IMAGE CLASSIFICATION
USING CIFAR-10 DATASET

DEEP LEARNING PROJECT

INTRODUCTION :

We chose Image classification as the subject of our project. the dataset we adopted is CIFAR-10 dataset it is a collection of images of 10 different classes like cars, birds, dogs, ships, etc.

The idea of the project is to build an image classification model that will be able to identify what label the input image belongs to.

about the dataset we choose :

CIFAR-10 is a very popular computer vision dataset. This dataset is well studied in many types of deep learning research for object recognition.
This dataset consists of 60,000 images divided into 10 target classes, with each category containing 6000 images of shape 32*32*3, 3 is for the RGB values. This dataset contains images of low resolution (32*32*3), which allows researchers to try new algorithms. The 10 different classes of this dataset are:

1. Airplane
2. Car
3. Bird
4. Cat
5. Dog
6. Ship

etc..

What we did (steps)

Understanding the original image dataset:

First of all we used Keras model and libraries, we declared Keras sequential model and added the layers that will be explained as well in the next pages.

Since this project is going to use CNN for the classification tasks, the original row vector is not appropriate. In order to feed an image.

We reshaped the train set and the data set with np library to 10 categories . The line is bellow:

y_train = np_utils.to_categorical(y_train, num_classes).

x_train = np_utils.to_categorical(x_train, num_classes).

Using np.mean and np.std we normalized our data.

For the next step in the model which is the first layer with 32 filters and 2x2 kernel.

PRE-PROCESS FUNCTIONS :

NORMALIZE: (PRE-PROCESS FUNCTION)

normalize function takes data, x, and returns it as a normalized Numpy array.

And By applying Min-Max normalization, **the original image data** is going to be **transformed in range of 0 to 1.**

Also we used ELU activation function which takes an input value and outputs a new value ranging from **0 to infinity and the we used BatchNormalization to normalize the values between 0 to 1**. When the input value is somewhat large, the output value increases linearly. However, when the input value is somewhat small, the output value easily reaches the max value 0. We normalized the matrecies of the layer because we want our model to work faster. So the complexity will we improved.

CNN MODEL :

CONV2D :

- For each layer we defined a Kernel and strides ,also padding in order preventing loss of data. The conv2d layer applies this list of methods that we learned at class like : the kernel size, the number of filters which is 32 in the first layer.
- Also the strides which is the kernel jump in horizontal way and vertical way .
- We also added kernel_regularization to avoid OV(Over Fitting) using I2 vector norm with weight decay parameter these values will be added to the loss function.
- This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.
- And the input for the first layer that we reshaped and normalized before.
- Also we look at the formula of calculating the params in each layer which is it = ((m*n*d)+1)*k).
- M = filter width , n = filter height , d = number of previous filter/channels , k = number of filters in the current layer. For example in the first conv2d layer the number of params is ((3*3*3)+1)*32) = 896.

ACTIVATION & NORMALIZATION

- We used Elu activation function to the output of our first layer which is by the way will be normalized in the next layer.
- How we got 128 parameters in the batchNormalization? The last dimension is 32 and by default we had 4 parameters which will be multiplied by the last axis value.

MAXPOOLIG2D:

- Max pooling operation for 2D spatial data.
- Down samples the input representation by taking the maximum value over the window defined by pool size for each dimension along the features axis. The window is shifted by strides in each dimension. The resulting output when using "valid" padding option the meaning that we didn’t use padding. The formula for the shape (number of rows or columns) of: output shape = (input shape – pool size + 1) / strides) if there is no padding.
- This output will be activated with RELU activation function and will be normalized in the next layer.
- The resulting output shape when using the "same" padding option is: output shape = input shape / strides.

BACK PROPAGATION

- As we learned in the class in order to find the best weights and biases intending to minimize our cost function and claiming a good accuracy to the model by using optimizers that will help us to find the best weights and biases .
- Back propagation is only used in the train with Forward propagation of course.

ADDING HIDDEN LAYERS:

To complete our model, feed the last output tensor from the capacitive base to one or more dense layers to perform classification. Dense layers take vectors as input (which are 1D), while the current output is a three-dimensional tensor. First, flatten (or comb) the 3D output to 1D, then add one or more dense layers on top. CIFAR has 10 output classes, so you use a final dense layer with 10 ports using at the last layer Softmax function that generates probabilities between 1 to 0, we take the bigger probability which represents our label and we update our cost function that declared in the model compile under the name categorial cross entropy that we have learned in the class -tf.reduce_sum(y_ * tf.log(y) when the probabilty of our prediction is high we get a small number, how the cost function works? In order to answer this question we have to define the term of back propagation of our NN in the next slide.

