Project code:

- ❖ INTRODUCTION:
  - ➢ Project Title: transfer learning-based classification of poultry diseases for enhanced health managemen

This code defines a convolutional neural network (CNN) using the Keras framework. Let's break down each part:

- `from keras.models import Sequential`: This imports the Sequential model from the Keras library, which allows us to build a linear stack of layers.
- `from keras.layers import Conv2D, MaxPool2D, Flatten, Dense, InputLayer, BatchNormalization, Dropout`: This imports various layer types from Keras that will be used to define the architecture of the CNN.
- `model = Sequential()`: This initializes a Sequential model, which is an empty neural network architecture.
- `model.add(Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1', input_shape=(255,255,3)))`: This adds a convolutional layer with 32 filters of size 3x3, using ReLU (Rectified Linear Unit) activation function. The 'same' padding is used to maintain the spatial dimensions of the input. This layer is named 'conv_1', and it expects input images of size 255x255 with 3 channels (RGB).

. model.add(MaxPooling2D((2, 2), name='maxpool_1'))`: This adds a max- pooling layer with a pool size of 2x2. Max-pooling reduces the spatial dimensions of the feature maps while retaining the most important information.

- This sequence of adding convolutional and max-pooling layers is repeated three more times with increasing numbers of filters (64, 128, 128), resulting in a hierarchical representation of the input images.

- `model.add(Flatten())`: This adds a flatten layer that converts the 3D feature maps into a 1D vector, preparing them to be input into a dense (fully connected) layer.

.`model.add(Dropout(0.5))`: This adds a dropout layer with a dropout rate of 0.5, which helps prevent overfitting by randomly setting a fraction of input units to zero during training.

-
-

- `model.add(Dense(128, activation='relu', name='dense_2'))`: This adds a fully connected (dense) layer with 128 units and ReLU activation function.

.`

model.add(Dense(7, activation='softmax', name='output'))`: This adds the output layer with 7 units (assuming a classification task with 7 classes) and softmax activation function, which outputs probabilities for each class.

- `model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])`: This compiles the model, specifying the loss function (categorical crossentropy), optimizer (Adam), and metrics to be evaluated during training (accuracy).

In summary, this code defines a CNN architecture for image classification with convolutional, max-pooling, dropout, and fully connected layers, suitable for processing RGB images of size 255x255 pixels.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_1 (Conv2D)             (None, 255, 255, 32)      896

 maxpool_1 (MaxPooling2D)    (None, 127, 127, 32)      0

 conv_2 (Conv2D)             (None, 127, 127, 64)      18496

 maxpool_2 (MaxPooling2D)    (None, 63, 63, 64)        0

 conv_3 (Conv2D)             (None, 63, 63, 128)       73856

 maxpool_3 (MaxPooling2D)    (None, 31, 31, 128)       0

 conv_4 (Conv2D)             (None, 31, 31, 128)       147584

 maxpool_4 (MaxPooling2D)    (None, 15, 15, 128)       0

 flatten (Flatten)           (None, 28800)             0

 dropout (Dropout)           (None, 28800)             0

 dense_2 (Dense)             (None, 128)               3686528

 output (Dense)              (None, 7)                 903

=================================================================
Total params: 3928263 (14.99 MB)
Trainable params: 3928263 (14.99 MB)
Non-trainable params: 0 (0.00 Byte)
```

Using model.summary(), we can view all the layers in the CNN model.

```
[ ]  model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
         filepath="model_cnn.h5",
         monitor='val_accuracy',
         mode='max',
         save_best_only=True,
         verbose=1)
```

The provided code snippet configures a ModelCheckpoint callback in TensorFlow/Keras, which is a useful tool for automatically saving the best model during training. By specifying parameters such as the file path to save the model, the monitored metric (in this case, validation accuracy), and whether to save only the best model observed, the callback ensures that the most optimal model is preserved. This is particularly beneficial for preventing overfitting and ensuring that the model generalizes well to unseen data.

Additionally, the verbosity parameter controls the level of output displayed during training, allowing users to monitor the process.

```
[ ]  model.compile(loss='categorical_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

The provided code snippet compiles a Keras model using the Adam optimizer and categorical cross-entropy loss function, which is commonly used for multi-class classification tasks. Additionally, the accuracy metric is specified to monitor the performance of the model during training and evaluation. This configuration prepares the model for the training process, where it learns to minimize the specified loss function using the optimizer while maximizing the chosen metric to improve its classification accuracy.

