

## Homework 5 - Transfer Learning with Advanced CNN Models - Spring 2025

**Essay Question:** *Even though the basic convolution neural network (CNN) architecture appeared during the second wave of neural network research [Fukushima, IEEE SMC-Trans, 1983], there were several unique features in Krizhevsky's CNN model [Krizhevsky, NeurIPS, 2012] that were critical for the renewed interest in deep learning models over the past decade.* Write a one paragraph essay that starts with the preceding prompt. The following sentences should support this assertion by identifying the unique features of Krizhevsky's work and discussing why these features were instrumental in driving the recent resurgence of neural network research. Your paragraph should close with your opinion regarding the relative importance of these features and how they may drive or hinder future neural network research.

**Problem 1:** Consider a CNN model that takes an JPG image of shape (128,128,3) and passes it through 3 Conv2D layers with ReLU activations, a string of 2, and no zero padding. The  $\ell$ th Conv2D layer ( $\ell = 1, 2, 3$ ) outputs  $\ell \times 16$  channels. The output of the last Conv2D layer is flattened and mapped to 5 real-valued outputs through a Dense layer using a softmax activation function.

- Write a Keras script that instantiates the *Model* object for this CNN.
- Determine the shape of each layer's output.
- Determine how many trainable parameters are in the model.

**Solution:** This network would be used for a multi-class classification problem

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(128,128,3))
x = layers.Rescaling(1./255)(inputs)
num_channels = 16
for ell in range(3):
    x = layers.Conv2D(num_channels,3,activation="relu",strides=2)(x)
    num_channels = num_channels*2
x = layers.Flatten()(x)
outputs = layers.Dense(5,activation="softmax")(x)
model = keras.Model(inputs,outputs)
```

| layer     | output shape  | trainable parameters                           | so the number of trainable parameters is |
|-----------|---------------|--|--|
| input     | (128, 128, 3) | 0  |  |
| Rescaling | (128, 128, 3) | 0  |  |
| Conv2D1   | (63, 63, 16)  | $(3 \times 3 \times 3 + 1) \times 16 = 448$    |  |
| Conv2D2   | (31, 31, 32)  | $(3 \times 3 \times 16 + 1) \times 31 = 4640$  |  |
| Conv2D3   | (15, 15, 64)  | $(3 \times 3 \times 32 + 1) \times 64 = 18496$ |  |
| Flatten   | (14400,)      | 0  |  |
| Dense     | (5,)          | $((14400 + 1) \times 5) = 72005$               |  |

$$448 + 4640 + 18496 + 72005 = 95,589$$

**Problem 2:** Consider the 1D convolutional network declared by the following Keras script,

```
inputs = Keras.Input(shape=(6,1))
outputs = layers.Conv1D(1,3, activation = "relu", use_bias=False)(inputs)
model = keras.Model(inputs = inputs, outputs = outputs)
```

If all trainable parameters are equal to one, then determine the model's output when the input is a rank-1 tensor  $\begin{bmatrix} 0 & -1 & 2 & -1 & 1 & 0 \end{bmatrix}$ .

**Solution:**

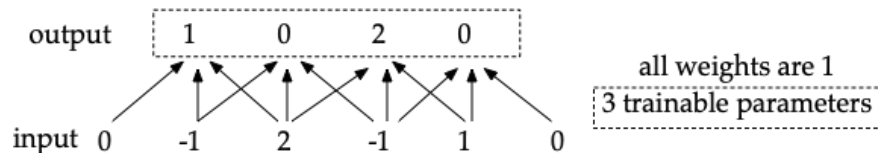


Figure 1: solution

**Notebook Assignment 1 (brain tumor classification):** Gathering and labeling data is expensive and time consuming. It is therefore important to know how to train models when there is limited data available. One way of doing this is to *transfer* the knowledge contained in one model to another model. This notebook assignment asks to use a pretrained CNN model and use it for the classification analysis of brain MRI scan images. The classification task is to classify each MRI image as either having a "tumor" or "no-tumor".

The Brain MRI image dataset (zip file on course website) consists of 253 image samples of high-resolution brain MRI scans. The images are grayscale in nature and vary in size. The dataset has been pre-processed so the images are placed in two directories TRAIN and TEST. Each of these directories has two subdirectories NO or YES representing the true label of each image. The images are jpg files with shape (244,244,3).

1. Create a `train_ds` and `test_ds` dataset objects using the keras utility "image data set from directory". Your dataset should declare the image size as (244,244) and create batches of size 32. Use a 30% validation split to decompose the training dataset object into a p-training and validation dataset object. Print out the number of batches in the p-training, the validation, and the testing dataset objects.

**Solution:** I used the following script

