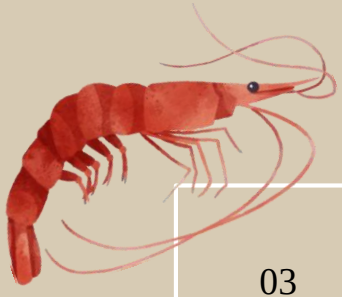
魚類辨識

110034557 方際勛

指導教授：邱銘傳　博士



研究背景

研究背景

當民眾想去漁港、市場及餐廳消費時，有些不肖業者會因為顧客不熟悉魚之品種，而刻意提高魚類售價，賺取更多利潤，而此些現象在觀光漁港更為常見。

研究背景

本研究欲透過CNN方法將九種魚之品種（分別為黑海棱鯡、金頭鯛、竹筴魚、紅鰭魚、真鯛、海鱸魚、縱帶羊魚、鱒魚及蝦）加以分類，並將本研究之CNN模型提供予消費者，以協助消費者辨識魚之品種。

5W1H

What	消費者不熟悉魚之品種，於觀光漁港或海鮮餐廳消費時，易被不肖業者無理剝削。
Where	消費者之手機、相機等等
Who	缺乏魚類知識之消費者
When	當消費者於魚市場挑選魚貨時
Why	減少民眾於觀光漁港或海鮮餐廳消費時，被餐廳之不肖業者或黑心商家無理剝削之情事發生。
How	以九種魚之品種的資料訓練CNN模型，以協助消費者辨識魚類之品種

研究過程

資料說明

本研究利用CNN模型進行魚類圖像辨識，而本研究所選擇之資料集為Kaggle公開數據集中選擇之A Large Scale Fish Dataset資料集，其中包含九種魚蝦圖像。每一種類各1000張照片共9000張

資料前處理

```
# filter folders with GT (Ground Truth) at the end
df = df[df['Labels'].apply(lambda l: l[-2:] != 'GT')]
```

用alpha圖過濾原始圖片

資料前處理

```
# stratisfied train and test (10%) datasets
X_train, X_test = train_test_split(df, test_size=0.1, stratify=df['Labels'])

print('Shape of Train Data: ', X_train.shape)
print('Shape of Test Data: ', X_test.shape)

# stratisfied train and val (20%) datasets
X_train, X_val = train_test_split(X_train, test_size=0.2, stratify=X_train['Labels'])

print('Shape of Train Data: ', X_train.shape)
print('Shape of Val Data: ', X_val.shape)

# ordered count of rows per unique label
X_train['Labels'].value_counts(ascending=True)
```

```
Shape of Test Data:  (900, 2)
Shape of Train Data:  (6480, 2)
Shape of Val Data:  (1620, 2)
Sea Bass              720
Shrimp                720
Red Sea Bream         720
Trout                 720
Black Sea Sprat       720
Hourse Mackerel       720
Gilt-Head Bream       720
Striped Red Mullet    720
Red Mullet            720
Name: Labels, dtype: int64
```

```
# number of samples/images per iteration
BATCH_SIZE = 32
# input image size
IMG_SIZE = (224, 224)

# image preprocessing
img_data_gen = ImageDataGenerator(preprocessing_function=preprocess_input)

X_train = img_data_gen.flow_from_dataframe(dataframe=X_train,
                                           x_col='FilePaths',
                                           y_col='Labels',
                                           target_size=IMG_SIZE,
                                           color_mode='rgb',
                                           class_mode='categorical',
                                           batch_size=BATCH_SIZE,
                                           seed=42)

X_val = img_data_gen.flow_from_dataframe(dataframe=X_val,
                                         x_col='FilePaths',
                                         y_col='Labels',
                                         target_size=IMG_SIZE,
                                         color_mode='rgb',
                                         class_mode='categorical',
                                         batch_size=BATCH_SIZE,
                                         seed=42)

X_test = img_data_gen.flow_from_dataframe(dataframe=X_test,
                                          x_col='FilePaths',
                                          y_col='Labels',
                                          target_size=IMG_SIZE,
                                          color_mode='rgb',
                                          class_mode='categorical',
                                          batch_size=BATCH_SIZE,
                                          seed=42)
```

```
Found 6480 validated image filenames belonging to 9 classes.
Found 1620 validated image filenames belonging to 9 classes.
Found 900 validated image filenames belonging to 9 classes.
```

