CIFAR-10 Dataset Hands-on

22 November 2025 10:18

CIFAR# Canadian Institute for Advanced Research.

CIFAR-10 is a widely used benchmark dataset in computer vision, specifically for "image classification tasks" (CNN)
→ 10 labels or categories

Total images # 60,000
categories # 10
each category = 60,000/10 → 6000 images per class

The CIFAR-10 dataset

32 pixels x 32 pixels

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

0 airplane
1 automobile
2 bird
3 cat
4 deer
5 dog
6 frog
7 horse
8 ship
9 truck

Total no. of images = 60,000 across 10 classes/categories

32x32 (32 ↑, ←32→)

→ Training dataset # 50,000 → 50,000/10 = 5000/class

→ Validation/Testing dataset # 10,000 → 10,000/10 = 1000/class

Balanced dataset ⇒ each class has 6000 images

60,000/10 = 6000/class

The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

↓ same type of image can't be in two classes

↓ Automobiles don't include "big trucks"

CIFAR-10 dataset background

created by: Alex Krizhevsky et. al.

↓

from source of 80 million images → (tiny) for MIT

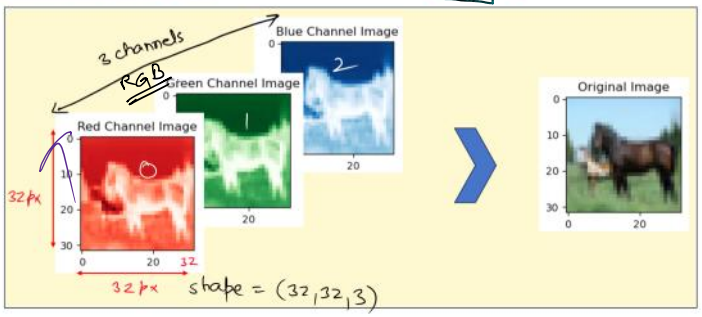
real world images from the web.

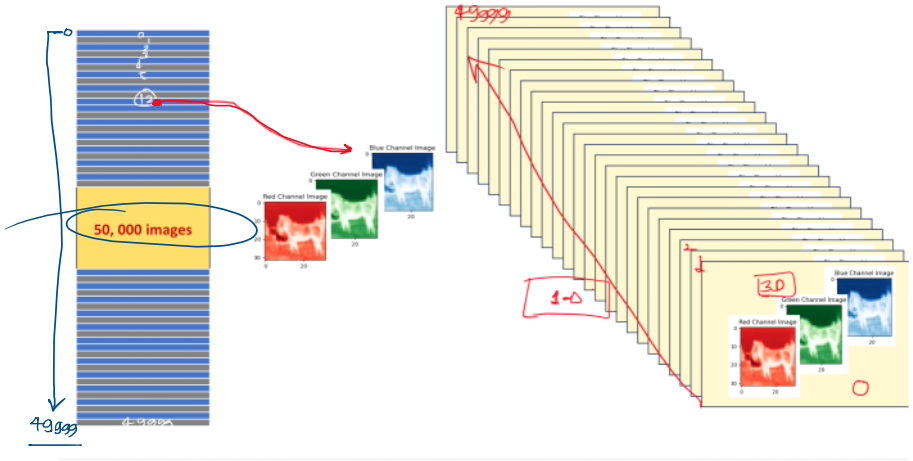
and then labelled it into 10 categories

Shape of the training images: (50000, 32, 32, 3)

3-dimensional view of the 13th image from the training dataset (out of 50,000 images)

13th (32, 32, 3)



