Final Project Individual Report

Mahikshit Kurapati

Project link: https://github.com/Archonz-crazy/DL_final_project

Dataset 1 link: https://www.kaggle.com/datasets/anamibnjafar0/flamevision/code

Dataset 2 link: https://www.kaggle.com/datasets/elmadafri/the-wildfire-dataset/data

Introduction

The title of our project is “**Deep Learning for Wildfire Detection and Prediction**”. So we had a dataset from Kaggle which contained 3000 hi-res images of forests on fire and not on fire. I took the job of creating an algorithm which can predict accurately whether the image uploaded has fire in it or not, and thereby, the forest is on fire or not. This application is developed as a first response confirmation.

- We had divided our project into 3 parts: EDA, Algorithm, Streamlit.
- I had the Algorithm part where I started working on the images using CNN.
- This was a very basic model and was not performing well on the images.
- Then, I slowly developed the model using VGG16 pretrained weights and a Sequential Dense model.
- While I could see that the model was performing well, there were very less images in the dataset and I thought this was making my model a bit biased towards images with fire in it.
- So, we searched for some other datasets and found our second dataset.
- We merged the two datasets and made a single dataset with over 13000 images.
- Then I trained my model on this dataset, which balanced the data and gave me appropriate metrics.

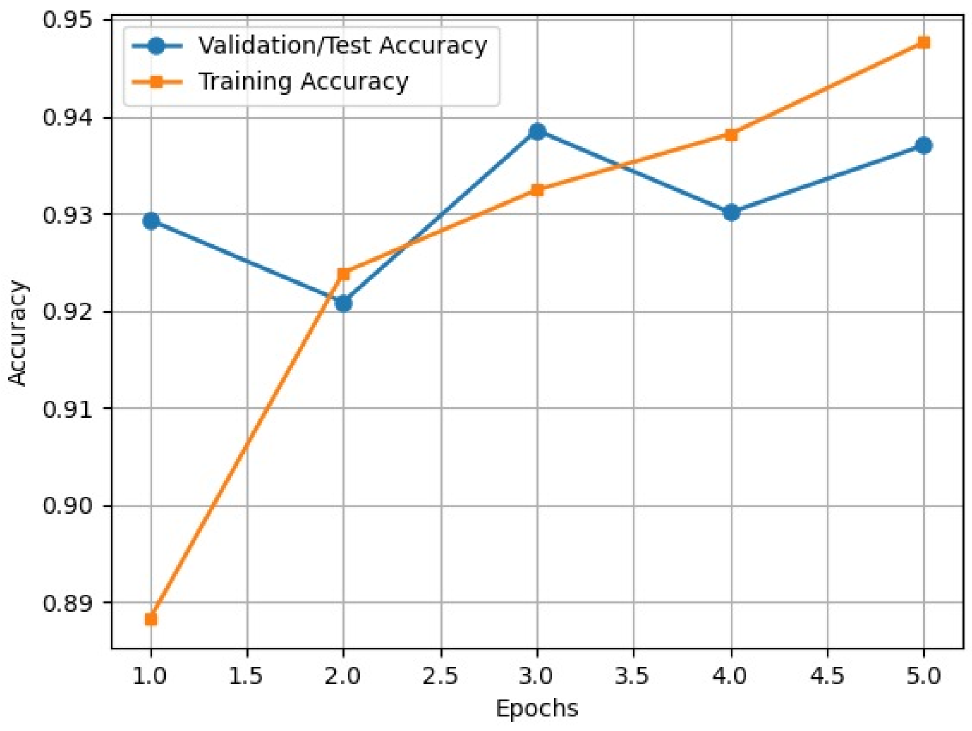
Description

- This is the model I wrote to train on the images.

