

بسمه تعالیٰ



پروژه پایانی یادگیری عمیق

Stacked ensemble learning traffic-sign recognition

تهییه کننده :

زهرا ختنلو

استاد:

آقای دکتر حامد ملک

تیر ۱۴۰۱

مقدمه

در دنیای امروز شبکه های عصبی و یادگیری عمیق در حوزه های مختلف مورد استفاده قرار می گیرند و توانسته‌اند در چالش ها و مسائل گسترده‌ای مورد استفاده قرار بگیرند. یکی از حوزه هایی که یادگیری عمیق مورد استفاده قرار گرفته است، زمینه ماشین های خودران است که لازم است اتومبیل ها به صورت خودکار در کی از فضای پیرامون خود داشته باشند، بنابراین یکی از زیرنسائلی که می‌توان تعریف کرد تشخیص علائم راهنمایی و رانندگی توسط اتومبیل هاست تا بتوانند به صورت خودکار قوانین را در محیط خود تشخیص بدهند. در این پژوهه قصد داریم این مسئله را به کمک شبکه های عصبی مختلف حل کنیم، شبکه هایی که هر یک نقاط قوت خاصی دارند و اگر بتوان این شبکه ها را در کنار یکدیگر قرار داد می‌توانند نقص های یکدیگر را نیز پیوشنند و نتیجه بهتری را ارائه دهند. در این پژوهه پنج شبکه انتخاب شده است که در نهایت به صورت پشته ای در کنار هم قرار گرفته‌اند و جمهه بندی بین پیش‌بینی این شبکه ها توانسته است نتیجه فوق العاده صدرصد را به ارمغان بیاورد.

گزارشی از پیاده سازی انجام شده

در این پژوهه به علت آموزش دادن چند شبکه که هریک پارامتر های زیادی داشتند و از چند ایپاک برای آموزش آن استفاده شده است به CPU نیاز داشتیم به همین دلیل از سرویس colab استفاده کردیم.

بنابراین در سلول اول دستور مقابل تنها مشخصات منابعی که اختصاص داده شده است را نمایش می‌دهد.

```
!nvidia-smi
Sun Jul 10 18:08:11 2022
+-----+
| NVIDIA-SMI 460.32.03     Driver Version: 460.32.03    CUDA Version: 11.2 |
+-----+
| GPU  Name      Persistence-M| Bus-Id     Disp.A  Volatile Uncorr. ECC | | |
| Fan  Temp  Perf  Pwr:Usage/Cap| Memory-Usage | GPU-Util  Compute M. |
|                               |             |            | MIG M. |
+-----+
|   0  Tesla T4           Off  00000000:00:04.0 Off           0 |
| N/A   36C    P8    9W /  70W \ 0MiB / 15109MiB |      0%  Default |
|                               |             |            | N/A |
+-----+
+-----+
| Processes:                               |
| GPU  GI  CI      PID  Type  Process name          GPU Memory |
| ID   ID              ID               Usage          |
+-----+
| No running processes found                         |
+-----+
```

در ادامه برای آنکه با هر بار اجرا فایل های دیتابانس از درایو اکانت مورد نظر خوانده شود باید این دستور را اجرا کرده تا درایو به نوت بوک در حال اجرا mount شود.

```
[ ] from google.colab import drive  
drive.mount('/content/drive')  
  
[ ] Mounted at /content/drive
```

در سلول بعدی نیز کتابخانه های مورد نیاز در پیاده سازی های انجام شده را import می کنیم.

دیتابانس استفاده شده در این پژوهه مجموعه تصاویری از ۴۳ علائم راهنمایی و رانندگی است و در کل ۵۱۸۳۹ تصویر بوده که در اسکریپتی این تصاویر به شکل دادهای ماتریسی درآمده و به نسبت ۷۰٪/۱۰٪ و ۲۰٪ برای داده های آموزشی، اعتبار سنجی و تست تقسیم شده و برای آنکه لازم نباشد هر بار این عملیات روی داده ها اجرا شود به شکل فایل های pickle در درایو مونت شده به پژوهه ذخیره شده است.

```
[ ] training_file = "/content/drive/MyDrive/train.p"  
validation_file= "/content/drive/MyDrive/valid.p"  
testing_file = "/content/drive/MyDrive/test.p"  
  
with open(training_file, mode='rb') as f:  
    train = pickle.load(f)  
with open(validation_file, mode='rb') as f:  
    valid = pickle.load(f)  
with open(testing_file, mode='rb') as f:  
    test = pickle.load(f)
```

در سلول فوق داده ها از فایل خوانده شده و در متغیر های متناظر ذخیره شده است و در ادامه نیز اطلاعات این دادگان به شکل زیر نمایش داده شده است که ابعاد داده ها و تعداد کلاس ها را مشخص می کند.

```
▶ #Split Train, Validation and Test samples and labels
X_train, y_train = train['features'], train['labels']
X_valid, y_valid = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']

# Check to ensure features equals labels for each data set
assert(len(X_train) == len(y_train))
assert(len(X_valid) == len(y_valid))
assert(len(X_test) == len(y_test))

# Print shapes of training, validation and test data
print("X_train shape:", X_train[0].shape)
print("y_train shape:", y_train.shape)
print("X_valid shape:", X_valid[0].shape)
print("y_valid shape:", y_valid.shape)
print("X_test shape:", X_test[0].shape)
print("y_test shape:", y_test.shape)

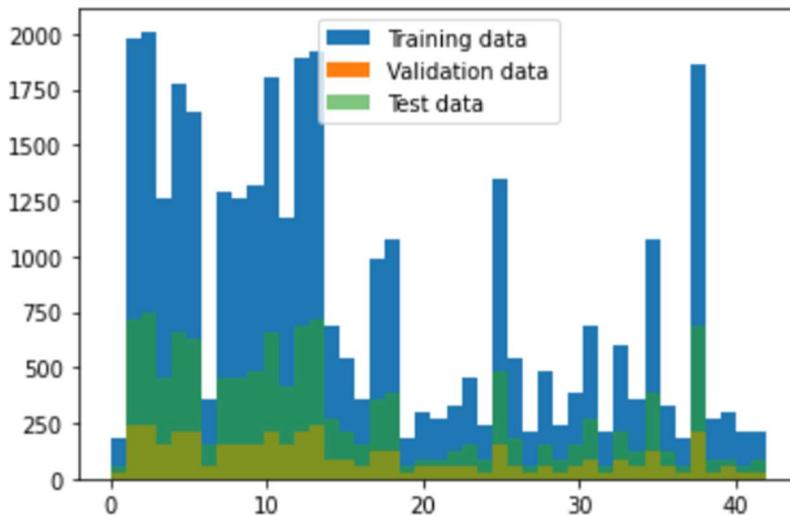
n_train = len(X_train)
n_validation = len(X_valid)
n_test = len(X_test)
image_shape = X_train[0].shape
n_classes = len(np.unique(y_train))
print("Number of classes =", n_classes)

⇒ X_train shape: (32, 32, 3)
y_train shape: (34799,)
X_valid shape: (32, 32, 3)
y_valid shape: (4410,)
X_test shape: (32, 32, 3)
y_test shape: (12630,)
Number of classes = 43
```

```
a = y_train
b = y_valid
c = y_test
bins = n_classes
plt.hist(a, bins, alpha = 1.0, label='Training data')
plt.hist(b, bins, alpha = 1.0, label='Validation data')
plt.hist(c, bins, alpha = 0.6, label='Test data')
plt.legend(loc='upper center')

plt.show()
```

در این قسمت برای آنکه یک بازنمایی کلی از داده ها داشته باشیم، دیتاست های train, valid و test را بر اساس تعداد تصاویر مربوط به هر کلاس رسم می کنیم. همانطور که می دانیم ۴۳ کلاس در داده هایمان تعریف شده است که یک نودار هیستوگرام برای مشخص کردن پراکندگی داده ها در کلاس ها را نمایان می کند.



در ادامه برای افزایش داده ها و همچنین کاهش حساسیت شبکه ها به جهت تصاویر از data augmentation استفاده می کنیم به این صورت که هر تصویر را د درجه به چپ و راست می چرخانیم و سپس برای حفظ ابعاد تصاویر به شکل، یکسان لازم است که تصاویر چرخیده شده را برش دهیم. بنابراین دوتابع زیر را تعریف می کنیم. در تابع اول با استفاده از دستور `scipy.ndimage.rotate(im, degree)` تصویر ورودی به میزان degree چرخیده می شود. در تابع دوم نیز هر تصویر از آرایه ورودی به کمک دستور `imager[i][2:34,2:34]` به اندازه 32×32 برش داده می شود.

```
## Data augmentation-Rotation
def rotate_image(imager,degree):
    X_rotated=[]
    for im in (imager):
        rotated_image = scipy.ndimage.rotate(im, degree)
        X_rotated.append(rotated_image)
    return X_rotated

def crop_image(imager):
    for i in range(len(imager)):
        imager[i] = imager[i][2:34,2:34] # box=(y:y+crop, x:x+crop)
    return imager
```

