DEEP LEARNING

May 2024

TP4 Report:

Transfer Learning

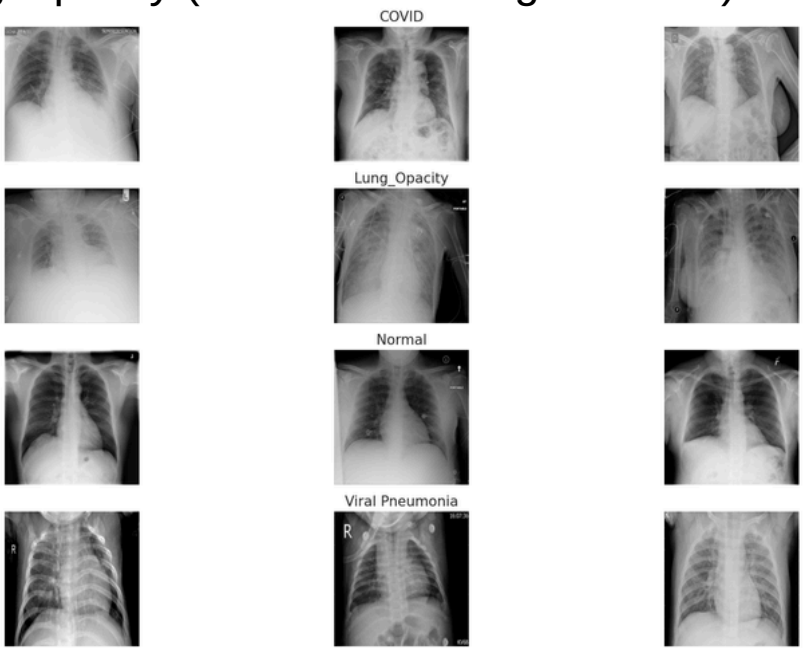
- HACHOUD Mohammed
- SEHILI Chaima

1. Load the **Chest X-rays** dataset and perform pre-processing: data normalization.

Chest X-rays dataset:

Database of CXR images included 3616 COVID-19 positive cases along with 10,192 Normal, 6012 Lung Opacity (Non-COVID lung infection) and 1345 Viral Pneumonia images
The classes encompass the following:

- COVID-19
- Normal
- Lung Opacity
- Viral Pneumonia



Data Preprocessing:

Data Normalization:

Resize the image into (224,244) then convert the pixel values data type to float32 type, and then normalizes them by dividing by the 255.

```
dataset['image'] = None

for i in range(0,len(dataset)):
    image_path = dataset['path'].iloc[i]
    image = cv2.imread(image_path)
    if image is not None:
        image=cv2.resize(image, (224, 224))
        image=np.asarray(image).astype('float32')
        normalized_image = image/255
        dataset.at[i, 'image'] = normalized_image
```

One hot encoding labels:

convert the class labels to one-hot vectors

```
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical

# Initialize the encoder
encoder = LabelEncoder()

# Fit the encoder and transform the 'label' column
dataset['label_encoded'] = encoder.fit_transform(dataset['label'])

# One-hot encode the 'label_encoded' column
one_hot_labels = to_categorical(dataset['label_encoded'], num_classes=4)
```

Data Split:

split the dataset into 70% for training ,15% for validation and 15% for Test .

```
from sklearn.model_selection import train_test_split
X = dataset['image'].values
y = one_hot_labels

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, stratify=y)

# Further split the validation set into validation and test sets
X_val, X_test, y_val, y_test = train_test_split(X_val, y_val, test_size=0.5, stratify=y_val)
```

2.**MobileNetV2** (x0.5) with **ImageNet** pretrained weights:

