فاطمه توکلی_۴۰۰۱۳۱۰۱۶

۱. شبکه lenet5 از ۵ لایه تشکیل شده است که دو لایه اخر fully-connected و بقیه ترکیبی از لایه کانولوشنال و poling می‌باشد ورودی این مدل یک تصویر ۳۲ * ۳۲ در مقیاس خاکستری است، اولین لایه کانولوشن را با اندازه فیلتر ۵*۵ و ۶ فیلتر داریم. در نتیجه، ما یک نقشه ویژگی به اندازه ۲۸*۲۸*۶ دریافت می کنیم. در اینجا تعداد کانال ها برابر با تعداد فیلترهای اعمال شده است.سپس pooling را اعمال می‌کنیم و نقشه ویژگی نصف میشود ولی تعداد کانال ها ثابت است. دوباره لایه کانولوشن با اندازه فیلتر ۵*۵ داریم و نقشه ویژگی به ۱۰*۱۰*۱۶ تغییر می‌یابد و دوباره subsampling, average pooling اعمل میشود ونقشه ویژگی تبدیل به ۵*۵*۱۶ میشود در اخر یک لایه کانولوشن با اندازه فیلتر ۵*۵ و تعدا فیلتر ۱۲۰ خواهیم داشت که به نقشه ویژگی را به ۱*۱*۱۲۰ میرساند و در نهایت ۲ لایه dense داریم که در لایه اخر با تابع فعالسازی softmax کلاس بندی انجام می‌شود

Inception v3 یک مدل تشخیص تصویر است که نشان داده شده است که در مجموعه داده ImageNet دقتی بیش از ۷۸.۱٪ دارد. معماری شبکه Inception v3 به صورت تدریجی و گام به گام ساخته می شود که در زیر توضیح داده شده است:

Convolutions Factorized این به کاهش کارایی محاسباتی کمک می کند زیرا تعداد پارامترهای درگیر در یک شبکه را کاهش می دهد همچنین کارایی شبکه را بررسی می کند.

Smaller convolution جایگزینی کانولوشن های بزرگتر با پیچ های کوچکتر قطعاً منجر به آموزش سریعتر می شود.

Asymmetric convolutions یک پیچیدگی ۳×۳ را می‌توان با یک پیچش ۱×۳ و به دنبال آن یک پیچش ۳×۱ جایگزین کرد. اگر کانولوشن ۳ × ۳ با پیچ ۲ × ۲ جایگزین شود، تعداد پارامترها کمی بیشتر از پیچیدگی نامتقارن پیشنهادی خواهد بود.

Auxiliary classifier یک طبقه‌بندی‌کننده کمکی یک CNN کوچک است که در طول آموزش بین لایه‌ها قرار می‌گیرد.

و کاهش اندازه شبکه معمولا با عملیات ادغام انجام می شود.

در کل پارامتر ها در این مدل کاهش یافته در حالیکه قابلیت تعمیم پذیر افزایش داشته است.

برتری inception v3 به lenet5 تعداد لایه ها و همچنین ورودی ۳ کانال رنگی ان است.

۲. ابتدا داده‌ها را در سه بخش test,train,validation بارگذاری می‌کنیم و سپس نمونه اول از هرکدام را به نمایش می‌گذاریم و به عنوان پیش پردازش ابتدا سایز عکس ها را تغییر داده و سپس scale می‌کنیم:

```
ds_train , info = tfds.load('beans', split='train', shuffle_files=True, with_info=True)
ds_test = tfds.load('beans', split='train', shuffle_files=True)
ds_valid = tfds.load('beans', split='validation', shuffle_files=True)
```

اطلاعات دیتاست:

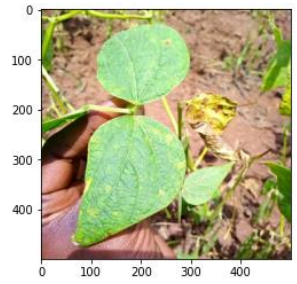
```
1  info.features
```

```
FeaturesDict({
    'image': Image(shape=(500, 500, 3), dtype=tf.uint8),
    'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=3),
})
```

نمونه از هر مجموعه :

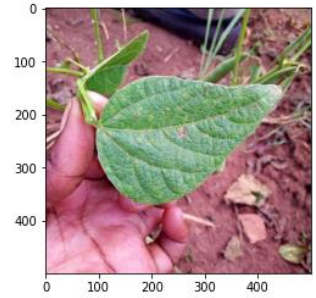
```python
for index, a in enumerate(ds_valid):
    plt.imshow(a['image'])
    plt.show()
    print(a['label'].numpy())
    print(tf.shape(a['image']))

    if index >=0:
        break
```



```
1
tf.Tensor([500 500   3], shape=(3,), dtype=int32)
```