در ادامه از توابع تعریف شده برای داده های آموزشی مان استفاده کرده و آرایه های خورجی را به دیتابست اولیمان اضافه می کنیم همچنین لازم است که برچسب های داده های نیز به `y_train` اضافه شود و چون داده ها با استفاده از دستور `append` در حلقه اضافه شده اند به این معنی است که هم اکنون در دیتابستان سه تصویر مشابه پشت سر هم داریم که تنها ۱۰ درجه به چپ و راست چرخیده شده اند، بنابراین باید برچسب ها را نیز در یک حلقه دوبار به `y_train` اضافه کنیم.

```
# Convert the data into list type to use the method "append"
X_train = list(X_train)

# combine the Lists
for i in range(len(X_train_rotated_positive)):
    X_train.append(X_train_rotated_positive[i])
    X_train.append(X_train_rotated_negative[i])

#Convert the data back to a np.array
X_train = np.array(X_train)

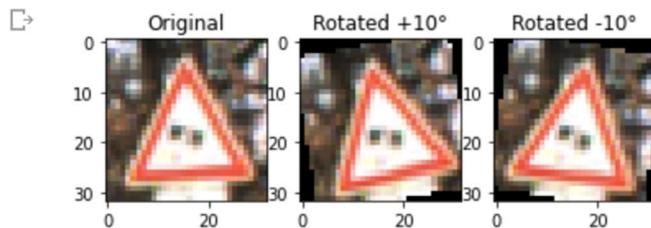
# New number of training examples (after data augmentation)
new_n_train = len(X_train)
print("New number of training samples after data augmentation =", new_n_train)
# do the same for the labels y_train
# Convert the data into list type to use the method "append"
y_train = list(y_train)

# lengthen the list
for i in range(len(y_train)):
    y_train.append(y_train[i])
    y_train.append(y_train[i])
#Convert the data back to a np.array
y_train = np.array(y_train)
# New length of ground truth labels
new_n_train_y = len(y_train)
print("New length of ground truth labels =", new_n_train_y)
```

در سلول بعدی با استفاده از توابع کتابخانه `matplotlib.pyplot` داده های چرخیده شده را به عنوان نمونه نمایش می دهیم. و در خط آخر دستورات این سلول به همان دلیلی که بیان شد در دیتابستان سه تصویر از یک کلاس پشت سر هم آمده ایند تصاویر را در با استفاده از دستور `shuffle` از این نظم خارج می کیم.

```
# Display an example for rotation

fig, (ax1, ax2, ax3) = plt.subplots(1,3)
ax1.imshow(X_train[2018])
ax1.set_title('Original')
ax1.axis('ON') # clear x- and y-axes
ax2.imshow(X_train_rotated_positive[2018])
ax2.set_title('Rotated +10°')
ax2.axis('ON') # clear x- and y-axes
ax3.imshow(X_train_rotated_negative[2018])
ax3.set_title('Rotated -10°')
ax3.axis('ON') # clear x- and y-axes
plt.show()
X_train, y_train = shuffle(X_train, y_train)
```



در مرحله بعد باید داده هایمان را نرمال سازی کنیم بنابراین یک تابع به شکل زیر تعریف می کنیم که توسط `cv2.equalizeHist` کنترانست تصویر را نرمال می کند و ان تابع در سلول بعدی روی تمام داده ها فراخوانی می شود.

```
[ ] def normalize(image):
    normal=[]
    for im in image:
        norm=cv2.equalizeHist(norm2)
        normal.append(np.reshape(norm, (im.shape[0],im.shape[1],im.shape[2])))
    return normal

#Normalize images
import gc
gc.collect()
X_train_normalized = []
X_valid_normalized = []
X_test_normalized = []
X_train_normalized=normalize(X_train)
X_valid_normalized=normalize(X_valid)
X_test_normalized=normalize(X_test)

X_train_normalized=np.array(X_train_normalized)
X_valid_normalized=np.array(X_valid_normalized)
X_test_normalized=np.array(X_test_normalized)

print(X_train_normalized.shape)
print(X_valid_normalized.shape)
print(X_test_normalized.shape)
```

حال پس از آماده کردن داده ها باید مدل هایمان را بسازیم. همانطور که در گزارش بیان شده است، در این پروژه ابتدا چند شبکه از پیش آموزش داده شده انتخاب می کنیم و با اضافه کردن چند لایه و آموزش دادن پارامتر های شبکه ها متناسب با دیتاست مسئله مان شبکه های fine-tune شده جدیدی می سازیم. به علاوه در

فاز اول یک شبکه CNN نیز بدون استفاده از شبکه های از قبل آموزش دیده شده، طراحی کردیم. شبکه هایی که با آزمون و بررسی نتایج هر یک روی دیتابست این مسئله انتخاب شده‌اند عبارتند از: MobileNet، DenseNet، ResNet، EfficientNet می‌پردازیم.

مدل CNN :

CNN یک ساختار ریاضی است که معمولاً از سه نوع لایه تشکیل شده است: کانولوشن، pooling و لایه‌های fully-connected. دو لایه اول، لایه‌های کانولوشن و pooling، استخراج ویژگی را انجام می‌دهند، در حالی که لایه سوم، ویژگی‌های استخراج شده را در خروجی نهایی، مانند طبقه بندی، نگاشت می‌کند. این لایه کانولوشن نقش کلیدی را در مدل CNN ایفا می‌کند، که از مجموعه‌ای از عملیات ریاضی تشکیل شده است، مانند عملیات کانولوشن و یک نوع خاص از عملیات خطی. در تصاویر دیجیتال، مقادیر پیکسل در یک شبکه دو بعدی ذخیره می‌شود، به عنوان مثال، آرایه‌ای از اعداد، و شبکه کوچکی از پارامترها به نام کرنل، یک استخراج کننده ویژگی قابل بهینه، در هر موقعیت تصویر اعمال می‌شود. ، که CNN‌ها را برای پردازش تصویر بسیار کارآمد می‌کند، زیرا یک ویژگی ممکن است در هر جایی از تصویر رخ دهد. همانطور که یک لایه خروجی خود را به لایه بعدی تغذیه می‌کند، ویژگی‌های استخراج شده می‌توانند به صورت سلسله مراتبی و به تدریج پیچیده‌تر شوند.

```

#Building the CNN model
modelCNN=None
modelCNN = Sequential()
modelCNN.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same'))
modelCNN.add(Activation('relu'))
modelCNN.add(MaxPooling2D(pool_size=(2, 2)))

modelCNN.add(Conv2D(64, (3, 3), padding='same'))
modelCNN.add(Activation('relu'))
modelCNN.add(MaxPooling2D(pool_size=(2, 2)))

modelCNN.add(Conv2D(128, (3, 3), padding='same'))
modelCNN.add(Activation('relu'))
modelCNN.add(MaxPooling2D(pool_size=(2, 2)))

modelCNN.add(Flatten()) # this converts our 3D feature maps to 1D feature vectors
modelCNN.add(Dense(512))
modelCNN.add(Activation('relu'))
modelCNN.add(Dropout(0.5))
modelCNN.add(Dense(43))
modelCNN.add(Activation('softmax'))

modelCNN.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
                  optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                  metrics=[ 'accuracy'])

plot_model(modelCNN, to_file='/content/drive/MyDrive/modelCNN.png', show_shapes=True, show_layer_names=True)
modelCNN.summary()

historyCNN=modelCNN.fit(X_train, y_train, batch_size=64, epochs=20, validation_data=(X_valid, y_valid))
modelCNN.save('/content/drive/MyDrive/modelCNN.h5')

```

در پیاده سازی این شبکه از دستور (`Sequential`) استفاده می کنیم که کمک می کند لایه های شبکه را تعریف و به هم متصلشان کنیم. ابتدا یک لایه کانولوشن دو بعدی که ورودی آن تصاویر 32×32 و رنگی یعنی بعد ۳ هستند را به شبکه اضافه می کنیم. سپس یک تابع فعالیت ReLU و یک لایه maxpooling برای پیدا کردن ویژگی هایی همچون لبه و... اضافه می کنیم. این بلوک هار ۱ سه بار با ابعاد ورودی متفاوت (به دلیل اینکه ابعاد داده ها در طول مسیر شبکه با توجه به اعمال عملیات های کانولوشن و pooling تغییر می کند) به شبکه اضافه می کنیم. و در نهایت نیز برای آنکه بتوانیم طبقه بندی را به ۴۳ کلاسی که برای علائم راهنمایی تعریف شده است انجام دهیم از یک لایه `dense` با تابع فعالیت softmax استفاده می کنیم. این لایه در واقع یک شبکه fully-connected است که ابعاد خروجی را به مقداری که در ورودی دریافت کرده تغییر می دهد و تابع فعالیت آمده را نیز روی داده ها اعمال می کند.

در مرحله بعد مدل تعریف شده را با بهینه کننده adam و تابع SparseCategoricalCrossentropy به عنوان تابع loss می سازیم. در نهایت نیز شبکه را با مقدار های پارامتر epoch=20 و batch-size=64 روی داده های آموزشی، آموزش می دهیم و اطلاعات خروجی آن را در متغیری به نام historyCNN ذخیره می کنیم.

```
[ ] from sklearn.metrics import accuracy_score, recall_score
results = modelCNN.evaluate(X_train, y_train, batch_size=32)
results = modelCNN.evaluate(X_valid, y_valid, batch_size=32)
results = modelCNN.evaluate(X_test, y_test, batch_size=32)

pred1 = np.argmax(modelCNN.predict(X_test), axis=-1)
print(accuracy_score(y_test, pred1))
print(recall_score(y_test, pred1, average='micro'))
```

سلول فوق برای مشاهده نتایج روی داده های آموزشی، validation و تست پیاده سازی شده است که در انتهای گزارش به تحلیل و بررسی آن ها می پردازیم.

```
▶ plt.plot(historyCNN.history['accuracy'])
plt.plot(historyCNN.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(historyCNN.history['loss'])
plt.plot(historyCNN.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

در سلول بعدی با کمک توابع plt به رسم نموداری برای نمایش درصد دقت در ایپاک های مختلف برای داده های آموزشی و validation می پردازیم.

مدل MobileNet

این مدل از زیرمجموعه های شبکه های CNN ای است و اولین برای استفاده در سیستم های بینایی موبایل ها طراحی شد و به همین دلیل لازم بود که در عین حفظ دقت شبکه، تعداد پارامتر های آن کاهش یابد. MobileNet از کانولوشن های قابل تفکیک عمیق استفاده می کند. این به طور قابل توجهی تعداد پارامترها را در مقایسه با شبکه های معمول CNN ای و با عمق یکسان کاهش می دهد. درنتیجه شبکه های عصبی عمیق سبک وزن ایجاد می شوند.

یک کانولوشن قابل تفکیک عمیق از دو عملیات ساخته شده است.

- کانولوشن عمیق

نوعی پیچیدگی است که در آن یک فیلتر کانولوشن برای هر کanal ورودی اعمال می کنیم. در کانولوشن دو بعدی معمولی که روی چندین کanal ورودی انجام می شود، فیلتر به اندازه ورودی است و به ما اجازه می دهد آزادانه کanal ها را برای تولید هر عنصر در خروجی ترکیب کنیم.

• کانولوشن نقطه ای

نوعی کانولوشن است که از یک کرنل 1×1 استفاده می کند: هسته ای که در هر نقطه تکرار می شود. این کرنل دارای یک عمق برای هر تعداد کanalی است که تصویر ورودی دارد.

