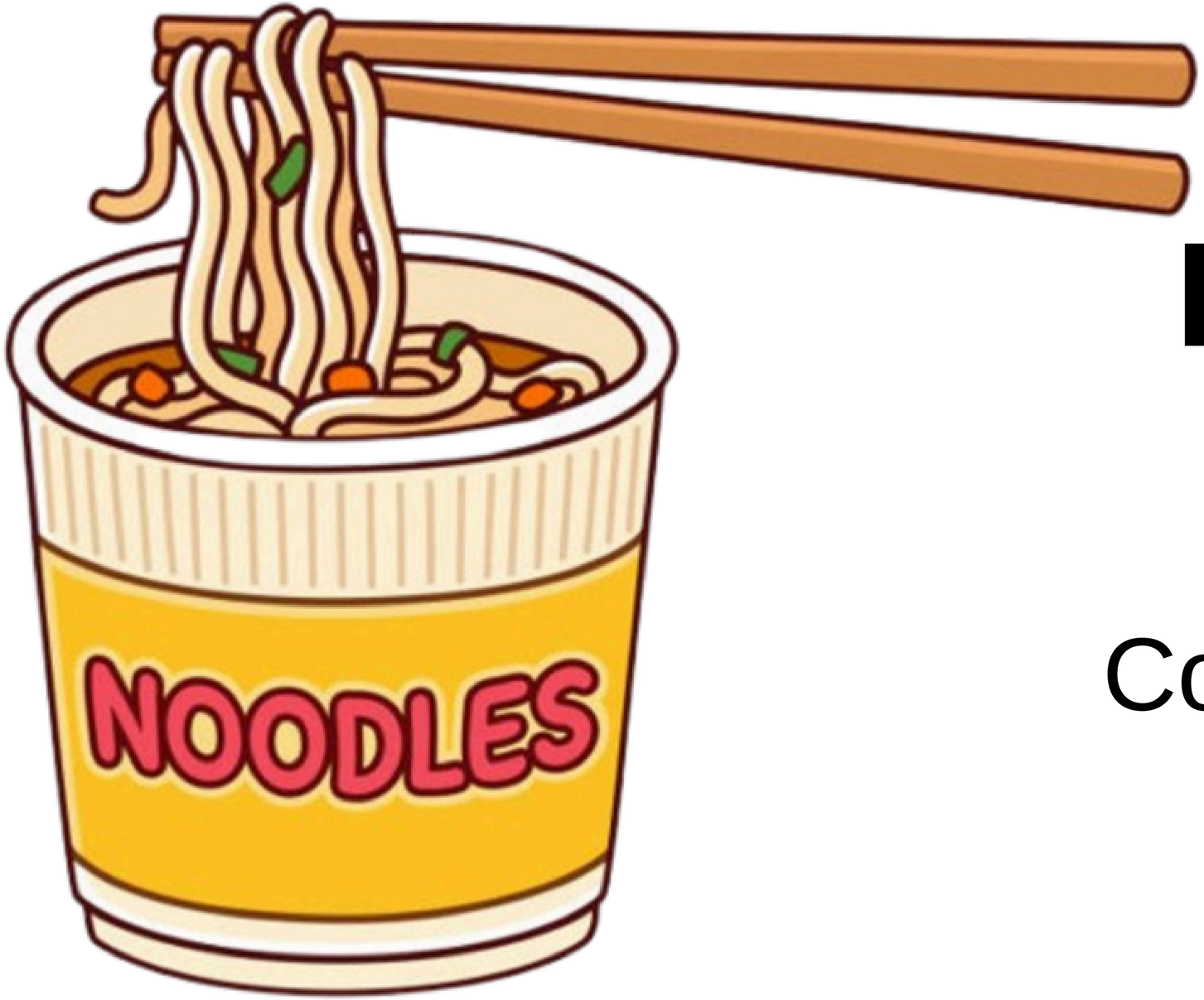


Instant Cup Noodles

Condition Detector

By: Bryan (P2319104)
Ei Zin (P2346438)

Problem statement

Why this project?

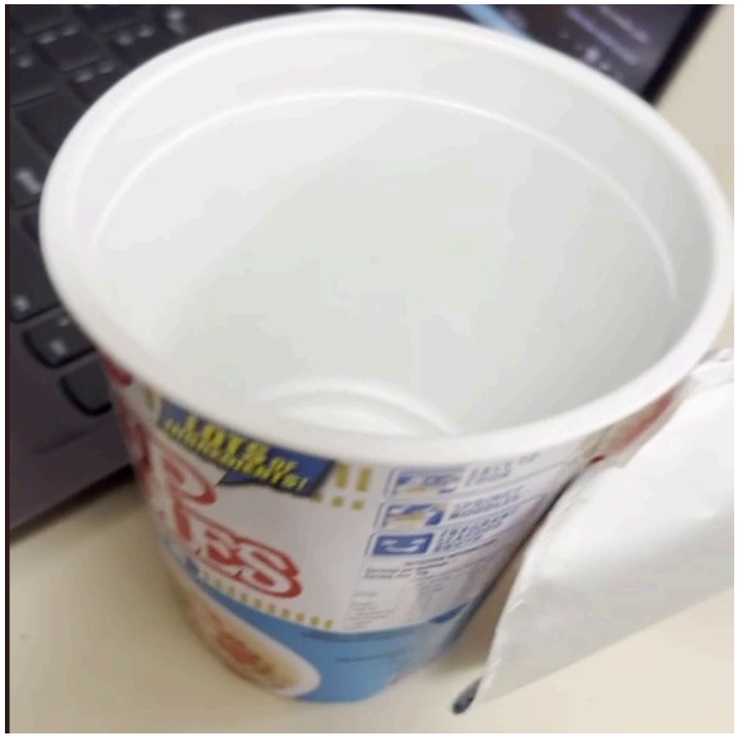
- Visual differences between cup noodle states can be subtle, and yet
- Manual inspection is:
  - Time-consuming
  - Inconsistent & Tedious
- A computer vision model can:
  - Learn visual patterns (cup noodles have a distinct shape)
  - Classify conditions consistently
  - Work in real time

Project Objective

Objective

- Detect the condition of instant cup noodles
- The conditions of the cup noodles are:
  - Sealed
  - Opened
  - Finished
- Support image testing and live camera detection.
- Be able to detect the condition reliably across multiple brands and sizes of cup noodles.

