به نام خدا

ملیکا احمدی رنجبر 97521036

1) سوالات Dropout:

a. یکی از روش‌های جلوگیری از Over Fitting و بهبود Generalization استفاده از لایه Dropout است. این لایه به صورت **Random** برخی از Neuronها را Ignore می‌کند، مقدار آن‌ها را برابر صفر قرار می‌دهد. در حقیقت، مقدار نویز ضرب شده برابر 0 خواهد شد، و معمولا احتمال این کار برای هر Neuron برابر 0.5 است. این کار باعث می‌شود، ابر پارامترها تنظیم شوند، چراکه سایر Neuronها باید تا حد امکان تشخیصی درست برای Input دهند و بر اساس آن Train شوند. مقادیر وزن‌ها نیز باید Scale شود چرا که تعدادی از Neuronها عملا حذف شده، مقادیر وزن بیشتر می‌شود. به طور خلاصه، Dropout برای جلوگیری از Over Fitting با استفاده از حذف و تنظیم Hyper Parameterها است، و به صورت کاملا تصادفی Neuronها را حذف و یا نگهداری می‌کند. مقدار احتمال آن برابر با 0.5 است، به این معنا که به احتمال حذف ار Neuron 50% است. انتخاب این Parameter بر اساس مکان لایه ممکن است باشد، به این معنا که لایه‌های نزدیک‌تر به Input Layer می‌توانند Dropout Rate بیشتری نسبت به لایه‌های Output داشته باشند. دلیل آن است که تصمیم‌گیری نهایی در لایه پایانی با تعداد Neuron بسیار کم دقت را پایین می‌آورد. توجه به این نکته که مقدار بالا برای Dropout Rate ممکن است باعث Under Fitting شود، به دلیل تعدا پایین Neuron باقی‌مانده و همچنین استفاده از این لایه برای شبکه‌های بزرگ بهتر است.

b. از نظر من افزایش Dropout Rate تاثیر مستقیم بر روی ظرفیت شبکه دارد، چرا که با محدود کردن تعداد Neuron برای قسمت آموزش، ظرفیت آن را کاهش می‌دهد و از Over Fitting جلوگیری می‌کند. همانطور که از قبل می‌دانستیم، زیاد بودن تعداد Parameterهای شبکه احتمال Over Fitting را افزایش می‌هد، چون باعث توجه و یادگیری نکات جزئی Input و Extract کردن Featureهای بی‌اهمیت می‌شود. بنابراین حذف تعدادی از Neuronها با Dropout ظرفیت شبکه را کاهش می‌دهد و سایر Neuronهای باقی سعی در برقراری تعدل شبکه خواهند داشت.

2) لایه‌های متفاوتی می‌تواند در ساخت شبکه‌های عمیق به کار رود.

a. لایه Fully Connected: در این مدل لایه، تمامی Neuronها به لایه بعدی خود متصل هستند. این به آن معنا است که وزن‌ها و Parameterهای شبکه با توجه به تمامی Neuronها تنظیم می‌شود. اگر یک Input را در نظر بگیریم، در لایه Fully Connected تمامی پیکسل‌های ورودی (Image Input) به لایه متصل است، و Featureها با توجه به بررسی تمام عکس است، نه یک قسمت خاص. این مدل لایه برای هر نوع ورودی قابل استفاده است و پیش فرض خاصی نیاز نیست در نظر گرفته شود.

b. لایه Convolutional: در این لایه، هر Neuron به تعدادی از Neuronهای اطرافش برای بدست آوردن Featureهای مشخصی متصل است و در واقع همچون Filter عمل می‌کند. این Filter بر روی تمام Neuronهای ورودی (در واقع پیکسل‌های ورودی) با توجه به Kernel Size حرکت می‌کند و اینگونه نیست که چند Filter با وزن‌های متفاوت باشد (بر خلاف Locally Connected که در بخش بعد اشاره می‌شود). این گونه لایه با توجه به تعداد Parameterهای بسیار کمتری که دارد هزینه محاسباتی را نیز کاهش می‌دهد. از طرفی تنها بر روی بری از Inputها قابل استفاده است، مثلا Image. *یادگیری الگوهای محلی*