Running and Evaluating the model

```
[ ]  history_cnn = model.fit(x=train_gen, epochs=40, verbose=1, validation_data=valid_gen,
                     validation_steps=None,  shuffle=True, callbacks = [model_checkpoint_callback])

    Epoch 1/40
    21/21 [==============================] - ETA: 0s - loss: 1.9391 - accuracy: 0.1682
    Epoch 1: val_accuracy improved from -inf to 0.14286, saving model to model_cnn.h5
    21/21 [==============================] - 23s 639ms/step - loss: 1.9391 - accuracy: 0.1682 - val_loss: 1.8991 - val_accuracy: 0.1429
    Epoch 2/40
    /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning:

    You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_mode

    21/21 [==============================] - ETA: 0s - loss: 1.8263 - accuracy: 0.2396
    Epoch 2: val_accuracy improved from 0.14286 to 0.33333, saving model to model_cnn.h5
    21/21 [==============================] - 13s 630ms/step - loss: 1.8263 - accuracy: 0.2396 - val_loss: 1.7282 - val_accuracy: 0.3333
    Epoch 3/40
    21/21 [==============================] - ETA: 0s - loss: 1.6822 - accuracy: 0.3408
    Epoch 3: val_accuracy improved from 0.33333 to 0.35714, saving model to model_cnn.h5
    21/21 [==============================] - 14s 656ms/step - loss: 1.6822 - accuracy: 0.3408 - val_loss: 1.6533 - val_accuracy: 0.3571
    Epoch 4/40
    21/21 [==============================] - ETA: 0s - loss: 1.4567 - accuracy: 0.4688
    Epoch 4: val_accuracy did not improve from 0.35714
    21/21 [==============================] - 14s 660ms/step - loss: 1.4567 - accuracy: 0.4688 - val_loss: 1.6986 - val_accuracy: 0.3571
    Epoch 5/40
    21/21 [==============================] - ETA: 0s - loss: 1.2583 - accuracy: 0.5580
    Epoch 5: val_accuracy improved from 0.35714 to 0.39286, saving model to model_cnn.h5
    21/21 [==============================] - 14s 684ms/step - loss: 1.2583 - accuracy: 0.5580 - val_loss: 1.4605 - val_accuracy: 0.3929
    Epoch 6/40
    21/21 [==============================] - ETA: 0s - loss: 1.0670 - accuracy: 0.6324
```

The provided code snippet trains a convolutional neural network (CNN) model using TensorFlow's Keras API. It utilizes data generators `train_generator` and `valid_generator` to feed augmented image data batches into the model for training and validation, respectively. The model is trained over 10 epochs, with each epoch iterating through the entire training dataset and adjusting the model's weights to minimize categorical cross-entropy loss. The validation data is used to evaluate the model's performance on unseen data after each epoch. Training progress and metrics are displayed on the console (`verbose=1`). The

`history` object returned by the `fit()` method stores information about training and validation loss and accuracy for further analysis. Overall, this process facilitates the training of the CNN model, allowing it to learn from the training data and improve its performance over successive epochs.

ResNet50 Model Initialisation

ResNet-50, short for Residual Network with 50 layers, is a deep convolutional neural network architecture that has significantly impacted the field of computer vision. Introduced by Microsoft Research in 2015, ResNet-50 is renowned for its remarkable depth while mitigating the vanishing gradient problem commonly encountered in deep networks. The key innovation of ResNet-50 lies in the use of residual connections, or skip connections, which allow the network to learn residual mappings instead of directly fitting the desired underlying mapping. This enables the training of very deep networks, up to 50 layers or more, by mitigating

the degradation problem caused by the increased depth. ResNet-50 has demonstrated superior performance on various visual recognition tasks, including image classification, object detection,

and semantic segmentation, and it serves as a foundational architecture in modern deep learning frameworks and applications.

```
[ ] base_model=tf.keras.applications.ResNet50(include_top=False, weights="imagenet",input_shape=(255,255,3))
    print('Created ResNet50  model')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_
94765736/94765736 [==============================] - 0s 0us/step
Created ResNet50  model
```

Adding Dense Layers and Compiling the Model

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.

```
for layer in base_model.layers:
    layer.trainable = False
```

+ Code + Text

```
[ ] for layer in base_model.layers[173:]:
        layer.trainable = True
```

```
[ ] x1 = base_model.output

    #Global average pool to reduce number of features and Flatten the output
    x2 = tf.keras.layers.GlobalAveragePooling2D()(x1)
```

```
[ ] # adding extra layers
    x3 = tf.keras.layers.Dense(1024,activation='relu',kernel_initializer= "he_uniform")(x2)
    x4 = tf.keras.layers.Dropout(0.4)(x3)
    x5=  tf.keras.layers.Dense(512,activation='relu',kernel_initializer= "he_uniform")(x4)
```

In the provided code snippet, a pre-trained base model is used, and its layers are frozen by setting

`trainable` to `False` for all layers. Then, starting from the 173rd layer, the layers are unfrozen to allow fine-tuning of these layers during training.

The output of the base model (`base_model.output`) is passed through a global average pooling layer (`GlobalAveragePooling2D`) to reduce the number of features and flatten the output. Subsequently, two dense layers with 1024 and 512 units, respectively, are added, each followed by ReLU activation and dropout regularization with a dropout rate of 0.4. These dense layers serve to further process the features extracted by the base model before the final classification layer. The `kernel_initializer` parameter is set to "he_uniform" for both dense layers, which initializes the layer weights using a He uniform

```
[ ] #Add output layer
    prediction = tf.keras.layers.Dense(7,activation='softmax')(x5)
```

```
final_model = tf.keras.models.Model(inputs=base_model.input, #Pre-trained model input as input layer
                                    outputs=prediction)
```

```
[ ] final_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[ ] model_checkpoint_callback_rs = tf.keras.callbacks.ModelCheckpoint(
        filepath="model_50.h5",
        monitor='val_accuracy',
        mode='max',
        save_best_only=True,
        verbose=1)
```

In this code snippet, a final prediction layer is added to the model (`final_model`) after the last dense layer (`x5`). This prediction layer consists of a dense layer with 9 units (assuming it's a classification task with 9 classes) and softmax activation function, which is suitable for multi-class classification problems.

The `final_model` is then defined using the `tf.keras.models.Model` class, specifying the inputs as the input layer of the pre-trained model (`base_model.input`) and the outputs as the prediction layer (`prediction`).