```
height = 244
width = 244
batch_size = 10
dataset_dir = "data/TRAIN/"
test_dir = "data/TEST/"

train_ds = image_dataset_from_directory(
    dataset_dir, image_size = (244,244),
    batch_size = 32
)

val_split = 0.30
train_ds_size = len(list(train_ds))
val_size = int(val_split*train_ds_size)
ptrain_size = train_ds_size-val_size
ptrain_ds = train_ds.take(ptrain_size)
val_ds = train_ds.skip(ptrain_size).take(val_size)

test_ds = image_dataset_from_directory(
    test_dir, image_size = (244,244),
    batch_size=32
)

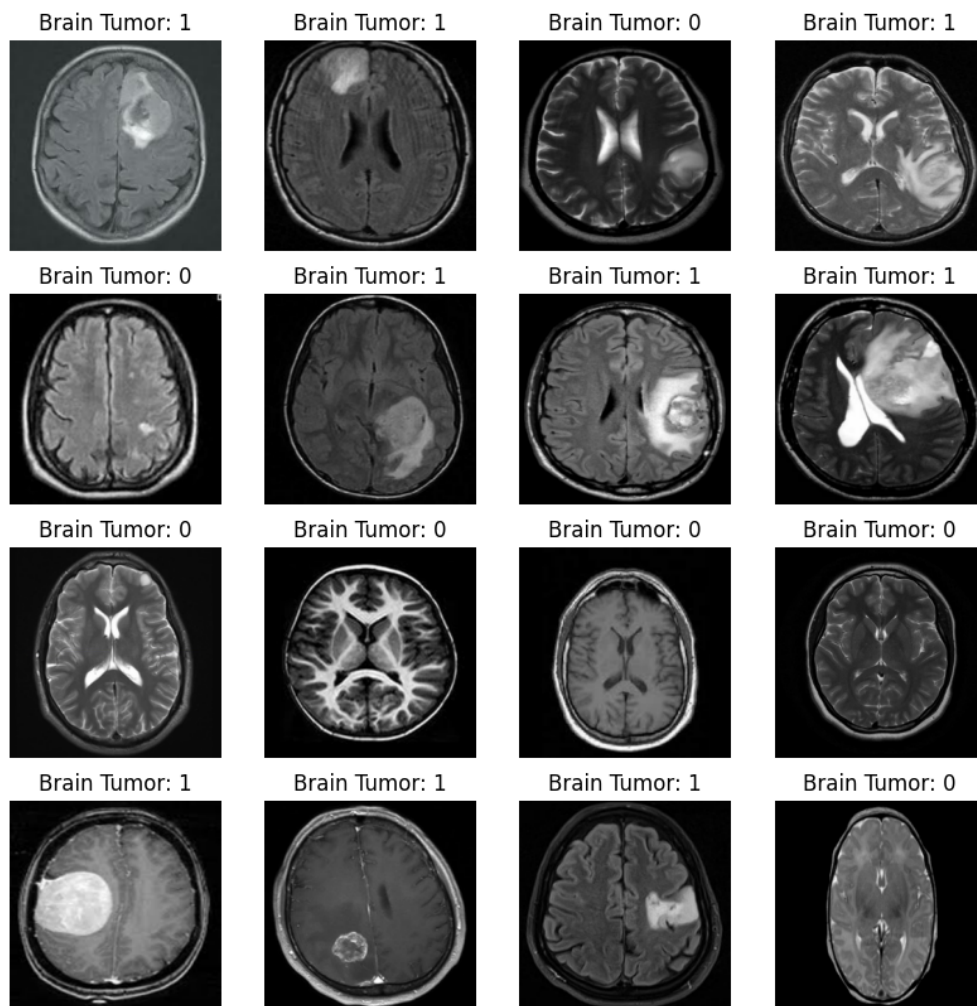
print(f"ptraining dataset size = {len(ptrain_ds)}")
print(f"validation dataset size = {len(val_ds)}")
print(f"testing dataset size = {len(test_ds)}")
```

2. Display 16 of the training images in a 4 by 4 array and have the title on each image indicates its true class label

**Solution:** I used the following script

