A Novel Data Augmentation-Based Brain Tumor Detection Using Convolutional Neural Network

• It is crucial to detect brain tumors early enough for successful treatment.

• Brain tumors are the 10^{th} leading cause of death worldwide. In 2020, almost 308,102 people were diagnosed with brain tumors.

• In recent years, deep learning has contributed a lot to the health industry medical diagnoses. CNNS have shown great performance in detecting many diseases including brain tumors.

Dataset

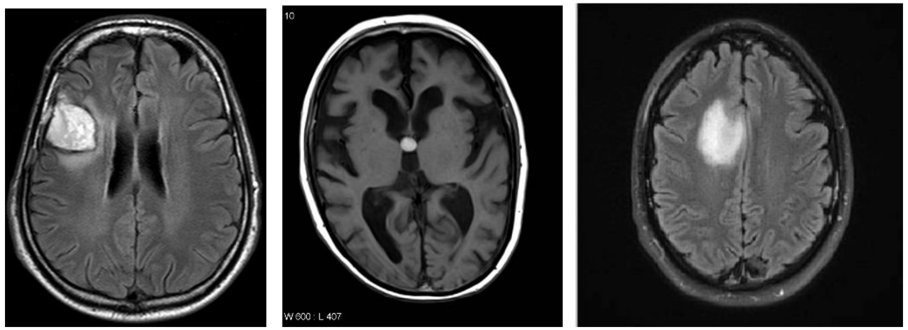
- We work on a dataset from kaggle.com:

Brain MRI Images for Brain Tumor Detection

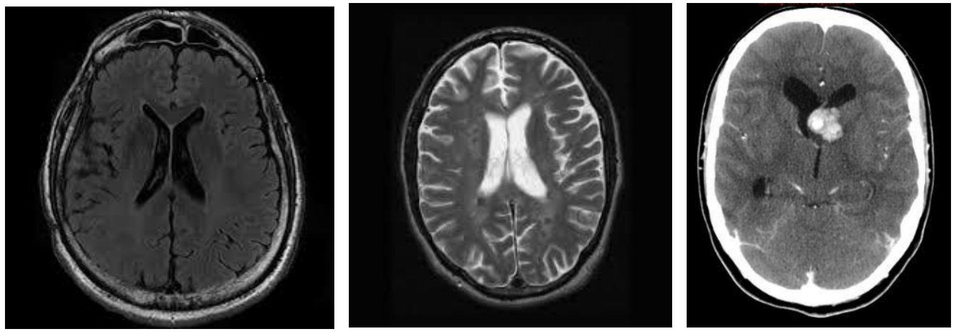
- The dataset includes 253 images of brain with 155 positive cases and 98 negative cases.

- Our task is to use a CNN model to predict whether or not one has a brain tumor given an image.jpg 224×224.

