Convolutional Network

Tan Wen Tao Bryan
2214449
DAAA/FT/2B/01

Background Research

Vegetables Image Classification

- DCNN-Based Vegetable Image Classification is used to conduct an experiment on the performance of CNN. This work proposes the study between such typical CNN and its architectures (VGG16, MobileNet, InceptionV3, ResNet etc.) to build up which technique would work best regarding accuracy and effectiveness with new image datasets.
- Additionally, several pre-trained CNN architectures using transfer learning are employed to compare the accuracy with the typical CNN. Study shows that by utilizing previous information gained from related large-scale work, the transfer learning technique can achieve better classification results over traditional CNN with a small dataset.

Convolutional Neural Network

- Take in input image, assign importance (learnable weights and biases) to various aspects in the image, and be able to differentiate one from the other
- Extracts features from the image
- CNN contains the following layers:
  - Input layer (ex. grayscale image)
  - Output layer which is a binary or multi-class labels
  - Hidden layers consisting of convolution layers, ReLU (rectified linear unit) layers, the pooling layers and a fully connected network

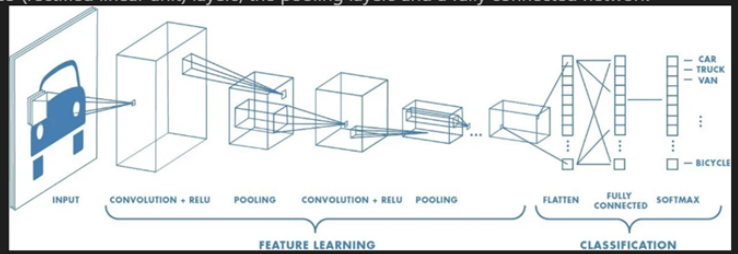


Image Source: Standford University, 2018

Step 1: EDA

Image Visualisation

- Let's see what we are working with

```python
# Retrieve the list of classes
image_categories = os.listdir(train_path)

def plot_images(image_categories):
    plt.figure(figsize=(10, 6))

    for i, cat in enumerate(image_categories):
        image_path = f"{train_path}/{cat}"
        images_inFolder = os.listdir(image_path)
        firstImage_inFolder = images_inFolder[0]
        firstImage_path = f"{image_path}/{firstImage_inFolder}"

        # Load images
        img = tf.keras.utils.load_img(firstImage_path)
        # Normalise the pixel values to the range 0 to 1
        img_arr = tf.keras.utils.img_to_array(img)/255.0

        # Create subplots and plot images
        plt.subplot(3, 5, i+1)
        plt.imshow(img_arr)
        plt.title(cat)
        plt.axis('off')
    plt.tight_layout()
    plt.show()

plot_images(image_categories)
```

Observations

- There are 15 types of vegetables as labels.
- Data augmentation like image rotation can help for a better prediction.
- Some images like bean and tomato above are quite zoomed out, so zooming in can help with a better prediction.
- Images do not seem to be mislabelled according to the sample of images.

Step 1: EDA

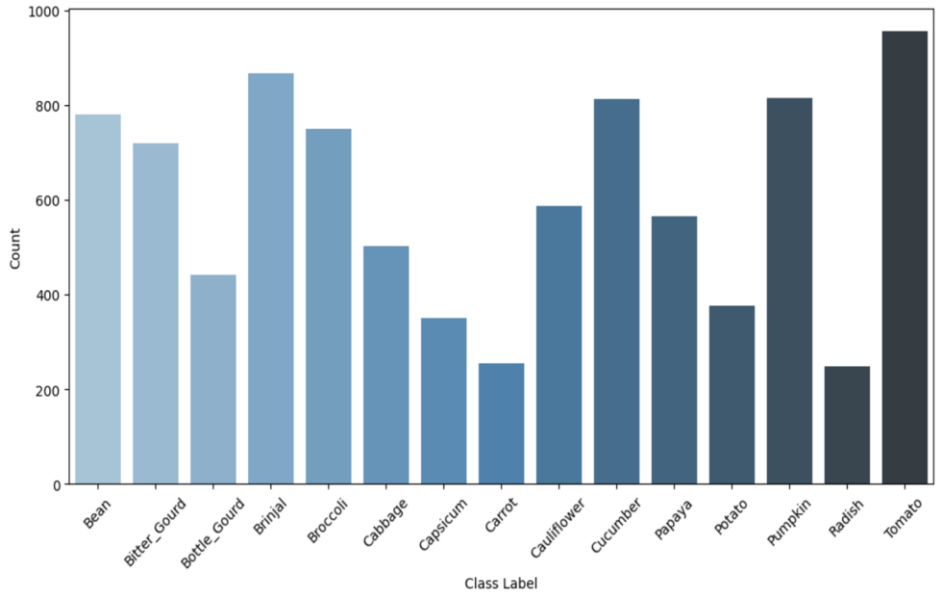
Class Distribution

- Look at the distribution of images for each class in each dataset

```python
def get_label_counts(image_categories, path):
    # Returns classes and their counts
    label_counts={}

    for class_name in image_categories:
        image_path = f"{path}/{class_name}"
        images_inFolder = os.listdir(image_path)

        label_counts[class_name] = len(images_inFolder)
    return label_counts
```

```python
# Train set
train_label_counts = get_label_counts(image_categories, train_path)

print("No of Labels in Train Set:")
for class_name, count in train_label_counts.items():
    print(f"{class_name}: {count}")
```

```
No of Labels in Train Set:
Bean: 780
Bitter_Gourd: 720
Bottle_Gourd: 441
Brinjal: 868
Broccoli: 750
Cabbage: 503
Capsicum: 351
Carrot: 256
Cauliflower: 587
Cucumber: 812
Papaya: 566
Potato: 377
Pumpkin: 814
Radish: 248
Tomato: 955
```



Step 1: EDA

```
No of Labels in Validation Set:
Bean: 200
Bitter_Gourd: 200
Bottle_Gourd: 200
Brinjal: 200
Broccoli: 200
Cabbage: 200
Capsicum: 200
Carrot: 200
Cauliflower: 200
Cucumber: 200
Papaya: 200
Potato: 200
Pumpkin: 200
Radish: 200
Tomato: 200
```

```
No of Labels in Test Set:
Bean: 200
Bitter_Gourd: 200
Bottle_Gourd: 200
Brinjal: 200
Broccoli: 200
Cabbage: 200
Capsicum: 200
Carrot: 200
Cauliflower: 200
Cucumber: 200
Papaya: 200
Potato: 200
Pumpkin: 200
Radish: 200
Tomato: 200
```

Observations

- There are unbalanced images in training set
- Equal number of images in validation and testing set

Step 1: EDA

Image Pixel Distribution

- Helps understand how pixel values are distributed across the images

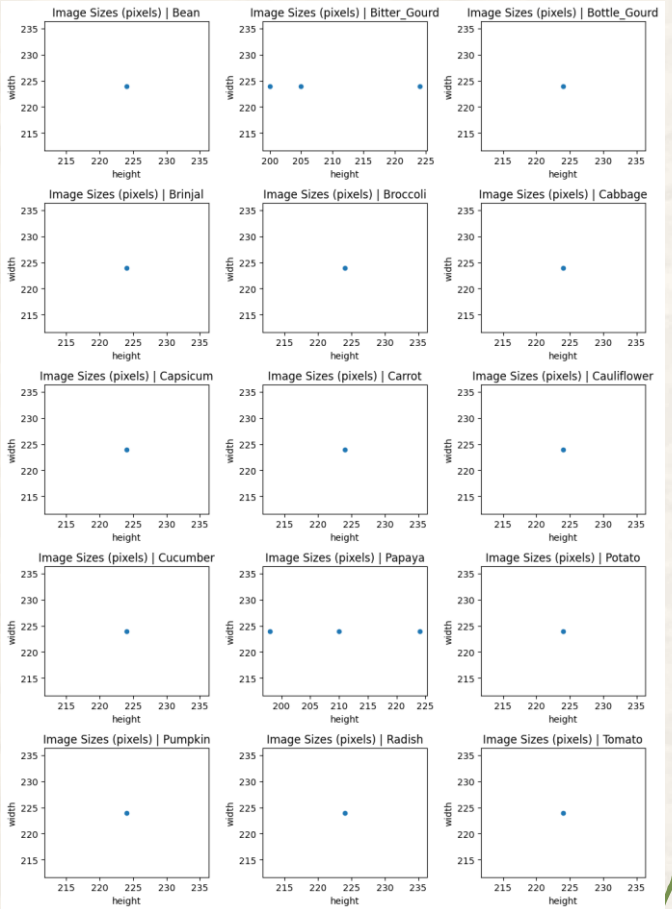
```python
# Function to retrieve the dimensions of the images
def get_pixels(file):
    im = Image.open(file)
    arr = np.array(im)
    h,w,d = arr.shape
    return h, w, d
```

Pytho

```python
fig, axes = plt.subplots(5, 3, figsize=(10, 14))
axes = axes.ravel()

for i, class_name in enumerate(image_categories):
    image_path = f"{train_path}/{class_name}"
    fileList = [f"{image_path}/{file}" for file in os.listdir(image_path)]

    # Retrieve the dimensions of the images
    dims = [get_pixels(file) for file in fileList]
    dim_df = pd.DataFrame(dims, columns=["height", "width", "depth"])

    # Get the unique dimension values for the current class
    unique_dims = dim_df[["height","width"]].drop_duplicates()
    unique_dims_str = [f"{height}x{width}" for height, width in unique_dims.values]
    print(f"Unique Dimensions for {class_name}:")
    string = "\n".join(unique_dims_str)
    print(string)
    print()

    # Plot the scatterplot to demonstrate the image size
    ax = axes[i]
    sizes = dim_df.groupby(["height","width"]).size().reset_index().rename(columns={0:"count"})
    sizes.plot.scatter(x="height", y="width", ax=ax)
    ax.set_title(f"Image Sizes (pixels) | {class_name}")
    plt.tight_layout()
plt.show()
```

Observation

- Only some images from bitter gourd and papaya has other pixel dimensions.
- The other classes only have images that are 224x224.