DATA AUGMENTATION

- From keras.processing we imported the imageDataGenerator in order to get a different shapes of same image meaning that every time we flip the picture in horizontal way.

Data augmentation is used in order to enhance the model performance for example if we want to classify a cat picture in the test, let's say the same picture or a picture that is so similar to the picture that the model saw in the train but with a small rotation or flipped in horizintal way , the model will fail or he will get right but with a low probability in softmax affecting the cost function that will be increased. And that not what we are aiming to. So Data Augmentation will help our model to see same pictures that modified to enhance the performance (or in another word to enhance the train).

Then we will fit the Image Data Generator on x train.

OPTIMIZER

- We will use Keras optimizer with learning rate 0.001 and decay 1e-6
- The decay terms mean that we will change the learning rate and decrease it helping the optimizer while training in order to increase the model accuracy.
- We used RMS optimizer of Keras.
- This optimizer will be used of course in compiling the model.

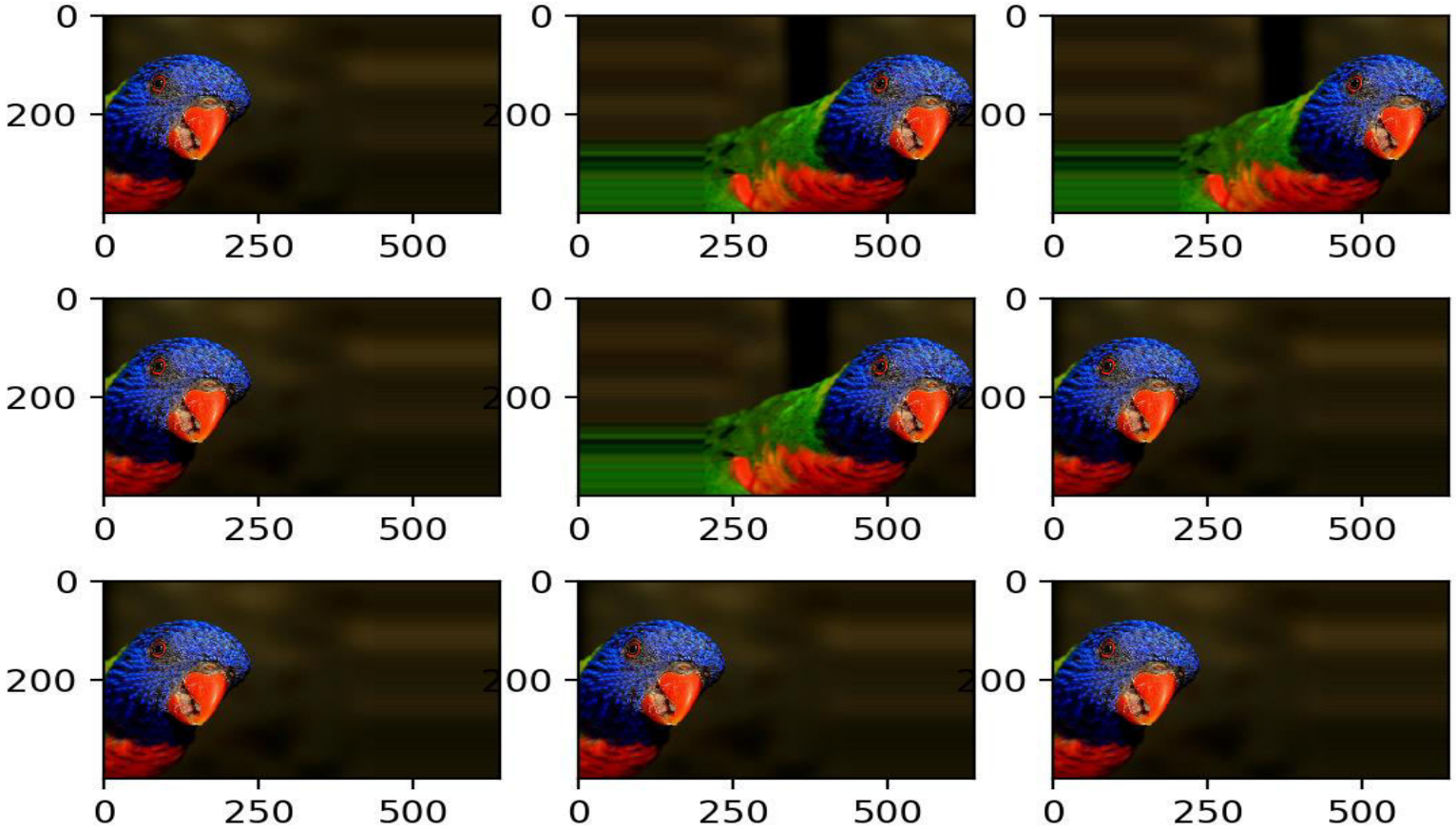
COMPILING WITH KERAS

- We will compile our model with cross entropy categorical loss function using the optimizer that was explained in previous slide.
- Also we had the accuracy metrics to print the accuracy while training.
- Compiling the model is like declaring a session like we did in the previous assignment of this course.

