دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

بسم الله الرحمن الرحیم

گزارش پروژه درس مقدمه ای

بر هوش محاسباتی

موضوع: پردازش تصویر

استاد: دکتر فرزانه عبدالهی

اعضای گروه:

ریحانه آهنی ۹۸۲۳۰۰۹

فاطمه رفیعی ۹۸۲۳۰۳۹

مهدیه سادات بنیس ۹۸۲۳۰۴۵

سمیرا سلجوقی ۹۸۲۳۰۴۸

فهرست مطالب

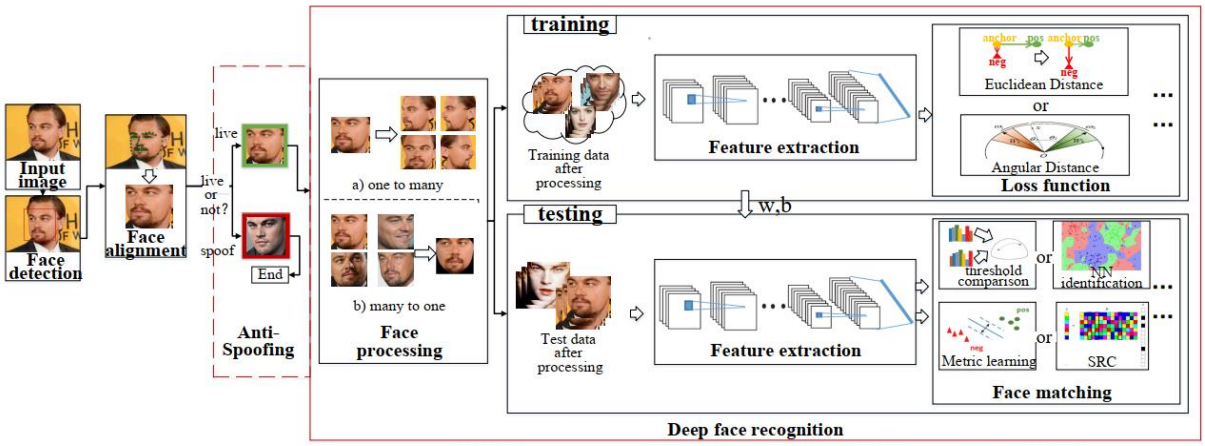
عناوین **صفحه**

فصل اول، مقدمه ای بر پردازش تصویر

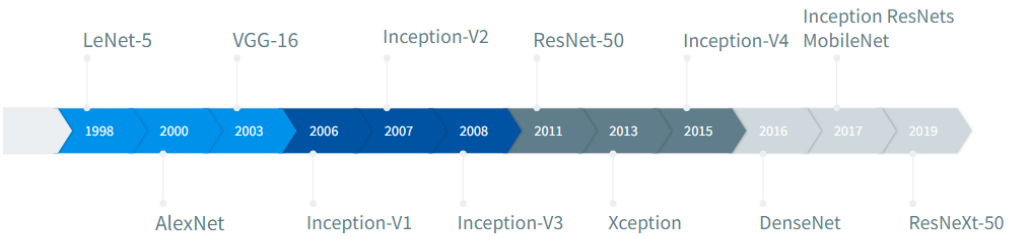
۱-۱ تشخیص چهره (FR)

تشخیص چهره (FR) تکنیکی است که برای تأیید یا شناسایی هویت یک فرد با تجزیه و تحلیل و مرتبط کردن الگوها بر اساس ویژگی‌های صورت فرد استفاده می‌شود.

در تصویر زیر ابتدا از یک آشکارساز چهره برای بومی سازی چهره ها استفاده می شود. دوم، چهره ها با مختصات متعارف عادی تراز شده اند. سوم، ماژول FR پیاده سازی شده است. در ماژول FR مرحله ای به نام Face Anti-Spoofing وجود دارد که تشخیص می دهد چهره زنده یا جعلی است. در ماژول دیگری به نام پردازش چهره، که برای رسیدگی به دشواری تشخیص قبل از آموزش و آزمایش استفاده می‌شود و در حین فرآیند آموزش، ویژگی عمیق تمایزآمیز استخراج می‌شود، ما از معماری‌ها و توابع از دست دادن مختلف استفاده می‌کنیم. در مورد روش های تطبیق چهره که برای انجام طبقه بندی ویژگی ها هنگام استخراج ویژگی عمیق داده های آزمایش استفاده می شود.[۱]

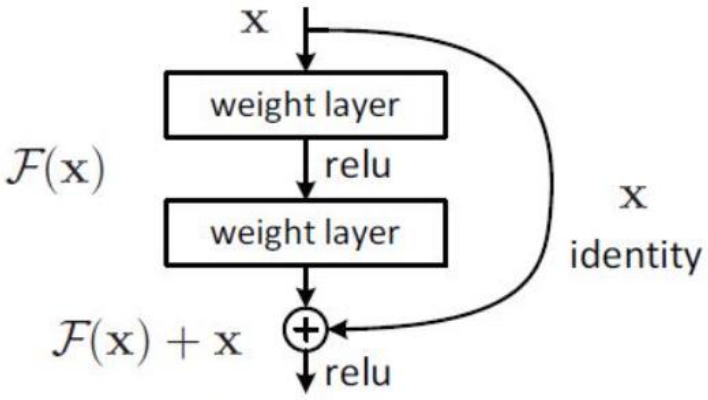


توسعه روش های مختلف پردازش چهره در طول سالیان به صورت زیر است:



۲-۱ معماری ResNet

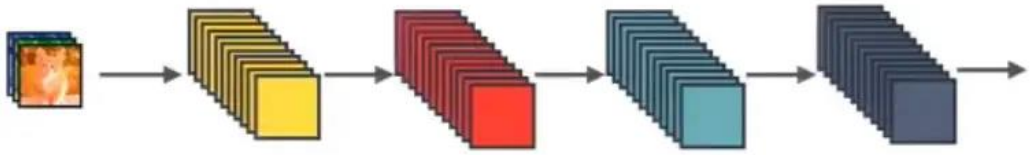
شبکه‌های یادگیری عمیق معمولی، مانند AlexNet ، ZFNet وVGGNet ، اغلب لایه‌های کانولوشنی و سپس لایه‌های کاملاً متصل (Fully Connected) برای طبقه‌بندی دارند، بدون هیچ‌گونه اتصال میان‌بر. ما در اینجا آن‌ها را شبکه‌های ساده (Plain Networks) می‌نامیم. وقتی شبکه‌ی ساده (Plain Networks) عمیق‌تر هستند (یعنی لایه‌ها افزایش می‌یابند)، مشکل محوشدگی گرادیان (Vanishing Gradient) یا انفجار گرادیان (Exploding Gradient) رخ می‌دهد؛ بنابراین عمیق‌ترکردن شبکه کار راحتی محسوب نمی‌شد که تنها با اضافه‌کردن لایه به شبکه آن را عمیق‌تر کنیم. اینجا بود که شبکه‌ی رزنت (ResNet) معرفی شد تا این مشکل را حل کند. این شبکه می‌تواند تا ۱۵۲ لایه داشته باشد.



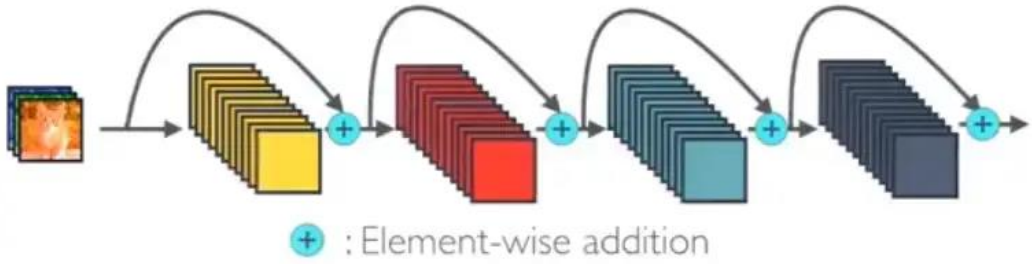
اتصالات میانبر (Skip Connections) یا اتصالات اضافی (Residual Connections) راه‌حلی بود که شبکه رزنت (ResNet) برای حل مشکل شبکه‌های عمیق ارائه کرد. در شکل ۱ یک بلاک اضافی (Residual Block)را مشاهده می‌کنیم. همان‌طورکه مشخص است، فرق این شبکه با شبکه‌های معمولی این است که یک اتصال میان‌بر دارد که از یک یا چند لایه عبور می‌کند و آن‌ها را در نظر نمی‌گیرد؛ درواقع به‌نوعی میان‌بر می‌زند و یک لایه را به لایه‌ی دورتر متصل می‌کند[۲].

۳-۱ معماری DenseNet

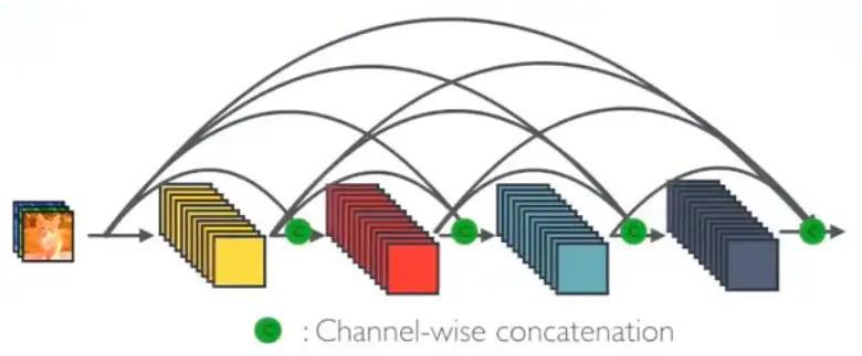
در شبکه ConvNet استاندارد، تصویر ورودی وارد چندین کانولوشن شده و ویژگی‌های سطح بالا دریافت می‌کند.



در ResNet برای ارتقای انتشار گرادیانی از تابع نگاشت همانی استفاده می‌شود .عملیات جمع مؤلفه‌ای به کاررفته را می‌توان به صورت الگوریتم‌هایی در نظر گرفت که حالتی را از یک ماژول ResNet به ماژول دیگر آن منتقل می‌کنند.

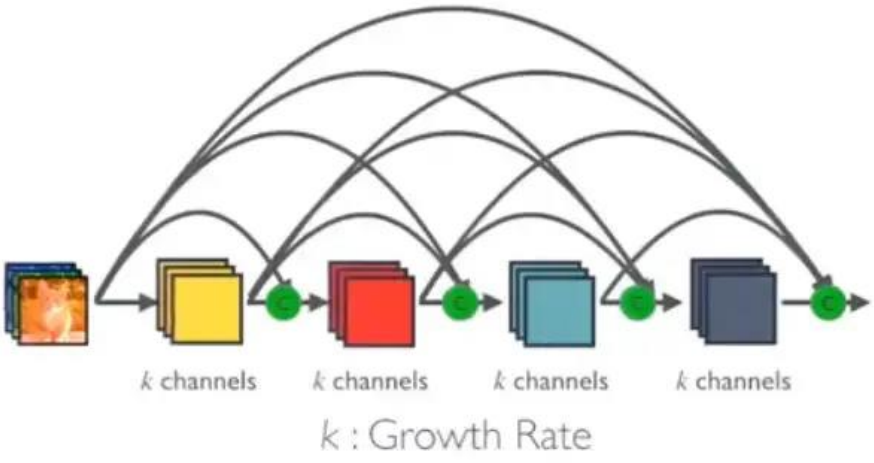


هرکدام از لایه‌های شبکه پیچشی متراکم ورودی‌هایی اضافی از همه‌ی لایه‌های قبلی دریافت و نگاشت‌های ویژگی خود را به لایه‌های بعدی منتقل می‌کند. از روش الحاق نیز می‌توان استفاده کرد؛ در این روش، هر لایه دانش جمعی همه‌ی لایه‌های قبلی را دریافت می‌کند.