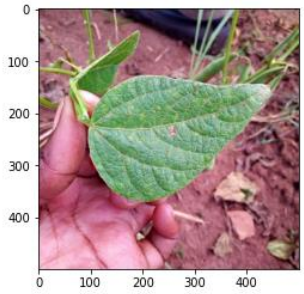
```python
for index, a in enumerate(ds_test):
    plt.imshow(a['image'])
    plt.show()
    print(a['label'].numpy())
    print(tf.shape(a['image']))
    if index >=0:
        break
```



```
1
tf.Tensor([500 500   3], shape=(3,), dtype=int32)
```

```python
for index, a in enumerate(ds_train):
    plt.imshow(a['image'])
    plt.show()
    print(a['label'].numpy())
    print(tf.shape(a['image']))

    if index >=0:
        break
```



```
1
tf.Tensor([500 500   3], shape=(3,), dtype=int32)
```

متدی برای تغییر سایز و scaling تعریف می‌کنیم و نیازی به تغییر برچسب هر عکس نیست:

```python
def augment_hue(tensor):
    return tf.image.resize(tensor['image'], (299,299)), tensor['label']

def normalize_image(image, label):
    return image / 255.0 , label
```

پیاده‌سازی Lenet5 و summery ان به صورت زیر می‌باشد:

```python
lenet_5_model = keras.models.Sequential([
    keras.layers.Conv2D(filters=6, kernel_size=(5,5), strides=1, input_shape=(299,299,3),  activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=16, kernel_size=(5,5), strides=1, activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(120,(5,5), activation='tanh'),
    keras.layers.Flatten(),
    keras.layers.Dense(84, activation='tanh'),
    keras.layers.Dense(3, activation='softmax')
])

lenet_5_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 295, 295, 6)       456

 average_pooling2d_8 (Averag  (None, 147, 147, 6)      0
 ePooling2D)

 conv2d_10 (Conv2D)          (None, 143, 143, 16)      2416

 average_pooling2d_9 (Averag  (None, 71, 71, 16)       0
 ePooling2D)

 conv2d_11 (Conv2D)          (None, 67, 67, 120)       48120

 flatten_3 (Flatten)         (None, 538680)            0

 dense_8 (Dense)             (None, 84)                45249204

 dense_9 (Dense)             (None, 3)                 255

=================================================================
Total params: 45,300,451
Trainable params: 45,300,451
Non-trainable params: 0
_________________________________________________________________
```

دقت بدون اعمال تنظیم کننده و تغییرات دیگر در ساختار شبکه lenet5:

```
lenet_5_model.compile(
    loss=keras.losses.sparse_categorical_crossentropy,
    optimizer=tf.keras.optimizers.Adam(),
    metrics=['accuracy'],
)
```

```
BATCH_SIZE = 3
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)

lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)

```

```
Epoch 1/3
344/344 [==============================] - 124s 356ms/step - loss: 1.1170 - accuracy: 0.3411
Epoch 2/3
344/344 [==============================] - 116s 338ms/step - loss: 1.1155 - accuracy: 0.3565
Epoch 3/3
344/344 [==============================] - 117s 339ms/step - loss: 1.1164 - accuracy: 0.3507
<keras.callbacks.History at 0x7f2c02f17a90>
```

تنظیم کننده زیر را به ۲ لایه‌ی کانولوشنی به ترتیب اعمال میکنیم:

```
kernel_regularizer=tf.keras.regularizers.L1(l1=0.01)
```

```
lenet_5_model = keras.models.Sequential([
    keras.layers.Conv2D(filters=6, kernel_size=(5,5), strides=1, input_shape=(299,299,3),  activation='tanh', padding='valid',kernel_regularizer=tf.keras.regularizers.L1(l1=0.01)),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=16, kernel_size=(5,5), strides=1, activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(120,(5,5), activation='tanh'),
    keras.layers.Flatten(),
    keras.layers.Dense(84, activation='tanh'),
    keras.layers.Dense(3, activation='softmax')
])
```

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 142s 1s/step - loss: 3.9137 - accuracy: 0.3240
Epoch 2/3
103/103 [==============================] - 137s 1s/step - loss: 1.2242 - accuracy: 0.3462
Epoch 3/3
103/103 [==============================] - 137s 1s/step - loss: 1.2046 - accuracy: 0.3529
<keras.callbacks.History at 0x7f9827e4fd90>
```