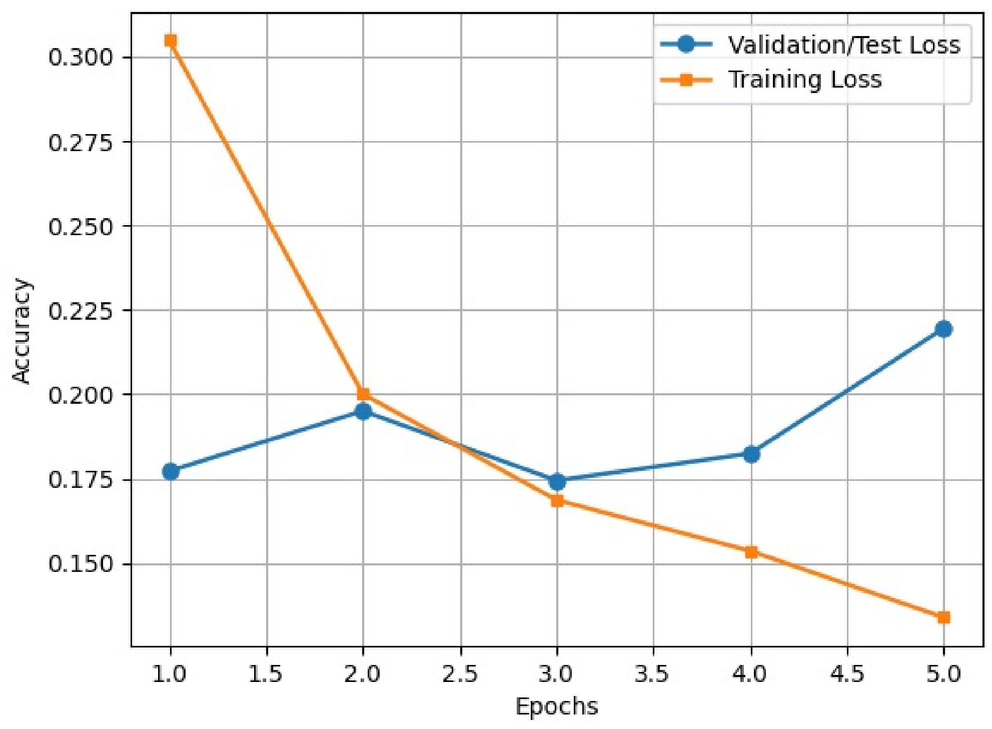
```
if os.path.exists(os.path.join(code_path, 'model.h5')):
    VGG16_model = VGG16(
        include_top = False,
        weights="imagenet",
        input_shape = input_shape
    )
    for layer in VGG16_model.layers :
        layer.trainable = False

    from keras import activations
    model = Sequential()
    model.add(VGG16_model)
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2,2)))
    model.add(Flatten())
    model.add(Dense(512,activation = 'relu'))
    model.add(Dropout(0.3))
    model.add(Dense(256,activation = 'relu'))
    model.add(Dense(128,activation = 'relu'))
    model.add(Dense(num_classes,activation = 'softmax'))
    model.summary()
```

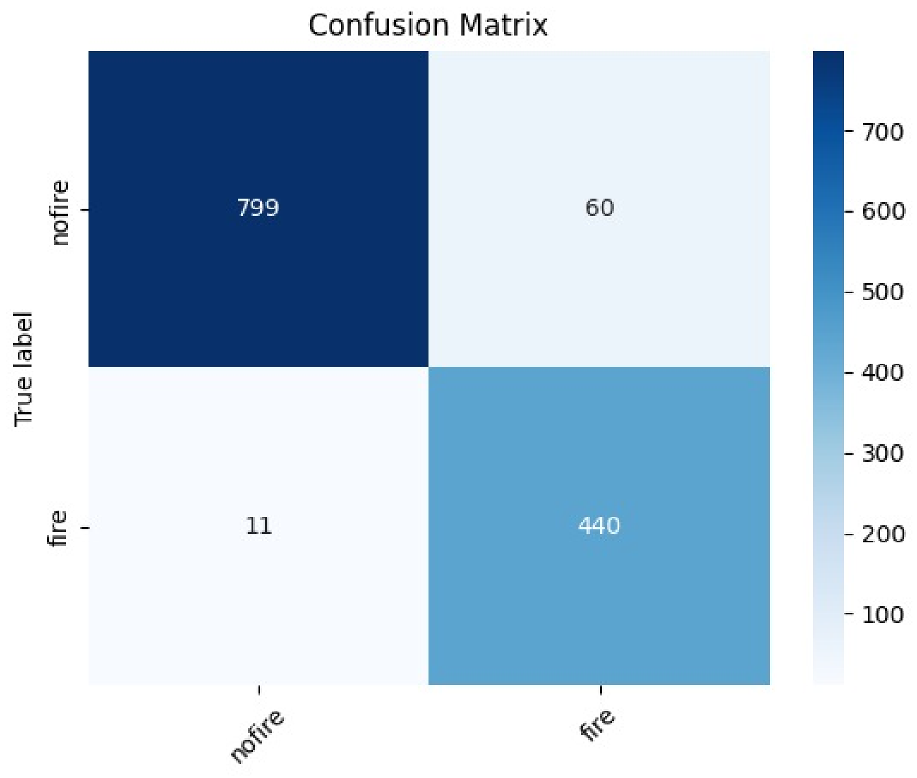
- The accuracy and loss on this model for our dataset was pretty good and it was even able to distinguish between different types of fires.