Classes



Finished

Opened

Sealed

Data Collection

How the dataset was created

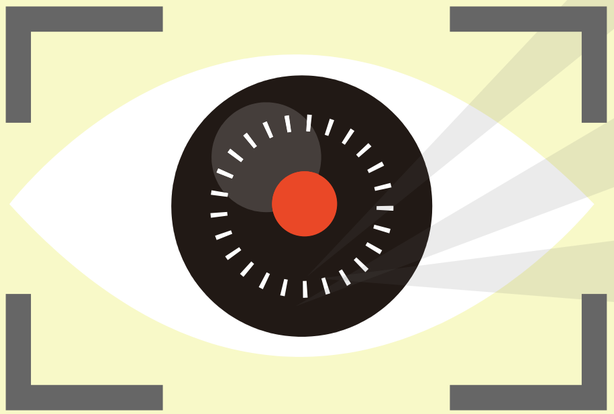


Image captured — Mobile phone camera & video

Multiple environments
- Different lighting conditions
- Different backgrounds
- Different angles

Dataset — Dataset prepared manually and labelled into 3 classes

Data Collection

```python
cap = cv2.VideoCapture(video_path) # Opens video file
```

```python
while True:
    ret, frame = cap.read() # read NEXT frame
    if not ret:
        break # no more frames in the video

    if frame_idx % step == 0:
        cv2.imwrite(f"{output_dir}/{filename}_{saved_idx}.jpg", frame)
        saved_idx += 1

    frame_idx += 1
```

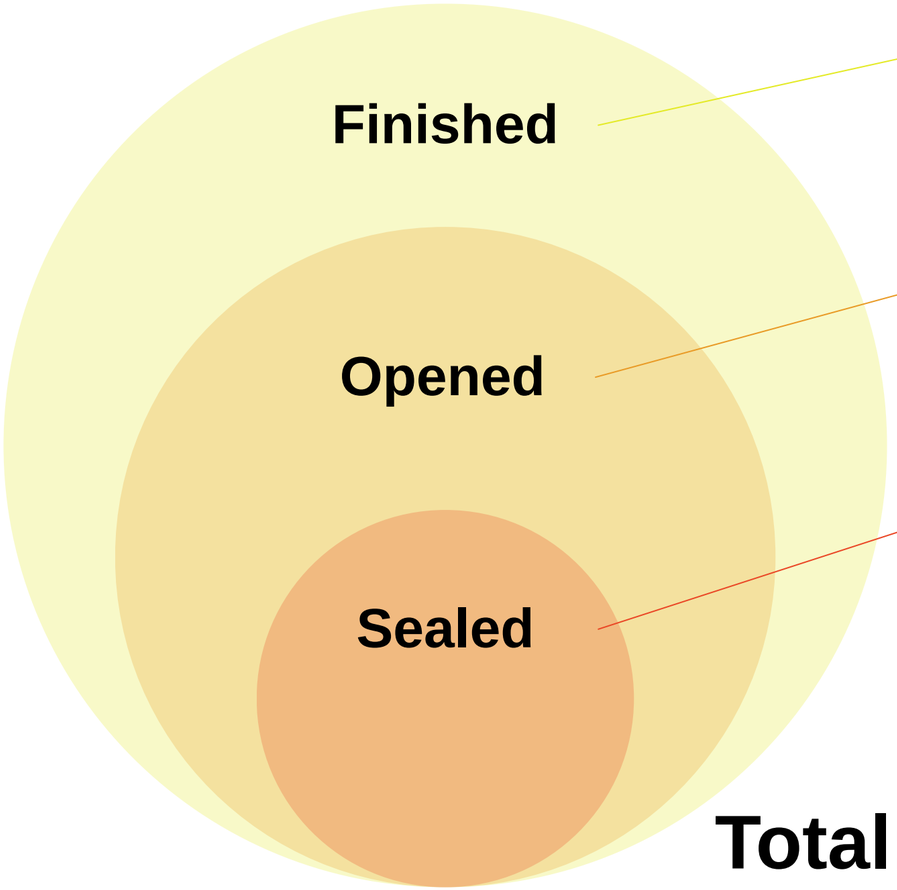
Video to Image Extraction

- Video files were converted into image frames using OpenCV
- Frames were extracted at 6 frames per second (FPS) to balance:
  - Dataset size
  - Redundancy between frames
  - Ensures enough time between frames so they are more unique
- Every extracted frame from a video was classified into:
  - Finished
  - Opened
  - Sealed

Why this approach was used?

- Increases dataset size without needing more physical samples.
- Captures different angles, lighting conditions, and motion blur.
- Improves model robustness to real-world camera input.
- Reduces overfitting by increasing data diversity.

Dataset Distribution

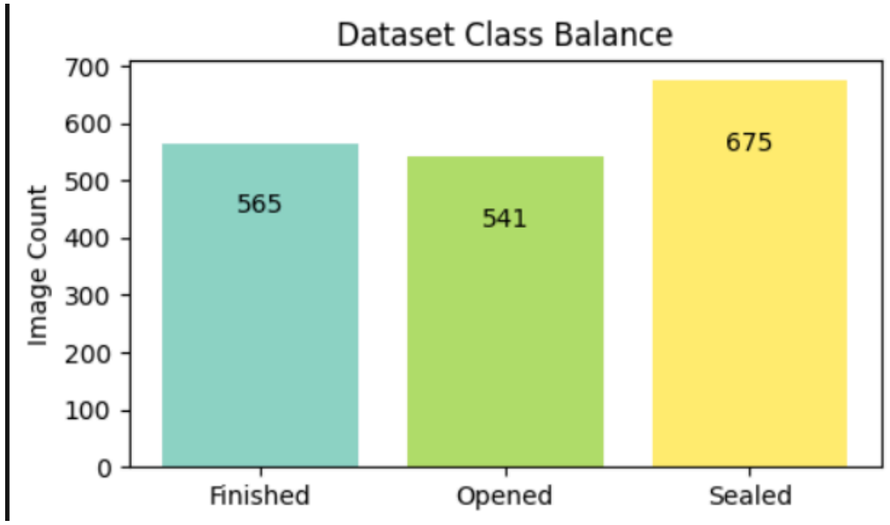


Finished: 31.7% (565 images)

Opened: 30.4% (541 images)

Sealed: 37.9% (675 images)

Total: 1781



- Class imbalance handled using class weighting
- Prevents bias towards majority class