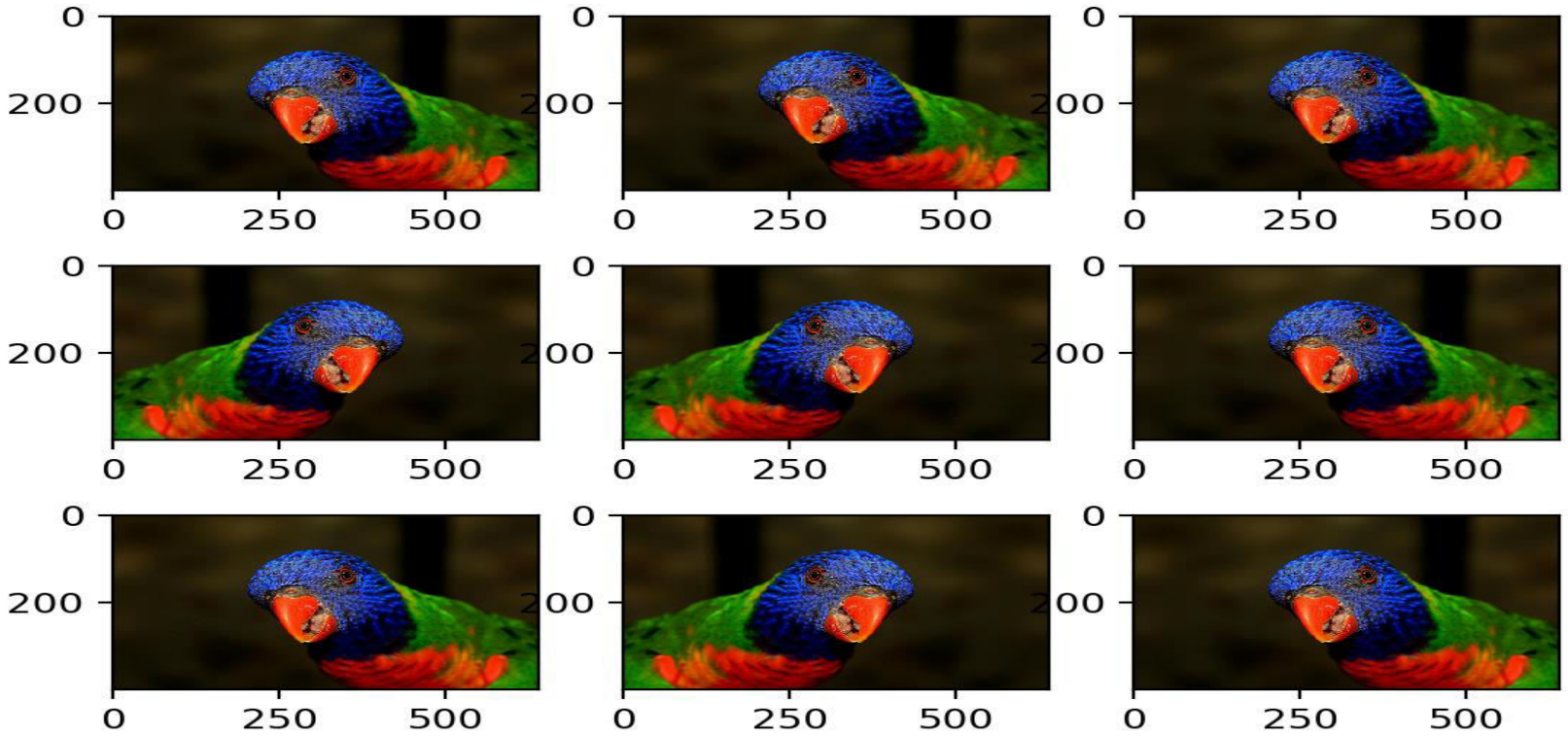
MODEL FIT_GENERATOR

- Using the flow function of image data generator returns iterator of one batch augmented images every iteration.
- The batch size is 64.
- We start training our model with the data we have prepared, like and the train set will be the image data generator in Data augmentation.
- We declared the batch size to be 64 so we have 50,000/64 = 781 image every step in epoch because we used data augmentation.
- Image data augmentation allows both random pictures and modified image to be used in training.
- The model will train 125 times and every time will train on 781 images.
- Verbose = 1 showing us the animated progress bar.
- Validation data is the test data set to get the accuracy of the model in test case.
- Calls back: calls the learning schedule function to update the learning rate after 75 and 100 epoch.