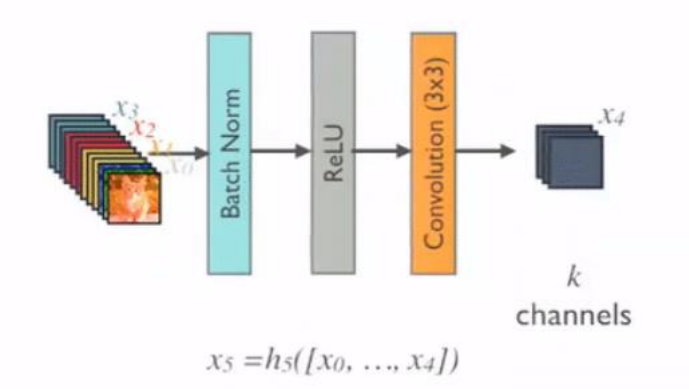


از آن‌جایی که هر لایه نگاشت‌های ویژگی همه‌ی لایه‌های قبلی را دریافت می‌کند، شبکه می‌تواند باریک‌تر و فشرده‌تر باشد؛ یعنی تعداد کانال‌های کمتری داشته باشد .نرخ رشد k معیاری است که تعداد کانال‌های اضافه شده در هر لایه را نشان می‌دهد.

بنابراین می‌توان گفت شبکه پیچشی متراکم از نظر محاسباتی و حافظه کارآیی بیشتری دارد.

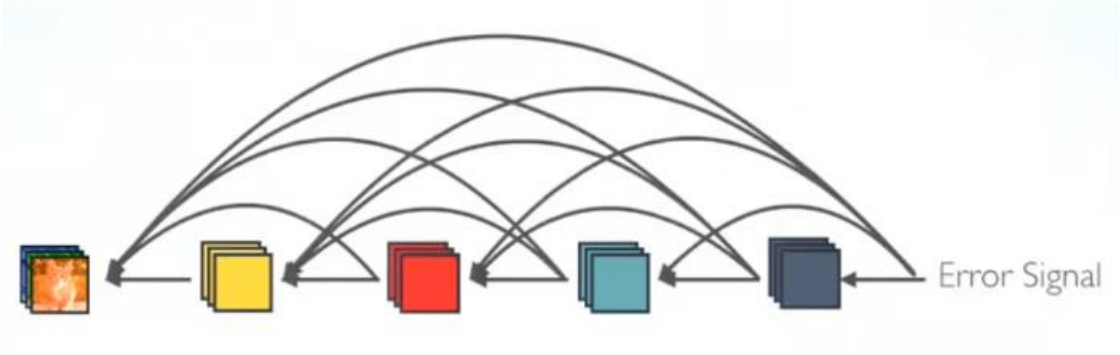


برای هر لایه‌ی تشکیل‌دهنده، تابع پیش-فعال‌سازی BN (Batch Norm) و ReLU و سپس کانولوشن ۳×۳ را اجرا می‌کنیم؛ خروجی این توابع نگاشت‌های ویژگی از k کانال است که، برای مثال، به منظور تبدیل x_0, x_1, x_2, ۳ به x_4 مورد استفاده قرار می‌گیرند. ایده‌ی زیربنایی این مرحله از Pre-Activation *ResNet* گرفته شده است.



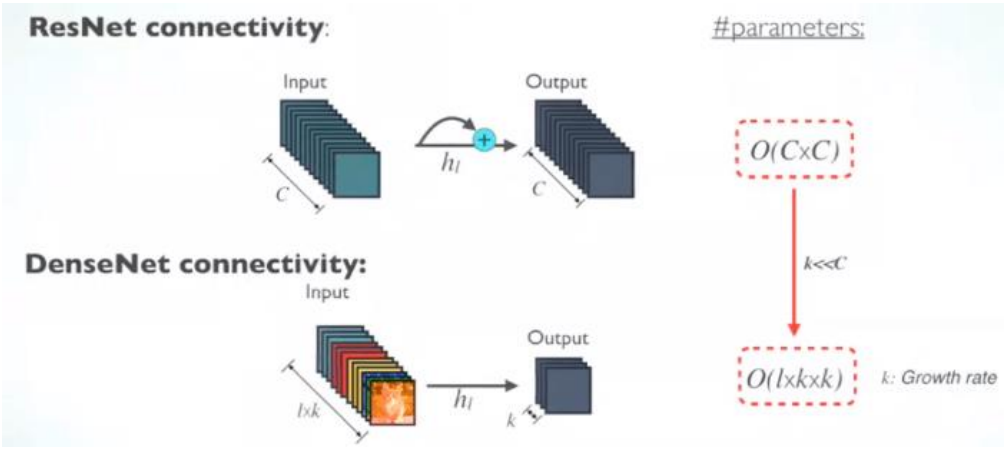
مزایای معماری DenseNet

-گردش گرادیان قوی: سیگنال خطا را می‌توان به راحتی و به صورت مستقیم‌تر به لایه‌های قبلی انتشار داد. این عمل یک نظارت عمیق و ضمنی به شمار می‌رود، زیرا لایه‌های قبلی می‌توانند تحت نظارت مستقیم لایه‌ی رده‌بندی نهایی قرار گیرند.

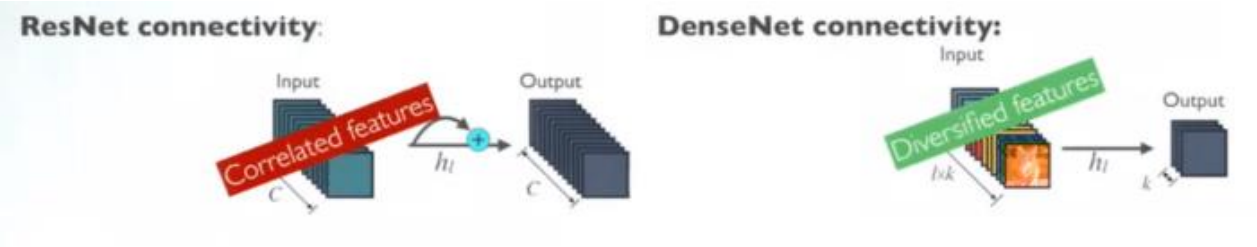


-کارآیی محاسباتی و پارامتری: تعداد پارامترهای هرکدام از لایه‌های ResNet به صورت مستقیم از نسبت CxC تأثیر می‌پذیرند؛ اما تعداد پارامترهای شبکه پیچشی متراکم تحت تأثیر مستقیم نسبت و مقدار lxkxk هستند.

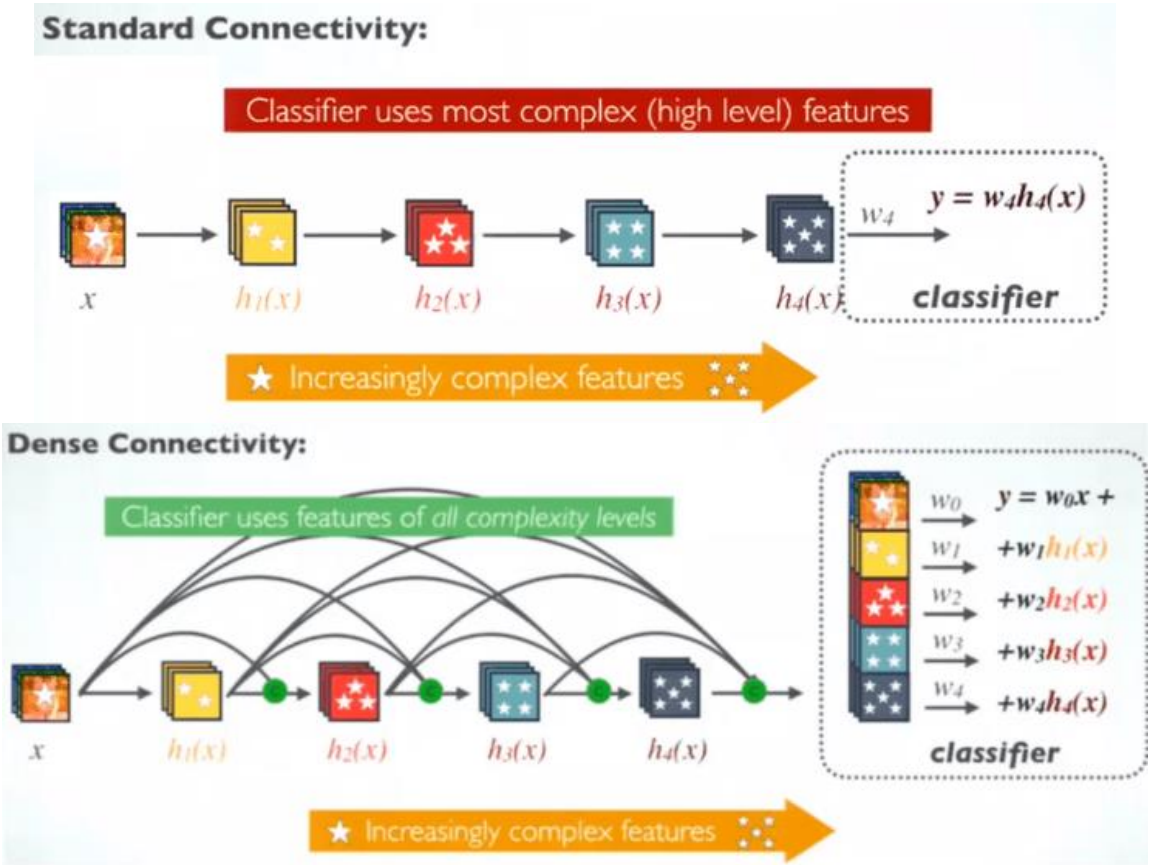
از آن‌جایی که $k<<C$، اندازه‌ی شبکه پیچشی متراکم بسیار کوچک‌تر از ResNet است.



-ویژگی‌های متفاوت‌تر (نامتجانس‌تر): از آن‌جایی که هر لایه‌ی شبکه پیچشی متراکم همه‌ی لایه‌های قبلی را به عنوان ورودی دریافت می‌کند، ویژگی‌های متفاوت‌تر و الگوهای غنی‌تری دارد.

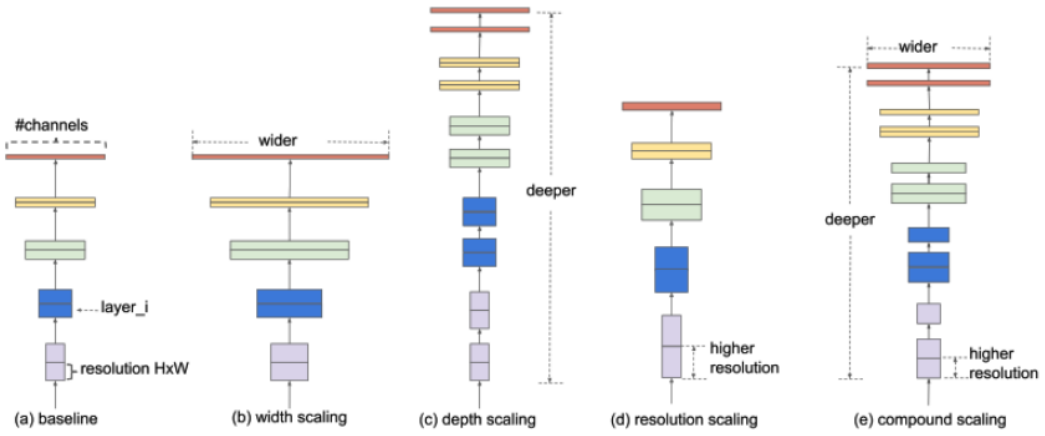


-نگهداری ویژگی‌هایی با پیچیدگی کمتر: در شبکه‌ی استانداردConvNet ، طبقه بند از پیچیده‌ترین ویژگی‌ها استفاده می‌کند. اما در شبکه پیچشی متراکم، ویژگی‌هایی که طبقه بند استفاده می‌کند سطوح متفاوتی از پیچیدگی دارند. در نتیجه این شبکه مرزهای تصمیم‌گیری تولید می‌شود که درجه اطمینان بالاتری دارند. این امر می‌تواند توجیه‌کننده‌ی این موضوع باشد که چرا شبکه پیچشی متراکم روی داده‌های آموزشی محدود، همچنان عملکرد خوبی از خود نشان می‌دهد[۳].