Step 1: EDA

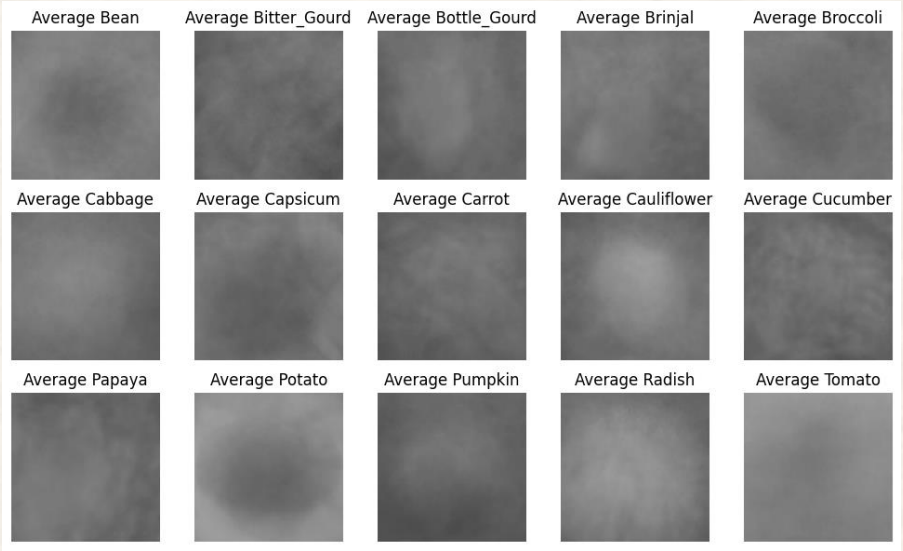
Image Averaging

- Calculates the average image for each class within a set of image categories
- Helps in understand the typical appearance of images in each class

```python
# Making (n x m) images
def img2np(path, list_of_filename, size=(64, 64)):
    # Store image arrays
    img_list = []

    for fn in list_of_filename:
        filepath = f"{path}/{fn}"
        # Load images
        current_img = tf.keras.utils.load_img(filepath, target_size=size, color_mode="grayscale")
        img_arr = tf.keras.utils.img_to_array(current_img)
        img_list.append(img_arr)

    # Convert to numPy array
    full_mat = np.array(img_list)
    return full_mat
```

Python

```python
def calculate_avg_img(image_categories, train_dir):
    average_images={}
    for cat in image_categories:
        image_path = f"{train_dir}/{cat}"
        # Retrieve list of images in each class folder
        images_inFolder = os.listdir(image_path)

        # Get the list of images for the current class
        class_images = img2np(image_path, images_inFolder)

        # Get the average image by taking the mean along axis 0
        average_image = np.mean(class_images, axis=0)
        average_images[cat] = average_image

    return average_images
```



Observation

- The average image of each class label is not very clear to view and predict what each image portrays
- Caused by the different orientation of each image, as well as the background of the main object, hence model might have a hard time trying to find features to map to the different images.

Step 1: EDA

Extract Image (Grayscale: 31x31)

- Retrieval of training images of certain dimensions and channel

```
# Loading data
def load_data(directory, img_height, img_width, colormode, batch_size, seed):
    data = tf.keras.preprocessing.image_dataset_from_directory(
        directory=directory,
        image_size=(img_height, img_width),
        color_mode=colormode,
        batch_size=batch_size,
        seed = seed
    )
    return data
```

```
# Load train data
train_data_31 = load_data(train_path, 31, 31, "grayscale", 10, 42)
```

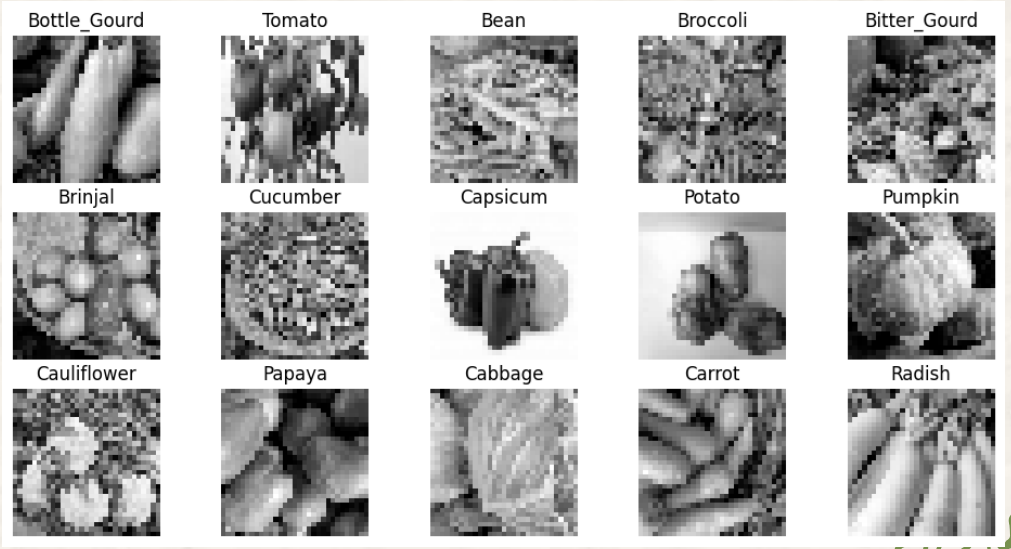
```
Found 9028 files belonging to 15 classes.
```

```
# Load validation data
val_data_31 = load_data(val_path, 31, 31, "grayscale", 10, 42)
```

```
Found 3000 files belonging to 15 classes.
```

```
# Load test data
test_data_31 = load_data(test_path, 31, 31, "grayscale", 10, 42)
```

```
Found 3000 files belonging to 15 classes.
```



Step 1: EDA

Extract Image (Grayscale: 128x128)

- Retrieval of training images of certain dimensions and channel

```
# Load train data
train_data_128 = load_data(train_path, 128, 128, "grayscale", 10, 42)
```

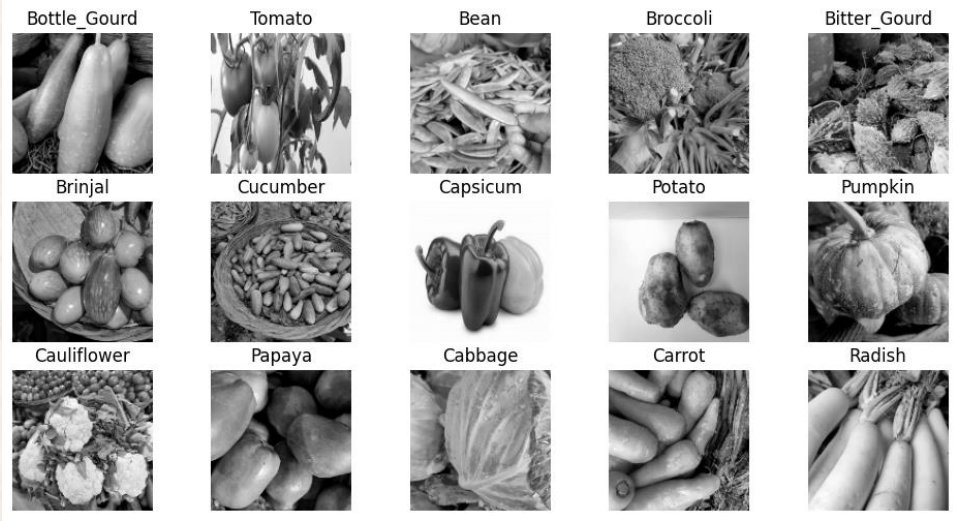
```
Found 9028 files belonging to 15 classes.
```

```
# Load validation data
val_data_128 = load_data(val_path, 128, 128, "grayscale", 10, 42)
```

```
Found 3000 files belonging to 15 classes.
```

```
# Load test data
test_data_128 = load_data(test_path, 128, 128, "grayscale", 10, 42)
```

```
Found 3000 files belonging to 15 classes.
```



Step 2: Feature Engineering

Normalisation

- Normalise inputs from 0 to 1, leading to faster convergence using the Rescaling layer
- Helps optimize the algorithm to better converge during gradient descent

```
# Normalise the data within the range of 0 and 1
normalised_data = Sequential(
    name = "normalised_data",
    layers = [
        Rescaling(1./255),  # Normalised values within the range of 0 and 1
    ]
)
```

This will be added to the first layer of every model.

Loss Function - Sparse Categorical Cross Entropy (~~One Hot Encoding~~)

- For multi-class classification, categorical cross entropy is used.
- Instead of one hot encoding, sparse categorical crossentropy is used as the labels are provided as integers:
  - For one hot encoding, each class label is represented as a binary vector with a 1 at the index corresponding to the class and 0s elsewhere. Many classes lead to large, sparse vectors. Example, 100 classes mean 99 zeros and only one 1.
  - For sparse categorical cross entropy, they can handle classification tasks with integer labels directly. Takes the model's predicted probability distribution over classes.