```
▶ #Model Mobile Net

base_modelMN = tf.keras.applications.MobileNet(classes=43,input_shape = (32, 32, 3), include_top = False, weights = 'imagenet')

modelMN = Sequential()
modelMN.add(base_modelMN)
modelMN.add(Flatten())
modelMN.add(Dense(512, activation='relu'))
modelMN.add(Dropout(0.6))
modelMN.add(Dense(43, activation='sigmoid'))

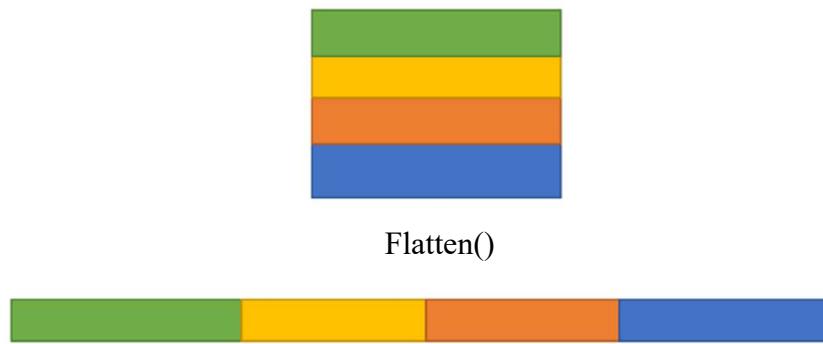
base_learning_rate = 0.001
modelMN.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
                 loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
                 metrics=['accuracy'])

plot_model(modelMN, to_file='/content/drive/MyDrive/modelMN.png', show_shapes=True, show_layer_names=True)
modelMN.summary()

historyMN = modelMN.fit(X_train,y_train,epochs = 20,batch_size=64 , validation_data = (X_valid, y_valid))
modelMN.save('/content/drive/MyDrive/modelMN.h5')
```

در کد بالا ابتدا یک شبکه اولیه mobileNet متناسب با ابعاد تصاویر دیتاستمان و از قبل آموزش دیده شده روی دیتاست (transfer learning) imangenet معرفی می کنیم. همچنین متغیر `include_top` که مقدار `false` می گیرد به این معنی است که اجازه میدهد لایه های دیگری به شبکه اضافه شود. بنابراین در ادامه با استفاده از دستور `Sequential()` می توانیم لایه هایی به شبکه اضافه کنیم.

در این مدل ابتدا شبکه ابتدایی اصلی را و سپس یک لایه `Flatten()` اضافه می کنیم. وظیفخ این لایه آن است که ورودی های به شکل ماتریس را به آرایه یک بعدی تبدیل کند.



در ادامه نیز یک لایه Dense() با تابع فعالیت ReLU و اندازه خروجی ۵۱۲ اضافه می کنیم. این لایه در واقع بعد ورودی خود را به مقدار داده شده درورودی تبدیل می کند و همچنین تابع ReLU را روی ورودی اعمال و در خروجی می دهد. یک لایه dropout نیز به مدل اضافه کردہ ایم تا به صورت رندوم بعضی نورون ها را غیر فعال و از overfitting جلوگیری کند. در انتها نیز مجدداً یک لایه dense برای تبدیل بعد از تعداد ملاس های موجود در دیتا استمان اضافه کردہ ایم تا بتواند عملیات طبقه بندی را انجام داد.

در مرحله بعد مدل تعریف شده را با بهینه کننده Adam و تابع SparseCategoricalCrossEntropy loss آن است که کلاس هایی که می خواهیم طبقه بندی کنیم به صورت دسته ای است.

در انتها مدل ساخته شده را روی داده های آموزشی و با هایپر پارامتر های epoch=20 و batch size = 64 آموزش می دهیم و اطلاعات آن را در historyMN ذخیره می کنیم. (مقادیر این هایپر پارامتر ها سعی شده با آزمون و خطا به دست آید اما به دلیل کمبود منابع قادر به انجام تعداد زیادی از تغییرات برای آن ها نبوده ایم)

```
from sklearn.metrics import accuracy_score,recall_score
results = modelMN.evaluate(X_train, y_train, batch_size=64)
results = modelMN.evaluate(X_valid, y_valid, batch_size=64)
results = modelMN.evaluate(X_test, y_test, batch_size=64)

pred1 = np.argmax(modelMN.predict(X_test), axis=-1)
print(accuracy_score(y_test, pred1))
print(recall_score(y_test, pred1, average='micro'))
```

سلول فوق برای مشاهده نتایج روی داده های آموزشی، validation و تست پیاده سازی شده است که در انتها گزارش به تحلیل و بررسی آن ها می پردازیم.

```

▶ plt.plot(historyMN.history['accuracy'])
plt.plot(historyMN.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(historyMN.history['loss'])
plt.plot(historyMN.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

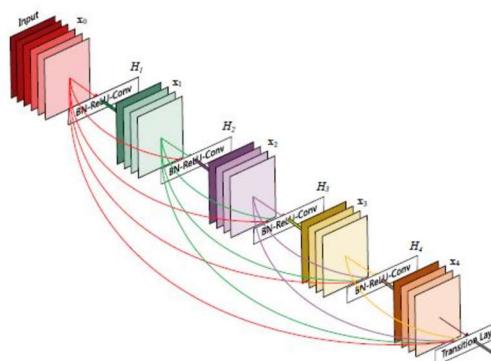
```

در سلول بعدی با کمک توابع plt به رسم نموداری برای نمایش درصد دقت در ایپاک های مختلف برای داده های آموزشی و validation می پردازیم.

مدل DenseNet121

در یک شبکه عصبی کانولوشن معمول (CNN)، هر لایه کانولوشنی به جز اولین لایه (که ورودی را می گیرد)، خروجی لایه کانولوشنی قبلی را دریافت می کند و یک `imap` از ویژگی ها در خروجی تولید می کند که سپس به لایه کانولوشن بعدی منتقل می شود. بنابراین، برای n لایه، n اتصال مستقیم وجود دارد. هر اتصال بین هر لایه و لایه بعدی. بنابراین، با افزایش تعداد لایه ها در CNN، یعنی با عمیق تر شدن آنها، مشکل محوش دگی گردید. این بدان معناست که افزایش مسیر اطلاعات از لایه های ورودی به خروجی، می تواند باعث ناپدید شدن یا گم شدن اطلاعات خاصی شود که توانایی شبکه برای آموزش مؤثر را کاهش می دهد.

DenseNets این مشکل را با اصلاح معماری استاندارد CNN و ساده کردن الگوی اتصال بین لایه ها حل می کند. در معماری DenseNet، هر لایه به طور مستقیم با هر لایه دیگر متصل است، از این رو شبکه نامیده می شود. برای n لایه، $n(n+1)/2$ اتصال وجود دارد.



```
[ ] #model Densnet121

base_modelDN = tf.keras.applications.DenseNet121(classes=43,input_shape = (32, 32, 3), include_top = False, weights = 'imagenet')
modelDN = Sequential()
modelDN.add(base_modelDN)
modelDN.add(Flatten())
modelDN.add(Dense(512, activation='relu'))
modelDN.add(Dropout(0.5))
modelDN.add(Dense(43, activation='sigmoid'))

base_learning_rate = 0.001
modelDN.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
                 loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
                 metrics=['accuracy'])

plot_model(modelDN, to_file='/content/drive/MyDrive/modelDN.png', show_shapes=True, show_layer_names=True)
modelDN.summary()

historyDN = modelDN.fit(X_train,y_train,epochs = 10,batch_size=64 , validation_data = (X_valid, y_valid))
modelDN.save('/content/drive/MyDrive/modelDN.h5')
```

پیاده سازی این مدل نیز همانند مدل قبلی است و تنها در مقدار بعضی پارامتر ها تفاوت وجود دارد و علت کمتر بودن تعداد ایپاک ها به نسبت mobileNet، کمتر بودن تعداد پارامتر های قابل آموزش شبکه است.

در سلول های بعدی نیز پیاده سازی هایی در جهت نمایش نتایج شبکه آمده است.

مدل ResNet50

شبکه های عصبی کانولوشن یک نقطه ضعف بزرگ به نام محوشدگی گرادیان دارند. در حین backpropagation مقدار گرادیان به طور قابل توجهی کاهش می یابد، بنابراین به ندرت تغییری در وزن ایجاد می شود. برای رفع این مشکل مدل resNet از skip connection استفاده می کند.

Skip connection یک اتصال مستقیم است که از روی برخی از لایه های مدل می گذرد. خروجی به دلیل این اتصال پرش یکسان نیست. بدون اتصال پرش، ورودی X در وزن لایه و به دنبال آن یک عبارت بایاس ضرب می شود. سپس تابع فعال سازی خروجی را به صورت زیر دریافت می کنیم:

$$F(w^*x + b) (=F(X))$$

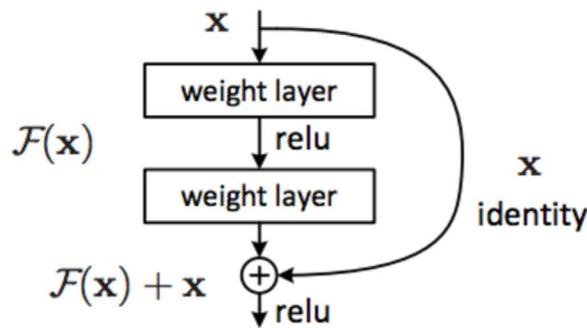
اما با تکنیک اتصال پرش، خروجی به صورت زیر است:

$$F(X)+x$$

در ResNet-50، دو نوع بلوک وجود دارد:

- بلوک هویت
- بلوک کانولوشن

مقدار ' x ' به لایه خروجی اضافه می شود اگر و فقط اگر اندازه ورودی برابر اندازه خروجی باشد. اگر اینطور نیست، یک بلوک کانولوشن در مسیر میانبر اضافه می کنیم تا اندازه ورودی برابر با اندازه خروجی باشد.



نحوه پیاده سازی این مدل نیز همانند مدل های قبلی است و تنها در بخش اضافه کردن لایه ها به شبکه اولیه یک لایه pooling و یک لایه flatten به شبکه اضافه می کنیم. در ادامه نیز مدل را با بهینه کننده RMSprop می سازیم و در نهایت با مقادیر های پر پارامتر ها آموزش می دهیم.