۴-۱ معماری EfficientNet

در معماری های طراحی شده در شبکه های کانولوشنی، سه روش برای افزایش دقت استفاده می‌شود.این سه روش شامل: افزایش عمق شبکه، ارتفاع شبکه و همچنین افزایش رزولوشن ورودی می‌باشد. که افزایش هر کدام از این ویژگی‌ها می‌تواند باعث بهبود عملکرد شبکه شود. در این معماری ایده ای در رابطه با طراحی شبکه جدید مطرح نشده است. بلکه با توجه به اینکه دستگاه های مختلف از توان پردازشی متفاوتی بهره مند هستند می‌خواهیم شیوه ای داشته باشیم که با توجه به دستگاه در دسترس و توانایی پردازش موجود چگونه یک شبکه را Scale کنیم. همچنین اگر از نظر زمانی، میزان زمان لازم برای آموزش شبکه را در نظر بگیریم با scale down شبکه زود تر و با scale up شبکه به مدت طولانی تری نیاز به آموزش دارد. به طور مثال اگر بخواهیم شبکه ی ما سریع‌تر آموزش ببیند و کمی کاهش دقت در نتایج شبکه مسئله ی خیلی مهمی نباشد میتوان از این روش استفاده نمود. بنابراین Efficient-net یک راهی برای به دست آوردن بهینه ترین میزان برای scale up کردن با توجه به شرایط موجود می باشد.



همان طور که در شکل اول می توان مشاهده نمود، با در نظر گرفتن یک baseline به صورت کلی سه روش برای scaling شبکه برای به دست آوردن دقت بهینه وجود دارد.

روش اول افزاش عمق: این روش بیشترین استفاده را در معماری های موجود تا کنون داشته است. منظور از افزایش عمق، افزایش تعداد لایه های یک شبکه می باشد.

روش دوم افزایش عرض: از این روش نسبت به عمق کمتر استفاده می‌شود. منظور از عرض نیز مقدار کانال های یک شبکه می باشد.

روش سوم افزایش resolution عکس ورودی: از این روش نیز گاها در مقالات مشاهده شده که استفاده شده است.[۴]

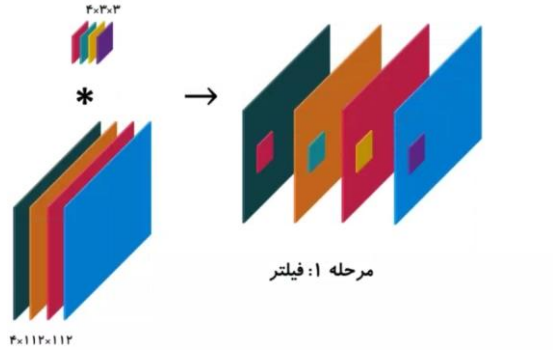
۵-۱ معماری MobileNet و Xception

شبکه عصبی mobilenet و xception از شاخص‌ترین شبکه‌های سبک هستند که از شبکه‌های کانولوشنی سبک با پارامترهای کم، سرعت اجرای بالا و دقت قابل‌قبول استفاده می کنند.

در هر دو شبکه برای کاهش تعداد پارامتر های شبکه کانولوشنال از دو عملیات کانولوشن به نام های کانولوشن نقطه ای(pointwise) و کانولوشن بر حسب کانال(depthwise) استفاده میشود که در حین کاهش قابل توجه پارامتر ها،دقت نیز در حد مطلوب باقی میماند.[۵]

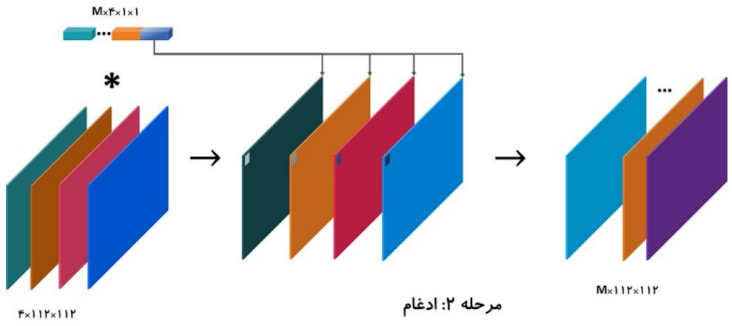
- کانولوشن depth-wise:

معادل همان فیلتر کردن در کانولوشن استاندارد است اما با یک تفاوت مهم، در کانولوشن استاندارد M کرنل k×k وجود دارد اما در کانولوشن عمقی تنها یک کرنل k×k استفاده می شود. به این مرحله کانولوشن عمقی گفته می‌شود. چون در راستای عمق یا صفحات، روی هر صفحه کانولوشن انجام داده‌ایم و صفحات خروجی را باهم جمع نکردیم.

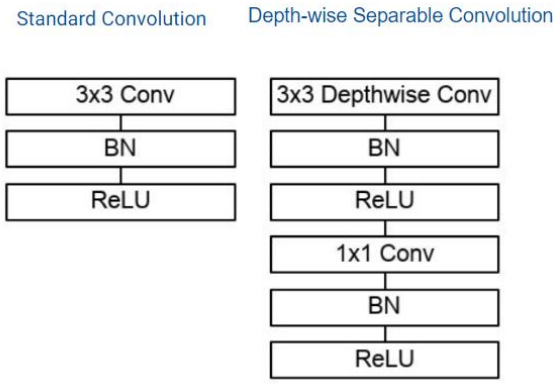


- کانولوشن point-wise:

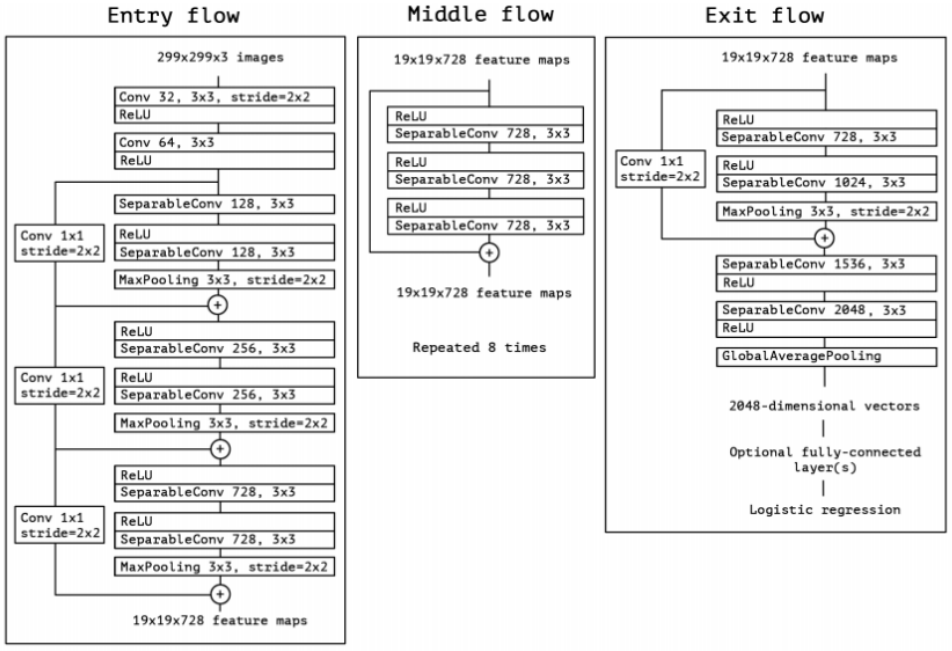
این مرحله معادل با مرحله ادغام در کانولوشن استاندارد است. اما یک تفاوت اساسی بین مرحله ادغام در کانولوشن استاندارد و کانولوشن dws وجود دارد، مرحله ادغام در کانولوشن استاندارد، یک جمع ساده است اما مرحله ادغام در کانولوشن عمقی شامل یک کانولوشن ۱×۱ است. کانولوشن ۱×۱ همچون یک نورون. با وزن‌دهی به صفحات مختلف آنها را باهم جمع می‌کند.



مقایسه کانولوشن استاندارد و کانولوشن depth-wise separable:



ساختار داخلی معماری xception به صورت زیر است:



ساختار داخلی معماری MobileNet به صورت زیر است:

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

فصل دوم، سطح مقدماتی

۲-۱ مجموعه داده

در اولین بخش هدف آموزش یک شبکه عصبی برای تشخیص میان گل های مختلف از دیتاست Petals to Metal است. تصویر گل های مختلف به عنوان ورودی شبکه و نوع گل به عنوان خروجی شبکه در نظر گرفته می شود که در ۱۰۴ کلاس تقسیم شده اند. چند نمونه ای از داده ها به صورت زیر است:

۲-۲ استفاده از TPU

TPU ها با بهره‌مندی از تجربه و پیشتازی گوگل در زمینه یادگیری ماشین طراحی شده و برای استفاده از نرم‌افزار متن‌باز TensorFlow توسعه داده شده‌اند. TPUها زمان یادگیری مدل‌های شبکه عصبی بزرگ و پیچیده را به حداقل می‌رساند. با استفاده ازTPU ، مدل‌های یادگیری عمیق (Deep Learning) که قبلا با پردازنده‌های گرافیکی (GPU) هفته‌ها طول می‌کشید، اکنون در TPU تنها ساعت‌ها زمان می‌برند.

ویژگی‌های واحد پردازش تانسور(TPU) عبارت اند از: سخت افزار ویژه برای پردازش ماتریسی، تأخیر بالا در مقایسه با CPU ،توان عملیاتی بسیار بالا، رایانش با حداکثر موازی‌سازی

در کد این قسمت پس از اضافه کردن و نصب پکیج ها و کتابخانه های مورد نیاز، تیکه کد زیر را جهت استفاده از TPU سایت kaggle اجرا می کنیم.

```
# Detect hardware, return appropriate distribution strategy
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()  # TPU detection. No parameters n
ecessary if TPU_NAME environment variable is set. On Kaggle this is always the case.
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    strategy = tf.distribute.get_strategy() # default distribution strategy in Tensorflow. Work
s on CPU and single GPU.

print("REPLICAS: ", strategy.num_replicas_in_sync)
```

۳-۲ پیش پردازش داده

در قسمت بعد پس از گرفتن path از دیتاست kaggle ، سایز عکس ها ، تعداد تکرار ها و bach size و تعداد عکس های هر دسته (آموزش، ارزیابی و تست) را مشخص می کنیم.

```
GCS_DS_PATH = KaggleDatasets().get_gcs_path()
```

```
IMAGE_SIZE = [331, 331] # at this size, a GPU will run out of memory. Use the TPU
EPOCHS = 25
BATCH_SIZE = 16 * strategy.num_replicas_in_sync

#NUM_TRAINING_IMAGES = 68094
NUM_VALIDATION_IMAGES = 3712
NUM_TEST_IMAGES = 7382
NUM_TRAINING_IMAGES = 12753
#NUM_TEST_IMAGES = 7382
STEPS_PER_EPOCH = NUM_TRAINING_IMAGES // BATCH_SIZE
TEST_STEPS = -(-NUM_TEST_IMAGES // BATCH_SIZE)
VALIDATION_STEPS = -(-NUM_VALIDATION_IMAGES // BATCH_SIZE)
```

تعداد داده های این دیتاست به نسبت زیاد است اما پس از بررسی مدل های مختلف و صحت آن ها دریافتیم که لازم است قبل از استفاده از داده ها عملیات های data augmentation را بر روی آن ها اجرا کنیم که شامل عملیات های (resize,crop,rotation,flip,transpose) می باشد. با این کار شبکه هر عکس را بهتر یاد گرفته و صحت شبکه بالاتر می رود.

```
def data_augment(image, label):
    p_spatial = tf.random.uniform([], 0, 1.0, dtype=tf.float32)
    p_rotate = tf.random.uniform([], 0, 1.0, dtype=tf.float32)

    image = tf.image.resize(image, [331+30, 331+30], method=tf.image.ResizeMethod.NEAREST_NEIG
HBOR)
    image = tf.image.random_crop(image, size=[331, 331, 3])

    if p_rotate > .8:
        image = tf.image.rot90(image, k=3)
    elif p_rotate > .6:
        image = tf.image.rot90(image, k=2)
    elif p_rotate > .4:
        image = tf.image.rot90(image, k=1)

    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)

    if p_spatial > .75:
        image = tf.image.transpose(image)

    return image, label
```

به عنوان نمونه برای یک تصویر خواهیم داشت:

در ادامه نیز با دستورات زیر محتوای عکس ها را decode کرده و داده های با لیبل و بدون لیبل را دریافت می کنیم تا داده هایی که اطلاعات کامل ندارند هم مشخص شده باشند.

```
def decode_image(image_data):
    image = tf.image.decode_jpeg(image_data, channels=3)
    image = tf.cast(image, tf.float32) / 255.0  # convert image to floats in [0, 1] range
    image = tf.reshape(image, [*IMAGE_SIZE, 3]) # explicit size needed for TPU
    return image

def read_labeled_tfrecord(example):
    LABELED_TFREC_FORMAT = {
        "image": tf.io.FixedLenFeature([], tf.string), # tf.string means bytestring
        "class": tf.io.FixedLenFeature([], tf.int64),  # shape [] means single element
    }
    example = tf.io.parse_single_example(example, LABELED_TFREC_FORMAT)
    image = decode_image(example['image'])
    label = tf.cast(example['class'], tf.int32)
    return image, label # returns a dataset of (image, label) pairs

def read_unlabeled_tfrecord(example):
    UNLABELED_TFREC_FORMAT = {
        "image": tf.io.FixedLenFeature([], tf.string), # tf.string means bytestring
        "id": tf.io.FixedLenFeature([], tf.string),  # shape [] means single element
        # class is missing, this competitions's challenge is to predict flower classes for the t
est dataset
    }
    example = tf.io.parse_single_example(example, UNLABELED_TFREC_FORMAT)
    image = decode_image(example['image'])
    idnum = example['id']
    return image, idnum # returns a dataset of image(s)
```

بعد از این کار با تابع زیر داده های با لیبل را بدون ترتیب مشخصی خوانده و خروجی آن به صورت tuple از عکس و کلاس آن خواهد بود.

```
def load_dataset(filenames, labeled=True, ordered=False):
    # Read from TFRecords. For optimal performance, reading from multiple files at once and
    # disregarding data order. Order does not matter since we will be shuffling the data anyway.

    ignore_order = tf.data.Options()
    if not ordered:
        ignore_order.experimental_deterministic = False # disable order, increase speed

    dataset = tf.data.TFRecordDataset(filenames) # automatically interleaves reads from multipl
e files
    dataset = dataset.with_options(ignore_order) # uses data as soon as it streams in, rather t
han in its original order
    dataset = dataset.map(read_labeled_tfrecord if labeled else read_unlabeled_tfrecord)
    # returns a dataset of (image, label) pairs if labeled=True or (image, id) pairs if labeled=
False
    return dataset
```

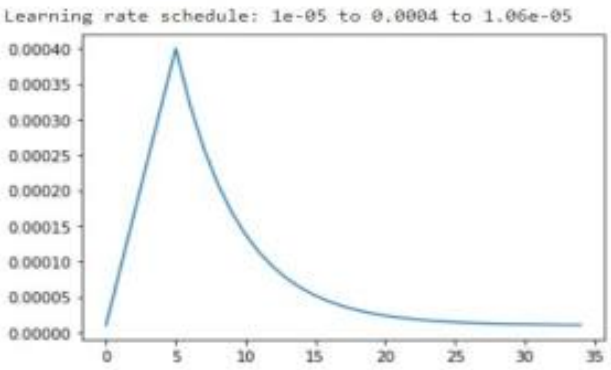
در آخر نیز با استفاده از توابع زیر دیتاست را به صورت سه دسته آموزش و ارزیابی و تست جدا کرده و همچنین یک تابع برای مجموع داده های اموزش و ارزیابی در نظر می گیریم زیرا هر دو دسته دارای لیبل هستند و در پایان انتخاب بهترین مدل می توان از هر دو برای آموزش بیشتر شبکه استفاده کرد.

```
def get_training_dataset():
    dataset = load_dataset(tf.io.gfile.glob(GCS_DS_PATH + '/tfrecords-jpeg-331x331/train/*.tfr
ec'), labeled=True)
    # Augmentation
    dataset = dataset.map(data_augment, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.repeat() # the training dataset must repeat for several epochs
    dataset = dataset.shuffle(2048)
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.prefetch(tf.data.AUTOTUNE)
    return dataset

def get_validation_dataset():
    dataset = load_dataset(tf.io.gfile.glob(GCS_DS_PATH + '/tfrecords-jpeg-331x331/val/*.tfre
c'), labeled=True, ordered=False)
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.prefetch(tf.data.AUTOTUNE)
    return dataset

def get_test_dataset(ordered=False):
    dataset = load_dataset(tf.io.gfile.glob(GCS_DS_PATH + '/tfrecords-jpeg-331x331/test/*.tfre
c'), labeled=False, ordered=ordered)
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.prefetch(tf.data.AUTOTUNE)
    return dataset
def get_all_training_dataset():
    dataset = load_dataset(tf.io.gfile.glob(GCS_DS_PATH + '/tfrecords-jpeg-331x331/train/*.tfr
ec') + tf.io.gfile.glob(GCS_DS_PATH + '/tfrecords-jpeg-331x331/val/*.tfrec')
, labeled=True)
    # Augmentation
    dataset = dataset.map(data_augment, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.repeat() # the training dataset must repeat for several epochs
    dataset = dataset.shuffle(2048)
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.prefetch(tf.data.AUTOTUNE)
    return dataset

training_dataset = get_training_dataset()
validation_dataset = get_validation_dataset()
all_training_dataset = get_all_training_dataset()
```

قبل از تعریف مدل ها برای آموزش بهتر شبکه و بالا رفتن صحت، مدل تغییرات ضریب یادگیری را به صورت کاهشی پیوسته تعریف می کنیم.

```
LR_START = 0.00001
LR_MAX = 0.00005 * strategy.num_replicas_in_sync
LR_MIN = 0.00001
LR_RAMPUP_EPOCHS = 5
LR_SUSTAIN_EPOCHS = 0
LR_EXP_DECAY = .8

def lrfn(epoch):
    if epoch < LR_RAMPUP_EPOCHS:
        lr = (LR_MAX - LR_START) / LR_RAMPUP_EPOCHS * epoch + LR_START
    elif epoch < LR_RAMPUP_EPOCHS + LR_SUSTAIN_EPOCHS:
        lr = LR_MAX
    else:
        lr = (LR_MAX - LR_MIN) * LR_EXP_DECAY**(epoch - LR_RAMPUP_EPOCHS - LR_SUSTAIN_EPOCHS)
+ LR_MIN
    return lr

lr_callback = tf.keras.callbacks.LearningRateScheduler(lrfn, verbose=1)

rng = [i for i in range(25 if EPOCHS<25 else EPOCHS)]
y = [lrfn(x) for x in rng]
plt.plot(rng, y)
print("Learning rate schedule: {:.3g} to {:.3g} to {:.3g}".format(y[0], max(y), y[-1]))
```

که نمودار آن به صورت زیر خواهد بود:



۴-۲ مدل Xception

در این قسمت با استفاده از کد زیر مدل xception را تعریف کرده که با دیتاست imagenet وزن دهی اولیه شده است و بعد با اضافه کردن پولینگ و batch normalization و drop out شبکه را بهبود می بخشیم که در نهایت پس از تعیین تابع خطا و metrics و optimizer مناسب مدل را compile کرده و بر روی دادگان آموزش با ضریب یادگیری پیوسته کاهشی، به تعداد تکرار های مورد نظر fit می کنیم.

Xception model

```
with strategy.scope():
    base_model = xception.Xception(
        weights='imagenet',
        input_shape=(*IMAGE_SIZE, 3),
        include_top=False,
        pooling=None
    )
    base_model.trainable = False

    model = Sequential()
    model.add(base_model)
    model.add(GlobalAveragePooling2D())
    model.add(BatchNormalization())

    model.add(Dense(512, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Dense(104, activation='softmax'))
```

```
model.compile(
    optimizer=RMSprop(learning_rate=0.0005),
    loss='sparse_categorical_crossentropy',
    metrics=['sparse_categorical_accuracy']
)
```

```
history = model.fit(training_dataset,
          steps_per_epoch=STEPS_PER_EPOCH,
          epochs=EPOCHS,
          callbacks = [lr_callback],
          validation_steps=VALIDATION_STEPS,
          validation_data=validation_dataset)
```

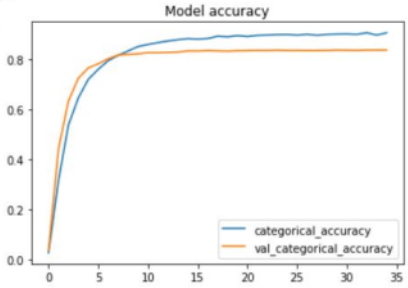
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.title('Model loss')
plt.show()
```

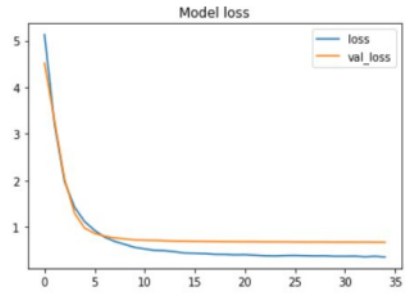
```
plt.plot(history.history['sparse_categorical_accuracy'])
plt.plot(history.history['val_sparse_categorical_accuracy'])
plt.legend(['sparse_categorical_accuracy', 'val_sparse_categorical_accuracy'])
plt.title('Model accuracy')
plt.show()
```

در نهایت نتایج به صورت زیر خواهد شد.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
xception (Functional)        (None, 11, 11, 2048)      20861480
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
batch_normalization_4 (Batch (None, 2048)              8192
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 104)               53352
=================================================================
Total params: 21,974,160
Trainable params: 1,107,560
Non-trainable params: 20,866,600
_________________________________________________________________
```

مدل Xception





0.8357
Best validation accuracy

0.6770
Best validation loss

0.9061
Best train accuracy

0.3525
Best train loss

۵-۲ مدل DenseNet

همان روند قسمت قبل را به طور مشابه برای این مدل اجرا می کنیم

DenseNet model

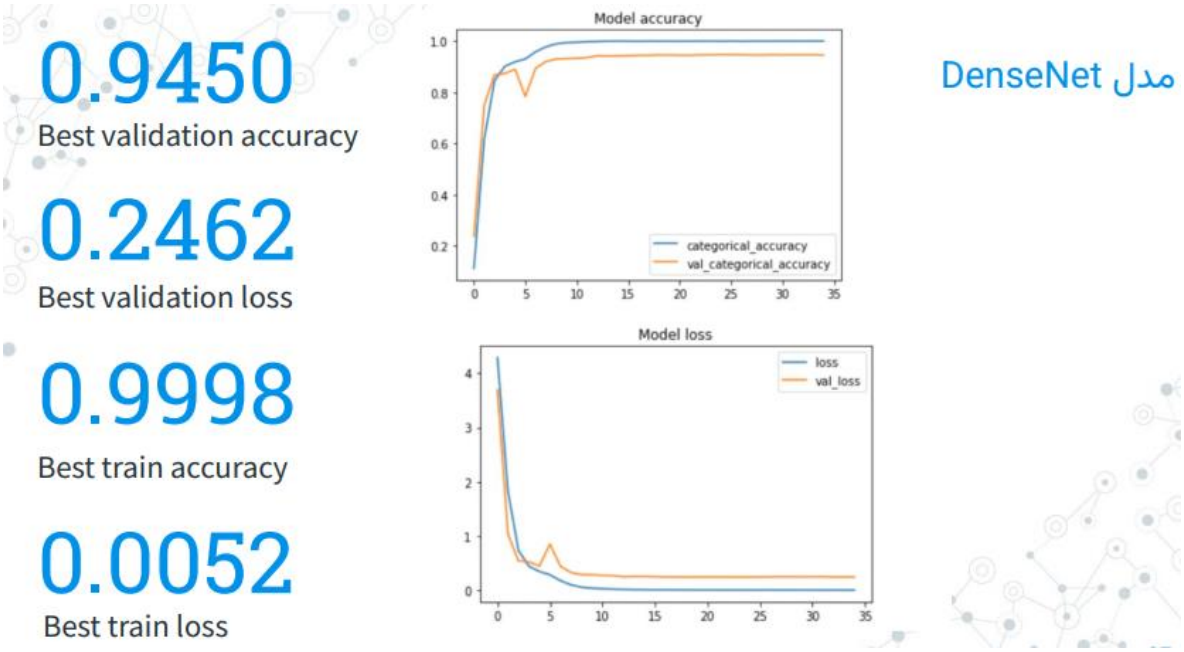
```
def DenseNet():
    with strategy.scope():
        rnet = DenseNet201(
            input_shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3),
            weights='imagenet',
            include_top=False
        )
        # trainable rnet
        rnet.trainable = True
        model = tf.keras.Sequential([
            rnet,
            tf.keras.layers.GlobalAveragePooling2D(),
            tf.keras.layers.Dense(104, activation='softmax',dtype='float32')
        ])
    model.compile(
    optimizer='adam',
    loss = 'sparse_categorical_crossentropy',
    metrics=['sparse_categorical_accuracy']
    )
    model.summary()
    return model
```

```
model.compile(
    optimizer=RMSprop(learning_rate=0.0005),
    loss='sparse_categorical_crossentropy',
    metrics=['sparse_categorical_accuracy']
)
```

```
model = DenseNet();
history = model.fit(training_dataset,
          steps_per_epoch=STEPS_PER_EPOCH,
          epochs=EPOCHS,
          callbacks = [lr_callback],
          validation_steps=VALIDATION_STEPS,
          validation_data=validation_dataset)
```