Step 2: Feature Engineering

Data Augmentation

- To address the problem of class imbalance, I will augmented the current existing data to generate more training data such that every class will contain the same number of labels.
- Need to reload the train and test data again in batch size of None so all images will get accounted for.

```python
# Reload train data
train_data_31V2 = load_data(train_path, 31, 31, "grayscale", None, 42)

train_data_128V2 = load_data(train_path, 128, 128, "grayscale", None, 42)
```

Python

```
Found 9028 files belonging to 15 classes.
Found 9028 files belonging to 15 classes.
```

```python
# Create a list of images and labels
data_split = [(img.numpy(), classes[labels]) for img, labels in train_data_31V2]

# Convert the list to numpy arrays
X_train_31 = np.array([img for img, label in data_split])
y_train_31 = np.array([label for img, label in data_split])
print(X_train_31.shape)
print(y_train_31.shape)
```

Python

```
(9028, 31, 31, 1)
(9028,)
```

```python
# Create a list of images and labels
data_split = [(img.numpy(), classes[labels]) for img, labels in train_data_128V2]

# Convert the list to numpy arrays
X_train_128 = np.array([img for img, label in data_split])
y_train_128 = np.array([label for img, label in data_split])
print(X_train_128.shape)
print(y_train_128.shape)
```

Python

```
(9028, 128, 128, 1)
(9028,)
```

```python
# Split the data by classes
def split_data_by_classes(X_train, y_train):
    class_data = {}
    for class_label in classes:
        class_indices = np.where(y_train == class_label)[0]
        class_data[class_label] = X_train[class_indices]
    return class_data

# Determine the class max size
def get_max_class_size(class_data):
    max_class_size = max(len(class_images) for class_images in class_data.values())
    print(f"Max class size: {max_class_size}")
    return max_class_size

# Upsample the data to resolve the issue of class imbalance
def upsample_class_with_augmentation(class_data, datagen):
    augmented_images = []
    augmented_labels = []
    max_class_size = get_max_class_size(class_data)
    for class_label, class_images in class_data.items():

        # Set seed for reproducibility to fit the data generator to the images
        datagen.fit(class_images,augment=True, seed=42)

        # Generate augmented images and labels until reaching maximum class size
        generator =datagen.flow(
            class_images,
            y=np.full((len(class_images),),class_label),
            seed = 42,
            batch_size=len(class_images))
        batch_images, batch_labels = generator.next()

        # Repeat data until reaching maximum class size
        while len(batch_images) < max_class_size:
            additional_images, additional_labels = generator.next()
            batch_images = np.concatenate([batch_images, additional_images])
            batch_labels = np.concatenate([batch_labels, additional_labels])

        # Append to lists until max class size is reached
        augmented_images.append(batch_images[:max_class_size])
        augmented_labels.append(batch_labels[:max_class_size])

    # Combine the augmented data for all classes
    augmented_images = np.concatenate(augmented_images)
    augmented_labels = np.concatenate(augmented_labels)

    # Shuffle the data
    shuffle_indices = np.arange(len(augmented_labels))
    np.random.seed(42)
    np.random.shuffle(shuffle_indices)

    augmented_images = augmented_images[shuffle_indices]
    augmented_labels = augmented_labels[shuffle_indices]

    return augmented_images, augmented_labels
```

Balancing of classes with the use of **upsampling from data augmentation**

Step 2: Data Augmentation

Solve Class Imbalance - 128 x 128 images

- Types of Augmentation done for 128 x 128 images:
  - **Rotation**: Allow image to be rotated in the range of -20 to +20 degrees to help the model become invariant to the orientation of objects in the images. Avoid excessive distortion while model learn from images
  - **Zoom**: Allow image to be zoomed in or out by up to 20% to allow for moderate zooming without distorting the images too much
  - **Shear**: Shifting one part of the image of a maximum magnitude of 20%, keeping the rest stationary as it introduces deformations to the images, helping the model become more robust to different shapes
  - **Horizontal Flip**: Images may be horizontally flipped with a 50% probability to increase the variability in the dataset.
  - **Fill Mode - Reflect**: Smoothly fill in new pixels created during rotations or other transformations, prevent the introduction of sharp edges or artificial patterns

```
# Augment the images (128 x 128)
datagen = ImageDataGenerator(
    rotation_range=20, # rotation
    zoom_range=0.2, # zoom
    shear_range=0.2, # shear
    horizontal_flip=True, # horizontal flip
    fill_mode = "reflect" # fill mode
)

class_data_128 = split_data_by_classes(X_train_128, y_train_128)
augmented_images_128, augmented_labels_128 = upsample_class_with_augmentation(class_data_128, datagen)
```

```
Max class size: 955
```

```
# Check number of samples in each class
for class_label, class_images in class_data_128.items():
    num_samples_before_augmentation = len(class_images)
    num_samples_after_augmentation = len(augmented_labels_128[augmented_labels_128 == class_label])
    print(f"Class {class_label}: {num_samples_after_augmentation} samples after augmentation (originally {num_samples_before_augmentation} samples)")
```

```
Class Bean: 955 samples after augmentation (originally 780 samples)
Class Bitter_Gourd: 955 samples after augmentation (originally 720 samples)
Class Bottle_Gourd: 955 samples after augmentation (originally 441 samples)
Class Brinjal: 955 samples after augmentation (originally 868 samples)
Class Broccoli: 955 samples after augmentation (originally 750 samples)
Class Cabbage: 955 samples after augmentation (originally 503 samples)
Class Capsicum: 955 samples after augmentation (originally 351 samples)
Class Carrot: 955 samples after augmentation (originally 256 samples)
Class Cauliflower: 955 samples after augmentation (originally 587 samples)
Class Cucumber: 955 samples after augmentation (originally 812 samples)
Class Papaya: 955 samples after augmentation (originally 566 samples)
Class Potato: 955 samples after augmentation (originally 377 samples)
Class Pumpkin: 955 samples after augmentation (originally 814 samples)
Class Radish: 955 samples after augmentation (originally 248 samples)
Class Tomato: 955 samples after augmentation (originally 955 samples)
```

Solve Class Imbalance - 31 x 31 images

- **Rotation**: Allow image to be rotated in the range of -10 to +10 degrees to help the model become invariant to the orientation of objects in the images. Avoid excessive distortion while model learn from images
- **Shear**: Shifting one part of the image of a maximum magnitude of 10%, keeping the rest stationary as it introduces deformations to the images, helping the model become more robust to different shapes
- **Horizontal Flip**: Images may be horizontally flipped with a 50% probability to increase the variability in the dataset.
- **Fill Mode - Reflect**: Smoothly fill in new pixels created during rotations or other transformations, prevent the introduction of sharp edges or artificial patterns

```
# Augment the images (31 x 31)
datagen31 = ImageDataGenerator(
    rotation_range=10, # rotation
    shear_range=0.1, # shear
    horizontal_flip=True, # horizontal flip
    fill_mode = "reflect" # fill mode
)

class_data_31 = split_data_by_classes(X_train_31, y_train_31)
augmented_images_31, augmented_labels_31 = upsample_class_with_augmentation(class_data_31, datagen31)
```

```
Max class size: 955
```

```
# Check number of samples in each class
for class_label, class_images in class_data_31.items():
    num_samples_before_augmentation = len(class_images)
    num_samples_after_augmentation = len(augmented_labels_31[augmented_labels_31 == class_label])
    print(f"Class {class_label}: {num_samples_after_augmentation} samples after augmentation (originally {num_samples_before_augmentation} samples)")
```

```
Class Bean: 955 samples after augmentation (originally 780 samples)
Class Bitter_Gourd: 955 samples after augmentation (originally 720 samples)
Class Bottle_Gourd: 955 samples after augmentation (originally 441 samples)
Class Brinjal: 955 samples after augmentation (originally 868 samples)
Class Broccoli: 955 samples after augmentation (originally 750 samples)
Class Cabbage: 955 samples after augmentation (originally 503 samples)
Class Capsicum: 955 samples after augmentation (originally 351 samples)
Class Carrot: 955 samples after augmentation (originally 256 samples)
Class Cauliflower: 955 samples after augmentation (originally 587 samples)
Class Cucumber: 955 samples after augmentation (originally 812 samples)
Class Papaya: 955 samples after augmentation (originally 566 samples)
Class Potato: 955 samples after augmentation (originally 377 samples)
Class Pumpkin: 955 samples after augmentation (originally 814 samples)
Class Radish: 955 samples after augmentation (originally 248 samples)
Class Tomato: 955 samples after augmentation (originally 955 samples)
```

Step 3: Model Selection

3) Model Selection & Model Evaluation

Model Building

Attempted **transfer learning** too but was **not included** as **one of the final models** I used

Model List (128 x 128 images):

1. Custom Conv2D Neural Network Model

Model List (31 x 31 images):

1. Custom Conv2D Neural Network Model

```
# Load 128 x 128 data as None to consider all images
train_data_128 = load_data(train_path, 128, 128, "grayscale", None, 42)
val_data_128 = load_data(val_path, 128, 128, "grayscale", None, 42)
test_data_128 = load_data(test_path, 128, 128, "grayscale", None, 42)
```

[38]

```
Found 9028 files belonging to 15 classes.
Found 3000 files belonging to 15 classes.
Found 3000 files belonging to 15 classes.
```

```
# Load 31 x 31 images as batch size None to consider all images
train_data_31 = load_data(train_path, 31, 31, "grayscale", None, 42)
val_data_31 = load_data(val_path, 31, 31, "grayscale", None, 42)
test_data_31 = load_data(test_path, 31, 31, "grayscale", None, 42)
```

[39]

```
Found 9028 files belonging to 15 classes.
Found 3000 files belonging to 15 classes.
Found 3000 files belonging to 15 classes.
```

Conv2D Neural Network Model

- A CNN architecture is well built for image processing as they can effectively extract features from images and learn to recognise patterns, making it suitable for tasks such as object detection, image segmentation and classification.
- Key Layers:
  - Convolutional Layers: Involves sliding a small filter (kernel) over the input to detect features. Conv layers capture local patterns and hierarchical features in the data.
  - Pooling Layers: Downsample the feature maps generated by the convolutional layers. Reduces the spatial dimensions of the data, making the model more robust to variations in position and scale. (Examples: Max-pooling and Average-pooling)
  - Fully Connected Layers: One or more fully connected layers are used to perform high-level reasoning and classification. Layers connect every neuron to every neuron in the previous and subsequent layers, allowing the model to learn complex patterns.
  - Activation Function: Introduces non-linearity into the model. Common functions include: ReLU (Rectified Linear Unit), sigmoid and tanh.
  - Dropout: Regularization technique that randomly drops a fraction of neurons during training to prevent overfitting.
  - Flatten Layer: Used to reshape the data into a 1D vector before passing it to the fully connected layers.
  - Output Layer: Has as many neurons as there are classes. Activation function depends on the problem type. Example, Softmax is commonly used for multi-class classification.
  - Loss Function: Choice of loss function depends on the task. For classification, cross-entropy loss is often used. For regression, mean squared error is commonly used.
- Padding - addition of pixels to the edge of the image
  - Pixels on the edge of the input are only ever exposed to the edge of the filter hence padding can be used to resolved this issue.
  - Padding of value 'same' calculates and adds the padding required to the input image to ensure that the output has the same shape as the input.
- Stride - Amount of movement between applications of the filter to the input image

Step 3: Model Selection

Training without Data Augumentation

- We will start by fitting the unaugumented data (imbalanced class) to the model as training.