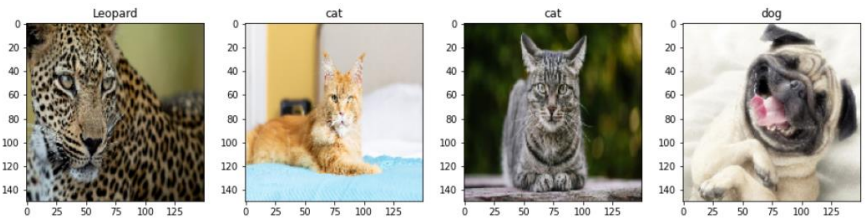
c. لایه Locally Connected: در این مدل لایه،ما برای هر بخش از Input تعدادی Neuron جدا در نظر می‌گیریم که Featureهای مرتبط با همان قسمت را بدست آورند. یعنی هر Neuron به تعدادی از Neuronهای اطرافش متصل است. در واقع Featureهای Input را به صورت Locally بدست می‌آورد و نکته بسیار مهم که تفاوت آن را با Convolutional مشخص می‌کند آن است که هر Neuron خروجی Filter مخصوص خود را دارد در حالی که در Convolutional یک Filter برای تمام Input است و بنابراین تعدا Parameterها در آن بسیار کمتر از Locally Connected است. اما به طور کلی Concept یکسانی دارند. یعنی اگرچه هر Neuron به تعدادی از اطرافیانش متصل است، اما برای هر همسایه یک Filter خاص داریم با وزن‌های متفاوت. *یادگیری الگوهای محلی* در واقع استفاده از Convolutional و Locally Connected برای Feature Extraction است، و Fully Connected برای آموزش. در نهایت نیز پس از لایه‌های Convolutional پس از Flatten کردن خروجی آن، Fully Connected استفاده می‌کنیم. بنابراین Convolutional سربار محاسباتی را کاهش می‌دهد. و از طرفش Locally Connected هم به همین شکل کمک به بهتر Extract شدن Featureها می‌کند و تا حدی Parameterها را کاهش می‌دهد (اگرچه به دلیل Share نبودن وزن‌ها قطعا تعداد بیشتر Parameter نسبت به Convolutional دارد).

3) پیاده‌سازی:

a. ImageDataGenerator را به شکل زیر قرار دادم:

```
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory('train',
        target_size=(150, 150),
        batch_size=batch_size,
        color_mode='rgb',
        class_mode='categorical'
        )
validation_generator = test_datagen.flow_from_directory('test',
        target_size=(150, 150),
        batch_size=batch_size,
        color_mode='rgb',
        class_mode='categorical'
        )
```

Shape ورودی را انتخاب کرده، نوع Class آن، و همچنین Directory که Inputها در آن قرار دارند را مشخص می‌کنیم. علاوع بر آن Rescale برای Map کردن مقادیر بین 0 و 1 به 0 تا 255 استفاده شده است، برای رنگی شدن ورودی‌ها. نتیجه نیز به شکل زیر خواهد بود:



برای ساخت مدل، همانطور که گفته شد، از 4 لایه Conv2D و 2 لایه Dense که خروجی به دلیل Class 5 بودن سوال، 5 Neuron دارد طراحی می‌کنیم:

```
def build_model():
    model = keras.Sequential()
    model.add(layers.InputLayer(input_shape=(150, 150, 3)))
    model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(layers.Conv2D(filters=32, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(layers.Conv2D(filters=128, kernel_size=(3, 3), padding='same', activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(32, activation='relu'))
    model.add(layers.Dense(5, activation='softmax'))
    return model
```

تعداد Filter را به ترتیب افزایش دادیم، و Kernel Size ثابت و برابر 3 است. Activation Function برای لایه های میانی ReLU است و برای لایه پایانی Softmax.

```
model = build_model()
loss = 'categorical_crossentropy'
optimizer = 'Adam'
EPOCHS = 20
BATCH_SIZE = 16
model.compile(loss= loss, optimizer= optimizer, metrics=['accuracy'])
model.fit(train_generator, epochs=EPOCHS, batch_size=BATCH_SIZE, validation_data=validation_generator)
```