```
▶ from keras.layers.normalization.batch_normalization import BatchNormalization

#Resnet50
from tensorflow.keras.applications import ResNet50V2

base_modelRN = ResNet50V2(input_shape = (32, 32, 3), include_top = False, weights = 'imagenet')
modelRN = Sequential()
modelRN.add(base_modelRN)
modelRN.add(GlobalAveragePooling2D())
modelRN.add(Flatten())
modelRN.add(Dense(43, activation='softmax'))

base_learning_rate = 0.001
modelRN.compile(optimizer=tf.keras.optimizers.RMSprop(lr=0.001),
                 loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
                 metrics=['accuracy'])

plot_model(modelRN, to_file='/content/drive/MyDrive/modelRN.png', show_shapes=True, show_layer_names=True)
modelRN.summary()

historyRN = modelRN.fit(X_train,y_train,epochs = 10 , validation_data = (X_valid, y_valid),steps_per_epoch = 100)
modelRN.save('/content/drive/MyDrive/modelRN.h5')
```

: EfficientNet مدل

یک معماری شبکه عصبی کانولوشن و روش مقیاس بندی است که با استفاده از یک ضریب ترکیبی، تمام ابعاد عمق / عرض / ارزویلشون را به طور یکنواخت مقیاس بندی می کند. برخلاف روش مرسوم که به صورت دلخواه این عوامل را مقیاس بندی می کند، روش مقیاس بندی EfficientNet به طور یکنواخت عرض، عمق و وضوح شبکه را با مجموعه ای از ضرایب مقیاس بندی ثابت مقدار می دهد. برای مثال، اگر بخواهیم منابع محاسباتی 2^n برابر

بیشتری استفاده کنیم، می‌توانیم به سادگی عمق شبکه را α^n ، عرض را β^n برابر و اندازه تصویر را با γ^n افزایش دهیم. این ضرایب ثابت با جستجوی شبکه کوچک در مدل کوچک اصلی تعیین می‌شوند. EfficientNet از یک ضریب ترکیبی برای مقیاس یکنواخت عرض، عمق ووضوح شبکه به روشنی اصولی استفاده می‌کند.

روش مقیاس‌بندی ترکیبی با این تصور توجیه می‌شود که اگر تصویر ورودی بزرگ‌تر باشد، شبکه به لایه‌های بیشتری برای افزایش میدان دریافت و کانال‌های بیشتری برای ثبت الگوهای ریزدانه‌تر روی تصویر بزرگ‌تر نیاز دارد.

```
#Efficient Model
base_modelEN=tf.keras.applications.efficientnet.EfficientNetB0(input_shape = (32, 32, 3), classes=43,include_top = False, weights = 'imagenet',
modelEN = Sequential()
modelEN.add(base_modelEN)
modelEN.add(Dense(43, activation='sigmoid'))

modelEN.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), metrics = ['accuracy'])

plot_model(modelEN, to_file='/content/drive/MyDrive/modelEN.png', show_shapes=True, show_layer_names=True)
modelEN.summary()

historyEN = modelEN.fit(X_train,y_train, validation_data = (X_valid, y_valid), steps_per_epoch = 100, epochs = 20)
modelEN.save('/content/drive/MyDrive/modelEN.h5')
```

نحوه پیاده‌سازی این مدل نیز همانند مدل‌های قبلی است و تنها به شبکه اصلی یک لایه dense با تابع فعالیت sigmoid اضافه می‌شود تا ابعاد خروجی شبکه متناسب با تعداد کلاس‌هایی باشد که در این مسئله تعریف شده است.

سلول‌های بعدی مربوط به بخشیست که لازم است مدل‌های شاخته شده را به شکل پشت‌هار در کنار هم قراردهیم و نتایج پیش‌بینی روی داده‌ای تست را به صورت جمع‌بندی از نتایج هر شبکه که روش آن توضیح داده خواهد شد بدست می‌آوریم.

در سلول اول ابتدا توابع و کتابخانه‌های مورد نیاز این بخش را import می‌کنیم. و در ادامه به تعریف توابع مورد نیاز می‌پردازیم. در قسمت ساخت مدل‌ها، در انتهای همه‌ی سلول‌ها با دستور `model.save('path')` مدل‌های ساخته شده را به صورت فایل‌هایی با فرمت h5 ذخیره کرده ایم تا لازم نباشد برای هربار استفاده از قسمت پشت‌های و پیش‌بینی روی داده‌های تست، پنج شبکه مجدد آموزش ببینند و اجرا شوند. ینابراین در این قسمت اولین تابع این مدل‌های ذخیره شده در آدرس‌های مشخص را می‌خواند و در یک لیستی پشت سرهم مدل‌ها را قرار می‌دهد و در خروجی تابع بر می‌گرداند.

```

from sklearn.datasets import make_blobs
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from keras.models import load_model
from tensorflow.keras.utils import to_categorical
from numpy import dstack

# load models from file

def load_all_models():
    all_models = list()
    model = load_model('/content/drive/MyDrive/modelCNN.h5')
    all_models.append(model)
    model = load_model('/content/drive/MyDrive/modelMN.h5')
    all_models.append(model)
    model = load_model('/content/drive/MyDrive/modelDN.h5')
    all_models.append(model)
    model = load_model('/content/drive/MyDrive/modelRN.h5')
    all_models.append(model)
    model = load_model('/content/drive/MyDrive/modelEN.h5')
    all_models.append(model)
    return all_models

```

تابع بعدی با ورودی لیستی از مدل‌ها که توسط تابع قبلی ساخته می‌شد و x ‌های داده‌ای تست فراخوانی می‌شود، به شکل زیر است:

```

def stacked_dataset(members, inputX):
    stackX = None
    for model in members:
        # make prediction
        yhat = model.predict(inputX, verbose=0)
        # stack predictions into [rows, members, probabilities]
        if stackX is None:
            stackX = yhat
        else:
            stackX = dstack((stackX, yhat))
    # flatten predictions to [rows, members x probabilities]
    stackX = stackX.reshape((stackX.shape[0], stackX.shape[1]*stackX.shape[2]))
    return stackX

```

در این تابع ابتدا پیش‌بینی را به کمک تابع `predict` روی تمام مدل‌ها و با داده‌های x_input انجام می‌دهیم و نتایج بدست آمده را به شکل پشته‌ای با استفاده از تابع `dstack` در کنار هم قرار می‌دهیم. درواقع خروجی تابع `predict` به صورت احتمالاتی محاسبه می‌شود به این صورت که هر سطر از داده‌ها به تعداد کلاس‌های مدنظر خروجی دارد که احتمال تعلق داده هر سطر به هذ کلاس را مشخص می‌کند، بنابراین `yhat` ماتریس‌هایی به ابعاد n^*43 یعنی تعداد داده‌ها و تعداد کلاس‌های طبقه‌بندی است. که در ادامه تابع `dstack` ماتریس‌های ورودی

را به شکل سه‌تایی در بعد سوم در کنار هم به شکل پشته‌ای قرار می‌دهد بنابراین در انتهای stackX یک تنسور به بعد $n^{*}43^{*}5$ است به این معنی که نتایج yhat همه مدل‌ها در کنار هم قرار گرفته‌اند.

```
# fit a model based on the outputs from the ensemble members
def fit_stacked_model(members, inputX, inputy):
    # create dataset using ensemble
    stackedX = stacked_dataset(members, inputX)
    # fit standalone model
    model = LogisticRegression()
    model.fit(stackedX, inputy)
    return model

def stacked_prediction(members, model, inputX):
    # create dataset using ensemble
    stackedX = stacked_dataset(members, inputX)
    # make a prediction
    yhat = model.predict(stackedX)
    return yhat
```

در تابع fit_stack_model هدف آن است که از بین خروجی‌های پنج مدل یک جمع‌بندی و پیش‌بینی نهایی مشخص شود به همین دلیل ابتدا تابع stacked_dataset فراخوانی می‌شود تا پیش‌بینی پنج مدل برای هر سطر به صورت احتمالی برای هر کلاس بدست آید، سپس یک مدل logisticRegression تعریف می‌شود که با آموزش داده می‌شود به این معنا که مدلی ساخته می‌شود که بتواند نزدیک ترین پیش‌بینی stackedX و inputy را داشته باشد. بنابراین تابع fit_stacked_model مدلی را بر اساس پیش‌بینی‌های پنج مدل اولیه که نزدیک ترین حالت به inputy داشته باشد را بر می‌گرداند. و در نهایت تابع stacked_prediction از این دو تابع شرح داده شده استفاده می‌کند و پیش‌بینی نهایی را روی مدل ساخته شده انجام می‌دهد.