Dataset



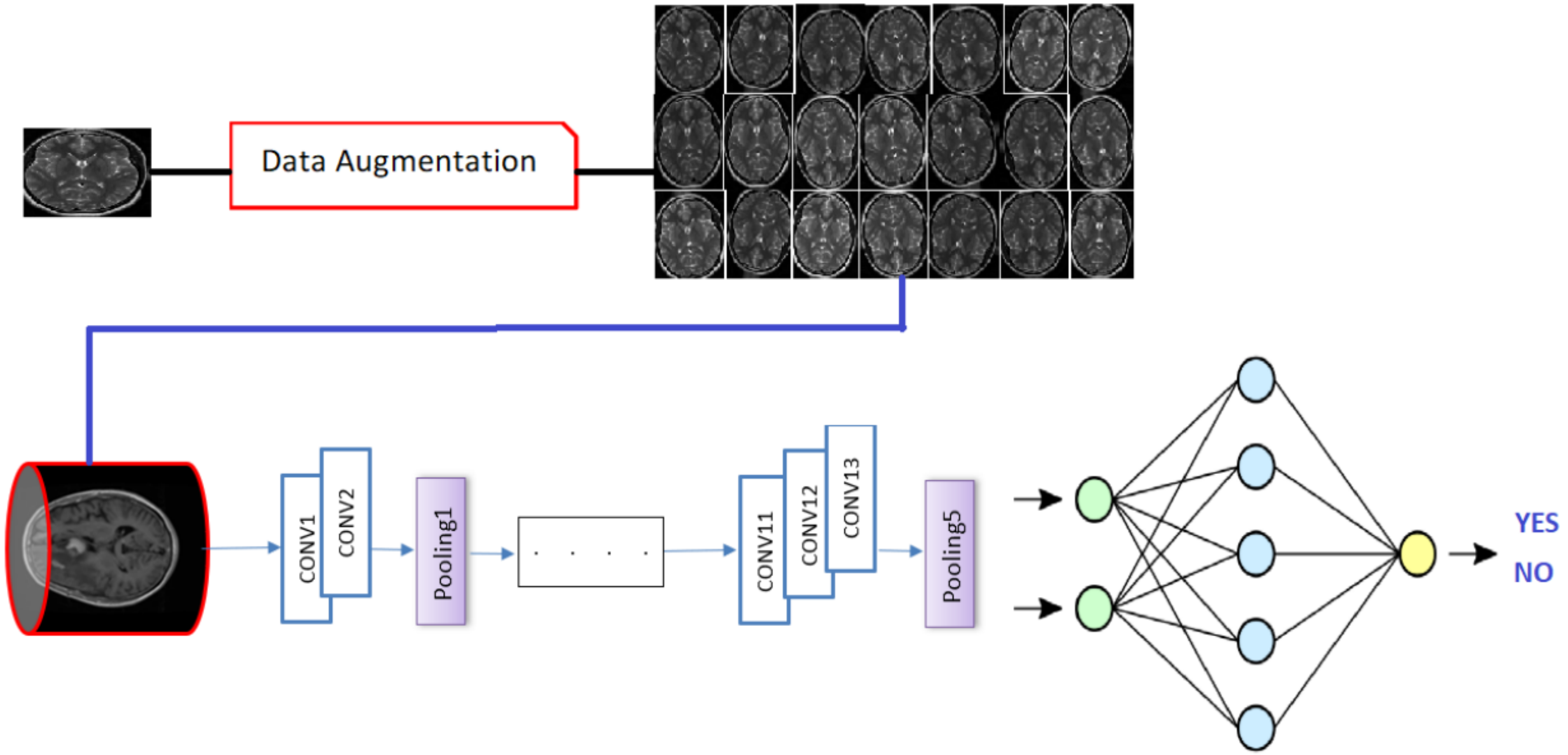
yes



no

• In this paper, we use the popular CNN called VGG-16 to predict the result. We will use Data Augmentation because 253 images are not enough.

Methodology

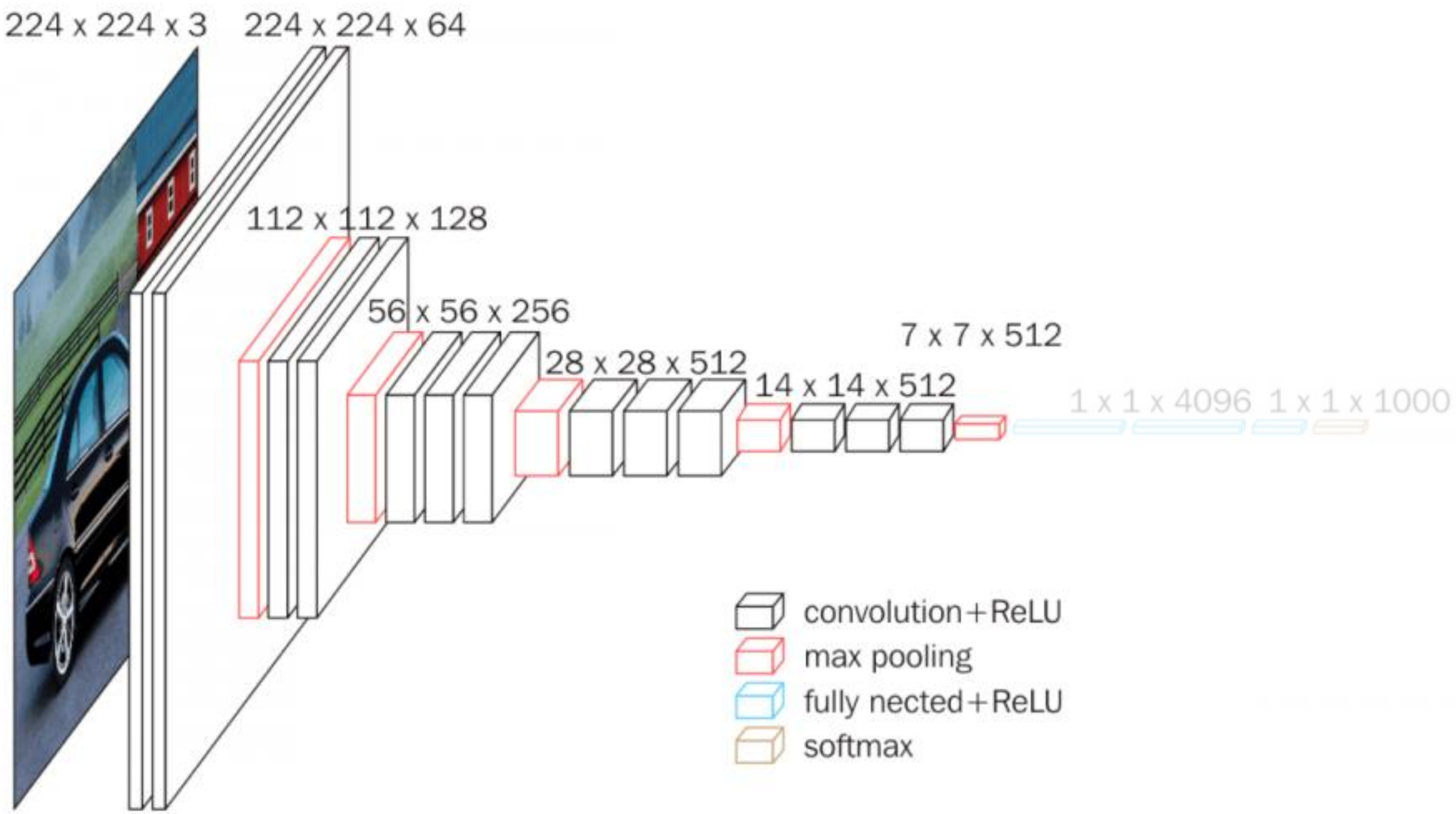


- We make use of a VGG16 CNN up to the ending pooling layer.
- Then we add a [2, 5, 1] fully connected neural network.