سایر Parameterها نیز همانطور که در تصویر بالا است، مشخص شدند. اما با توجه به تمام این حالات، به شدت Over Fitting رخ می‌دهد. به شکل زیر دقت کنید:

```
Epoch 15/20
14/14 [==============================] - 8s 537ms/step - loss: 0.0236 - accuracy: 1.0000 - val_loss: 2.5655 - val_accuracy: 0.4500
Epoch 16/20
14/14 [==============================] - 8s 539ms/step - loss: 0.0028 - accuracy: 1.0000 - val_loss: 3.0931 - val_accuracy: 0.4500
Epoch 17/20
14/14 [==============================] - 8s 539ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 3.3001 - val_accuracy: 0.4333
Epoch 18/20
14/14 [==============================] - 8s 538ms/step - loss: 5.9661e-04 - accuracy: 1.0000 - val_loss: 3.4161 - val_accuracy: 0.4333
Epoch 19/20
14/14 [==============================] - 8s 538ms/step - loss: 3.9997e-04 - accuracy: 1.0000 - val_loss: 3.4752 - val_accuracy: 0.4000
Epoch 20/20
14/14 [==============================] - 8s 536ms/step - loss: 3.1761e-04 - accuracy: 1.0000 - val_loss: 3.4999 - val_accuracy: 0.4000
```

با افزایش دقت بر روی Train Data، دقت Test همچنان پایین است و پدیده Over Fitting را داریم، در مراحل بعد با ثابت نگه داشتن کلیت شبکه، Data Augmentation انجام می‌دهیم که یکی از روش‌های بهبود Generalization است، به دلیل افزایش Train Data. به طور کلی دارای 20 Epoch هستیم.

```
4/4 [==============================] - 1s 138ms/step
              precision    recall  f1-score   support

     Leopard       0.31      0.38      0.34        13
        bird       0.17      0.10      0.12        10
         car       0.57      0.36      0.44        11
         cat       0.38      0.38      0.38        13
         dog       0.28      0.38      0.32        13

    accuracy                           0.33        60
   macro avg       0.34      0.32      0.32        60
weighted avg       0.34      0.33      0.33        60

array([[5, 1, 1, 3, 3],
       [3, 1, 1, 1, 4],
       [2, 0, 4, 1, 4],
       [3, 2, 1, 5, 2],
       [3, 2, 0, 3, 5]])
```

b. حال نوبت Data Augmentation است، برای این کار از همان ImageDataGenerator استفاده می‌کنیم.

```
train_datagen = ImageDataGenerator(rescale=1./255,
        rotation_range=20,
        shear_range=0.2,
        zoom_range=0.2,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        )
```

پس از آموزش همان شبکه با Train Data جدید، نتایج زیر حاصل شد، که بیانگر کاهش زیاد Over Fitting است، تا حدی که Under Fitting رخ داد. برای حل این مشکل افزایش تعداد Epoch راه معقولی است، اما برای مقایسه نتایج هر بخش همان 20 ماند. (با افزایش تعداد Epoch قطعا نتیجه بهتری حاصل خواهد شد، اما صرفا به دلیل مقایسه در شرایط یکسان این تعداد Epoch گذاشته شد)

و در اینجا دقت Test و Train بیشتر متعادل شدند و Generalization هم کمی بهبود یافت.

```
Epoch 15/20
14/14 [==============================] - 8s 536ms/step - loss: 1.4555 - accuracy: 0.3303 - val_loss: 1.4738 - val_accuracy: 0.3667
Epoch 16/20
14/14 [==============================] - 8s 537ms/step - loss: 1.4354 - accuracy: 0.3624 - val_loss: 1.3376 - val_accuracy: 0.4167
Epoch 17/20
14/14 [==============================] - 8s 537ms/step - loss: 1.3505 - accuracy: 0.4312 - val_loss: 1.3052 - val_accuracy: 0.4667
Epoch 18/20
14/14 [==============================] - 8s 534ms/step - loss: 1.3723 - accuracy: 0.4266 - val_loss: 1.3378 - val_accuracy: 0.4000
Epoch 19/20
14/14 [==============================] - 8s 536ms/step - loss: 1.3049 - accuracy: 0.4266 - val_loss: 1.2531 - val_accuracy: 0.4500
Epoch 20/20
14/14 [==============================] - 8s 536ms/step - loss: 1.2324 - accuracy: 0.4495 - val_loss: 1.2981 - val_accuracy: 0.4667
```

```
4/4 [==============================] - 1s 149ms/step
              precision    recall  f1-score   support

     Leopard       0.00      0.00      0.00        13
        bird       0.14      0.10      0.12        10
         car       0.18      0.18      0.18        11
         cat       0.05      0.08      0.06        13
         dog       0.16      0.23      0.19        13

    accuracy                           0.12        60
   macro avg       0.11      0.12      0.11        60
weighted avg       0.10      0.12      0.11        60

array([[0, 2, 1, 5, 5],
       [0, 1, 4, 3, 2],
       [0, 1, 2, 6, 2],
       [0, 2, 3, 1, 7],
       [2, 1, 1, 6, 3]])
```

