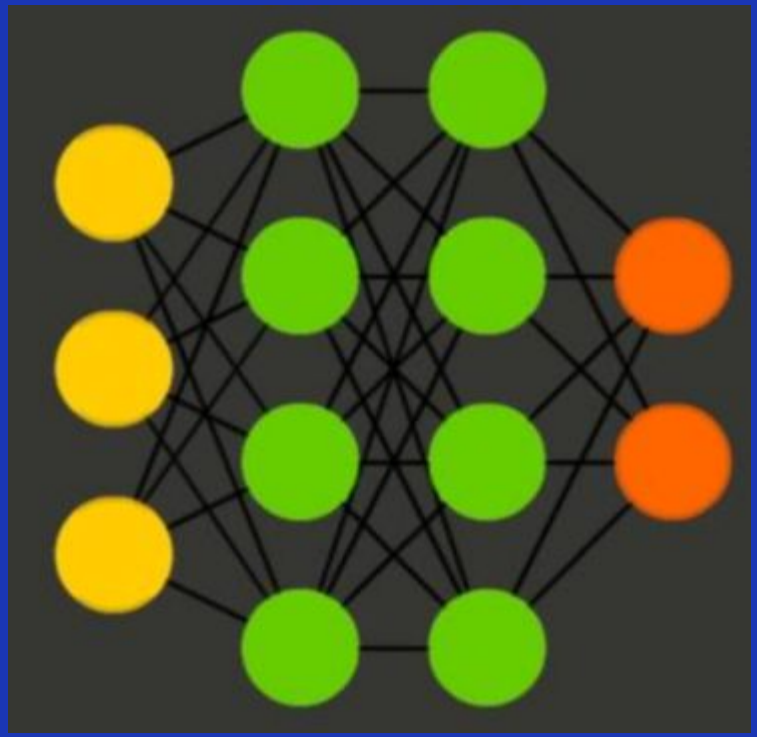
AI-201

Image Classification

Naveen Prabakar, Gregory Chernyavskiy, Riza Danurdoro



Our Project

- Image classification model on 7 different types of vegetables
- Acquired dataset from kaggle. 7 different vegetables with 1000 photos each
- Libraries used:
  - TensorFlow for CNN and Keras
  - Pandas & SeaBorn for visualization
  - Numpy for image manipulation
  - ChatGPT & Gemini used

ImageData Generator

- Picked dimensions for the image (180 × 180)
- Selected a batch size for training (32 images)
- Split the Images: 80 percent training & 20 percent validation
- Rescaled images (converted pixels to range 0-1)
- Rotation Range set to 20, so images randomly rotate during training
- width/height shift randomly moves the image up/down or right/left
- Shear range tilts the image on an axis
- Zoom range zooms in or out on the image
- Horizontal flip randomly flips the images left or right

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    validation_split=0.2
)
```

Pre-Training

- Loads in images from ‘train’ directory
- Target size is the size of the images (predefined before)
- Batch size is the number of images loaded in one batch
- Class mode is set to categorical, since there is multiple images
- Subset is defined for training (80 percent) and validation (20 percent)

```
# Training data generator
train_generator = train_datagen.flow_from_directory(
    'train',
    target_size=(img_height, img_width),  # Resize images
    batch_size=batch_size,  # Batch size
    class_mode='categorical',  # Use categorical for multi-class classification
    subset='training'  # Use 80% for training
)

# Validation data generator
val_generator = train_datagen.flow_from_directory(
    'train',
    target_size=(img_height, img_width),  # Resize images
    batch_size=batch_size,  # Batch size
    class_mode='categorical',  # Use categorical for multi-class classification
    subset='validation'  # Use 20% for validation
)
```

CNN Model

- Models Sequential initializes the sequential model
- Conv2D
  - First Parameter is the filter. The model tries to x number of ways
  - Second Parameter is pixel size. The model looks at n x n size pixels.
  - Third Parameter is activation. Relu adds non-linearity
  - Fourth Parameter is the shape of the image: 180 × 180 size with 3 color channels
- Max Pooling 2D shrinks the image, but keeps it's biggest value
- Flatten, flattens the 2-d image to 1-d
- Dense, creates neuron layer with x number of units
- Batch Normalization and kernel regularizer is similar to Normalizer in ML