در نهایت نتایج به صورت زیر خواهد شد:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet201 (Functional)     (None, 10, 10, 1920)      18321984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1920)              0
_________________________________________________________________
dense (Dense)                (None, 104)               199784
=================================================================
Total params: 18,521,768
Trainable params: 18,292,712
Non-trainable params: 229,056
_________________________________________________________________
```



۲-۶ مدل EfficientNet

به طور مشابه مدل را به صورت زیر اجرا می کنیم:

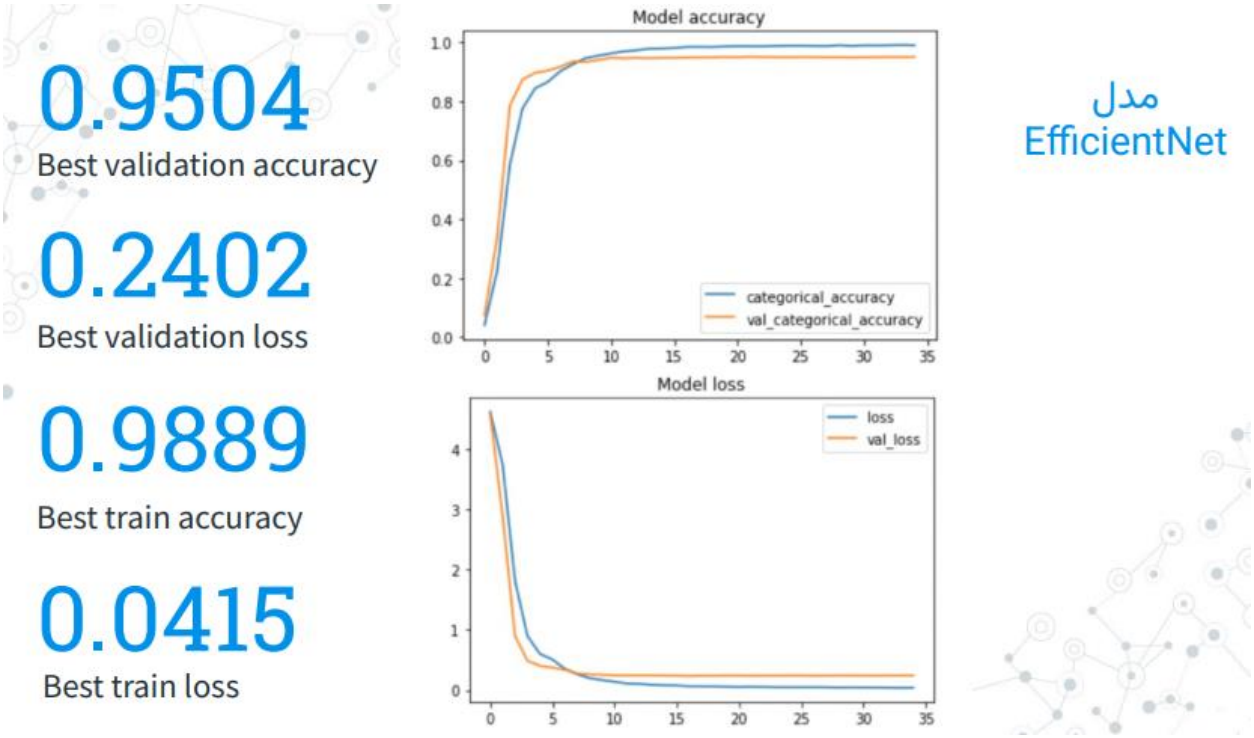
Efficient model

```
def EfficientNetb7():
    with strategy.scope():
        enet = efn.EfficientNetB7(weights='noisy-student',
                                  include_top=False,
                                  pooling='avg',
                                  input_shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3))
        enet.trainable = True
        model2 = tf.keras.Sequential([
            enet,
            tf.keras.layers.Dense(104, activation='softmax',dtype='float32')
        ])
    model2.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['sparse_categorical_accuracy']
    )
    model2.summary()
    return model2
```

```
model2 = EfficientNetb7()
history2 = model2.fit(training_dataset,
          steps_per_epoch=STEPS_PER_EPOCH,
          epochs=EPOCHS,
          callbacks = [lr_callback],
          validation_data=validation_dataset)
```

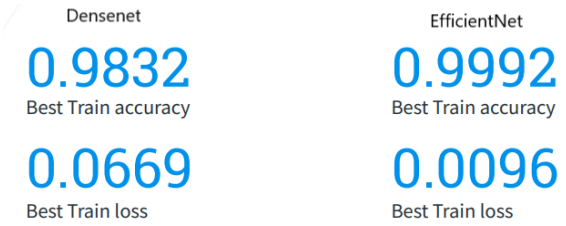
در نهایت نتایج به صورت زیر خواهد شد:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnet-b7 (Functional) (None, 2560)              64097680
_________________________________________________________________
dense_1 (Dense)              (None, 104)               266344
=================================================================
Total params: 64,364,024
Trainable params: 64,053,304
Non-trainable params: 310,720
_________________________________________________________________
```



۷-۲ تست بهترین مدل

در آخر از آنجایی که هر دو مدل densenet و efficientnet صحت بالا و عملکرد خوبی داشتند، دو مدل را بر روی مجموع دادگان آموزش و ارزیابی اجرا کرده که به نتایج زیر خواهیم رسید



و در نهایت هر دو مدل به صورت زیر با هم ترکیب کرده و میانگین گیری می کنیم تا به بهترین نتیجه ممکن برسیم. در آخر نیز مدل نهایی را بر روی دادگان تست اجرا کرده و در سایت kaggle سابمیت می کنیم.

Test predict

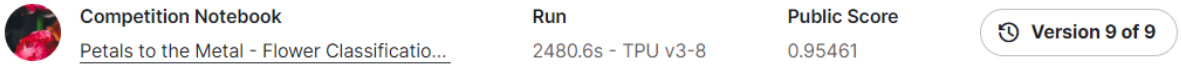
```
model3 = EfficientNetb7()
history3 = model3.fit(all_training_dataset,
          steps_per_epoch=STEPS_PER_EPOCH,
          epochs=EPOCHS,
          callbacks = [lr_callback])
```

```
model4 = DenseNet();
history4 = model4.fit(all_training_dataset,
          steps_per_epoch=STEPS_PER_EPOCH,
          epochs=EPOCHS,
          callbacks = [lr_callback])
```

```
def run_inference(model):
    test_ds = get_test_dataset(ordered=True) # since we are splitting the dataset and iterating separately on images and ids, orc
    test_images_ds = test_ds.map(lambda image, idnum: image)
    preds = model.predict(test_images_ds,verbose=0, steps=TEST_STEPS)
    return preds
```

```
test_ds = get_test_dataset(ordered=True) # since we are splitting the dataset and iterating separately on images and ids, order r

print('Calculating predictions...')
probs1 = run_inference(model3)
probs2 = run_inference(model4)
probabilities = (probs1 + probs2)/2
predictions = np.argmax(probabilities, axis=-1)

print('Generating submission file...')
test_ids_ds = test_ds.map(lambda image, idnum: idnum).unbatch()
test_ids = next(iter(test_ids_ds.batch(NUM_TEST_IMAGES))).numpy().astype('U') # all in one batch
np.savetxt('submission.csv', np.rec.fromarrays([test_ids, predictions]), fmt=['%s', '%d'], delimiter=',', header='id,label', comm
```

مقدار صحت مدل بر روی دادگان تست در نهایت ۰,۹۵۴۶۱ می شود که نتیجه بسیار مطلوبی است.



فصل سوم، سطح متوسط

۳-۱ دیتاست ورودی

در بخش ۲ از ما خواسته شده که روی دیتاستی که شامل ۱۴ سلبریتی می‌باشد classification انجام دهیم. نمونه ای از تصاویر دیتاست به صورت زیر است. در این دیتاست تعداد داده‌های آموزش ما ۲۲۰ عدد و تعداد داده‌های validation ۷۰ تا می‌باشد.

تمام مراحل این بخش نیز مانند قسمت اول، در کگل انجام شده است. پس از اضافه کردن کتابخانه های موردنیاز، دیتاستی که در همین سایت وجود دارد را به notebook اضافه می‌کنیم. سپس آن را از حالت zip خارج می‌نماییم تا بتوانیم از پوشه‌ی train و valid به راحتی استفاده کنیم.

```
import cv2
import requests
import zipfile
import os
import keras
import numpy as np
from PIL import Image
from io import BytesIO
import tensorflow as tf
import matplotlib.pyplot as plt
from keras import backend as K
from keras.optimizers import RMSprop
from keras.preprocessing import image
from keras.models import Model, Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization, GlobalAveragePooling2D
```

```
base_dir = "../input/14-celebrity-faces-dataset"
celeb14 = os.path.join(base_dir, "14-celebrity-faces-dataset.zip")

with zipfile.ZipFile(celeb14, "r") as z:
    z.extractall('.')
```

```
data_dir = '/kaggle/working/14-celebrity-faces-dataset/data'
resnet50weight = '../input/keras-pretrained-models/resnet50_weights_tf_dim_ordering_tf_kernels
_notop.h5'
```

سپس باید مشخصات مدل را تا حدودی تعیین نماییم. سایز عکس در هنگام آموزش مؤثر است. در ابتدا سایز را ۲۲۴ x ۲۲۴ در نظر گرفتیم ولی با سعی و خطا دریافتیم که با افزایش آن به مقدار ۲۵۶ x ۲۵۶ می‌توان نتیجه‌ی بهتری گرفت. البته توجه داریم که برای این شبکه‌ها نمی‌توان از هر سایز دلخواهی استفاده نمود، به طور مثال نباید سایز از ۳۲ کوچکتر باشد.

سپس مسیر دایرکتوری train و valid را هم به صورت جداگانه مشخص می‌کنیم. در این دیتاست تعداد داده‌های آموزش ما ۲۲۰ عدد و تعداد داده‌های validation ۷۰ تا می‌باشد. تعداد بهینه برای epoch ها نیز عدد ۴۵ در نظر گرفته شده چرا که با افزایش آن شانس اورفیت شدن افزایش می‌یابد و با کاهش آن (مثلاً به مقدار ۳۰) در نمودارها مشاهده می‌کنیم که در واقع زمان کافی به شبکه ندادیم که آموزش لازم را ببینند. در خصوص batch size نکته قابل ذکر این است که با افزایش آن، فرضاً به مقدار ۳۲ یا ۶۴ صحت خروجی کاهش پیدا می‌کند و این شبکه با این دیتاست احتیاج دارد که روی تعداد کمتری از سمپل‌ها در هر ایپاک عملیات را انجام دهد. تعداد کلاس‌ها هم مستقیماً مربوط به خود دیتاست هست که چون ۱۴ نفر داریم، پس کلاس‌های خروجی نیز ۱۴ عدد هستند.