```
lenet_5_model = keras.models.Sequential([
    keras.layers.Conv2D(filters=6, kernel_size=(5,5), strides=1, input_shape=(299,299,3),  activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=16, kernel_size=(5,5), strides=1, activation='tanh', padding='valid',kernel_regularizer=tf.keras.regularizers.L1(l1=0.01)),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(120,(5,5), activation='tanh'),
    keras.layers.Flatten(),
    keras.layers.Dense(84, activation='tanh'),
    keras.layers.Dense(3, activation='softmax')
])
```

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 139s 1s/step - loss: 4.2654 - accuracy: 0.3365
Epoch 2/3
103/103 [==============================] - 138s 1s/step - loss: 1.3657 - accuracy: 0.3413
Epoch 3/3
103/103 [==============================] - 137s 1s/step - loss: 1.1673 - accuracy: 0.3317
<keras.callbacks.History at 0x7f9827cbd090>
```

```
lenet_5_model = keras.models.Sequential([
    keras.layers.Conv2D(filters=6, kernel_size=(5,5), strides=1, input_shape=(299,299,3),  activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=16, kernel_size=(5,5), strides=1, activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(120,(5,5), activation='tanh',kernel_regularizer=tf.keras.regularizers.L1(l1=0.01)),
    keras.layers.Flatten(),
    keras.layers.Dense(84, activation='tanh'),
    keras.layers.Dense(3, activation='softmax')
])
```

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 139s 1s/step - loss: 6.3824 - accuracy: 0.3029
Epoch 2/3
103/103 [==============================] - 137s 1s/step - loss: 2.6022 - accuracy: 0.3385
Epoch 3/3
103/103 [==============================] - 144s 1s/step - loss: 1.8768 - accuracy: 0.3433
<keras.callbacks.History at 0x7f9827b4e210>
```

`kernel_regularizer=tf.keras.regularizers.L2(l2=0.01)`

```
lenet_5_model = keras.models.Sequential([
    keras.layers.Conv2D(filters=6, kernel_size=(5,5), strides=1, input_shape=(299,299,3),  activation='tanh', padding='valid',kernel_regularizer=tf.keras.regularizers.L2(l2=0.01)),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=16, kernel_size=(5,5), strides=1, activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(120,(5,5), activation='tanh'),
    keras.layers.Flatten(),
    keras.layers.Dense(84, activation='tanh'),
    keras.layers.Dense(3, activation='softmax')
])
```

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 138s 1s/step - loss: 4.2967 - accuracy: 0.3385
Epoch 2/3
103/103 [==============================] - 137s 1s/step - loss: 1.1850 - accuracy: 0.3173
Epoch 3/3
103/103 [==============================] - 137s 1s/step - loss: 1.1379 - accuracy: 0.3452
<keras.callbacks.History at 0x7f9827a2a510>
```

```
lenet_5_model = keras.models.Sequential([
    keras.layers.Conv2D(filters=6, kernel_size=(5,5), strides=1, input_shape=(299,299,3),  activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=16, kernel_size=(5,5), strides=1, activation='tanh', padding='valid',kernel_regularizer=tf.keras.regularizers.L2(l2=0.01)),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(120,(5,5), activation='tanh'),
    keras.layers.Flatten(),
    keras.layers.Dense(84, activation='tanh'),
    keras.layers.Dense(3, activation='softmax')
])
```

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 140s 1s/step - loss: 3.1916 - accuracy: 0.3481
Epoch 2/3
103/103 [==============================] - 136s 1s/step - loss: 1.1304 - accuracy: 0.3462
Epoch 3/3
103/103 [==============================] - 136s 1s/step - loss: 1.1158 - accuracy: 0.3481
<keras.callbacks.History at 0x7f982788bf90>
```

```
lenet_5_model = keras.models.Sequential([
    keras.layers.Conv2D(filters=6, kernel_size=(5,5), strides=1, input_shape=(299,299,3),  activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=16, kernel_size=(5,5), strides=1, activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(120,(5,5), activation='tanh',kernel_regularizer=tf.keras.regularizers.L2(l2=0.01)),
    keras.layers.Flatten(),
    keras.layers.Dense(84, activation='tanh'),
    keras.layers.Dense(3, activation='softmax')
])
```

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 137s 1s/step - loss: 3.3366 - accuracy: 0.3106
Epoch 2/3
103/103 [==============================] - 144s 1s/step - loss: 1.1284 - accuracy: 0.3385
Epoch 3/3
103/103 [==============================] - 138s 1s/step - loss: 1.1420 - accuracy: 0.3337
<keras.callbacks.History at 0x7f982776f1d0>
```