```
members = load_all_models()
# evaluate standalone models on test dataset
for model in members:
    _, acc = model.evaluate(X_test, y_test, verbose=0)
    print('Model Accuracy: %.3f' % acc)

# fit stacked model using the ensemble
model = fit_stacked_model(members, X_test, y_test)
# create stacked model input dataset as outputs from the ensemble
# evaluate model on test set
yhat = stacked_prediction(members, model, X_test)
acc = accuracy_score(y_test, yhat)
print('Stacked Test Accuracy: %.3f' % acc)
```

در تیکه کد آمده در انتهای سلول ابتدا مدل های ذخیره شده به صورت فایل را می خوانیم و سپس درصد دقت هر یک از مدل ها در حلقه چاپ می کنیم. در نهایت نیز به کمک تابع های توضیح داده شده مدل پشتهای از چند شبکه را ساخته و پیش بینی را به کمک آن و روی داده های تست انجام می دهیم، سپس درصد دقت این پیش بینی را چاپ می کنیم که در بخش تخلیل نتایج خواهیم دید که این درصد برابر ۱۰۰ است.

```
def most_frequent(List):
    counter = 0
    num = List[0]

    for i in List:
        curr_frequency = List.count(i)
        if(curr_frequency > counter):
            counter = curr_frequency
            num = i

    return num

img = image.load_img('/content/drive/MyDrive/00008.png', target_size = (32,32)) #load the image
x = image.img_to_array(img)
#x = np.expand_dims(x, axis=0)
#images = np.vstack([x])
image1=np.reshape(x,(1,32,32,3))
valu=[]
valu.append(np.argmax(members[0].predict(image1),axis=-1)) #predict the label for the image
valu.append(np.argmax(members[1].predict(image1),axis=-1))
valu.append(np.argmax(members[2].predict(image1),axis=-1))
valu.append(np.argmax(members[3].predict(image1),axis=-1))
valu.append(np.argmax(members[4].predict(image1),axis=-1))
print(valu[0],valu[1],valu[2],valu[3],valu[4])
classes=most_frequent(valu)
if classes[0]==0:
    print('Speed limit (20km/h)') #print the content