Hyperparameters

```python
HYPER_PARAMS = {
    'IMG_SIZE': (299, 299), # Max: 299x299 for InceptionV3 but 224x224 also good
    'BATCH_SIZE': 24, # ~24 for Laptop, ~48 for PC
    'EPOCHS': 20, # 5 is ok, 12-18 is when it will early stop on its own
    'DATA_SPLIT': (80, 10, 10), # probably should not change
    'DF_SPLIT_SEED': 42, # controlled seed = replicatable outcomes & fair testing for hyperparam tuning
    'LEARNING_RATE': 1e-3,
    'DROPOUT_RATE': 0.3,
    'DENSE_UNITS': 128,
    'REDUCE_LR_PATIENCE': 2, # x epochs until it reduces learning rate to prevent bouncing
    'EARLY_STOPPING_PATIENCE': 3, # x epochs until it early stops
    'DISTRIBUTE_CLASS_WEIGHTS': True, # Default: True, will alleviate uneven class image distribution
}
```

Balancing Dataset Class Weights

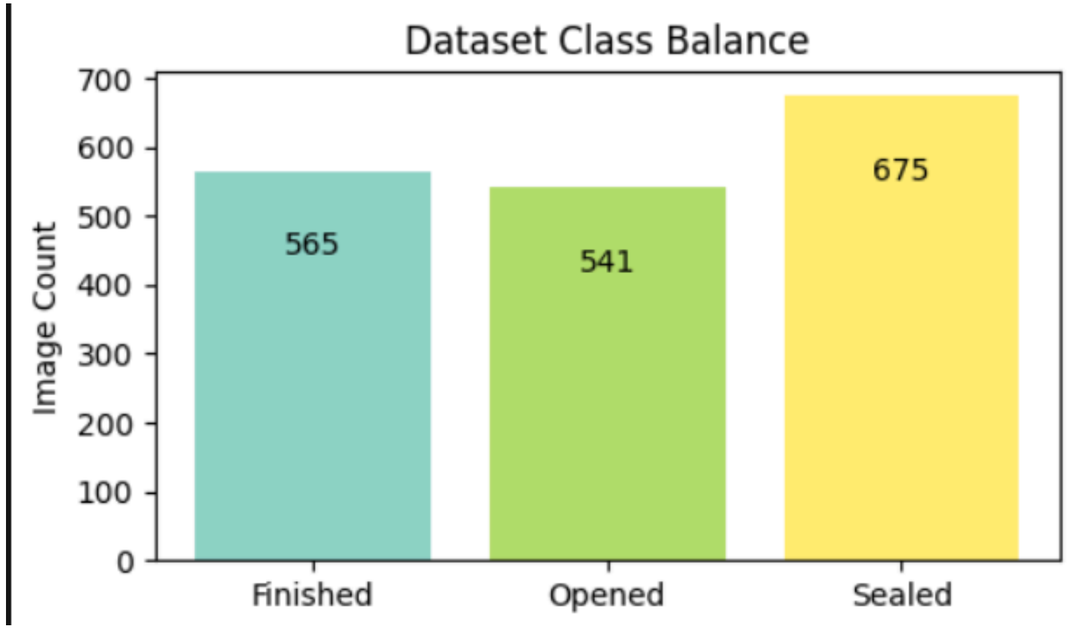
```python
print("\n⚖️ Balancing class weights...")
class_weights = compute_class_weight(
    'balanced', classes=np.unique(train_gen.classes), y=train_gen.classes
)
class_indices = train_gen.class_indices
class_weight_dict = {class_indices[class_name]: weight
                     for class_name, weight in zip(class_indices.keys(), class_weights)}
```

compute_class_weight() from **sklearn** is used.
class_weights_dict is passed into the model.fit()

```python
history = model.fit(
    train_gen,
    epochs=HYPER_PARAMS['EPOCHS'],
    validation_data=val_gen,
    class_weight=class_weight_dict,
    callbacks=callbacks,
    verbose=1 # prints the cool progress bar
)
```

Dataset Class Balance

- Dataset balance is checked before training
- Based on the number of images per class, weights are allocated to each class.
- For example (refer to the bar graph on the right):
  - **Sealed** will have a lower weightage having the **most** images (675)
  - **Opened** will have a higher weightage having the **least** images (541)
- This ensures fairness across classes & hence a better model overall.



Data Preprocessing

```python
# returns the 3 respective image generators
def build_image_generators():
    img_size = HYPER_PARAMS["IMG_SIZE"]
    batch_size = HYPER_PARAMS["BATCH_SIZE"]
    train_df, val_df, test_df = build_df_splits(data_dir)

    train_datagen = ImageDataGenerator(
        rescale=1./255, # normalizes pixel values
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True # 5 methods of data augmentation
    )
    val_test_datagen = ImageDataGenerator(rescale=1./255) # only normalize pixel values, no data aug
```

Image Preprocessing

- Pixel normalization:
  - rescale = 1 / 255
- Converts pixel values from 0–255 → 0–1
- Ensures consistent input across all images

Data Augmentation

Applied only to training data:

- Rotation
- Width & height shift
- Zoom
- Horizontal flip

Purpose:

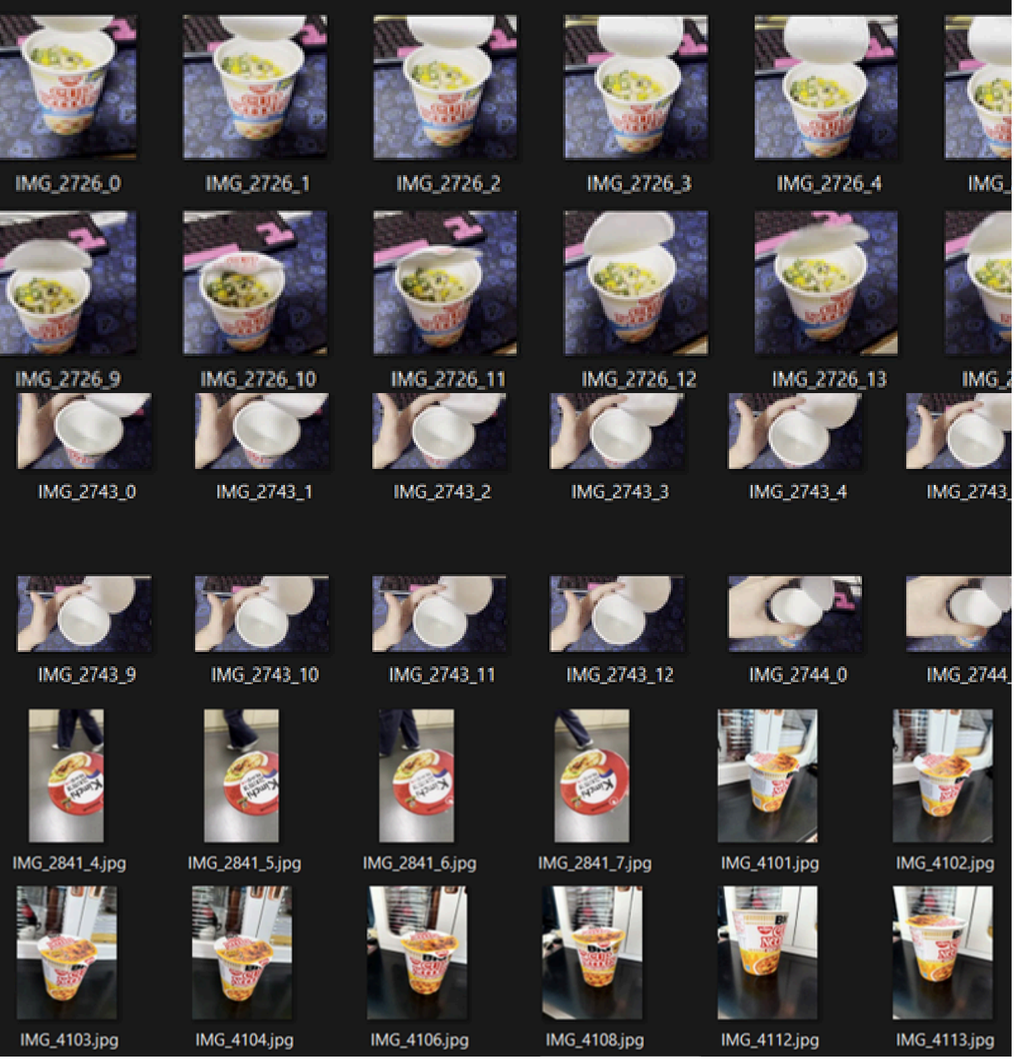
- Simulate real-world variations
- Reduce overfitting
- Improve robustness

Dataset Expansion/Splitting

```python
# Returns 3 shuffled data frames of the 80/10/10 data split
def build_df_splits(data_dir):
    seed = 42 # seed only controls the df splits
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)
    files = []
    labels = []

    valid_ext = (".jpg", ".jpeg", ".png")

    for cls in class_names:
        cls_dir = os.path.join(data_dir, cls)
        if not os.path.isdir(cls_dir):
            continue

        for f in os.listdir(cls_dir):
            if f.lower().endswith(valid_ext):
                files.append(os.path.join(cls_dir, f)) # gets all filepaths of images
                labels.append(cls) # gets all labels/class names of images

    df = pd.DataFrame({"filename": files, "class": labels})
    df = df.sample(frac=1, random_state=seed).reset_index(drop=True) # randomises its order

    n = len(df) # total number of images
    train_end = int(0.8 * n)
    val_end = int(0.9 * n)

    train_df = df[:train_end] # 80% goes to training dataframe
    val_df   = df[train_end:val_end] # 10% goes to training dataframe
    test_df  = df[val_end:] # 10% goes to training dataframe
```

Data Splitting Strategy (80 / 10 / 10)

- Dataset is split into:
  - 80% Training
  - 10% Validation
  - 10% Testing
- Split is:
  - Randomized
  - Reproducible using a fixed seed
- Prevents data leakage between sets
- Only .jpg, .jpeg, .png used



299x299 (Early stop, same seed) → 13min

✅ Test accuracy: 0.9832

224x224 (Early stop, same seed) → 10min

✅ Test accuracy: 0.9608

Model Preparation

Model Preparation

- All images are resized to 299 × 299
- Reason:
  - InceptionV3 is pretrained on ImageNet using 299×299 images
  - Preserves more spatial detail compared to smaller inputs
  - Our dataset consists of images of many different sizes
- Trade-offs:
  - Higher validation accuracy (97.7% at convergence)
  - BUT Longer training time

➡️ Chosen input size: 299 × 299 for improved accuracy

Model Architecture

```python
def build_model():
    base_model = InceptionV3(input_shape=(*HYPER_PARAMS["IMG_SIZE"], 3), include_top=False, weights='imagenet')
    base_model.trainable = False
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.3)(x)
    x = Dense(128)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Dropout(0.3)(x)
    predictions = Dense(len(class_names), activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=predictions)
    return model
```

Model Architecture

- **Base model:** InceptionV3
  - Pretrained on ImageNet
- **Feature extractor layers are:**
  - Frozen
  - Not retrained
- Custom classification head added on top

Why InceptionV3?

- Strong feature extraction
- Good performance on real-world images
- Faster convergence
- Higher val_acc than other models (we tried)

Custom Classification Head

Added layers:

- Global Average Pooling
- Dense layer (128 units)
- Batch Normalisation
- ReLU activation
- Dropout (0.3)
- Final Softmax layer (3 classes)

Benefits:

- GAP: reduces parameters
- BatchNorm: stabilizes training
- Dropout: prevents overfitting
- Softmax: outputs probabilities

Model Callbacks

```
callbacks = [
    EarlyStopping(patience=3, restore_best_weights=True, monitor='val_accuracy'),
    ReduceLROnPlateau(factor=0.2, patience=2, min_lr=1e-7) # slows learning rate
]
```

Early Stopping

Stops the model early after **3 epochs** *(AKA patience)* of no improvements in validation accuracy.

Helps save time when the model has possibly reached its minima (converged).

Reduce LR on Plateau

(LR = Learning Rate)

Lowers learning rate of model after **2 epochs** *(AKA patience)* of no improvements in validation accuracy.

Helps the model escape local minima, where it bounces near the minima because learning rate is too high.