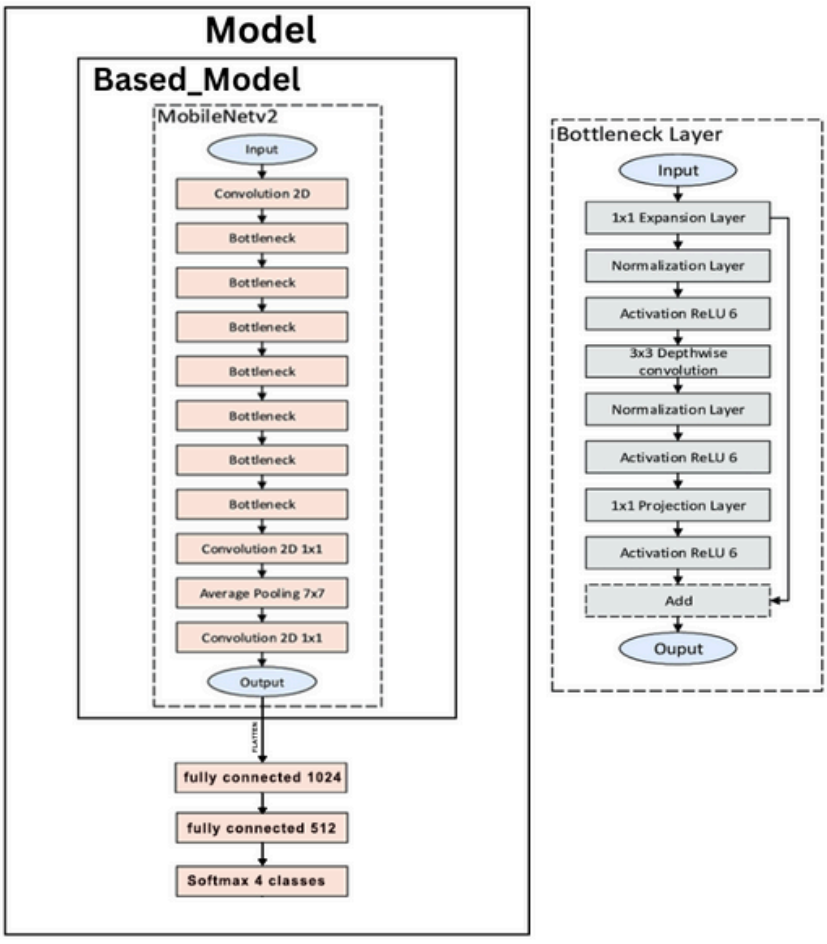
MobileNetV2 is a convolutional neural network architecture designed for mobile and embedded vision applications. It is an evolution of the original MobileNet architecture, aiming to improve efficiency and performance while maintaining low computational costs.

- Load

```
import tensorflow as tf
from keras.applications import MobileNetV2

based_model = MobileNetV2(input_shape=(224, 224, 3),alpha=0.5, weights='imagenet', include_top=False)
based_model.summary()
```

- Architecture

```
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_0.50_224 (Funct  (None, 7, 7, 1280)        706224
ional)

flatten_1 (Flatten)          (None, 62720)             0

dense_3 (Dense)              (None, 1024)              64226304

dense_4 (Dense)              (None, 512)               524800

dense_5 (Dense)              (None, 4)                 2052

=================================================================
Total params: 65,459,380
Trainable params: 65,037,636
Non-trainable params: 421,744
_________________________________________________________________
```