Model Parameters

```
img_width, img_height = 224, 224

train_data_dir = os.path.join(data_dir, 'train')
validation_data_dir = os.path.join(data_dir, 'val')
nb_train_samples = 220
nb_validation_samples = 70
epochs = 35
batch_size = 16
numclasses = 14
```

۳-۲ پیش پردازش داده ها

در مرحله‌ی بعدی به augmentation عکس‌های train می‌پردازیم. برای این کار از ImageDataGenerator که در کراس وجود دارد استفاده می‌کنیم و برخی ویژگی‌های تصویر را تغییر می‌دهیم. به عنوان مثال حتماً scaling باید انجام شود که باعث نرمال شدن وکتورها خواهد شد؛ میزان شیفت و چرخش تصویر هم تا حدودی تغییر پیدا کرده اند. برای فلیپ کردن باید توجه داشت که فقط به صورت افقی انجام شود و نه به شکل عمودی، چرا که سمپل‌ها در واقع تصاویر اشخاص هستند و با فلیپ عمودی به طور کلی شکل انسان تغییر خواهد کرد و اصلاً مطلوب نیست. برای داده‌های تست نیز فقط scale انجام می‌دهیم. در قدم بعدی با متد flow_from_directory، عکس‌ها را مستقیماً از همان دایرکتوری‌ای که قبلاً آدرس‌دهی کرده بودیم می‌خوانیم و augment را اعمال می‌کنیم. برای این کار به عنوان آرگومان، آدرس، batch size، سایز عکس و نوع کلاس-بندی را به این متد می‌دهیم. سپس مشخص می‌کنیم که ترتیب قرارگیری ورودی ما به تنسور چگونه باشد، یعنی اول تعداد کانال‌ها قرار بگیرد یا سایز عکس. در این جا مشخص کرده ایم که اگر فرمت عکس به شکل channels_first باشد ما نیز ابتدا همان سه کانال RGB را می‌دهیم و در غیر این صورت اگر فرمت عکس ما به شکل برعکس بود باید ابتدا سایز عکس مقداردهی شود و سپس تعداد کانال‌ها مشخص گردد.

Processing images

```
# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
    rescale=1./ 255,
    rotation_range=10,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    vertical_flip=False,
    horizontal_flip=True)


test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')
```

```
Found 220 images belonging to 14 classes.
Found 70 images belonging to 14 classes.
```

```
if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)
```

۳-۳ مدل DenseNet

در این بخش به قسمت اصلی شبکه می‌رسیم. قبل از هرچیز نکته قابل ذکر این است که در ابتدا ما از densenet۱۲۱ استفاده کردیم و چون بهبود زیادی در عملکرد مدل شاهد نبودیم، آن را به densenet۲۰۱ ارتقاء دادیم و خروجی حدوداً ۱۰٪ بهتر شد. برای ساخت شبکه لازم است در گام اول وزن‌دهی مربوط به densenet انجام گیرد، که ما این وزن‌ها را بر اساس imagenet تنظیم کردیم، البته وزن‌های خود densenet هم مورد تست قرار گرفتند ولی مشخص شد که با این دیتاست، imagenet انتخاب مناسب‌تری ست. توجه داریم که لایه آخر را حذف می‌کنیم تا خودمان بتوانیم آن را طراحی کنیم.

برای این کار یک مدل sequential ایجاد می‌کنیم و اولین قدم این است که خود densenet را اضافه نماییم. سپس برای اتصال لایه‌های قبلی به لایه‌ی بعدی از GlobalAveragePooling۲D استفاده می‌کنیم که average pooling را روی ابعاد اعمال می‌کند تا زمانی که هریک از ابعاد فضایی به یک برسند و آن را ۀآبقیه را هم دست نخورده باقی می‌گذارد. مثلاً اگر در حالت channels last باشیم (یعنی ابعاد دوم و سوم ما فضایی باشند)، تنسوری که در ابتدا به شکل (samples, ۱۰, ۲۰, ۱) بوده به صورت (samples, ۱, ۱, ۱) درمی‌آید. بعد از آن لایه‌ی dense را که تابع فعال‌ساز ReLu دارد قرار می‌گیرد، تعداد نورون‌های این لایه یکی دیگر از چالش‌ها بود. برای این مرحله مقادیر از ۱۲۸ تا ۸۱۹۲ امتحان شدند که بهترین آن‌ها ۴۰۹۶ بود. همچنین اعمال dropout برای این لایه تأثیر به‌سزایی دارد. برای dropout هم مقدارهای ۰,۵ ,۰,۴ ,۰,۳ ,۰,۲ مورد آزمایش قرار گرفتند که ۰,۲ نسبت به همه برتری داشت. در آخرین گام نیز برای کلاس‌بندی و گرفتن خروجی از شبکه لازم است که یک لایه dense با تعداد کلاس‌های همین دیتاست (یعنی ۱۴) قرار بدهیم و طبیعتاً از تابع فعال‌ساز softmax یا sigmoid استفاده می‌گردد چرا که رنج خروجی آن به صورت محدود و بین ۰ و ۱ است و همچنین جمع تمام مقادیر آن هم همیشه ۱ می‌باشد.

بعد از این نیاز داریم که مدل را کامپایل کنیم. برای معیار سنجش و همچنین loss می‌توان از پیش‌فرض‌های دیگری مانند accuracy و یا RMSprop هم استفاده نمود ولی طی تست‌هایی که انجام شد دریافتیم که این شبکه و دیتاست با categorical_crossentropy نتیجه‌ی مطلوب‌تری خواهند داد.

```
"""densenet = tf.keras.applications.DenseNet121(
    weights='/kaggle/input/densenet-keras/DenseNet-BC-121-32-no-top.h5',
    include_top=False,
    input_shape=input_shape)"""
densenet = tf.keras.applications.DenseNet201(
    weights='imagenet',
    include_top=False,
    input_shape=input_shape)
densenet.trainable = True
def build_model():
    model = Sequential()
    model.add(densenet)
    model.add(GlobalAveragePooling2D())
    model.add(Dropout(0.5))
    model.add(Dense(512, activation='relu')) #sigmoid
    model.add(Dropout(0.3))
    model.add(Dense(numclasses, activation='softmax'))

    model.compile(
        loss='categorical_crossentropy',
        optimizer=keras.optimizers.adam_v2.Adam(lr=1e-4), #1e-4
        metrics=['categorical_accuracy'])

    return model

model = build_model()
```

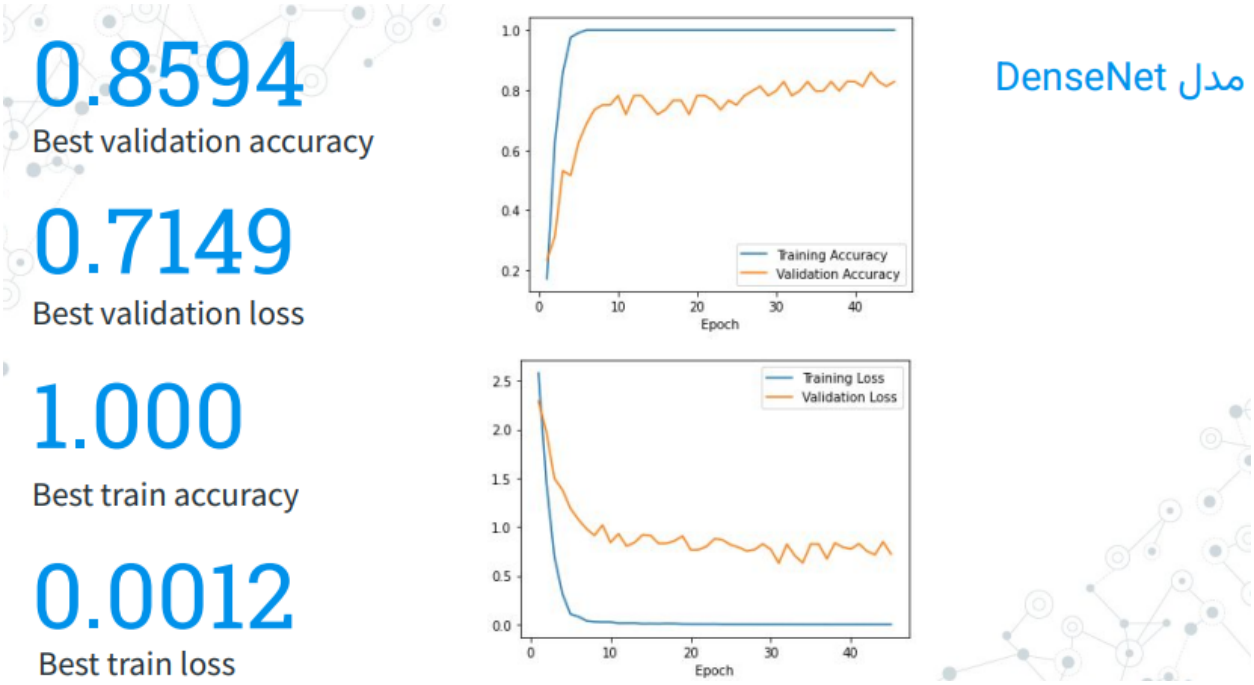
بعد از ساخت مدل نوبت به فیت کردن آن روی دیتاها می‌رسد. برای این کار از fit_generator استفاده می‌کنیم و داده‌های آموزش، گام‌ها در هر ایپاک، تعداد ایپاک‌ها و مجموعه‌ی valid را اختصاص می‌دهیم.

```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet201 (Functional)     (None, 8, 8, 1920)        18321984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1920)              0
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              7868416
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_3 (Dense)              (None, 14)                57358
=================================================================
Total params: 26,247,758
Trainable params: 26,018,702
Non-trainable params: 229,056
_________________________________________________________________
```

بعد از آن با رسم نمودار accuracy و loss برای هر دو دیتای train و validation نتایج را مشاهده می‌نماییم. در این قسمت برای ما مهم بود که شیب accuracy برای دیتای valid همانند داده‌ی train به شکل صعودی باشد، چرا که اگر نزولی باشد نشان‌دهنده‌ی این است که overfit در حال انجام است و در آن صورت نتایج معتبر نخواهند بود.

مدل DenseNet



پس از آن مدل را به همراه وزن‌های آن ذخیره می‌کنیم تا بتوانیم حجم آن‌ها را هم مشاهده کنیم. برای densenet۲۰۱ حجمی در حدود ۳۱۵,۸ را شاهد خواهیم بود که حجم نسبتاً زیادی‌ست به خصوص اگر بخواهیم آن را روی device های دیگری پیاده‌سازی نماییم که به این اندازه ظرفیت پردازش ندارند. در مرحله‌ی آخر نیز برای این که بتوانیم یک تست جداگانه از شبکه داشته باشیم، تمام عکس‌های valid را به عنوان تست به شبکه دادیم و تعداد دفعاتی که درست شخص را پیش‌بینی کرده بود محاسبه نمودیم که باز هم شاهد این هستیم که این مدل صحت مطلوبی دارد.

```
saveweight = 'celebriytag_weight.h5'
model.save_weights(saveweight)
```

```
labels = ['anne_hathaway', 'arnold_schwarzenegger', 'ben_afflek', 'dwayne_johnson', 'elton_john', 'jerry_seinfeld', 'kate_beckinsal
e', 'keanu_reeves', 'lauren_cohan', 'madonna', 'mindy_kaling', 'simon_pegg', 'sofia_vergara', 'will_smith']
test_imgs = []
for celebrity in labels:
    tmp = (os.listdir(f'/kaggle/working/14-celebrity-faces-dataset/data/val/{celebrity}'))
    tmp2 = []
    for s in tmp:
        tmp2.append(f'{celebrity}/'+ s)

    test_imgs.extend(tmp2)
```

```
err_cnt = 0
count = 0
for test in test_imgs:
    test_img = os.path.join(validation_data_dir, test)
    img = image.load_img(test_img, target_size=(img_width, img_height))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x /= 255.
    classes = model.predict(x)
    result = np.squeeze(classes)
    result_indices = np.argmax(result)
    count+=1
    if labels[result_indices] != test.split("/")[0]:
        err_cnt += 1
#     print(labels[result_indices], test.split("/")[0])
#     print('------------------------')
print(err_cnt, count)
print(f'Accuracy: {(count-err_cnt)/count}')
```