Image Size Experiment

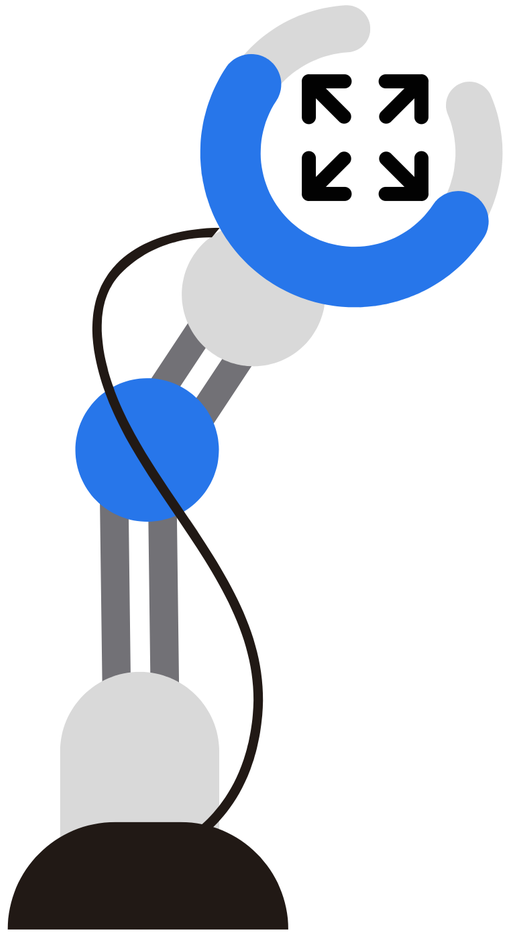


Image Size Experiment

- Initial experiments used smaller image sizes
- Later experiments increased image size
- Observation:
  - Larger image size improved validation accuracy (12.7%)
  - Training time increased (31.3%)

Conclusion:

Increasing image size improves accuracy when using transfer learning models like InceptionV3

```
# ===== CONFIG =====
IMG_SIZE = (75, 75)
BATCH_SIZE = 32
EPOCHS = 8
SEED = 42
```

Smaller image size (75x75)

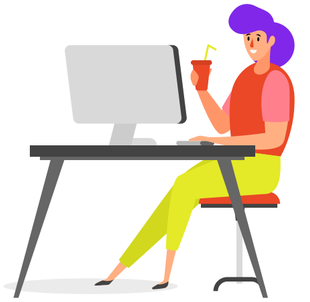
```
accuracy: 0.7959
```

```
HYPER_PARAMS = {
    'IMG_SIZE': (299, 299), # Max: 29
    'BATCH_SIZE': 48, # ~24 for Laptc
    'EPOCHS': 1, # 12-18 is when it w
    'DISTRIBUTE_CLASS_WEIGHTS': True
}
```

Larger image size (299x299)

```
accuracy: 0.8968
```

Experiment Comparison



Experiment Comparison

- Average training time increased:
  - +45.8%
- Final validation accuracy improved:
  - +12.7%
- Trade-off:
  - Slower training
  - Higher accuracy

Conclusion:

The improvement in accuracy is worth the increased training time

```
Epoch 1/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 122s 4s/step - accuracy: 0.5990 - loss: 1.2552 - val_accuracy: 0.8101 - val_loss: 0.4866
Epoch 2/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 111s 4s/step - accuracy: 0.7239 - loss: 0.8375 - val_accuracy: 0.7475 - val_loss: 0.5711
Epoch 3/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 121s 5s/step - accuracy: 0.7743 - loss: 0.6203 - val_accuracy: 0.8255 - val_loss: 0.4284
Epoch 4/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 120s 4s/step - accuracy: 0.7911 - loss: 0.5140 - val_accuracy: 0.8441 - val_loss: 0.3921
Epoch 5/20
27/27 ━━━━━━━━━━━━━━━━━━━━ 120s 5s/step - accuracy: 0.7959 - loss: 0.5301 - val_accuracy: 0.8661 - val_loss: 0.3444
```

IMG_SIZE: 75x75

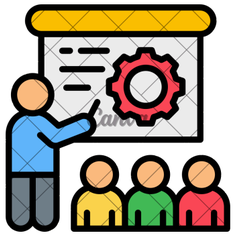
```
accuracy: 0.7959
```

```
Epoch 1/5
35/35 ━━━━━━━━━━━━━━━━━━━━ 192s 5s/step - accuracy: 0.6466 - loss: 1.0876 - val_accuracy: 0.8666 - val_loss: 0.3747
Epoch 2/5
35/35 ━━━━━━━━━━━━━━━━━━━━ 170s 5s/step - accuracy: 0.8018 - loss: 0.4843 - val_accuracy: 0.9058 - val_loss: 0.2629
Epoch 3/5
35/35 ━━━━━━━━━━━━━━━━━━━━ 166s 5s/step - accuracy: 0.8457 - loss: 0.3841 - val_accuracy: 0.8741 - val_loss: 0.3043
Epoch 4/5
35/35 ━━━━━━━━━━━━━━━━━━━━ 178s 5s/step - accuracy: 0.8685 - loss: 0.3583 - val_accuracy: 0.9374 - val_loss: 0.1820
Epoch 5/5
35/35 ━━━━━━━━━━━━━━━━━━━━ 160s 5s/step - accuracy: 0.8968 - loss: 0.3069 - val_accuracy: 0.9466 - val_loss: 0.1674
```

IMG_SIZE: 224x224

```
accuracy: 0.8968
```

Training Configuration



Training Configuration

- Optimizer: Adam
- Loss function: Categorical Cross-Entropy
- Learning rate adjusted dynamically
- Callbacks:
  - EarlyStopping
  - ReduceLROnPlateau
- Class weights applied to balance dataset

```python
# 3. Build the model's layers
print("\n🔨 Building model...")
model = build_model()
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# 4. Training the model
print("\n🚀 Training...")
callbacks = [
    EarlyStopping(patience=3, restore_best_weights=True, monitor='val_accuracy'), # patience means wait x epochs before doing smth
    ReduceLROnPlateau(factor=0.2, patience=2, min_lr=1e-7) # slows learning rate to prevent it from bouncing
]
start_time = time.time()
history = model.fit(
    train_gen,
    epochs=HYPER_PARAMS['EPOCHS'],
    validation_data=val_gen,
    class_weight=class_weight_dict,
    callbacks=callbacks,
    verbose=1 # prints the cool progress bar
)
training_time = time.time() - start_time
model_accuracy = model.evaluate(test_gen)[1]
```