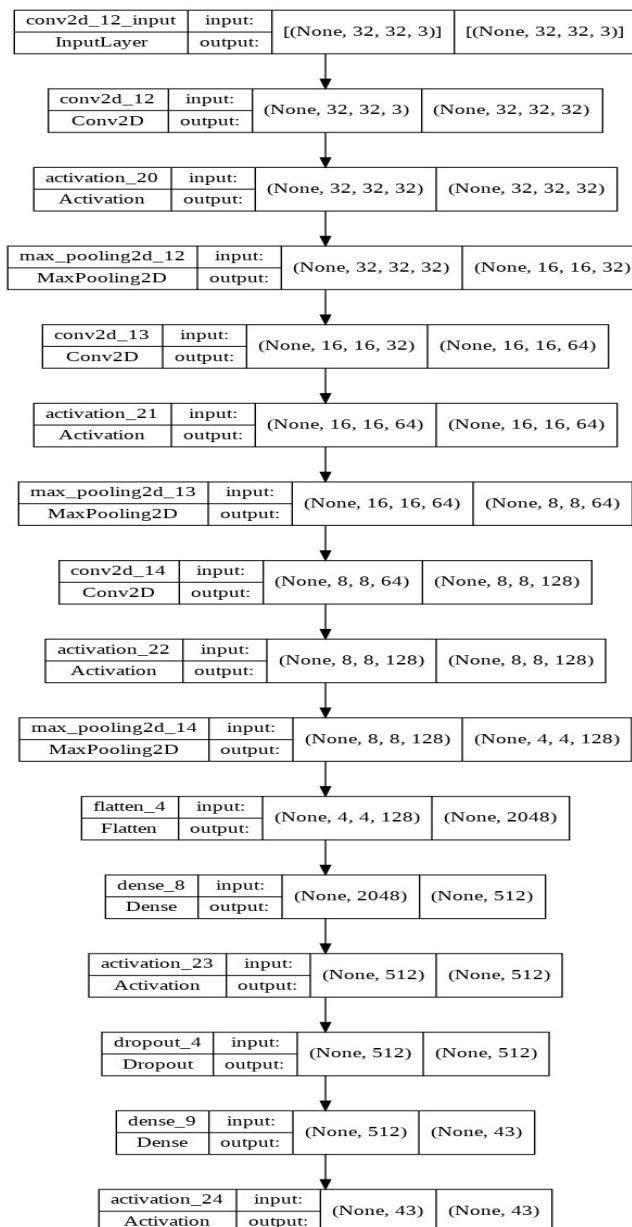
elif classes[0]==1:
    print('Speed limit (30km/h)') #print the content
```

در انتهای پروژه تیکه کدی آمده است که ابتدا تابعی برای برگرداندن مقداری که بیشتر در آرایه ورودی تکرار شده است تعریف می کنیم. در ادامه یک تصویر از فایل میخوانیم و تعلق آن به یکی از ۴۳ کلاس را به هر ۵ مدل انحصار می دهیم و مقادیر پیش بینی شده را در یک آرایه قرار می دهیم و این آرایه را به تابع most_frequent می دهیم تا مشخص کند این پنج مدل بیشتر کدام کلاس را پیش بینی کرده اند و در نهایت مناسب با آن عبارتی را چاپ می کنیم.

تحلیل و بررسی نتایج

همانطور که توضیح داده شده است در این پروژه پنج شبکه ساخته شده که به بررسی جزئیات و نتایج آن ها می پردازم.

: CNN مدل



ساختار کلی شبکه همانطور که در تصویر مشخص است از سه بلوکی که شامل لایه کانولوشن، تابع فعالیت و است تشمیل شده و در انتهای نیز یک شبکه fully-connected آمده تا دسته بندی را بر اساس کلاس ها تعریف شده انجام دهد.

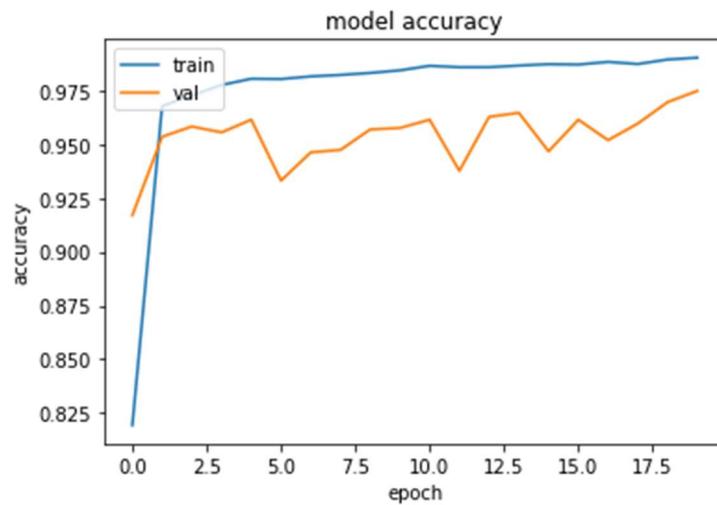
Layer (type)	Output Shape	Param #
=====		
conv2d_12 (Conv2D)	(None, 32, 32, 32)	896
activation_20 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_12 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_13 (Conv2D)	(None, 16, 16, 64)	18496
activation_21 (Activation)	(None, 16, 16, 64)	0
max_pooling2d_13 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_14 (Conv2D)	(None, 8, 8, 128)	73856
activation_22 (Activation)	(None, 8, 8, 128)	0
max_pooling2d_14 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_4 (Flatten)	(None, 2048)	0
dense_8 (Dense)	(None, 512)	1049088
activation_23 (Activation)	(None, 512)	0
dropout_4 (Dropout)	(None, 512)	0
dense_9 (Dense)	(None, 43)	22059
activation_24 (Activation)	(None, 43)	0
=====		
Total params: 1,164,395 Trainable params: 1,164,395 Non-trainable params: 0		

خلاصه بالا از مدل ساخته شده توسط تابع `summary()` رسم شده که اطلاعات کلی شبکه نظیر لایه ها و اندازه ورودی و خروجی آن ها را نمایش می دهد. همچنین همانطور که مشخص است این مدل ۱۱۶۴۳۹۵ پارامتر داشته مه همگی آموزش داده شده اند.

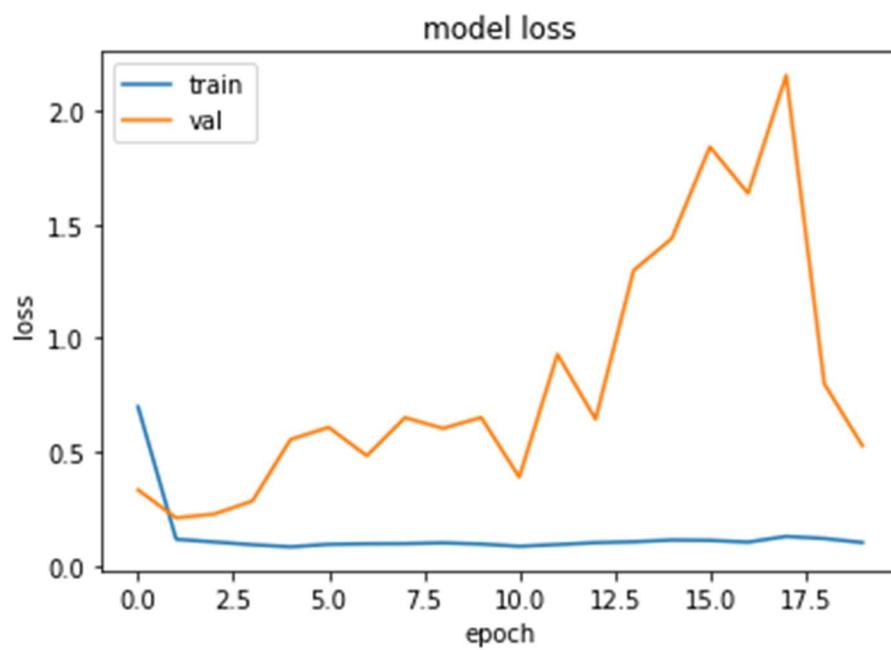
در ادامه دقت شبکه روی دیتابست ها به شکل نمودار مقابل است:

نوع دیتابست	Loss	Accuracy
آموزشی	0.0142	0.99
Validation	0.5274	0.97
تست	1.2261	0.96

نمودار زیر درصد دقت شبکه در طول اجرای ایپاک ها را بر روی داده های آموزشی و validation نمایش می دهد. همانطور که مشخص است به طور کلی دقت دیتاست validation کمتر از آموزشی است و نوسانات آن می تواند نشان دهنده overfit نشدن شبکه باشد.

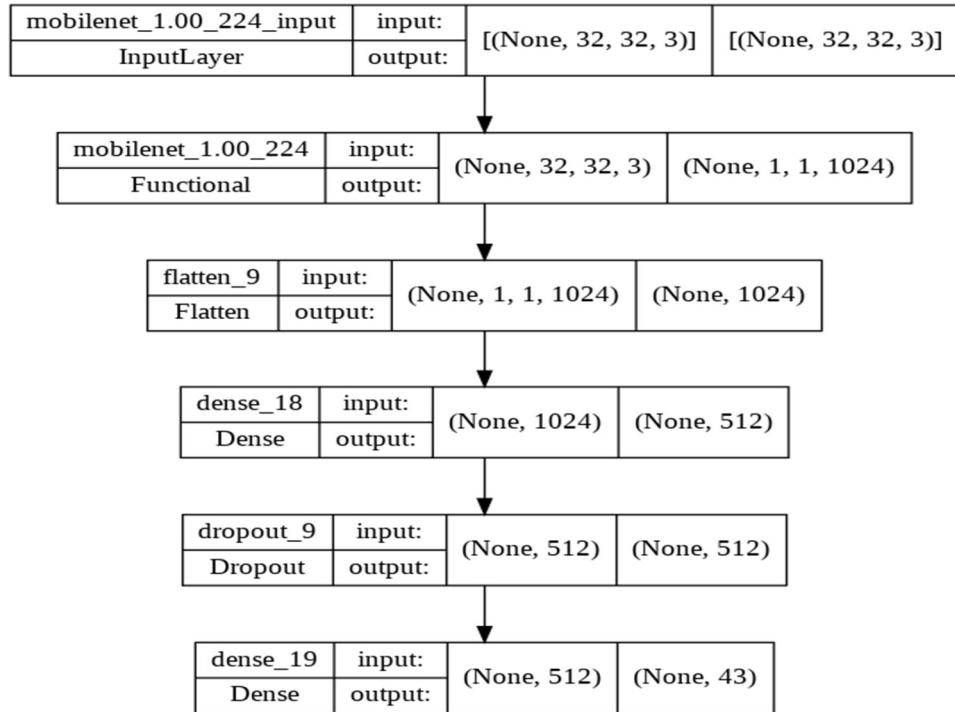


در نمودار بعدی تابع loss دو دیتاست رسم شده است که اهمیت تعداد بیشتر epoch ها را نیز مشخص می کند. همانطور که مشاهده می کنید حتی در بازه ای مقدار تابع افزایش یافته ولی در انتها با افزایش تعداد ایپاک ها مقدار تابع به طور ناگهانی کاهش یافته است.



مدل MobileNet

ساختار کلی این مدل به شکل زیر است که یک شبکه mobileNet از پیش آموزش دیده شده را fine-tune کرده‌ایم و تعدادی لایه به شبکه برای تطابق آن با تسك پروژه اضافه شده است و همچنین لایه dropout برای کاهش احتمال overfit شدن.