Implementation

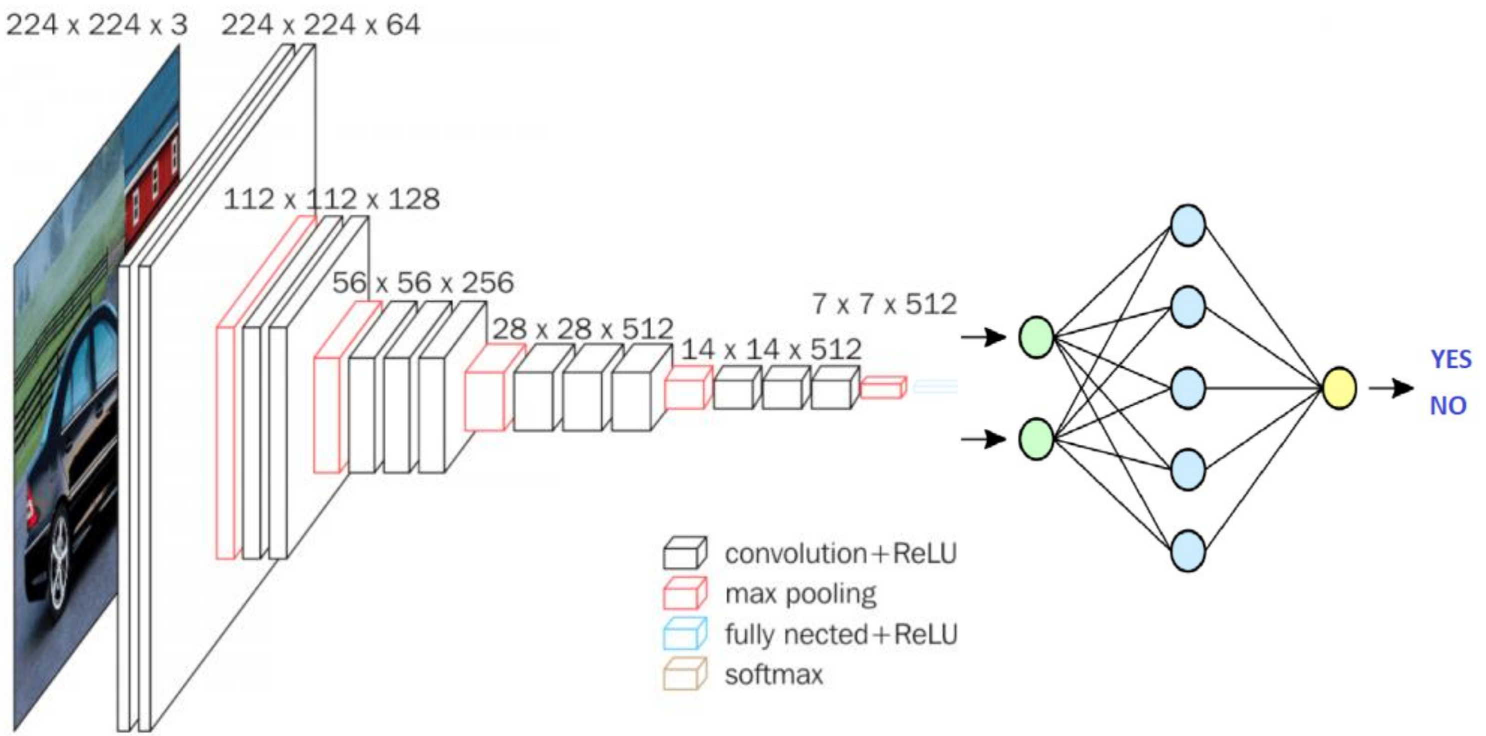
```python
from tensorflow.keras.applications.vgg16 import VGG16
```

```python
vgg16 = VGG16( weights='imagenet', include_top=False, input_shape=(224,224,3))
```



