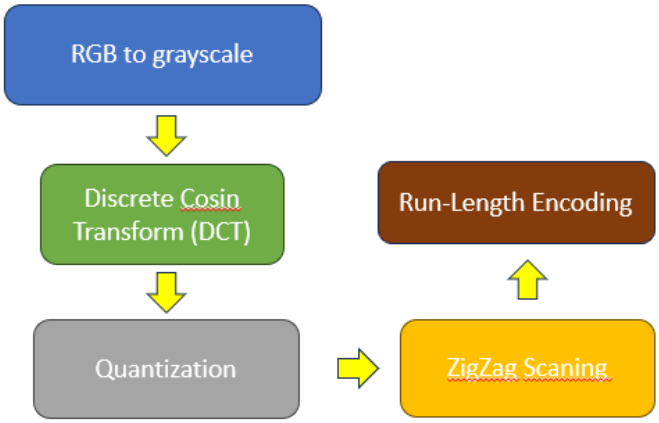
Final project report

1. Introduction



上圖是我們的 Compression Procedure，首先將 RGB 圖片轉成灰階，再透過 DCT、Quantization、ZigZag Scaning、Run-Length Encoding，最後輸出 bitstream，架構跟作業 4 是一樣的，我們主要是打算用不同的 DCT type 或 DST 和不同的 Scaning 方式，去分析壓縮的效果。

2. Experimental Dataset

Dataset: Septuplet dataset

Contain: 91,701 7-frame sequences with fixed resolution 448 x 256

我們使用的是 Septuplet dataset，這個 dataset 涵蓋了各式各樣的場景和動作，他有 91,701 個 sequences，每個 sequences 有 7-frame，每個 frame 的解析度是 448*256，上圖是這個 dataset 的其中一張圖片。

3. Implementation description and results

(1) Various DCT、DST

- Formula

	DCT	DST
I	$X_k = \frac{1}{2}(x_0 + (-1)^k x_{N-1}) + \sum_{n=1}^{N-2} x_n \cos\left[\frac{\pi}{N-1} n k\right]$	$X_k = \sum_{n=0}^{N-1} x_n \sin\left[\frac{\pi}{N+1}(n+1)(k+1)\right]$
II	$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right]$	$X_k = \sum_{n=0}^{N-1} x_n \sin\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)(k+1)\right]$
III	$X_k = \frac{1}{2}x_0 + \sum_{n=1}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(k+\frac{1}{2}\right)n\right]$	$X_k = \frac{(-1)^k}{2}x_{N-1} + \sum_{n=0}^{N-2} x_n \sin\left[\frac{\pi}{N}(n+1)\left(k+\frac{1}{2}\right)\right]$
IV	$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]$	$X_k = \sum_{n=0}^{N-1} x_n \sin\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]$

上表格是 DCT 根 DST 的各種 TYPE 的公式。DCT-I: 這種類型的 DCT 較少使用。它類似於真正的傅立葉轉換，但只使用餘弦函數。在 DCT-I 中，序列的第一個和最後一個元素是以兩倍權重計算，其餘元素則正常計算。DCT-II: 這是最常用的 DCT 類型，也是我們之前作業使用的 DCT，在 JPEG compression 中，DCT-II 可以將信號分解為不同頻率的餘弦波的和。DCT-III 通常被視為 DCT-II 的 inverse，用於將 DCT-II 的輸出轉換回原始序列。然後最後是 DCT-IV 的公式。右邊是 DST 公式，DST-II 是最常用的 DST 形式，DST-III 則一樣可以看作是 DST-II 的 inverse。

- Result

DCT

Type	PSNR	Average Bytes (KB)	bpp	Compression ratio
DCT-I	23.49	55.90	3.99	2.00
DCT-II	29.57	30.28	2.17	3.69
DCT-III	20.10	70.13	5.00	1.60
DCT-IV	18.19	76.13	5.44	1.47

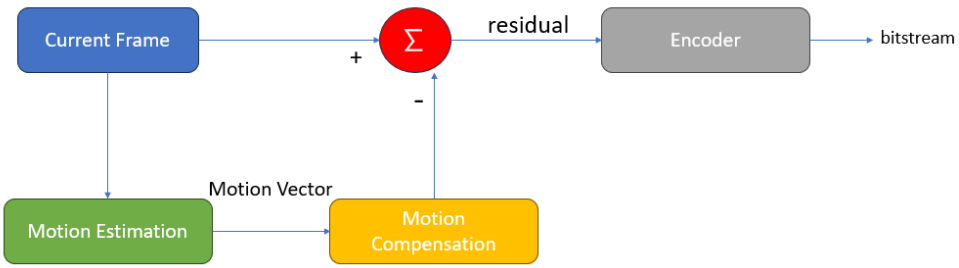
DST

Type	PSNR	Average Bytes (KB)	bpp	Compression ratio
DST-I	23.45	69.58	4.97	1.61
DST-II	17.34	83.45	5.96	1.34
DST-III	17.63	67.63	4.83	1.66
DST-IV	18.17	73.08	5.22	1.53