128 x 128 images

Baseline Model (128 x 128)

- To start off, this is my baseline model which I will use to compare the loss and accuracy with the other models.

Baseline Model
(128 x 128)

```
# Try for 128 x 128 images
tf.keras.backend.clear_session()

Conv2D_128_Baseline = Sequential(
    name = "Conv2D_128_Baseline",
    layers = [
        normalised_data,
        Conv2D(32, (3, 3),input_shape=(128,128,1), activation='relu', padding='same', strides = (4,4)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu', padding='same', strides=(2,2)),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu', padding='same', strides=(1,1)),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu', padding='same', strides=(1,1)),
        MaxPooling2D((2, 2)),

        Flatten(),

        Dense(128, activation='relu'),
        Dropout(0.6),

        Dense(num_classes, activation='softmax')
    ]
)

# Compile model
opt = Adam(learning_rate=0.0009)
Conv2D_128_Baseline.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Conv2D_128_Baseline.build(input_shape=(None, 128, 128, 1))

Conv2D_128_Baseline_history = Conv2D_128_Baseline.fit(
    train_data_128.batch(10),
    epochs=100,
    validation_data=val_data_128.batch(10)
    )
```

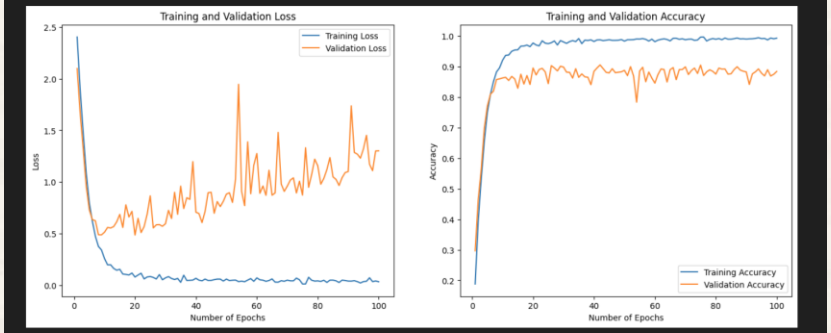
```
Model: "Conv2D_128_Baseline"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalised_data (Sequential  (None, None, None, 1)    0
 )

 conv2d (Conv2D)             (None, 32, 32, 32)        320

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 8, 8, 64)          18496

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 128)         73856

 max_pooling2d_2 (MaxPooling  (None, 2, 2, 128)        0
 2D)

 conv2d_3 (Conv2D)           (None, 2, 2, 128)         147584

 max_pooling2d_3 (MaxPooling  (None, 1, 1, 128)        0
 2D)

 flatten (Flatten)           (None, 128)               0

 dense (Dense)               (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 15)                1935

=================================================================
Total params: 258,703
Trainable params: 258,703
Non-trainable params: 0
_________________________________________________________________
```

Step 3: Model Evaluation

```python
# Plot learning_curve
def plot_learning_curve(history):
    history_df = pd.DataFrame(history)
    epochs = list(range(1,len(history_df)+1))

    fig, ax = plt.subplots(1,2, figsize=(16,6))

    # Training loss and validation loss
    ax1=ax[0]
    ax1.plot(epochs, history_df["loss"], label="Training Loss")
    ax1.plot(epochs, history_df["val_loss"], label="Validation Loss")
    ax1.legend()
    ax1.set_ylabel("Loss")
    ax1.set_xlabel("Number of Epochs")
    ax1.set_title("Training and Validation Loss")

    # Training accuracy and validation accuracy
    ax2=ax[1]
    ax2.plot(epochs, history_df["accuracy"], label="Training Accuracy")
    ax2.plot(epochs, history_df["val_accuracy"], label="Validation Accuracy")
    ax2.legend()
    ax2.set_ylabel("Accuracy")
    ax2.set_xlabel("Number of Epochs")
    ax2.set_title("Training and Validation Accuracy")
    plt.show()
```

```python
plot_learning_curve(Conv2D_128_Baseline_history.history)
```

Python



+ Code + Markdown

Observations

- From the loss curve, as the epochs increased, the loss function generally starts to increase.
- From the accuracy curve, as the epochs increased, the accuracy is fluctuating around 0.80+.
- Both graphs imply that the learning rate might be too high, causing the model to overshoot the minimum of the loss function due to the major fluctuations present in both curves.

```python
Conv2D_128_Baseline.evaluate(test_data_128.batch(10))
```

```
300/300 [==============================] - 1s 3ms/step - loss: 1.1186 - accuracy: 0.8940

[1.1186254024505615, 0.8939999938011169]
```

Baseline model is **terrible**

Step 3: Model Selection

CNN Version 3 (128 x 128)

- Next attempt to build a model by increasing the kernel size, the first fully connected layer and the Dropout layer (Introduces noise during training to help netowork be more robust).
- This can allow the convolutional layer to capture more global information in the input and hopefully help the accuracy and loss to converge towards the training model.

```
# Try for 128 x 128 images
tf.keras.backend.clear_session()

Conv2D_128_V3 = Sequential(
    name = "Conv2D_128_V3",
    layers = [
        normalised_data,
        Conv2D(64, (7, 7),input_shape=(128,128,1), activation = "relu", padding='same', strides = (4,4)),
        MaxPooling2D((2, 2)),
        Conv2D(128, (5, 5), activation = "relu", padding='same', strides = (2, 2)),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3,3), activation = "relu", padding='same', strides = (1, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(512, (3,3), activation = "relu", padding='same', strides = (1, 1)),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dropout(0.5),

        Dense(1028, activation='relu'),
        Dropout(0.6),
        Dense(128, activation='relu'),
        Dropout(0.6),

        Dense(num_classes, activation='softmax')
    ]
)

# Compile model
opt = Adam(learning_rate=0.00001)
Conv2D_128_V3.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Conv2D_128_V3.build(input_shape=(None, 128, 128, 1))

Conv2D_128_V3_history = Conv2D_128_V3.fit(
    train_data_128.batch(10),
    epochs=300,
    validation_data=val_data_128.batch(10)
    )
```

(128 x 128)
Unaugmented Images

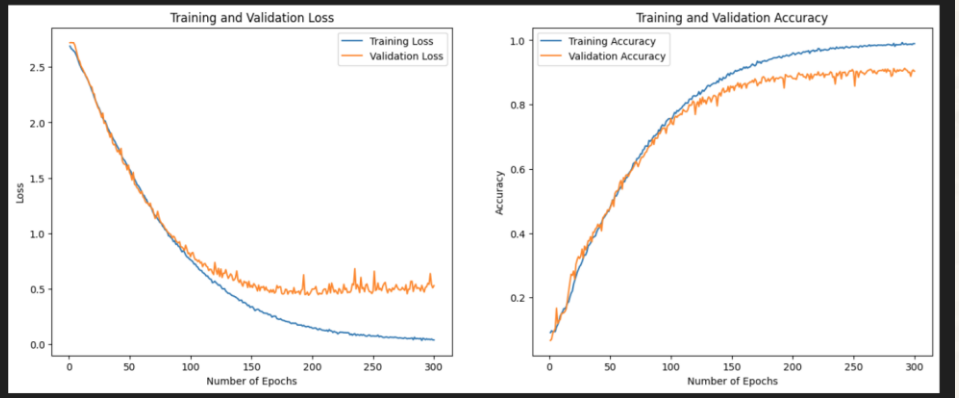
Increases **filters** and **kernel size**, added mores **strides**, increased **Dense layers units** & **Dropout** layers

```
Conv2D_128_V3.summary()
```

```
Model: "Conv2D_128_V3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalised_data (Sequential  (None, None, None, 1)    0
 )

 conv2d (Conv2D)             (None, 32, 32, 64)        3200

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 8, 8, 128)         204928

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 256)         295168

 max_pooling2d_2 (MaxPooling  (None, 2, 2, 256)        0
 2D)

 conv2d_3 (Conv2D)           (None, 2, 2, 512)         1180160

 max_pooling2d_3 (MaxPooling  (None, 1, 1, 512)        0
 2D)

 flatten (Flatten)           (None, 512)               0

 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 1028)              527364

 dropout_1 (Dropout)         (None, 1028)              0

 dense_1 (Dense)             (None, 128)               131712

 dropout_2 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 15)                1935

=================================================================
Total params: 2,344,467
Trainable params: 2,344,467
Non-trainable params: 0
_________________________________________________________________
```

Step 3: Model Evaluation

```
plot_learning_curve(Conv2D_128_V3_history.history)
```



Observations

- For the training and validation loss curve, both were converging quite well with few to none fluctuations and the validation loss curve converge around 0.60+.
- For the training and validation accuracy curve, both were increasing quite well at a steady rate and validation accuracy curve converge around 0.9+.
- Generally, this architecture is well built but I will improve it to see how much better it can do.

Model is **decent**

```
Conv2D_128_V3.evaluate(test_data_128.batch(10))
```

```
300/300 [==============================] - 1s 3ms/step - loss: 0.5101 - accuracy: 0.9020

[0.5100677013397217, 0.9020000100135803]
```

Observations

- Test accuracy at 0.9 is quite high but we can improve this score.

Step 3: Model Selection

31 x 31 images

CNN Version 1 (31 x 31)

- Since the image resolution is lower, I shall decrease the kernel size to capture more local field instead of the global field.
- I will also increase the fully connected layers units to allow the model to learn complex representations for a higher level of abstraction.