Overfitting Prevention Techniques

```python
# returns the 3 respective image generators
def build_image_generators():
    img_size = HYPER_PARAMS["IMG_SIZE"]
    batch_size = HYPER_PARAMS["BATCH_SIZE"]
    train_df, val_df, test_df = build_df_splits(data_dir)

    train_datagen = ImageDataGenerator(
        rescale=1./255, # normalizes pixel values
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True # 5 methods of data augmentation
    )
    val_test_datagen = ImageDataGenerator(rescale=1./255) # only normalize pixel values, no data aug
```

Techniques Used

- Data augmentation
- Dropout layers
- Batch normalization
- Early stopping
- Learning rate reduction
- Class weighting

Effectiveness

- Reduced training–validation gap
- Improved stability

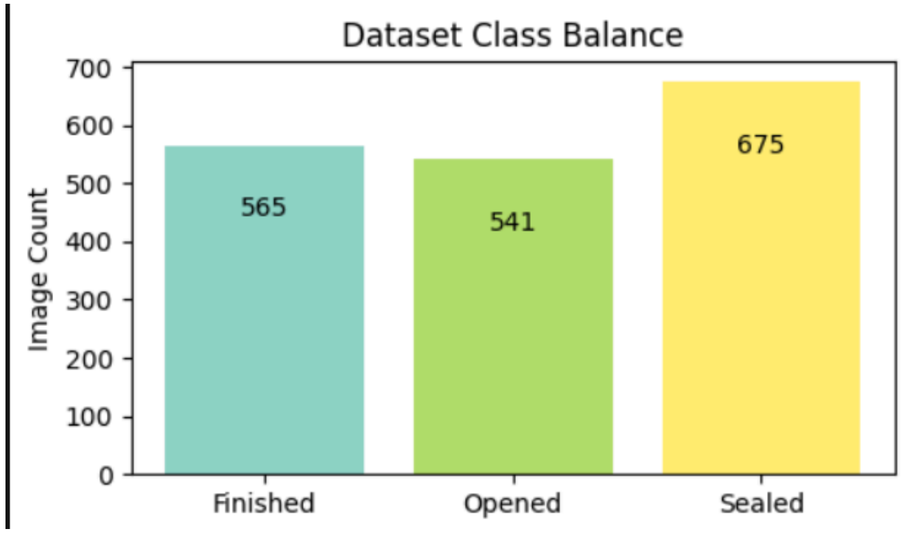
```python
# 4. Training the model
print("\n🚀 Training...")
callbacks = [
    EarlyStopping(patience=3, restore_best_weights=True, monitor='val_accuracy'), # patience means wait x epochs before doing smth
    ReduceLROnPlateau(factor=0.2, patience=2, min_lr=1e-7) # slows learning rate to prevent it from bouncing
]
```

Dataset Class Balance

Dataset Class Balance

- Dataset balance is checked before training
- Number of images per class is displayed
- Bar chart used to visualise class distribution
- Class imbalance handled using:
  - Computed class weights
  - Prevents bias towards dominant classes
- Visualization created using Matplotlib bar charts

```python
def check_dataset_balance():
    counts = {}
    for class_dir in sorted(class_names):
        class_path = os.path.join(data_dir, class_dir)
        if os.path.isdir(class_path):
            counts[class_dir] = len([f for f in os.listdir(class_path)
                                     if f.lower().endswith(('.jpg', '.png', '.jpeg'))])

    total = sum(counts.values())
    print("Dataset Balance:")
    print(counts)
    print("Ratios:", {k: f"{v/total*100:.1f}%" for k,v in counts.items()})

    plt.figure(figsize=(5, 4))
    colors = plt.cm.Set3(np.linspace(0, 1, len(class_names)))
    bars = plt.bar(class_names, counts.values(), color=colors)
    plt.bar_label(bars, padding=-30)
    plt.ylabel('Image Count')
    plt.title('Dataset Class Balance')
    plt.tight_layout()
    plt.show()

check_dataset_balance()
```



Accuracy & Loss Analysis

```python
print(history.history)
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Accuracy
axes[0].plot(history.history['accuracy'], label='Train Acc')
axes[0].plot(history.history['val_accuracy'], label='Val Acc')
axes[0].set_title('Accuracy over Epochs')
axes[0].set_xlabel('Epoch')
axes[0].set_ylabel('Accuracy')
axes[0].legend()
axes[0].grid(True)

# Loss
axes[1].plot(history.history['loss'], label='Train Loss')
axes[1].plot(history.history['val_loss'], label='Val Loss')
axes[1].set_title('Loss over Epochs')
axes[1].set_xlabel('Epoch')
axes[1].set_ylabel('Loss')
axes[1].legend()
axes[1].grid(True)

plt.tight_layout()
plt.show()
plt.close()
```

Why this matters

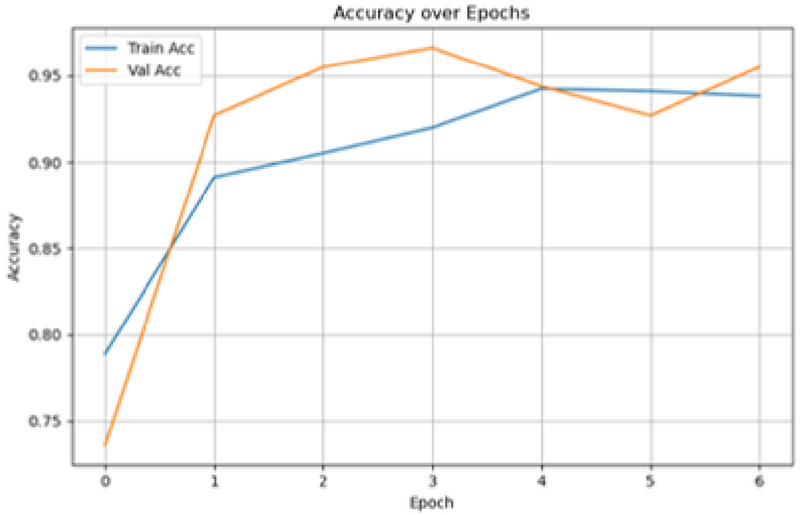
- Detect overfitting
- Check convergence behaviour