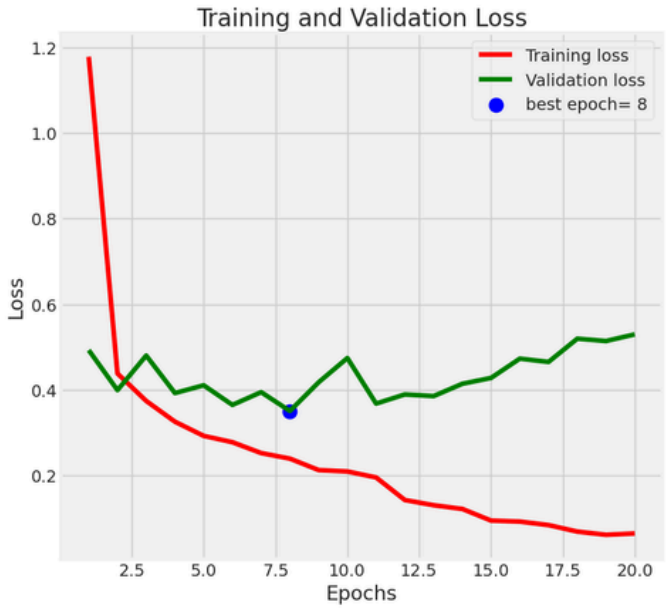


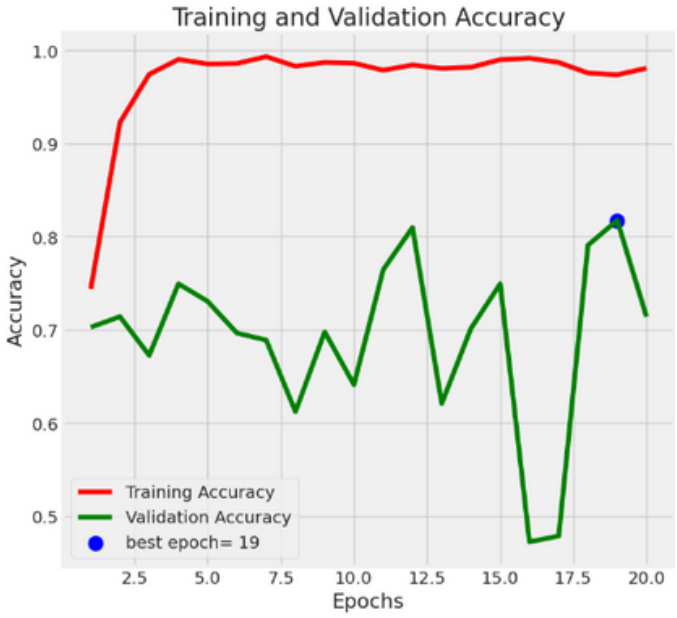
3.Training and Results

After adding the last three fully connected , our model got 65,459,380 trainable parameters. we trained our model for 20 epochs with a batch size of 64 , as an optimizer we used The ADAM optimizer with learning rate of 1e-3 .

A. Last 3 fully connected layers:

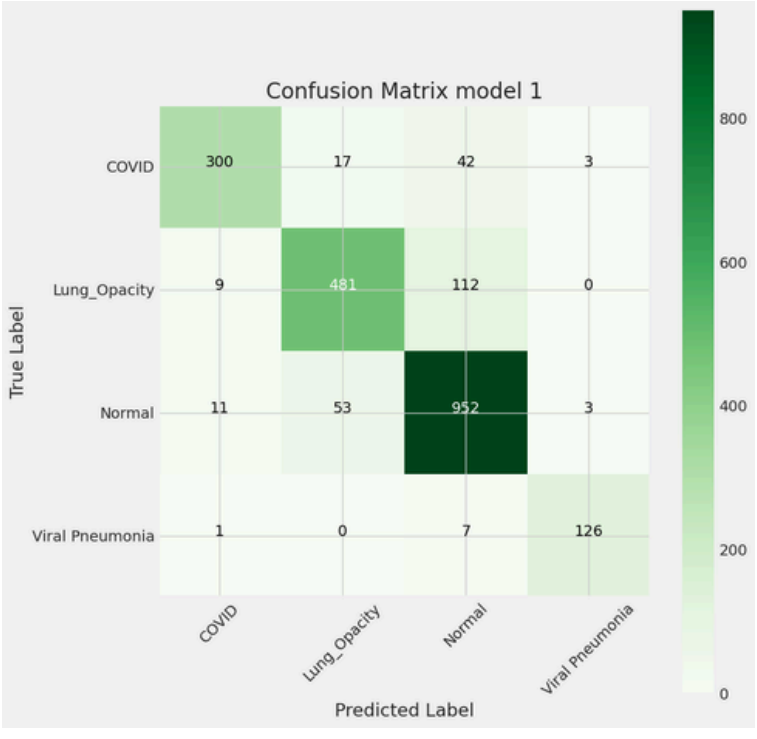
Model 1 Achieves decent training, validation, and test accuracies., it shows good balance between training and validation/test performance, indicating reasonable generalization.and relatively low risk of overfitting compared to other models.