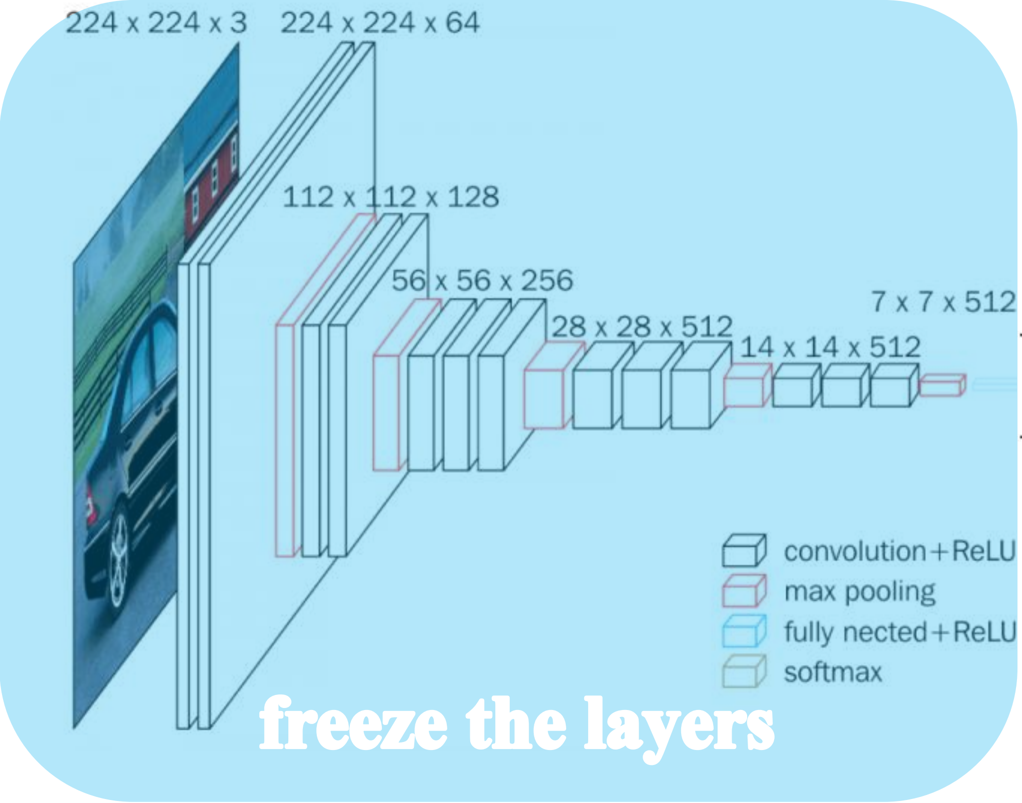
Implementation

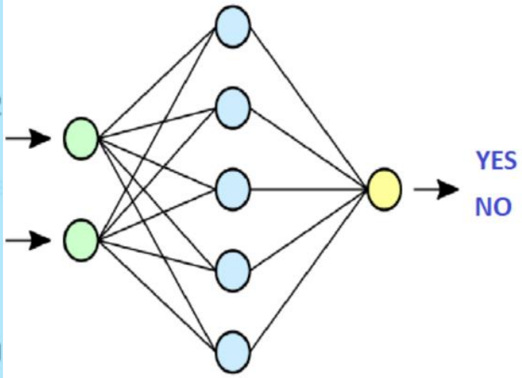
```
vgg16.trainable=False

myModel = tf.keras.Sequential()
myModel.add(vgg16)
myModel.add(tf.keras.layers.Flatten())
myModel.add (keras.layers.Dense(2,activation='relu'))
myModel.add (keras.layers.Dense(5, activation='relu'))
myModel.add (keras.layers.Dense(1,activation='sigmoid'))
```



Implementation

```python
vgg16.trainable=False

myModel = tf.keras.Sequential()
myModel.add(vgg16)
myModel.add(tf.keras.layers.Flatten())
myModel.add (keras.layers.Dense(2,activation='relu'))
myModel.add (keras.layers.Dense(5, activation='relu'))
myModel.add (keras.layers.Dense(1,activation='sigmoid'))
```