اضافه کردن لایه‌ی dropout :

```
lenet_5_model = keras.models.Sequential([
    keras.layers.Conv2D(filters=6, kernel_size=(5,5), strides=1, input_shape=(299,299,3),  activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(filters=16, kernel_size=(5,5), strides=1, activation='tanh', padding='valid'),
    keras.layers.AveragePooling2D(pool_size=(2,2)),
    keras.layers.Conv2D(120,(5,5), activation='tanh'),
    keras.layers.Dropout(.2)
    keras.layers.Flatten(),
    keras.layers.Dense(84, activation='tanh'),
    keras.layers.Dense(3, activation='softmax')
])
```

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 143s 1s/step - loss: 2.9038 - accuracy: 0.3548
Epoch 2/3
103/103 [==============================] - 142s 1s/step - loss: 1.1084 - accuracy: 0.3346
Epoch 3/3
103/103 [==============================] - 142s 1s/step - loss: 1.1078 - accuracy: 0.3462
<keras.callbacks.History at 0x7f98275cbd90>
```

افزایش تعداد کرنل:

Number of kernel = 9

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 90s 859ms/step - loss: 1.2990 - accuracy: 0.3308
Epoch 2/3
103/103 [==============================] - 89s 858ms/step - loss: 1.1160 - accuracy: 0.3385
Epoch 3/3
103/103 [==============================] - 87s 844ms/step - loss: 1.1137 - accuracy: 0.3298
<keras.callbacks.History at 0x7f9875f5ae90>
```

```
BATCH_SIZE = 10
dataset_test = normalized_dataset_test.batch(BATCH_SIZE)
test_loss,test_accuracy = lenet_5_model.evaluate(dataset_test,verbose=2)
```

```
104/104 - 29s - loss: 1.0995 - accuracy: 0.3172 - 29s/epoch - 282ms/step
```

Number of kernel = 10,16

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 184s 2s/step - loss: 3.8423 - accuracy: 0.3356
Epoch 2/3
103/103 [==============================] - 184s 2s/step - loss: 1.1056 - accuracy: 0.3529
Epoch 3/3
103/103 [==============================] - 184s 2s/step - loss: 1.1052 - accuracy: 0.3606
<keras.callbacks.History at 0x7f98274efcd0>
```

Number of kernel = 20,36,140

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 323s 3s/step - loss: 3.3898 - accuracy: 0.3058
Epoch 2/3
103/103 [==============================] - 273s 3s/step - loss: 1.1133 - accuracy: 0.3375
Epoch 3/3
103/103 [==============================] - 273s 3s/step - loss: 1.1062 - accuracy: 0.3548
<keras.callbacks.History at 0x7f9827484a50>
```

افزایش سایز کرنل:

Kernel_size = 9

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 205s 2s/step - loss: 1.3768 - accuracy: 0.3423
Epoch 2/3
103/103 [==============================] - 231s 2s/step - loss: 1.1240 - accuracy: 0.3298
Epoch 3/3
103/103 [==============================] - 206s 2s/step - loss: 1.1172 - accuracy: 0.3365
<keras.callbacks.History at 0x7f2bfa479190>
```

```
BATCH_SIZE = 10
dataset_test = normalized_dataset_test.batch(BATCH_SIZE)
test_loss,test_accuracy = lenet_5_model.evaluate(dataset_test,verbose=2)
```

```
104/104 - 66s - loss: 1.1052 - accuracy: 0.3337 - 66s/epoch - 633ms/step
```

Kernel_size = 20

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 621s 6s/step - loss: 1.2450 - accuracy: 0.3317
Epoch 2/3
103/103 [==============================] - 617s 6s/step - loss: 1.1222 - accuracy: 0.3519
Epoch 3/3
103/103 [==============================] - 616s 6s/step - loss: 1.1171 - accuracy: 0.3385
<keras.callbacks.History at 0x7f2bfa22b890>
```

```
BATCH_SIZE = 10
dataset_test = normalized_dataset_test.batch(BATCH_SIZE)
test_loss,test_accuracy = lenet_5_model.evaluate(dataset_test,verbose=2)
```

```
104/104 - 125s - loss: 1.1000 - accuracy: 0.3337 - 125s/epoch - 1s/step
```

Kernel_size = 2

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
lenet_5_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=3)
```