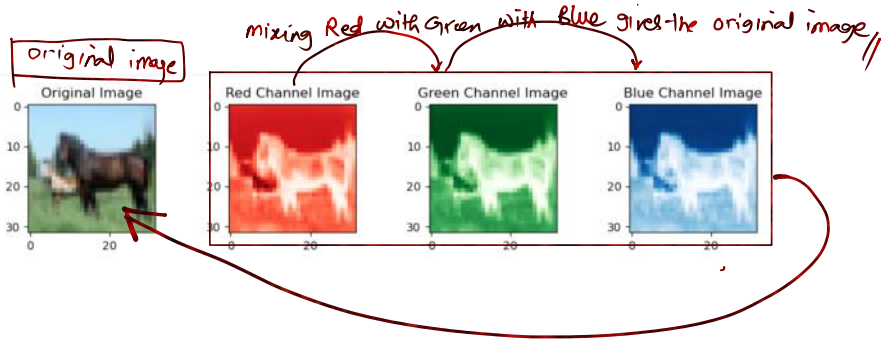


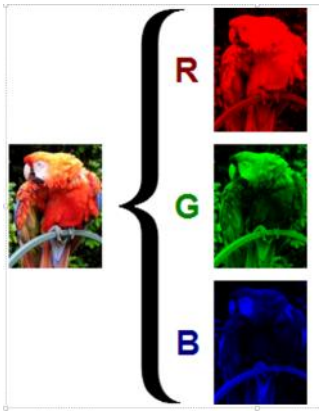

```
train_images.shape
```

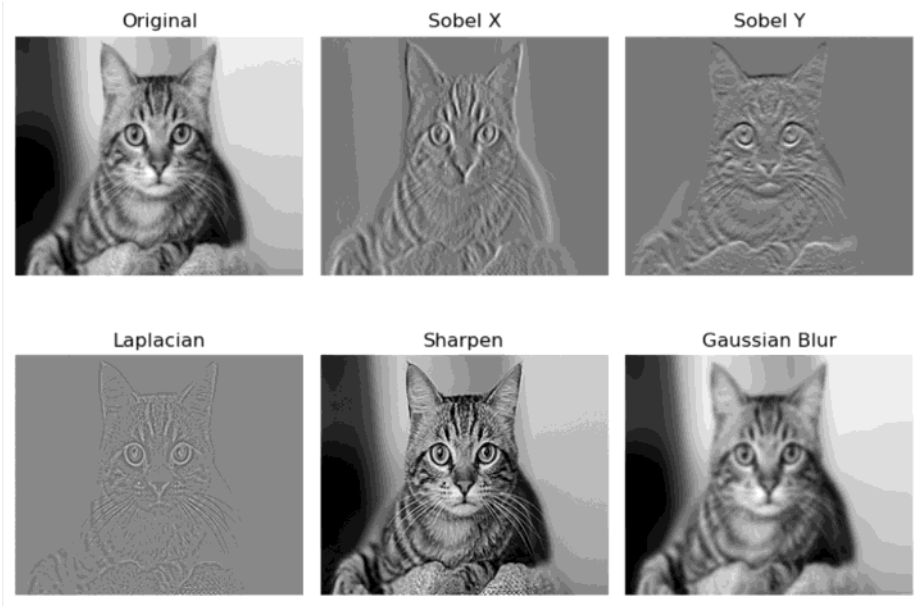
```
(50000, 32, 32, 3)
```

```
print("Dimension of training dataset is:", train_images.ndim)
```

```
Dimension of training dataset is: 4
```







UNDERSTANDING THE CONVOLUTION STEP (FILTERING)

SOBEL X vs SOBEL Y FILTERS

SOBEL operator is known to be classic edge - detection filter in image processing.

