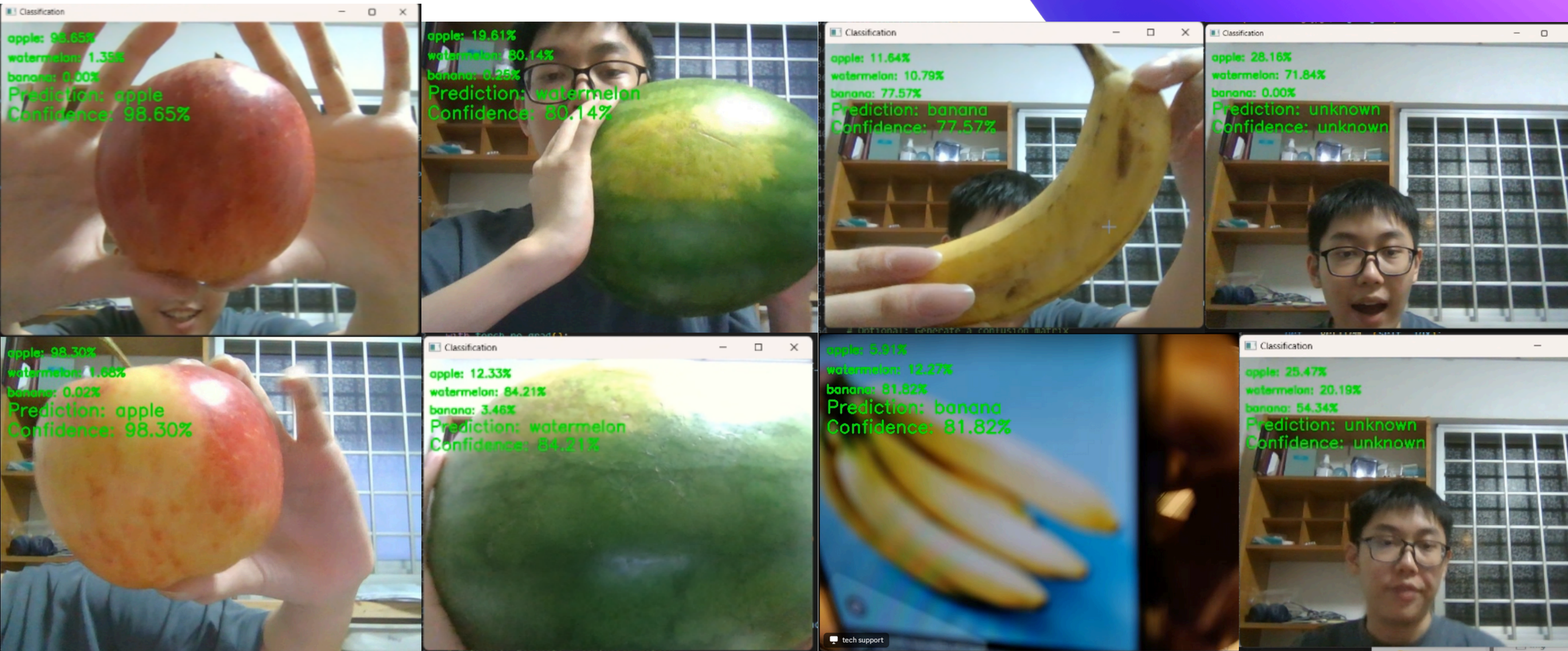
```
Test Loss: 0.0396, Test Accuracy: 0.9911
```

- Test Loss: 0.0396
- Test Accuracy: 0.9911
- Validation Loss: 0.0324 (Last Epoch)
- Validation Accuracy: 0.9874 (Last Epoch)

Conclusion:

- Test Loss is slightly higher than Validation Loss
  - Model is accurate as results are almost identical
- Test Accuracy slightly higher than Validation Accuracy
  - Model appears to perform better in the Test Dataset

Test with Unseen Images (Real Time)



Findings & Issues

- **Had many problems with recognizing single and multiple of the same fruit**
  - We had to take more pictures of different scenarios and increase our dataset
  - Lowering the Learning Rate improves the model slightly as it ensures the model learns more of the details of the images
- **Model was randomly classifying everything as an Apple at high confidence levels, even though nothing was changed between model trainings, causing us to waste a lot of our time building models**
  - “Fixes” found were to delete the previous model before training another, and re-train it until it goes away
  - Will always happen with no fix at certain class weights
- **Performance of the model can be heavily impacted by simple things such as learning rate as well as class weights**
  - A small increase of 0.001 to the class weight of Apple will cause the model to forget how a banana looks like

Potential Improvements

- **Increase size of dataset further with larger variety**
  - This includes more background variety and fruit counts
  - Larger dataset allows the model to cover more scenarios with higher accuracy
- **Setting “Unknown” as its own class**
  - Instead of using confidence level, we could use a random set of images with no fruits to determine unknown, which makes it more likely to classify new images as unknown rather than labelling it wrongly
- **Develop an objection detection model instead**
  - This will allow us to debug the model more easily as we can tell what exactly the model is looking at with its bounding boxes, and therefore work towards finetuning its sensitivity
- **Fine-tune the layer-building process**
  - The model can be further improved by spending time to finetune the parameters which affects the model accuracy

Thank you

References

- **Base Tensorflow Code**
  - Gautam, T., 2020. Image classification in Python with keras. [online] Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2020/10/create-image-classification-model-python-keras/> [Accessed 2 July 2024].
- **Conversion to PyTorch**
  - ChatGPT / Bard / Github Copilot
- **Resizing Python Script**
  - ChatGPT / Bard / Github Copilot
- **Sorting + Shuffling Python Script**
  - ChatGPT / Bard / Github Copilot
- **Real-time Object Detection Script**
  - R-P-S Sample Model Project (Brightspace)