```
fig, ax = plt.subplots(4,4, figsize=(10,10))
images, labels = next(iter(train_ds))
for k in range(16):
    #print(images[0].shape)
    image = images[k].numpy().astype('uint8')
    label = labels[k].numpy().astype('uint8')
    i, j = k//4, k%4
    ax[i, j].imshow(image)
    ax[i, j].set_title(f"Brain Tumor: {label}")
    ax[i, j].axis('off')
plt.show()
```

The images are shown below.



3. From `keras.applications` import the pre-trained model `ResNet50V2`. Set the input shape to `(244,244,3)` and leave off the top layer of the model. Fix the model weights and then add a classification layer on top that consists of data augmentation that randomly flips, rotates, and zooms the input images, a `GlobalAveragePooling2D`, 64 node `Dense` layer, a `Dropout` layer with 0.3 dropout rate and a dense layer that has a single node as its output

with a sigmoid activation function. Use the model summary routines to determine the total number of parameters in the model and the number of trainable weights. Be sure to rescale the inputs to the convolutional base so they are in the range from 0 to 1. Compile the model with a Adam optimizer using a binary crossentropy loss function. Print out the number of trainable parameters in your refined model. How many parameters are in the fixed convolutional base?

**Solution:** I used the following script

```
n_hidden = 64

data_augmentation = keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.2),
])

conv_base = ResNet50V2(input_shape=(244,244,3), include_top=False)
conv_base.trainable = False
#conv_base.summary()

inputs = keras.Input(shape=(244,244,3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(inputs)
x = conv_base(x)
x = GlobalAveragePooling2D()(x)
x = layers.Dense(n_hidden,activation = "relu")(x)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model_resnet50v2 = keras.Model(inputs,outputs)

model_resnet50v2.compile(optimizer=Adam(),
                        loss="binary_crossentropy",
                        metrics=["accuracy"])

model_resnet50v2.summary()

# Total params: 23,696,001 (90.39 MB)
# Trainable params: 131,201 (512.50 KB)
# Non-trainable params: 23,564,800 (89.89 MB)

conv_base.summary()
#Total params: 23,564,800 (89.89 MB)
#Trainable params: 0 (0.00 B)
#Non-trainable params: 23,564,800 (89.89 MB)
```

4. Checkpoint the model to save the model with the smallest validation loss. The train the model using p-training dataset for 20 epochs and that validates the models using the validation dataset. Plot the training curves for your model.

**Solution:** I used the following script

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="best_model.keras",
        save_best_only = True,
        monitor="val_loss"
    )
]

num_epochs = 20
history = model_resnet50v2.fit(ptrain_ds, epochs=num_epochs,
                             validation_data = val_ds,
                             callbacks = callbacks)

def training_curves(history):
    train_loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    train_accuracy = history.history["accuracy"]
    val_accuracy = history.history["val_accuracy"]

    epochs = np.arange(1,len(train_loss) + 1)
```

```

figure, axis = plt.subplots(1,2,figsize=(10,4))
axis[0].plot(epochs, train_loss, "b--", label="p-training loss")
axis[0].plot(epochs, val_loss, "b", label="validation loss")
axis[0].set_title("p-Training and Validation Losses")
axis[0].set_xlabel("Epochs")
axis[0].set_ylabel("Loss")
axis[0].legend()

axis[1].plot(epochs, train_accuracy, "b--", label="p-training accuracy")
axis[1].plot(epochs, val_accuracy, "b", label="validation accuracy")
axis[1].set_title("p-Training and Validation accuracy")
axis[1].set_xlabel("Epochs")
axis[1].set_ylabel("Accuracy")
axis[1].legend()

training_curves(history)