It works by convolving the image with two 3×3 kernels that approximate the 1st order derivative (gradient) in the horizontal (X) and vertical (Y) directions respectively

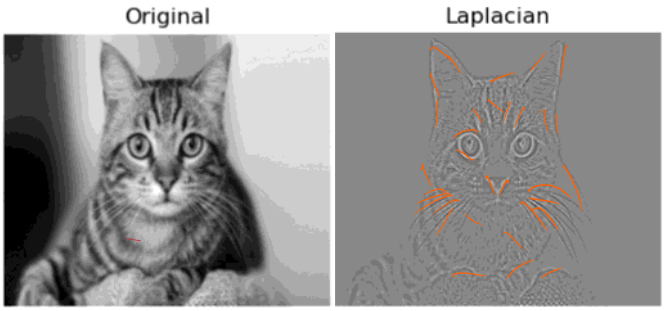
History: The SOBEL operator is named after Irwin Sobel — an American computer scientist

Transpose

SOBEL X: $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ SOBEL Y: $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

SOBEL X	SOBEL Y
– it detects vertical edges by emphasizing changes in the horizontal direction	– it detects horizontal edges by emphasizing changes in the vertical direction
– left column has –ve values and right has +ve values	– Top row is –ve and bottom row is +ve.
– It highlights the regions where pixel intensities changes left to right	– It highlights regions where pixels intensity changes from top to bottom

LAPLACIAN FILTER



– it is sometimes called laplacian operator

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

∇^2 ↓ Laplacian operator

– it measures rate of change of gradients → second order derivative

$\frac{\partial}{\partial x}, \frac{\partial}{\partial y}$ 1st order derivative → gradient

$\frac{\partial}{\partial x}\left(\frac{\partial}{\partial x}\right) \rightarrow \frac{\partial^2}{\partial x^2}$ $\quad \frac{\partial}{\partial y}\left(\frac{\partial}{\partial y}\right) \rightarrow \frac{\partial^2}{\partial y^2}$ rate of change

– In a region, it highlights the area where pixels intensity changes → edge in fine details, and sometimes noise too

Note: Unlike SOBEL (directional), the laplacian is isotropic

↓

it detects edges in all directions simultaneously

Two commonly used Kernels (filters) for Laplacian filters:

1. 4-connected neighbors:

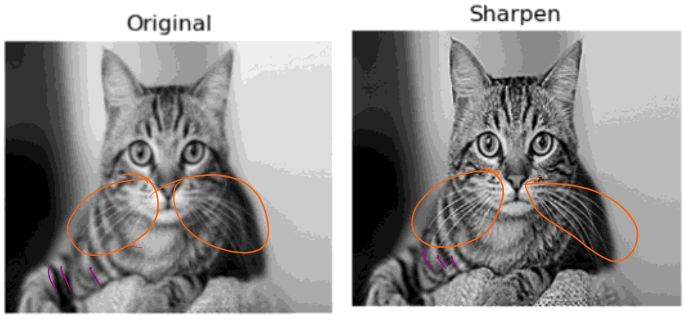
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}_{3\times3}$$

2. 8-connected neighbors:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Note: laplacian filter is very sensitive to Noise!

SHARPEN FILTER



Observations

1. Edges such as fur lines, whiskers, paws, eyes are more pronounced.
2. it provides the impression of more detail even though no new details were added.

- sharpen filter enhances 'edges' and fine details by making the image crispier and by increasing contrast at the boundaries.

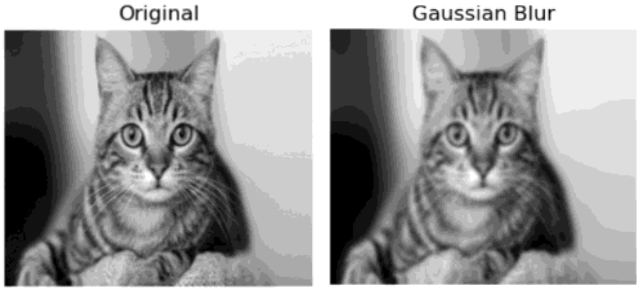
Common Filter/Kernel

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

↳ central value of 5 intensifies the current pixel and neighbors subtract the edges' context → basically emphasizing the boundaries.

sharper smooth

GAUSSIAN BLUR FILTER



- smoothens the image by averaging pixels based on Gaussian distribution
- it is used to reduce the noise, and smooth features before the edge detection

common filter example:

3×3 kernel ($\sigma \triangleq 1$) : $\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$

Filter	Purpose
SOBEL X	it detects vertical edges
SOBEL Y	it detects horizontal edges
Laplacian	all-directions → edge detection
sharpen	edge enhancement
Gaussian	Noise reduction.