將9000張之魚類照片分成6480張、1620張、900張，
分別做為訓練資料、驗證資料及測試資料

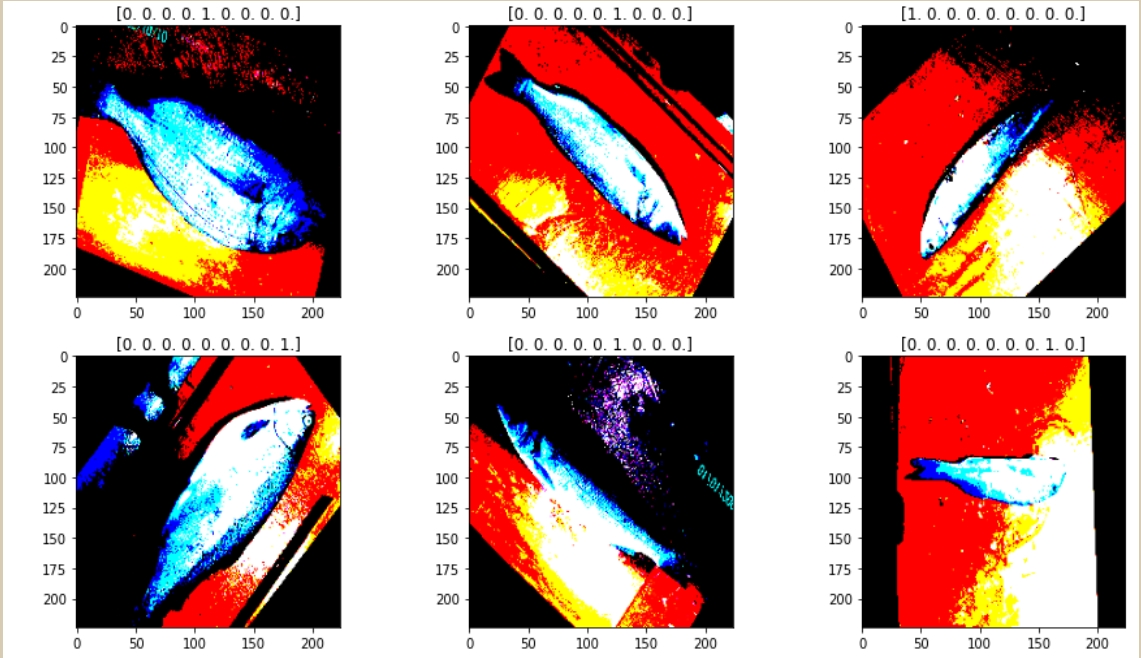
資料前處理

```
fit,  ax  =  plt.subplots(nrows=2,  ncols=3,  figsize=(13,7))

for  i,  a  in  enumerate(ax.flat):
        img,  label  =  X_train.next()
        a.imshow(img[0],)
        a.set_title(label[0])

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

將原始圖片轉換為RGB三原色



模型建構

建立之CNN模型其中包含三層卷積層、三層池化層、一層扁平層及兩層全連接層。

```
model = Sequential()
# scale image values to 0..1
model.add(tf.keras.layers.experimental.preprocessing.Rescaling(1./255))

# 1. Conv2D layer
model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same', input_shape=(224, 224, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=2, padding='valid'))

# 2. Conv2D layer
model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=2, padding='valid'))

# 3. Conv2D layer
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=2, padding='valid'))

# scale to 1 dimensional input for NN
model.add(Flatten())

# hidden fully connected layer
model.add(Dense(256))
model.add(Activation('relu'))

# inhibit overfitting
model.add(Dropout(0.2))

# output fully connected layer
model.add(Dense(9))
model.add(Activation('softmax'))

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

研究結果

研究結果

使用EarlyStopping，使準確率不會過低

```
cb = tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=3)
# train model
hst = model.fit(X_train, validation_data=X_val, epochs=10, callbacks=cb)
```