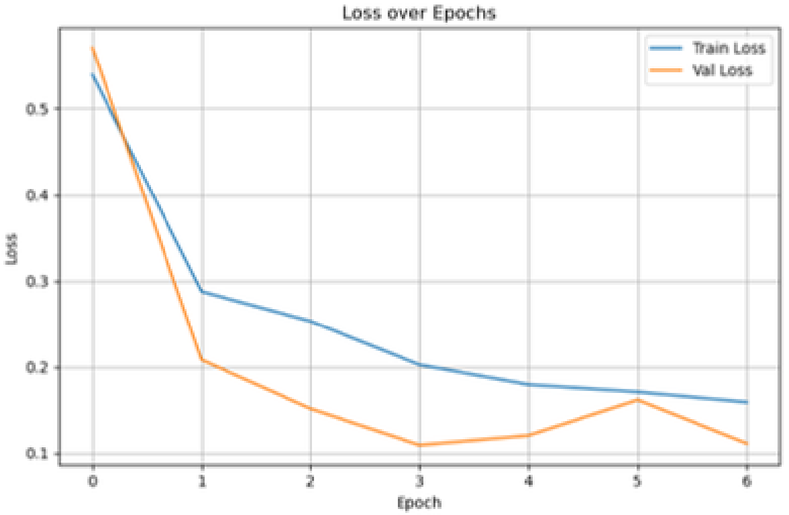
What we plotted

- Training Accuracy vs Validation Accuracy
- Training Loss vs Validation Loss

Observation

- Validation curves closely follow training curves
- No severe overfitting





Confusion Matrix

```python
if 'y_pred_prob' not in locals(): # skips code if already have the values
    print("🔄 Computing predictions...")
    test_gen.reset() # resets generator's pointer to 0
    y_pred_prob = model.predict(test_gen, verbose=0)
    y_pred = np.argmax(y_pred_prob, axis=1) # predictions
    y_true = test_gen.classes # labels

cm = confusion_matrix(y_true, y_pred)
cmdisplay = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
cmdisplay.plot(cmap=plt.cm.Blues, xticks_rotation=45)
plt.title('Confusion Matrix (From Test Dataset)')
plt.show()
plt.close()
```

Purpose

- Identify which classes confuses the model
- Analyse misclassification patterns

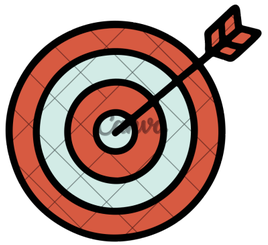
Findings

- Most errors occur between:
  - Opened ↔ Sealed
- Finished class highly distinct

Example (refer to the image):

- The **opened** class has the most **wrongly predicted** images (model confused) - so more images may be required in that class for training.

Precision, Recall & F1 Score



```python
if 'y_pred_prob' not in locals():
    print("🔄 Computing predictions...")
    test_gen.reset()
    y_pred_prob = model.predict(test_gen, verbose=0)
    y_pred = np.argmax(y_pred_prob, axis=1)
    y_true = test_gen.classes

print(classification_report(y_true, y_pred, target_names=class_names))
```

	precision	recall	f1-score	support
Finished	0.88	0.86	0.87	57
Opened	0.89	0.51	0.65	61
Sealed	0.68	0.98	0.81	61
accuracy			0.78	179
macro avg	0.81	0.78	0.77	179
weighted avg	0.81	0.78	0.77	179

Why metrics beyond accuracy?

- Accuracy alone hides class-specific errors

Metrics Used

- Precision
- Recall
- F1-Score
- Support

Interpretation

- Balanced precision & recall
- Model not biased towards any class

Example (Image on left)

- Opened has a much higher precision than recall, which means the model is too **careful** in that aspect.
- Sealed has a much higher recall than precision, which means the model is **many false alarms** there.

Misclassified Image Analysis

```python
COLUMNS = 4
SORT_BY_DESCENDING_CONFIDENCE = True

if 'y_pred_prob' not in locals(): # skips code if already have the values
    print("🔄 Computing predictions...")
    test_gen.reset() # resets generator's pointer to 0
    y_pred_prob = model.predict(test_gen, verbose=0)
    y_pred = np.argmax(y_pred_prob, axis=1) # predictions
    y_true = test_gen.classes # labels

mis_idx = np.where(y_true != y_pred)[0]
confidences = np.max(y_pred_prob[mis_idx], axis=1)
misclassified = test_df.iloc[mis_idx].copy()
misclassified['pred_class'] = [class_names[p] for p in y_pred[mis_idx]]
misclassified['confidence'] = confidences

misclassified = misclassified.sort_values('confidence', ascending=not SORT_BY_DESCENDING_CONFIDENCE) # Sort by highest confidence first

print(f"🔍 Found {len(misclassified)} misclassified images")

count = len(misclassified)
rows = (count + COLUMNS - 1) // COLUMNS
```

IMG_4239.jpg
Predicted: SEALED (0.965)
Is actually: OPENED

Angle too low to see inside contents

IMG_2736_2.jpg
Predicted: SEALED (0.703)
Is actually: OPENED

Seal covers too much to be “OPENED”

IMG_2851_0.jpg
Predicted: FINISHED (0.612)
Is actually: OPENED

Poor lighting - Noodles too bright; blends in

Purpose

- Identify difficult samples
- Improve dataset quality

Approach

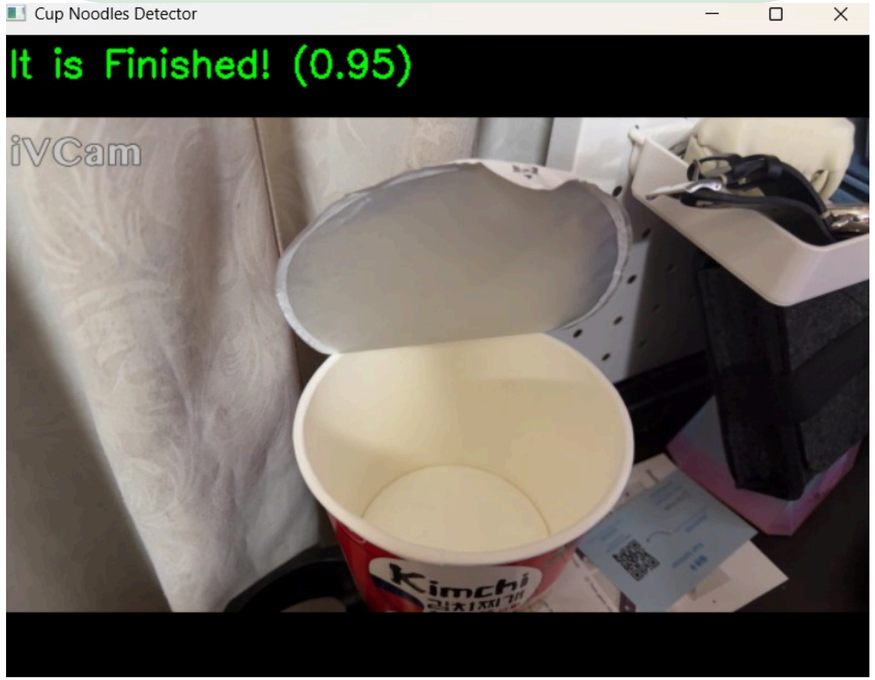
- Extract wrongly predicted images
- Sort by confidence
- Visual inspection

