Brief Model Definition

In [ ]:
```
model_brief=models.Sequential()
model_brief.add(layers.Conv2D(32, (3,3) , padding='same',activation='relu', inpu
model_brief.add(layers.BatchNormalization())
model_brief.add(layers.MaxPooling2D(pool_size=(2,2)))

model_brief.add(layers.Flatten())
model_brief.add(layers.Dense(128, activation='relu'))
model_brief.add(layers.Dense(5, activation='softmax'))
```

Training the Brief Model

In [ ]:
```
model_brief.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tru
              metrics=['accuracy'])
```

In [ ]:
```
%%time
with tf.device('/device:GPU:0'):
  history = model_brief.fit(X_train, y_train, epochs=100,
                    validation_data=(X_val, y_val), batch_size=128)
```

Deeper Model Definition

In [ ]:
```
#output softmax layer should have 5 outputs
# Building a ConvNet
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_sha
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.Dropout(0.25))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.25))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(5, activation='softmax'))
```

Training the Deeper Model

In [ ]:
```
%%time
with tf.device('/device:GPU:0'):
  model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tru
              metrics=['accuracy'])
  history_deeper = model.fit(X_train, y_train, epochs=80,
                    validation_data=(X_val, y_val), batch_size=128)
```

Loading ResNet

In [ ]:
```
#Resnet
base_model = tf.keras.applications.ResNet152(weights = 'imagenet', include_top =
for layer in base_model.layers:
    layer.trainable = False
```

Define Output Layer

In [ ]:
```
x = layers.Flatten()(base_model.output)
x = layers.Dense(1000, activation='relu')(x)
predictions = layers.Dense(5, activation = 'softmax')(x)
```

Training the Model

In [ ]:
```
resnet_model = Model(inputs = base_model.input, outputs = predictions)
resnet_model.compile(optimizer='adam', loss=losses.sparse_categorical_crossentro
```

In [ ]:
```
%%time
with tf.device('/device:GPU:0'):
  history_resnet = resnet_model.fit(X_train_tl, y_train, batch_size=128, epochs=
```

Loading VGGNet

In [ ]:
```
base_model = tf.keras.applications.VGG16(weights = 'imagenet', include_top = Fal
for layer in base_model.layers:
  layer.trainable = False
base_model.summary()
```

Define Output Layer

In [ ]:
```
x = layers.Flatten()(base_model.output)
x = layers.Dense(4096, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(4096, activation='relu')(x)
x = layers.Dropout(0.5)(x)
predictions = layers.Dense(5, activation = 'softmax')(x)
head_model = Model(inputs = base_model.input, outputs = predictions)
head_model.compile(optimizer='adam', loss=losses.sparse_categorical_crossentropy
```

Training the Model

In [ ]:
```
%%time
with tf.device('/device:GPU:0'):
  history = head_model.fit(X_train_tl, y_train, batch_size=128, epochs=100, vali
```