```

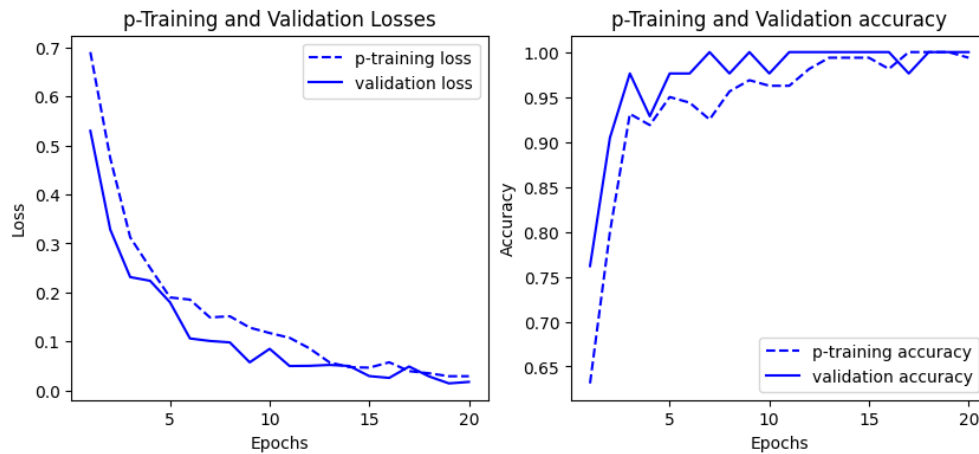


Figure 2: Training Curves

5. Evaluate the accuracy of your final trained model on the testing dataset. Let TP denote the number of true positive classifications, FP denotes the number of false positive classifications, TN being the number of true negative classifications, and FN being the number of false negative classifications. Accuracy is the  $(TP+TN)$  divided by total number of examples when the model output is greater than 0.5. It is a coarse measure of a classifier's performance. In practice, we look at the

$$\begin{aligned} \text{TPR} &= \text{True Positive Rate} = \frac{\text{TP}}{\text{TP} + \text{FN}} \\ \text{FPR} &= \text{False Positive Rate} = \frac{\text{FP}}{\text{TN} + \text{FP}} \end{aligned}$$

for a variety of threshold, not just 0.5. Compute the TPR and FPR for your classifier for the following thresholds

$$\text{thresholds} = \{0.025, 0.05, 0.075, .1, .2, .3, .4, .5, .6, .7, .8, .9, .925, .950, .975\}$$

and plot the TPR vs FPR to form the classifiers Receiver Operating Characteristic. Also plot the heatmap of the classifier's confusion matrix when the threshold is 0.5.

**Solution:** I used the following script. This gave an 86% accuracy on the testing data.

```

best_model = keras.models.load_model("best_model.keras")
test_loss, test_accuracy = best_model.evaluate(test_ds)
print(f"test loss = {test_loss:.2f}")
print(f"test accuracy = {test_accuracy:.2f}%")

threshold_table = [.025, .05, .075, .1, .2, .3, .4, .5, .6, .7, .8, .9, .925, .95, .975]
roc = [(0,0)]

for threshold in threshold_table:

    y_true=[]
    y_pred=[]

    for x,y in test_ds:
        y_true = np.hstack((y_true,y.numpy()))
        predictions = best_model.predict(x)
        predictions = predictions.reshape((len(predictions),))
        predictions = (predictions>=threshold).astype('int32')
        y_pred = np.hstack((y_pred,predictions))
    y_pred_not = (1-y_pred).astype('int32')
    tp = np.sum((y_pred*(y_true==1)).astype('int32'))
    fn = np.sum((y_pred*(y_true==0)).astype('int32'))
    fp = np.sum((y_pred_not*(y_true==1)).astype('int32'))
    tn = np.sum((y_pred_not*(y_true==0)).astype('int32'))

    tpr = tp/(tp+fn)      #true positive rate
    tnr = tn/(tn+fp)      #true negative rate
    fpr = 1-tnr           #false positive rate
    fnr = 1-tpr           #false negative rate

    roc.append((tpr,fpr))

    if (threshold == .5):
        cm = confusion_matrix(y_true,y_pred)
        sns.heatmap(cm, annot=True, fmt='d')
        plt.xlabel('Predicted')
        plt.ylabel('True')

rocx=np.array(roc)
tprx = rocx[:,0]
fprx = rocx[:,1]
plt.figure(2)
plt.plot(fprx,tprx)

```

