Project Idea:

Python is a versatile programming language able to quickly create a variety of apps and programs for a wide range of tasks. Artificial intelligence and machine learning is an area of computer science that has been quickly emerging for various applications from investing to medical advising . Python has the ability to employ machine learning through the TensorFlow library. We want to learn and use Python's machine learning tools like TensorFlow to learn how to use,train and test machine learning models. To do this we use a labeled kaggle dataset of images consisting of airplanes labeled ‘planes’ or no airplanes labeled ‘no planes’.

After sorting the images of planes and non-planes in the python environment, we created a Convolutional Neural Network which is a type of learning model that can specifically process data like images. The model analyzes the photos, pixel by pixel, and is trained to learn what a plane looks like based on certain patterns such as colors, edges, shapes, etc. After it learns this, the model can receive new images and can predict if the photo is a plane or non-plane based on previous knowledge.

Code Break down:

Libraries used:

- Numpy
- Pandas
- Tensorflow
- Sklearn
- Matplot
- Itertools
- Shutil
- Os
- Glob
- Random

Dataset: https://www.kaggle.com/datasets/rhammell/planesnet

Image sorting:

First line of code sets the file path to the correct directory containing the dataset. Then using the glob command from the glob library we open all the images into a list given the parameters. The ‘0*’ parameters opens all files starting with ‘0’ aka the non-plane images into a list. Then we call the command with ‘1*’ as the parameter to save all the plane image paths to a separate list.

We use the Os library to create directories for our data sets. We also remove 10,000 images from the ‘not planes’ list to shorten the training time.

Next we sort the two lists into training sets, validation sets, and test sets by iterating through the lists based on our specific ratios for the sets (training is .7, validation is .1, testing is .2).

Image Processing:

We process the data sets, by using the Image Data Generator to scale all the rgb values of the pixels to a scale between 0 and 1 by dividing by 255. We save these new datasets as generator objects. Next, using our ImageDataGenerator objects for the training, validation, and testing sets we call 'the flow_from_directory()' method to format our images to be processed in batches. We give the method parameters for the saved file path for the images, the size of the image (20x20 pixels) , amount of images we want to processed per batch (32), and our 'class_mode' or how we want to sort the images, in our case it's 'binary'/plane or not plane.

We then make a function using matplotlib to display a sample of images stored in the train generator and their labeled values.

When training our model on the train and valid sets, we have to make sure that they are weighing equally for the planes and non planes. We use the compute_class_weight() command.

Parameters,

- 'Balanced' ; makes it so differing class frequencies in the set have same cumulative weight
- Classes = np.unique(train_generator.classes); gets all classes found in the set (0 and 1)
- Y = train_generator.classes; all the labeled classes

Returns list of each class and their respective weights. We save this into a dictionary to be used later on.

Model:

Next we create our model as a sequential object. This means per processing step, the output for that step will be used in the next step.

First we do a convolution layer using the Conv2D() method from Keras. The convolution goes through the image with parameters given and converts the pixels into a different output. As a result it sharpens the image.

For our Conv2D call we use the parameters:

- Filters = 32 ; (perform convolution 32 times)
- Kerne_size = (4,4) ; area size of pixels to perform each convolution
- Activation = 'relu' ; 'relu' converts all negative outputs to 0, helps with processing
- Padding = (20,20) ;since our filtering combines pixels to a value, we add padding to sustain the 20 by 20 image resolution.
- Input_shape = (20,20,3) ; defines images it takes in as 20by 20 and using RGB

Next we use MaxPool2D() method in keras with parameters 'pool_size' =(2,2) and 'strides' =2 which replaces each element per 2by2 area of the output with the max value of that respective area.

We run this same processing using Conv2D and MaxPool2D two more times in our model, each time increasing the amount of filters.

Then we call the Flatten() method to turn our (20,20,3) output into an array.

Then we call the dense function which creates a layer of neural network with 1256 nodes(first parameter) We also set the activation parameter to ‘relu’ to turn negative values to 0 to make it easier for the model to process.
Then we drop half the nodes using the dropout() method, to make sure our model isn’t overcompensating. So we only relay from 128 of the nodes for the next layer.(based on statistics for our type of model).
Then we call a dense function with 1 node as the first parameter, and the
Activation parameter as ‘sigmoid’ to give us a binary value (0 or 1/plane or no plane)

Model Compilation and results:

Then we compile our model with a compile command.
For parameters,
Optimizer = ‘adam’; we used ADAM optimizer because it shortens the training time, and requires low amount of memory to be run on laptops
Loss = ‘binary_crossentropy’; this is the lost function used in binary sorting tasks
Metrics = [‘accuracy’’]; this will have the model keep track of the accuracy

We then use the fit command with parameters of our train_generator object, valid_generator object to train the model of these data. We also give the parameter of epochs = 20, to train the model on the set 20 times, and give the class_weights_dicts(from earlier) so it is trained on a balancing weighing for each of their classes.
We can save all the runs onto a variable - history.

Then using matplotlib we plot the accuracy of the training set and validation set over each epoch.

Then we use the evaluate command to test the model with the test_genetor set, and actually use our model to discern planes with a test and get accuracy and loss values.

Then We use the predict command with the test_generator to get the predicted values for each of the elements and put that as a parameter along with the actual labeled values for the classification report().

We then use the y_pred_binary and y_true values again as parameters for a command to create a confusion matrix. We use the sns library to do some minor color formatting.

Model is finished and saved.