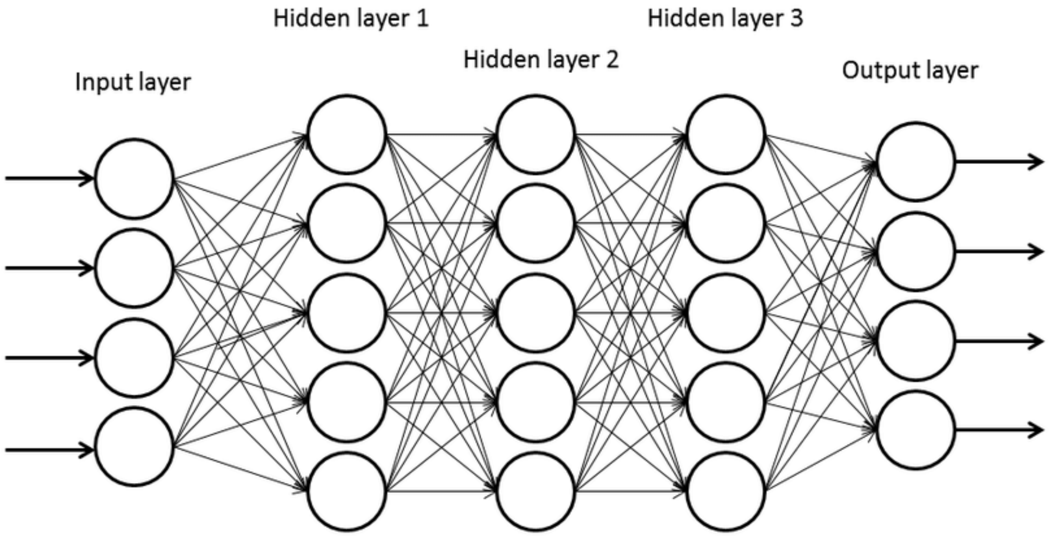


14M weights → 50K weights

آرش رحمتی

Transfer Learning

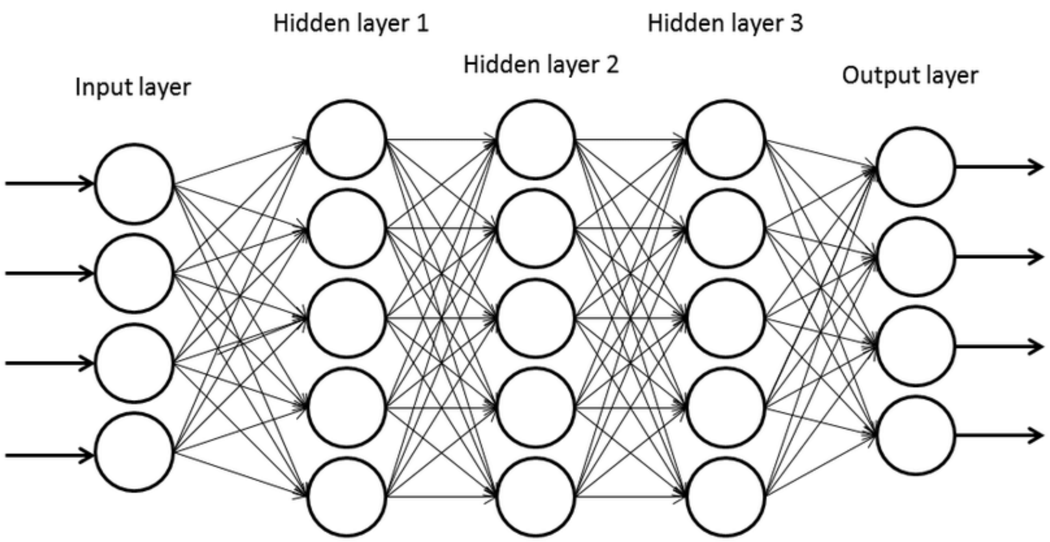
x_train



y_train

output: WEIGHTS

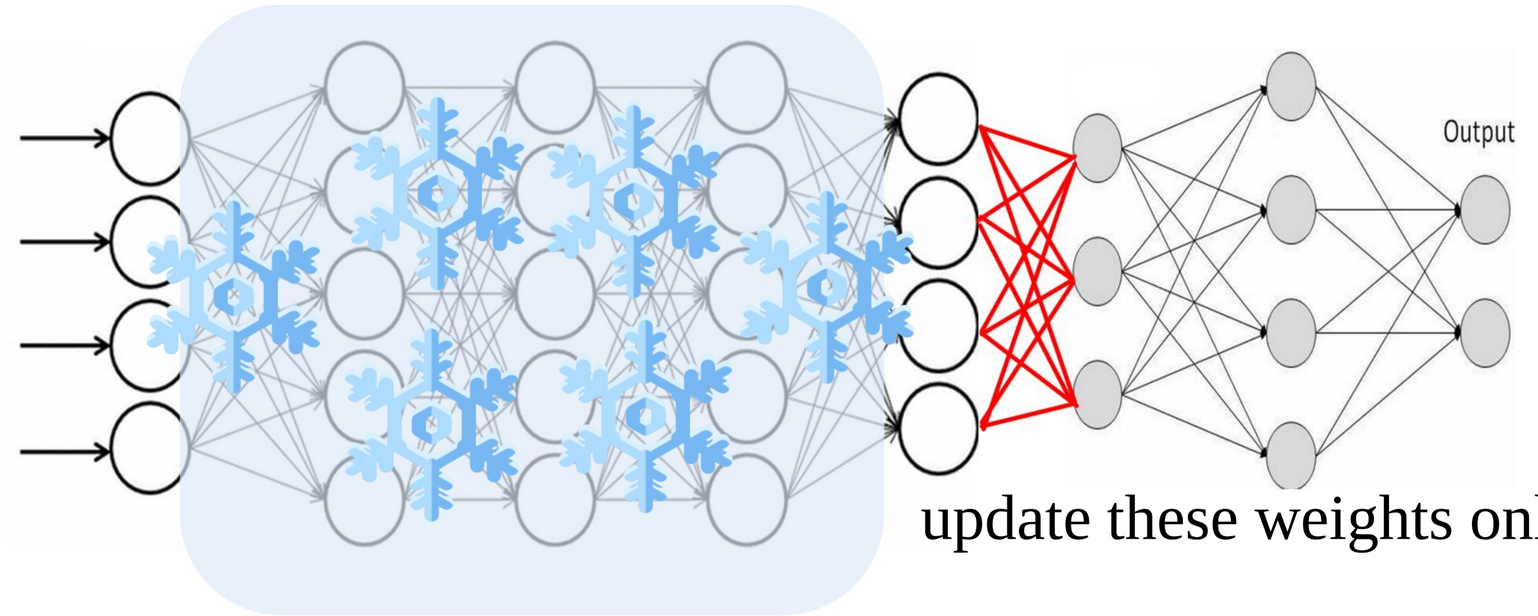
Transfer Learning



x_train

y_train

output: WEIGHTS



x_train2

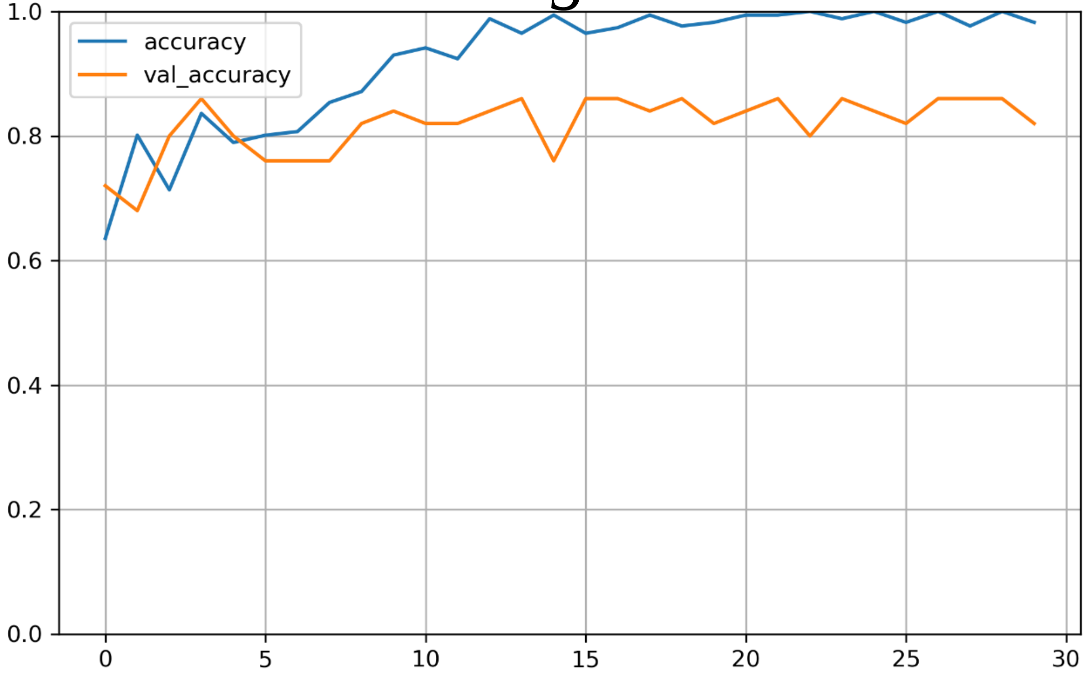
y_train2

update these weights only

Implementation and results

- For comparison, we first train this model on the data we have. (No Augmentation) The result will be:

without augmentation

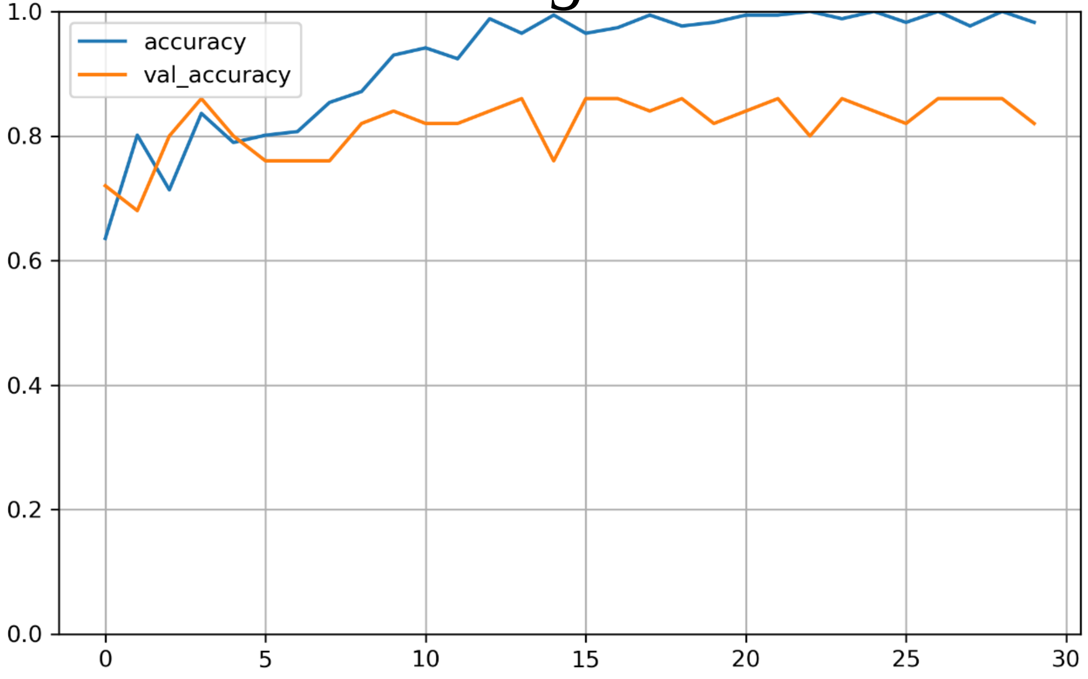


آرش رحمتی

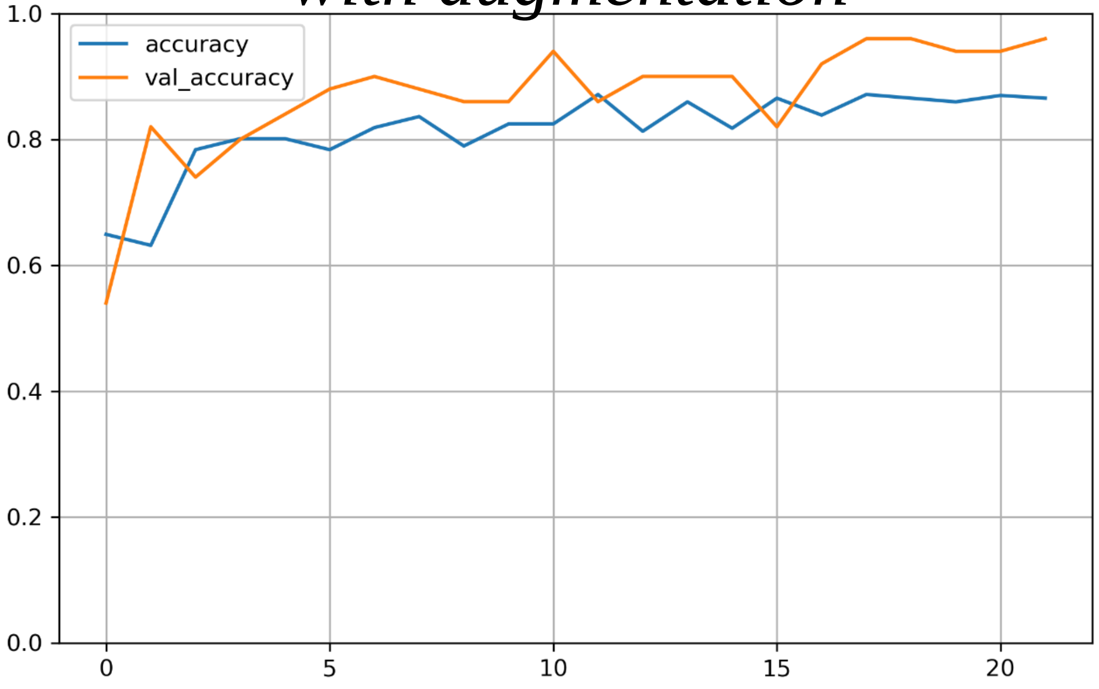
Implementation and results

- For comparison, we first train this model on the data we have. (No Augmentation) The result will be:

without augmentation



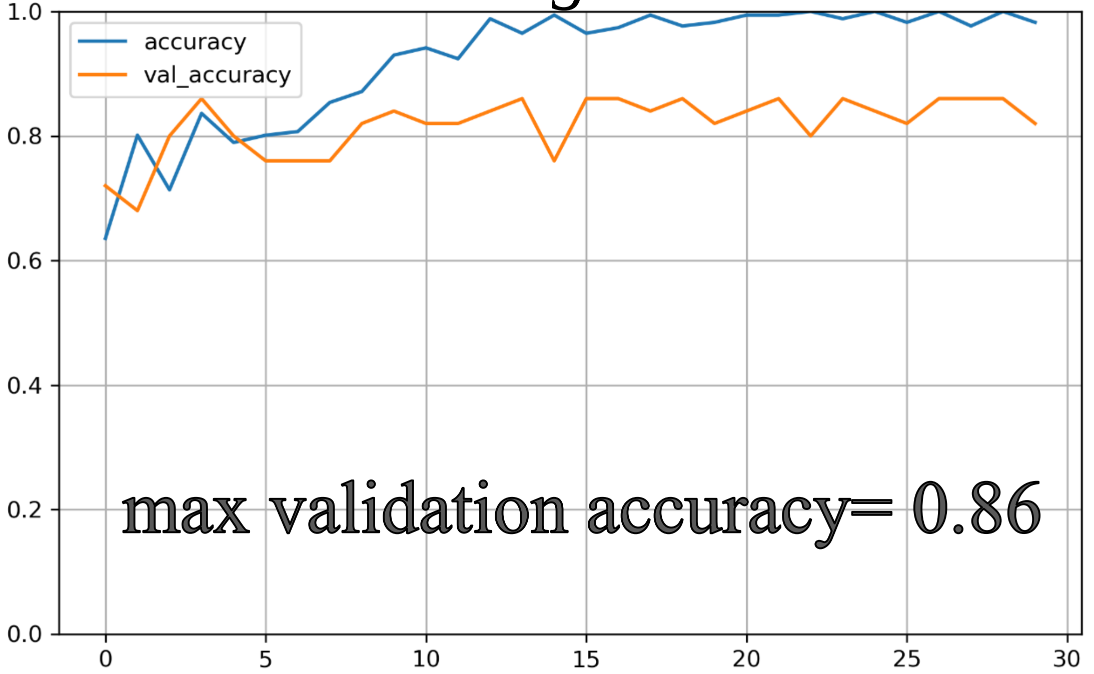
with augmentation



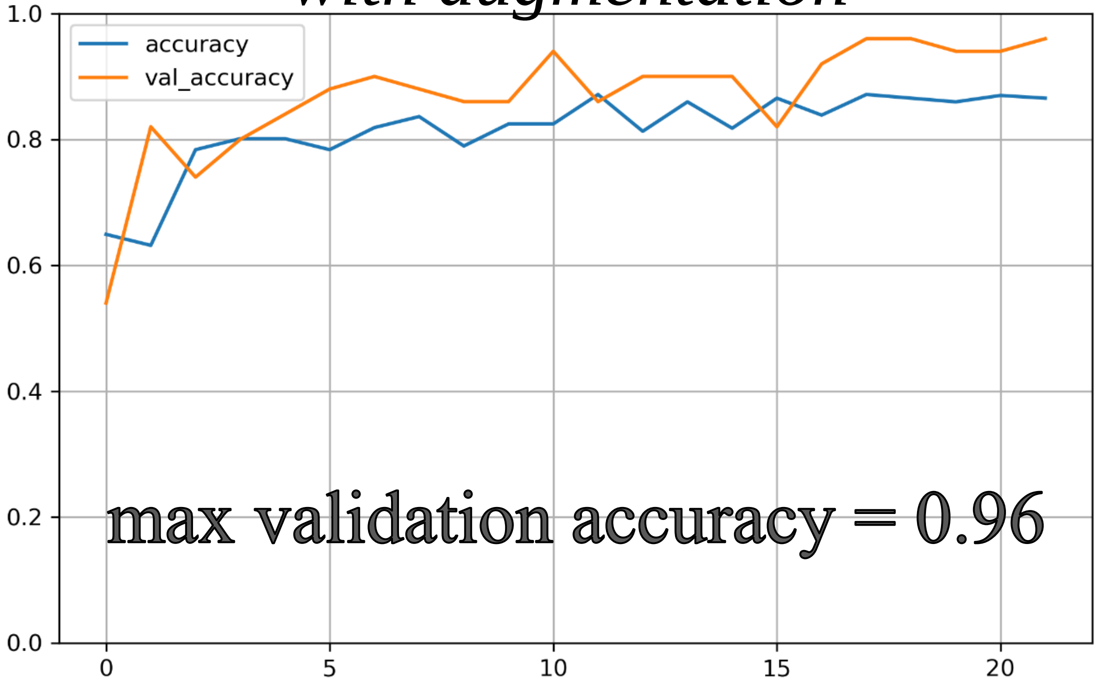
Implementation and results

- For comparison, we first train this model on the data we have. (No Augmentation) The result will be:

without augmentation



with augmentation



Implementation tips

- During training we will use: *ModelCheckPoint* and *Earlystopping* call backs

```
temp = tf.keras.callbacks.ModelCheckpoint(
    filepath="presentOne",
    save_weights_only=False,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

temp2 = tf.keras.callbacks.EarlyStopping(
    monitor="accuracy",
    patience=10)
```

**meaning: save the model with
maximum val_acc
that has been gained so far**

**meaning: stop if train_acc is not
improving for 10 consecutive
epochs**

Implementation tips

• During training we will use a library called *ImageDataGenerator* for data augmentation

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
train_datagen2 = ImageDataGenerator(rescale=1./255,
                                    rotation_range=15,
                                    width_shift_range=0.1,
                                    height_shift_range=0.1,
                                    shear_range=0.1,
                                    brightness_range=[0.5, 1.5],