```
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=input_shape),
    layers.BatchNormalization(),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(256, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(2,2),
    layers.Flatten(),
    layers.Dense(512, activation='relu', kernel_regularizer=l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])
```

Model Training

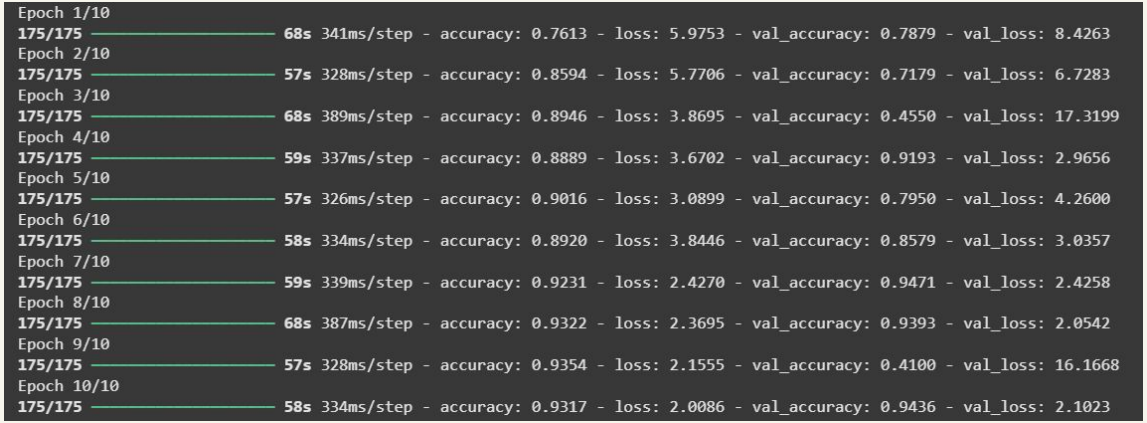
- Optimizer: ‘adam’, it helps the model by adjusting weights based on errors
- Loss: ‘categorical_crossentropy’, is the loss function. It tells the model how far it is off
- Metrics: ‘accuracy’ tells the model what accuracy metrics to present
- Model fitting involves passing in the pre trained generators (train and validation), along with epochs
- Epochs is how many times it will loop through the images: in this case it will run the model 10 times

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Model Training
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator
)
```

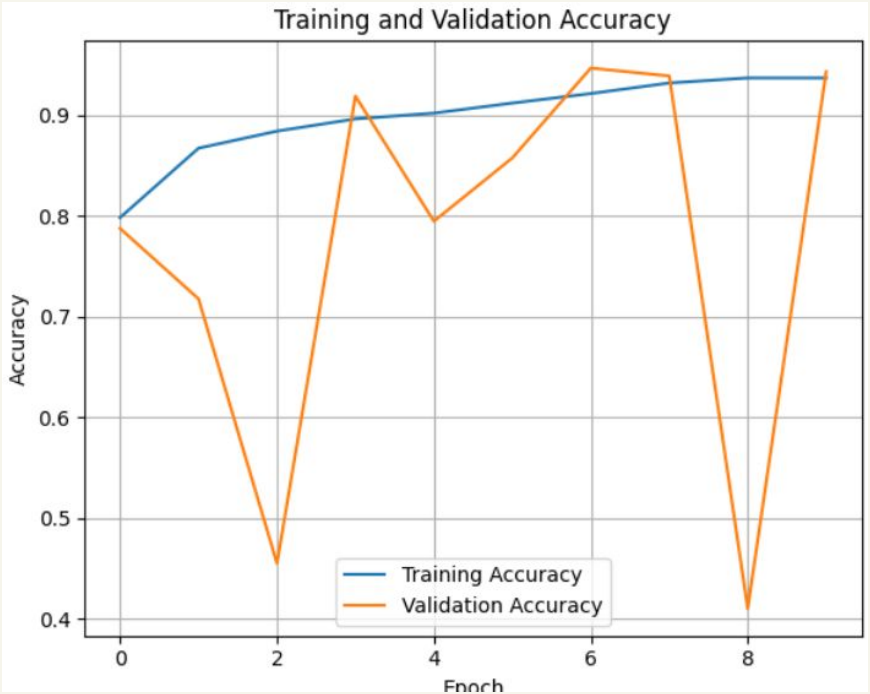
Processing

- Total of 10 Epoch
- Accuracy is the correctness in prediction
- Loss is how wrong the model was
- Val accuracy is the accuracy on data it's never seen before
- Val loss is the how wrong the model was on data it's never seen before