Finally, the model is compiled using the Adam optimizer, categorical crossentropy as the loss function (since it's a multi-class classification problem), and accuracy as the evaluation metric

Create callbacks

Callbacks in convolutional neural networks (CNNs) are objects that can perform actions at various stages during the training process. They are commonly used in TensorFlow and Keras to monitor the training progress, adjust learning rates dynamically, save model checkpoints, and more.

```
[ ]  model.compile(loss='categorical_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

The provided code snippet compiles a Keras model using the Adam optimizer and categorical cross-entropy loss function, which is commonly used for multi-class classification tasks. Additionally, the accuracy metric is specified to monitor the performance of the model during training and evaluation. This configuration prepares the model for the training process, where it learns to minimize the specified loss function using the optimizer while maximizing the chosen metric to improve its classification accuracy.

Train the model

Now, let us train our model with our image dataset. The model is trained for 10 epochs. We can see that the training loss decreases in almost every epoch till 10 epochs and probably there is further scope to improve the model.

fit functions used to train a deep-learning neural network

Arguments:

epochs: an integer and number of epochs we want to train our model for. validation_data can be either:

- an inputs and targets list
- a generator
- an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.

validation_steps: only if the validation_data is a generator then only this argument

can be used. It specifies the total number of steps taken from the generator before it is

stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the

validation batch size.

```
history_resnet = final_model.fit(train_gen,
                    epochs = 20,
                    validation_data = valid_gen,callbacks = [model_checkpoint_callback_rs])
```

```
Epoch 1/20
21/21 [==============================] - ETA: 0s - loss: 2.2749 - accuracy: 0.2515
Epoch 1: val_accuracy improved from -inf to 0.29762, saving model to model_50.h5
21/21 [==============================] - 23s 813ms/step - loss: 2.2749 - accuracy: 0.2515 - val_loss: 1.6784 - val_accuracy: 0.2976
Epoch 2/20
21/21 [==============================] - ETA: 0s - loss: 1.5786 - accuracy: 0.3750
Epoch 2: val_accuracy improved from 0.29762 to 0.47619, saving model to model_50.h5
21/21 [==============================] - 14s 657ms/step - loss: 1.5786 - accuracy: 0.3750 - val_loss: 1.4757 - val_accuracy: 0.4762
Epoch 3/20
21/21 [==============================] - ETA: 0s - loss: 1.3548 - accuracy: 0.4985
Epoch 3: val_accuracy improved from 0.47619 to 0.48810, saving model to model_50.h5
21/21 [==============================] - 14s 626ms/step - loss: 1.3548 - accuracy: 0.4985 - val_loss: 1.3685 - val_accuracy: 0.4881
Epoch 4/20
21/21 [==============================] - ETA: 0s - loss: 1.2655 - accuracy: 0.5283
Epoch 4: val_accuracy did not improve from 0.48810
21/21 [==============================] - 14s 645ms/step - loss: 1.2655 - accuracy: 0.5283 - val_loss: 1.3939 - val_accuracy: 0.4881
Epoch 5/20
21/21 [==============================] - ETA: 0s - loss: 1.2197 - accuracy: 0.5491
Epoch 5: val_accuracy improved from 0.48810 to 0.57143, saving model to model_50.h5
21/21 [==============================] - 15s 704ms/step - loss: 1.2197 - accuracy: 0.5491 - val_loss: 1.2603 - val_accuracy: 0.5714
Epoch 6/20
21/21 [==============================] - ETA: 0s - loss: 1.1768 - accuracy: 0.5744
Epoch 6: val_accuracy did not improve from 0.57143
21/21 [==============================] - 14s 644ms/step - loss: 1.1768 - accuracy: 0.5744 - val_loss: 1.2398 - val_accuracy: 0.5357
Epoch 7/20
21/21 [==============================] - ETA: 0s - loss: 1.1414 - accuracy: 0.5685
Epoch 7: val_accuracy improved from 0.57143 to 0.58333, saving model to model_50.h5
21/21 [==============================] - 14s 693ms/step - loss: 1.1414 - accuracy: 0.5685 - val_loss: 1.1773 - val_accuracy: 0.5833
```

Visualising Performance

We will plot the accuracy and loss of both training and accuracy for all 40 epochs for the CNN model and 20 epochs of the transfer learning model.

```
[ ] import matplotlib.pyplot as plt
```

```
[ ] acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(5)

    plt.figure(figsize=(15, 15))
    plt.subplot(2, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(2, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()
```

```
import os
import numpy as np
import tensorflow as tf
from PIL import Image
from flask import Flask, render_template, request
from keras.preprocessing.image import load_img, img_to_array

# Initialize Flask app
app = Flask(_name_)

# Load the trained model
model = tf.keras.models.load_model("model.h5")

# Home page route
@app.route('/')
def index():
    return render_template("index.html")

# Prediction route
@app.route('/predict', methods=['GET', 'POST'])
def output():
    if request.method == 'POST':
        f = request.files['pc_image']

        # Save the uploaded image to static/uploads/
        upload_folder = "static/uploads/"
        if not os.path.exists(upload_folder):
            os.makedirs(upload_folder)

        img_path = os.path.join(upload_folder, f.filename)
```

```
        f.save(img_path)

        # Load and preprocess the image
        img = load_img(img_path, target_size=(224, 224))
        image_array = img_to_array(img) / 255.0  # Normalize
        image_array = np.expand_dims(image_array, axis=0)

        # Predict
        pred = model.predict(image_array)
        pred_class = np.argmax(pred, axis=1)[0]

        # Map prediction index to labels
        classes = ['Coccidiosis', 'Healthy', 'New Castle Disease', 'Salmonella']
        prediction = classes[pred_class]

        return render_template('contact.html', predict=prediction)

    return render_template('contact.html', predict="No file uploaded.")