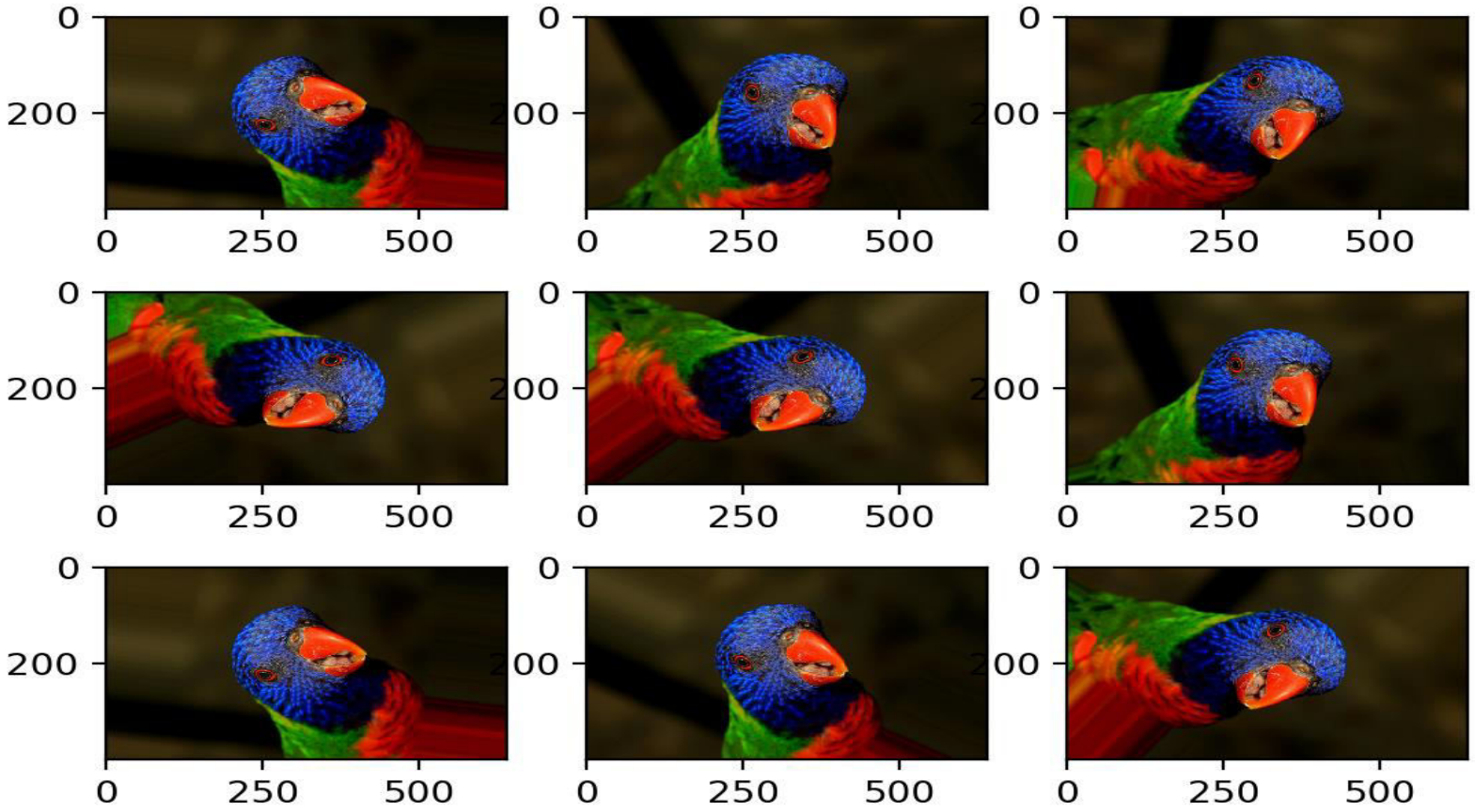
EXAMPLE OF DATA AUGMENTATION USING WIDTH SHIFT



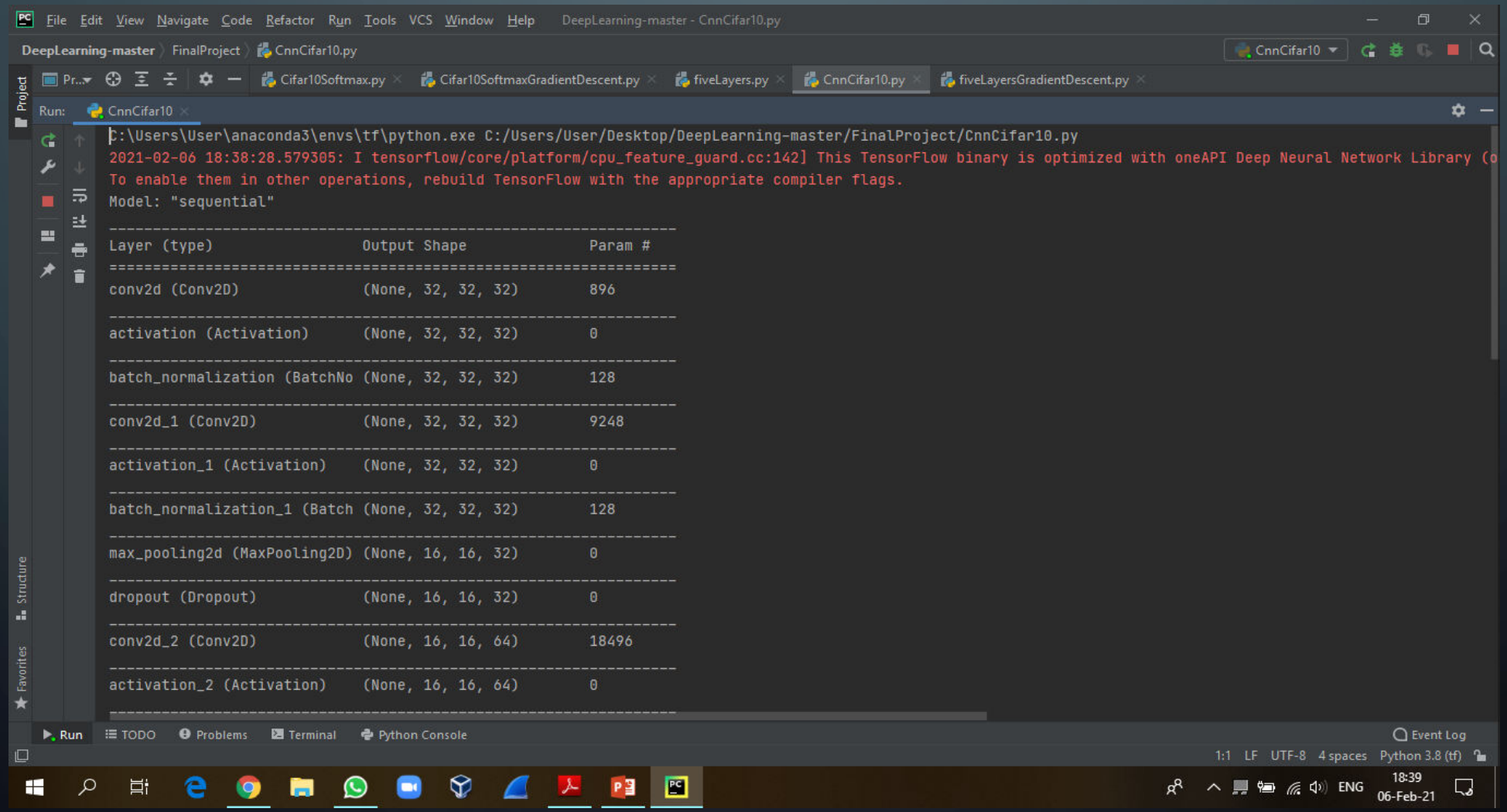