Insights

- Training and Validation accuracy increase till they plato in the 0.90-0.95 range
- Not underfitting, since the training accuracy is high
- Possible overfitting because of dramatic drops in validation accuracy, but recovered in the end
- There were fluctuations in the validation accuracy, so we decided to tune the model for better accuracy



Insights

- Training & Validation loss is steadily decreasing
- Not underfitting since loss is gradual for Training
- Possible overfitting due to sharp increase of loss in validation, but it recovers towards the end
- There were fluctuations, in both the loss and accuracy, so we decided to tune the model for more consistency

Parameter Tuning

- Increased the epochs from 10 to 20. This will increase the number of times the model trains on the dataset
- Lowered the learning rate. It will help the model learn slower, which could help it predict better
- Adding in Early Stopping. A parameter that will stop the model if it starts to over fit

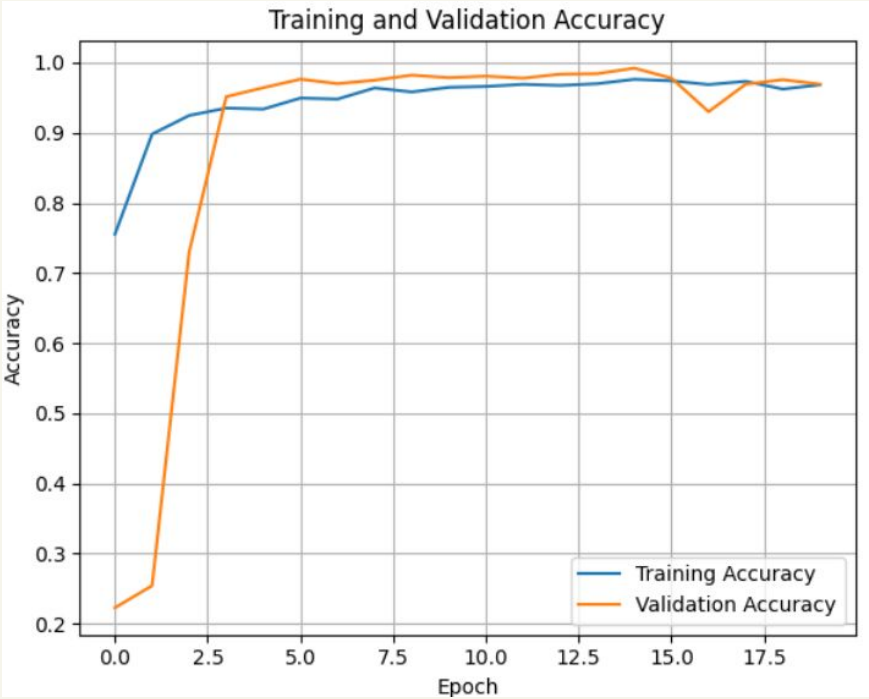
```
#With hyper-parameter tuning
def trainer(model, train_generator, val_generator, epochs=20): #more epochs, lower learning rate
    model.compile(
        optimizer=Adam(learning_rate=0.0001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    early_stopping = EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True
    )

    history = model.fit(
        train_generator,
        epochs=epochs,
        validation_data=val_generator,
        callbacks=[early_stopping]
    )
```