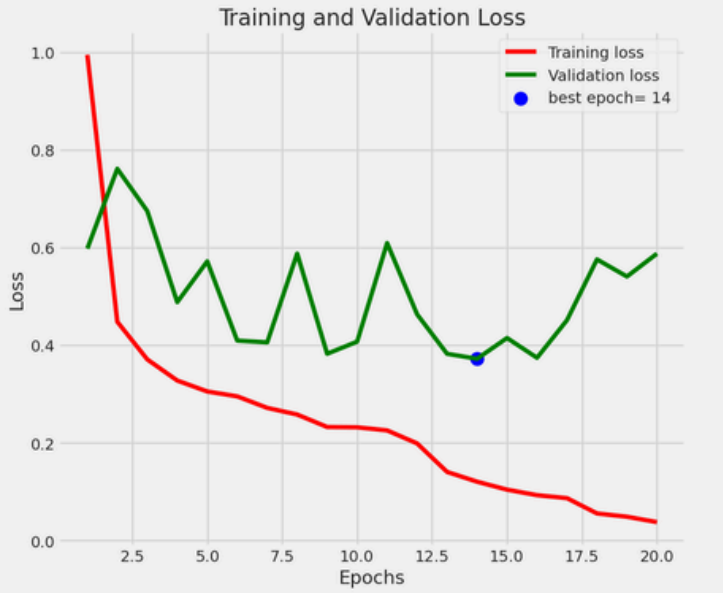
Confusion Matrix

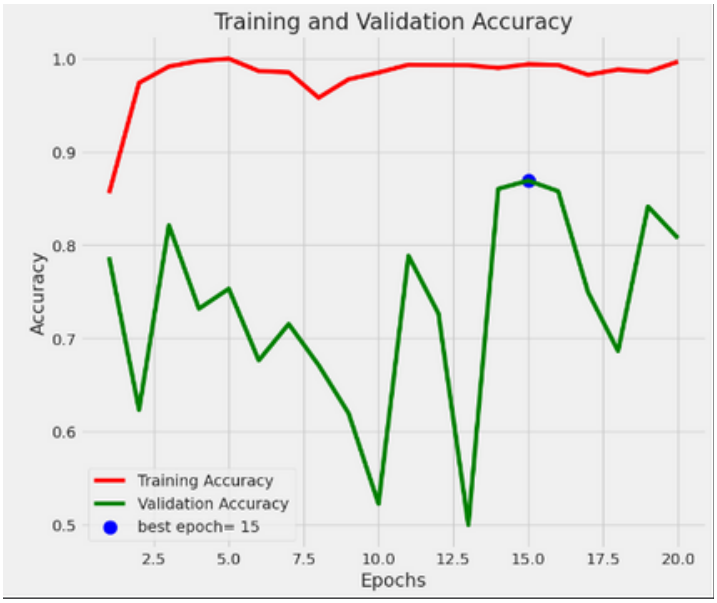
	precision	recall	f1-score	support
Class 1	0.53	0.87	0.66	135
Class 2	0.88	0.67	0.76	226
Class 3	0.78	0.80	0.79	383
Class 4	1.00	0.20	0.33	50
accuracy			0.74	794
macro avg	0.80	0.64	0.64	794
weighted avg	0.78	0.74	0.73	794



B. one convolutional layers + the last three fully connected:

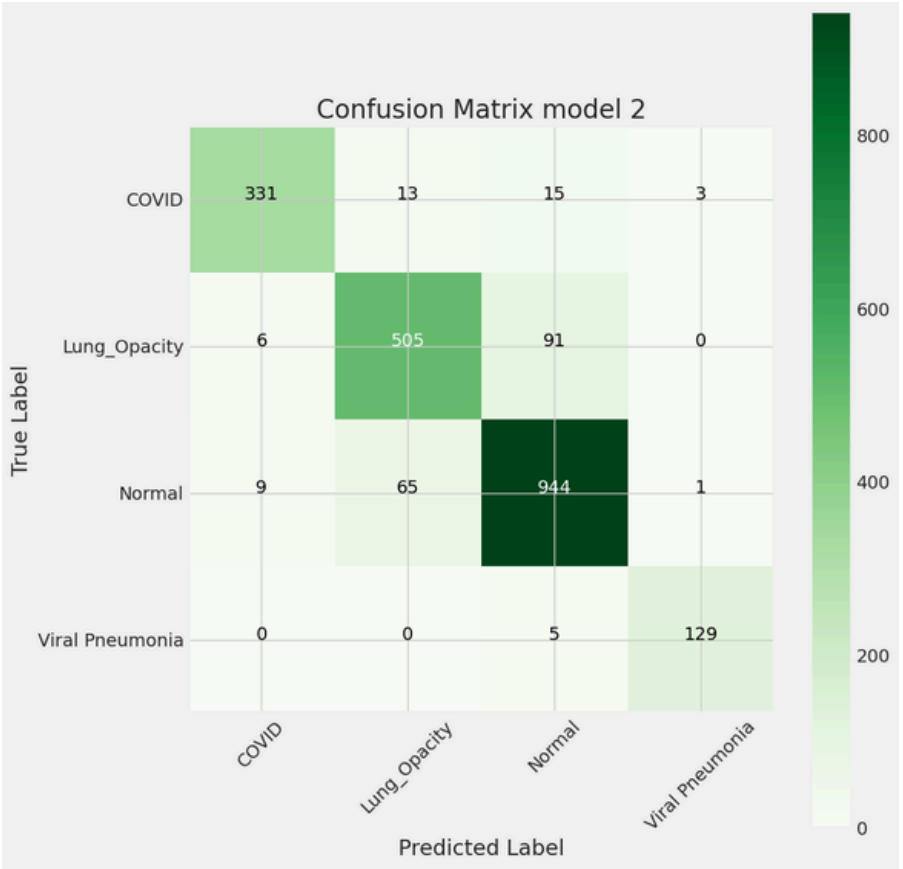
Model 2 achieves the highest training accuracy but suffers from overfitting. validation and test accuracies are slightly lower than Model 1, indicating reduced generalization. and requires further optimization to mitigate overfitting issues.