這是我們的實驗結果，從上面兩張表格可以看到，DCT-II 的表現，不管是 psnr、average byte、bpp 還是 compression ratio 都比其他的 DCT type 跟 DST 都好。這個結果與為什麼我們在影像壓縮中通常都使用 DCT-II 相符，我們進一步驗證了他優異的壓縮效果，不過對於這個結果，我們也推測可能會跟各個公式轉換出來的頻域特性有關。所以有可能我們在做 compression 時，使用 zigzag 的掃描方式是對 DCT-II 最友善的掃描

方式。其他的公式，我們推測可能會需要其他的掃描方式來符合他們各個轉換的特性，以讓數字可以重複出現，做出最有效的壓縮。

(2) Motion Estimation & Compensation



- Residual Result

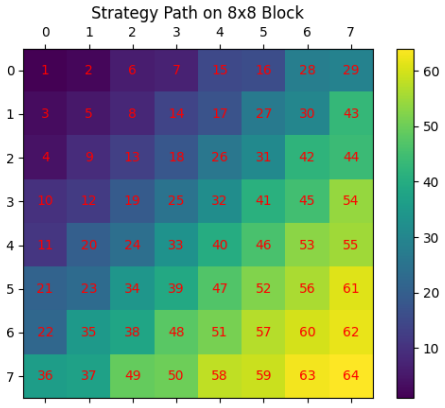
Type	PSNR	Average Bytes (KB)	bpp	Compression ratio
DCT-I	37.81	23.11	1.65	4.84
DCT-II	38.64	22.12	1.58	5.06
DCT-III	37.75	24.24	1.73	4.62
DCT-IV	36.90	25.13	1.80	4.45

Type	PSNR	Average Bytes (KB)	bpp	Compression ratio
DST-I	37.81	24.97	1.78	4.49
DST-II	36.11	25.47	1.82	4.40
DST-III	37.28	25.53	1.82	4.39
DST-IV	36.98	24.39	1.74	4.59

上表格是我們的結果，因為用了 Motion Estimation 跟 Compensation ，預測 current frame 與之前 frame 之間的運動變化，可以只存儲這些變化（即 residual），而不是完整的 frame，因為 residual 通常比完整 frame 包含的數據要少很多，因此可以實現更高的壓縮率。從上表格可以看出 DCT 跟 DST 的結果是差不多的，但是之前演講的時候演講者好像有說，DST 比 DCT 好，但效果不顯著，所以 H.265 會同時考慮 DCT 跟 DST 這兩個，選比較好的去 encode，我們覺得可能跟每個資料集的偏差也有關係。

(3) Scanning strategy for run-length encoding

在標準 JPEG compression 中，會使用 zigzag scan 進行 run-length encoding，就像下圖是 zigzag scanning 的路徑。



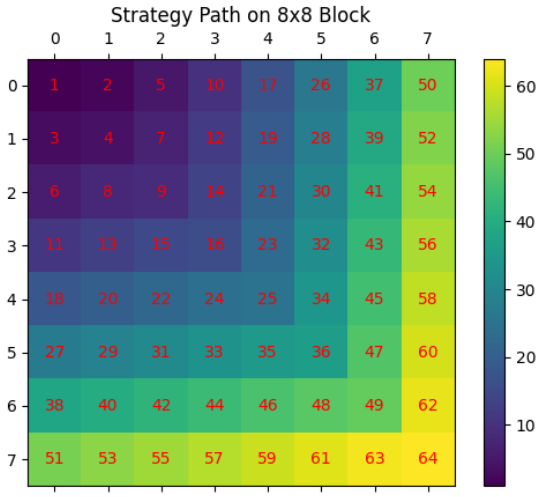
這種路徑目的在找到 DCT 中連續的 0。通過這種方式，我們可以使用更少的符號來 encode DCT 塊。例如如果我想要 encode AABABBA 跟 AAAABBB，他們都有 4 個 A 和 3 個 B，再 run-length encoding 後，結果就是下面這樣。

'AABABBA' -> [(2, 'A'), (1, 'B'),(1, 'A'), (2, 'B'),(1, 'A')]
'AAAABBB' -> [(4, 'A'), (3, 'B')]

我們需要 5 個 symbol 去 encode AABABBA，但是對於 AAAABBB，只需要兩個 symbol 去 encode，所以我們想嘗試不同的 scanning strategy，看看會對 encode 出來的 data 大小有什麼影響。

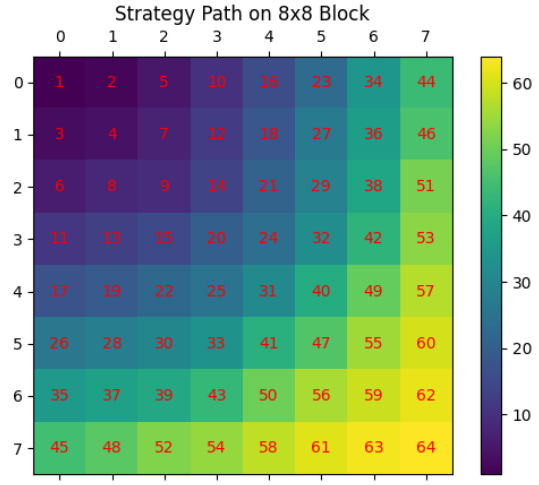
在這個實驗中，除了 zigzag strategy 之外，我們還嘗試了四種不同的 strategy。