TensorFlow > API > TensorFlow v2.16.1 > Python

Was this helpful?

tf.keras.layers.Conv2D

View source on GitHub

2D convolution layer.

Inherits From: `Layer`, `Operation`

View aliases

#filters

size of the convolution window

Conv2D(32, (3,3))

```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

(image classification related tasks)

filters → No. of output channels → Feature Maps [one filter → one feature map]

kernel_size → spatial size of filter → Std. size is small grid # (3x3)

strides=(1, 1) → stride (1,1) → filter movement's step size across height & width

shape: [batch, H, W, C] or [batch, C, H, W]

C → channels_last

C → channels_first

kernel_size can be square or non-square grid.

Conv2D(32, (3,3))

or

Conv2D(32, (3,1))

```
padding='valid',
```

↳ by default padding is 'valid'

↓

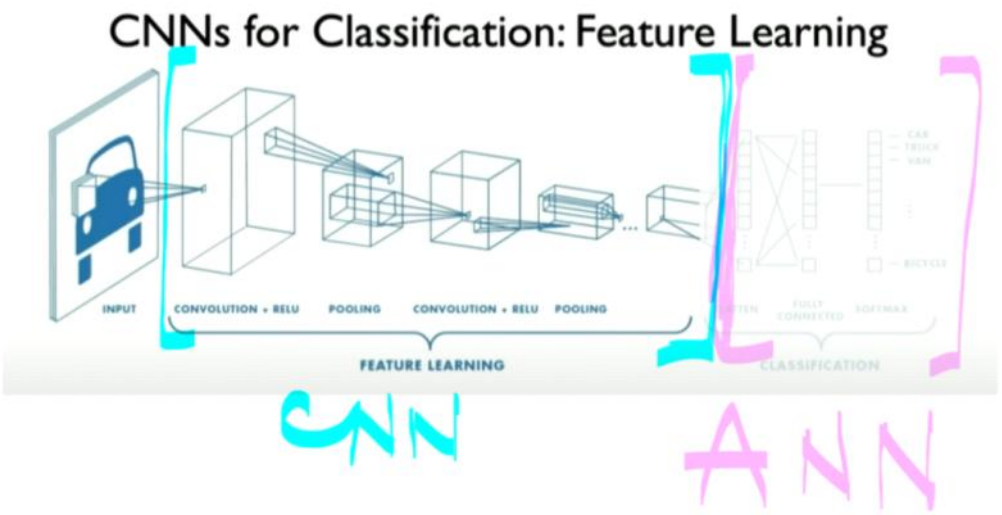
(No padding)

padding = 'same'

↳ zero padding to maintain the output size equal to input size

feature map size (output) = Input image size (input)

padding same means