(31 x 31) Unaugmented Images

```
# Try for 31 x 31 images
tf.keras.backend.clear_session()

Conv2D_31 = Sequential(name="Conv2D_31V1",
    layers = [
        normalised_data,
        Conv2D(32, (3, 3),input_shape=(31, 31 ,1), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dropout(0.5),

        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(256, activation='relu'),
        Dropout(0.5),

        Dense(num_classes, activation='softmax')
    ]
)

# Compile model
opt = Adam(learning_rate=0.00001)
Conv2D_31.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Conv2D_31.build(input_shape=(None, 31, 31, 1))

Conv2D_31_history = Conv2D_31.fit(
    train_data_31.batch(10),
    epochs=200,
    validation_data=val_data_31.batch(10)
    )
```

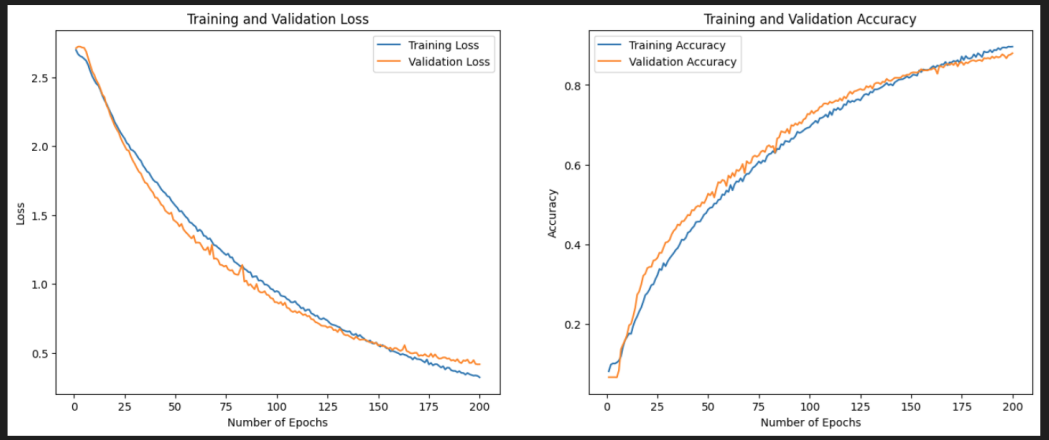
```
Conv2D_31.summary()
```

```
Model: "Conv2D_31V1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalised_data (Sequential  (None, None, None, 1)    0
 )

 conv2d (Conv2D)             (None, 31, 31, 32)        320

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 15, 15, 64)        18496

 conv2d_2 (Conv2D)           (None, 15, 15, 128)       73856

 conv2d_3 (Conv2D)           (None, 15, 15, 128)       147584

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 flatten (Flatten)           (None, 6272)              0

 dropout (Dropout)           (None, 6272)              0

 dense (Dense)               (None, 512)               3211776

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_2 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 256)               65792

 dropout_3 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 15)                3855

=================================================================
Total params: 3,653,007
Trainable params: 3,653,007
Non-trainable params: 0
_________________________________________________________________
```

Step 3: Model Evaluation

```
plot_learning_curve(Conv2D_31_history.history)
```

Python



Observations

- From the training and validation loss curve, we can see the validation curve converges quite well around 0.3+ but maybe changing the learning rate and adding regularisation can improve this.
- The validation accuracy is quite high, reaching 0.94, but I think we can improve this model.

This model is quite **accurate**

```
Conv2D_31.evaluate(test_data_31.batch(10))
```

Python

```
300/300 [==============================] - 1s 2ms/step - loss: 0.3666 - accuracy: 0.8957

[0.3666374981403351, 0.8956666588783264]
```

Step 3: Model Selection

Training with Data Augumentation

- Using the same network architecture as before, we shall see if the augmentation of data being upsampled into the existing training data to see if augmentating the data and balancing the data helps the model

128 x 128 images

CNN Augmented Balance Version 1 (128 x 128)

(128x128) Balanced Augmented Images

```
# Try for 128 x 128 images
tf.keras.backend.clear_session()

Conv2D_128V1_aug = Sequential(name="Conv2D_128V1_Augmentation",
    layers = [
        normalised_data,
        Conv2D(64, (5, 5), activation='relu', padding='same', input_shape=(128, 128, 1), strides=(4, 4)),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu', padding='same', strides=(1, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu', padding='same', strides=(1, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu', padding='same', strides=(1, 1)),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dropout(0.5),

        Dense(1024, activation='relu'),
        Dropout(0.5),
        Dense(128, activation = 'relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ]
)

# Compile model
opt = Adam(learning_rate=0.00001)
Conv2D_128V1_aug.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Conv2D_128V1_aug.build(input_shape=(None, 128, 128, 1))

Conv2D_128V1_aug_history = Conv2D_128V1_aug.fit(
    train_128V2.batch(10),
    epochs=200,
    validation_data=val_data_128.batch(10)
    )
```

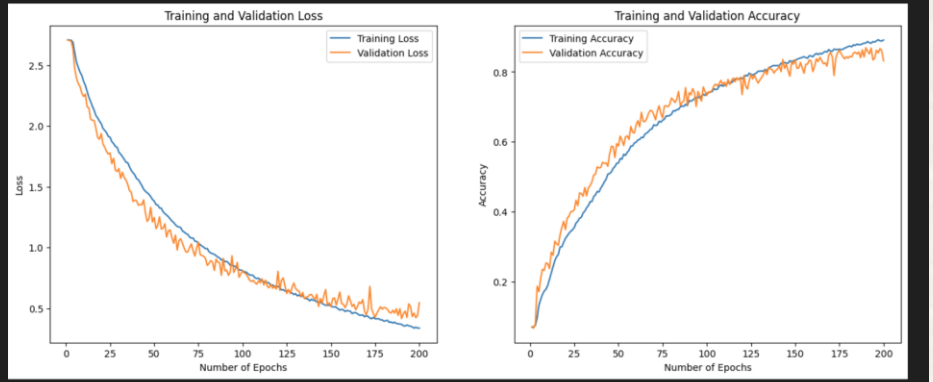
```
Conv2D_128V1_aug.summary()
```

```
Model: "Conv2D_128V1_Augmentation"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalised_data (Sequential  (None, None, None, 1)    0
 )

 conv2d (Conv2D)             (None, 32, 32, 64)        1664

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 16, 16, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 conv2d_3 (Conv2D)           (None, 4, 4, 256)         295168

 max_pooling2d_3 (MaxPooling  (None, 2, 2, 256)        0
 2D)

 flatten (Flatten)           (None, 1024)              0

 dropout (Dropout)           (None, 1024)              0

 dense (Dense)               (None, 1024)              1049600

 dropout_1 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 128)               131200

 dropout_2 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 15)                1935

=================================================================
Total params: 1,701,007
Trainable params: 1,701,007
Non-trainable params: 0
_________________________________________________________________
```

Step 3: Model Evaluation

```
plot_learning_curve(Conv2D_128V1_aug_history.history)
```

Python



Observations

- This model still has room for improvement as the training accuracy only reaches 0.89 hence maybe an increase in epochs or a decrease in learning rate will increase the accuracy.
- Validation loss and accuracy is quite close towards the training loss and accuracy which shows that the model is quite reliable.
- Compared to the unaugmented data, the accuracy curve and loss curve of the validation data does not diverge away from the training curve that much, which shows that augmenting the data does improve the prediction of the model.

```
Conv2D_128V1_aug.evaluate(test_data_128.batch(10))
```

```
300/300 [==============================] - 3s 7ms/step - loss: 0.5392 - accuracy: 0.8360

[0.5391502976417542, 0.8360000252723694]
```

+ Code + Markdown

Observations

- Test accuracy of 0.80+ is relatively quite high.

This model using **augmented** and **balanced** images is quite **accurate, better than unaugmented**

Step 3: Model Selection

CNN Augmented Balance Version 2 (31 x 31)

- Increased units of Conv2D and kernel size

```
# Try for 31 x 31 images
tf.keras.backend.clear_session()

Conv2D_31_aug_V2 = Sequential(name="Conv2D_31_Augmentation_V2",
    layers = [
        normalised_data,
        Conv2D(64, (5, 5),input_shape=(31, 31 ,1), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        Conv2D(256, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dropout(0.6),

        Dense(128, activation='relu'),
        Dropout(0.6),

        Dense(num_classes, activation='softmax')
    ]
)

# Compile model
opt = Adam(learning_rate=0.0001)
Conv2D_31_aug_V2.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Conv2D_31_aug_V2.build(input_shape=(None, 31, 31, 1))

Conv2D_31_aug_V2_history = Conv2D_31_aug_V2.fit(
    train_31V2.batch(10),
    epochs=200,
    validation_data=val_data_31.batch(10)
    )
```

(31x31) Balanced Augmented Images

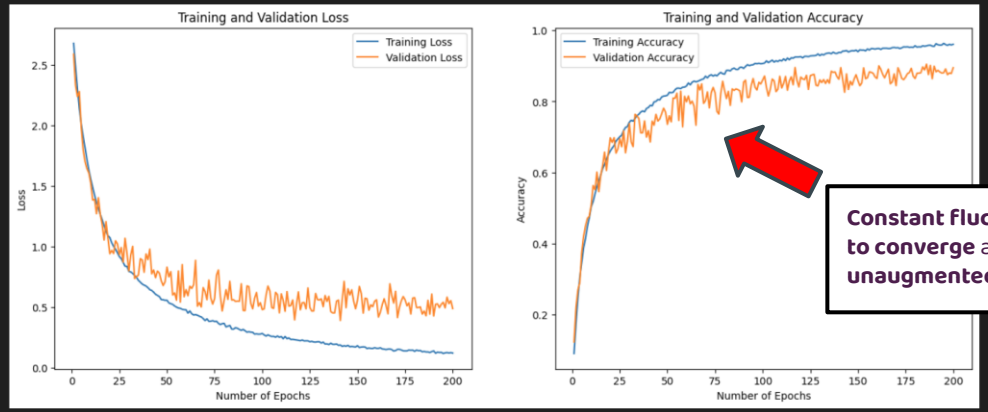
```
Conv2D_31_aug_V2.summary()
```

```
Model: "Conv2D_31_Augmentation_V2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalised_data (Sequential  (None, 31, 31, 1)        0
 )

 conv2d (Conv2D)             (None, 31, 31, 64)        1664

 max_pooling2d (MaxPooling2D  (None, 15, 15, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 15, 15, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 256)         295168

 max_pooling2d_2 (MaxPooling  (None, 3, 3, 256)        0
 2D)

 flatten (Flatten)           (None, 2304)              0

 dropout (Dropout)           (None, 2304)              0

 dense (Dense)               (None, 128)               295040

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 15)                1935

=================================================================
Total params: 667,663
Trainable params: 667,663
Non-trainable params: 0
_________________________________________________________________
```