- This shows the Accuracy of the model on train and test images.



- This shows the loss of the model on the train and test images.
- I took this model and it was predicting pretty well for the images in the dataset.
- Next, I checked the confusion matrix for this model.



- This seemed a good enough model for me.

Streamlit

- Next, I shifted my focus to developing a working Streamlit application now that I have my model.
- My teammate already created a very good application where I had to integrate my model and create an endpoint.
- So, I started working on Streamlit and used the following code to create our website:
- I preprocessed the uploaded image in the same way I did it in the training file:

```
def preprocess_image(img):
        img = img.resize((224, 224))
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img_array = img_to_array(img)
        img_array = preprocess_input(img_array)
        img_array = np.expand_dims(img_array, axis=0)
        return img_array
```

- Then I wrote the below streamlit application:

```
st.title("Wildfire Prediction")

# Add a slider for the confidence threshold
confidence_threshold = st.sidebar.slider("Set the confidence threshold (%)", 0, 100, 50)  # Start at 50%

# File uploader allows user to add their own image
uploaded_file = st.file_uploader("Upload an image", type=["jpg", "png", "jpeg"])

if uploaded_file is not None:
    # Display the uploaded image
    image = Image.open(uploaded_file)
    st.image(image, caption='Uploaded image', use_column_width=True)
    st.write("")
    st.write("Classifying...")

    # Preprocess the image and predict
    processed_image = preprocess_image(image)
    prediction = model.predict(processed_image)

    class_names = ['fire', 'no fire']
    class_index = np.argmax(prediction)
    confidence = np.max(prediction) * 100  # Convert to percentage

    # Check if the confidence is above the threshold
    if confidence >= confidence_threshold:
        predicted_class = class_names[class_index]
        st.write(f"Prediction: {predicted_class} (Class {class_index})")
        st.write(f"Confidence Score: {confidence:.2f}%")
    else:
        st.write("Confidence score too high to make a prediction.")
```

- I added a slider for to adjust the confidence interval to predict whether an image has fire or not with a particular level of confidence.

Future Enhancement

- When I was presenting our project, I saw that our model was not able to distinguish between certain types of fire or entirely different images like cats etc. I would like to add to my code so that the algorithm is able to distinguish between real fire images and random images.
- We can also make this model take in video files and predict live whether there is a chance for a wildfire or not. We can add it factors like how many days it will take the fires to stop, and how much acres of forest will be lost etc.

Percentage of code

- As the dataset I took from Kaggle don't have any dedicated notebooks, every code I've written is written by me or referenced from GPT.
- When I did the calculation, I got 24% of my code is from GPT.

References

- https://keras.io/api/applications/vgg/

- https://www.kaggle.com/datasets/anamibnjafar0/flamevision/code
- https://www.kaggle.com/code/gauravduttakiit/flamevision-resnet50-efficientdet