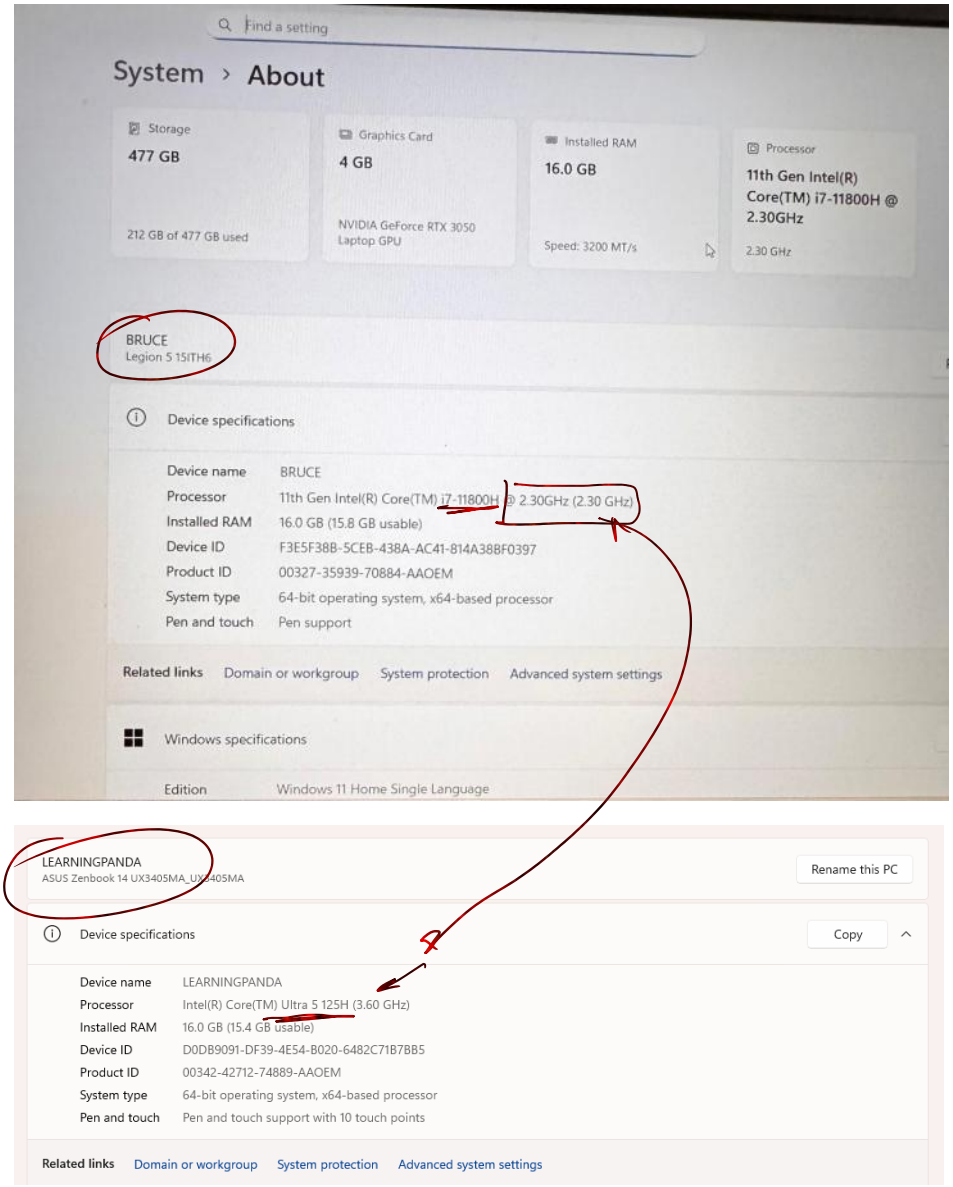

```
history_01 = base_model.fit(train_images, train_labels, epochs = 50, validation_data=(test_images, test_labels))
```

```
Epoch 1/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.3123 - loss: 2.9519 - val_accuracy: 0.4829 - val_loss: 1.4312
Epoch 2/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.5166 - loss: 1.3621 - val_accuracy: 0.5465 - val_loss: 1.2675
Epoch 3/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.5650 - loss: 1.2288 - val_accuracy: 0.5933 - val_loss: 1.1636
Epoch 4/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 7ms/step - accuracy: 0.6161 - loss: 1.0847 - val_accuracy: 0.5876 - val_loss: 1.1713
Epoch 5/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 12s 8ms/step - accuracy: 0.6522 - loss: 0.9927 - val_accuracy: 0.6401 - val_loss: 1.0564
Epoch 6/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 12s 7ms/step - accuracy: 0.6782 - loss: 0.9280 - val_accuracy: 0.6290 - val_loss: 1.0882
Epoch 7/50
```

'Legion'
Gaming laptop