```
Epoch 1/3
103/103 [==============================] - 269s 3s/step - loss: 1.3366 - accuracy: 0.3135
Epoch 2/3
103/103 [==============================] - 267s 3s/step - loss: 1.1179 - accuracy: 0.3260
Epoch 3/3
103/103 [==============================] - 269s 3s/step - loss: 1.1139 - accuracy: 0.3308
<keras.callbacks.History at 0x7f2bd6670fd0>
```

به طور کلی نتایج بهتر برای تعداد کرنل کمتر با سایز کوچکتر به دست آمده است

۳. مدل lenet با توجه به تعداد پارامترهای کم و همچنین تعداد محدود داده آموزشی برای این داده های سرعت کمتری دارد و سریعا overfit میشود و باعث کاهش کارایی این مدل میشود. در یادگیری انتقالی میتوان از دانش مسئله قبلی برای کامل کردن و بهبود مسائل دیگر استفاده کرد مثلا در inception با استفاده از وزن های آموزش دیده با imagenet میتوان با داده کمتر به نتایج بهتری دست یافت.

۴. از inception پیش‌آموزش دیده به صورت tf.keras.applications.InceptionV3 استفاده می‌کنیم و سپس پارامتر include_top را False قرار میدهیم تا بتوان لایه‌های آخر را تغییر دهیم و به صورت زیر پیاده می‌کنیم:

```
inception = tf.keras.applications.InceptionV3(weights='imagenet', include_top=False, input_shape=(299,299,3))
x = inception.output
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dense(84, activation='tanh')(x)
output = keras.layers.Dense(3, activation='softmax')(x)
inceptoin_model = keras.models.Model(inputs=inception.input, outputs=output)
```

```
for i in inception.layers:
    i.trainable = False
```

یک لایه pooling و یک لایه Dense قبل از خروجی اضافه می‌کنیم و لایه اخر را با ۳ کلاس و تابع فعالسازی softmax تعری می‌کنیم. و به ازای لایه‌ها پارامتر trainable را به False تغییر میدهیم.

```
BATCH_SIZE = 10
dataset = normalized_dataset_train.repeat().batch(BATCH_SIZE)
inceptoin_model.fit(dataset, steps_per_epoch=len(ds_train)/BATCH_SIZE, epochs=10)
```

```
Epoch 1/10
103/103 [==============================] - 203s 2s/step - loss: 0.7555 - accuracy: 0.7029
Epoch 2/10
103/103 [==============================] - 199s 2s/step - loss: 0.4252 - accuracy: 0.8346
Epoch 3/10
103/103 [==============================] - 198s 2s/step - loss: 0.3255 - accuracy: 0.8712
Epoch 4/10
103/103 [==============================] - 199s 2s/step - loss: 0.2709 - accuracy: 0.8971
Epoch 5/10
103/103 [==============================] - 199s 2s/step - loss: 0.2281 - accuracy: 0.9144
Epoch 6/10
103/103 [==============================] - 199s 2s/step - loss: 0.1731 - accuracy: 0.9423
Epoch 7/10
103/103 [==============================] - 199s 2s/step - loss: 0.1561 - accuracy: 0.9519
Epoch 8/10
103/103 [==============================] - 198s 2s/step - loss: 0.1135 - accuracy: 0.9663
Epoch 9/10
103/103 [==============================] - 200s 2s/step - loss: 0.1002 - accuracy: 0.9740
Epoch 10/10
103/103 [==============================] - 199s 2s/step - loss: 0.0897 - accuracy: 0.9779
<keras.callbacks.History at 0x7f06263acb90>
```

```
BATCH_SIZE = 10
dataset_test = normalized_dataset_test.batch(BATCH_SIZE)
test_loss,test_accuracy = inceptoin_model.evaluate(dataset_test,verbose=2)
```

```
104/104 - 198s - loss: 0.0952 - accuracy: 0.9739 - 198s/epoch - 2s/step
```

این مدل بر روی داده های imagenet آموزش دیده است و با فریز لایه های بیشتر سرعت آموزش و همگرایی بیشتر می‌شود.با توجه به تعداد پارامترهای زیاد این مدل و محدودیت تعداد نمونه های دیتاست انتخابی در صورت کاهش لایه های فریز شده به سرعت مدل overfit میشود.

این مدل در لایه های اول ویژگی های عمومی تصاویر را استخراج میکند و بر همین اساس لایه های ابتدایی فریز می شوند و لایه های انتها برای استخراج ویژگی های دقیق تر استفاده میشوند.

میزان همگرایی تعمیم‌پذیری و دقت شبکه پیش آموزش دیده inception v3 به نسبت شبکه lenet5 سریعتر و بهتر است. زیرا پارامترهای زیادی دارد میتواند ویژگی های بیشتری در ابعادهای مختلف از تصویر استخراج نماید و با توجه به اینکه از پیش آموزش داده شده است سرعت همگرایی بیشتری دارد و همچنین میزان تعمیم پذیری ان به نسبت lenet بیشتر است. و دقت بالاتری را برای ما بدست آورد.