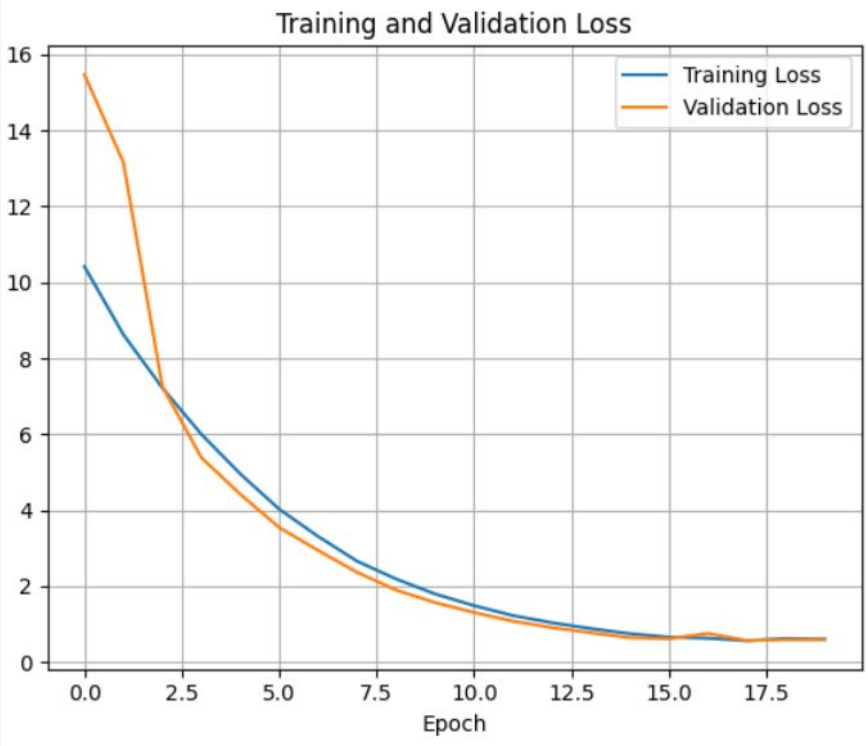
New Results

- Better consistency in accuracy for the validation. There were no more fluctuations, like before.
- Model is not over/under fitting, since Training & Validation accuracy are about the same
- Sharp increase in accuracy in the third epoch, with slow gradual improvement after that



New Results

- Improved consistency in validation loss. No fluctuations, like before.
- No over/under fitting, since the loss of both training and validation are around the same
- Gradual loss over time for both sets.
- Losses plato after epoch 15



New Results

- Higher Training accuracy: 0.9317 to 0.9694
- Higher Validation accuracy: 0.9436 to 0.9693
- Overall, the tuning helped improve both accuracies, with more consistencies
- Since the tuning fixed the previous problems, we decided to move to the next phase

```
Epoch 10/20
175/175 ━━━━━━━━━━━━━━━━━━━━ 59s 337ms/step - accuracy: 0.9642 - loss: 1.8650 - val_accuracy: 0.9786 - val_loss: 1.5650
Epoch 11/20
175/175 ━━━━━━━━━━━━━━━━━━━━ 59s 336ms/step - accuracy: 0.9669 - loss: 1.5418 - val_accuracy: 0.9807 - val_loss: 1.3044
Epoch 12/20
175/175 ━━━━━━━━━━━━━━━━━━━━ 59s 339ms/step - accuracy: 0.9709 - loss: 1.2772 - val_accuracy: 0.9779 - val_loss: 1.0750
Epoch 13/20
175/175 ━━━━━━━━━━━━━━━━━━━━ 58s 333ms/step - accuracy: 0.9692 - loss: 1.0629 - val_accuracy: 0.9836 - val_loss: 0.9059
Epoch 14/20
175/175 ━━━━━━━━━━━━━━━━━━━━ 59s 336ms/step - accuracy: 0.9712 - loss: 0.9022 - val_accuracy: 0.9843 - val_loss: 0.7697
Epoch 15/20
175/175 ━━━━━━━━━━━━━━━━━━━━ 60s 342ms/step - accuracy: 0.9733 - loss: 0.7840 - val_accuracy: 0.9921 - val_loss: 0.6378
Epoch 16/20
175/175 ━━━━━━━━━━━━━━━━━━━━ 58s 334ms/step - accuracy: 0.9765 - loss: 0.6588 - val_accuracy: 0.9779 - val_loss: 0.6145
Epoch 17/20
175/175 ━━━━━━━━━━━━━━━━━━━━ 58s 331ms/step - accuracy: 0.9647 - loss: 0.6490 - val_accuracy: 0.9300 - val_loss: 0.7527
Epoch 18/20
175/175 ━━━━━━━━━━━━━━━━━━━━ 60s 341ms/step - accuracy: 0.9742 - loss: 0.5755 - val_accuracy: 0.9693 - val_loss: 0.5666
Epoch 19/20
175/175 ━━━━━━━━━━━━━━━━━━━━ 59s 340ms/step - accuracy: 0.9657 - loss: 0.5820 - val_accuracy: 0.9757 - val_loss: 0.5863
Epoch 20/20
175/175 ━━━━━━━━━━━━━━━━━━━━ 59s 339ms/step - accuracy: 0.9694 - loss: 0.5988 - val_accuracy: 0.9693 - val_loss: 0.5936
```