در تصویر زیر اطلاعات کلی شبکه را مشاهده می‌کنیم و با توجه به اینکه هدف شبکه آن بود که روی دستگاه‌های امبدد و موبایل‌ها استفاده شود و بنابراین باید تعداد پارامترهایی کمتر باشد می‌تواندیم مشاهده کنیم که ۳۷۵۳۸۳۵ پارامتر دارد و از این تعداد بعضی آموزش داده نشده‌اند.

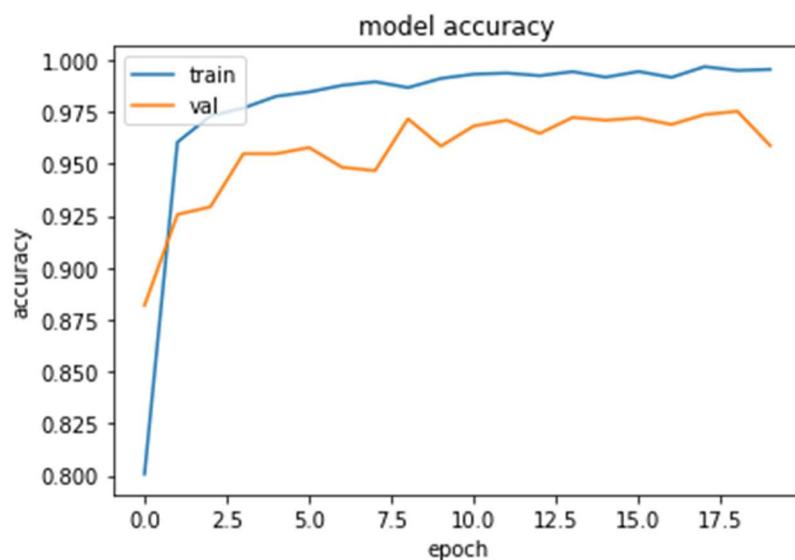
Layer (type)	Output Shape	Param #
mobilenet_1.00_224 (Functional)	(None, 1, 1, 1024)	3228864
flatten_9 (Flatten)	(None, 1024)	0
dense_18 (Dense)	(None, 512)	524800
dropout_9 (Dropout)	(None, 512)	0
dense_19 (Dense)	(None, 43)	22059

Total params: 3,775,723
Trainable params: 3,753,835
Non-trainable params: 21,888

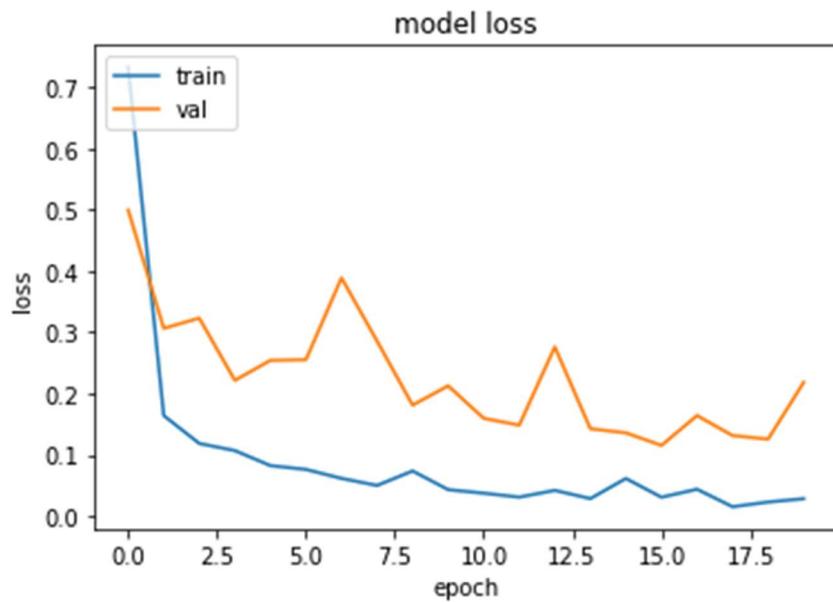
در پیاده سازی این شبکه قسمتی از کد درصد دقت و مقدار تابع loss را روی دیتاست های آموزشی، تست و validation نمایش می داد که به شرح نمودار مقابل است:

Dataset	loss	Accuracy
Train	0.0632	0.98
Validation	0.2184	0.95
Test	0.2570	0.95

نمودار زیر درصد دقت شبکه mobileNet را روی داده های آموزشی و validation نمایش می دهد.

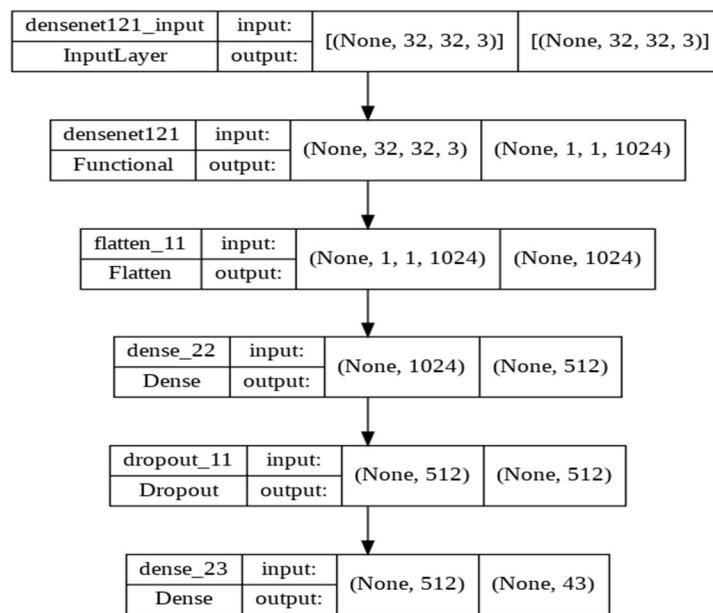


نمودار زیر تابع loss را برای داده های آموزشی و اعتبارسنجی را در طول ۲۰ ایپاک نشان می دهد که همانطور که مشخص است هدف کاهش مقدار آن است.



: DenseNet121 مدل

ساختار کلی این مدل به شکل زیر است که یک شبکه denseNet از پیش آموزش دیده شده را fine-tune کرده ایم و تعدادی لایه به شبکه برای تطابق آن با تسك پروژه اضافه شده است و همچنین لایه dropout برای کاهش احتمال overfit شدن.



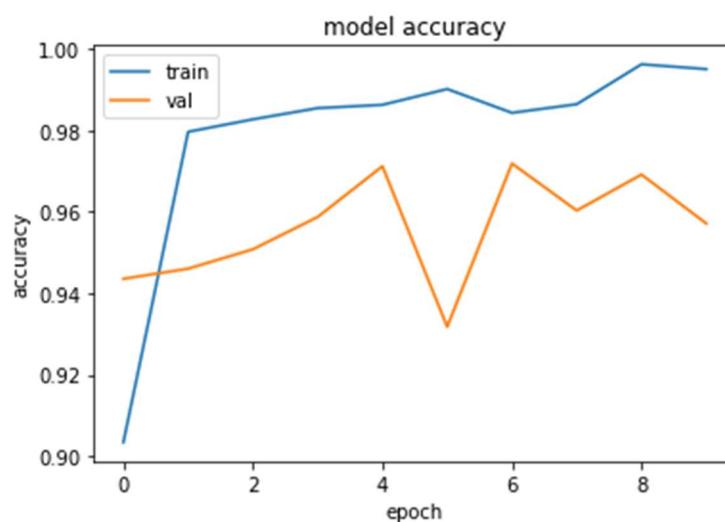
اطلاعات کلی زیر درمورد مدل DenseNet است که لایه ها و ابعاد ورودی و خروجی آنها نمایش می دهد و همچنین آمده است که تعداد پارامتر های این شبکه برابر ۷۵۸۴۳۶۳ است که در مقایسه با شبکه قبلی این میزان به مقدار قابل توجهی افزایش یافته در واقعه می توان گفت که شبکه mobileNet در کاهش تعداد پارامتر ها به خوبی عمل کرده است.

Layer (type)	Output Shape	Param #
<hr/>		
densenet121 (Functional)	(None, 1, 1, 1024)	7037504
flatten_11 (Flatten)	(None, 1024)	0
dense_22 (Dense)	(None, 512)	524800
dropout_11 (Dropout)	(None, 512)	0
dense_23 (Dense)	(None, 43)	22059
<hr/>		
Total params:	7,584,363	
Trainable params:	7,500,715	
Non-trainable params:	83,648	

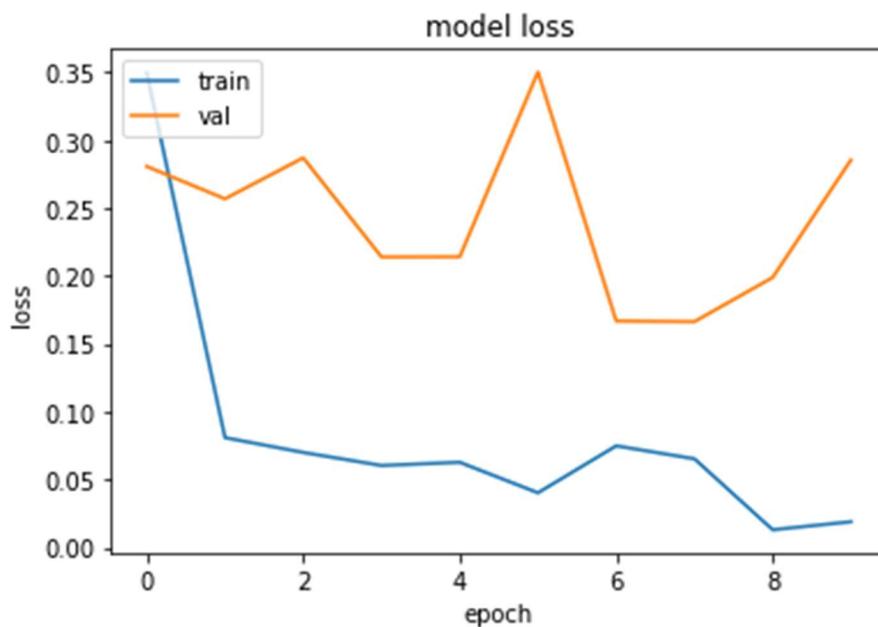
ارزیابی این مدل:

Dataset	Loss	Accuracy
Train	0.0192	0.99
Validation	0.2856	0.95
Test	0.2466	0.95

نمودار زیر درصد دقت مدل روی داده های آموزشی و اعتبار سنجی را نمایش می دهد.

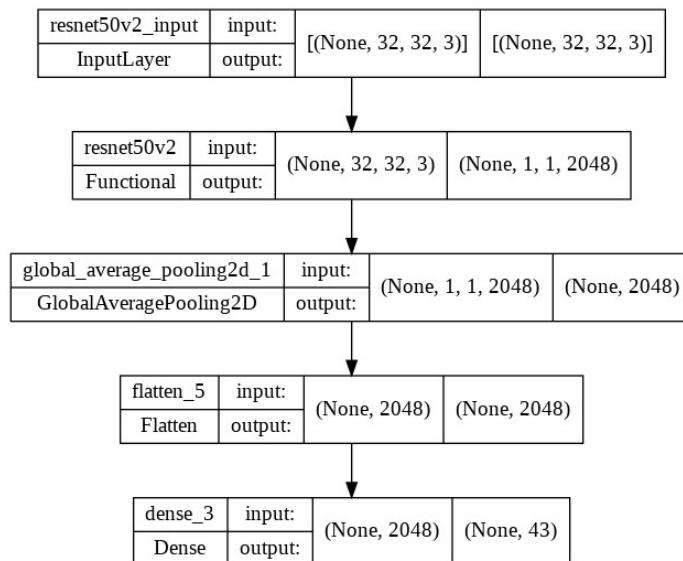


نمودار زیر مقدار تابع loss را نمایش می دهد که با توجه به آن به نظر می رسد این مدل روی داده های آموزشی overfit شده است چرا که تفاوت زیادی بین مقادیر تابع loss داده های آموزشی و اعتبار سنجی وجود دارد و همچنین در داده های اعتبار سنجی فرایند تغییرات در طول ایپاک ها نزولی نبوده است، البته به نظر می رسد که با افزایش تعداد ایپاک ها بتوان تا حدی مشکل را برطرف کرد.



: Resenet50 مدل

