Task 1

We can use Poisson distribution that describes the probability of an event happening k number of times within a given interval of time.

$$P(X \leq x) = \int_0^x \lambda e^{-\lambda x}\, dx$$
$$= [-e^{-\lambda x}]_0^x$$
$$= \left(-e^{-\lambda x}\right) - \left(-e^{-\lambda(0)}\right)$$
$$= 1 - e^{-\lambda x}$$

Plugging in 60%,

$$P(X \leq 3) = 0.60$$
$$1 - e^{-\lambda(3)} = 0.60$$
$$e^{-\lambda(3)} = 0.40$$
$$-3\lambda = \ln(0.40)$$
$$\lambda = -\frac{\ln(0.40)}{3}$$
$$\lambda = 0.3054$$

The λ here defines the mean rate, that is 0.3054 plane observed per minute.
Using the λ solved,

$$P(X \leq 1) = 1 - e^{-\lambda x}$$
$$= 1 - e^{-0.3054}$$
$$= 0.2632$$

Hence, the probability of spotting a plane within 1 minute is around 26.32%.

Task 2

Data Preprocessing and Model Building

i) Splitting the images

The dataset given contains 10 classes, each with 100 examples. To use it to train a model to classify the identity documents, we need to split them into training, validation, and testing sets. In this case the approach used is to draw random samples out from each class, with 70% in total as training sets, 15% each for validation and testing.

ii) Preprocessing

It is observed that the images of the identity documents collected are having various different backgrounds that might introduce noise and potentially deteriorate the model's learning in identifying the documents' class. In this case, a preprocessing step is proposed utilizing transfer learning. We attempt to segment the identity documents out from the full image by training a YOLO model using this dataset. This allows us to simplify our classification model, that will reduce computation in terms of training and inference.

Figure 1: Before preprocess

Figure 2: After preprocess

iii) Data Augmentation

Data augmentation is an important step to improve the performance of our model, and its ability to generalize. Since we have limited data, it helps to increase our dataset size. By implementing multiple augmentation techniques, we can reduce overfitting and make our model more robust. The augmentation techniques implemented include random horizontal and vertical flipping, rotation, zooming, setting random contrast and brightness, and injecting random gaussian noise.

iv) Model Building

Since we have preprocessed our data, we can reduce the complexity of our model. In this case, a simple Convolutional Neural Network (CNN) is built to classify the identity documents. This model contains 3 convolutional layers, with 64, 32, and 16 filters. From the input, we capture complex features and then reduce computation by parsing through smaller filters. These convolutional layers are each followed by a pooling layer to reduce the spatial dimensions of the feature maps. Then, we implement dropout layers which help prevent overfitting. Lastly, the fully connected layers deal with the high-level feature abstraction and finally return the predicted label.

The application

Our goal is to build an application with UI and backend services to serve predictions by wrapping them all in a Docker Compose configuration for easy deployment.

i) Model Container

To deploy our model, we can wrap it in a separated Docker container that acts as a shared volume for the other services. This container contains the model that we trained and exported. By building such container, we can easily replace and manage the models for deployment without interrupting the other services by rebuilding the container individually.

ii) Backend Service

The backend service needs to be able to handle fetching user input, using the model we trained to perform inference, and then return the result back to frontend. In this case, an inference API is built based on FastAPI. FastAPI is suitable in this case as it is optimized for performance and supports asynchronous request handling via async and await which is crucial in production setting. It is also easy to scale and containerize using Docker. The API will read the deployed model from the model container that we configured as a shared volume, and use it for inference. The inference API operates on a POST request method to receive the input image and send the prediction to frontend.

iii) User Interface (frontend service)

The UI is built using React. It features a simple form for the user to upload an image for classification. Upon submission, the frontend makes a POST request to our inference API in the backend service, and displays the prediction on the UI. While simple alternatives like Streamlit can be considered, React offers component-based architecture that allows us to break down the UI into reusable pieces. In this case, the image upload component for example, is built as a standalone component. This allows for easy modification and update in future, even reusing it in another application. It is also easy to expand the UI's functionality, simply by adding components.