Step 3: Model Evaluation

```
plot_learning_curve(Conv2D_31_aug_V2_history.history)
```

Python



Constant fluctuations, unable to converge as well as the **unaugmented images**

Observations

- The loss curve shows constant fluctuations in the validation loss which is generally higher than the training loss.
- The accuracy curve also shows constant fluctuations in the validation accuracy which is generally lower than the training accuracy.

```
Conv2D_31_aug_V2.evaluate(test_data_31.batch(10))
```

Python

```
300/300 [==============================] - 1s 3ms/step - loss: 0.4920 - accuracy: 0.8943

[0.49203088879585266, 0.8943333625793457]
```

+ Code + Markdown

Observations

- The accuracy is slightly better than the model before but still not as good as the unaugmented one.

This model using **augmented** and **balanced** images is **less accurate** than the one with **unaugmented images**

Step 4: Model Improvement

128 x 128 images

Time-based Decay

- Used to adjust the learning rate during training by decreasing it overtime
- Starts with a relatively large learning rate and gradually reduce it as training progress.
- Beneficial for optimization and help the model to converge more efficiently
- We will make use of learning rate scheduler to make this work

$$\text{learning\_rate} = \text{learning\_rate} \times \left(1 + \frac{k \cdot \text{initial learning\_rate}}{\text{epochs} + 1}\right)^{-1}$$

Time based Decay

```
def time_based_decay(epoch, lr):
    initial_lr = 0.00025
    #declare number of steps
    k = 0.1
    #Avoid division error
    decay = initial_lr /(epoch+1)
    lr *= (1. / (1. + k*decay * initial_lr))
    return lr

lr_schedule = LearningRateScheduler(time_based_decay, verbose=1)
```

```
Epoch 98: LearningRateScheduler setting learning rate to 0.000250000011858419.
Epoch 98/100
1433/1433 [==============================] - 8s 6ms/step - loss: 0.0465 - accuracy: 0.9902 - val_loss: 0.3100 - val_accuracy: 0.94

Epoch 99: LearningRateScheduler setting learning rate to 0.00025000001185858.
Epoch 99/100
1433/1433 [==============================] - 8s 6ms/step - loss: 0.0454 - accuracy: 0.9893 - val_loss: 0.5453 - val_accuracy: 0.92

Epoch 100: LearningRateScheduler setting learning rate to 0.00025000001185873785.
Epoch 100/100
1433/1433 [==============================] - 8s 6ms/step - loss: 0.0481 - accuracy: 0.9880 - val_loss: 0.3320 - val_accuracy: 0.93
```

Observations:

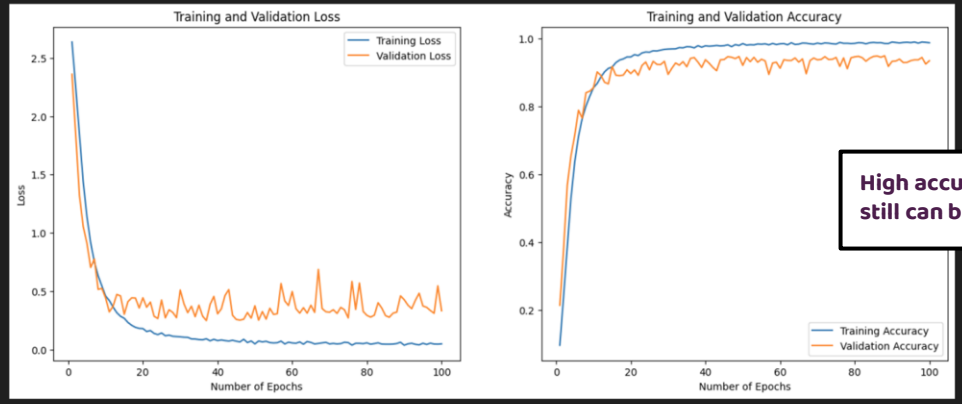
- Learning rate did not really change much but accuracy for validation is still relatively quite high, validation loss is also quite low

```
# Try for 128 x 128 images
tf.keras.backend.clear_session()

Conv2D_128V1_aug_lr = Sequential(name="Conv2D_128V1_Augment_LearningRate",
    layers = [
        normalised_data,
        Conv2D(64, (5, 5), activation='relu', padding='same', input_shape=(128, 128, 1), strides=(4, 4)),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu', padding='same', strides=(1, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu', padding='same', strides=(1, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu', padding='same', strides=(1, 1)),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dropout(0.5),

        Dense(1024, activation='relu'),
        Dropout(0.5),
        Dense(128, activation = 'relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ]
)

# Compile model
opt = Adam(learning_rate=0.00025)
Conv2D_128V1_aug_lr.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Conv2D_128V1_aug_lr.build(input_shape=(None, 128, 128, 1))

Conv2D_128V1_aug_lr_history = Conv2D_128V1_aug_lr.fit(
    train_128V2.batch(10),
    epochs=100,
    validation_data=val_data_128.batch(10),
    callbacks=[lr_schedule]
    )
```

Time-Based Decay (128 x 128)

```
plot_learning_curve(Conv2D_128V1_aug_lr_history.history)
```

Python



High accuracy but still can be improved

Observations:

- Model seems to generate quite a low validation loss and a high validation accuracy, showing that the model architecture was quite good.

```
Conv2D_128V1_aug_lr.evaluate(test_data_128.batch(10))
```

Python

```
300/300 [==============================] - 2s 6ms/step - loss: 0.3115 - accuracy: 0.9377

[0.31153255701065063, 0.937666654586792]
```

+ Code + Markdown

Observations:

- Test accuracy is quite high 0.94 which means this model architecture is quite good.

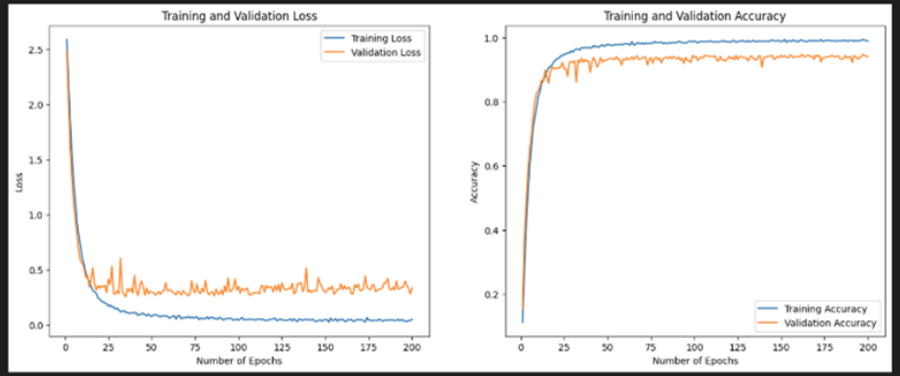
Time-Based Decay (31 x 31)

Time-based Decay

```
# Try for 31 x 31 images
tf.keras.backend.clear_session()

Conv2D_31_lr = Sequential(name="Conv2D_31V1",
    layers = [
        normalised_data,
        Conv2D(32, (3, 3),input_shape=(31, 31 ,1), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dropout(0.5),

        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(256, activation='relu'),
        Dropout(0.5),

        Dense(num_classes, activation='softmax')
    ]
)

# Compile model
opt = Adam(learning_rate=0.00025)
Conv2D_31_lr.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Conv2D_31_lr.build(input_shape=(None, 31, 31, 1))

Conv2D_31_lr_history = Conv2D_31_lr.fit(
    train_data_31.batch(10),
    epochs=200,
    validation_data=val_data_31.batch(10),
    callbacks=[lr_schedule]
    )
```

```
plot_learning_curve(Conv2D_31_lr_history.history)
```

Python



Observations

- Validation loss and accuracy are quite close to the training loss and accuracy, achieving close to 0.9+ accuracy, loss is also kept as low as 0.5+

```
Conv2D_31_lr.evaluate(test_data_31.batch(10))
```

Python

```
300/300 [==============================] - 2s 5ms/step - loss: 0.2863 - accuracy: 0.9433

[0.28634753823280334, 0.9433333277702332]
```

Observations

- Might have to decrease the steps for the learning rate to have a more drastic change
- Test accuracy is still 90+ which shows how accurate this model is.