第一次實驗

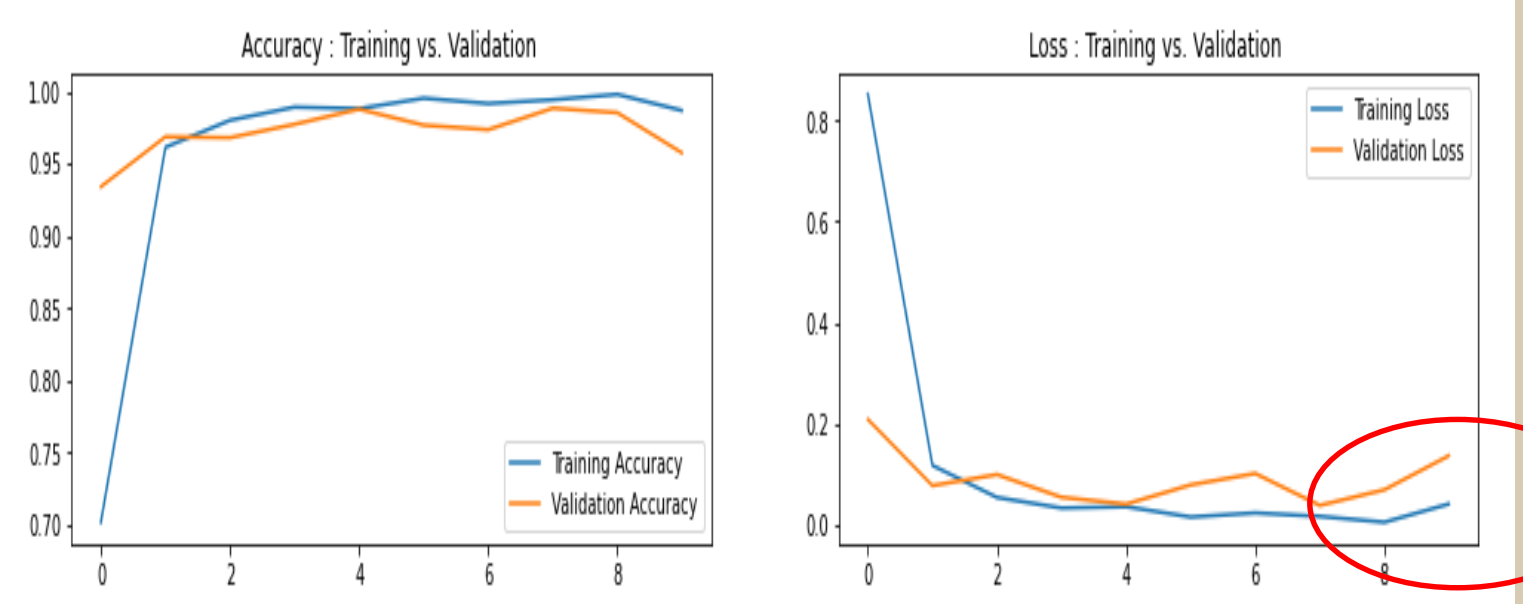
Epoch =10, Dropout =0.2, Batch size = 32

```
print(f'Train Accuracy: {hst.history["accuracy"][-1:][0] * 100:.2f}')
print(f'Val Accuracy: {hst.history["val_accuracy"][-1:][0] * 100:.2f}')
print(f'Test Accuracy: {res[1] * 100:.2f}')
```

```
Train Accuracy: 98.67
Val Accuracy: 95.74
Test Accuracy: 96.44
```

測試準確率達96.44%

研究結果



Overfitting

實驗結果統整

	Epoch	Dropout	Batch size	Accuracy
1	10	0.2	32	0.96
2	10	0.5	32	0.99
3	5	0.2	32	0.97
4	5	0.5	32	0.98