# Run the app
if _name_ == '_main_':
    app.run(debug=True)
```

Testing Model & Data Prediction

```
train_gen.class_indices.keys()
```

```
dict_keys(['Coccidiosis', 'Healthy', 'New Castle Disease', 'Salmonella'])
```

```
labels = ['Coccidiosis', 'Healthy', 'New Castle Disease', 'Salmonella']
```

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
def get_model_prediction(image_path):
    img = load_img(image_path, target_size=(224, 224))
    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)
    predictions = model.predict(x, verbose=0)
    return labels[predictions.argmax()]
```

The code defines a function named build_model that creates a CNN model using Keras.

It then uses this function along with Hyperparameter, a hyperparameter tuning technique, to find the optimal configuration (e.g., number of layers, learning rate) for the model that achieves the best validation accuracy.

```
get_model_prediction('/content/data/data/test/Coccidiosis/cocci.0.jpg_aug33.JPG')
```

```
'Coccidiosis'
```

```
get_model_prediction('/content/data/data/test/Healthy/healthy.1003.jpg_aug47.JPG')
```

```
'Healthy'
```

```
get_model_prediction('/content/data/data/test/New Castle Disease/ncd.1.jpg_aug197.JPG')
```

```
'New Castle Disease'
```

```
get_model_prediction('/content/data/data/test/Salmonella/pcrsalmo.111.jpg_aug29.JPG')
```

```
'Salmonella'
```

```
get_model_prediction('/content/data/data/test/Salmonella/pcrsalmo.115.jpg_aug28.JPG')
```

```
'Salmonella'
```

Manual Testing

```
pred = []
for file in test_df['path'].values:
    pred.append(get_model_prediction(file))
```

```
[ ] fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(15, 10))
    random_index = np.random.randint(0, len(test_gen), 16)

    for i, ax in enumerate(axes.ravel()):
        img_path = test_df['path'].iloc[random_index[i]]

        ax.imshow( load_img(img_path))
        ax.axis('off')

        if test_df['label'].iloc[random_index[i]] == pred[random_index[i]]:
            color = "green"
        else:
            color = "red"

        ax.set_title(f"True: {test_df['label'].iloc[random_index[i]]}\nPredicted: {pred[random_index[i]]}", color=color)

    plt.tight_layout()
    plt.show()
```

```
[ ] fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(15, 10))
    random_index = np.random.randint(0, len(test_gen), 16)

    for i, ax in enumerate(axes.ravel()):
        img_path = test_df['path'].iloc[random_index[i]]

        ax.imshow( load_img(img_path))
        ax.axis('off')

        if test_df['label'].iloc[random_index[i]] == pred[random_index[i]]:
            color = "green"
        else:
            color = "red"

        ax.set_title(f"True: {test_df['label'].iloc[random_index[i]]}\nPredicted: {pred[random_index[i]]}", color=color)

    plt.tight_layout()
    plt.show()
```

The provided code segment generates predictions for a subset of images from the test dataset using the

`get_model_prediction` function. It then visualizes the predicted results alongside the true labels for comparison.

A subplot grid with 4 rows and 4 columns is created using `plt.subplots()`, with a figure size of 15x10 inches. A random index is generated to select 16 images from the test dataset.

For each subplot, an image is loaded from the test dataset using the image path stored in `test_df['path']`. The image is displayed using `imshow()`, and the axis is turned off using `ax.axis('off')`.

The true label and predicted label for each image are displayed in the subplot title. If the true label matches the predicted label, the title text is displayed in green; otherwise, it is displayed in red.

Finally, `plt.tight_layout()` is called to adjust subplot parameters for better layout, and `plt.show()` displays the plot.

Overall, this code segment provides a visual representation of the model's predictions on a subset of test images, allowing for easy interpretation and assessment of the model's performance.

Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

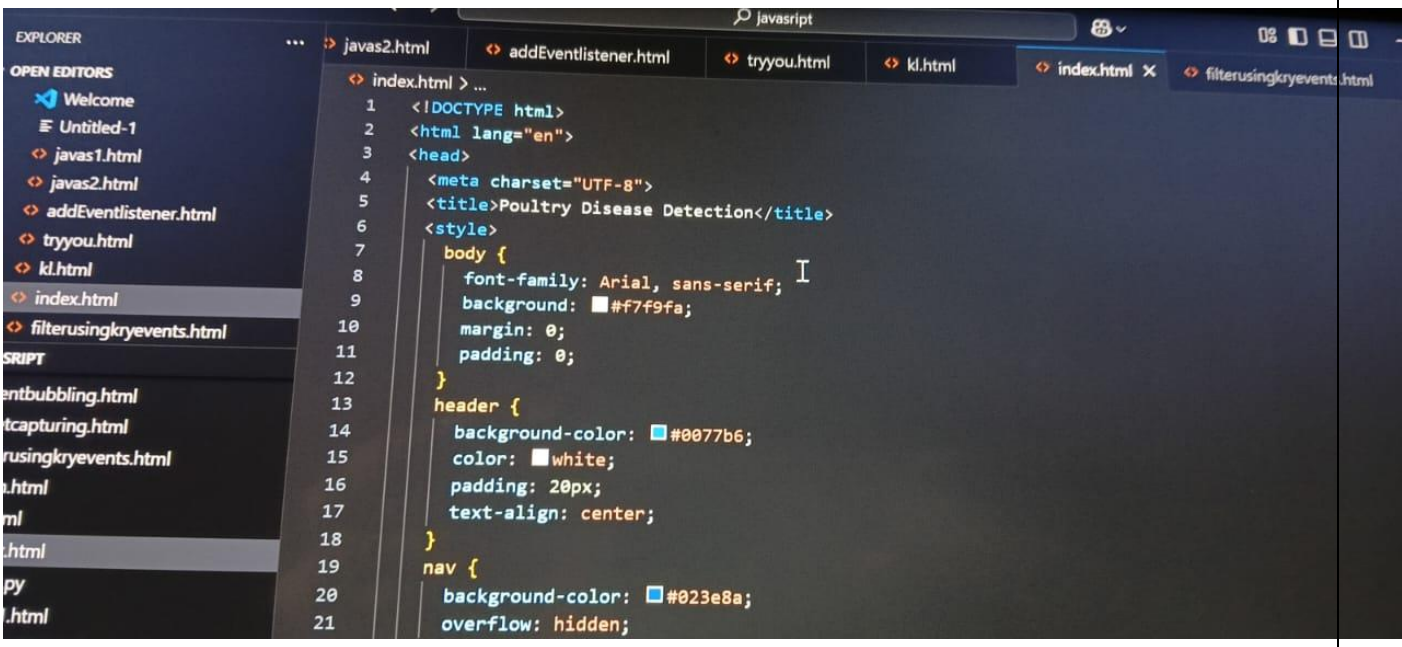
- Building HTML Pages
- Building server-side script