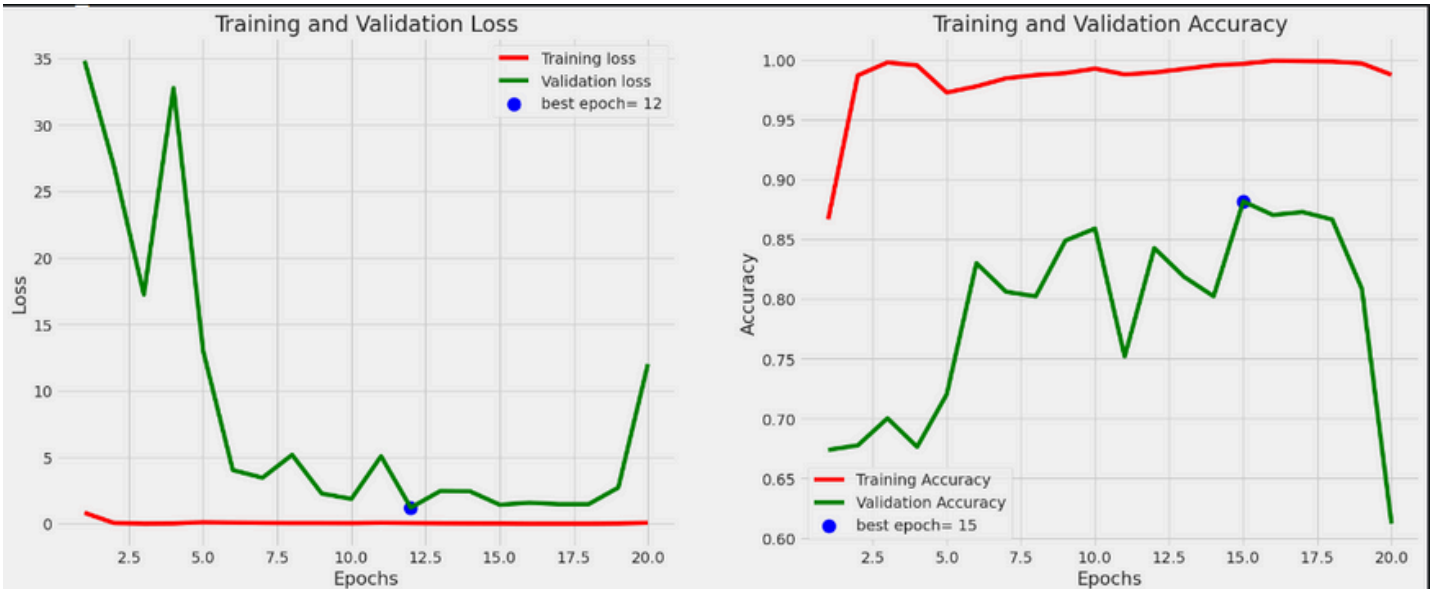
Confusion Matrix

	precision	recall	f1-score	support
Class 1	0.56	0.84	0.67	135
Class 2	0.82	0.81	0.82	226
Class 3	0.87	0.79	0.83	383
Class 4	1.00	0.30	0.46	50
accuracy			0.78	794
macro avg	0.81	0.69	0.69	794
weighted avg	0.81	0.78	0.77	794