۴-۳ مدل ResNet

برای رزنت ۵۰ هم که در ابتدا امتحان کرده بودیم، دقیقا همین روند طی شده است و فقط در بخش شبکه، وزن‌های خود resnet۵۰ را اعمال نمودیم و یک لایه‌ی dense با تعداد نورون ۲۰۴۸ و dropout = ۰,۵ و سپس یک لایه dense دیگر با نورون‌های کمتر (۱۰۲۴) با همان dropout اعمال کردیم. همچنین برای metric هم از accuracy استفاده شده است. خروجی این شبکه دقتی در حدود ٪۷۶ به ما می‌دهد.

```
def resnet50tl(input_shape, outclass, sigma='sigmoid'):
    base_model = None
    base_model = keras.applications.resnet50.ResNet50(weights=None, include_top=False, input_s
hape=input_shape)
    base_model.load_weights(resnet50weight)

    top_model = Sequential()
#     top_model.add(Flatten(input_shape=base_model.output_shape[1:]))
    top_model.add(GlobalAveragePooling2D(input_shape=base_model.output_shape[1:]))
    top_model.add(Dense(2048, activation='relu'))
    top_model.add(Dropout(0.5))

    top_model.add(Dense(1024, activation='relu'))
    top_model.add(Dropout(0.5))

    top_model.add(Dense(outclass, activation=sigma))

    model = None
    model = Model(inputs=base_model.input, outputs=top_model(base_model.output))

    return model


model = resnet50tl(input_shape, numclasses, 'softmax')
opt = keras.optimizers.Adam(lr=3e-5, decay=1e-7)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

بخشی از summary مدل:

Layer (type)	Output Shape	Param #	Connected to
input_4 (InputLayer)	(None, 224, 224, 3)	0	
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_4[0][0]
conv1 (Conv2D)	(None, 112, 112, 64)	9472	conv1_pad[0][0]
bn_conv1 (BatchNormalization)	(None, 112, 112, 64)	256	conv1[0][0]
activation_148 (Activation)	(None, 112, 112, 64)	0	bn_conv1[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 55, 55, 64)	0	activation_148[0][0]
res2a_branch2a (Conv2D)	(None, 55, 55, 64)	4160	max_pooling2d_4[0][0]
bn2a_branch2a (BatchNormalizati	(None, 55, 55, 64)	256	res2a_branch2a[0][0]
activation_149 (Activation)	(None, 55, 55, 64)	0	bn2a_branch2a[0][0]
res2a_branch2b (Conv2D)	(None, 55, 55, 64)	36928	activation_149[0][0]
bn2a_branch2b (BatchNormalizati	(None, 55, 55, 64)	256	res2a_branch2b[0][0]
activation_150 (Activation)	(None, 55, 55, 64)	0	bn2a_branch2b[0][0]
res2a_branch2c (Conv2D)	(None, 55, 55, 256)	16640	activation_150[0][0]
res2a_branch1 (Conv2D)	(None, 55, 55, 256)	16640	max_pooling2d_4[0][0]
bn2a_branch2c (BatchNormalizati	(None, 55, 55, 256)	1024	res2a_branch2c[0][0]
bn2a_branch1 (BatchNormalizatio	(None, 55, 55, 256)	1024	res2a_branch1[0][0]
add_49 (Add)	(None, 55, 55, 256)	0	bn2a_branch2c[0][0] bn2a_branch1[0][0]
activation_151 (Activation)	(None, 55, 55, 256)	0	add_49[0][0]
res2b_branch2a (Conv2D)	(None, 55, 55, 64)	16448	activation_151[0][0]
bn2b_branch2a (BatchNormalizati	(None, 55, 55, 64)	256	res2b_branch2a[0][0]
activation_152 (Activation)	(None, 55, 55, 64)	0	bn2b_branch2a[0][0]
res2b_branch2b (Conv2D)	(None, 55, 55, 64)	36928	activation_152[0][0]
bn2b_branch2b (BatchNormalizati	(None, 55, 55, 64)	256	res2b_branch2b[0][0]
activation_153 (Activation)	(None, 55, 55, 64)	0	bn2b_branch2b[0][0]
res2b_branch2c (Conv2D)	(None, 55, 55, 256)	16640	activation_153[0][0]
bn2b_branch2c (BatchNormalizati	(None, 55, 55, 256)	1024	res2b_branch2c[0][0]
add_50 (Add)	(None, 55, 55, 256)	0	bn2b_branch2c[0][0] activation_151[0][0]
sequential_4 (Sequential)	(None, 14)	6300878	avg_pool[0][0]

Total params: 29,896,590
Trainable params: 29,843,470
Non-trainable params: 53,120

نتایج به صورت زیر خواهد بود:

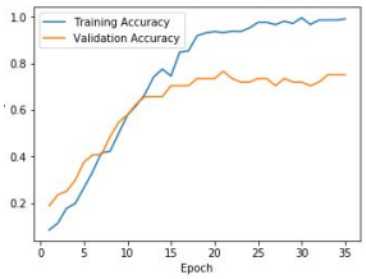
مدل ResNet

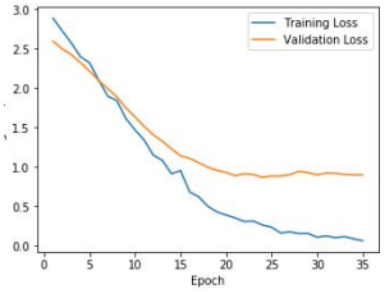
0.7656
Best validation accuracy

0.8883
Best validation loss

0.9904
Best train accuracy

0.0611
Best train loss





فصل چهارم، سطح پیشرفته

۴-۱ داده ورودی و پیش پردازش

در ابتدا کتابخانه های مور نیاز را اضافه می کنیم و در قسمت بعد فایل دیتاست را می خوانیم.

```
import requests
import zipfile
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras

from keras import models, layers, optimizers, callbacks, losses
from keras.preprocessing.image import ImageDataGenerator
```

```
base_dir = "../input/14-celebrity-faces-dataset"
celeb14 = os.path.join(base_dir, "14-celebrity-faces-dataset.zip")

with zipfile.ZipFile(celeb14, "r") as z:
    z.extractall('.')
```

در این بخش با توجه به اندازه ی عکس های دیتاسیت مناسب ترین اندازه را انتخاب میکنم همچنین باید مد نظر داشته باشیم که بهتر است سایز عکس ها مضاربی از ۲ باشد. سپس مجموعه های آموزش و تست را مشخص میکنیم.

در این دیتاست تعداد کل عکس های مربوط به train برابر است با ۲۲۰ و تعداد کل عکس های مجموعه ی validation برابر است با ۷۰ عکس , که به نسبت با دیتاست کوچکی مواجه هستیم.

تعداد اپوک ها را برابر با ۵۰ در نظر میگیریم که این عدد با استفاده از early topping بدست آمده است .

مقدار batch size را برابر با ۱۶ در نظر میگیریم .(با توجه به حجم دیتاست این عدد انتخاب شده است.) و در نهایت نعداد کلاس ها بربر است با تعداد افراد مشهور که باید شناسایی شوند که برابر است با ۱۴.

Model Parameters

```
img_width, img_height = 224, 224

train_data_dir = os.path.join(data_dir, 'train')
validation_data_dir = os.path.join(data_dir, 'val')
nb_train_samples = 220
nb_validation_samples = 70
epochs = 50
batch_size = 16
numclasses = 14
```

در بخش پیش پردازش ابتدا رنج هر پیکسل را از ۰ تا ۲۵۵ به ۰ تا ۱ اسکیل می کنیم این کار برای این انجام میشود که شبکه بتواند بهتر یادگیری را انجام بدهد. با توجه به حجم کم دیتاست باید از data augmentation استفاده کنیم به این جهت از دستورات زیر استفاده می کنیم:

Processing images

```
# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
    rescale=1./ 255,
    rotation_range=10,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    vertical_flip=False,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')
```

```
Found 220 images belonging to 14 classes.
Found 70 images belonging to 14 classes.
```

```
input_shape = (img_width, img_height, 3)
```

۲-۴ مدل MobileNet

در بخش بعدی به تعریف خود مدل میپردازیم. ما در این بخش از پروژه مدل را به صورت functional تعریف میکنیم. تا بتوانیم عکس هایی که از لایه های مختلف کانولوشن بدست می آیند را هم در اخر مشاهده کنیم. همچین در این بخش هم از ترنسفر لرنینگ استفاده میکنیم از طرفی چون هدف از انحام این بخش کم کردن حجم مدل است باید از شبکه ها سبکتری مثل mobilenet استفاده کنیم. همچنین وزن های اولیه ی شبکه مربوط به شبکه ی imagenet میباشد . با استفاده از ترنسفر لرنینگ ما فقط باید لایه ی نورونی آخر را با توجه به نعداد کلاس های مد نظر بازسازی کنیم. در لایه ی آخر از تابع فعال ساز softmax استفاده شده است و از اپتیمایز RMSprop استفاده کردیم که این اپتیمایز همان gradiend descent به همراه momentunمیباشد.

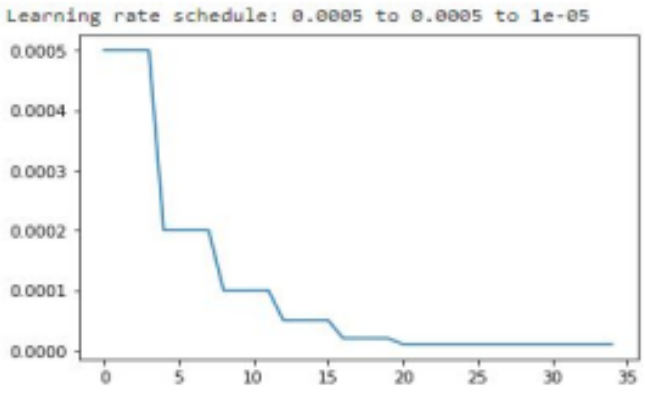
```
def mobile_net():
    model_input = layers.Input(input_shape)
    base_model = keras.applications.mobilenet.MobileNet(weights='imagenet',include_top=False,input_shape=input_shape,pooling=Fals
    x = base_model(model_input)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Flatten()(x)
    x = layers.Dropout(0.6)(x)
    model_output = layers.Dense(numclasses, activation='softmax')(x)

    model = models.Model(inputs=model_input, outputs=model_output)

    return model


model = mobile_net()
model.compile(loss=losses.CategoricalCrossentropy(),
              optimizer=optimizers.rmsprop_v2.RMSprop(learning_rate=0.0005),
              metrics=['accuracy'])
```

```
history = model.fit(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size,
    callbacks=[lr_callback])
```

همچنین در این بخش از پروژه به جهت بالاتر بردن صحت و دقت مدل ما از ضریب یادگیری متغیر استفاده کردیم. در واقع با توجه به تجربه ای که از بخش اول داشتیم و میدانستیم که با استفاده از ضریب یادگیری متغیر میتوانیم به صحت بالاتر برسیم پس در ابتدا ضریب یادگیری را مثل بخش یک تعریف کردیم اما به دقت مناسبی نرسیدیم ازطرفی تعداد داده ها در این دیتاست هم کمتر از دیتاست بخش اول بود بنابراین از ایده ی ضریب یادگیری پلکانی استفاده کردیم. در این شیوه مقدار ضریب یادگیری با توجه به تعداد اپوک های انجام شده متغیر است و از یک تابع به نام call_back استفاده کردیم . کار این تابع این است که در هر اپوک ضریب یادگیری تغییر یافته را به مدل میدهد.

```
LR_START = 0.00001
LR_MAX = 0.00005 * strategy.num_replicas_in_sync
LR_MIN = 0.00001
LR_RAMPUP_EPOCHS = 5
LR_SUSTAIN_EPOCHS = 0
LR_EXP_DECAY = .8


def scheduler(epoch):
    if epoch < 4:
        return 0.0005
    elif epoch < 8:
        return 0.0002
    elif epoch < 12:
        return 0.0001
    elif epoch < 16:
        return 0.00005
    elif epoch < 20:
        return 0.00002
    else:
        return 0.00001

lr_callback = tf.keras.callbacks.LearningRateScheduler(scheduler, verbose = True)
```

که نمودار آن به صورت زیر خواهد بود:

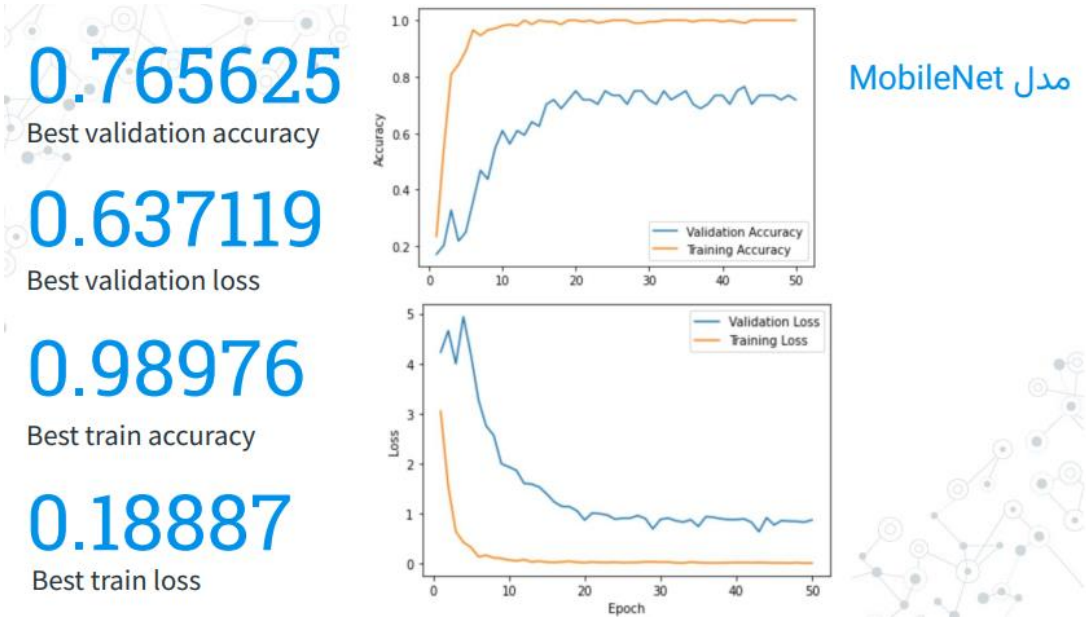


در نهایت نمودار های val_accuracy و accuracy را رسم میکنم تا بتوانیم نتایج را به صورت بصری مشاهده کنیم.

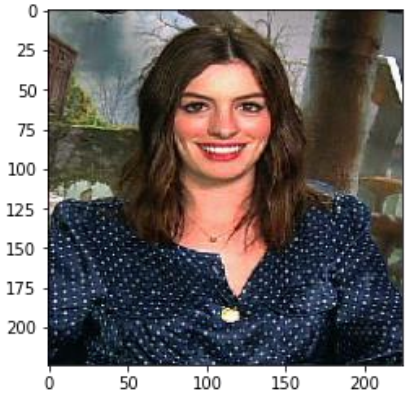
```
# Get training and test loss histories
val_acc = history.history['val_accuracy']
training_acc = history.history['accuracy']