EXAMPLE OF DATA AUGMENTATION FLIP HORIZONTAL



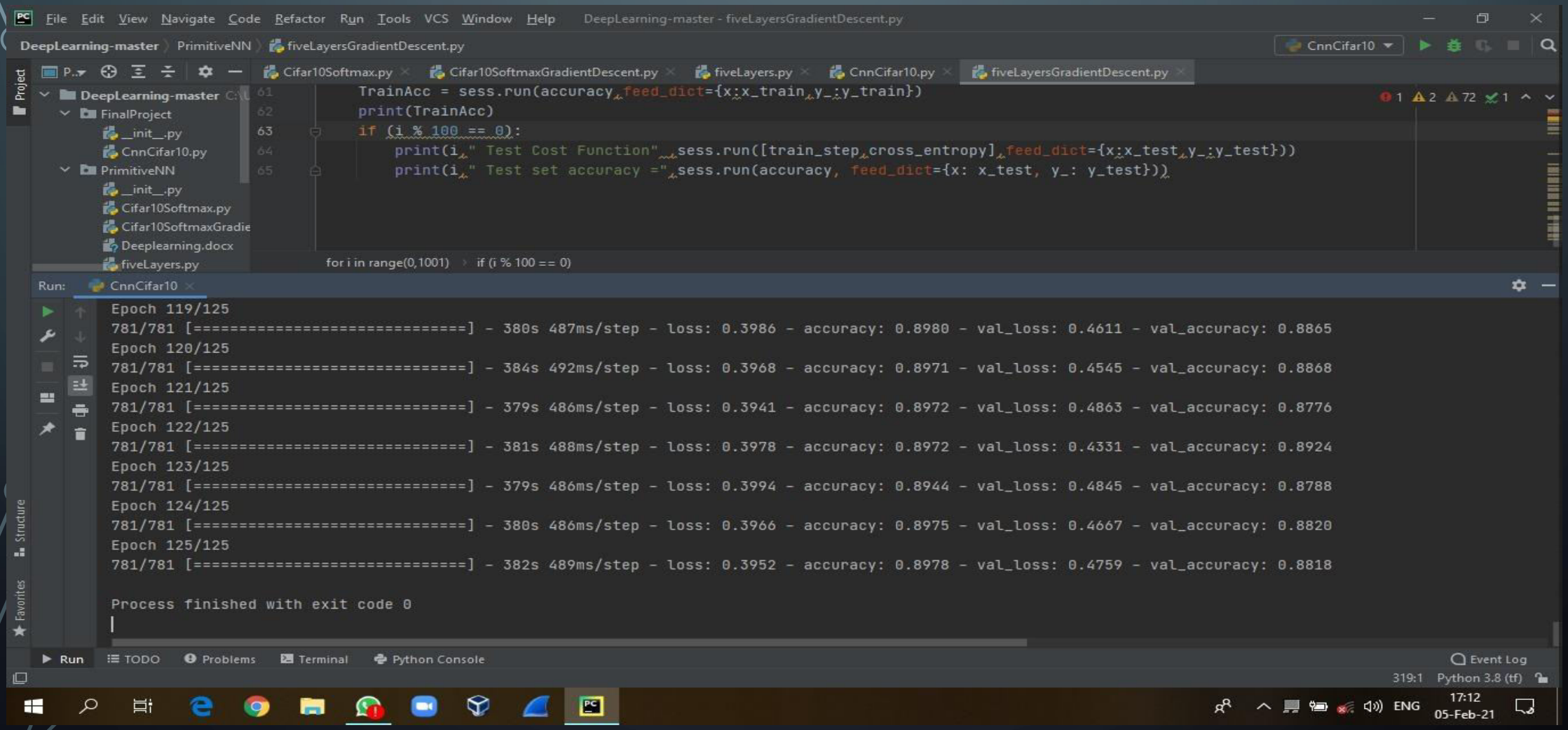
DATA AUGMENTATION ROTATION



THE ARCHEICHT OF OUR MODEL :



FINALLY!!!! THE ACCURACY IS 88% :



CREATED BY :

AIMAN YOUNIS - 207054354

ZHRAA ZAID – 206604027

HTTPS://GITHUB.COM/AIMANYOUNISES1/DEEPLEARNING