Keras Tuner (Optimizer)

```python
def build_128_aug_model(hp):
    model = Sequential(
        layers = [
            normalised_data,
            Conv2D(64, (5, 5), activation='relu', padding='same', input_shape=(128, 128, 1), strides=(4, 4)),
            MaxPooling2D((2, 2)),
            Conv2D(128, (3, 3), activation='relu', padding='same', strides=(1, 1)),
            MaxPooling2D((2, 2)),
            Conv2D(128, (3, 3), activation='relu', padding='same', strides=(1, 1)),
            MaxPooling2D((2, 2)),
            Conv2D(256, (3, 3), activation='relu', padding='same', strides=(1, 1)),
            MaxPooling2D((2, 2)),

            Flatten(),
            Dropout(0.5),

            Dense(1024, activation='relu'),
            Dropout(0.5),
            Dense(128, activation = 'relu'),
            Dropout(0.5),
            Dense(num_classes, activation='softmax')
        ]
    )

    optimizer_name = hp.Choice('optimizer', values=['adam', 'sgd', 'rmsprop'])

    if optimizer_name == 'adam':
        optimizer = Adam(learning_rate = 0.00025)
    elif optimizer_name == 'sgd':
        optimizer = SGD(learning_rate = 0.00025, momentum = 0.9, nesterov= True)
    elif optimizer_name == 'rmsprop':
        optimizer = RMSprop(learning_rate = 0.000025, momentum= 0.9)

    model.build(input_shape=(None, 128, 128, 1))

    # Compile model
    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

```python
tuner = GridSearch(build_128_aug_model, objective='val_accuracy', max_trials=3)

#search the best parameter
best_params_128 = tuner.search(
    train_128V2.batch(10),
    epochs=200,
    validation_data=val_data_128.batch(10)
    )
```

Keras Tuner shows **Adam optimizer** is **best** for both **31 x 31 & 128 x 128 images**

Keras Tuner

- An easy-to-use, scalable hyperparameter optimization framework that search the hyperparameter space to find the best set of hyperparameters

Optimizers

Adam

- An adaptive learning rate optimization algorithm
- Maintains a moving average of both the gradients and the second moments of the gradients
- Adapts the learning rate of each parameter individually

SGD (Stochastic Gradient Descent)

- Basic optimization algorithm which update the model parameters based on the negative gradient of the loss function with respect to each parameter.
- Each update is based on a random subset of the training data (mini batch)

RMSprop (Root Mean Square Propagation)

- An adaptive learning rate optimization algorithm
- Address some of the limitations of basic SGD by adapting the learning rates of each parameter
- Divides the learning rate for a weight by the moving average of the magnitude of recent gradients for that weight

```
Trial 3 Complete [00h 30m 15s]
val_accuracy: 0.859333336353302

Best val_accuracy So Far: 0.9490000009536743
Total elapsed time: 01h 29m 32s
```

```python
# Get the best trial
best_trial_128 = tuner.oracle.get_best_trials(num_trials=1)[0]

# Access the best parameters
best_params_128 = best_trial_128.hyperparameters.values

print(f"Best Parameters: {best_params_128}")
```

```
Best Parameters: {'optimizer': 'adam'}
```

Observations

- Keras Tuner shows that Adam Optimizer is the best optimizer

Regularisation

Regularization - Reduces overfitting and improve generalisation performance of the model

- **L1 Regularization**:
  - Penalise the absolute value of the weights. Weights may be reduced to zero which is useful when compressing the model.

$$L1(\theta) = \lambda \sum_{i=1}^{n} |w_i|$$

- **L2 Regularization**:
  - Known as weight decay and usually better over L1. Forces the weights to decay towards zero.

$$L2(\theta) = \lambda \sum_{i=1}^{n} w_i^2$$

- **Dropout**
  - Have been using this form of regularisation so far.
  - At every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections as shown below.
- I have also increased the learning rate so that the model will learn and converge faster

Regularisation (128 x 128)

```python
# Try for 128 x 128 images
tf.keras.backend.clear_session()

Conv2D_128_improved_aug = Sequential(name="Conv2D_128_Final_Augmentation",
    layers = [
        normalised_data,
        Conv2D(64, (5, 5), activation='relu', padding='same', input_shape=(128, 128, 1), strides=(4, 4)),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu', padding='same', strides=(1, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu', padding='same', strides=(1, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu', padding='same', strides=(1, 1)),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dropout(0.5),

        Dense(1024, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.5),
        Dense(128, activation = 'relu', kernel_regularizer=l2(0.001)),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ]
)

# Compile model
opt = Adam(learning_rate=0.0001)
Conv2D_128_improved_aug.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Conv2D_128_improved_aug.build(input_shape=(None, 128, 128, 1))

Conv2D_128_improved_aug_history = Conv2D_128_improved_aug.fit(
    train_128V2.batch(10),
    epochs=200,
    validation_data=val_data_128.batch(10)
    )
```

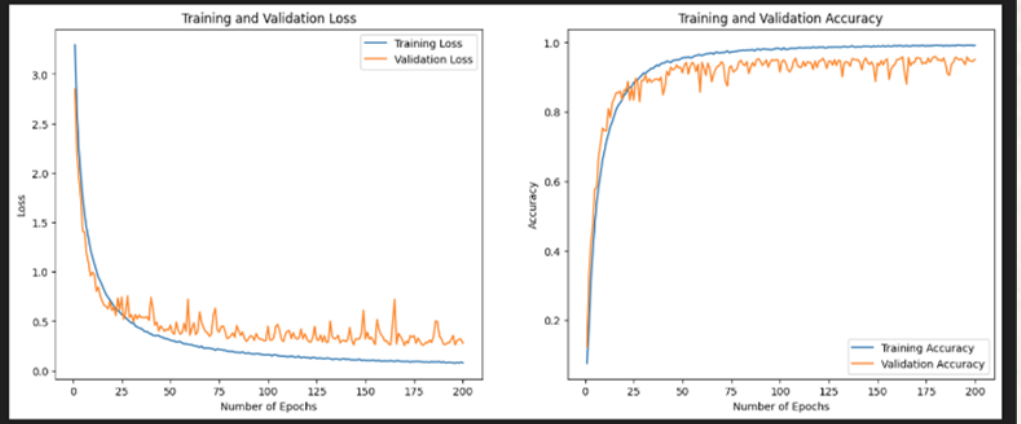
Increased learning rate, Added **L2 kernel regularizer**

```
Conv2D_128_improved_aug.summary()
```

```
Model: "Conv2D_128_Final_Augmentation"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalised_data (Sequential  (None, None, None, 1)    0
 )

 conv2d (Conv2D)             (None, 32, 32, 64)        1664

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 16, 16, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 conv2d_3 (Conv2D)           (None, 4, 4, 256)         295168

 max_pooling2d_3 (MaxPooling  (None, 2, 2, 256)        0
 2D)

 flatten (Flatten)           (None, 1024)              0

 dropout (Dropout)           (None, 1024)              0

 dense (Dense)               (None, 1024)              1049600

 dropout_1 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 128)               131200

 dropout_2 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 15)                1935

=================================================================
Total params: 1,701,007
Trainable params: 1,701,007
Non-trainable params: 0
_________________________________________________________________
```

Regularisation (128 x 128)

```
plot_learning_curve(Conv2D_128_improved_aug_history.history)
```



Observations

- Validation accuracy and loss curve are converging towards the training curve, which shows the decrease in learning rate and regularisation do indeed improve the accuracy.

```
Conv2D_128_improved_aug.evaluate(test_data_128.batch(10))
```

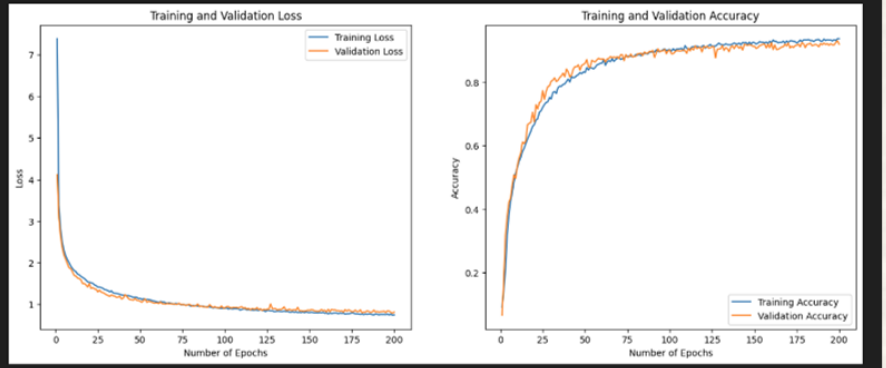
```
300/300 [==============================] - 2s 7ms/step - loss: 0.2710 - accuracy: 0.9493

[0.2709842622280121, 0.9493333101272583]
```

Regularisation (31 x 31)

```python
## Using L2 Regularization for 31 x 31 images
tf.keras.backend.clear_session()

Conv2D_31_L2 = Sequential(name="Conv2D_31_L2",
    layers = [
        normalised_data,
        Conv2D(32, (3, 3),input_shape=(31, 31 ,1), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dropout(0.5),

        Dense(512, activation='relu', kernel_regularizer=l2(0.01)),
        Dropout(0.5),
        Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
        Dropout(0.5),
        Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
        Dropout(0.5),

        Dense(num_classes, activation='softmax')
    ]
)

# Compile model
opt = Adam(learning_rate=0.0001)
Conv2D_31_L2.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Conv2D_31_L2.build(input_shape=(None, 31, 31, 1))

Conv2D_31_L2_history = Conv2D_31_L2.fit(
    train_data_31.batch(10),
    epochs=200,
    validation_data=val_data_31.batch(10)
    )
```

```python
plot_learning_curve(Conv2D_31_L2_history.history)
```

Python



Observations

- The curves show how accurate the training and validation accuracy is after including L2, the losses experienced are also quite similar.