```
history_01 = base_model.fit(train_images, train_labels, epochs = 50, validation_data = (test_images, test_labels))
```

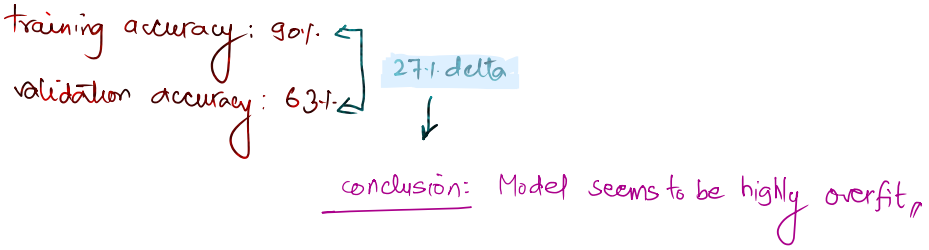
```
Epoch 1/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 52s 29ms/step - accuracy: 0.1200 - loss: 2.9709 - val_accuracy: 0.3599 - val_loss: 1.7135
Epoch 2/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 38s 24ms/step - accuracy: 0.4096 - loss: 1.6014 - val_accuracy: 0.5230 - val_loss: 1.3263
Epoch 3/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.5367 - loss: 1.3065 - val_accuracy: 0.5538 - val_loss: 1.2627
Epoch 4/50
```




```
Epoch 31/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 42s 27ms/step - accuracy: 0.8961 - loss: 0.3054 - val_accuracy: 0.6291 - val_loss: 2.1242
Epoch 32/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 52s 34ms/step - accuracy: 0.8968 - loss: 0.3048 - val_accuracy: 0.6315 - val_loss: 2.1665
Epoch 33/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 52ms/step - accuracy: 0.9010 - loss: 0.3013 - val_accuracy: 0.6300 - val_loss: 2.1951
Epoch 34/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 46s 30ms/step - accuracy: 0.9016 - loss: 0.2964 - val_accuracy: 0.6343 - val_loss: 2.1024
Epoch 35/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 45s 29ms/step - accuracy: 0.8966 - loss: 0.3116 - val_accuracy: 0.6321 - val_loss: 2.3190
Epoch 36/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 57s 36ms/step - accuracy: 0.9079 - loss: 0.2803 - val_accuracy: 0.6280 - val_loss: 2.2500
Epoch 37/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 136s 87ms/step - accuracy: 0.9057 - loss: 0.2945 - val_accuracy: 0.6263 - val_loss: 2.4384
Epoch 38/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 150s 96ms/step - accuracy: 0.9080 - loss: 0.2833 - val_accuracy: 0.6230 - val_loss: 2.3281
Epoch 39/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 189s 87ms/step - accuracy: 0.9076 - loss: 0.2885 - val_accuracy: 0.6310 - val_loss: 2.2641
Epoch 40/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 177s 113ms/step - accuracy: 0.9139 - loss: 0.2635 - val_accuracy: 0.6287 - val_loss: 2.3339
Epoch 41/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 204s 114ms/step - accuracy: 0.9116 - loss: 0.2736 - val_accuracy: 0.6281 - val_loss: 2.4073
Epoch 42/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 238s 137ms/step - accuracy: 0.9093 - loss: 0.2829 - val_accuracy: 0.6317 - val_loss: 2.4319
Epoch 43/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 218s 108ms/step - accuracy: 0.9121 - loss: 0.2803 - val_accuracy: 0.6260 - val_loss: 2.4315
Epoch 44/50
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 123s 57ms/step - accuracy: 0.9162 - loss: 0.2635 - val_accuracy: 0.6247 - val_loss: 2.5641
Epoch 45/50
```

Base Model Run #1

Epochs # 50



EARLY STOPPING - will discuss about it