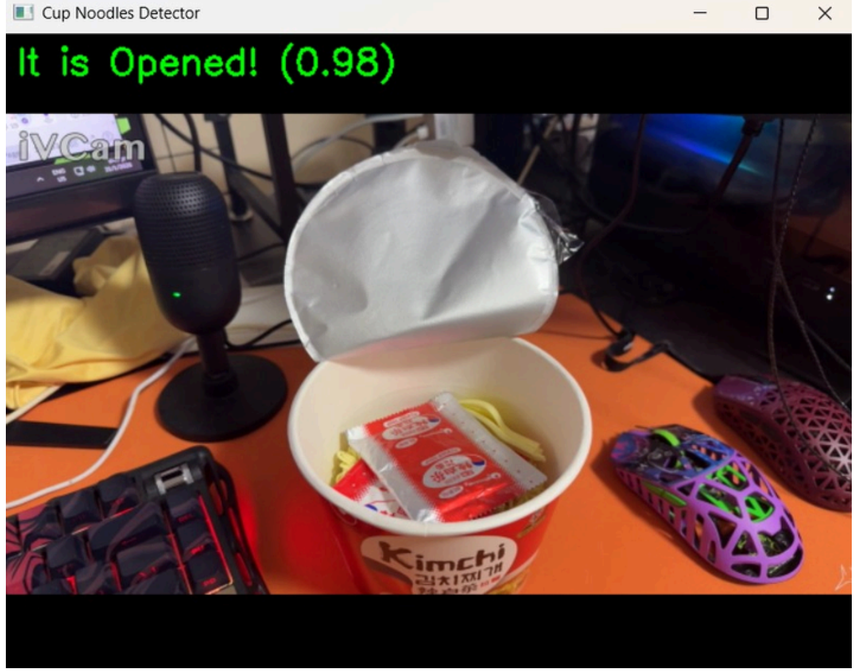
Insights

- Some images:
  - Poor lighting
  - Ambiguous content
  - Confusing angles
- Helps guide future data collection

Live Camera Detection

- Uses OpenCV camera stream
- Each frame is:
  - Preprocessed
  - Passed through the trained model
- Output displayed in real time:
  - Predicted class
  - Confidence score
- Press Q to stop the camera





Training Results

```
🚀 Training...
Epoch 1/6
60/60 ━━━━━━━━━━━━━━━━━━━━ 353s 5s/step - accuracy: 0.7992 - loss: 0.5533 - val_accuracy: 0.8876 - val_loss: 0.3517 - learning_rate: 0.0010
Epoch 2/6
60/60 ━━━━━━━━━━━━━━━━━━━━ 212s 4s/step - accuracy: 0.9031 - loss: 0.2524 - val_accuracy: 0.9157 - val_loss: 0.2320 - learning_rate: 0.0010
Epoch 3/6
60/60 ━━━━━━━━━━━━━━━━━━━━ 210s 3s/step - accuracy: 0.9115 - loss: 0.2582 - val_accuracy: 0.9382 - val_loss: 0.1736 - learning_rate: 0.0010
Epoch 4/6
60/60 ━━━━━━━━━━━━━━━━━━━━ 242s 4s/step - accuracy: 0.9305 - loss: 0.1952 - val_accuracy: 0.9438 - val_loss: 0.1448 - learning_rate: 0.0010
Epoch 5/6
60/60 ━━━━━━━━━━━━━━━━━━━━ 246s 4s/step - accuracy: 0.9361 - loss: 0.1712 - val_accuracy: 0.9551 - val_loss: 0.1174 - learning_rate: 0.0010
Epoch 6/6
60/60 ━━━━━━━━━━━━━━━━━━━━ 232s 4s/step - accuracy: 0.9487 - loss: 0.1467 - val_accuracy: 0.9326 - val_loss: 0.1545 - learning_rate: 0.0010
8/8 ━━━━━━━━━━━━━━━━━━━━ 24s 3s/step - accuracy: 0.9777 - loss: 0.0684
⏱ Total training time: (25.14 minutes)
```

Model Performance – Test Accuracy

- Test Accuracy: 97.7%
- The test dataset contains images not used during training or validation
- Measures the model’s ability to generalise to unseen data
- The model correctly classifies about 98 out of every 100 images

This indicates that the model is:

- Reliable
- Not overfitted
- Able to generalise well to new images
- Suitable for real-world image classification tasks

Model Saving Strategy

```python
# 5. Simple post-training metrics
print(f"⏳ Total training time: ({training_time/60:.2f} minutes)")
print("\n✅ Test accuracy:", model_accuracy)

# 6. Save the model as a .keras file with accuracy & timestamp
timestamp = datetime.datetime.now().strftime("%b%d_%H%M")
filename = f"{int(model_accuracy * 100)}acc_model_{timestamp}.keras"
model.save(f'./models/{filename}')
print(f"💾 Saved: {filename}")
```

Model Saving Strategy

- Model saved in .keras format
- Filename includes:
  - Test accuracy
  - Timestamp
- Allows:
  - Easy version comparison
  - Model reuse without retraining

Improvements & Future Work

- Increase dataset size
- Collect more edge cases
- Fine-tune deeper layers of InceptionV3
- Implement confidence threshold for unknown detection
- Explore other pre-trained models (EfficientNet, MobileNet)

Conclusion & Key Findings

- Developed an Instant Cup Noodles Condition Detector (Finished, Opened, Sealed)
- Transfer learning (InceptionV3) improved accuracy and training efficiency
- Larger image sizes (224×224 / 299×299) improved feature extraction
- Overfitting was reduced using data augmentation, dropout, batch normalization, and early stopping
- Video-to-image extraction increased dataset diversity and robustness
- Final model shows high test accuracy and good generalisation, making it suitable for real-world use


THANK YOU