                                    horizontal_flip=True,
                                    vertical_flip=True,
                                   preprocessing_function=preprocess_input)

test_datagen2 = ImageDataGenerator(preprocessing_function=preprocess_input)
```

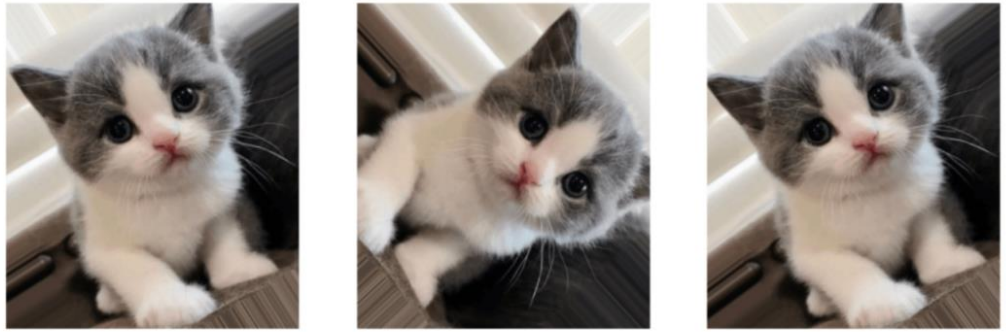
Implementation tips

- How ImageDataGenerator works:

width_shift

height_shift

rotation



Implementation tips

- How ImageDataGenerator works:

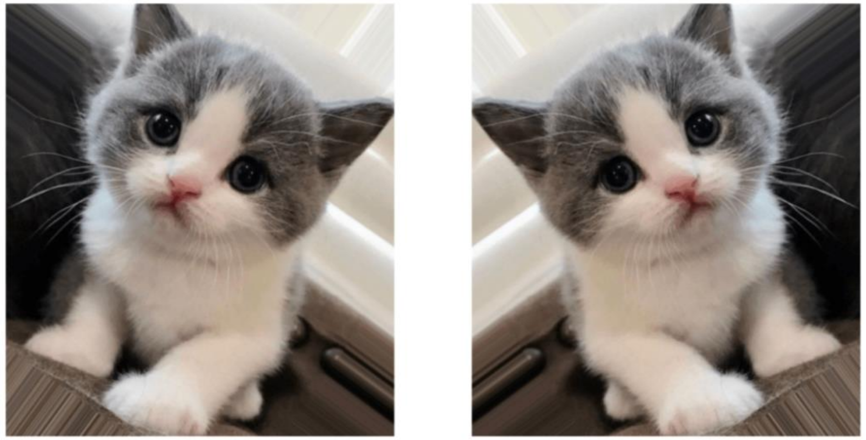
brightness

shear

Implementation tips

- How ImageDataGenerator works:

horizontal flip



vertical flip

Implementation tips

- Why ImageDataGenrator?

- Because it provides *real-time* data augmentation.

- We don’t have to save thousands of images anywhere. They will be produced automatically when needed and then removed.

Finally

- Now let's go to python implementation and compare results with the article.

Table 4. Comparison table between different models.

Model	Accuracy	Precision	Recall	F1-Score
VGG16	0.96	0.93	1.0	0.97
ResNet-50	0.89	0.87	0.93	0.90
VGG-19	0.93	0.94	0.93	0.93
Inception-V3	0.75	0.77	0.71	0.74
ResNet-101	0.74	0.74	0.74	0.73
DenseNet121	0.49	0.50	0.48	0.49
[69]	0.97	0.98	0.95	0.96
[70]	0.96	0.96	0.98	0.95
[71]	0.97	0.97	0.97	0.97
[72]	0.79	0.76	0.86	0.81
[73]	0.96	0.97	0.80	0.88