# Create count of the number of epochs
epoch_count = range(1, len(val_acc) + 1)

# Visualize loss history
plt.figure()
plt.plot(epoch_count, val_acc)
plt.plot(epoch_count, training_acc)
plt.legend(['Validation Accuracy', 'Training Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show();
```

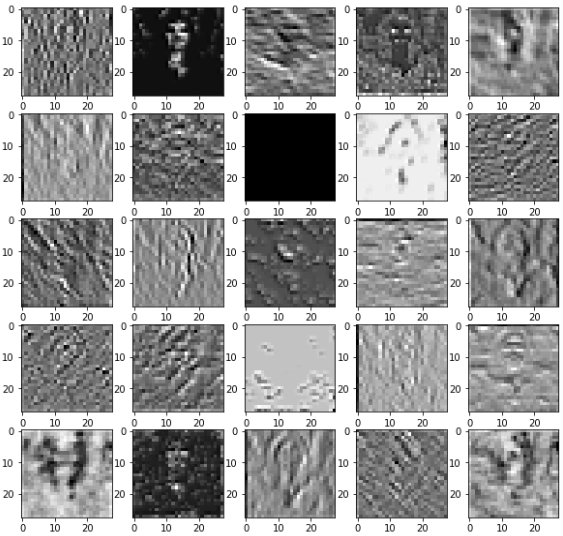
در آخر نتایج به صورت زیر خواهد شد:

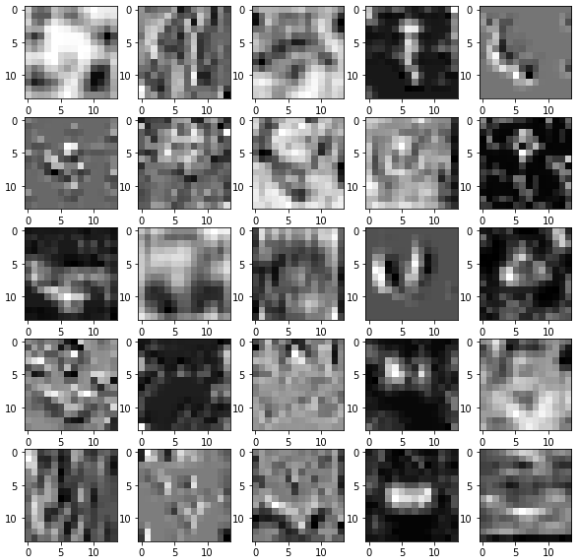
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
mobilenet_1.00_224 (Function (None, 7, 7, 1024)        3228864
_________________________________________________________________
global_average_pooling2d (Gl (None, 1024)              0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 14)                14350
=================================================================
Total params: 3,243,214
Trainable params: 3,221,326
Non-trainable params: 21,888
_________________________________________________________________
```

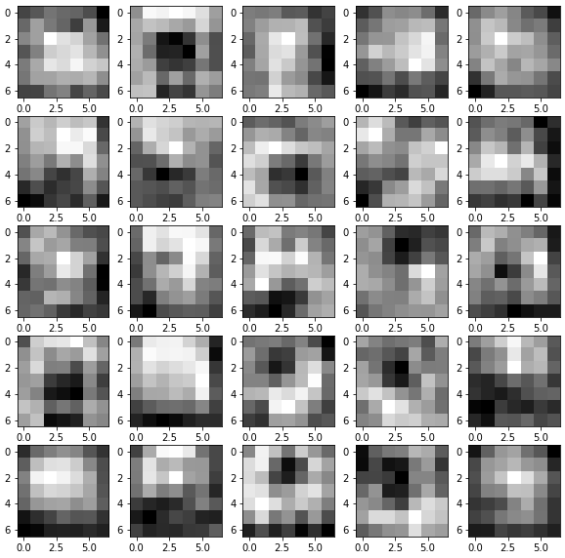


به عنوان نمونه برای یک عکس چند لایه کانولوشن به صورت زیر به ترتیب لایه ها از چپ به نمایش داده شده اند که فایل کامل آن در پوشه درون زیپ قرار دارد که شماره عکس شماره لایه آن است:





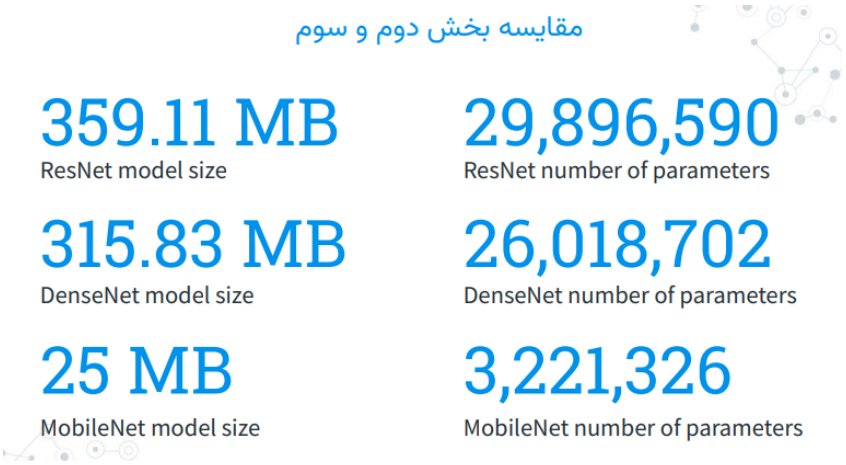




فصل پنجم، نتیجه گیری

۵-۱ مقایسه حجم مدل دیتاست افراد مشهور

هدف از بخش سوم این است که مدل مان برای استفاده بر روی موبایل همچنان صحت خوبی داشته باشد اما حجم و پارامتر های آن به میزان قابل قبولی کاهش یابد تا در موبایل قابل استفاده باشد. با ذخیره کردن و مقایسه حجم و تعداد پارامتر های مدل های بخش دوم و سوم پروژه مشاهده می کنیم که با استفاده از مدل mobilenet حجم و تعداد پارامتر ها به مقدار خوبی کاهش یافته و صحت نیز همچنان قابل قبول است.



۵-۲ روش های بهبود مدل

در بخش اول در ابتدا مدل densenet را بر دادگان تست سابمیت کردیم که به صحت ۰,۹۲ رسید. همچنین مدل efficientnet را نیز جداگانه سابمیت کرده که به صحت ۰,۹۳ منجر شد. پس از بررسی عملکرد مطلوب دو مدل densenet و efficientnet برای بالاتر رفتن صحت بر دادگان تست هر دو مدل را باهم ترکیب کرده و به روش میانگین گیری وقتی هر دو مدل را بر دادگان تست سابمیت کردیم به نتیجه صحت ۰,۹۵ رسیدیم که نشان می دهد شبکه بهتر شده است.

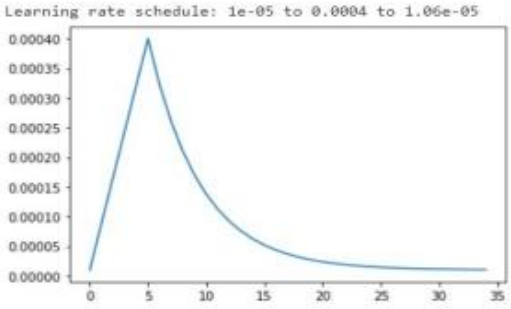
در بخش دوم به دلیل آنکه دیتاست خیلی کوچک است قبل از استفاده از مدل های ذکر شده، ابتدا با یک CNN ساده شبکه را آموزش دادیم اما هرچه المان های مختلف نظیر توابع فعالساز، تعداد لایه ها و نوع لایه ها، توابع خطا و optimizer ها را تغییر داریم، اما صحت شبکه بر دادگان ارزیابی از ۵۰ درصد بالاتر نرفت که در نهایت معماری شبکه را تغییر دادیم.

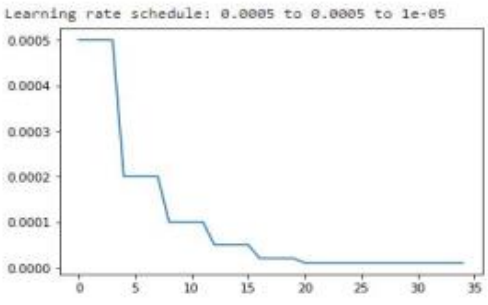
در بخش سوم نیز درابتدا تنها از مدل mobile net استفاده کردیم که به اورفیت منجر شد و در ادامه با استفاده از Drop out این مشکل را بر طرف کردیم.

۵-۳ نحوه استفاده از ضریب یادگیری

در بخش اول در ابتدا ضریب یادگیری را ثابت در نظر گرفته بودیم اما به نتیجه مطلوبی نمی رسیدیم. به همین منظور آن را متغیر و به صورت پیوسته کاهشی در نظر گرفته که باعث می شود در طی تکرار های اموزش مدل، شبکه صحت بالاتری بر دادگان ارزیابی داشته باشد.

در بخش سوم نیز این مدل را تکرار و ضریب یادگیری را مثل بخش یک تعریف کردیم اما به دقت مناسبی نرسیدیم ازطرفی تعداد داده ها در این دیتاست هم کمتر از دیتاست بخش اول بود بنابراین از ایده ی ضریب یادگیری پلکانی استفاده کردیم که نتیجه بهتری داد.





منابع و مراجع

[۱] https://bhashkarkunal.medium.com/face-recognition-real-time-webcam-face-recognition-system-using-deep-learning-algorithm-and-۹۸cf۸۲٥٤def۷

[۲] http://cafetadris.com/blog/%D۸٪B۱٪D۸٪B۲٪D۹٪۸٦٪D۸٪AA-resnet/

[۳] https://hooshio.com/%D۸٪B٤٪D۸٪A۸٪DA%A۹٪D۹٪۸۷-%D۹٪BE%DB٪۸C%DA٪۸٦٪D۸٪B٤٪DB٪۸C-%D۹٪۸٥٪D۸٪AA%D۸٪B۱٪D۸٪A۷٪DA%A۹٪D۹٪۸٥-%DB٪۸C%D۸٪A۷-densenet-%D۹٪۸۸-%D۹٪۸٥٪D۸٪B۱٪D۹٪۸۸٪D۸٪B۱٪DB٪۸C-%D۸٪A۸٪D۸٪B۱-%D۸٪A۲٪D۹٪۸٦-%D۸٪AF%D۸٪B۱-%D۹٪۸٥/

[٤] https://virgool.io/@farzane_hatami/efficientnet-v۱-efficientnet-v۲-fzckic۳k۹lft

[٥] https://howsam.org/mobilenet/