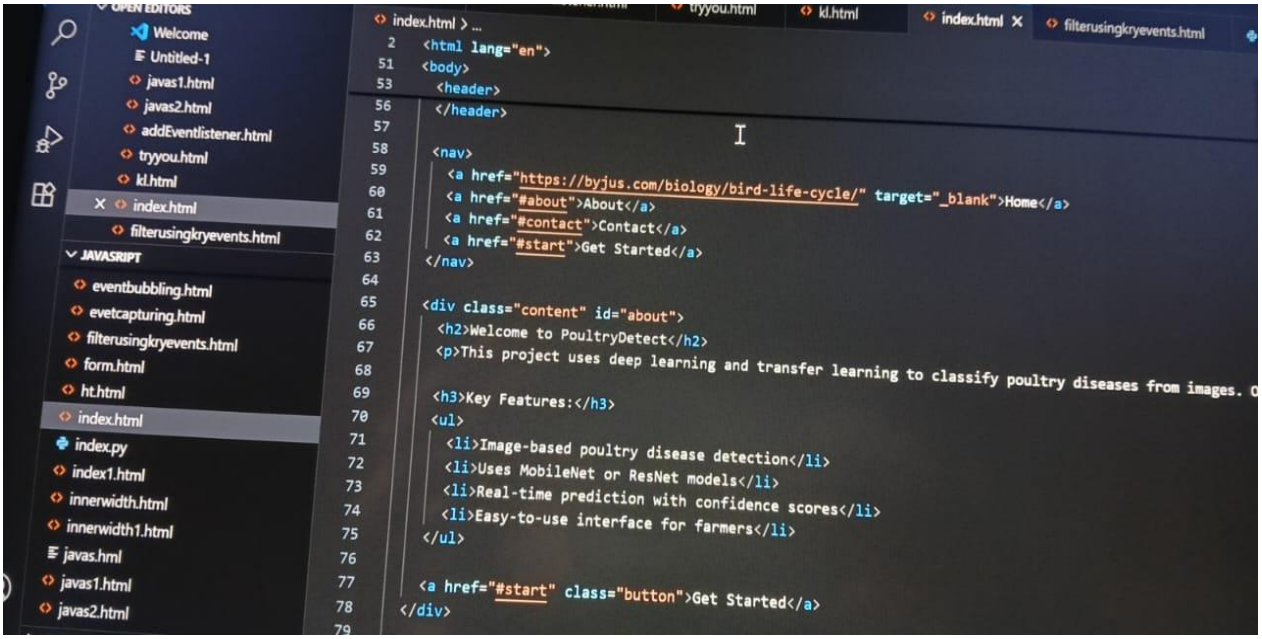
- **Building Html Page**

For this project create one HTML file namely

- home.html

INPUT CODE BE LIKE:

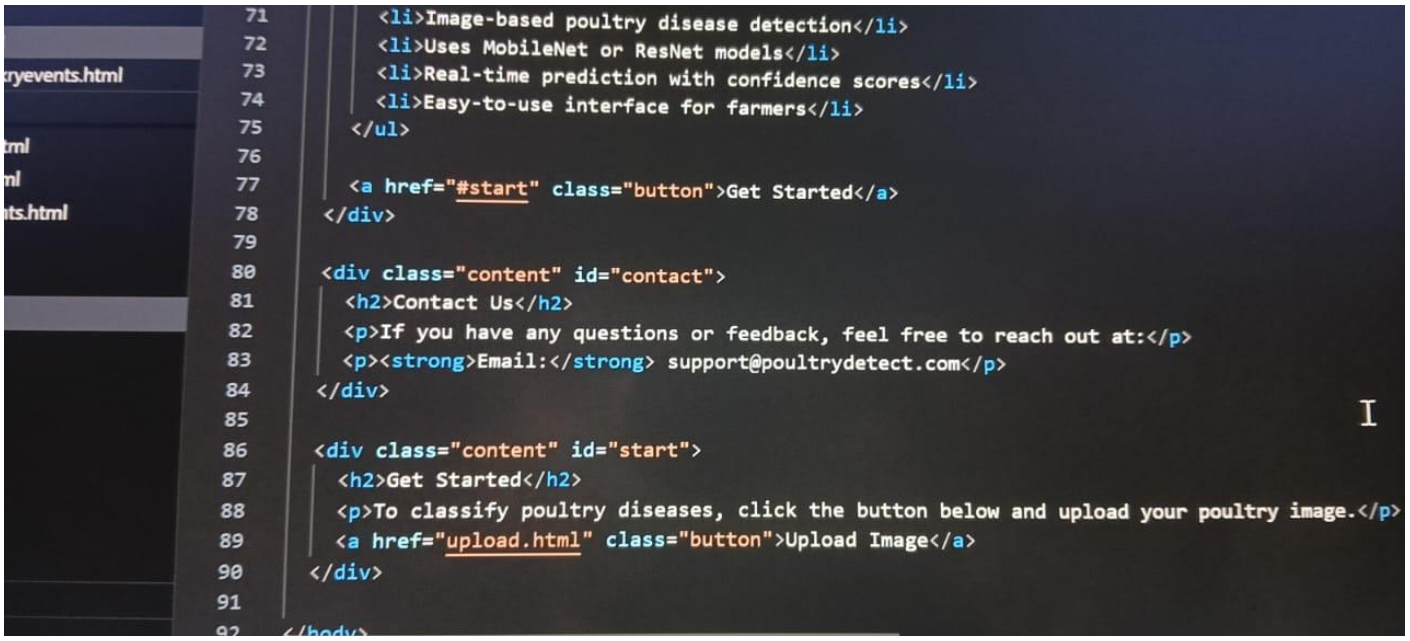



OPEN EDITORS
Welcome
Untitled-1
javas1.html
javas2.html
addEventlistener.html
tryyou.html
kl.html
index.html
filterusingkryevents.html
JAVASRIPT
eventbubbling.html
evetcapturing.html
filterusingkryevents.html
form.html
ht.html
index.html
index.py
index1.html
innerwidth.html
innerwidth1.html
javas.html
javas1.html
javas2.html
index.html > ...
<html lang="en">
<body>
<header>
</header>
<nav>
Home
About
Contact
Get Started
</nav>

<h2>Welcome to PoultryDetect</h2>
This project uses deep learning and transfer learning to classify poultry diseases from images.
<h3>Key Features:</h3>

Image-based poultry disease detection
Uses MobileNet or ResNet models
Real-time prediction with confidence scores
Easy-to-use interface for farmers