- Handcrafted strategy 1



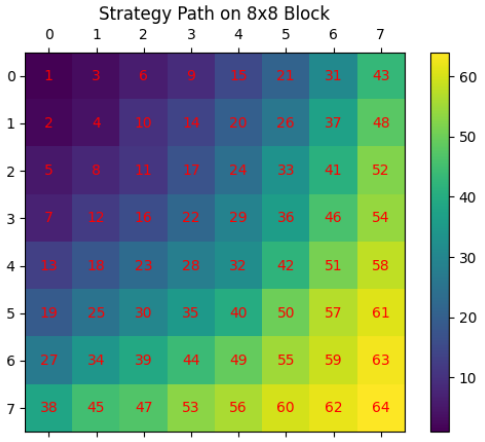
第 1 種 strategy 基於選擇較高頻率的元素，但它更傾向於選擇非對角線元素。

- Handcrafted strategy 2



第 2 種 strategy 一樣是基於選擇較高頻率的元素，但它更傾向於選擇對角線元素。

- Zero count strategy



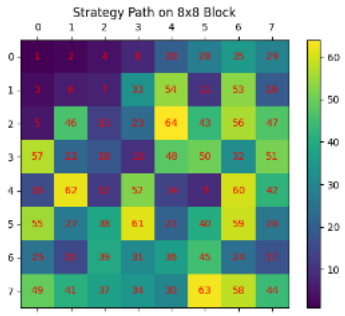
第 3 種 strategy 是基於先選擇出現次數最少的零。

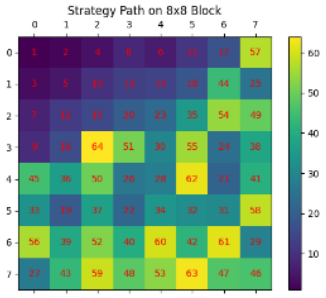
- Neural network strategy

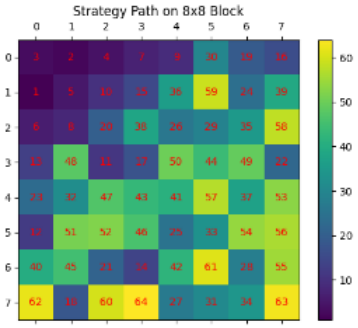
第四種 strategy 是我們設計了一種 Neural network 去反覆產生 strategy，下方是我們的 pseudo code。

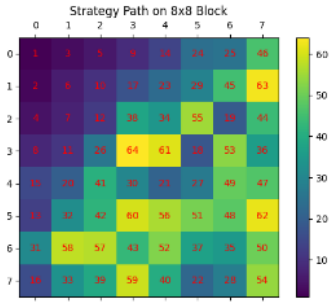
```
for iteration in range(training_iteration):
current_step = array((8,8))
strategy = []
for i in range(64): #64 is the number of block element
strategy_prob = model(quantized_dct, current_step)
strategy_prob = set_have_chosen_minus_one(mean(strategy_prob))
chosen_step = argmax(strategy_prob)
strategy.append(chosen_step)
current_step[argmax] = -1

rle_data = run_length_encoding(quantized_dct, strategy)

#minimize the number of symbol in rle_data
model_optimize(rle_data)
```

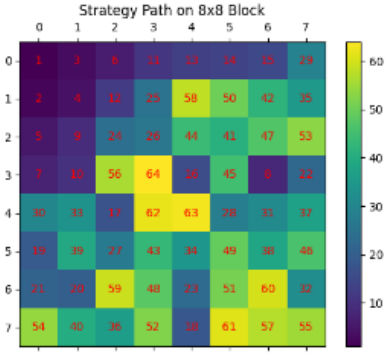
Neural network strategy 的方法就是，我們去訓練一個模型，讓他輸出 strategy。我首先將模型訓練到收斂，再將最終輸出的 strategy 應用在我們實驗，因為我們訓練過模型很多次，所以有很多的 strategy，下面是一些我們的 strategy 的 example。











- Result and analysis

	Zigzag	Zero count	strategy 1	strategy 2	Neural network (Best)
Mean of the number of symbols	35.17	35.72	36.02	35.96	37.35
Median of the number of symbols	36	37	37	37	39

這是我們的結果，上表格顯示了我們使用不同的策略對每個 DCT 區塊進行編碼需要多少個 symbol。我們可以看到，zigzag strategy 對 DCT 區塊 encode 的符號數量最少，所以 對於傳統的 DCT，使用 zigzag 策略似乎是最好的選擇。儘管 Neural network strategy 表現不佳，但是我們仍然可以觀察到，Neural network strategy 傾向於首先選擇低頻，就像 zigzag strategy 一樣。