```python
Conv2D_31_L2.evaluate(test_data_31.batch(10))
```

Python

```
300/300 [==============================] - 2s 5ms/step - loss: 0.7772 - accuracy: 0.9347

[0.7772353291511536, 0.9346666932106018]
```

Weights Save

Final Models

- Based on observations of the learning curve, I have picked my two final models to classify 31 x 31 images and 128 x 128 images.
  - **31 x 31 image**: L2 Regularizer Conv2D
  - **128 x 128 image**: L2 Regularizer Conv2D with Augmentation

```python
# Save the weights for 31 x 31
Conv2D_31_L2.save_weights("./CNN_FinalModels/Final_31x31.h5")
```

Python

```python
# Save the weights for 128 x 128
Conv2D_128_improved_aug.save_weights("./CNN_FinalModels/Final_128x128.h5")
```

Python

Step 4: Model Evaluation (31x31)

```
report_31 = classification_report(
    y_test_int_31,
    np.argmax(y_pred_31, axis=1),
    target_names=classes)
print(report_31)
```

	precision	recall	f1-score	support
Bean	0.95	0.94	0.94	200
Bitter_Gourd	0.95	0.94	0.94	200
Bottle_Gourd	0.97	0.99	0.98	200
Brinjal	0.92	0.94	0.93	200
Broccoli	0.94	0.95	0.95	200
Cabbage	0.93	0.92	0.92	200
Capsicum	0.95	0.95	0.95	200
Carrot	0.98	0.89	0.93	200
Cauliflower	0.90	0.94	0.92	200
Cucumber	0.92	0.98	0.95	200
Papaya	0.94	0.94	0.94	200
Potato	0.90	0.90	0.90	200
Pumpkin	0.91	0.95	0.93	200
Radish	0.97	0.92	0.94	200
Tomato	0.89	0.88	0.88	200
accuracy			0.93	3000
macro avg	0.94	0.93	0.93	3000
weighted avg	0.94	0.93	0.93	3000

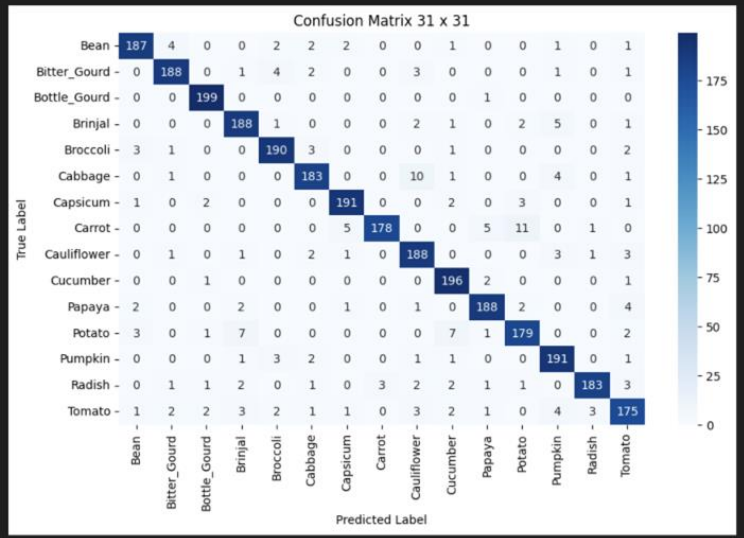
Classification Report

Observations

- Based on the other score metrics like precision, recall and f1, the scores seem to show that the vegetables are being identified quite well.

```
# Display the confusion matrix
cm_31 = confusion_matrix(y_test_int_31, np.argmax(y_pred_31, axis=1))

cm_df = pd.DataFrame(cm_31, index=classes, columns=classes)
plt.figure(figsize=(10, 6))
sns.heatmap(cm_df, annot=True, cmap="Blues", fmt="d")
plt.title("Confusion Matrix 31 x 31")
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.show()
```

Confusion Matrix

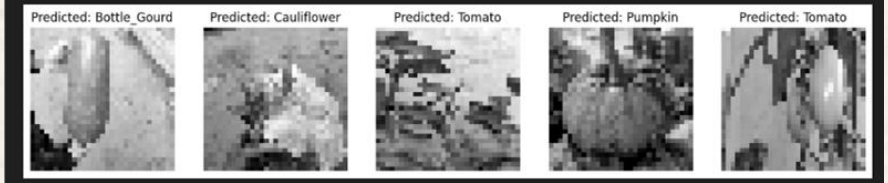


Observations:

- Based on observations, it seems that cabbage and cauliflower are getting mixed up, as well as carrot and potato. Most likely the lower pixels caused some details to be lost.
- Cabbage and cauliflower have similar sizes and the lower pixels caused details like shapes and textures to be lost.

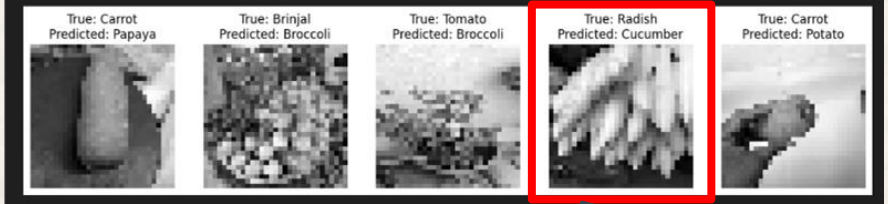
Step 4: Model Evaluation (31x31)

Prediction Visualisation




```
incorrect_indices_31 = np.where(np.argmax(y_pred_31, axis=1) != y_test_int_31)[0]

# Randomly select a few incorrect predictions for visualisation
num_samples = min(5, len(incorrect_indices_31))
selected_indices_31 = np.random.choice(incorrect_indices_31, size=num_samples, replace=False)

#Visualise the images
plt.figure(figsize=(15, 5))

for i, index in enumerate(selected_indices_31):
    plt.subplot(1, num_samples, i + 1)
    plt.imshow(X_test_31[index], cmap='gray')
    plt.title(f"True: {classes[y_test_int_31[index]]}\nPredicted: {classes[np.argmax(y_pred_31, axis=1)[index]]}")
    plt.axis("off")
```

Visual Error Analysis



Observations

- As such, the lower pixels caused the textures to be lost, causing images like radish and cucumber to be mixed up.

Step 4: Model Evaluation (128x128)

```
report_128 = classification_report(
    y_test_int_128,
    np.argmax(y_pred_128, axis=1),
    target_names=classes)
print(report_128)
```

Classification Report

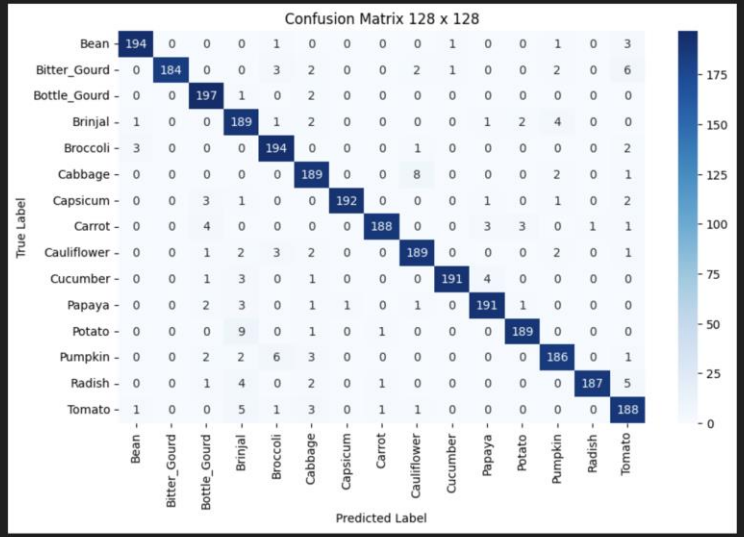
	precision	recall	f1-score	support
Bean	0.97	0.97	0.97	200
Bitter_Gourd	1.00	0.92	0.96	200
Bottle_Gourd	0.93	0.98	0.96	200
Brinjal	0.86	0.94	0.90	200
Broccoli	0.93	0.97	0.95	200
Cabbage	0.91	0.94	0.93	200
Capsicum	0.99	0.96	0.98	200
Carrot	0.98	0.94	0.96	200
Cauliflower	0.94	0.94	0.94	200
Cucumber	0.99	0.95	0.97	200
Papaya	0.95	0.95	0.95	200
Potato	0.97	0.94	0.96	200
Pumpkin	0.94	0.93	0.93	200
Radish	0.99	0.94	0.96	200
Tomato	0.90	0.94	0.92	200
accuracy			0.95	3000
macro avg	0.95	0.95	0.95	3000
weighted avg	0.95	0.95	0.95	3000

Observations

- Based on the other score metrics like precision, recall and f1, the scores seem to show that the vegetables are being identified quite well.

```
# Display the confusion matrix
cm_128 = confusion_matrix(y_test_int_128, np.argmax(y_pred_128, axis=1))

cm_df = pd.DataFrame(cm_128, index=classes, columns=classes)
plt.figure(figsize=(10, 6))
sns.heatmap(cm_df, annot=True, cmap="Blues", fmt="d")
plt.title("Confusion Matrix 128 x 128")
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.show()
```

Confusion Matrix



Observations:

- Vegetables like Brinjal and Potato are easy for the CNN to mix up as they are shaped quite similarly, hence it has the most labels getting mixed up.
- Otherwise, the model is able to predict most labels accurately.

Step 4: Model Evaluation (128x128)

Prediction Visualisation

```
sample_images = PrintSampleImages(model=Conv2D_128_improved_aug, test_data=test_data_128.batch(10))
```

Python

```
1/1 [==============================] - 0s 16ms/step
```

Error Analysis

- We will only inspect the 128 x 128 images as it is clearer for us to visualise what went wrong.

Visual Error Analysis

```
incorrect_indices_128 = np.where(np.argmax(y_pred_128, axis=1) != y_test_int_128)[0]

# Randomly select a few incorrect predictions for visualisation
num_samples = min(5, len(incorrect_indices_128))
selected_indices_128 = np.random.choice(incorrect_indices_128, size=num_samples, replace=False)

#Visualise the images
plt.figure(figsize=(15, 5))

for i, index in enumerate(selected_indices_128):
    plt.subplot(1, num_samples, i + 1)
    plt.imshow(X_test_128[index], cmap='gray')
    plt.title(f"True: {classes[y_test_int_128[index]]}\nPredicted: {classes[np.argmax(y_pred_128, axis=1)[index]]}")
    plt.axis("off")
```

Python

Observations

- Based on the images, we can see that vegetables like radish is very hard to predict correctly without the use of colors, same goes with the others.
- Shapes of the other vegetables like papaya and potato are also quite similar hence the CNN model will also think the same way.
- I am not sure why papaya is even considered a vegetable in the dataset but it is.

Summary

When training CNN models, **regularization techniques** like **L2 and Dropout** are very efficient when dealing with **images of smaller batch sizes**. **Data augmentation** does help the model's accuracy but it **depends on case-to-case** scenarios. Room for improvement can be made to the images like **trying image segmentation** or even trying out **other models like VGG** to classify the vegetables.

Thank you!

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics and images by **Freepik**