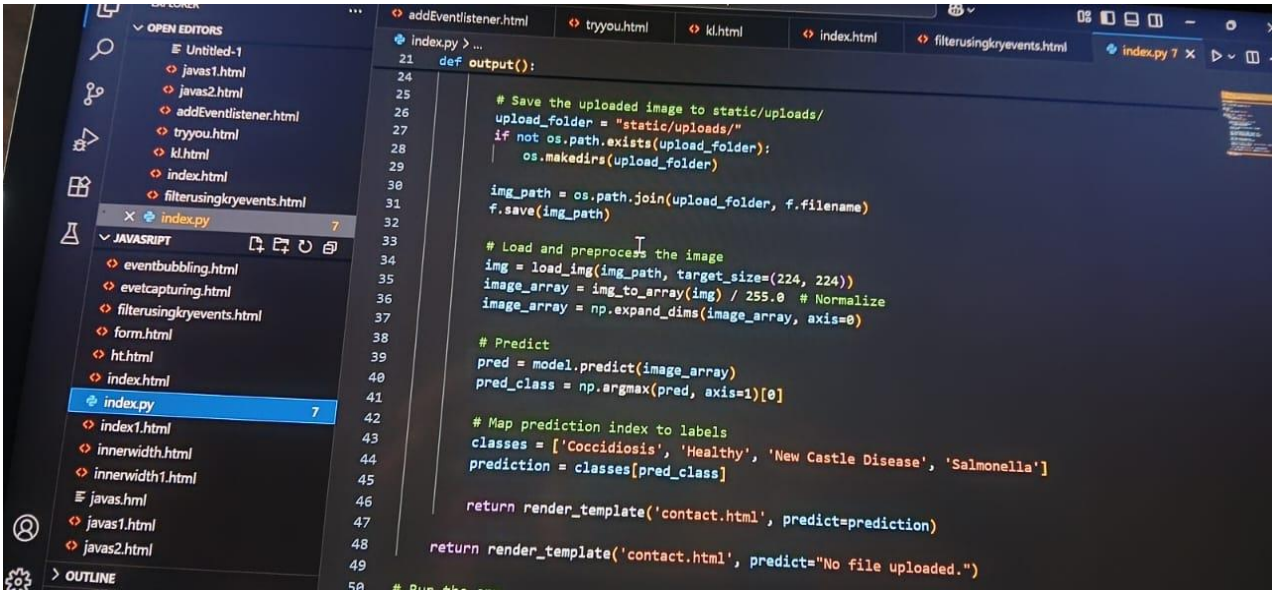
Get Started

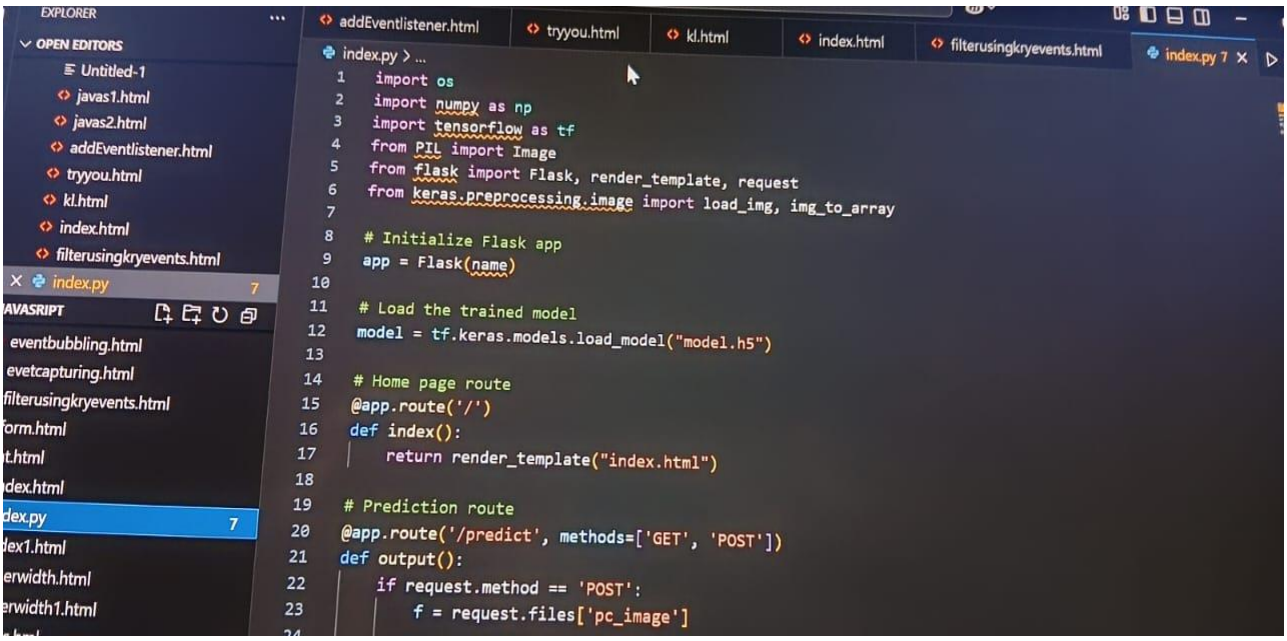

71 Image-based poultry disease detection
72 Uses MobileNet or ResNet models
73 Real-time prediction with confidence scores
74 Easy-to-use interface for farmers
75
76
77 Get Started
78
79
80
81 <h2>Contact Us</h2>
82 If you have any questions or feedback, feel free to reach out at:
83 Email: support@poultrydetect.com
84
85
86
87 <h2>Get Started</h2>
88 To classify poultry diseases, click the button below and upload your poultry image.
89 Upload Image
90
91

```
        <li>Image-based poultry disease detection</li>
        <li>Uses MobileNet or ResNet models</li>
        <li>Real-time prediction with confidence scores</li>
        <li>Easy-to-use interface for farmers</li>
      </ul>

      <a href="#start" class="button">Get Started</a>
    </div>

    <div class="content" id="contact">
      <h2>Contact Us</h2>
      <p>If you have any questions or feedback, feel free to reach out at:</p>
      <p><strong>Email:</strong> support@poultrydetect.com</p>
    </div>

    <div class="content" id="start">
      <h2>Get Started</h2>
      <p>To classify poultry diseases, click the button below and upload your poultry image.</p>
      <a href="upload.html" class="button">Upload Image</a>
    </div>
```

Build Python code:

- Import the libraries
- Loading the saved model and initializing the Flask app
- Render HTML pages:
- Once we upload the file into the app, then verifying the file uploaded properly or not. Here we will be using the declared constructor to route to the HTML page that we have created earlier.



- 

- In the above example, '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered. Whenever you enter Here we are routing our app to the prediction function. This function retrieves all the values from the HTML page using a Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. This prediction value will be rendered to the text that we have mentioned in the index.html page earlier.

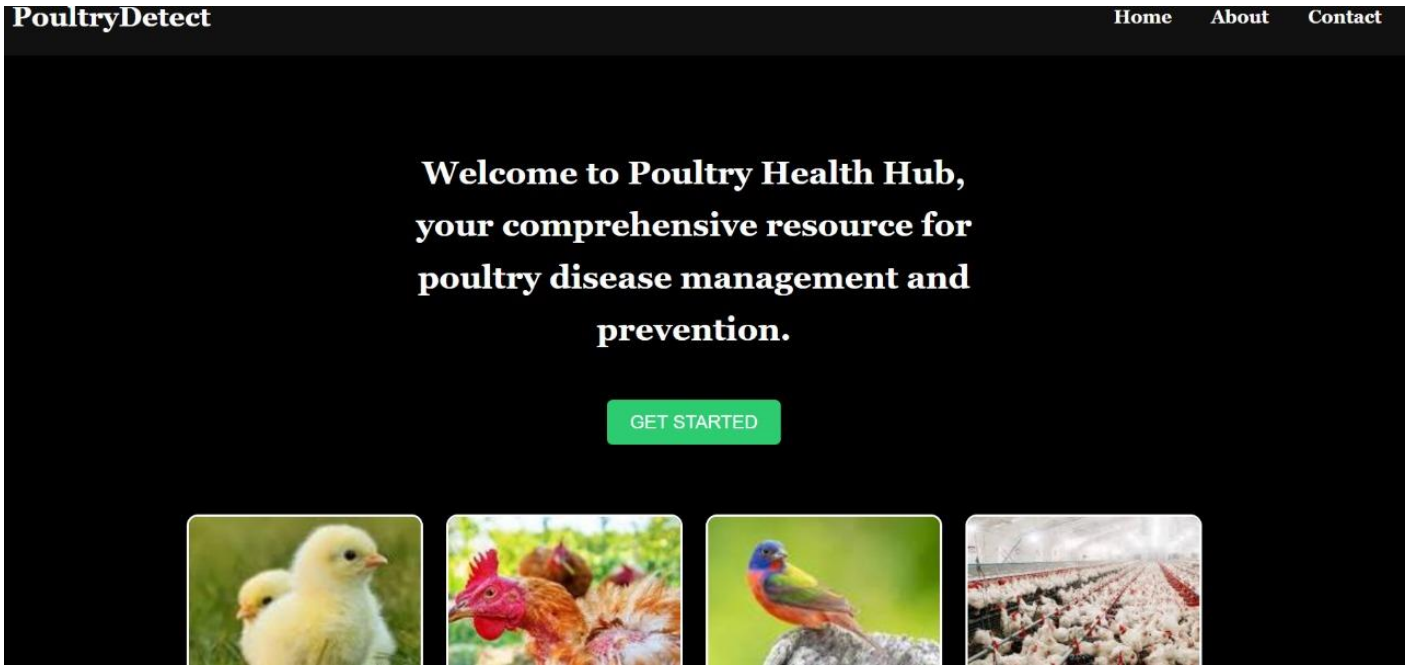
Run the application:

- Open the Anaconda prompt from the start menu.
- Navigate to the folder where your Python script is.
- Now type the "python my_app.py" command.
- Navigate to the localhost where you can view your web page.

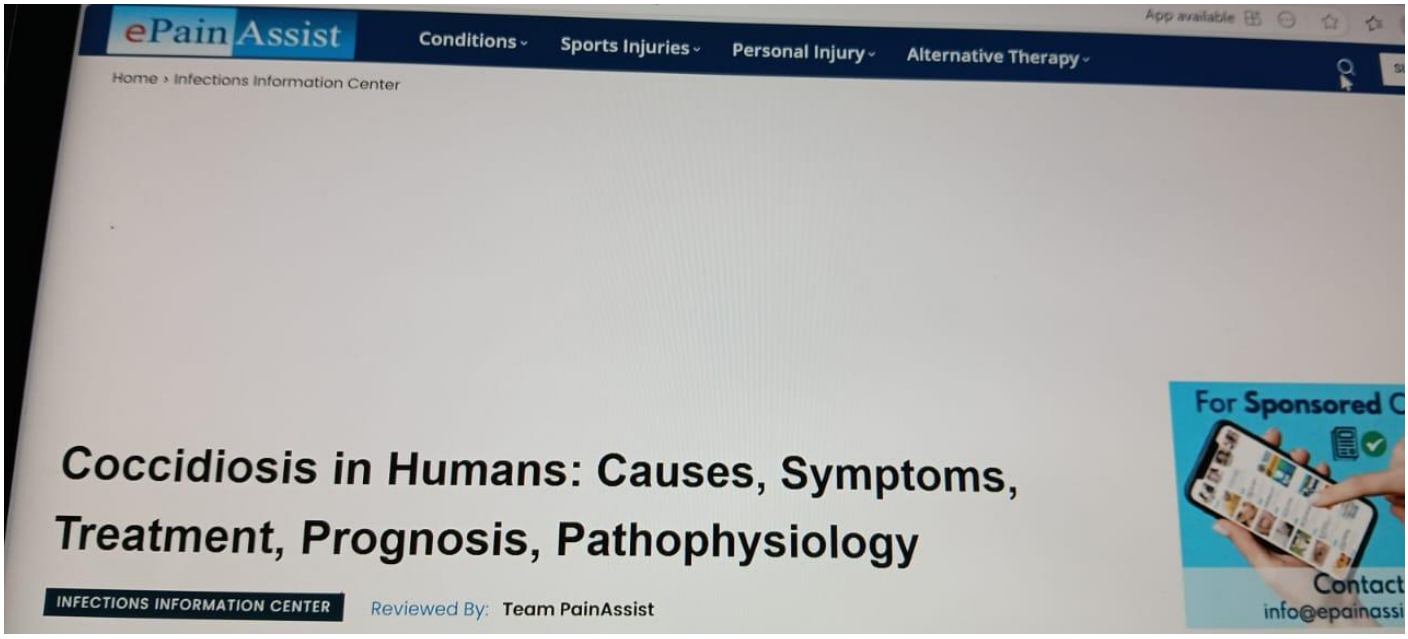
- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate
compiler flags.
 * Debugger is active!
 * Debugger PIN: 946-817-010
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Click on Get Started at the top right corner to go to the inner page as below:



Output2:


ePain Assist
Conditions
Sports Injuries
Personal Injury
Alternative Therapy
App available
Home › Infections Information Center
Coccidiosis in Humans: Causes, Symptoms, Treatment, Prognosis, Pathophysiology
INFECTIONS INFORMATION CENTER
Reviewed By: Team PainAssist
For Sponsored
Contact
info@epainassi

FUTURE ENHANCEMENTS:

- ✓ We can provide our basic requirements how our design should be, it will directly provide the best top most designated structure for the customer satisfaction.

- ✓ We can add the AI chatbot to this basic classification fabrics using deep learning

Project which benefits the easy interface for all the new users and the bugs can be easily Solved in time.

- ✓ We can add the all the textile companies collab with one-platform to interact with Each other and increase their Business strategies.

- ✓ By providing the all the textile related information in one-platform all the small textile Producers can be benefitted more.

- ✓ Can be add tie-up with foreign traders for the export and import the textiles which

Is not only benefit to the textile industries but also increase our Indian economy growth.

THE END

vv

- ✓