Testing

- For testing, a test set of 1400 images the model has never seen before was used
- 200 images per vegetable (7).
- A Test generator was created for testing the model.
- Batch size & Target size remained the same for consistency

```
test_generator = train_datagen.flow_from_directory(
    'test', #use the test file, images the model has never seen before
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
  )
```

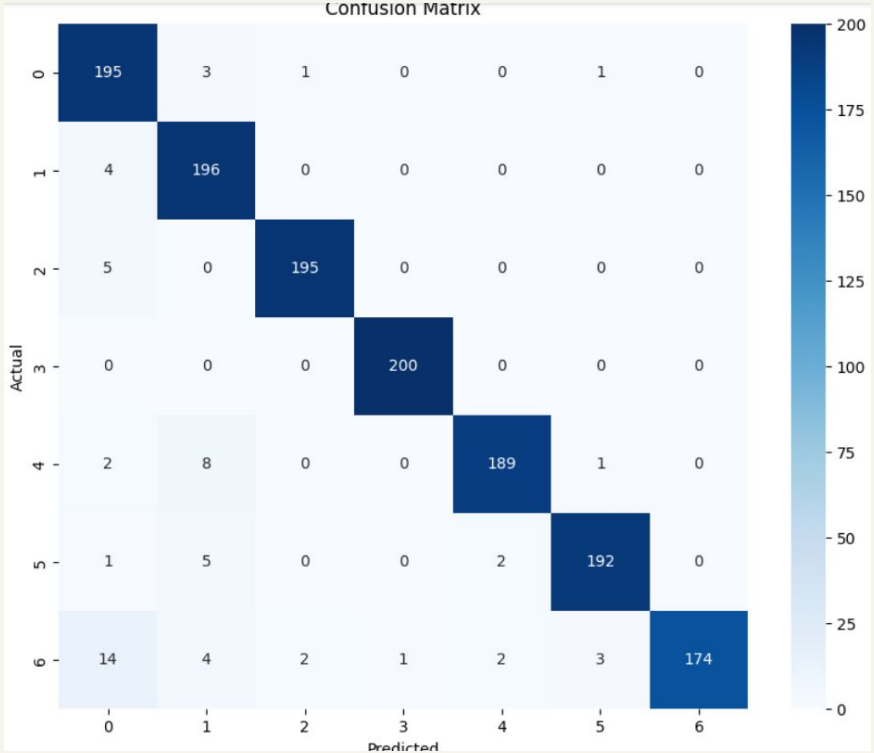
Test Results

- Precision checks how many were correct out of all predictions
- Recall checks out of all instances, how many did the model identify
- F1-score is the mean between precision and recall
- Support is the number of samples
- For vegetable three, the accuracy was a 100%!
- Average accuracy was 96 percent across 1400 images.

	precision	recall	f1-score	support
0	0.88	0.97	0.93	200
1	0.91	0.98	0.94	200
2	0.98	0.97	0.98	200
3	1.00	1.00	1.00	200
4	0.98	0.94	0.96	200
5	0.97	0.96	0.97	200
6	1.00	0.87	0.93	200
accuracy			0.96	1400
macro avg	0.96	0.96	0.96	1400
weighted avg	0.96	0.96	0.96	1400

Test Results

- Each Row is the the actual class
- Each column is the predicted class
- All the classes except number 6 have accuracies of higher then 94 percent
- Class 6 had an accuracy of 87 percent, however this can be explained by the fact this vegetable had the same color as another
- Overall, the model is very good at predicting for the most part



Test Results

```
Total Misclassifications: 59
```

- 59 images predicted incorrectly out of the 1400
- Main cause was similar characteristics (colors, shapes) to other vegetables.
- The main vegetable the model could not get right was Brinjal (eggplant), due to its color and shape being similar to other vegetables in the data set

Future Improvements

- To deal with the confusion in color and shapes, use better data augmentation to be able to differentiate better.
- Try testing the cnn model with different parameters, such as different target size and batch size
- Try different image classifications models if it is better suited for the task

Sources:

Image Dataset: Vegetable Image Dataset

Google Colab:
https://colab.research.google.com/drive/1KqsCwGgSOdYRKDFs9BZz_23OuAudzhf0?usp=sharing

Thank you!

Naveen, Gregory , Riza