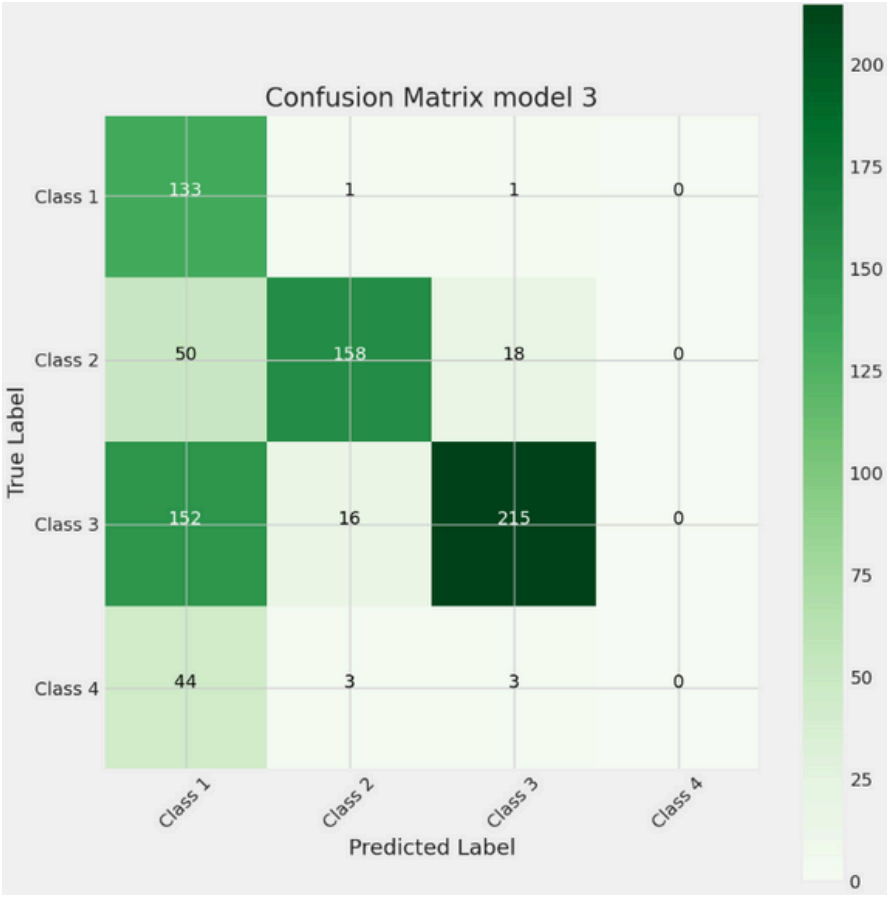
C. Two convolutional layers + the last three fully connected:

Model 3 Exhibits severe overfitting with significantly lower validation and test accuracies. it has poor generalization and high training loss indicate inadequacy for real-world deployment. and itequires substantial improvements before it can be considered for practical use.



Classification Report & Confusion Matrix

	precision	recall	f1-score	support
Class 1	0.35	0.99	0.52	135
Class 2	0.89	0.70	0.78	226
Class 3	0.91	0.56	0.69	383
Class 4	0.00	0.00	0.00	50
accuracy			0.64	794
macro avg	0.54	0.56	0.50	794
weighted avg	0.75	0.64	0.65	794



4. **Conclusion:**

- while Model 2 shows the highest test accuracy, further optimization is needed to mitigate overfitting issues and improve generalization. Model 1, with slightly lower accuracy but better generalization, might be a more reliable choice for real-world deployment. Model 3 requires substantial improvements before it can be considered for practical use.

- The depth of fine-tuning is an important factor to consider, as it determines how many layers are being updated during training. In general, fine-tuning only the last few layers is often sufficient for achieving good performance, as the earlier layers of the network are already able to extract useful features. However, in some cases, fine-tuning more layers may be necessary to achieve optimal performance as we have seen in this casees