ساختار کلی این مدل به شکل زیر است که از یک شبکه از قبل آموزش دیده شده و لایه های pooling و لایه های fully-connected تشکیل شده است.



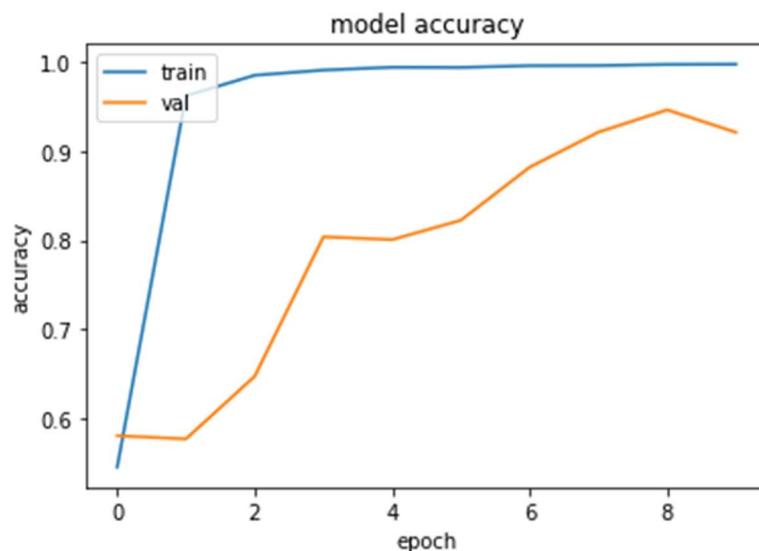
از اطلاعات کلی این مدل مشخص است که تعداد پارامتر های این شبکه به نسبت شبکه های دیگر بسیار زیاد است در حالی مه از نمودار ارزیابی مشخص است که عملکرد آن چندان بهتر نبوده و حتی به نظر می رسد تا حدی دچار overfit نیز شده است.

Layer (type)	Output Shape	Param #
<hr/>		
resnet50v2 (Functional)	(None, 1, 1, 2048)	23564800
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 2048)	0
flatten_5 (Flatten)	(None, 2048)	0
dense_3 (Dense)	(None, 43)	88107
<hr/>		
Total params: 23,652,907		
Trainable params: 23,607,467		
Non-trainable params: 45,440		

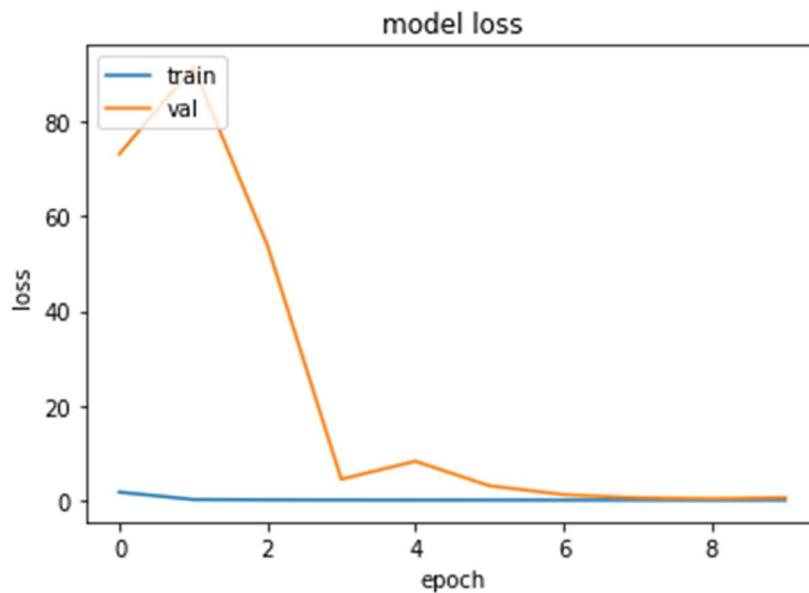
جدول ارزیابی:

Dataset	Loss	Accuracy
Train	0.0944	0.98
Validation	0.5364	0.92
Test	0.6191	0.92

نمودار زیر درصد دقت را نمایش می دهد که می تواند بیانگر اثر گذاری تعداد ایپاک های بیشتر برای دست یابی به دقت بیشتر باشد.

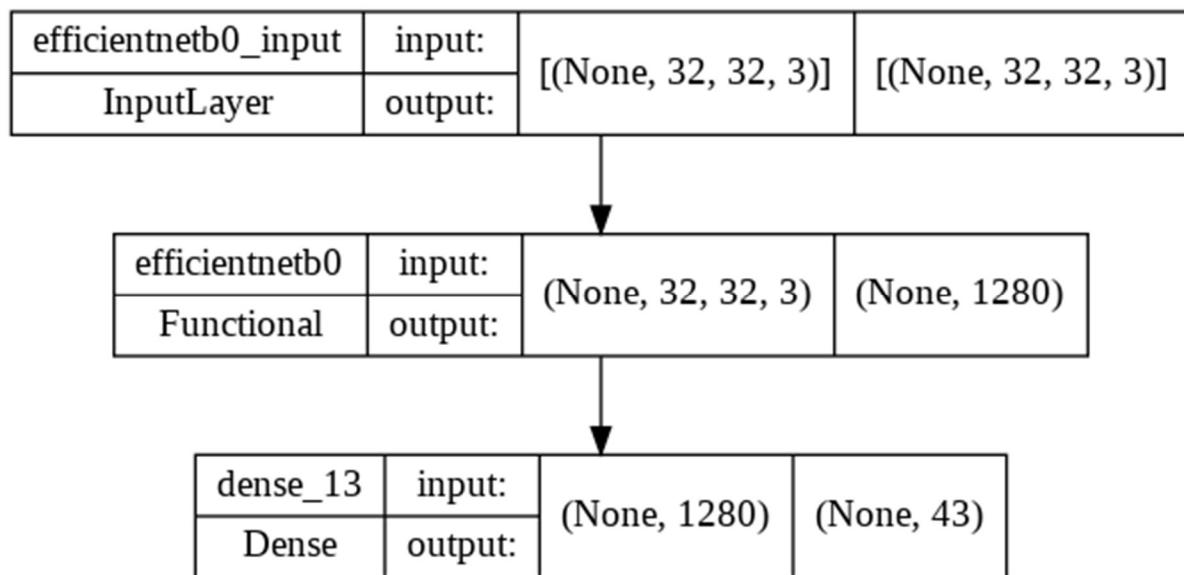


نمودار زیر مقدار تابع loss در طول ایپاک ها را نمایش می دهد که مقدار تابع برای داده های اعتبارسنجی بسیار مناسب به نظر می رسد.



: EfficientNet مدل

ساختار کلی این شبکه از یک شبکه از پیش آموزش داده شده و یک لایه fully-connected تشکیل شده است.



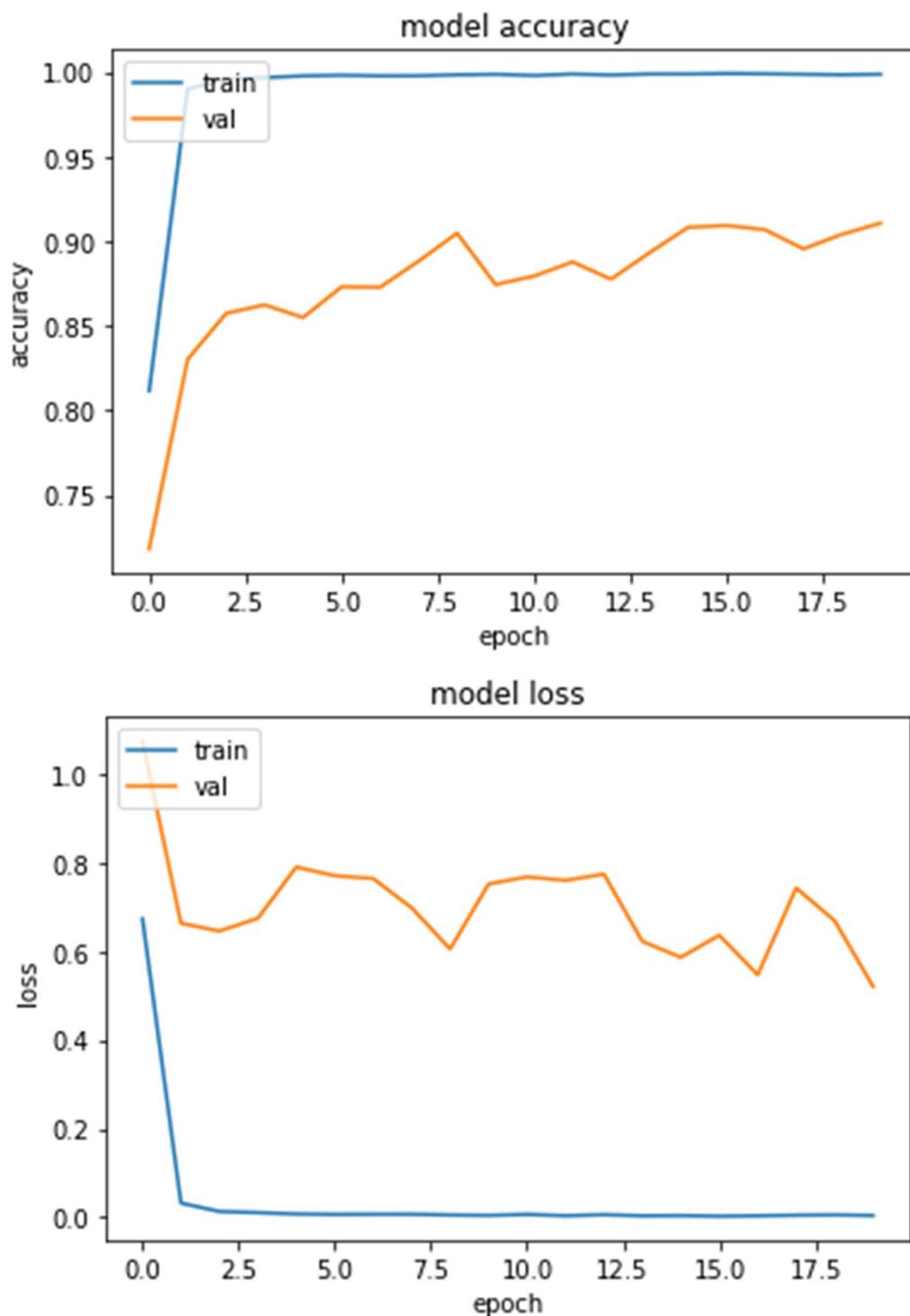
اطلاعات این شبکه مشخص می کند که لایه آخر برای تطبیق خروجی های شبکه آموزش داده شده به ملاس های مدنظر پژوهه ماست. و همچنین تعداد پارامتر های آن ۴۱۰۴۶۵۴ ذکر شده است که به نسبت تعداد دو شبکه قبل تعداد کمتری است.

Layer (type)	Output Shape	Param #
=====		
efficientnetb0 (Functional)	(None, 1280)	4049571
dense_7 (Dense)	(None, 43)	55083
=====		
Total params: 4,104,654		
Trainable params: 4,062,631		
Non-trainable params: 42,023		

جدول ارزیابی:

Dataset	Loss	Accuracy
Train	0.0077	0.99
Validation	0.5206	0.91
Test	0.4565	0.92

با توجه به نمودار زیر که دقت و مقدار تابع loss در دو دیتاست آموزشی و اعتبار سنجی نمایش می‌دهد به نظر می‌رسد که این شبکه تا حدود overfit شده چرا که نتایج روی داده‌های آموزشی بسیار خوب بوده در حالی که اختلاف قابل توجهی با نتایج روی داده‌ای اعتبار سنجی دارد.



مدل : stacked ensemble

همانطور که در کد های پیاده سازی شده مشاهده کردیم داده های تست روی تمامی مدل های تست شده و نتایج دقت آن ها به شکل نمودار زیر است.

Model	Accuracy
CNN	0.964
MobileNet	0.944
DenseNet	0.95
ResNet	0.918
EfficientNet	0.92
Stacked model	1

از بین مدل هایی که به صورت مستقل آموزش دیده شده اند طبق جدول فوق به نظر می رسد که مدل CNN با توجه به اینکه پیچیدگی کمتری داشته عملکرد بهتری ارائه کرده است اما نقطه عطف این پروژه پیاده سازی یک مدل به صورت پشته ای از سایر شبکه هاست که این امر باعث می شود شبکه ها نقض یکدیگر را پوشش دهند و درنهایت مدلی به دقت ۱۰۰ درصد پیدید آید. نکته قابل توجه این است که می توان چنین رویکردی را با شبکه های مختلف و تعداد بیشتر روی مسائل مختلف اعمال کرد و چنین نتایج قابل توجهی بدست آورد.