c. حال یکی دیگر از روش‌های کاهش Over Fitting افزودن لایه‌های Dropout است، چرا که تعداد Parameterها را با حذف برخی Neuronها کم می‌کند. در این قسمت لایه Dropout با Rateهای متفاوت اضافه کردیم، توجه شود در لایه‌های ابتدای اگر Rate بزرگ باشد، و در انتها اگر کوچک باشد، بهتر است. اما این نکته هم مهم است که اگر تعداد و مقدار Dropout زیاد باشد، احتمال Under Fitting دارد. تاثیر تغییرات اعمال شده زیاد به چشم نمی‌آید، شاید به دلیل کم بودن تعداد Epoch. ما به طور میانگین، دقت بر روی آموزش و تست افزایش می‌یابد. شاید دقت نهایی بسیار کم افزایش یافته باشد، اما می‌توان روند کلی بهتر شدن Generalization را دید. البته که در هر بار Run شدن نیز، جواب متفاوت است و احتمال آنکه بسیار بهتر از پاسخ کنونی شود وجود دارد.

```
Epoch 15/20
14/14 [==============================] - 8s 569ms/step - loss: 1.0557 - accuracy: 0.5688 - val_loss: 1.9629 - val_accuracy: 0.4667
Epoch 16/20
14/14 [==============================] - 8s 575ms/step - loss: 0.9934 - accuracy: 0.5780 - val_loss: 1.2531 - val_accuracy: 0.4667
Epoch 17/20
14/14 [==============================] - 8s 574ms/step - loss: 1.0580 - accuracy: 0.5505 - val_loss: 1.1217 - val_accuracy: 0.5333
Epoch 18/20
14/14 [==============================] - 8s 571ms/step - loss: 1.0332 - accuracy: 0.5321 - val_loss: 1.2078 - val_accuracy: 0.4833
Epoch 19/20
14/14 [==============================] - 8s 572ms/step - loss: 1.0947 - accuracy: 0.5596 - val_loss: 1.4198 - val_accuracy: 0.5667
Epoch 20/20
14/14 [==============================] - 8s 572ms/step - loss: 0.9574 - accuracy: 0.5872 - val_loss: 1.1758 - val_accuracy: 0.4833

4/4 [==============================] - 1s 155ms/step
              precision    recall  f1-score   support

     Leopard       0.19      0.23      0.21        13
        bird       0.00      0.00      0.00        10
         car       0.17      0.18      0.17        11
         cat       0.20      0.08      0.11        13
         dog       0.20      0.31      0.24        13

    accuracy                           0.17        60
   macro avg       0.15      0.16      0.15        60
weighted avg       0.16      0.17      0.15        60

array([[3, 2, 3, 2, 3],
       [7, 0, 1, 0, 2],
       [2, 3, 2, 0, 4],
       [2, 0, 3, 1, 7],
       [2, 2, 3, 2, 4]])
```

d. امتیاز f1 میانگین هارمونیک دقت و یادآوری را به شما می دهد. نمرات مربوط به هر کلاس، دقت طبقه بندی کننده را در طبقه‌بندی نقاط داده در آن کلاس خاص در مقایسه با سایر کلاس‌ها به شما می‌گوید. Support تعداد نمونه‌هایی از پاسخ واقعی است که در آن کلاس وجود دارد.

e. Confusion Matrix یک معیار بسیار مناسب برای دیدن عملکرد مدل است، چرا که پیشبینی مدل و همچنین دسته درست را در کنار یکدیگر قرار می‌دهد و همچنین پیشبینی غلط را نیز می‌گوید مربوط به کدام دسته بوده‌است. دو معیار مطرح شده تا حد زیادی مشابه یکدیگر هستند.

4) منابع:

a. https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

b. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html#sklearn.metrics.confusion_matrix

c. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html