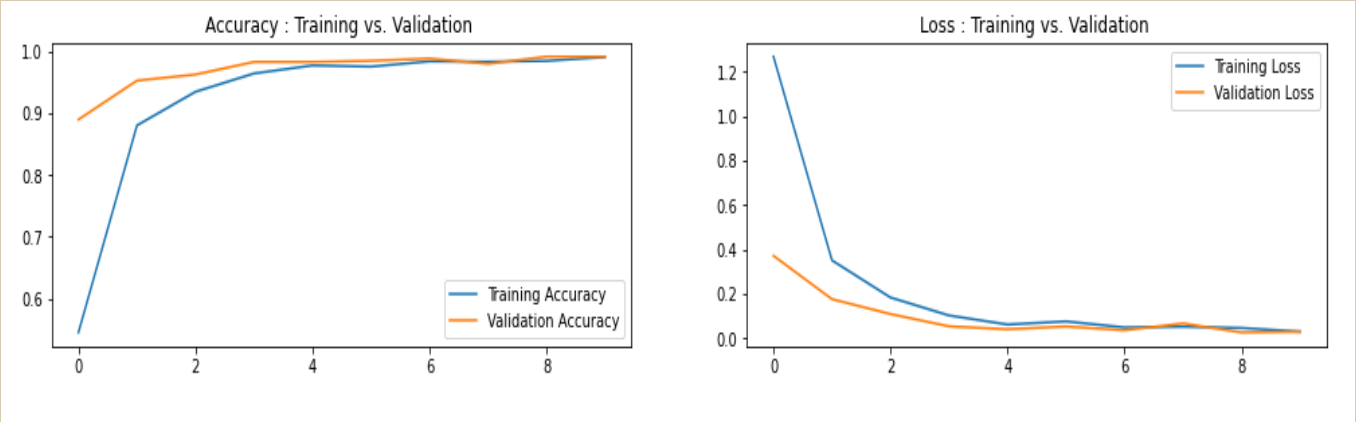
超參數調整

最終結果

epochs=10, Dropout = 0.5, Batch size =32

```
Epoch 1/10
203/203 [==============================] - 3818s 19s/step - loss: 1.2670 - accuracy: 0.5454 - val_loss: 0.3709 - val_accuracy: 0.8901
Epoch 2/10
203/203 [==============================] - 120s 590ms/step - loss: 0.3502 - accuracy: 0.8806 - val_loss: 0.1757 - val_accuracy: 0.9531
Epoch 3/10
203/203 [==============================] - 115s 566ms/step - loss: 0.1831 - accuracy: 0.9350 - val_loss: 0.1087 - val_accuracy: 0.9630
Epoch 4/10
203/203 [==============================] - 114s 562ms/step - loss: 0.1024 - accuracy: 0.9647 - val_loss: 0.0533 - val_accuracy: 0.9833
Epoch 5/10
203/203 [==============================] - 114s 560ms/step - loss: 0.0616 - accuracy: 0.9776 - val_loss: 0.0404 - val_accuracy: 0.9833
Epoch 6/10
203/203 [==============================] - 113s 555ms/step - loss: 0.0757 - accuracy: 0.9758 - val_loss: 0.0526 - val_accuracy: 0.9852
Epoch 7/10
203/203 [==============================] - 113s 559ms/step - loss: 0.0485 - accuracy: 0.9843 - val_loss: 0.0358 - val_accuracy: 0.9889
Epoch 8/10
203/203 [==============================] - 112s 554ms/step - loss: 0.0519 - accuracy: 0.9836 - val_loss: 0.0663 - val_accuracy: 0.9802
Epoch 9/10
203/203 [==============================] - 113s 555ms/step - loss: 0.0461 - accuracy: 0.9850 - val_loss: 0.0260 - val_accuracy: 0.9920
Epoch 10/10
203/203 [==============================] - 112s 551ms/step - loss: 0.0296 - accuracy: 0.9912 - val_loss: 0.0291 - val_accuracy: 0.9920
```

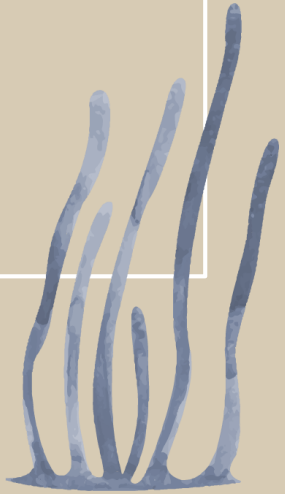
調整結果




```
print(f'Train Accuracy: {hst.history["accuracy"][-1:][0] * 100:.2f}')
print(f'Val Accuracy: {hst.history["val_accuracy"][-1:][0] * 100:.2f}')
print(f'Test Accuracy: {res[1] * 100:.2f}')
```

```
Train Accuracy: 99.12
Val Accuracy: 99.20
Test Accuracy: 99.67
```

結論與未來展望



結論

超參數調整後，本次研究之模型準確率達到99.12%，可準確分辨各種魚隻品種，若將此模型提供予消費者使用，可有效協助消費者辨識，大幅降低受騙之情形發生

未來展望

這次使用之樣本集只有9個品種，若未來將資料集擴展至更多品種，可以創造更全面更通用之CNN模型

若要實際運用於市場上，未來可考慮使用魚肉切片(如油魚、鱈魚)之圖像資料，如此一來，顧客購買經宰殺後之海鮮時，也能透過魚肉之紋理分辨魚品種

參考資料

- https://zh.wikipedia.org/wiki/%E5%8D%B7%E7%A7%AF%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C
- https://ithelp.ithome.com.tw/articles/10192028
- https://www.kaggle.com/arminfuchs/a-large-scale-fish-dataset-with-cnn-99-accuracy/notebook
- https://www.kaggle.com/crowww/a-large-scale-fish-dataset

THANKS