To check repo → git remote -v.

For existing

heroku git : remote -a $\underbrace{\text{thawing-inlet-61413}}_{\text{name.}}$

IMAGE DATA AUGMENTATION WITH KERAS

→ Solve overfitting
→ When data is small
→ Let's say we need to differentiate between cats & dogs.

TASK 1 - IMPORT LIB

TASK 2 - Rotation

generator = tf. keras. preprocessing .image. ImageDataGenerator(

degree at which rotation should be done. Here -40 to 40 It is random.

→ rotation_range = 40

this function gives an iterator

```
x,y = next (generator. flow_from_directory
        ('images', batch_size = 1))
```

no. of images

To display

```
plt. imshow (x[0].astype ('unit8'));
```

TASK 3 - Width & Height Shifts

Again imageDataGenerator.

```
generator = tf.keras.preprocessing.image.ImageDataGenerator
    (
    width_shift_range = ~~40~~ [-100, -50, 0, 50, 100]
```

This is list of possible values not range. So with width can shift ~~to~~ 5 different points

This time we will use list, we can use previous ways also

```
    height_shift_range = [-50, 0, 50]
    )
```

Ismein jo age pixel generate karne ho toh nearest picture value repeate kar dete hain.

TASK-4 Brightness

Saame Generator.

```
generator = tf.keras.preprocessing.image.ImageDataGenerator
(
    brightness_range = (0.5, 2.0)
```

THis time we used tupple. It means we have defined range unlike previous where we gave explicit values

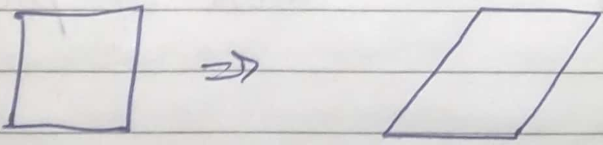
0.5 → Brighness 1/2.
2.0 → 2x Brightness
1 → Default or natural Brightness of image.

TASK-5 SHEER Transformation

Same Generator

```
generator = tf. keras. . . . . (
    shear_range = 40
)
```

It also ~~s~~ means rotation. -40° — 40° angle sheer rotation means bottom corner remain fixed but top corner moves.



Sometimes they may not give desired examples i.e the final result may not look like the req class here cat.

TASK 6 - zoom (Simple zooming in or out)
Same generator.

zoom_range=[0.5, 1.2]

1 - Default zoom
0.5 → zoom 1/2
1.2 → zoom × 1.2
We used list so we have 2 explicit zooms & not range

zoom_range = [0.5, 1.5] → List not tupple

isko hum ese bhi likh sakte hai
zoom_range = 0.5

TASK 7 Channel Shift.
Same generator
channel_shift_range = 100

Here -100 to 100 will be added to all RGBs

How to see change

x.mean() → mean of augmented picture value

→ np.array(Image.open(image_path)).mean()

original mean

TASK 8 : Flips.

(horizontal_flip = True,
vertical_flip = True

)

Random peega. Kabhi horizontal Kabhi vertical
Kabhi dono Kabhi Koi bhi nhi.

TASK 9 : Normalization

A → feature wise Normalization.
B → sample wise "

A → Feature wise
same generator with some ~~to~~ loading data
in x_train, y_train format. (See Code)

(
Your values will be updated and mean value will be subtracted from all → featurewise_center = True
featurewise_std_~~no~~normalization = True.
→ Everything divided by std normalization value
generator. fit (x_train). → Fit karna jaruri for normalization feature wise.

B → Samplewise

samplewise_center = True
samplewise_std_normalization = True

__/__/__

TASK 10: Rescale & Preprocessing function

Same generator

(rescale = 10 → Everything multiply by 10

= 1 (we will use no rescale)

→ = 1/255 ⇒ Everything divided by 255.

Information only

→ (preprocessing_function).

Any function It takes 3-D tensor or ~~not~~ numpy array & will return the same.

Actual code

preprocessing_function = tf.keras.applications.mobilenet_v2.preprocess_input.

TASK 11: Using in Model Training

Same generator.

we have generator

```
(
    preprocessing_function = same as above
    horizontal_flip = True,
    rotation_range = 20
)
```

Let's create a model

MobilenetV2

First layer

2nd layer:

```
model = tf.keras.models.Sequential([
    tf.keras.applications.mobilenet_v2.MobileNetV2(
        include_top = false, input_shape = (32,32,3)
        pooling = ~~avg~~ 'avg'
    ]), tf.keras.layers.Dense(10, activation = "softmax")
```

))   __/__/__

Now compiling   model.compile(

loss = 'sparse_categorical_crossentropy',

lossfunction. The specific fuction is used here because we are not one hot encoding but usig 1-9 values

optimizer = "adam"
metrics = ['accuracy']
)

_ = model.fit(

generator.flow(x_tragin, y_train, batch_size = 32),

epochs = 1 , steps_per_epoch = 10

)

How many batches are going to be in one epoch

So, 10 batches of batch-size 32 will conclude 1 epoch i.e our training will be complete.

We are not actualy trying to get a well trained model. We can use much other values also for increased accuracy